МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»

Н.О. Комлева

ОСНОВИ ПРОГРАМУВАННЯ

НАВЧАЛЬНИЙ ПОСІБНИК

Одеса

2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»

Н.О. Комлева

ОСНОВИ ПРОГРАМУВАННЯ

НАВЧАЛЬНИЙ ПОСІБНИК

Рекомендовано Вченою радою
Національного університету «Одеська політехніка»
як навчальний посібник
для здобувачів вищої освіти
за спеціальністю 121 – Інженерія програмного забезпечення

(протокол № 13 від 29.05.2024)

Одеса
2024

Рецензент:
Завідувач кафедри програмного
забезпечення автоматизованих систем
Національного університету кораблебудування
імені адмірала Макарова, д.т.н. проф. Приходько С.Б.

Комлева Н. О. Основи програмування. Навч. посібник для закладів вищої освіти. – Одеса: 2024. – 283 с.

Посібник присвячений викладенню основ програмування з використанням поширеної мови програмування С. Він містить вступ, дев’ятнадцять розділів та три додатки. Наприкінці кожного розділу наведено контрольні питання та тести для самоконтролю.

Перші два розділи присвячено розкриттю основних понять з мов програмування та принципів створення структурних програм. *Розділи з третього по шостий* розглядають засоби побудови лінійних програм та програм з розгалуженнями і циклами. У *сьомому* розділі наведено основні поняття про статичні одновимірні та багатовимірні масиви. *Восьмий та дев’ятий розділи* висвітлюють основні аспекти роботи з покажчиками та динамічним розподілом пам’яті. Роботою з рядками як з елементами масиву та з використанням спеціальних функцій займається *десятий розділ*. В *одинадцятому розділі* наведено опис структур та об’єднань: їх галузь застосування, розміщення в пам’яті, звертання до полів. *Три наступні розділи* присвячені роботі з функціями, у тому числі з використанням посилань та засобів локалізації консольних програм. *П’ятнадцятий та шістнадцятий розділи* висвітлюють принципи роботи з файлами та можливості, які надають команди препроцесора. *Три останніх розділи* присвячені використанню бітових операцій, динамічних структур даних та створенню і використанню статичних та динамічних бібліотек.

Навчальний посібник є методичним матеріалом, що забезпечує дисципліну «Основи програмування». Він може бути корисним для студентів, що навчаються за напрямками «Інженерія програмного забезпечення», «Комп'ютерні науки», а також для широкого кола фахівців з суміжних спеціальностей.

ЗМІСТ

ВСТУП

Найбільш універсальним інструментом, який має людина, є комп'ютер. Комп'ютери здатні виконувати обчислення з величезною швидкістю і точністю, вони дозволяють зберігати практично необмежену кількість інформації. Завдяки обчислювальній техніці спрощується та автоматизується багато рутинних завдань. Це дозволяє звільнити від цих процесів людину і дати їй можливість займатися більш творчими процесами. Однак щоб автоматизувати навіть саме просте завдання, комп'ютеру необхідно вказати, що саме він повинен робити. Крім того, сказати це йому треба на доступній для нього мові [1].

Як засіб спілкування між людиною і комп'ютером, придумано величезну кількість мов програмування. За допомогою мов програмування людина створює програми, з якими комп'ютер вже безпосередньо працює. Програми являють собою набори інструкцій, які комп'ютер може розуміти і виконувати. Процес написання програми тісно пов'язаний з поняттям алгоритму [2].

Під алгоритмом розуміється точне розпорядження, яке визначає послідовність дій, що забезпечує отримання необхідного результату з початкових даних. Алгоритм може бути призначений для виконання його людиною або автоматичним пристроєм. Створення алгоритму, нехай навіть самого простого, – процес творчий. Інша справа – реалізація вже існуючого алгоритму. Її можна доручити суб'єкту чи об'єкту, який не зобов'язаний вникати в суть справи, а можливо, і не здатний його зрозуміти. Такий суб'єкт або об'єкт прийнято називати формальним виконавцем. Прикладом формального виконавця може служити мікрохвильова піч, яка працює за закладеною в неї програмою, навіть якщо в неї забули покласти їжу. Людина теж може виступати в ролі формального виконавця, але в першу чергу формальними виконавцями є різні автоматичні пристрої, і комп'ютер в тому числі. Кожен алгоритм створюється у розрахунку на цілком конкретного виконавця. Ті дії, які може здійснювати виконавець, називаються його допустимими діями. Сукупність допустимих дій утворює систему команд виконавця. Алгоритм повинен містити тільки ті дії, які допустимі для даного виконавця. Об'єкти, над якими виконавець може вчиняти дії, утворюють так звану середу виконання. Наприклад, для алгоритмів, що зустрічаються в математиці, середовищем того чи іншого виконавця можуть бути числа різної природи – натуральні, дійсні і т.п., літери, літерні вирази, рівняння, тотожності і т.п. [3]

На будь-якій стадії існування алгоритми і програми представляють за допомогою конкретних зображувальних засобів, склад і правила вживання яких утворюють конкретні засоби або форми запису. Донині склалися п'ять найбільш уживаних способів запису: словесний, формульно-словесний, графічний, за допомогою псевдокоду і мов програмування [4].

Словесне завдання описує алгоритм – інструкцію про виконання дій в певній послідовності за допомогою слів і речень природної мови. Форма викладу довільна і встановлюється розробником. У формульно-словесному способі записи інструкція про дії містить формальні символи і вирази (формули) у поєднанні зі словесними поясненнями. Графічна запис або схема – це зображення алгоритму за допомогою геометричних фігур, званих блоками. Послідовність блоків і сполучних ліній утворюють схему.

Поряд зі схемами для зображення алгоритмів широко використовується псевдокод. Псевдокодом називається система правил запису алгоритму з використанням набору певних конструкцій для опису керуючих дій. Псевдокод дозволяє формально зображати логіку алгоритму, використовуючи стандартизовані конструкції природної мови для зображення управління і зберігаючи можливості мови для опису дій з обробки інформації. Даний спосіб тісно пов'язаний зі структурним підходом до програмування. Псевдокод займає проміжне положення між природною мовою та мовою програмування. Його застосовують переважно для того, щоб докладніше пояснити роботу програми, що полегшує перевірку правильності програми. Крім того, псевдокод дає програмісту велику свободу в зображенні алгоритму. Потрібно тільки вживати стандартні керуючі конструкції і правила запису [5].

Останнім способом запису алгоритмів є мова програмування. Розглянуті вище способи зручні для програміста, але не прийнятні для ЕОМ, оскільки вони не можуть бути однозначно зрозумілі.

На сьогодні існує дуже багато вже готових реалізацій. І, вирішуючи ту чи іншу інформаційну задачу, необхідно вибрати адекватний програмний засіб. Це можуть бути електронні таблиці, системи управління базами даних, математичні пакети і т.п. І тільки в тому випадку, коли подібні засоби не дають можливості вирішити завдання, слід вдаватися до універсальних мов програмування та створювати власні алгоритми і програми.

Програмістів прийнято розділяти за двома категоріями: прикладні і системні. Системні програмісти – це розробники базових програмних засобів

ЕОМ (операційних систем, трансляторів, сервісних засобів тощо). Прикладні програмісти розробляють засоби прикладного програмного забезпечення ЕОМ, призначені для вирішення завдань з різних областей (наука, техніка, виробництво, сфера обслуговування, навчання тощо). Вимоги до якості як прикладних програм, так і системних сьогодні дуже високі. Програма повинна не тільки правильно вирішувати задачу, а й мати сучасний інтерфейс, бути високонадійною, дружньою по відношенню до користувача і т.д. Тільки такі програми можуть витримувати конкуренцію на світовому ринку програмних продуктів.

З розвитком комп'ютерної техніки розвивалися методика, і технологія програмування. Спочатку виникає командне і операторне програмування, в 1960-х рр. бурхливо розвивається структурне програмування, з'являються лінії логічного та функціонального програмування, а останнім часом – об'єктно-орієнтоване і візуальне програмування. Завдання, яке слід ставити при первісному вивченні програмування, – освоєння основ структурної методики програмування.

В наш час у світі існує кілька сотень реально використовуваних мов програмування. Для кожного є своя область застосування [6].

Будь-який алгоритм є послідовністю розпоряджень, виконавши які можна за кінцеве число кроків перейти від початкових даних до результату. Залежно від ступеня деталізації розпоряджень зазвичай визначається рівень мови програмування – чим менше деталізація, тим вище рівень мови.

За цим критерієм можна виділити такі рівні мов програмування:

- машинні;
- машинно-орієнтовані (асемблери);
- машинно-незалежні (мови високого рівня).

Машинні та машинно-орієнтовані мови – це мови низького рівня, що вимагають вказівок дрібних деталей процесу обробки даних. Мови ж високого рівня імітують природні мови, використовуючи деякі слова розмовної мови і загальноприйняті математичні символи. Ці мови більш зручні для людини.

При програмуванні машинною мовою програміст може тримати під своїм контролем кожну команду і кожну комірку пам'яті, використовувати всі можливості наявних машинних операцій. Але процес написання програми на машинній мові дуже трудомісткий і виснажливий. Програма виходить громіздкою, її важко налагоджувати, змінювати і розвивати. Тому у випадку, коли потрібно мати ефективну програму, яка в максимальному ступені враховує специфіку

конкретного комп'ютера, замість машинних мов використовують близькі до них машинно-орієнтовані мови – асемблери.

Мова асемблера – це машинно-залежна мова низького рівня, в якої короткі мнемонічні імена відповідають окремим машинним командам. Вона використовується для представлення в більш наочній формі програм, записаних в машинному коді.

За допомогою мов низького рівня створюються дуже ефективні і компактні програми, оскільки розробник отримує доступ до всіх можливостей процесора. З іншого боку, при цьому потрібно дуже добре розуміти будову комп'ютера, ускладнюється налагодження великих програм, а остаточна програма не може бути перенесена на комп'ютер з іншим типом процесора. Подібні мови зазвичай застосовуються для написання невеликих системних програм, драйверів пристроїв, модулів стикування з нестандартним обладнанням, коли найважливішими вимогами стають компактність, швидкодія і можливість прямого доступу до апаратних ресурсів. У деяких областях, наприклад в машинній графіці, на мові асемблера пишуться бібліотеки, які ефективно реалізують алгоритми обробки зображень, що вимагають інтенсивних обчислень.

Мови високого рівня були розроблені для того, щоб звільнити програміста від урахування технічних особливостей конкретних комп'ютерів, їх архітектури. Рівень мови характеризується ступенем її близькості до природної, людської мови. Машинна мова не схожа на людську, вона украй бідна у своїх образотворчих засобах. Засоби запису програм на мовах високого рівня більш виразні і звичні для людини. Наприклад, алгоритм обчислення за складною формулою не розбивається на окремі операції, а записується компактно у вигляді одного виразу з використанням звичної математичної символіки. Скласти свою або зрозуміти чужу програму на такій мові набагато простіше.

Важливою перевагою мов високого рівня є їх універсальність, незалежність від ЕОМ. Програма, написана такою мовою, може виконуватися на різних машинах. Укладачеві програми не потрібно знати систему команд ЕОМ, на якій він має намір проводити обчислення. При переході на іншу ЕОМ програма не вимагає переробки. Такі мови – не тільки засіб спілкування людини з машиною, а й людей між собою. Програма, написана мовою високого рівня, легко може бути зрозуміла будь-яким фахівцем, який знає мову і характер завдання.

Таким чином, можна сформулювати основні переваги мов високого рівня перед машинними:

- алфавіт мови високого рівня значно ширше алфавіту машинної мови, що істотно підвищує наочність тексту програми;
- набір операцій, допустимих для використання, не залежить від набору машинних операцій, а вибирається з міркувань зручності формулювання алгоритмів розв'язання задач певного класу;
- формат речень досить гнучкий і зручний для використання, що дозволяє за допомогою одного речення задати досить змістовний етап обробки даних;
- необхідні операції задаються за допомогою загальноприйнятих математичних позначень;
- даним в мовах високого рівня присвоюються індивідуальні імена, які обираються програмістом;
- в мові може бути передбачений значно ширший набір типів даних у порівнянні з набором машинних типів даних.

Таким чином, мови високого рівня в значній мірі є машинно-незалежними. Вони полегшують роботу програміста і підвищують надійність створюваних програм [7].

Мови високого рівня поділяються на:

- процедурні;
- логічні;
- об'єктно-орієнтовані.

Процедурні мови призначені для однозначного опису алгоритмів. При вирішенні завдання процедурні мови вимагають в тій чи іншій формі явно записати процедуру її вирішення.

Першим кроком у розвитку процедурних мов програмування була поява проблемно-орієнтованих мов. У цій назві знайшов відображення той факт, що при їх розробці йдуть не від «машини» , а «від завдання»: у мові прагнуть максимально повно врахувати специфіку класу задач, для вирішення яких її передбачається використовувати. Наприклад, для багатьох науково-технічних завдань характерні великі розрахунки за складними формулами, тому в орієнтованих на такі завдання мовах вводять зручні засоби їх запису. Використання понять, термінів, символів, звичних для фахівців відповідної галузі знань, полегшує їм вивчення мови, спрощує процес складання і налагодження програми.

Різноманітність класів задач призвело до того, що на сьогоднішній день розроблено декілька сотень алгоритмічних мов. Правда, широке поширення і міжнародне визнання отримали лише 10-15 мов. Серед них слід відзначити Fortran, Cobol, Basic.

У той же час в середині 60-х років почали розробляти алгоритмічні мови широкої орієнтації – універсальні мови. Зазвичай вони будувалися за принципом об'єднання можливостей вузько-орієнтованих мов. Серед них найбільш відомі PL/1, Pascal, C, Modula, Ada.

Логічні мови (Prolog, Lisp, Mercury, KLO тощо) орієнтовані не на запис алгоритму розв'язання задачі, а на систематичний і формалізований опис задачі з тим, щоб рішення випливало з складеного опису. У цих мовах вказується – що дано і що потрібно отримати. При цьому пошук рішення задачі покладається безпосередньо на ЕОМ.

Розглянемо тепер об'єктно-орієнтовані мови (ObjectPascal, C++, Java, ObjectiveCaml і др.). Керівна ідея об'єктно-орієнтованих мов полягає в прагненні зв'язати дані з обробляючими ці дані процедурами в єдине ціле – об'єкт. Програма на об'єктно-орієнтованій мови, вирішуючи деяку задачу, по суті, описує частину світу, що відноситься до цього завдання. Опис дійсності у формі системи взаємодіючих об'єктів природніший, ніж у формі взаємодіючих процедур.

Однією з найбільш популярних мов програмування високого рівня є мова С [8]. Вона створювалася як мова для розробки операційної системи UNIX. С часто називають «асемблером, що переноситься», маючи на увазі те, що він дозволяє працювати з даними практично так само ефективно, як на асемблері, надаючи при цьому структуровані конструкції, що управляють, і абстракції високого рівня (структури і масиви). Саме з цим пов'язана його величезна популярність і понині. Дана мова пропагує ідеологію добре структурованих (розбитих на блоки) програм. Вона дозволяє писати програми під Windows. Адже перші операційні системи були написані саме на мові Сі. Програмування під Windows зараз дуже поширене в світі, але для цього необхідно багато праці і зусилля [9].

Незважаючи на низькорівневі можливості, мову С проектували для машинно-незалежного програмування. Сумісна зі стандартами та машинно-незалежно написана мовою С програма може легко компілюватися на великій кількості апаратних платформ та операційних систем з мінімальними змінами. Мова стала доступною для великої кількості платформ – від вбудованих мікроконтролерів до суперкомп'ютерів [10, 11].

Метою даного підручника є викладення основних концепцій і особливостей мови програмування С. Підручник містить 19 розділів, кожен з яких включає теоретичний матеріал, забезпечений багатьма практичними прикладами. Приклади представлені у вигляді програм, розроблених у сучасному середовищі програмування Microsoft Visual Studio C++ 2013. З метою підвищення ефективності навчання рекомендується обов'язково практично опрацьовувати всі наведені приклади.

Наприкінці кожного розділу наведені контрольні питання, що дозволяють оцінити ступінь засвоєння пройденого матеріалу. Також для кожного розділу наведені тести для самоконтролю, побудовані за принципом один-з-багатьох. У кінці підручника наведено правильні відповіді на тести.

Додатки містять опис роботи у різноманітних інтегрованих середовищах програмування, принципи створення консольних багатомодульних програм та ін. Наведений предметний покажчик полегшує навігацію по навчальному матеріалу.

РОЗДІЛ 1 ОСНОВНІ ПОНЯТТЯ

1.1 Мова програмування С: історія створення мови, модель програмування, принципи створення програми

Мова С (СІ) була розроблена в 1972 р. Д. Рітчі (Dennis M. Ritchie) у фірмі Bell Laboratories. Витоком мови прийнято вважати мову BCPL (Basic Combined Programming Language), розроблену М. Річардсом (Martin Richards) в Кембриджі. У 1970 р. в Bell Labs Кеном Томпсоном (Ken Thompson) був розроблений варіант мови BCPL – мова В (БІ) для ранньої версії операційної системи (ОС) Unix. Недоліки мови В, зокрема, відсутність типів даних призвели до створення на її основі нової мови С.

Мова С спочатку створювалася для розробки ОС Unix, яка слабо залежить від конкретної архітектури ЕОМ і володіє розвиненими інструментальними можливостями. Переваги ОС Unix вивели її на одне з перших місць у світі за популярністю серед фахівців у галузі інформаційних технологій. Всі основні компоненти ОС Unix, транслятор і обслуговуючі програми написані на мові С.

Традиційно для розробки операційних систем та їх компонентів використовують Асемблери. Програмування на Асемблері дуже трудомістке й орієнтоване на конкретну архітектуру ЕОМ, що не відповідало завданню, поставленої перед розробниками ОС Unix. Мова С, як мова програмування високого рівня, вільна від цих недоліків. Вона не пов'язана з конкретною архітектурою ЕОМ. Компілятори мови функціонують на різних за архітектурою ЕОМ під управлінням ОС Unix, Windows, Linux і т. д. Технологія підготовки і налагодження програм на С характерна для мов високого рівня (Fortran, Pascal, Ada). Водночас їй притаманні багато рис мов "низького рівня" (доступ до адресів об'єктів і робота з ними, робота з бітовими величинами), що дозволяють організувати ефективне управління пристроями і програмами. За обсягом займаної пам'яті і часу виконання програми на С близькі до програм на Асемблері. У певному сенсі С займає проміжне положення між мовами високого рівня і машинно-орієнтованими мовами.

Для вироблення визначення мови С інститутом американських національних стандартів (ANSI) у 1983 році засновано спеціальний комітет. Результатом його роботи у 1989 році з'явився стандарт ANSI-C. Міжнародна організація по

стандартизації (International Standard Organization, ISO) у 1990 році прийняла свій стандарт ISO C, який практично збігається з ANSI C.

Новий міжнародний стандарт мови С ISO/IEC 9899:1999, що отримав назву С99, прийнятий в 1999 р. Новий стандарт переслідує мети розширення області застосування мови, його інтернаціоналізації та усунення явних недоліків, виявлених програмістами-практиками. Так, з метою інтернаціоналізації мови впроваджуються способи обробки міжнародних наборів символів. Поява стандарту С 99 говорить про велике значення мови С у міжнародному програмуванні.

Мова С використовує структурну модель програмування, в основі якої лежить уявлення програми у вигляді ієрархічної структури блоків. Вона була запропонована в 70-х роках XX століття Е. Дейкстрой, розроблена і доповнена Н. Віртом.

Відповідно до даної методології, будь-яка програма являє собою структуру, побудовану з трьох типів базових конструкцій:

− послідовне виконання – однократне виконання операцій в тому порядку, в якому вони записані в тексті програми;

− розгалуження – одноразове виконання однієї з двох або більше операцій, залежно від виконання деякої заданої умови;

− цикл – багаторазове виконання однієї і тієї ж операції доки виконується деяка задана умова (умова продовження циклу).

У програмі базові конструкції можуть бути вкладені одна в одну довільним образом, але ніяких інших засобів управління послідовністю виконання операцій не передбачається.

Повторювані фрагменти програми (або ті, що не повторюються, але представляють собою логічно цілісні обчислювальні блоки) можуть оформлятися у вигляді так званих підпрограм (процедур або функцій). У цьому випадку в тексті основної програми замість поміщеного в підпрограму фрагмента вставляється інструкція виклику підпрограми. При виконанні такої інструкції виконується викликана підпрограма, після чого виконання програми триває з інструкції, наступної за командою виклику підпрограми.

Розробка програми ведеться пошагово, методом «зверху вниз».

Спочатку пишеться текст основної програми, в якому замість кожного зв'язного логічного фрагмента тексту вставляється виклик підпрограми, яка буде виконувати цей фрагмент. Замість справжніх працюючих підпрограм в програму

вставляються «заглушки», які нічого не роблять. Отримана програма перевіряється та налагоджується. Після того, як програміст переконається, що підпрограми викликаються в правильній послідовності (тобто загальна структура програми вірна), підпрограми-заглушки послідовно замінюються на реально працюючі, причому розробка кожної підпрограми ведеться тим же описаним вище методом.

Розробка закінчується тоді, коли не залишиться жодної «заглушки», яка не була б видалена. Така послідовність гарантує, що на кожному етапі розробки програміст одночасно має справу з доступною для огляду і зрозумілою йому множиною фрагментів, і може бути впевнений, що загальна структура всіх вищих рівнів програми вірна. При супроводженні та внесення змін в програму з'ясовується, в які саме процедури потрібно внести зміни, і вони вносяться, не зачіпаючи частини програми, безпосередньо не пов'язані з ними. Це дозволяє гарантувати, що при внесенні змін і виправленні помилок не вийде з ладу якась частина програми, яка знаходиться в даний момент поза зоною уваги програміста.

1.2 Алфавіт мови. Лексеми

Алфавіт мови – це сукупність припустимих у мові символів чи груп символів, розглянутих як єдине ціле. Для мови С це символи стандарту ASCII, що задаються кодами від 0 до 255. Набір символов ASCII (зазвичай вимовляється як «аскєй») і його еквівалент ISO представляють собою спосіб кодування всіх літер, цифр і знаків пунктуації.

Елементи алфавіту можна умовно розбити на 4 групи:

– символи, що використовуються в ідентифікаторах;

– керуючі символи та роздільники;

– спеціальні символи;

– символи, що використовуються тільки в рядках символів і коментарях.

Лексема – найменша одиниця мови, що має певне значення в програмі – константа, ім'я, роздільник.

Ідентифікатор – ім'я будь-якого елемента програми: змінної, константи, типу, функції, мітки. Це ім'я повинне починатися з букви латинського алфавіту, за якою можуть йти інші букви, цифри і символ підкреслення.

Будь-який нестандартний ідентифікатор, використовуваний в операторах, що виконуються, повинен бути попередньо описаний. У мові С ідентифікатор

може бути описаний у **будь-якому** місці програми до його першого використання. Стандартні ідентифікатори, якщо вони використовуються в програмі, описувати не потрібно.

Вимога попереднього опису ідентифікаторів здається надмірно строгою, яка робить мову менш вільною. Насправді в ньому виявляється тенденція розвитку мов програмування убік підвищення надійності створюваних програм. Описати ідентифікатор – це значить указати тип зв'язаного з ним об'єкта програми.

До групи символів «керуючі символі на роздільники» належать: пробіл, символи табуляції, переведення рядка, повернення каретки, нова сторінка і новий рядок. Ці символи відокремлюють один від одного зарезервовані слова та об'єкти, визначені користувачем, до яких відносяться константи і ідентифікатори. Послідовність розділових символів розглядається компілятором як один символ (послідовність пробілів). Керуючі символи не відображаються на екрані, а виконують деякі дії [12].

Існують парні та непарні спеціальні символи. Парні спеціальні символи служать для позначення початку і кінця фрагменту тексту і використовуються тільки парно. До цієї групи відносяться різноманітні види, найпоширеніші такі:

- круглі дужки ‘(‘ та ‘)’;
- фігурні дужки ‘{‘ та ‘}’;
- квадратні дужки ‘[‘ та ‘]’;
- кутові дужки ‘<’ та ‘>’;
- подвійні лапки ‘"’ та ‘"’, найчастіше служать для позначення рядкових літералів;
- одинарні лапки ' та ', найчастіше служать для позначення рядкових літералів;
- /* та */, використовуються для позначення коментарів у багатьох мовах програмування.

Крім виділених груп символів в мові С широко використовуються так звані, керуючі послідовності, тобто спеціальні символьні комбінації, використовувані у функціях введення і виведення інформації. Керуюча послідовність будується на основі використання зворотної дробової риси (\) (обов'язковий перший символ) і комбінацією латинських букв і цифр (табл.1.1).

Таблиця 1.1 – Таблиця керуючих послідовностей

Керуюча послідовність	Призначення	Шістнадцятковий ASCII-код

\a	дзвоник	007
\b	повернення на шаг вліво (backspace)	008
\t	горизонтальна табуляція	009
\n	новий рядок	00A
\v	вертикальна табуляція	00B
\r	повернення каретки	00C
\f	перевод формату	00D
\”	символ ”	022
\’	апостроф	027
\0	ноль-символ	000
\\	зворотна коса риса \	05C
\ddd	символ набору кодів ПЕОМ в вісімковому представленні	
\xddd	символ набору кодів ПЕОМ в шістнадцятковому представленні	

Непарні символи можуть використовуватися, як парно так і поодиноко, наприклад знаки пунктуації (‘.’, ‘,’, ‘;’, ‘:’ і т.д.);

Також серед спеціальних символів можна виділити знаки операцій (арифметичних, логічних чи інших дій – '+', '–', '*', '/' і т.д.).

Також важливою частиною програми є коментарі – фрагменти коду, що не компілюються і дозволяють описати прямо в програмі призначення окремого рядка або цілого блоку. У коментарях, так само як і у рядках символів, можна використовувати букви російського (чи національного) алфавіту – рядкові та прописні.

Позначається коментар поєднанням слеша й зірочки (/*). Він дає команду компілятору ігнорувати все, що піде за символами (/*) до того моменту, поки не зустрінеться символ завершення коментаря: зірочка і слеш (*/). Кожній парі символів, що відкривається, /* повинна відповідати пара символів /*.

Однак, не рекомендується занадто перевантажувати програму безліччю коментарів. Коментувати програмний код слід тільки тоді, коли це необхідно для полегшення його читання і розуміння. Не слід, наприклад, описувати, що означає кожна змінна. Добре написаний програмний код не потребує додаткових коментарів.

1.3 Типи даних мов програмування

Будь-які дані, тобто константи, змінні, значення функцій чи вирази, характеризуються своїми типами [13]. Тип даних визначає, по-перше, спосіб внутрішнього для комп'ютера представлення об'єкта і, по-друге, дії, що дозволяється над ним виконувати. Формально типи даних потрібні для:

1) виділення місця в пам'яті;

2) визначення формату запису константи;

3) визначення операцій, які можна виконувати з константою.

На наступній схемі показана ієрархія типів даннях у мові С, при цьому вся множина типів розділяється на дві категорій: базові (вбудовані) і похідні типи (рис. 1.1) [14].

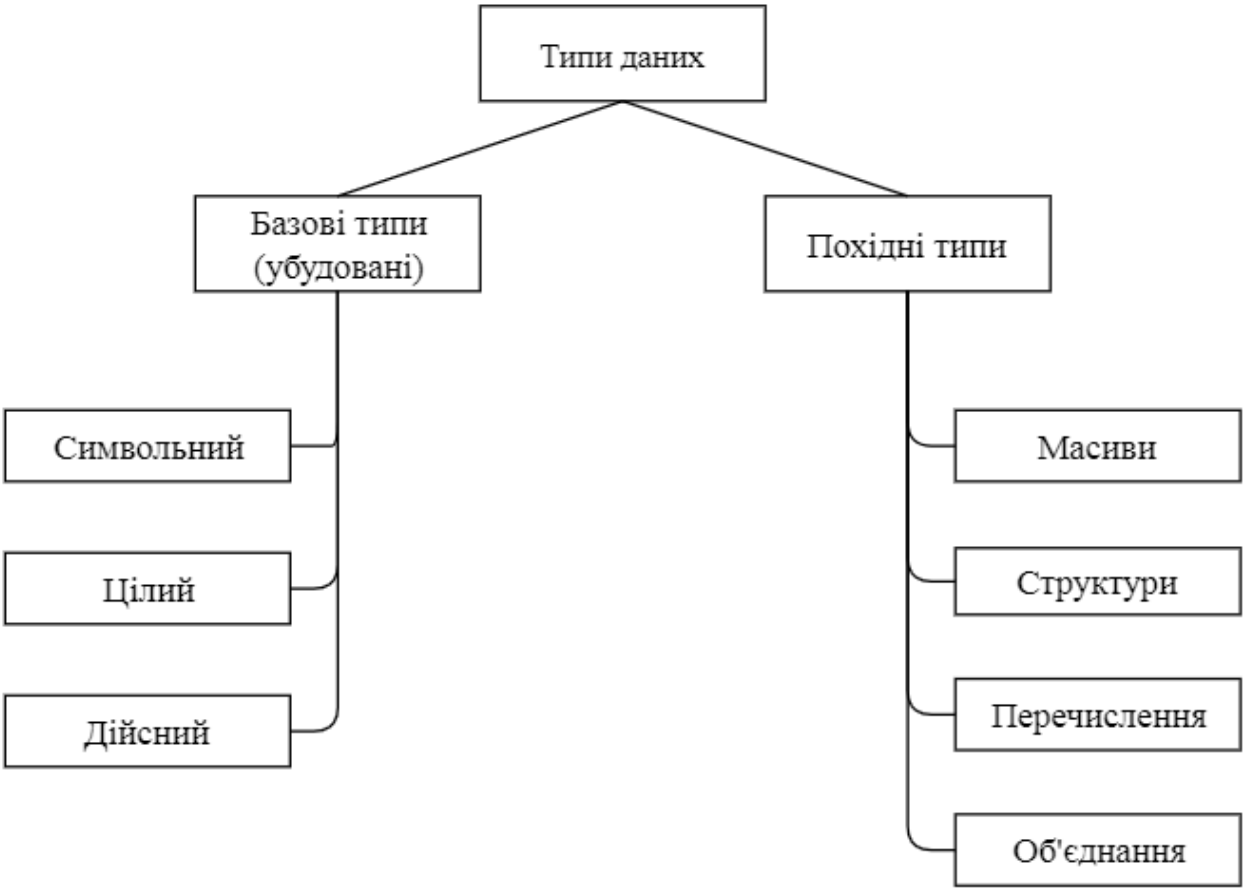


Рисунок 1.1 – Класифікація типів даних

Цілі типи діляться за знаком на знакові і беззнакові (`signed int` і `unsigned int`), а за розміром – на короткі довгі (`short int` і `long int`).

Таблиця 1.2 – Цілі типи даних

Позначення	Діапазон значень (залежить від реалізації)	Довжина в байтах
short (short int)	-32768..32767	2
int	-32768..32767	2
unsigned int	0..65535	2
long	-2147483648..2147483647	4
unsigned long	0..4294967295	4

З таблиці видно, що різні типи даних займають різний розмір в пам'яті, тому рекомендуємо враховувати, в якому діапазоні вам доведеться працювати. Однак оскільки пам'ять в наші дні стає все дешевше, можна використовувати тип int, еквівалентний signed short int.

Дійсні типи містять дробні значення чисел.

Таблиця 1.3 – Дійсні типи даних

Позначення	Діапазон значень (залежить від реалізації)	Довжина в байтах
float	$3.14*10^{-38}$ .. $3.14*10^{38}$	4
double	$1.7*10^{-308}$ .. $1.7*10^{308}$	8
long double	$3.4*10^{-4932}$ .. $1.1*10^{4932}$	10

Як видно з таблиць, діапазони значень можуть залежати від конкретної реалізації (32-біт, 64-біт, ANSI). Детальніше дивись про це у заголовних файлах `limits.h` та `float.h`.

Для оголошення змінної необхідно вказати її тип, потім ім'я самої змінної. Також можна поєднати оголошення з ініціюванням, додавши символ '=' і значення, що присвоюється [15].

Приклад оголошення:

```
float x,y=2.34,z=0;
double p;
```

Тип даних bool використовується виключно для зберігання результатів логічних виразів. У логічного виразу може бути один з двох результатів true або false. true – якщо логічний вираз істинний, false – якщо логічний вираз помилковий. Але оскільки діапазон допустимих значень типу даних bool від 0 до 255, то необхідно було якось зіставити даний діапазон з визначеними в мові

програмування логічними константами true і false. Таким чином, константі true еквівалентні всі числа від 1 до 255 включно, тоді як константі false еквівалентно тільки одне ціле число – 0.

Хоча стандарт С89 не передбачає логічний тип, в стандарті С99 був доданий тип _Bool що дозволяє зберігати значення 1 і 0 (тобто істинне і помилкове). На відміну від мови С++ в стандарті С99 немає ключових слів true і false. Отже, тип _Bool, визначений у стандарті С99, несумісний з типом bool, визначеним у мові С++. Однак у багатьох існуючих програмах на мові С програмісти визначали свій власний варіант типу bool.

Символьний тип даних char призначений для збереження або одного символу (типографського знаку), або керуючого коду. Для збереження одного даного цього типу виділений 1 байт.

Діапазони представлення значень:

char -128 – 127

unsigned char 0 – 255

Приклади типографський знаків: ‘s’, ‘я’, ‘U’, ‘П’, ‘5’,’*’, ‘?’.

Приклад оголошення:

```
char a,b,c='*';
a='0'; b='!';
```

1.4 Конверсія типів даних

У мові С існують ручна і автоматична конверсії типів даних. У програмах в одному виразі можна зустріти змінні різних типів int чи float, такі режими роботи називають змішаними (mixed). На відміну від інших мов програмування, С виконує автоматичну конверсію типів даних. Дані різних типів по-різному зберігаються в пам'яті [16].

Наприклад:

```
float_result=float_value2*int value;
```

Коли цей вираз буде виконуватися, то `int_value` буде перетворене в `float_point` перед тим як операція виконається, тобто компілятор візьме з пам'яті `int_value` у якусь комірку пам'яті, значення перетвориться у відповідне `float_value`, тобто буде займати 4 байти. При цьому значення `int_value` залишається незміненим, тобто у форматі цілого числа.

Існує ієрархія конверсій, при якій об'єкт низького пріоритету конвертується в об'єкт вищого пріоритету на етапі тимчасового виконання обчислень. Отже, ієрархія конверсій від нижчого до вищого пріоритетам

`short > int > long > float > double`,

тобто перетворення виконуються по типу, що містить більшу кількість значущих цифр.

```
int_value=3;
float_value=4.5;
float_result= int_value+ float_value;
```

Значення змінної `int_value` у рамках обчислення перетвориться в 3.0 і результат `float_result` буде 7.5.

Тепер розглянемо випадок, коли має місце конверсія від `float` до `int`.

```
int_value1=3;
int_vaue2=4;
float_value=7.0;
float_result=float_value+int_value1/int_value2;
```

Вираз `int_value1/int_value2` – не змішана операція, тому що в ньому присутні дві цілі змінні. Хоча результатом буде 0.75, якщо таке число розмістити в комірці пам'яті, оголошеної як ціле, дробова частина відкидається. Тому результат, збережений в `int_value1` 7.0.

Тип результату виразу буде таким, яким була оголошена змінна, що розташована ліворуч від оператора присвоювання.

```
float_x=7.0;
float_y=2;
int_result=4.0 +float_x/float_y,
```

Результат операції `float_x/float_y` становить 3.5, коли він додається до 4.0, то становить 7.5. Однак його не можна зберегти в `int_result`, тому що для неї виділене місце типу `int`, тобто дробова частина буде відкинута, а результат – число 7.5, – таким чином, конвертується в ціле – дробова частина буде загублена.

Хоча існує автоматична конверсія типів, потрібно, щоб була і ручна конверсія. Якщо необхідно змінити тип змінної у програмі, ця змінна попереджається типом у круглих дужках (типом, у який потрібно перетворити змінну).

Три наступних рядка виконують однакові операції:

```
float_result=float_value+(float)int_value1/int_value2;
```

```
float_result=float_value+int_value1/(float)int_value2;
float_result=float_value+(float)int_value1/(float)int_value2;
```

1.5 Зарезервовані слова в С

Зарезервовані, або ключові слова – це зарезервовані ідентифікатори, які наділені певним змістом. Їх можна використовувати тільки у відповідності зі значенням, яке відомо компілятору мови С [17].

Таблиця 1.4 – Ключові слова

auto	break	case	char
const	contine	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

1.6 Контрольні питання

1. Що таке алфавіт мови? Яким є алфавіт для мови програмування С?
2. Що таке лексема?
3. Ідентифікатор: призначення, правила створення. Стандартні та нестандартні ідентифікатори.
4. Призначення роздільників. Парні та непарні розділові символи. Групи роздільників.
5. Керуючі символи: позначення, призначення, ASCII-код.
6. Коментарі в програмі. Правила оформлення однорядкових та багаторядкових коментарів.
7. Типи даних: призначення, категорії, ієрархія типів.
8. Цілі типи даних: позначення, діапазон значень, довжина в байтах. Приклади оголошень.

9. Дійсні типи даних: позначення, діапазон значень, довжина в байтах. Приклади оголошень.

10. Тип даних `bool`: призначення, допустимі значення, зіставлення з числовими значеннями. Приклади оголошень.

11. Символьний тип `char`: призначення, діапазони значень для знакового та беззнакового типів `char`. Приклади оголошень.

12. Автоматичне та ручне перетворення (конверсія) типів. Змішані операції. Ієрархія конверсій.

13. Ключові (зарезервовані) слова.

1.7 Тести для самоконтролю

1) До якої моделі належить мова програмування С?

А) логічна

Б) об'єктно-орієнтована

В) структурна

Г) немає правильної відповіді

2) На якій мові ґрунтується мова програмування С?

А) В

Б) Algol

В) Cobol

Г) Pascal

3) Ким була створена мова С?

А) Н. Віртом

Б) Б. Гейтсом

В) співробітником фірми Bell Labs Денісом Рітчи

Г) групою розробників фірми Panasonic під керівництвом Кена Томпсона

4) Чи є у мови С власний редактор?

А) так

Б) ні

В) тільки в ОС UNIX

Г) тільки в ОС WINDOWS

5) Чим визначається мобільність мови С?

А) будь-яка програма, написана на Сі для однієї обчислювальної системи, може бути перенесена без жодних змін на іншу систему

Б) програма, написана на С для однієї обчислювальної системи, може бути перенесена з невеликими змінами або взагалі без них на іншу

В) мобільність мови С визначається її ефективністю

Г) відсутністю переносимості

6) Як створити проект у IDE Visual Studio? Послідовно вибрати:

А) Проект Новий Проект

Б) Файл Новий Проект

В) Проект Новий

Г) Файл Новий Проект

7) З допомогою якої директиви можна підключити бібліотеку?

А) `#include`

Б) `#define`

В) `#ifdef`

Г) `#endif`

8) Як виконати коментування для кількох рядків?

А) `//`

Б) `#`

В) `##`

Г) `/*…*/`

9) До цілого типу не відноситься такий тип як

А) `short int`

Б) `struct`

В) `int`

Г) `long`

10) До дійсного типу не відноситься такий тип як

А) `float`

Б) `double`

В) `long double`

Г) `bool`

11) Що з наступного не може бути збережено як елемент типу `char`?

А) ‘A’

Б) `‘*’`

В) ‘char’

Г) ‘\n’

12) Що буде при виконанні даного коду: `int IntVal; char CharVal; CharVal = IntVal;`

А) все буде виконуватися успішно

Б) далі в програмі не можна буде користуватися цими змінними

В) можлива втрата інформації через присвоєння більшого типу до меншого

Г) немає правильної відповіді

13) Неявне перетворення виконується як …

А) перетворення до більш загального типу (супертипу)

Б) перетворення до меншого типу

В) перетворення у проміжний тип

Г) у мові С неявного перетворення не існує

14) На що вказує модифікатор типу `signed`?

А) змінна приймає тільки від'ємні значення

Б) змінна може приймати як позитивні, так і від'ємні значення

В) змінна приймає тільки позитивні значення

Г) немає правильної відповіді

15) На що вказує модифікатор типу `unsigned`?

А) змінна приймає тільки парні від'ємні значення

Б) змінна приймає усі значення

В) змінна приймає невід'ємні значення

Г) змінна приймає буквено-символьні значення

16) Яке з слів зарезервовано як службове в мові С?

А) `goto`

Б) `def`

В) `var`

Г) `private`

РОЗДІЛ 2 СКЛАДОВІ ПРОГРАМИ

2.1 Структура програми. Ввід та вивід даних

У загальному випадку програма на мові С визначається як сукупність одного чи декількох модулів – файлів, що самостійно компілюються. Такий файл звичайно містить директиви препроцесора, а також одну чи кілька функцій, що складаються з операторів мови С. З іншого боку, програму на мові С можна розглядати як сукупність наступних елементів: директив препроцесора, указівок компілятору, оголошень і визначень [18].

Директиви препроцесора уточнюють дії препроцесора по перетворенню тексту програми перед компіляцією. Указівки компілятору – це спеціальні інструкції, які впливають на процес компіляції.

Вихідний файл може містити будь-як цілісну комбінацію директив препроцесора, указівок компілятору, оголошень і визначень. Під цілісністю мається на увазі, що такі об'єкти, як визначення функцій, структури даних, або зв'язані між собою директиви умовної компіляції, повинні цілком розташовуватися в одному файлі, тобто не можуть починатися в одному файлі, а продовжуватися в іншому.

Традиційно в програмах використовуються два види файлів: файли реалізації, що мають розширення .c (.cpp), і заголовні файли (файли інтерфейсу), що мають розширення .h. У заголовних файлах звичайно розташовуються іменовані константи, макровизначення і оголошення, а у файлах реалізації – визначення змінних і функцій.

Програма мовою С має гнучку структуру. Спочатку наводяться директиви препроцесора, що обробляються до компіляції програми. За допомогою #include підключаються заголовні файли, що містять інформацію і оголошення, необхідні компілятору для обробки викликів стандартних функцій. Далі при необхідності можуть бути приведені глобальні описи змінних, констант, типів та ін.

В кожній програмі обов'язково повинна бути присутня функція main(), з якої починається виконання програми. Функція main() містить код, що виконується при запуску програми. Перед ім'ям функції main() указується тип значення, що повертається функцією; якщо значення не повертається, то пишеться void. У круглих дужках перелічуються необов'язкові параметри функції.

Оголошення констант, типів, змінних і інших блоків програми можна наводити в будь-якій частині цього блока до їхнього першого використання чи при першому використанні. Формально структуру програми наведено на наступній схемі (рис. 2.1).

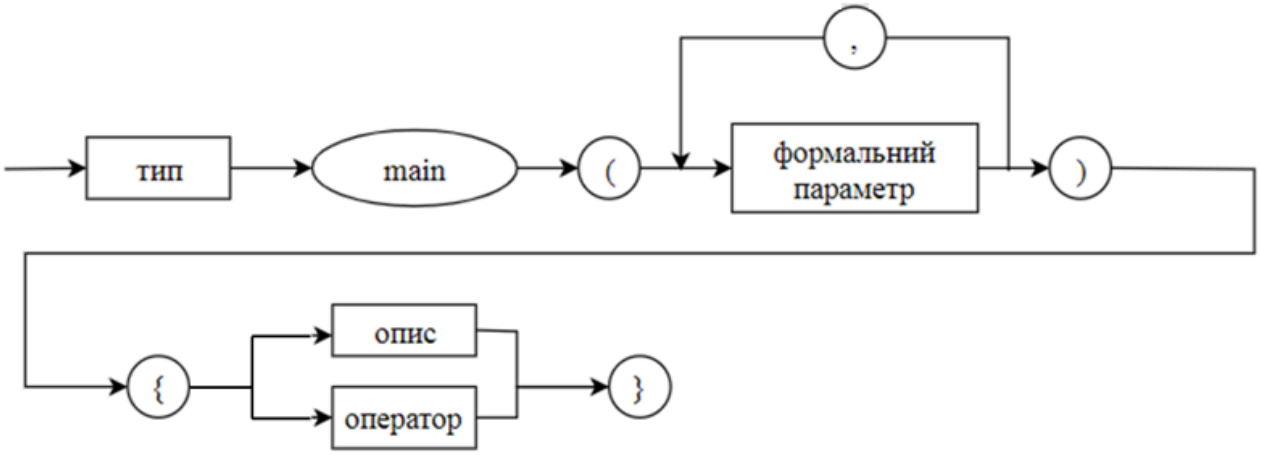


Рисунок 2.1 – Формальне представлення структури програми

2.2 Функції стандартного вводу та виводу

Для вводу і виводу даних в програмах на мові С використовуються функції `printf()` і `scanf()`. Для використання в програмі цих функцій необхідно підключити заголовний файл «stdio.h» [19].

Функція printf()

Функція `printf()` є функцією стандартного виводу. За допомогою цієї функції можна вивести на екран монітора рядок символів, число, значення змінної. Функція `printf()` повертає число виведених символів у разі успіху або від'ємне значення у випадку помилки.

Прототип функції `printf()`:

```
int printf(const char * керуючий рядок, ...);
```

Керуючий рядок складається з елементів двох видів. Перший з них – це символи, які належить вивести на екран; другий – це специфікатори перетворення, які визначають спосіб виведення наступних аргументів. Кожен такий специфікатор починається із символу %, за яким йде код формату. Наступні параметри – змінні, значення яких необхідно вивести.

Рядок формату читається зліва направо. Коли зустрічається перша специфікація формату (якщо вона є), то значення першого аргументу після рядка

формату перетворюється і виводиться згідно заданої специфікації. Друга специфікація формату викликає перетворення та виведення другого аргументу і так далі, до кінця рядка формату.

Якщо аргументів більше, ніж специфікацій формату, то ці додаткові аргументи ігноруються. Результат є невизначеним, якщо аргументів недостатньо для всіх специфікаций формату. Найпростіша специфікація формату містить тільки символ знаку відсотка і символ типу.

Таблиця 2.1 – Деякі специфікатори формату

%c	символ
%d або %i	ціле десяткове число
%f або %F	десяткове число з плаваючою комою xx.xxxx
%u	десяткове ціле без знака
%s	рядок символів
%%	символ %
%p	покажчик
%e або %E	експоненціальне подання (‘e’ – на нижньому регістрі, ‘E’) – на верхньому
%x або %X	шістнадцяткове без знака (літери на нижньому або на верхньому регістрі)
%p	виводить покажчик
%o	восьмеричне без знака
%g або %G	залежно від того, який вивід буде коротшим, використовується %e або %f (%E або %F)

Крім того, до команд формату можуть бути застосовані деякі модифікатори.

Таблиця 2.2 – Модифікатори l і h

%ld	друк long int
%hd	друк short int
%lf	друк double

Крім того, у багатьох специфікаторах перетворення можна додатково вказати модифікатори, які з легкістю міняють їх значення. Наприклад, можна вказувати мінімальну ширину поля, кількість десяткових розрядів і вирівнювання по лівому краю. Модифікатор формату поміщають між знаком відсотка і кодом формату.

Модифікатори мінімальної ширини поля

Ціле число, розташоване між знаком % і кодом формату, грає роль модифікатора мінімальної ширини поля. Якщо вказаний модифікатор мінімальної ширини поля, то для того, щоб ширина поля виводу була не меншою зазначеної мінімальної довжини, при необхідності вивід буде доповнений пробілами. Якщо ж виводяться рядки або числа, які довше зазначеного мінімуму, то вони все одно будуть відображатися повністю. За замовчуванням для доповнення використовуються пробіли. А якщо для цього треба використовувати нулі, то перед модифікатором ширини поля слід помістити 0. Наприклад , %05d означає, що будь-яке число, кількість цифр якого менше п'яти, буде доповнено такою кількістю нулів, щоб число складалося з п'яти цифр.

Приклад. Демонстрація застосування модифікатора мінімальної ширини поля.

```
#include <stdio.h>
int main()
{
  double value;
  value = 12.34567;

  printf("%f\n", value);
  printf("%10f\n", value);
  printf("%014f\n", value);

  return 0;
}
```

Отриманий результат:

```
12.345670
 12.345670
0000012.345670
```

Модифікатор мінімальної ширини поля найчастіше використовується при створенні таблиць, в яких стовпці повинні бути вирівняні по вертикалі. Крім специфікаторів формату даних в керуючому рядку можуть перебувати керуючі символи (табл. 1.1). Найчастіше використовуються символи \n і \t, за допомогою яких можна переходити на новий рядок або робити відступ таблуляції.

Приклад. Вивести таблицю первинних значень разом з їх вдвічі збільшеними та вдвічі зменшеними значеннями.

```
#include <stdio.h>
int main()
{
     float i;

     i=23.4;
     printf(“%12f  %12f  %12f\n”, i, i*2, i/2);

     i=47.432;
     printf(“%12f  %12f  %12f\n”, i, i*2, i/2);

     i=-190.0;
     printf(“%12f  %12f  %12f\n”, i, i*2, i/2);
return 0;
}
```

Отриманий результат:

```
   23.400000   46.799999    7.000000
   47.431999   94.863998   15.810666
 -190.000000 -380.000000  -63.333333
```

Варто зауважити, що отримання значення `46.799999` замість `46.800000` при подвоєнні `23.400000` пов'язано з машинною точністю обчислень.

Модифікатори точності

Модифікатор точності йде за модифікатором мінімальної ширини поля (якщо такий є). Він складається з точки і розташованого за нею цілого числа. Значення цього модифікатора залежить від типу даних, до яких його застосовують. Коли модифікатор точності застосовується до даних з плаваючою точкою, для перетворення яких використовуються специфікатори перетворення %f , %e або %E, то він визначає кількість виведених десяткових розрядів. Наприклад , %10.4f означає , що ширина поля виводу буде не менше 10 символів, причому для десяткових розрядів буде відведено чотири позиції. Якщо модифікатор точності застосовується до %g або %G, то він визначає кількість значущих цифр. Застосований до рядків, модифікатор точності визначає максимальну довжину

поля. Наприклад , %5.7s означає, що довжина виводимого рядка становитиме мінімум п'ять і максимум сім символів. Якщо рядок виявиться довше, ніж максимальна довжина поля, то кінцеві символи виводитися не будуть. Якщо модифікатор точності застосовується до цілих типів, то він визначає мінімальну кількість цифр, які будуть виведені для кожного з чисел. Щоб отримати необхідну кількість цифр, додається деяка кількість провідних нулів.

Приклад. Застосування модифікаторів точності.

```
#include <stdio.h>
int main()
{
     printf("%.3f\n", 123.987654);
     printf("%2.5d\n",345);
     printf("%5.15s\n","double float int char");
     printf("%5s\n","double float int char");
     printf("%15s\n","double float int char");
 return 0;
}
```

Отриманий результат з коментарями:

```
123.988 (результат округлений до 3 цифри після коми)
00345 (на початок рядка добавлені нулі)
double float in (довжина рядка 15 символів)
double float int char (точність не встановлена)
double float int char (точність не встановлена)
```

Вирівнювання виводу

За замовчуванням весь вивід вирівнюється по правому краю. Тобто якщо ширина поля більше ширини виведених даних, то ці дані розташовуються по правому краю поля. Вивід по лівому краю можна призначити примусово, помістивши знак мінус прямо за %. Наприклад, %-l0.2f означає, що число з плаваючою точкою і з двома десятковими розрядами буде вирівняно по лівому краю 10-символьного поля.

Приклад. Застосування вирівнювання виводу.

```
#include <stdio.h>
int main()
{
     printf("12345678\n"); // нумерація позицій
```

```
    printf("%8d\n", 100); // по правому краю
    printf("%-8d\n", 100); // по лівому краю
return 0;
}
```

Отриманий результат:

```
12345678
    100
100
```

Відзначимо той факт, що, якщо зворотна дробова риса передує символу, який не входить в керуючу послідовністю (тобто не включений до табл. 1.1) і яка не є цифрою, то ця риса ігнорується, а сам символ представляється як літеральний. Наприклад: символ \h представляється символом h в строковой або символьної константі. Окрім визначення керуючої послідовності, символ зворотної дробової риси (\) використовується також як символ продовження. Якщо за (\) стоїть (\n) , то обидва символи ігноруються, а наступний рядок є продовженням попереднього. Ця властивість може бути використана для запису довгих рядків.

Функція scanf()

Функція `scanf()` – функція форматованого вводу. З її допомогою можна вводити дані зі стандартного пристрою вводу (з клавіатури). Такими даними можуть бути цілі числа, числа з плаваючою комою, символи, рядки і покажчики. При введенні чисел роздільниками можуть бути символи пробілу, табуляції або нового рядка.

Прототип функции `scanf(): int scanf(const char * керуючий рядок, …)`

Все сказане для функції `printf()` можна застосувати і до `scanf()`. Перший параметр – рядок зі специфікаторами формату, наступні – змінні, куди будуть записані введені значення. Слід звернути увагу на те, що передавати змінні-параметри в функцію `scanf()` необхідно за адресою, тобто перед ідентифікатором змінної ставити символ & (наприклад `scanf ("%d", &x);`). Детальніше про передачу параметрів у функцію за адресою буде сказано далі.

Функція `scanf()` повертає число змінних, яким було присвоєно значення.

Приклад. Обчислити вираз `c=a*b+2.5`. Здійснити введення з використанням функції `scanf()`.

```
#include <stdio.h>
void main()
{
     int a;
     float b, c;
     printf("Input the variables a, b\n");
     scanf("%d%f", &a, &b);
     c=a*b+2.5;
     printf("The result is %f", c);
}
```

Введення рядків

Для читання з вхідного потоку рядка можна використовувати функцію `scanf()` зі специфікатором перетворення %s. Використання специфікатора перетворення %s змушує `scanf()` читати символи доки не зустрінеться який-небудь роздільник (пробіл, роздільник рядків, табуляція, вертикальна табуляція або подача сторінки). Зчитувані символи поміщаються в символьний масив, на який вказує відповідний аргумент, а після закінчення вводу до символів додається ще символ кінця рядка ('\0'). Якщо потрібно вводити рядки з пробілами, то використовуйте функцію `gets`:

```
char *gets( char *buf );
```

За допомогою функції `gets()` можна вводити повноцінні рядки. Функція `gets()` читає символи з клавіатури до появи символу нового рядка (`\n`). Сам символ нового рядка з'являється при натисканні на клавішу ENTER. Функція повертає покажчик на `buf`. `buf` – це буфер (пам'ять) для введеного рядка.

Управління зчитуванням символів з потоку вводу

Символ роздільника в керуючому рядку дає команду пропустити один або більше роздільників у потоці вводу. Роздільниками є пробіли, горизонтальні табуляції, вертикальні табуляції, подачі сторінок і роздільники рядків. По суті, один роздільник у керуючому рядку змушує `scanf()` читати, але не зберігати будь-яку кількість (у тому числі і нульову) роздільників, які знаходяться перед першим символом, що не є роздільником.

Також scanf() може прочитати поле, але не привласнювати прочитане значення ніякої змінної; для цього треба перед літерою-специфікатором формату поля поставити зірочку *.

Наприклад:

```
scanf ("%d%*c%d", &value1, &value2);
```

при введенні 30-20 присвоїть змінній value1 значення 30, змінній value2 – значення 20, а символ «-» буде прочитаний і проігнорований.

Аналогічно функції printf(), для scanf() у команді формату може бути вказана найбільша ширина поля, яка підлягає зчитуванню. Наприклад, якщо користувач увів рядок «abcdefghijk» у відповідь на команду

```
scanf("%4s",temp);
```

то змінна temp прийме значення abcd, а інші символи будуть проігноровані.

Якщо в керуючому рядку зустрічаються які-небудь інші символи, то вони призначаються для того, щоб визначити і пропустити відповідний символ. Потік символів 123plus456 оператором

```
scanf("%dplus%d", &x, &y);
```

присвоїть змінній x значення 123, змінній y – значення 456, а символи plus пропустить, так як вони зустрілися в керуючому рядку.

Множина пошуку (scanset)

Однією з потужних особливостей функції scanf() є можливість завдання множини пошуку (scanset). Множина пошуку визначає набір символів, з якими будуть порівнюватися символи, що читаються функцією scanf(). Функція scanf() читає символи доки вони зустрічаються в множині пошуку. Як тільки введено символ, який не зустрівся в множині пошуку, функція scanf() переходить до наступного специфікатора формату. Множина пошуку визначається списком символів, укладених у квадратні дужки. Перед відкриваючою дужкою ставитися знак %.

Приклад. Використання множини пошуку при введенні даних.

```
#include <stdio.h>
void main()
{
    char s1[15], s2[20];
    scanf("%[abcdefg]%s", s1, s2);
    printf("\ns1=%s\ns2=%s",s1,s2);
```

```
}
```

Введемо набор символів:

```
dabcjytop32ab
```

Отриманий результат:

```
s1=dabc
s2=jytop32ab
```

При завданні множини пошуку можна також використовувати символ "дефіс" для завдання проміжків, а також максимальну ширину поля введення. Наприклад:

```
scanf("%10[a-z1-9]", str1);
```

вводить максимум 10 символів з нижнього регістру латинського алфавіту та арабських цифр крім 0.

Можна також визначити символи, які не входять до множини пошуку. Перед першим з цих символів ставитися знак ^. Множина символів, звичайно, розрізняє малі та великі літери.

Слід звернути увагу на такий важливий момент: набір сканованих символів чутливий до регістру букв. Якщо потрібно сканувати букви і на верхньому, і на нижньому регістрі, то їх треба вказувати окремо для кожного регістру.

2.3 Контрольні питання

1. Загальна структура програми на мові програмування С.
2. Що таке директиви препроцесора?
3. Яке розширення мають файли реалізації та заголовні файли?
4. Призначення функції `main()`.
5. В якому місці програми оголошуються константи, типи, змінні та ін.?
6. Наведіть формальну схему структури програми.
7. Що таке структурна схема програми?
8. Які ви знаєте основні елементи блок-схем? Для чого вони служать та як зображуються?
9. Яка функція використовується для стандартного виводу? Наведіть прототип цієї функції. Що повертає функція? Що містить керуючий рядок?
10. Які ви знаєте специфікатори формату? Наведіть приклади.
11. Принцип роботи модифікатора мінімальної ширини поля. Наведіть приклади для створення таблиць.

12. Принцип роботи модифікатора точності. Наведіть приклади для виводу даних різних типів.

13. Вирівнювання виводу. Наведіть приклади вирівнювання виводу по правому та лівому краям.

14. Яка функція використовується для стандартного вводу? Наведіть прототип цієї функції. Що повертає функція? Що містить керуючий рядок?

15. Введення рядків за допомогою функції `scanf()`: потрібний специфікатор, обмеження використання. Яка існує альтернатива функції `scanf()` для введення рядків з пробілами?

16. Управління зчитуванням символів з потоку вводу: правила обробки роздільників, застосування *.

17. Завдання множини пошуку в керуючому рядку `scanf()`. Визначення символів, які не входять до множини пошуку.

18. Рекомендації щодо побудови схем алгоритмів.

2.4 Тести для самоконтролю

1) Яке розширення мають файли реалізації?

А) `.prog`

Б) `.h`

В) `.cpp`

Г) `.c++`

2) У програмі на мові С функція `main()`

А) використовується для опису змінних

Б) містить код, що виконується при запуску програми

В) є необов'язковою

Г) підключає заголовні файли

3) Для графічного зображення алгоритму використовується

А) структурна схема алгоритму

Б) діаграма класів

В) математична модель алгоритму

Г) немає правильної відповіді

4) Оберіть невірне твердження:

А) будь-який алгоритм має тільки один вхід

Б) будь-який алгоритм має тільки один вихід

В) для відображення переходу управління між блоками використовується символ лінія

Г) символ «паралелограм» використовується для коментування алгоритму

5) Оберіть вірне твердження:

А) функція `printf()` є функцією стандартного виводу

Б) функція `printf()` є функцією нестандартного виводу

В) функція `printf()` є функцією стандартного вводу

Г) функція `printf()` є функцією нестандартного вводу

6) Який специфікатор формату використовує функція printf() для вводу символу?

А) `%d`

Б) `%c`

В) `%f`

Г) `%s`

7) Модифікатор мінімальної ширини поля у функції `printf()`

А) використовувати обов’язково

Б) використовується тільки для цілих чисел

В) використовується тільки для дійсних чисел

Г) не працює, якщо виводиться рядок, довжина котрого більше зазначеного мінімуму

8) Модифікатор точності у функції `printf()`

А) йде перед модифікатором мінімальної ширини поля

Б) складається з точки і розташованого за нею цілого числа

В) не залежить від типу даних, до яких його застосовують

Г) не використовується для рядків

9) Як у функції `printf()` вирівняти весь вивід по правому краю?

А) поставити після «%» знак «-»

Б) поставити після «%» знак «+»

В) це зробити неможливо

Г) це виконується автоматично

10) Функція `scanf()` використовується для

А) вводу з файла

Б) виводу в файл

В) вводу з клавіатури

Г) виводу на екран

11) У функції `scanf()` перед ідентифікатором змінної треба ставити

А) символ «&» для всіх змінних, крім рядків

Б) символ «&» для всіх змінних

В) символ «*» для всіх змінних

Г) круглі дужки

12) Як можна проігнорувати введення символу у функції `scanf()`?

А) указати символ останнім в черзі вводу

Б) поставити перед специфікатором для відповідного символу «*»

В) вказати невірний формат

Г) не вказувати символ «&» перед ідентифікатором змінної

13) Множина пошуку у функції `scanf()`

А) дозволяє вводити символи без специфікатора формата

Б) дозволяє вводити рядки з пробілами

В) визначає набір символів, які будуть читатись та ігноруватись

Г) визначає набір символів, які будуть читатись та привласнюватись змінним.

РОЗДІЛ 3 ЗМІННІ ТА КОНСТАНТИ

3.1 Основні положення: змінні, константи

Змінними називаються об'єкти програми, що можуть змінювати своє значення в процесі виконання програми. З кожною змінною зв'язується ідентифікатор, за ім’ям якого надалі відбувається звертання до вмісту області пам'яті, що займає змінна. Значення змінної може бути введено користувачем, змінено, виведено, а також воно може бути використано у виразах [20].

Для мови С оголошення змінної можливо в будь-якій частині програми до першого використання цієї змінної. Змінна навіть може бути оголошена в тому виразі, у якому вона використовується вперше. Після вказівки типу перелічуються змінні цього типу.

На наступній схемі формалізоване оголошення змінних для випадків оголошення однієї змінної без ініціалізації (присвоювання початкового значення), декількох однотипних змінних без ініціалізації, однієї змінної з ініціалізацією, декількох однотипних змінних з ініціалізацією (рис. 3.1).

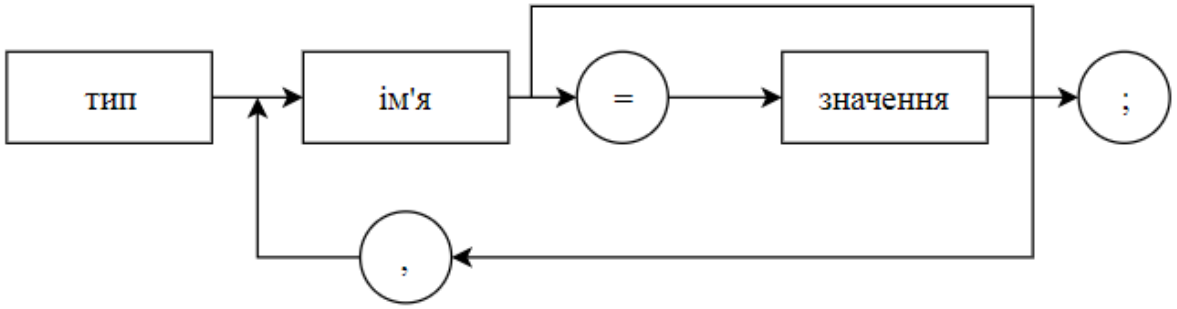


Рисунок 3.1 – Формальне представлення оголошення змінних

Наприклад:

```
int i, j;
float a=0, b=1.1, c=2.34;
bool b1, b2;
```

Константи – це лексеми, що представляють собою фіксовані числові чи символьні значення. На відміну від змінних їх значення не може бути змінено під час виконання програми.

На наступній схемі формалізоване оголошення констант, при чому тип константи вказувати не обов’язково.

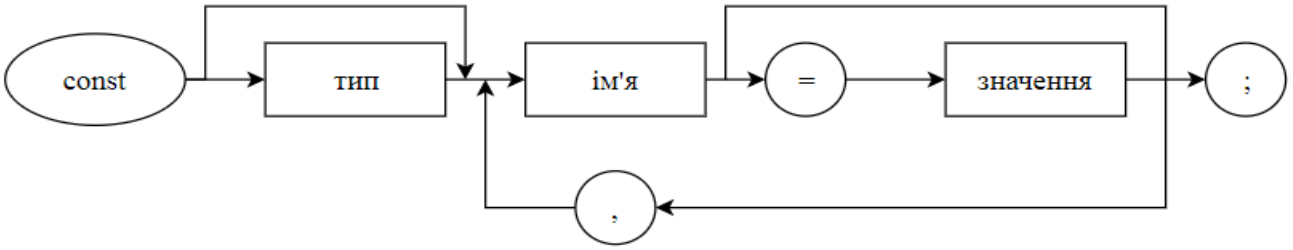


Рисунок 3.2 – Формальне представлення оголошення констант

Наприклад:

```
const n=3; // за замовчуванням буде цілий тип
const t=3.45; // за замовчуванням константи дійсного типа мають тип double
```

Також класичним способом при роботі з константами є застосування директиви препроцесора `#define`. Наприклад: `#define SIZE 255`
Докладніше про директиву `#define` та команди препроцесору дивіться розділ 15.

У мові С існує чотири типи констант: цілі, дійсні, символьні і типу, що перелічується.

Цілі числа можна виразити в одній із трьох систем числення: десятковій (по підставі 10), шістнадцятирічній (по підставі 16) і восьмеричної (по підставі 8).

Таблиця 3.1 – Представлення цілих чисел у системах числення

Тип	Синтаксичне правило	Приклад
Десятковий	Перша цифра 1-9, наступні 0-9	77 -512 77385L
Шістнадцятирічний	Починається з 0x чи 0X; наступні цифри 0-9, a-f чи A-F	0x4D -0X200 0x12e49L
Восьмирічний	Починається з 0; наступні цифри 0-7	0115 -01000 017067L

Єдине зауваження, яке варто зробити, полягає у тому, що будь-яка константа, що лежить поза діапазоном цілих чисел одинарної точності, чи константа, що закінчується на l чи L, розглядається як довге ціле число.

Дійсні константи, називані також константами з плаваючою крапкою, являють собою числа по підставі 10. З погляду мови С можна відзначити два найбільш істотні розходження між цілими і дійсними константами:

1) Дійсні константи завжди містять десяткову крапку і/чи позначення показника ступеня E (чи e) – наукова нотація.

2) Дійсні числа можуть починатися з цифри 0, однак завжди інтерпретуються як значення по підставі 10.

Символьна константа – це один символ, укладений в одинарні лапки (апострофи) ('). Весь використовуваний набір символів можна розділити на дві групи: символи, що друкуються і символи, що не друкуються. Символи, що друкуються, містять у собі букви, цифри й інші знаки, які можна набрати на клавіатурі. Символам, що не друкуються, відповідають спеціальні керуючі послідовності, називані також esc-послідовностями (наприклад, код переходу до нового рядка), що служать для керування зовнішніми пристроями чи для інших видів керування. У будь-якому випадку важливо пам'ятати, що в пам'яті комп'ютера всі значення представляються у виді двійкових чисел.

Приклади символів, що друкуються, досить тривіальні:
'G', 'g', '7', 'ю', '&'.

Константи типу, що перелічується, являють собою ідентифікатори, визначені за допомогою оголошення типу enum, котрі служать як мнемонічні імена, що підвищують читабельність програми.

3.2 Інструкція typedef

Мова С надає інструкцію `typedef`, яка дозволяє давати існуючих типів даних нові імена [21]. Наприклад, оголошення

```
typedef int Length;
```

робить ім'я `Length` синонімом `int`. З цього моменту тип `Length` можна застосовувати в оголошеннях, в операції приведення типу і т.д. точно так само, як тип `int`:

```
Length len, maxlen;
```

Слід підкреслити, що оголошення `typedef` не створює оголошення нового типу, воно лише дає нове ім'я вже існуючого типу. Ніякого нового сенсу ці нові імена не несуть, вони оголошують змінні в точності з тими ж властивостями, як якщо б ті були оголошені безпосередньо без перейменування типу. Фактично

`typedef` аналогічний `#define` з тією лише відмінністю, що при інтерпретації компілятором він може впоратися з деякими більш складними текстовими підстановками, які не можуть бути оброблені препроцесором.

3.3 Контрольні питання

1. Що називається змінною? Які правила оголошення змінних?
2. Наведіть формальну схему оголошення змінних.
3. Що називається константою? Які правила оголошення констант?
4. Наведіть формальну схему оголошення констант.
5. Типи даних. Наведіть приклади оголошення змінних та констант різних типів.
6. Директива `#define`.
7. Припустимі системи числення для цілих чисел.
8. Відмінності між цілими і дійсними константами.
9. Інструкція `typedef`.

3.4 Тести для самоконтролю

1) Що називається змінними у програмі?

А) об'єкти програми, для котрих програміст може встановлювати назву

Б) об'єкти програми, які можуть змінювати своє значення в ході роботи програми

В) об'єкти програми, які можна оголошувати в будь-якому місці програми

Г) об'єкти, без яких програма продовжує коректно працювати

2) Як коректно оголосити змінну?

А) вказати ідентифікатор змінної

Б) вказати им’я типу змінної

В) вказати им’я типу, потім ідентифікатор змінної

Г) вказати ідентифікатор змінної, потім им’я типу

3) Вкажіть правильне оголошення змінних цілого типу

А) `int as, qwerty;`

Б) `int a; b;`

В) `a int as;`

Г) `a,qwerty:int;`

4) Вкажіть невірне оголошення змінних дійсного типу

А) `double a,b;`

Б) `double a; double b;`

В) `float d,f;`

Г) `x,y:real;`

5) Вкажіть правильне оголошення змінних символьного типу

А) `char ‘s’,’g’;`

Б) `s,g:symbol;`

В) `char s; g;`

Г) `char s; char g;`

6) Вкажіть правильне оголошення для константи

А) `const int a=3;`

Б) `const float b;`

В) `int a=3;`

Г) `int a=3;`

7) Директива препроцесора `#define`

А) дозволяє визначати змінні цілих типів

Б) є альтернативою для роботи з константами

В) завжди стоїть на початку програми

Г) слідкує за розподілом пам’яті

8) У якій системі числення не можна виразити цілі числа?

А) у дванадцятирічній

Б) у десятковій

В) у шістнадцятирічній

Г) у восьмирічній

9) Яке значення не може приймати символьна константа

А) `‘&’`

Б) `‘h’`

В) `‘const’`

Г) `‘6’`

10) Інструкція `typedef`

А) створює оголошення нового типу

Б) використовується замість `#include`

В) дозволяє програмісту використовувати у програмі нестандартні типи даних

Г) надає існуючим типам даних нові імена

РОЗДІЛ 4 ВИРАЗИ І ОПЕРАТОРИ

4.1 Вирази та пріоритети операцій

В якості виразів у мові С розглядається сукупність елементів даних (змінних, констант і викликів функцій), об'єднаних знаками операцій і дужками. Кожне вираз повинен мати своє значення. Результат обчислення виразу залежить від розташування знаків операцій і круглих дужок у виразі, а також від пріоритету виконання операцій. Елементи даних, які використовуються у виразі, називаються операндами. Кожний операнд має тип. При обчисленні виразів тип кожного операнда може бути перетворений до іншого типу. Перетворення типів бувають неявними (при обчисленні виразів і викликах функцій) або явними (при виконанні операцій перетворення типів). Про це вже говорилось у розділу 1.

Якщо у якості операнду використовується числова константа, то тип операнда встановлюється залежно від значення константи. Константа, представлена цілим числом, може бути типу int, long, unsigned int або unsigned long залежно від її значення і від форми запису. За замовчуванням компілятор приписує константі тип найменшого розміру, у комірку якого може вміститися константа. Але це правило має вийняток: усім дійсним константам, навіть найменшим, приписується тип double. Більшість компіляторів забезпечує стандартний синтаксис мови С, згідно з яким в символьній константі може бути тільки один друкований символ або один керуючий код. Слідуючи стандартному синтаксису, символьна константа вимагає один байт пам'яті і має тип char. Строковий літерал складається з послідовності символів, укладених в лапки, і представляється в пам'яті як масив елементів типу char, що ініціалізується зазначеною послідовністю символів. Так як рядки є константами, їх не можна використовувати в лівій частині операції присвоювання. У мові С вважається, що вираз має значення true, якщо результат обчислень не дорівнює нулю, і false у противному випадку.

Операції у мові С розрізняються по типу:

1) арифметичні;

2) логічні;

3) бітові;

4) операції відносини;

5) адресна операція та ін.

Обчислення значень виразів виконується в певному порядку. Починається обчислення з визначення значень змінних і констант, що входять у вираз. Вони є основою для подальших обчислень. Подальші дії виконуються відповідно до їх пріоритету. Так, в першу чергу обчислюються вирази, укладені в круглі дужки. Для будь-яких двох вкладених один в одного пар круглих дужок обчислюються спочатку внутрішній вираз, а потім зовнішній. Далі обчислюються значення функцій, що входять у вираз і т.д. Пріоритети всіх дій, які виконуються при обчисленні виразів, наведені нижче (табл. 4.1) [22].

Таблиця 4.1 – Пріоритети операцій у мові С

Пріо-ритет	Опера-тор	Опис	Асоціативність
1	++ -- () [] . ->	префіксний інкремент префіксний декремент виклик функції взяття елемента масиву член структури вибір елемента структури по покажчику	зліва направо
2	! ~ + - * & sizeof	логічне заперечення порозрядне логічне «НІ» (доповнення до 1) унарний плюс унарний мінус (зміна знаку) звернення до пам’яті по значенню покажчика визначення адреси змінної визначення розміру у байтах	справа наліво
3	* / %	множення ділення залишок від ділення	зліва направо
4	+ -	додавання віднімання	зліва направо
5	<< >>	порозрядний зсув вліво порозрядний зсув вправо	зліва направо
6	< <= > >=	менше менше або дорівнює більше більше або дорівнює	зліва направо
7	== !=	дорівнює не дорівнює	зліва направо
8	&	порозрядне логічне «І»	зліва направо
9	^	порозрядне виключне «АБО» (XOR)	зліва направо
10	|	порозрядне логічне «АБО»	зліва направо

Продовження таблиці 4.1

Пріо-ритет	Опера-тор	Опис	Асоціативність
11	&&	логічне «І»	зліва направо
12	||	логічне «АБО»	зліва направо
13	?:	операція умови	зліва направо
14	= += -= *= /= %= <<= >>= &= ^= |=	присвоювання присвоювання з додаванням присвоювання з відніманням присвоювання з множенням присвоювання з діленням присвоювання з визначенням залишка від ділення присвоювання з порозрядним зсувом вліво присвоювання з порозрядним зсувом вправо присвоювання з порозрядним логічним «І» присвоювання з порозрядним виключним «АБО» присвоювання з порозрядним логічним «АБО»	зліва направо
15	, ++ --	операція «кома» постфіксний інкремент постфіксний декремент	зліва направо

4.2 Загальна класифікація операторів

Оператори мови описують деякі алгоритмічні дії, які необхідно виконати для вирішення задачі [23]. Тіло програми можна представити як послідовність таких операторів. У мові С крапка з комою є роздільником і одночасно приналежністю оператора.

Всі оператори можна розбити на дві групи: прості та структуровані.

До простих належать ті оператори, які не містять в собі інших операторів. До них відносяться:

1) оператор присвоювання;

2) оператор безумовного переходу `goto`;

3) пустий оператор.

Оператор безумовного переходу `goto` дозволяє змінити стандартний послідовний порядок виконання операторів і перейти до виконання програми, починаючи з заданого оператора. Оператор, на який відбувається перехід, повинен

бути позначений міткою. Ця ж мітка повинна бути вказана і в операторі `goto`. Ім'я для мітки обирається як для звичайного ідентифікатора.

Пустий оператор не виконує ніякої дії і ніяк не відображається в програмі (за винятком мітки, якій він може бути позначений, або крапок з комами, що відокремлюють пустий оператор від попередніх або наступних операторів). Пустий оператор може знадобитися, наприклад, для здійснення на нього безумовного переходу. Пустий оператор використовується, коли ніякої дії виконувати не потрібно, однак по синтаксису мови потрібно використання якого-небудь оператора.

Структурованими є такі оператори, які складаються з інших операторів. До них відносяться:

1) складений оператор;

2) умовні оператори (включаючи оператор вибору);

3) оператори циклу та ін.

Більш детально усі ці оператори будуть розглядатись далі.

4.3 Складання лінійних програм

Згадаємо програму мовою С з розділу 2, що дозволяє обчислювати значення функції і виводити його на екран.

```
/* Перша програма на С */
#include <stdio.h>

void main()
{
     int a;
     float b, c;
     printf("Input the variables a, b\n");
     scanf("%d%f", &a, &b);
     c=a*b+2.5;
     printf("The result is %f", c);
}
```

Незважаючи на простоту програми, за її допомогою можна проілюструвати деякі важливі особливості мови С.

Перший рядок починається символами /* і закінчується символами */, які означають, що ці рядки є коментарем, про них ви дізналися у главі 2.

Наступний рядок містить директиву препроцесора. Рядки, що починаються зі знака #, обробляються препроцесором до компіляції програми. Дана директива вимагає включити в текст програми зміст файлу заголовка стандартного введення/виведення. Цей заголовний файл містить інформацію й оголошення, необхідні компілятору для обробки викликів стандартних функцій введення/виведення. Інформація файлу допомагає компілятору перевірити коректність викликів бібліотечних функцій.

Рядок з викликом функції main повинен обов'язково бути присутнім у кожній програмі. Дужки після main означають, що main є «будівельним блоком» програми, називаним функцією. Програма на С може містити одну чи більшу кількість функцій, однак одна з функцій обов'язково повинна бути main. З неї починається виконання програми.

Ліва фігурна дужка повинна випереджати тіло кожної функції. Відповідно права фігурна дужка повинна стояти наприкінці кожної функції. Ця пара дужок і частина програми між ними називається блоком. Блок – важлива програмна одиниця в С.

Наступні два рядки містять оголошення змінних, які будуть використовуватися в нашій програмі. Спочатку вказується тип змінної, потім через пропуск її ім'я. Якщо потрібно оголосити декілька змінних одного типу, їх перелічують через кому.

Наприкінці оголошень змінних одного типу ставлять крапку з комою.

Наступний рядок дозволяє програмі видати на екран повідомлення, яке пропонує користувачу ввести значення для змінних, що є вихідними даними. Далі користувач уводить ці значення.

Обчислення значення для змінної с здійснюється за допомогою оператора присвоювання. Це дозволяє значення виразу, що знаходиться в правій частині оператора присвоювання, привласнити змінній, ім'я якої зазначено в лівій частині оператора присвоювання. Отримане значення за допомогою оператора виводу роздруковується на екрані.

Ще один приклад програми:

```
#include <stdio.h>
void main() {
     char ch;
```

```
    int i, j;
    float x, x2;
    char name[81];
    printf("\nВведіть Ваше ім’я: ");
    scanf("%s", name);
    printf("Шановний %s. Вас привітає дзвінком комп’ютер
\a\n",name);
    printf("Введіть деяке ціле число: ");
    fflush(stdin);
    scanf("%d",&i);
    j=i+4;
    printf("%d+4=%d, вірно?\n",i,j);
    printf("Введіть деяке дійсне число з плаваючою точкою: ");
    scanf("%f",&x);
    x2=x*x;
    printf("%f у квадраті = %f, вірно?\n",x,x2);
    printf("Введіть деякий символ: ");
    fflush(stdin);
    scanf("%c",&ch);
    printf("ASCII-код символа %c = %d (%#x)\n",ch,ch,ch);
}
```

Результат роботи програми:

Введіть Ваше ім’я: Ігор Петренко

Шановний Ігор. Вас привітає дзвінком комп’ютер

Введіть деяке ціле число: 36

36+4=40, вірно?

Введіть деяке дійсне число з плаваючою точкою: 5.e1

50.000000 у квадраті = 2500.000000, вірно?

Введіть деякий символ: 1

ASCII-код символа 1 = 49(0x31)

При складанні лінійних програм необхідно використовувати прості оператори відповідної мови програмування, що дозволяють реалізувати ряд послідовних кроків.

Слід виділити оператор присвоювання.

За допомогою цього оператора змінної чи функції привласнюється значення виразу.

Для цієї операції використовується знак присвоювання =. Ліворуч від знака присвоювання вказується ім'я змінної (типізованої константи) чи функції, а праворуч – вираз, значення якого обчислюється перед присвоюванням. Типи лівої і правої частин оператора присвоювання повинні бути сумісні.

Приклад: подвоїти введене користувачем значення.

```
int i;
scanf(“%d”, &i);
i=i*2;
printf(“%d”, i);
```

Увести 2 цілих значення. Обчислити 3 дійсні значення за формулами.

```
int i,j;
float a,b,c;
scanf(“%f%f”, &a, &b);
a=5.4;
b=a+1;
c=a/b+2*a/(3.3+b);
printf(“a=%f\tb=%f\tc=%f”, a, b, c);
```

Оператор присвоювання С має деякі відмінності. По-перше, можливо множинне присвоювання значень змінних, наприклад:

```
int a,b,c;
a=b=0;
c=5;
float d=c/3.2;
```

По-друге, можна використовувати склеєні операції присвоювання, тобто

```
a+=b --> a=a+b;
a*=b --> a=a*b;
a-=b --> a=a-b;
a/=b --> a=a/b;
```

Для збільшення і зменшення значення змінної на 1 можна використовувати: `a++, a-- чи ++a, --a (a=a+1, a=a-1)`. Розходження в префіксній (`++a`) і постфіксній (`a++`) операціях полягає в тому, що в префіксній операції значення змінної змінюється (збільшується чи зменшується на 1), а потім ця змінна бере участь в обчисленні виразу, а в постфіксній спочатку обчислюється значення

виразу з поточним значенням змінної, а потім це значення змінюється. (див. пріоритети операцій, глава 3).

Приклад.

```
#include <stdio.h>

void main()
{
     int i, j;
     double d;
     i = 10; //оператори присвоювання
     j = 20;
     d = 99.101;
     printf("Here are some numbers: ");
     printf("%d", i);
     printf(" ");
     printf("%d", j);
     printf(" ");
     printf("%lf", d);
}
```

Приклад.

```
 /*Дана програма визначає парність цілого*/
#include <stdio.h>

void main()
{
     int num;
     // читання числа
     printf("Уведіть число, що перевіряється:");
     scanf("%d", &num);
     // перевірка на парність
     if((num%2)==0) printf("Число парне\n");
     else printf("Число непарне\n");
}
```

Приклад.

```
#include <stdio.h>

void main()
{
     double hours, wage;
     printf("Уведіть кількість опрацьованих годин: ");
     scanf("%lf", &hours);
     printf("Уведіть погодинну оплату: ");
     scanf("%lf", &wage);
     printf("Зарплата дорівнює: $%lf", wage*hours);
}
```

4.4 Деякі математичні функції

Часто в процесі обчислень необхідно використовувати математичні функції. Якщо в програмі потрібно використовувати хоча б одну з них, то потрібно визначити звертання до них. Для цього в програму потрібно включити файл прототипів функцій `<math.h>` директивою препроцесора `#include`:

```
#include <math.h>
```

Таблиця 4.2 – Найбільш розповсюджені математичні функції С

Найменування	Тип аргументу	Написання функції на мові С	Тип результату
|n|	int	abs(n)	int
|x|	double	fabs(n)	double
√x	double	sqrt(x)	double
e^x	double	exp(x)	double
ln x	double	log(x)	double
lg x	double	log10(x)	double
sin x	double	sin(x)	double
cos x	double	cos(x)	double
tg x	double	tan(x)	double
x^y	double, double	pow(x,y)	double

Приклад. Обчислити площину чотирикутника ADFC, який задано згідно з рисунком. Довжини сторін AB, BC, AC, DB, BF та величина куту α = DBF є відомими [24].

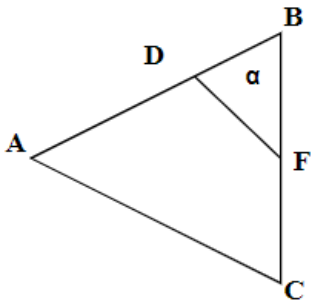


Рисунок 4.1 – Приклад чоторикутника

Для обчислення площі чотирикутника ADFC необхідно спочатку обчислити площу трикутника ABC по трьом сторонам (S1), потім площу трикутника DBC по двом сторонам і куту між ними (S2), і знайти різницю між отриманими значеннями (S). Поруч наведено відповідну схему алгоритму.

```
#include <stdio.h>
#include <math.h>

void main() {
	float AB, BC, AC, DB, BF, DBF, p, S1, S2, S;
	printf("Введіть довжини сторон та кут: ");
	scanf("%f%f%f%f%f%f", AB, BC, AC, DB, BF, DBF);
	p=(AB+BC+AC)/2;
	S1=sqrt(p*(p-AB)*(p-BC)*(p-AC));
	S2=0.5*DB*BF*sin(DBF);
	S=S1-S2;
	printf("\nПлоща чотирикутника ADFC дорівнює %f", S);
}
```

4.5 Контрольні питання

1. Що називається виразом? Як обчислюються вирази?
2. Які існують типи операцій у мові С?
3. Наведіть пріоритети дій, які виконуються при обчисленні виразів.
4. Які існують основні групи операторів?
5. Які оператори належать до простих?
6. Як працює оператор безумовного переходу?
7. Для чого використовується пустий оператор?

8. Які оператори належать до структурованих?

9. Що називається блоком?

10. Для чого використовується оператор присвоювання? Що слід знати про типи лівої і правої частин оператора присвоювання?

11. Як використовувати склеєні операції присвоювання?

12. Чим префіксна операція відрізняється від постфіксної?

13. Для чого використовується пустий оператор?

14. Як у програмі використовувати математичні функції? Що буде, якщо не підключити заголовний файл <math.h>?

15. Перелічить відомі вам математичні функції. Для кожної функції вкажіть: найменування, тип аргументу, написання функції на мові С, тип результату.

16. Як можна задавати аргументи для функції?

17. Як можна використовувати результат, що повертає функція?

4.6 Тести для самоконтролю

1) Що у мові С є виразом у загальному вигляді?

А) перелік змінних

Б) знаки операцій

В) перелік припустимих символів, укладений в дужки

Г) сукупність елементів даних, об'єднаних знаками операцій і дужками

2) Якого з типів операцій немає у мові С?

А) арифметичних

Б) тригонометричних

В) бітових

Г) логічних

3) Який знак операції має найвищий пріоритет?

А) `!`

Б) `<<`

В) `%`

Г) `?:`

4) Який знак операції має найнижчий пріоритет?

А) `||`

Б) `>=`

В) `&&`

Г) `&=`

5) Як записується оператор присвоювання у мові С?

А) `==`

Б) `=`

В) `:=`

Г) `::=`

6) Запишіть повну версію виразу `x / = y` ;

А) `x = x / y` ;

Б) `y = x / y` ;

В) `x = x / x` ;

Г) `x = y / y` ;

7) Як виконується вираз i++ ?

А) `i = i + i`

Б) `i = 1 + 1`

В) `i = i + 1`

Г) немає вірної відповіді

8) Чому дорівнює значення змінної b після наступного: `a=3; b=++a;`

А) 0

Б) 3

В) 4

Г) це потенційна помилка, компілятор видасть попередження

9) Який оператор не належить до простих операторів?

А) умовний

Б) присвоювання

В) пустий

Г) безумовного переходу

10) Який заголовний файл треба підключити для можливості використання математичних функцій?

А) `<mathematics.h>`

Б) `<math.h>`

В) `<calc.h>`

Г) `<mat_func>`

11) Чим відрізняються між собою функції `abs` та `fabs`?

А) кількістю аргументів

Б) вони виконують різні дії

В) вони належать до різних математичних бібліотек

Г) типом аргументів

12) Як за допомогою математичної функції обчислити 2^8?

А) `sqrt`(2,8)

Б) `pow`(2,8)

В) pow(8,2)

Г) `sqr`(2,8)

РОЗДІЛ 5 УМОВНІ ОПЕРАТОРИ

5.1 Програмування розгалужень. Складені оператори

Для організації вибору в програмі з групи альтернатив можливого продовження обчислювального процесу використовуються оператори розгалуження. При цьому вибір виконується виходячи зі значення заданого виразу.

Для організації вибору одного з двох можливих варіантів продовження програми використовується умовний оператор.

Наприклад, необхідно проаналізувати температуру тіла пацієнта. Якщо температура перевищує 37°C, програма повинна видати повідомлення про те, що людина хвора. Інакше програма повинна сказати, що людина здорова.

Далі наведено формальну схему, яка дає опис умовного оператора.

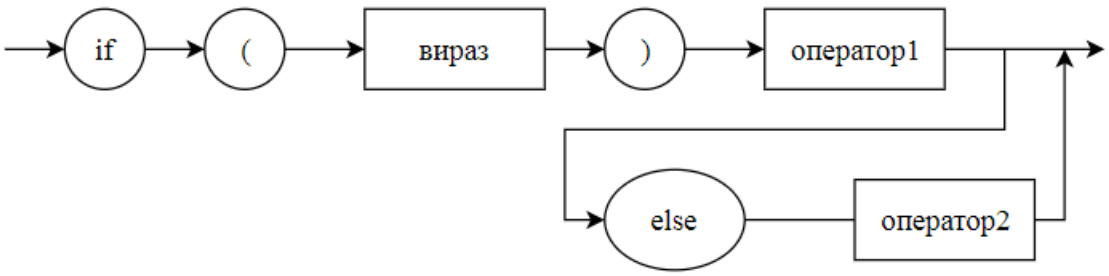


Рисунок 5.1 – Формальне представлення опису умовного оператора

Обчислюється значення виразу, наведеного після слова `if`. Якщо воно істинне – `TRUE` (будь-яке відмінне від 0 значення), то виконується оператор1. Якщо значення помилкове, виконується оператор2.

```
if (temper>37.0) printf(”\nЛюдина хвора”);
else printf(”\nЛюдина здорова”);
```

Якщо `оператор1` чи `оператор2` являють собою деякий набір операторів, їх необхідно представити як складений оператор. Складений оператор, чи блок – це будь-яка сукупність простих операторів, укладена у фігурні дужки {}. Складений оператор ідентичний простому оператору і може знаходитися в будь-якім місці програми, де синтаксис мови допускає наявність тільки одного оператора, а фактично їх потрібно використовувати декілька. Складені оператори – важливий

інструмент програмування, що дає можливість писати програми за сучасною технологією структурного програмування (без операторів безумовного переходу).

Мова С не накладає ніяких обмежень на характер операторів, що входять у складений оператор. Серед них можуть бути й інші складені оператори – у програмі допускається довільна глибина їхньої вкладеності. Наприклад:

```
{
     ..........
     {
          ..........
          {
               ............
               ............
          }
          ............
     }
     ............
}
```

Фактично весь розділ операторів, обрамлений дужками `{ ... }`, являє собою один складений оператор. Порожній оператор не містить ніякі дії, просто в програму додається зайва крапка з комою. В основному порожній оператор використовується для передачі керування в кінець складеного оператора.

Частина `else` у конструкції умовного оператора може опускатися. У цьому випадку, якщо вираз має значення `FALSE`, ніякі дії не виконуються, керування в програмі передається оператору, що йде за умовним оператором.

Умовні оператори можуть бути вкладеними, ступінь їхньої вкладеності не обмежений. Компілятор інтерпретує вкладені `if`, зіставляючи кожне з ключових слів `else` з останнім словом `if`, що зустрілося та не має "свого" `else`. Відповідність шукається в межах блоку, у який укладене слово `if`. Внутрішні і зовнішні блоки при цьому не розглядаються. Якщо відповідності для `if` не знайдено, компілятор полагає, що `if` не має частини `else`.

Наприклад:

1)
```
     if a=0
     c=5;
```
2)
```
     if (q<0 && w % 3==0) w=2*w+1;
     else { y=sin(w); w=3*w+2; }
```
3)
```
     if (x<0) a=1;
     else if (x>0) a=2;
```

```
else a=3;
```

Те ж саме можна представити таким чином:

```
if (x<0) a=1;
if (x>0) a=2;
if (x==0) a=3;
```

Приклад. Програма для рішення квадратного рівняння.

```
#include <stdio.h>
#include <math.h>

void main() {
float x1, x2, D;
int a, b, c;
printf("Input parametres (a, b, c) for equation a*x^2 + bx + c =
0\n");
scanf("%d%d%d", &a, &b, &c);

D=b*b-4*a*c;
if (D<0) printf("No real roots\n");// якщо дискримінант
// менше нуля, коренів немає
else if(D>0) {  // якщо дискримінант більше нуля, 2 кореня
     x1=(-b+sqrt(d))/(2*a);
     x2=(-b-sqrt(d))/(2*a);
     printf("Equation has 2 roots:\nx1 = %f\nx2 = %f\n", x1,
x2);
}
else {// якщо дискримінант дорівнює нулю, 1 корінь  x1 =
(-b/2*a);
     printf("Equation has 1 root:\nx = %f\n", x1);
}
}
```

5.2 Оператор ?

Для заміни стандартної конструкції `if/else` може використовуватися оператор `?`. Він має такий вид: `Вираз1?Вираз2:Вираз3`;

Обчислюється `Вираз1`. Якщо він істинний, обчислюється `Вираз2` і вся конструкція одержує обчислений вираз. Якщо `Вираз1` помилковий, обчислюється `Вираз3` і вся конструкція одержує обчислений вираз. Наприклад:

```
Y=a>0?10:15; ? if (a>0) y=10; else y=15;
```

5.3 Оператори варіантів

Інший варіант організації множинного вибору – використання оператора `switch`. При цьому керування в програмі передається оператору, у котрого як мітка використовується значення деякого виразу. Як вираз, так і мітки повинні мати значення цілого типу (допускається також тип `char`); крім того, мітки повинні бути константами чи константними виразами. Якщо деякому значенню вираза не відповідає ніяка мітка, керування передається оператору з міткою `default`. У випадку його відсутності керування передається оператору, що йде за оператором `switch`. Оператори можуть бути простими чи складеними, причому в даному випадку їх не обов'язково брати у фігурні дужки.

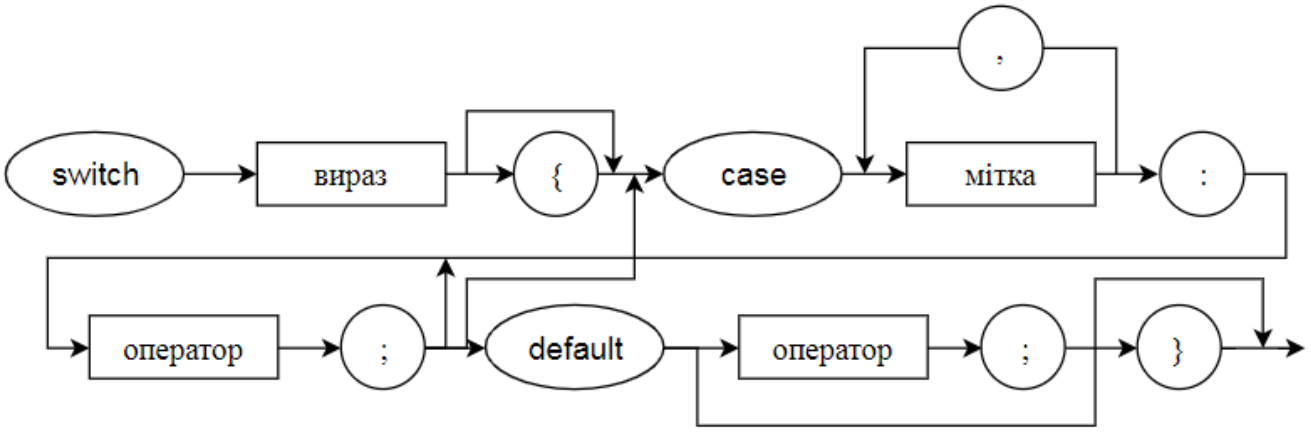


Рисунок 5.3 – Формальне представлення оператора варіантів

Приклад.

```
switch(i)
{
     case 1 : a=1;
               break;
     case 2 :  a=b=2;
               break;
     default: c=3;
}
```

Приклад.

```
switch (ch) {
     case ‘Y’:
          printf(”продовжуємо”);
          break;
     case ‘N’:
```

```
        printf(”закінчуємо”);
        exit();
        break;
}
```

Програма в залежності від натиснутої клавіші повинна вивести на екран відповідний текст, що і буде зроблено. Зерніть увагу, що після операторів кожного `case` необхідно писати ключове слово `break`. Інакше, після виконання операторів потрібного `case` програма продовжить виконувати наступні оператори, які виконувати вже непотрібно. Докладніше про оператор `break` ви дізнаєтесь, коли будете вивчати цикли.

5.4 Логічні операції

Мова С має рівно три логічні операції:

```
&& або  (AND);
|| або (OR);
! або (NOT).
```

Операція «&&» або операція «AND» називається ще операцією «і» чи логічним множенням.

Операція «||» або операція «OR» називається ще операцією «або» або логічним складанням.

Операція «!» або операція «NOT» називається ще операцією «ні» або логічним запереченням.

Розберемося, чому вони так називаються? Будемо вважати, що істина це «1», а брехня – це «0».

Операція «&&» називається логічним множенням тому, що виконується таблиця істинності цієї операції, дуже нагадує таблицю звичайного множення з арифметики.

Логічне множення це така операція, яка істинна тоді і тільки тоді, коли істинні обидва входять до неї висловлювання.

```
1 && 1 = 1
0 && 1 = 0
1 && 0 = 0
0 && 0 = 0
```

Операція "||" називається логічним складанням тому, що виконується таблиця істинності цієї операції, що дуже нагадує таблицю звичайного складання

з арифметики.

Логічне додавання це така операція, яка істинна тоді і тільки тоді, коли істинно хоча б одне з вхідних в неї висловлювань.

```
1 || 1 = 1
0 || 1 = 1
1 || 0 = 1
0 || 0 = 0
```

Операція "!" називається логічним запереченням тому, що виконується наступна таблиця істинності.

Логічне заперечення це така операція, яка істинна тоді і тільки тоді, коли хибне висловлювання, що входиить у неї, і навпаки.

```
!1 = 0
!0 = 1
```

Приклад.

```
if(a>10 && b<5) {    //якщо а більше 10 И b менше 5
     ……
}

if(a>10 || a<5) {    //якщо а більше 10 АБО а менше 5
     ……
}

if(!c) {             //якщо  НЕ с (тобто с – не істина)
     ……
}
```

5.5 Контрольні питання

1. Який оператор використовується для організації у програмі вибору з групи альтернатив?
2. Наведіть формальну схему опису умовного оператора.
3. Чому може дорівнювати значення умовного виразу у круглих дужках?
4. Наведіть фрагмент схеми алгоритму для умовного оператора.
5. Що робити, якщо `оператор1` чи `оператор2` являють собою деякий набір операторів?

6. Що таке складений оператор? Де він може знаходитись у програмі?

7. Які оператори можуть включатись у складений оператор?

8. Як виконується умовний оператор, який не має альтернативної гілки? Наведіть приклад. Як це виглядає на схемі алгоритму?

9. Наведіть приклади включення в умовний оператор: іншого умовного оператора у якості оператора1, іншого умовного оператора у якості оператора2, двох умовних операторів – і у якості оператора1, і у якості оператора2.

10. Для чого використовується оператор «?» ?

11. Для чого використовується оператор «switch»?

12. Що таке мітка? Якого вона може бути типу?

13. Чи може оператор «switch» виконувати складені оператори? Що для цього треба зробити?

14. Для чого використовується мітка default? Чи є вона обов'язковою? Як працюватиме оператор «switch» без мітка default, якщо жодна з альтернатив не підійшла?

15. Які логічні операції має мова С? Як вони позначуються?

16. Наведіть таблиці істинності для логічного множення, логічного додавання та логічного заперечення.

17. Наведіть приклади використання логічних операцій.

5.6 Тести для самоконтролю

1) Як називається оператор, який використовується для організації вибору в програмі одного з двох можливих варіантів продовження програми?

А) оператор вибору

Б) оператор бінарного вибору

В) безумовний оператор

Г) умовний оператор

2) Яким є тип виразу, наведеного після if ?

А) цілий

Б) логічний

В) дійсний

Г) структурний

3) Як ще називається умовний оператор?

А) оператор розгалуження

Б) оператор роздрібнення

В) оператор дії

Г) немає правильної відповіді

4) Вкажіть вид скороченого оператора `if`:

А) `if () {…} else {…}`

Б) `if [] {…}`

В) `if () then {…}`

Г) `if () {…}`

5) Вкажіть вид повного оператора `if`:

А) `if () then {…}`

Б) `if [] {…}`

В) `if () {…} else {…}`

Г) `if () {…}`

6) Для чого використовується оператор «?» ?

А) для заміни повної стандартної конструкції `if/else`

Б) для заміни скороченого оператора `if`

В) для зміни типу константи

Г) немає правильної відповіді

7) Що відбувається, якщо при виконанні оператора `Вираз1?Вираз2:Вираз3;` істинним виявляється `Вираз1`?

А) обчислюється `Вираз2` і результат виводиться на екран

Б) обчислюється `Вираз2` і вся конструкція одержує обчислений вираз

В) обчислюється `Вираз3` і результат виводиться на екран

Г) обчислюється `Вираз3` і вся конструкція одержує обчислений вираз

8) Який оператор використовується для організації множинного вибору?

А) `switch`

Б) `select`

В) `default`

Г) `choice`

9) Що буде, якщо деякому значенню вираза у `switch` не відповідає ніяка мітка і оператор `default` відсутній?

А) компілятор повідомить про помилку

Б) будуть виконані оператори, яким відповідає остання мітка

В) керування передається оператору, що йде за оператором `switch`

Г) немає правильної відповіді

10) Що з наведеного не є логічною операцією в С?

А) `&&`

Б) `||`

В) `!`

Г) `~~`

11) Яке значення приймає наступна складна умова, якщо `a=12, b=13`?

```
(a>10 || b<12)
```

А) `true`

Б) `false`

В) значення не визначено

Г) компілятор повідомить про помилку

12) Яке значення приймає наступна складна умова, якщо `a=12, b=13`?

```
(a>10 && b<12)
```

А) `true`

Б) `false`

В) немає правильної відповіді

Г) значення не визначено

РОЗДІЛ 6 ЦИКЛИ

6.1 Загальні положення

Цикл – це група команд, багаторазово виконуваних комп'ютером, поки деяка умова продовження циклу залишається істинною. Існує два способи повторення:

1) повторення, що керується лічильником;

2) повторення, що керується контрольним значенням.

Повторення, що керується лічильником, іноді називають визначеним повторенням, тому що ми точно знаємо, скільки разів буде виконаний цикл. Повторення, що керується контрольним значенням, іноді називають невизначеним повторенням, тому що заздалегідь невідомо, скільки разів буде потрібно виконати цикл.

У мові С є три різних оператора, за допомогою яких можна запрограмувати повторювані фрагменти програми. Це оператор циклу з передумовою, з постумовою і цикл із параметром. Кожний з цих циклів має свої особливості застосування.

6.2 Цикл з передумовою

Оператор циклу з передумовою використовується в тому випадку, коли число повторень оператора невідомо.

Обчислення виразу булевого типу здійснюється до того, як внутрішній оператор буде виконаний. Внутрішній оператор виконується повторно доки вираз приймає значення true. Якщо вираз із самого початку приймає значення false, то оператор, що міститься усередині оператора циклу з передумовою, не виконується. Якщо вираз не може прийняти значення false, то виходить нескінченний цикл. Фрагмент схеми алгоритму для циклу з передумовою наведено поруч.

Для мови С:

```
while (<умова>) {<оператор>}
```

Загальний вигляд оператор циклу з передумовою відображено на рис. 6.1.

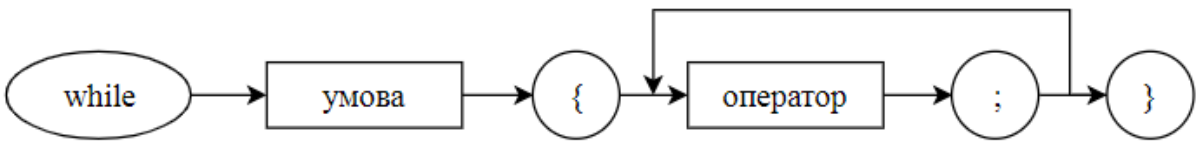


Рисунок 6.1 – Формальне представлення оператора циклу з передумовою

Приклад. Обчислити машинне епсілон.

```
#include <stdio.h>

void main()
{
     float epsilon=1;
     while (epsilon/2+1>1)
     epsilon/=2;
     printf("\nМашинне епсілон = %f", epsilon);
}
```

Приклад. Обчисліть суму членів спадного ряду.

```
#include <stdio.h>
#include <math.h>

void main()
{
     float s, sn, x, e, ch;
     int j, fact, znak;
     printf("Input x,e: ");
     scanf("%f%f", &x, &e);
     s=x/2;
     j=2;
     fact=1;
     ch=x*pow(x/2,2);
     znak=-1;
     sn=ch/(2*fact*fact*j)*znak;
     while (fabs(sn)>e)
     {
          s+=sn;
          ch*=sqrt(x/2);
          fact*=j;
          j++;
          znak=-znak;
```

```
            sn=ch/(2*fact*fact*j)*znak;
        }
        printf("Result S= %f", s);
}
```

6.3 Цикли з постумовами. Створення таблиць. Формати даних

Оператор циклу з постумовою звичайно використовується в тому випадку, коли число повторень оператора невідомо.

У мові С виконується оператор чи послідовність операторів, що йдуть за ключовим словом `do`. Обчислюється значення логічного виразу, розташованого за ключовим словом `while`. Якщо значення вірно, то повторюється виконання оператора (послідовності операторів), які йдуть за ключовим словом `do`. Якщо значення виразу невірно, то виконання зазначених операторів припиняється. Послідовність операторів виконується принаймні один раз, оскільки обчислення виразу здійснюється після кожного виконання послідовності операторів. Фрагмент схеми алгоритму для циклу з постумовою наведено поруч.

```
do{
<послідовність операторів>
} while (<умова>);
```

Формально оператор циклу з постумовою відображено на наступній схемі:

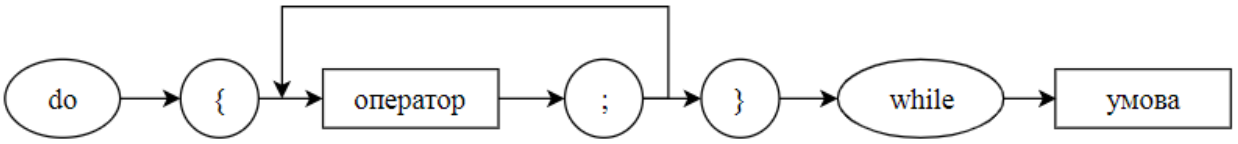


Рисунок 6.2 – Формальне представлення оператора циклу з постумовою

Наприклад: Підсумовувати числа, що вводяться з клавіатури, до введення першого нуля.

```
sum=0;
do
{
scanf (“%d”, &a);
sum+=a;
```

```
}
while (a!=0);
```

При наявності тільки одного оператора в тілі циклу в структурі `do/while` немає необхідності використовувати фігурні дужки. Однак фігурні дужки включаються, щоб уникнути плутанини між структурами `do/while` і `while`.

Наприклад:

`while (умова)` звичайно розглядається як заголовок структури `while`. Структура `do/while` без фігурних дужок навколо її тіла, що складається з одного оператора, виглядає як

```
do
оператор
while (умова);
```

що може привести до плутанини. Останній рядок – `while (умова);` може бути неправильно витлумачена читачем як структура `while`, що містить порожній оператор.

Тому структура `do/while` з одним оператором часто записується так:

```
do
{
оператор
}
while (умова);
```

Якщо умова продовження циклу ніколи не стає помилковою, виникають нескінченні цикли. Щоб уникнути цього, переконайтеся, що відразу після заголовка структури `for` чи `while` немає крапки з комою. У циклі, керованому контрольним значенням, переконайтеся в тому, що воно рано чи пізно вводиться.

Наступна програма використовується для виводу чисел від 1 до 10. Зверніть увагу, що до керуючої змінної `counter` при перевірці умови продовження циклу застосовується операція преінкремента. Зверніть також увагу на фігурні дужки, що укладають тіло структури `do/while` (яке складається з одного оператора).

```
#include <stdio.h>

void main()
{
     int counter=1;
```

```
     do
     {
          printf("%d", counter);
     }
     while (++counter<=10);
}
```

Результат роботи: 1 2 3 4 5 6 7 8 9 10.

Розглянемо ще декілька прикладів застосування циклів `while` та `do/while`.

Приклад. Побудувати таблицю значень функції $y = \cos\frac{(x-a)^2}{x-2a} - \frac{3.5}{\sqrt{xb}}$ (x змінюється від `xmin` до `xmax` із кроком `dx`) використовуючи оператори циклу з передумовою та постумовою. Проконтролювати правильність введення `xmin, xmax, dx` і коректність виразу, що обчислюється.

```
#include <math.h>
#include <conio.h>
#include <stdio.h>

int main()
{
float a, b, d, x, y, xmin, xmax, dx;
printf("Input a: ");
scanf("%f",&a);

do
{
     printf("Input b (not equal 0): ");
     scanf("%f",&b);
} while (b==0);

do
{
     printf("Input xmin<xmax: ");
     scanf("%f%f",&xmin,&xmax);
} while (xmin>=xmax);
```

```
d=xmax-xmin;

do
{
     printf("Input 0<dx<");
     printf("%6.2f: ",d);
     scanf("%f",&dx);
} while(dx<=0 || dx>d);

x=xmin;
printf("X\t\tY\n");
while (x<=xmax)
{
     printf("%f\t",x);
     if (x==2*a || x*b<=0)
printf("Function cannot be calculated!\n");
     else
          {
          y=cos(pow(x-a,2)/(x-2*a))-3.5/sqrt(x*b);
          printf("%f\n",y);
         }
     x=x+dx;
}
getch();
return 0;
}
```

6.4 Оператори break та continue

Для гнучкого керування циклічними операторами до складу мов включені дві процедури:

`break` – реалізує негайний вихід з циклу; дія процедури полягає в передачі керування оператору, що стоїть відразу за кінцем циклічного оператора;

`continue` – забезпечує дострокове завершення чергового проходу циклу; еквівалент передачі керування оператору, що стоїть відразу за кінцем циклічного оператора.

Приклад. Підрахувати кількість чисел, сума яких не перевищує 100. При введенні першого негативного числа припинити обчислення. Оформити рішення у вигляді таблиці.

```
#include <stdio.h>

void main()
{
     int sum, kol, a;
     sum=kol=0;
     printf("a\tsum\n"); //вивод заголовку таблиці
     do
                                {
          scanf("%d", &a);
          if (a<0) break;
          kol++;
          sum+=a;
          printf("%d\t%d\n", a, sum); // формування таблиці
     }
     while (sum<100);

if (a<0) printf("%d", kol);
else printf("%d", kol--);      //останне додавання було
//надлишковим та треба його компенсувати
}
```

Приклад. Виділіть цифри з десятичного числа. Схема алгоритму приведена поруч.

```
#include <stdio.h>

void main() {
     int a, b;
     printf("\nInput decimal number: ");
```

```
      scanf("%d", &a);
      do {
            b=a%10;
            printf("%d", b);
            a=(a-b)/10;
      }
      while (a>9);
      printf("%d", a);
}
```

6.5 Цикли з параметром

Оператор циклу з параметром for використовується, якщо число повторень заздалегідь відомо. Для підрахунку числа повторень використовується керуюча змінна циклу.

Формально оператор циклу з параметром відображен на наступній схемі:

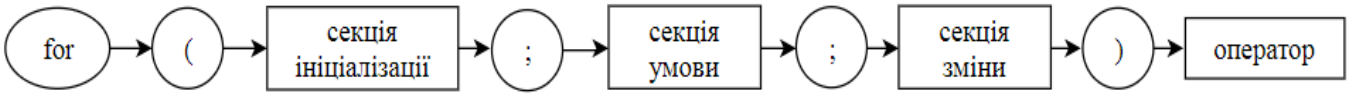


Рисунок 6.3 – Формальне представлення оператора циклу з параметром

У мові С після ключового слова `for` у круглих дужках указується три вирази. Перший містить початкову ініціалізацію змінної циклу. Другий вираз являє собою умову, виконання якої забезпечує входження в цикл. Коли умова стає помилковою, керування передається наступному за циклом оператору. Третій вираз забезпечує величину і характер зміни значення змінної циклу на кожній ітерації. Відповідний фрагмент схеми алгоритму для циклу з параметром приведений поруч.

Приклад. Підсумовування з `for`

```
#include <stdio.h>

void main()    {
      int sum=0, number;
      for (number=2; number<=100; number+=2)
```

```
          sum+=number;
     printf("Summ is %d", sum);
}
```

Приклад. Обчислення складних відсотків.

```
#include <stdio.h>
#include <math.h>

void main()
{
     int year;
     double amount, principal=1000.0, rate=0.05;
     printf("Year amount of deposite:\n");
     for (year=1; year<=10; year++)
     {
          amount = principal*pow(1.0+rate,year);
          printf("%lf\t%lf\n", year, amount);
     }
}
```

Приклад. Обчислити значення інтеграла функції `y=x*sin(x)` на інтервалі `xmin..xmax`

```
#include <stdio.h>
#include <math.h>

void main()
{
     double x, xmin, xmax, dx, y, n, s=0;
     int i;
     printf("Input xmin: ");
     scanf("%lf", &xmin);
     printf("Input xmax: ");
     scanf("%lf" , xmax);
     cout<<"Input dx: ";
     scanf("%lf" , dx);
     n=(xmax-xmin)/dx;
```

```
      x=xmin;
      for (i=0;i<n;i++)
      {
            y=x*sin(x);
            s=s+fabs(y)*dx;
            printf("\n%lf", s);
            x=x+dx;
      }
      printf("\nIntegral = %lf", s);
}
```

Кожний з виразів у заголовку циклу for може складатися з декількох частин, наприклад:

```
for (i=0, j=0; i<100, j!=25; i+=2, j++)
a = i*j;
```

З іншого боку, кожен з виразів чи навіть усі вирази в заголовку циклу можуть бути відсутніми. У випадку відсутності першого виразу це веде до відсутності ініціалізації в циклі, отже, вона повинна бути виконана до нього. Якщо не зазначений другий вираз, то немає умови виходу з циклу, отже, вихід потрібно організувати іншим чином, наприклад, за допомогою break чи операторів переходу. При відсутності третього виразу необхідно в явному виді в тілі циклу керувати зміною змінних циклу.

Розглянемо реалізацію завдання, наведеного в прикладі 3, з використанням циклів з передумовою та з параметром.

$$y = \frac{\sin(x+2a)^3}{e^{\frac{x}{b}} - \frac{x^2}{b-a}}$$

Приклад. Побудувати таблицю значень функції (`x` змінюється від `xmin` до `xmax` із кроком `dx`), використовуючи два види операторів циклу. Проконтролювати правильність введення `xmin, xmax, dx` і коректність вираження, що обчислюється.

```
#include <math.h>
#include <conio.h>
#include <stdio.h>
```

```
int main()
{
     bool correct=false;
     float a, b, y, x, xmin, xmax, dx;
printf("Enter value for variable a: ");
scanf("%f",&a);

while (!correct)
{
     printf("Enter value for variable b: ");
     scanf("%f",&b);
     if (b==0)
          printf("b cannot be a zero!");
     else if (b==a)
          printf("b cannot be as a!");
            else correct=true;
}

correct=false;
printf("Enter value for start x: ");
scanf("%f",&xmin);
while(!correct)
{
printf("Enter value for end x: ");
scanf("%f",&xmax);
if (xmax<=xmin)
     printf("End x cannot be smaller or equal to start x");
else correct=true;
}

correct=false;
while (!correct)
{
     printf("Enter value for step dx: ");
scanf("%f",&dx);
if (xmin+dx>=xmax)
```

```
     printf("Step too big!");
else correct=true;
     }

for (x=xmin; x<xmax; x+=dx)
{
     y=pow(sin(x+2*a),3)/(exp(x/b)-pow(x,2)/(b-a));
     printf("%4.1f%14.1f\n",x,y);
}
getch();
return 0;
}
```

6.6 Контрольні питання

1. Що таке цикл? Які є спсоби повторення у циклі?
2. Скільки у мові С є операторів циклу? Як вони називаються?
3. Дайте визначення для цикла з передумовою. Наведіть для нього схему алгоритму.
4. Наведіть формальну схему для цикла з передумовою. Яку мінімальну кількість разів виконуються оператори, що входять у цикл?
5. Наведіть приклад використання циклу з передумовою.
6. Дайте визначення для цикла з постумовою. Наведіть для нього схему алгоритму.
7. Наведіть формальну схему для цикла з постумовою. Яку мінімальну кількість разів виконуються оператори, що входять у цикл?
8. Наведіть приклад використання циклу з постумовою.
9. Для чого у програмі на С використовується `break`? Як це виглядає на схемі алгоритму? Наведіть приклад використання.
10. Для чого у програмі на С використовується `continue`? Як це виглядає на схемі алгоритму? Наведіть приклад використання.
11. Дайте визначення для цикла з параметром. Наведіть для нього схему алгоритму.
12. Наведіть формальну схему для цикла з параметром. Яку мінімальну кількість разів виконуються оператори, що входять у цикл?

13. Які існують варіанти циклу з параметром в залежності від операторів, що входять у його секції?

14. Наведіть приклад використання циклу з параметром: у всіх секціях по одному оператору, є секції з декількома операторами, є секції з відсутніми операторами.

15. Яке значення має параметр циклу `for` після виходу з циклу?

6.7 Тести для самоконтролю

1) Що називається циклом?

А) це група операторів, які заключні у фігурні дужки { }

Б) це група команд, яка завершує програму

В) це група команд, багаторазово виконуваних комп'ютером, поки програма не буде зупинена

Г) це група команд, багаторазово виконуваних комп'ютером, поки деяка умова продовження циклу залишається істинною.

2) Що з названого не є оператором цикла у мові С?

А) цикл з постумовою

Б) цикл з вибором

В) цикл з передумовою

Г) цикл з параметром

3) Дайте визначення циклу `while`.

А) не повторює блок операторів, поки умова у циклі `while` залишається істинною

Б) повторює одну і ту ж дію, поки умова у циклі `while` залишається помилковою

В) повторює блок операторів, поки умова у циклі `while` залишається істинною

Г) немає правильної відповіді

4) Якою має бути умова продовження циклу `while`?

А) `0`

Б) `false`

В) `true`

Г) `0.0`

5) Що виведе наступний фрагмент програми?

```
int i=0;
while (1)
     {
          printf("%d", i ++);
          if (i > 3)
               continue;
          else
               break;
     }
```

А) 0

Б) 3

В) 5

Г) 1

6) Скільки разів буде виконуватися цикл `while` с умовою `while(2)`?

А) 1 раз

Б) нескінченно

В) 2 рази

Г) жодного разу

7) Коли використовується оператор цикла з постумовою?

А) Коли число повторень оператора невідомо

Б) Коли число повторень оператора відомо

В) Коли потрібно мати вкладений цикл

Г) Коли потрібно мати більше 1 умови

8) Оберіть вірне оголошення циклу `do/while`:

А)
```
do
{
          блок інструкцій
}
    while (умова);
```

Б)
```
do (условие);
{
          блок інструкцій
}
while
```

В)
```
do
{
        блок інструкцій
}
while (умова)
```

Г) немає правильної відповіді

9) Скільки разів повинен виконатися `do/while` при невірній умові?

А) 0

Б) 2

В) 1

Г) немає правильної відповіді

10) Число з якого діапазону роздрукує наступна программа?

```
int num;
do
{
  scanf("%d",&num);
} while(num<1||num>10);
printf(“%d”,num);
```

А) 0-10

Б) 1-10

В) 2-10

Г) менше 1 або більше 10

11) В якій послідовності будуть виконуватися вкладені цикли?

А) Спочатку зовнішній потім внутрішній

Б) Спочтку внутрішній потім зовнішній

В) Для кожної умви зовнішнього буде виконана 1 умова внутрішнього

Г) Для кожної умови зовнішнього, внутрішній буде повністю виконан

12) Як правильно зробити об’явлення циклу `for` у програмі на С?

А)
```
for (початкова_інструкція, умова, вираз)
    {
    інструкції;
    }
```

Б)
```
for (умова; вираз)
    {
    інструкції;
```

```
}
```

В)
```
for (початкова_інструкція; умова; вираз)
{
інструкції;
}
```

Г) немає правильної відповіді

13) Як оголосити нескінченний цикл `for`?

А) `for(%)`

Б) `for();`

В) `for(;;)`

Г) `for(1)`

14) Для чого призначений оператор `break`?

А) негайно припиняє виконання самого внутрішнього з охоплюючих його операторів

Б) продовжує виконання самого внутрішнього з охоплюючих його циклів

В) нічого не робить

Г) немає правильної відповіді

15) Яким буде результат:

```
do
{
    break;
}
while(1);
```

А) нескінченний цикл

Б) переривання циклу

В) помилка у програмі

Г) немає правильної відповіді

16) Який оператор не може бути перерваним оператором `break`?

А) `for`

Б) `while`

В) `switch`

Г) `if`

17) Для чого призначений оператор `continue`?

А) перериває виконання циклу

Б) продовжує виконання самого внутрішнього з охоплюючих його циклів

В) перериває перевірку умови

Г) достроково завершує черговий прохід циклу

18) Що робить наступна програма з від'ємними числами?

```
#include <stdio.h>
int main()
{
int a,i,n,s;
n=6; s=0;
for(i=0; i<n; i++)
  {
   scanf("%d",&a);
   if(a<=0)
     continue;
   s+=a;
  }
printf("сума = %d \n",s);
return 0;
}
```

А) підсумовує від’ємні числа

Б) пропускає від’ємні числа, підсумовує додатні

В) підсумовує додатні та від’ємні числа

Г) підраховує кількість від’ємних чисел

РОЗДІЛ 7 СТАТИЧНІ МАСИВИ

7.1 Загальні положення

Поряд із простими типами даних, розглянутими вище, мова С містить ряд структурованих типів даних, що можуть представляти сукупності значень. Масиви, структури і файли характеризуються множиною утворюючих цей тип елементів, тобто змінна чи константа структурованого типу завжди має кілька компонентів. Кожен компонент, у свою чергу, може належати структурованому типу, що дозволяє говорити про можливу вкладеність типів.

Масив – це упорядкований набір змінних одного типу. Масиви містять фіксоване число компонентів, що задається при визначенні змінних типу масиву.

7.2 Статичні одновимірні масиви

Статичними називають масиви, які мають фіксоване число елементів. При необхідності роботи з масивом значень потрібно спочатку оголосити цей масив, указавши максимальну кількість елементів, що може вмістити масив.

У мові С до першого використання елементів масиву потрібно оголосити його, вказавши ім'я типу, назву масиву й у квадратних дужках – кількість елементів. Зверніть увагу, що кількість елементів масиву повинна бути константою для того, щоб під час компіляції для масиву було виділено необхідна кількість пам’яті. Як виділяти пам’ять для масивів під час викнання програми ви дізнаєтесь у наступній главі.

Для звертання до елемента масиву використовується вид: ім'я_масиву[індекс]. Індекс задає місце елемента в масиві. Обмежень на кількість індексів немає. При звернені до елементів масиву в мові С слід враховувати, що нумерація елементів починається з нуля. Таким же чином позначається масив і на схемі алгоритмів.

Приклад оголошення масивів:

```
const n=15;
int a,b[10];// b – масив
float c[n]; // c – масив
a[0]=2; a[0+1]=5;
b[9]=a[0]+a[1];
c[0]=(float)a[0]/b[9];
```

```
printf(“%d%d%d%f”, a[0], a[1], b[9], c[0]);
```

Як інший варіант можна спочатку визначити тип, що являє собою масив елементів заданого типу, а потім повідомляти змінні типу масиву.

Інший приклад оголошення масивів:

```
typedef int type_mas[10];
type_mas mas;
for (int i=0;i<10;i++)
{
mas[i]=i;
printf(“%d\n”, mas[i]);
}
```

Приклад. Ввести масив з клавіатури та вивести його на екран.

```
#include<stdio.h>

void main()
{
     const int N=10;
     int x[N];
     int i;
     printf("\nInput array: \n");
     for(i=0; i<N-1; i++)
          scanf("%d", &x[i]);
     printf("\n");
     for(i=0;i<N-1;i++)
          printf("%d ", x[i]);
}
```

У С відсутня перевірка перевірка меж масиву, можна вийти за один кінець масиву і записати значення в яку-небудь комірку, що не відноситься до масиву, чи навіть у код програми. Робота з перевірки меж масиву покладається на програміста.

Над елементами масивів можна виконувати дії, коло яких обмежено типом даних елементів масиву. У мові С присвоєння масивів потрібно виконувати поелементно.

Можлива початкова ініціалізація елементів масиву. Для мови С:

```
int a[10] = {12, 23, 34, 45, 56, 67, 78, 89, 90, 101};
```

Якщо розмір масиву не визначений, то з'являється масив з такою кількістю елементів, скільки їх у списку ініціалізації.

```
int b[]={1,2,3,4,5} // створюється масив з п'яти елементів
```

Приклад. Задані два масиви A і B, елементи яких упорядковані за зростанням. Скласти з елементів цього масиву новий масив C, елементи якого також упорядковані за зростанням.

```
#include <stdio.h>
#include <conio.h>

void main()
{
     const int m=10, n=8;
     int a[m], b[n], c[m+n], i, j, k;
     printf("\nInput %d elements of array a: ", m);
     for (i=0; i<m; i++)
          scanf("%d", &a[i]);
     printf("\nInput %d elements of array b: ", n);
     for (i=0; i<n; i++)
          scanf("%d", &b[i]);
     i=j=0;
     for (k=0;k<m+n;k++)
     {
          if (((a[i]<=b[j])&&(i<m))|| (j>=n))
          {
               c[k]=a[i];
               i++;
          }
          else if (((b[j]<a[i])&&(j<n))||(i>=m))
          {
               c[k]=b[j];
               j++;
                }
     }
```

```
     for (k=0; k<m+n; k++)
          printf("%d ", c[k]);

     getch();
}
```

Приклад. Задана послідовність цілих чисел, що закінчується нулем. Роздрукувати всі від'ємні числа цієї послідовності в зворотному порядку

```
#include <stdio.h>
#include <conio.h>

void main()
{
     const int n=10;
     int mas[n], i, k;
     printf("Input %d elements of array: ", n);

for (i=0; i<n; i++)
          scanf("%d", &mas[i]);
     for (i=0; i<n; i++)
          printf("%d ", mas[i]);
     for (i=0; i<n; i++)
          if (mas[i]==0)
          {
               k=i;
               break;
          }
     printf("\n");
     for (i=0; i<k; i++)
          if (mas[i]<0)
               printf("%d ", mas[i]);
     for (i=k-1; i>=0; i--)
          if (mas[i]>0)
               printf("%d ", mas[i]);
getch();
}
```

Приклад. Дано натуральне число N (N – парне) і одновимірний масив A_1, A_2, …, A_N логічних елементів. Виконати циклічний зсув першої половини масиву справа наліво, а другий – зліву направо.

```
#include <stdio.h>

void main()
{
     const int N = 10;
     bool temp;
     int i, booltmp, M;
     bool arr[N];

     printf("Input %d binary values into array\n", N);
     for(i=0; i<N; i++)
     {
          scanf("%d", &booltmp);
          arr[i] = booltmp;
     }

     printf("Original array:\n");

     for(i=0; i<N; i++)
     {
          if(arr[i])
          {
               printf("1 ");
          }
          else
          {
               printf("0 ");
          }
     }

     M=N/2-1;
```

```
	temp = arr[0];
	for(i=0; i<M; i++)
	{
		arr[i] = arr[i+1];
	}
	arr[M] = temp;

	M+=1;

	Temp = arr[N-1];
	for(i=N-1; i>=M; i--)
	{
		arr[i] = arr[i-1];
	}
	arr[M] = temp;

	printf("\nArray after cyclic shift:\n");
	for(i=0; i<N; i++)
	{
		if(arr[i])
		{
			printf("1 ");
		}
		else
		{
			printf("0 ");
		}
	}

	system(“pause”);
}
```

Приклад. Дано натуральне число N (N>5) і одновимірний масив A_1, A_2, …, A_N дійсних елементів. Визначити три максимальних і два мінімальних значення цього масиву.

```
#include <stdio.h>
```

```
int main() {
float max[3]={0}, min[2]={0};
int n, i;
float temp;
do
{
     printf("Input number of elements [n>=5]: ");
     scanf("%d",&n);
}
while(n<5);
printf("\4\4\4 Please make sure that the first 3 elements are
positive!\4\4\4\n");
for (i=0; i<n; i++)
{
     printf("Input element [%d]: ", i);
     scanf("%f",&temp);
     if (temp>max[0])
     {
          if (temp>max[1])
          {
               if (temp>max[2])
               {
                    max[0]=max[1];
                    max[1]=max[2];
                    max[2]=temp;
               }
               else
               {
                    max[0]=max[1];
                    max[1]=temp;
               }
          }
          else max[0]=temp;
     }
     if (i==0) min[0]=temp;
     else if (i==1)
      {
```

```
            if (temp>min[0]) min[1]=temp;
            else
            {
                  min[1]=min[0];
                  min[0]=temp;
            }
      }
      else
      {
            if (temp<min[0])
            {
                  min[1]=min[0];
                  min[0]=temp;
            }
            else if (temp<min[1])
                  min[1]=temp;
            }
      }
printf("Maximums: %4.1f %4.1f %4.1f\nMinimums: %4.1f %4.1f\n",
max[0], max[1], max[2], min[0], min[1]);
return 0;
}
```

Зазначимо, що для пошуку більшої кількісті максимумів та мінімумів доцільно зберегти дані в масиві, відсортувати цей масив, а потім вибирати потрібну кількість елементів з початку або з кінця масиву.

Приклад. Дано натуральне число N і одновимірний масив A_1, A_2, …, A_N натуральних чисел. Для кожного елемента визначити число його входжень у даний масив.

```
#include <conio.h>
#include <stdio.h>

int main()
{
      const int n=10;
      float mas[n];
```

```
     int counter=0, i, j;
     for (i=0;i<n;i++)
     {
          printf("Input mas[%d]=",i+1);
          scanf("%f",&mas[i]);
     }
     for (i=0;i<n;i++)
     {
          counter=0;
          for (j=0;j<n;j++)
               if (mas[i]==mas[j]) counter++;
          printf( "\nValue %4.2f includes in an array %d times",
mas[i],counter);
     }
     getch();
     return 0;
}
```

Деяким недоліком даної реалізації можна вважати форму виводу результату. Кількість входжень буде виводитися для кожного з елементів масиву, тобто при наявності однакових за значенням елементів може виникнути дублювання. Для усунення цього ефекту потрібно зберігати інформацію про те, чи виводилось певне значення на екран. Якщо так, повторний вивід не потрібний. Звісно, така реалізація потребує ускладнення алгоритму.

Приклад. Дано натуральне число N (N – парне) і одновимірний масив A_1, A_2, …, A_N дійсних чисел. Замінити елементи, розташовані в парних позиціях першої половини масиву, подвоєними значеннями елементів, розташованих у непарних позиціях другої половини масиву.

```
#include <stdio.h>
int main() {
     const int n=12;
     float array[n];
     int i;
     for (i=0;i<n;i++)
     {
          printf("Input value for element N %d: ",i+1);
```

```
        scanf("%f",&array[i]);
    }
    for (i=0;i<n/2;i++)
    {
        if (i%2!=0)
        {
            int newInd=i+n/2;
            if ((n/2)%2==0)
                newInd--;
            array[i]=array[newInd]*2;
        }
    }
    printf("New array:\n");
    for (i=0;i<n;i++)
        printf("Element N %d has value %4.2f\n",i+1,array[i]);
    return 0;
    }
```

Приклад. Дано натуральне число N і одновимірний масив A_1, A_2, …, A_N цілих чисел. Визначити найбільше й найменше значення, отримані значення розглядати як кінці відрізка. Розбити відрізок на 5 діапазонів значень за зростанням й підрахувати частоту влучень елементів масиву в кожен із цих діапазонів.

```
#include <stdio.h>
int main()
{
    const int n=10;
    int  array[n];
    int counter;
    float min, max;

    for (int i=0; i<n; ++i)
    {
        printf("A[%d]=",i+1);
        scanf("%d",&array[i]);
    }
```

```
	for (int i=1, temp; i<n; ++i) //sorting array
	{
		temp=array[i];
		for (int j=i-1; j>=0; --j)
		{
			if (array[j]<temp) break;
			array[j+1]=array[j];
			array[j]=temp;
		}
	}
	printf("Sorted array: ");
	for (int i=0; i<n; ++i)
		printf("%d  ",array[i]);
	printf("\nMin: %d\tMax: %d\n",array[0],array[n-1]);
	min=array[0];
	for (int i=0;i<5;++i)
	{
		counter=0;
		max=array[0]+(float)(array[n-1]-array[0])/5.0*(i+1);
		printf("(%4.2f;%4.2f):\t",min,max);
		for(int i=0;i<n && array[i]<max;++i)
		{
			if (array[i]>min)
			{
				printf(" %d",array[i]);
				++counter;
			}
		}
		printf("\tTotal: %d elements.\n",counter);
		min=max;
	}
	return 0;
}
```

При необхідності збереження та сортування деякого набору чисел (у загальному виді – ключів) потрібно використовувати масив, елементи якого обробляються за допомогою вкладених циклів.

Приклад. Виконати введення і сортування 5 цілих елементів за методом «бульбашки».

```
#include <stdio.h>
void main()
{
     const int numb=5;
     int mas[numb];
     int i,j,t;
     for(i=0;i<numb;i++)
          scanf("%d",&mas[i]);
     for(i=0;i<numb-1;i++)
          for(j=numb-1;j>i;j--)
               if(mas[j]<mas[j-1])
               {
                    t=mas[j];
                    mas[j]=mas[j-1];
                    mas[j-1]=t;
               }
     for(i=0;i<numb;i++)
                    printf("%d ",mas[i]);
}
```

Модифікація цього методу: відстежувати перестановку чисел у масиві, робити наступний перегляд елементів масиву тільки у разі наявності попередньої перестановки.

```
#include <stdio.h>

void main()
{
     const int N=5;
     int i, j, save, A[N];
     bool flag;
```

```
	for(i=0;i<N;++i)
		scanf("%d", &A[i]);

	for(j=1;j<=N-1;j++)
	{
		flag=false;
	for(i=N-2;i>=0;--i)
		if(A[i]>A[i+1])
		{
			save=A[i];
			A[i]=A[i+1];
			A[i+1]=save;
			flag=true;
		}
	if (flag==false) break;
	}

	for(i=0;i<N;++i)
		printf("%d ", A[i]);
}
```

7.3 Багатовимірні масиви

Як тип елементів масиву може бути використаний інший масив. Глибина вкладеності структурованих типів узагалі, а отже, і масивів – довільна, тому кількість елементів у списку індексних типів (розмірність масиву) не обмежена.

Більш детально зупинимося на двовимірних масивах. Двовимірні масиви зберігаються у виді матриці, де перший індекс відповідає за рядок, а другий за стовпець.

7.4 Статичні двовимірні масиви

Статичний двовимірний масив має фіксовану кількість рядків та стовпців. Для звернення до елементів масиву в мові С потрібно після імені змінної масиву вказати в одній парі квадратних дужок індекс рядка, а в другій – індекс стовпцю.

Це означає, що правий індекс змінюється швидше першого, якщо рухатися по масиву в порядку розташування елементів у пам'яті.

```
int a[2][2];
a[0][0]=1;
a[1][0]=2;
a[0][1]=3;
a[1][1]=4;
```

то в пам'яті послідовно один за одним будуть розташовані байти зі значеннями 1, 3, 2, 4.

Ініціалізація двовимірного масиву мовою С:

```
int b[2][2]={{1, 2}, {3, 4}};
```

Значення групуються у фігурних дужках по рядках. Таким чином, 1 і 2 ініціалізуватиме `b[0][0]` і `b[0][1]`, 3 і 4 ініціалізуватиме `b[1][0]` і `b[1][1]`. Якщо для даного рядка недостатньо ініціалізуючих значень, його елементи, що залишилися, ініціалізуватимуться нулями. Таким чином, оголошення `int b[2][2]={{1},{3,4}};` ініціалізуватиме `b[0][0]` одиницею, `b[0][1]` нулем, `b[1][0]` трійкою і `b[1][1]` четвіркою.

Приклад. У квадратній матриці поміняти місцями рядки з найбільшим і найменшим елементами

```
#include <stdio.h>

void main()
{
     const int n=5;
     int i, j, min, max, imin, imax, temp;
     int mas[n][n];

     for (i=0; i<n; i++)       // ввід матриці
          for(j=0; j<n; j++)
               scanf("%d", &mas[i][j]);
```

```
    min = max = mas[0][0];
    for (i=0; i<n; i++)
    {
        for (j=0; j<n; j++)
        {
            if (mas[i][j]<min)
            {
                min=mas[i][j];
                imin=i;// номер рядка з найменш.елементом
            }
            if (mas[i][j]>max)
            {
                max=mas[i][j];
                imax=i;//номер рядка з найбільш.елементом
            }
        }
    }
    for (j=0;j<n;j++)// обмін місцями рідків
    {
        temp=mas[imin][j];
        mas[imin][j]=mas[imax][j];
        mas[imax][j]=temp;
    }
    for (i=0;i<n;i++) // вивод матриці
    {
        for (j=0;j<n;j++)
        {
            printf("%d ",mas[i][j]);
        }
        printf("\n");
    }
}
```

Приклад. Задана квадратна матриця порядку n. З’ясувати, чи є матриця симетричною відносно головної діагоналі.

```
#include <stdio.h>
```

```
void main()
{
     const int n=5;
     int mas[n][n], i, j, flag=1;

     for (i=0;i<n;i++)
          for (j=0;j<n;j++)
               scanf("%d", &mas[i][j]);

     for (i=0;i<n;i++)
     {
          for (j=0;j<n;j++)
               printf("%d ", mas[i][j]);
          printf("\n");
     }

     for (i=0;i<n;i++)
          for (j=0;j<n;j++)
               if (mas[i][j]!=mas[j][i])
                    flag=0;
     if (flag)
          printf("\nMatrix is symmetric");
     else
          printf("\nMatrix is not symmetric");
}
```

Приклад. Дано матрицю розмірності N на M. Обчислити кількість рядків матриці, у яких немає жодного від'ємного елемента. Схема алгоритму приведена поруч.

```
#include <stdio.h>

void main () {
const int N=5, M=3;
     float matrix[M][N];
     int i, j, k;
```

```
     printf("Input matrix of %d columns and %d rows.\n", N, M);
     for(i=0; i<M; i++)
          for(j=0; j<N; j++)
               scanf("%f", &matrix[i][j]);
     k=M;
     for(i=0; i<M; i++)
          for(j=0; j<N; j++)
               if(matrix[i][j]<0) {
                    k--;
                    break;
               }

printf("Matrix has %d rows, which

don’t contain any negative elements", k);

     system(“pause”);
}
```

Приклад. Дано матрицю розмірності N на M. Знайти рядок, у якому максимальний елемент є мінімальним у відповідному стовпці.

```
#include <stdio.h>
int main()
{
     const int m=4, n=3; //m – rows, n – columns
     float a[m][n], max[n], min[m];
     int i,j;
     for (i=0;i<m;i++)
          for (j=0;j<n;j++)
          {
               printf("Element[%d][%d]=",i+1,j+1);
               scanf("%f",&a[i][j]);
          }

     printf("Your matrix is:\n");
```

```
for (i=0;i<m;i++)
{
     for (j=0;j<n;j++)
          printf("%6.2f\t",a[i][j]);
     printf("\n");
}

printf("\nMaximums in columns: ");
for (j=0;j<n;j++)
{
     for (i=0;i<m;i++)
     {
          if(i==0)
               max[j]=a[0][j];
          else if(a[i][j]>max[j])
                max[j]=a[i][j];
     }
     printf("%4.1f\t",max[j]);
}

printf("\nMinimums in rows: ");
for (i=0;i<m;i++)
{
     for (j=0;j<n;j++)
     {
          if(j==0)
               min[i]=a[i][0];
          else if(a[i][j]<min[i])
                min[i]=a[i][j];
     }
     printf("%4.1f\t",min[i]);
}
for (j=0;j<n;j++)
     for (i=0;i<m;i++)
          if (max[j]==min[i])
```

```
                    printf("\nElement %4.1f, row %d, column
%d\n",a[i][j],i+1,j+1);
     return 0;
}
```

Завжди потрібно оцінювати, чи може програма обчислити результат у будь-якому разі, тобто при будь-яких початкових даних, або може виникнути ситуація, коли резутат не може бути обчислений. Така ситуація потребує додаткової перевірки в програмі.

Приклад. Дано квадратну матрицю порядку N. У матриці обчислити середнє арифметичне позитивних елементів, що стоять на головній діагоналі.

```
#include <stdio.h>
#include <conio.h>
int main() {
     const int n=4;
     int i,j,counter=0;
     float matr[n][n],sum=0;
     for (i=0;i<n;i++)
          for (j=0;j<n;j++)
          {
               printf("Input the element of matrix – a[%d][%d] =
",i+1,j+1);
               scanf("%f",&matr[i][j]);
          }
     for (i=0;i<n;i++)
          if (matr[i][i]>0)
          { sum+=matr[i][i]; counter++; }
     printf("Your matrix is:\n");
     for (i=0;i<n;i++)
     {
          for (j=0;j<n;j++)
               printf("%6.2f\t",matr[i][j]);
          printf("\n");
     }
     if (counter==0)
          printf("No positive elements on the main diagonal");
```

```
    else printf("The arithmetic mean=%6.2f\n",sum/counter);
    getch();
    return 0;
}
```

7.5 Контрольні питання

1. Що називається масивом? Наведіть приклади, коли зручніше використовувати масив ніж сукупність простих змінних.
2. Принцип нумерації елементів масиву. Звернення до елементів масиву.
3. Які масиви називають статичними? Як вказується кількість єлементів для статичного масиву?
4. Як оголошується статичний масив? Якого типу можуть бути елементи масиву? Наведіть приклади оголошень.
5. Які дії можна виконувати над єлементами масива?
6. Як використання циклу `for` допомагає індексувати елементи масиву?
7. Як можна виконати початкову ініціалізацію одновимірного масиву?
8. Що є багатовимірним масивом? Наведіть приклади двовимірного та тривимірного масивів.
9. Статичний двовимірний масив: оголошення, принцип нумерації рядків та стовпців, принцип звертання до елементів масиву. Розташування елементів двовимірного масиву у пам’яті.
10. Як можна виконати початкову ініціалізацію двовимірного масиву? Розгляньте приклади, коли значень для ініціалізації достатньо та коли значень не вистачає.

7.6 Тести для самоконтролю

1) Що використовується при звертанні до елементу масиву?

А) покажчик

Б) індекс

В) довільне звернення

Г) посилання

2) Яким може бути тип елементів массиву?

А) будь-яким, але однаковим для всіх елементів

Б) цілим

В) дійсним

Г) будь-яким, різним для всіх елементів

3) Які масиви називають статичними?

А) такі, що не потрібно оголошувати

Б) такі, що мають фіксоване число елементів

В) такі, що містять тільки цілі числа

Г) немає правильної відповіді

4) З якого елемента починається індексація масиву?

А) з першого

Б) з останнього

В) це залежить від довжини масиву

Г) з нульового

5) Який фрагмент програми відповідає коректному оголошенню статичного масиву?

А)
```
const int a=3;
int b; b=a;
float mas[b];
```

Б)
```
int b; b=5;
float mas[b];
```

В)
```
int a=3;
const int b=a;
float mas[b];
```

Г)
```
int b;
scanf(“%d”,&b);
float mas[b];
```

6) Які дії можна виконувати над елементами масиву?

А) тільки арифметичні

Б) тільки логічні

В) будь-які

Г) коло дій обмежено типом даних елементів масиву

7) Як представляються двовимірні масиви?

А) у виді матриці

Б) у виді вектора

В) як множина елементів

Г) немає правильної відповіді

8) Як правильно оголосити двовимірний масив цілих чисел, що містить 3 рядки та 4 стовпця?

А) `float mas[4][3];`

Б) `int array[3][4];`

В) `int s[4][3];`

Г) `int massiv[3,4];`

9) Як індексуватись до елементів двовимірного масиву `mas`, що містить 3 рядки та 4 стовпця?

А) `mas[0][0] … mas[2][3]`

Б) `mas[1][1] … mas[3][4]`

В) `mas[1][1] … mas[2][3]`

Г) `mas[0][0] … mas[3][4]`

10) Як можна проініціалізувати двовимірний масив `mas`, що містить 2 рядки та 3 стовпці?

А) `int mas[2][3]={{1, 2}, {3, 4}, {5, 6}};`

Б) `int mas[3][2]={{1, 2}, {3, 4}, {5, 6}};`

В) `int mas[2][3]={{1, 2, 3}, {4, 5, 6}};`

Г) `int mas[3][2]={{1, 2, 3}, {4, 5, 6}};`

11) Що відбудеться при виконанні наступного коду:

```
int mas[5],i;
for (i=0;i<=5;i++)
     scanf(“%d”,&mas[i]);
```

А) вихід за ліву межу масиву

Б) вихід за праву межу масиву

В) помилки немає

Г) немає правильної відповіді

12) Чи можна двовимірний масив переписати в одновимірний?

А) ні, це помилка

Б) так, у будь-якому разі

В) так, але у будь-якому разі буде втрата рядка даних

Г) так, якщо одновимірний масив має достатній розмір та той же самий тип даних

РОЗДІЛ 8 ПОКАЖЧИКИ

8.1 Логічна структура пам’яті програми. Динамічна пам’ять

Область пам'яті, в якій розміщується програма, поділяється на розділи за призначенням і способам управління даними:

- *сегмент коду* використовується для зберігання коду програми – функцій. Функції поміщаються в сегмент коду на етапі компіляції програми і знаходяться там до завершення роботи програми, тому всі функції в С мають глобальний час життя й існують протягом усього часу виконання програми;
- *статична пам'ять (сегмент даних)* призначена для зберігання змінних протягом усього часу виконання програми. Якщо для змінної в який-небудь момент роботи програми виділена пам'ять в сегменті даних, вона там перебуватиме до завершення роботи програми, навіть якщо ця змінна більше не потрібна. За замовчуванням в сегменті даних зберігаються глобальні змінні;
- *стек* – це область пам'яті, в якій зберігаються локальні змінні і параметри функцій. При виклику функції її параметри і локальні змінні поміщаються в стек. Стек функціонує за принципом стопки тарілок – значення, поміщене в стек першим, виявляється на дні стека – унизу стопки, в той час як останнє значення – на вершині стека – зверху стопки. По завершенні роботи функції всі дані, що належать цій функції, видаляються з стека. Очищення стека починається з вершини, тобто з значень, поміщених в стек останніми;
- *динамічна пам'ять (купа)* дозволяє програмісту керувати процесом виділення пам'яті під змінні і звільненням пам'яті. Змінні, що розміщуються в динамічній пам'яті, називаються динамічними змінними.

Пам'ять для зберігання даних може виділятися як статично, так і динамічно. У першому випадку виділення пам'яті виконує компілятор, що зустрів при компіляції оголошення об'єкта. Відповідно до типу об'єкта обчислюється обсяг пам'яті, необхідний для його розміщення. Клас пам'яті задає місце, де ці об'єкти (дані) будуть розташовуватися. Це може бути сегмент даних або стек. Часто виникають ситуації, коли заздалегідь не відомо, скільки об'єктів – чисел, рядків тексту та інших даних буде зберігати програма. У цьому випадку використовується динамічне виділення пам'яті, коли пам'ять виділяється і звільняється в процесі виконання програми. При використанні динамічної пам'яті відпадає необхідність заздалегідь розподіляти пам'ять для зберігання даних, що

використовуються програмою. Управління динамічної пам'яттю – це здатність визначати розмір об'єкта і виділяти для його зберігання відповідну область пам'яті в процесі виконання програми. При динамічному виділенні пам'яті для зберігання даних використовується спеціальна область пам'яті, так звана «купа» (heap) . Обсяг «купи» і її місце розташування залежать від моделі пам'яті, яка визначає логічну структуру пам'яті програми.

8.2 Покажчик: загальні положення

Покажчик – це змінна, значенням якої є адреса деякого об'єкта (зазвичай іншої змінної) у пам'яті комп'ютера. Подібно до того, як змінна типу `char` має в якості значення символ, а змінна типу `int` – цілочисельне значення, змінна типу покажчика має в якості значення адреса комірки оперативної пам'яті.

Допустимі значення для змінної-покажчика – множина адресів оперативної пам'яті комп'ютера.

Покажчик є однією з найбільш важливих концепцій мови С.
Правильне розуміння і використання покажчиків особливо необхідно для складання хороших програм з наступних причин:

• покажчики є засобом, за допомогою якого функції можуть змінювати значення переданих в неї аргументів;

• за допомогою покажчиків виконується динамічний розподіл пам'яті;

• покажчики дозволяють підвищити ефективність програмування;

• покажчики забезпечують підтримку динамічних структур даних (двійкові дерева, зв'язкові списки).

Однак покажчик може викликати і ряд труднощів, наприклад, якщо покажчик містить неправильне значення, програма може бути непрацездатною. Можна легко помилитися при використанні покажчиків; до того ж помилки, пов'язані з неправильними значеннями покажчиків, знайти дуже важко.

Отже, покажчик – це новий тип даних. Для нього визначені поняття константи, змінної, масиву. Як і будь-яку змінну, покажчик необхідно оголосити. Оголошення покажчика складається з імені базового типу, символу * (зірочка) та імені змінної.

Загальна форма оголошення покажчика: `тип * ім'я;`

Тип покажчика візначає тип об'єкта, на Який покажчик буде посілатіся, Наприклад, `int * p1;`

Так само можна створювати покажчик на покажчик (`int ** ptr`).

Фактично покажчик будь-якого типу може посилатися на будь-яке місце в пам'яті, але виконуються над покажчиком операції істотно залежать від його типу. Так, якщо оголошений покажчик типу `int *`, компілятор припускає, що будь-яку адресу, на який він посилається, містить змінну типу int, хоча це може бути і не так. Отже, оголошуючи покажчик, необхідно переконатися в тому, що його тип сумісний з типом об'єкта, на який він буде посилатися.

8.3 Операція отримання адреси

Поняття покажчика тісно пов'язане з поняттям адреси об'єкта. У мові С є спеціальна операція, що дозволяє отримати адресу будь-якої змінної:
`&p` – отримання адреси, де `p` – ідентифікатор змінної. Результатом операції є адреса змінної `p`.

Приклад.

```
# include <stdio.h>

int main()
{
     int x=2,*p;
     p = &x;
     printf("x = %d\nAddress of x=%d", x, p);
     return 0;
}
```

Поняття змінної типу покажчик також пов'язане з операцією непрямої адресації *, званої ще операцією розіменування, яка має структуру:
`* p` – розіменування, де `p` – ідентифікатор змінної-покажчика.

Цей запис означає, що в осередок з адресою, записаним в змінну `p`, поміщено значення деякої величини.

Оператори `{ptr = &a; val = * ptr;}` рівнозначні оператору `val = a;`
Наприклад:

```
n = 32;
p = &n;     /* p-адреса комірки, куди записано n */
v = *p;
```

В результаті виконання цих дій в змінну v буде поміщено число 32.

8.4 Операції над покажчиками

Над покажчиками визначено 5 основних операцій:

1. Визначення адреси покажчика: &p, де p – покажчик (&p – адреса комірки, в якій знаходиться покажчик).

2. Присвоєння. Вказівником можна привласнити адресу змінної p = &q , де p – вказівник, q – ідентифікатор змінної.

3. Визначення значення, на яке посилається покажчик: *p (операція непрямої адресації).

4. Збільшення (зменшення) покажчика. Збільшення виконується як за допомогою операції додавання (+) , так і за допомогою операції інкремента (++). Зменшення – за допомогою операції віднімання (-) або декремента (--).

Наприклад, нехай p1 – покажчик, тоді p1++ переміщає покажчик на:

– 1 байт, якщо *p1 має тип char;

– 4 байти, якщо *p1 має тип int (в 32-розрядній операційній системі) або 2 байти (в 16 розрядній операційній системі);

– 4 байти , якщо *p1 має тип float.

5. Різниця двох покажчиків. Нехай p1 і p2 – покажчики одного і того ж типу. Можна визначити різницю p1 і p2, щоб знайти, на якій відстані один від одного знаходяться елементи масиву.

Операції адресної арифметики підпорядковуються наступним правилам. Після збільшення значення змінної – покажчика на 1 даний покажчик буде посилатися на наступний об'єкт свого базового типу. Після зменшення – на попередній об'єкт. Для всіх покажчиків адресу збільшується або зменшується на величину, рівну розміру об'єкта того типу, на який вони вказують. Тому покажчик завжди посилається на об'єкт з типом, тотожним базового типу покажчика.

Стосовно до вказівниками на об'єкт типу char операції адресної арифметики виконуються як звичайні арифметичні операції, тому що довжина об'єкта `char` завжди дорівнює 1.

Операції адресної арифметики не обмежені збільшенням (інкрементом) і зменшенням (декрементом). До покажчиків, наприклад, можна додавати або віднімати цілі числа.

При операції віднімання двох покажчиків можна визначити кількість об'єктів, розташованих між адресами, на які вказують ці два покажчика. При цьому необхідно, щоб покажчики мали той самий тип. Крім того, стандартом С дозволяється порівняння двох покажчиків. Як правило, порівняння покажчиків може виявитися корисним тільки тоді, коли два покажчика посилаються на загальний об'єкт, наприклад, на масив.

Всі інші операції над покажчиками заборонені.

Приклад: дано адреси змінних `&a = 63384, &b = 64390, &c = 64404`. Що надрукує ЕОМ?

```
#include <stdio.h>

int main()
{
     float a, *p1;
     int b, *p2;
     char c, *p3;
     a = 2.5; b = 3; c = 'A';
     p1 = &a; p2 = &b; p3 = &c;
     p1++; p2++; p3++;
     printf("p1 = %u, p2 = %u, p3 = %u", p1, p2, p3);
     return 0;
}
```

Результат: `p1 = 63388, p2 = 64392, p3 = 64405.`

8.5 Функції динамічного розподілу пам'яті

Покажчики дозволяють динамічно розподіляти пам'ять у програмі на мові С. Термін динамічний розподіл пам'яті означає, що програма може отримувати необхідну їй пам'ять вже в ході свого виконання. Як відомо, пам'ять для глобальних змінних виділяється на етапі компіляції. Локальні змінні зберігаються в стеку. Отже, в ході виконання програми неможливо оголосити нові глобальні або локальні змінні. І все ж у ході виконання програми виникає необхідність виділити додаткову пам'ять. Для роботи з динамічним виділенням пам’яті у мові С існують функції `malloc(), calloc(), realloc()` та `free()`.

```
void* malloc (size_t size);
```

Покажчики дозволяють динамічно розподіляти пам'ять у програмі на мові С. Термін динамічний розподіл пам'яті означає, що програма може отримувати необхідну їй пам'ять вже в ході свого виконання. Як відомо, пам'ять для глобальних змінних виділяється на етапі компіляції. Локальні змінні зберігаються в стеку. Отже, в ході виконання програми неможливо оголосити нові глобальні або локальні змінні. І все ж у ході виконання програми виникає необхідність виділити додаткову пам'ять.

```
void *calloc(size_t num,  size_t size);
```

Функція `calloc()` виділяє пам'ять, розмір якої дорівнює значенню виразу `num * size`, тобто пам'ять, достатню для розміщення масиву, що містить num об'єктів розміром `size`. Всі біти розподіленої пам'яті ініціалізуються нулями.

Функція `calloc()` повертає покажчик на перший байт виділеної області пам'яті. Якщо для задоволення запиту немає достатнього обсягу пам'яті, повертається нульовий вказівник.

```
void *realloc(void *ptr, size_t size);
```

Блок пам'яті, що адресується параметром `ptr`, звільняється, а замість нього виділяється новий блок. Вміст нового блоку збігається з вмістом вихідного (принаймні збігаються перші `size` байтів). Функція повертає покажчик на новий блок. Причому допускається, щоб новий і старий блоки починалися з однакової адреси (тобто покажчик, повертається функцією `realloc()`, може збігатися з покажчиком, переданим у параметрі `ptr`).

Якщо покажчик `ptr` нульовий, функція `realloc()` просто виділяє `size` байтів пам'яті і повертає покажчик на цю пам'ять. Якщо значення параметра `size` дорівнює нулю, пам'ять, що адресується параметром `ptr`, звільняється.

Якщо в динамічно розподіляємій області пам'яті немає достатнього обсягу

вільної пам'яті для виділення `size` байтів, повертається нульовий показчик, а вихідний блок пам'яті залишається незмінним.

Функція `free()` призначена для звільнення пам'яті, виділеної функціями `malloc(), calloc(), realloc()`. Після виконання функції звільнена пам'ять може бути виділена знову під інші дані:

```
void free(void *ptr);
```

де `ptr` – покажчик на область пам'яті, створеної лише функціями динамічного управління пам'яттю `malloc(), calloc(), realloc().` Використання інших покажчиків у функції `free` робить її поведінку невизначениою і може призвести до втрати управління пам'яттю.

8.6 Контрольні питання

1. Опишіть логічну структуру пам’яті програми.
2. Які особливості має динамічна пам’ять?
3. Що називається покажчиком? Які допустимі значення для змінної-покажчика?
4. Які є причини і обгрунтування для використання покажчиків?
5. Як треба оголошувати покажчики? Наведіть приклади оголошення покажчиків різних типів.
6. Що таке покажчики на покажчики? Наведіть приклади оголошення.
7. Як застосовується операція отримання адреси?
8. Які ви знаєте операції над покажчиками?
9. Додавання та віднімання покажчиків.
10. Порівняння покажчиків.
11. Динамічний розподіл пам'яті. Локальні та глобальні змінні.
12. Функція `malloc()`: призначення, прототип, аргументи, результат, що повертається. Навести приклади.
13. Функція `calloc()`: призначення, прототип, аргументи, результат, що повертається. Навести приклади.
14. Функція `realloc()`: призначення, прототип, аргументи, результат, що повертається. Навести приклади.
15. Функція `free()`: призначення, прототип, аргументи, результат, що повертається. Навести приклади для простих зміних та масивів.

8.7 Тести для самоконтролю

1) Що з наступного не є назвою розділу області пам'яті, в якій розміщується програма?

А) сегмент коду

Б) сегмент даних

В) сегмент файлу

Г) стек

2) Куди поміщаються на етапі компіляції програми усі функції?

А) у сегмент коду

Б) це залежить від розміру функції

В) це залежить від наявності параметрів у функції

Г) у стек

3) Що зберігається за замовчуванням у сегменті даних?

А) усі змінні

Б) тільки глобальні змінні

В) тільки локальні змінні

Г) немає правильної відповіді

4) Що зберігається у стеку?

А) локальні змінні і параметри функцій

Б) тільки локальні змінні

В) тільки параметри функцій

Г) глобальні змінні

5) Як відбувається управління пам'яттю при її динамічному розподілі?

А) пам'ять виділяється в процесі виконання програми, звільняється при закінченні програми

Б) пам'ять виділяється на початку роботи програми, звільняється в процесі виконання програми

В) пам'ять виділяється на початку роботи програми, звільняється при закінченні програми

Г) пам'ять виділяється і звільняється в процесі виконання програми

6) Що таке покажчик?

А) це змінна, яка приймає значення будь-якого типу

Б) це адреса, з якої починається стек даних

В) це змінна, значенням якої є адреса деякої змінної у пам'яті комп'ютера

Г) немає правильної відповіді

7) Виберіть загальну форму оголошення покажчика:

А) `ім'я * тип`

Б) `тип * ім'я`

В) `pointer ім'я`

Г) `ім'я * pointer`

8) Якщо `value` – це ідентифікатор змінної, що зробить операція `&value`?

А) отримання адреси

Б) отримання значення

В) збереження змінної у пам'яті комп'ютера

Г) збереження адреси у пам'яті комп'ютера

9) У якому випадку значення покажчика `p1` буде збільшено на 1?

```
char * p1; int * p2;
```

А) `p2=p1++;`

Б) `p2++; p1=p2;`

В) `p2=p1+1;`

Г) `p1=p1+p2;`

10) Наведіть практичний приклад визначення різниці двох покажчиків.

А) з'ясування, на якій відстані один від одного знаходяться елементи масиву

Б) з'ясування, чи належать обидва покажчики до однакового типу

В) з'ясування, яка з двох змінних, на які вказують покажчики, має більше значення

Г) немає правильної відповіді

11) Яка з операції над покажчиками є забороненою?

А) множення двох покажчиків

Б) додавання числа до покажчика

В) віднімання числа з покажчика

Г) порівняння двох покажчиків

12) Що з наступного не є назвою функції для динамічного виділення пам’яті у мові С?

А) `malloc()`

Б) `calloc()`

В) `new()`

Г) `realloc()`

13) Чим відрізняються функції `malloc()` та `calloc()` ?

А) за допомогою `calloc()` можна виділити невеликі об'єми пам'яті

Б) функції мають різні параметри

В) за допомогою `malloc()` можна виділити невеликі об'єми пам'яті

Г) уся виділена за допомогою `malloc()` пам'ять ініціалізується нулями

14) Що не дозволяє виконати функція `realloc()` ?

А) виділити новий блок, який більш ніж старий

Б) виділити новий блок, який менше ніж старий

В) виділити новий блок, використовуючи нульовий покажчик

Г) скопіювати дані в динамічно розподіляєму область пам'яті

15) Для чого призначена функція `free()` ?

А) для обнуління пам'яті

Б) для звільнення пам'яті

В) для тестування пам'яті

Г) немає правильної відповіді

РОЗДІЛ 9 ДИНАМІЧНІ МАСИВИ

9.1 Динамічні одновимірні масиви

У багатьох ситуаціях неможливо наперед знати розмір масиву. Розміри статичних змінних, у тому числі і масивів, визначаються на етапі компіляції, вони не можуть змінюватися на етапі виконання програми. Крім того, статичні перемінні віддаляються з пам'яті компілятором, тобто одержати доступ в область пам'яті – стек, де зберігаються статичні перемінні, вручну не можна.

Для того щоб розмір масиву можна було визначити на етапі виконання, а масив після того, як у ньому відпала необхідність, можна було вручну, тобто з програми, видалити з оперативної пам'яті використовуються, так звані, динамічні змінні.

Для створення одновимірного динамічного масиву спочатку необхідно створити покажчик такого типу, змінні котрого будуть зберігатися в масиві, а потім присвоїти цьому показчику значення, повернене однією з функцій для виділення динамічної пам'яті. Слід також звернути увагу, що функціям `malloc()` і `realloc()` необхідно передавати розмір в байтах пам'яті, яку слід виділити. Для обчислення розміру пам'яті слід помножити кількість елементів масива на розмір який займає 1 елемент. У свою чергу, щоб дізнатися який обсяг пам'яті займає змінна деякого типу, можна скористатися оператором `sizeof` (наприклад `sizeof (int)`).

Приклад. Ввести динамічний масив елементів цілого типу. Обчислити середнє арифметичне елементів масиву. Після використання масив видалити.

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>

void main ()
{
     int *data;
     int size, sum=0;
     printf("Введіть розмір масиву\n");
     scanf("%d", &size);
     data = (int*) malloc(size*sizeof(int)); //розміщення масиву
```

```
	printf("Введіть елементи масиву\n");


	for (int j=0; j<size; ++j)
	{
		scanf("%d", &data[j]);
		sum+=data[j];
	}
	printf("Середнє дорівнює %f", (float) sum/size);
	free(data);
	system("pause");
}
```

Приклад. Задана статична матриця цілих чисел розміром N x M. Сформувати динамічний одновимірний масив з парних елементів цієї матриці.

```
#include <stdio.h>
#include <malloc.h>
int main()
{
	const int N=3, M=4;
	int matr[N][M],i,j,k=0,size=0;
	for (i=0;i<N;i++)
		for (j=0;j<M;j++)
		{
			printf("Input the element [%d][%d]: ",i+1,j+1);
			scanf("%d",&matr[i][j]);
			if (matr[i][j]%2==0) size++;
		}

	printf("Your have inputed the matrix:\n");
	for (i=0;i<N;i++)
	{
		for (j=0;j<M;j++)
			printf("%5d",matr[i][j]);
		printf("\n");
```

```
    }
    printf("with %d even elements\nThe new array is:\n",size);
    int* mas=(int*)malloc(size*sizeof(int));
    if (!mas) {printf("Error!");
                return 1;
               }
    for (i=0;i<N;i++)
         for (j=0;j<M;j++)
              if (matr[i][j]%2==0)
              {
                   mas[k]=matr[i][j];
                   printf("%5d",mas[k++]);
              }

    free(mas);
    return 0;
}
```

9.2 Динамічні двовимірні масиви

У мові С багатовимірні динамічні масиви реалізуються за допомогою множинних покажчиків.

Для створення двовимірного масиву необхідно створити подвійний покажчик. Далі слід виділити пам'ять під масив покажчиків (рядків), а потім виділити пам'ять для кожного рядка окремо.

Приклад. Задано динамічну матрицю цілих чисел розмірності n x m (n і m вводяться користувачем у ході роботи програми). Упорядкувати рядки за зростанням числа нульових елементів для кожного рядка.

```
#include <stdio.h>
#include <malloc.h>

void main()
{
    double **a;
    int *b;
```

```
    int n, m, i, j, k, temp;

    printf("Введіть число рядків матриці\n");
    scanf("%d", &n);
    printf("Введіть число стовпців матриці\n");
    scanf("%d", &m);

    a = (double**) malloc(n*sizeof(double)); // виділення місця
// під масив покажчиків на рядки
    for (i=0; i<n; i++)
      a[i]=(double*)malloc(m*sizeof(double));//виділення місця
// безпосередньо під елементи массиву
      b=(int*)malloc(n*sizeof(int)); // виділення місця для
   // накопичення кількісті нульових елементів у кожному рядку
    for (i=0; i<n; i++)
    {
         b[i]=0;
         for (j=0; j<m; j++)
         {
              scanf("%lf", &a[i][j]);
              if (a[i][j] == 0)
              b[i]++; // накопичення кількісті
                    // нульових елементів
         }
    }

    for (i=0;i<n-1;i++)// сортування методом простого обміну
    {
         for (k=n-1; k>0; k--)
         if (b[k]<b[k-1])
         {
              temp = b[k]; // перестановка елементів
              b[k] = b[k-1]; // у масиві b
              b[k-1] = temp;
              for (j=0; j<m; j++)
              {
```

```
                    temp = a[k][j];  // перестановка рядків
                    a[k][j] = a[k-1][j]; // у матриці a
                    a[k-1][j] = temp;
               }
          }
     }

     printf("\nSorted matrix:\n");
     for (i=0;i<n;i++)
{ // вивід матриці
          for (j=0;j<m;j++)
               printf("%lf ", a[i][j]);
          printf("\n");
     }

     // видалення масиву
     for (i=0; i<n; i++)
          free(a[i]);    //знищення кожного рядку окремо
//за допомогою видалення посилання на нього
     free(a);            //знищення масиву покажчиків
}
```

9.3 Витік пам'яті

Слід також звернути увагу на те, що після використання масивів, виділену пам'ять потрібно звільняти. У невеликих програмах, які ми розглядаємо зараз це може не мати значення, проте в досить великих проектах зневага звільненням більше не використовуваної пам'яті може призвести до того, що програма може займати набагато більше пам'яті, ніж їй насправді потрібно, або навіть всю доступну пам'ять. Тому завжди звільняйте виділену пам'ять, якщо не збираєтеся використовувати її далі в програмі.

9.4 Контрольні питання

1. Які масиви називають динамічними? На якому етапі визначається розмір динамічного масиву?

2. Оголошення динамічних одновимірних масивів. Наведіть приклади оголошень.

3. Видалення одновимірного масиву. В якому місці програми можна робити видалення масиву?

4. Створення багатовимірних динамічних масивів.

5. Наведіть приклад створення двовимірного динамічного масиву.

6. Як треба видаляти багатовимірний динамічний масив?

7. Проблеми витіку пам'яті.

9.5 Тести для самоконтролю

1) Яка головна відмінність динамічного масиву від статичного?

А) заздалегідь не відомий розмір масиву

Б) швидше виділяється пам'ять

В) більш зручний при передачі у функцію як параметра

Г) швидше обчислюються операції над елементами масиву

2) Виберіть правильне оголошення динамічного масиву в С:

А) `char * mass = new char[10];`

Б) `char * array = (char*) calloc (10);`

В) `int * mass = (int) malloc (10);`

Г) `int * mass = (int*) malloc (10);`

3) Що є елементом динамічного багатовимірного масиву?

А) масив елементів довільного типу

Б) масив елементів того ж типу, що й масив

В) окремий одиничний елемент будь-якого типу

Г) залежить від конкретного випадку

4) Яке оголошення є вірним для двовимірної матриці в С?

А) `char** array[][] = (char*) malloc(sizeof(int)*10);`

Б) `char** array[][] = (char**) malloc(sizeof(char)*10);`

В) `float** array[][] = (float**) calloc(sizeof(char)*10);`

Г) `int* array[][] = (int*) malloc(sizeof(int)*10);`

5) Що потрібно виправити в запису: `char* array[]=("Hello world");` ?

А) дописати ще []

Б) замінити "" на "

В) замінити круглі дужки на фігурні

Г) додати ще одну *

6) Як у програмі виділити пам’ять для роботи з двовимірним масивом?

А) виділити пам'ять під масив рядків, а потім виділити пам'ять для кожного рядка окремо

Б) виділити пам'ять для кожного рядка окремо, а потім виділити пам'ять під масив рядків

В) виділити пам'ять для кожного рядка

Г) немає правильної відповіді

7) Як у програмі звільнити пам’ять після роботи з двовимірним масивом?

А) знищити масив покажчиків

Б) знищити кожний рядок за допомогою видалення покажчика на нього

В) знищити масив покажчиків, а потім знищити кожний рядок за допомогою видалення покажчика на нього

Г) знищити кожний рядок за допомогою видалення покажчика на нього, а потім знищити масив покажчиків

8) Для чого потрібно звільнення виділеної пам’яті після використання динамічних масивів?

А) звільнену пам’ять можна використовувати для інших дій

Б) не можна виділяти пам’ять для наступного масиву, доки не звільнена пам’ять, що зайнята попереднім масивом

В) не можна оголошувати покажчик на наступний масив, доки не звільнена пам’ять, що зайнята попереднім масивом

Г) немає правильної відповіді

РОЗДІЛ 10 РОБОТА З РЯДКАМИ

10.1 Загальні положення

Рядки, називані також строковими літералами, утворюють спеціальну конструкцію, яка використовується для роботи з послідовностями символів.

Рядок – це послідовність символів, укладена у лапки.

Для збереження рядків компілятор використовує по одному байту на кожен символ рядка і автоматично додає до неї ознаку кінця рядка, якою служить нульовий символ ‘\0’. Таким чином, для збереження рядка з N символів завжди віделяется N+1 байт пам'яті. Такі рядки часто називають ASCIIZ-рядками, оскільки вони являють собою послідовність ASCII-символів, що завершується нульовим символом, який служить обмежником рядка. Наприклад, для рядка "Тест" потрібно 5 байт пам'яті, а для рядка "" – 1 байт.

10.2 Опис рядків

Мова С не має убудованого рядкового типу. У програмі рядок можна розглядати як одномірний масив символів чи використовувати спеціальні рядкові функції, які присутні в будь-якій мові.

У мові С рядок є масивом символів, що закінчується нульовим символом '\0'. Доступ до рядка здійснюється через покажчик, що посилається на перший символ рядка. Значенням рядка є адреса її першого символу. Рядок може бути привласнений в оголошенні або масиву символів, або змінній типу char*.

Кожне з оголошень

```
char color []=”blue”; const char *colorPtr=”blue”;
```

ініціалізує змінну рядком “blue”. Перше оголошення створює масив color, що складається з 5 елементів і містить символи: 'b', 'l', 'u', 'e', '\0'. Друге оголошення створює змінна-покажчик colorPtr, що вказує на рядок “blue”, що знаходиться в якомусь місці пам'яті.

Стандарт обмежує довжину рядка 509 символами. Однак фірма Microsoft пішла своїм шляхом і тому в Visual C++ можна використовувати рядки довжиною до 2048 символів.

С дозволяє створювати масиви рядків. Для створення масиву рядків використовується двовимірний масив символів. Лівий індекс визначає число

рядків, а правий індекс – максимальне число символів у кожнім рядку. Даний фрагмент коду повідомляє масив з 24-х рядків, причому кожна з них може містити до 79 символів включно:

```
char str_ar [24] [80];
```

Для того, щоб одержати доступ до елементів масиву рядків, необхідно визначити лівий індекс. Щоб ввести рядок в оперативну пам'ять, необхідно указати

```
gets(str_ar [2] ),
```

в даному випадку як параметр передається третій рядок масиву `str_ar.`

Приклад. Використання масиву рядків як найпростішого текстового редактора.

```
#include <stdio.h>
#include <conio.h>
#define MAX 24
#define LEN 80
char text [MAX][LEN];

void main()
{
     int t, i, j;
     printf("\nInput array of the strings\n");
     for(t=0; t<MAX; ++t)
     {
          printf("the str %d: ", t);
          gets(text[t]);
          if (!*text[t])
               break; // вихід по порожньому рядку
     }
          // посимвольный вивод тексту
     for(i=1; i<MAX; i++)
     {
          for(j=0; text[i][j]; j++)
               printf("%c", text[i][j]);
          printf(" \n");
     }
```

```
}
```

Дана програма здійснює введення тексту, поки не зустрінеться порожній рядок. Потім вона відображає кожен рядок. З метою ілюстрації вона виводить текст посимвольно з використанням першого індексу.

10.3 Базові функції для роботи з рядками

Для застосування рядкових функцій треба підключити заголовний файл `string.h` за допомогою директиви препроцесора

```
#include <string.h>
```

Наведемо деякі функції для роботи з рядками в мові С:

`int getchar(void)` – уводить наступний символ зі стандартного пристрою введення і повертає його у форматі цілого.

`char *gets(char *s)` – уводить символи зі стандартного пристрою введення в масив `s` доки не зустріне символ нового рядка чи індикатор кінця файлу.

`int putchar(int c)` – друк символу, що зберігається в `c`.

`int puts(const char *s)` – друк рядка s з наступним символом нового рядка.

`char *strcpy(char *s1,const char *s2)` – копіює рядок `s2` у рядок `s1`.

`char *strncpy(char *s1,const char *s2,int maxlen)` – копіює перші maxlen символів рядка `s2` у рядок `s1`.

`char *strcat(char *s1,const char *s2)` – об'єднання рядків `s1` і `s2`.

`char *strncat(char *s1,const char *s2, int maxlen)` – приєднує перші maxlen символів рядка `s2` до рядка `s1`.

`int strcmp(const char *s1,const char *s2)` – порівняння рядків `s1` і `s2`.

`int stricmp(const char *s1,const char *s2)` – порівняння рядків `s1` і `s2` без різниці між буквами верхнього і нижнього регістрів.

`int strncmp(const char *s1,const char *s2,int maxlen)` – порівняння перших `maxlen` символів рядків `s1` і `s2`.

`size_t strlen (const char *s)` – визначає довжину рядка `s`.

`char *strchr(const char *s, int c)` – знаходить позицію першого входження символу `c` у рядок `s`.

`char *strrchr(const char *s, int c)` – знаходить позицію останнього входження символу `c` у рядок `s`.

`size_t strcspn(const char *s1, const char *s2)` – визначає і повертає довжину початкового сегмента рядка `s1`, що містить тільки ті символи, що не входять у рядок `s2`.

`char *strstr(const char *s1, const char *s2)` – знаходить позицію першого входження рядка `s2` у рядок `s1`.

`char *strset(char *s,int ch)` – заміняє усі символи у рядку на заданий символ.

`char *strnset(char *s,int ch,int n)` – заміняє перші `n` символів у рядку на заданий символ.

`char *strrev(char *s)` – реверсирує рядок (крім нульового кінцевого символу)

Приклад. Застосування функції `strcmp`.

```
#include <stdio.h>
#include <string.h>

void main()
{
     char *s1="1264",*s2="12345";
     int res;
     res=strcmp(s1,s2);
     if (res<0)
          printf("s1<s2");
     else if (res>0)
printf("s1>s2"); // коди символів: ‘6’>’3’ ?
                                   //результат s1>s2
          else
               printf("s1=s2");
}
```

Приклад. Застосування функції `strchr`.

```
#include <stdio.h>
#include <string.h>

void main()
{
```

```
     char *s="abcde", *res;
     int c='b';// c=98 – ASCII-код символу ‘b’
     res = strchr(s, c); // res=”bcde”
     printf("%s", res);
}
```

Приклад. Застосування функцій `strcat` і `strncat`.

```
#include <stdio.h>
#include <string.h>

void main()
{
     char *s1="123",*s2="456",*s3="abcde";
     int n=3;
     strcat(s1,s2);
     printf("\n%s", s1); // 123456
     strncat(s1, s3, n);
     printf("\n%s", s1);      // 123456abc
}
```

Приклад. Застосування функції `strstr`

```
#include <stdio.h>
#include <string.h>

void main()
{
     char *str1="Visual C++",*str2="sual", *ptr;
     ptr = strstr(str1,str2);
     printf("The substring is: %s", ptr); // sual C++
}
```

Приклад. Застосування функцій `strlen` і `strrev`.

```
#include <stdio.h>
#include <string.h>

void main()
```

```
{
     char *s="12345";
     int res, i;
     res=strlen(s);           //res=5
     printf("\n");
     for (i=res-1; i>=0; i--)
          printf("%c", s[i]);  //посимвольний вивод рядка
     strrev(s);
     printf("\n%s", s);       //вивод рядка
}
```

Приклад. Замінити у S1 кожний n-й символ на символ s, де n – кількість входжень s у S2.

```
#include <stdio.h>
#include <string.h>

void main()
{
     char s1[100], s2[100], *ptr;
     int n=0;

     //введення
     puts("Input string 1:");
     gets(s1);
     puts("Input string 2:");
     gets(s2);


     //пошук кількості входжень 's' в s2
     ptr=strchr(s2, 's');
     while(ptr != NULL)
{
          n++;
          ptr = strchr(ptr+1, 's');
     }
     printf("\nNumber of 's' occurrences in string 2: %d\n", n);
```

```
     puts("\nString 1 before character replacement:");
     puts(s1);

     //заміна кожного n-го символа в s1
     for(int i=n-1; i<strlen(s1); i+=n)
          s1[i] = 's';

     //вивід резульaту
     puts("String 1 after character replacement:");
     puts(s1);

     system(“pause”);
}
```

Приклад. Задані рядки символів `str` і `str_2`. Виконати наступні дії:

1) видалити з рядка `str` всі входження заданого підрядка `str1`;

2) вставити в рядок `str_2` задану підрядок `str2`, починаючи з вказаною позиції;

3) виконати циклічну прокрутку символів рядка `str3` зліва направо;

4) замінити в рядку `str` всі символи `'a'` - на `'b'`, `'b'` - на `'c'`, `'c' - 'd'`.

```
#include <stdio.h>
#include <string.h>
#include <locale.h>
#include <malloc.h>

void main()
{
     setlocale(LC_ALL,"rus");
     char *str, *str1="fg", *str2="xyz", temp[20]="", *ptr,
*str3="12345", c;
     int i,k;
     printf("Введите количество символов для строки str");
     int size;
     scanf("%d",&size);
     str=(char*) malloc(size*sizeof(char));
```

```
    printf("Початковий рядок: ");
    scanf("%s",str);//ввід початкового рядка

    //завдання 1
    printf("\nПідрядок для видалення    : ");
    printf("%s",str1);//виведення підрядка для видалення
    while (ptr=strstr(str,str1))
    {  //пошук входження підрядка в рядок
    strcpy(temp,"");
    k=ptr-str; //скільки символов від початку рядка
               //до цього підрядка
    strcpy(temp,str);
    temp[k]='\0';//частину рядка до входження підрядка
                  //зберігаємо в temp
    ptr=ptr+strlen(str1);  //ptr вказує на залишок рядка
              //після проходження підрядка, що видаляється
    strcat(temp,ptr);// цей залишок склеюємо з temp
    strcpy(str,temp);
    }
    printf("\nРезультат: ");
    printf("%s",str);// вивід зміненого рядка

    //завдання 2
    printf("Початковий рядок: ");
    scanf("%s",str);// введення початкового рядка
    printf("\nПідрядок для вставки: ");
    printf("%s",str2); //вивід рядка для вставки
    printf("\nЗадайте номер символу, після якого потрібно вставити
підрядок");
    scanf("%d",&k);
    strcpy(temp,str);
    temp[k]='\0';
    str=str+k;// зміщаємо покажчик на k позицій
    strcat(temp,str2); // приєднуємо до temp підрядок str2,
                       // що вставляється
    strcat(temp,str);// приєднуємо до temp залишок рядка str
```

```
    strcpy(str,temp);
    printf("%s",str);// вивід зміненого рядка

    //завдання 3
    printf("\nРядок для прокрутки:");
    printf("%s",str3);
    strcpy(temp,"");
    temp[0]=str3[strlen(str3)-1];// зберігаємо останній символ
                                 // рядка str
    temp[1]='\0';
    strncat(temp,str3,strlen(str3)-1);
    strncat(temp,"\0",1);
    printf("\nРезультат: ");
    printf("%s",temp);// вивід зміненого рядка

    //завдання 4
    printf("\nПочатковий рядок: ");
    printf("%s",str);// введення початкового рядка
    strcpy(temp,str);
    for (i=0;i<strlen(str);i++)
        if (temp[i]>='a' && temp[i]<='c')
            temp[i]++; // коди нових символів відрізняються
                       //від старих на 1
    printf("\nРезультат: ");
    printf("%s",temp);// вивід зміненого рядка
}
```

Буває, що використання рядкових функцій не є ефективним при розв'язку певного завдання. Тоді рядок розглядається як одновимірний масив елементів типу char.

Приклад. Дано рядки S1 та S2. Вивести кількість приголосних з S1, які зустрічаються в S2.

```
#include <stdio.h>
#include <string.h>
```

```
void main()
{
     char S1[20]="lkjh76re*)M&^nyyytt", S2[6]="knt*6",
sogl[20]="wrtpsdfghjklzxcvbnm";
     int i,j,counter=0,l1,l2,l3,f,f1;
     l1=strlen(S1);
     l2=strlen(S2);
     l3=strlen(sogl);
     for(i=0;i<l1;i++)
     {
          f=0;
          for(j=0;j<l3;j++)
               if(S1[i]==sogl[j])
               { f=1;
                    break;
               }
          if(f==1)
          {
               f1=0;
               for(j=0;j<l2;j++)
                    if(S1[i]==S2[j])
                    {
                         f1=1;
                         counter++;
                         break;
                    }
          }
     }
     printf("Number of consonants characters=%d\n",counter);
}
```

Приклад. Дано рядки `S1` та `S2`. Дописати у початок `S1` символи, які є присутніми в обох рядках.

```
#include <stdio.h>
#include <string.h>
```

```
void main()
{
     char S1[15], S2[5],temp[20]="";
     printf("Input 1 string=");
     gets(S1);// зчитування рядка
     printf("Input 2 string=");
     gets(S2);
     int l1=strlen(S1),l2=strlen(S2),i,j,c=0;
                         // strlen – визначення довжини рядка
     for(i=0;i<l1;i++)
          for(j=0;j<l2;j++)
               if(S1[i]==S2[j])
               {    printf("%c ",S1[i]);
                    temp[c]=S1[i];
                    c++;
              }
     strcat(temp,S1);// склеювання (temp=temp+S1)
     puts(temp);// виведення рядка
}
```

10.4 Контрольні питання

1. Що таке рядок символів? Чим закінчується у пам’яті рядок? Скільки байтів потрібно для зберігання рядка, що містить максимум 20 символів?
2. Чи існує у мові С спеціальний рядковий тип? За допомогою якої конструкції мови С можна розглядати рядок?
3. Чи можна створювати масиви рядків? Як це зробити?
4. Як можна використовувати функцію `gets()` для роботи з рядками та рядками рядків?
5. Заголовний файл якої бібліотеки треба підключити для роботи зі спеціальними рядковими функціями?
6. За допомогою яких функцій можна ввести з клавіатури одиночний символ та рядок? Наведіть приклади.
7. За допомогою яких функцій можна вивести на екран одиночний символ та рядок? Наведіть приклади.

8. За допомогою яких функцій можна копіювати один рядок в другий цілком або частково? Наведіть приклади.

9. За допомогою яких функцій виконується об'єднання одного рядка з іншим цілком або частково? Наведіть приклади.

10. За допомогою яких функцій виконується порівняння одного рядка з іншим цілком або частково? Для яких ситуацій це важливо? Наведіть приклади.

11. Як визначити довжину рядка? Для чого це потрібно? Як визначити довжину рядка без використання спеціальної функції? Наведіть приклади.

12. За допомогою яких функцій можна знайте перше входження в рядок символа або іншого рядка? Що ці функції повертають у якості результату? Як знайти подальші входження в рядок символа або іншого рядка? Наведіть приклади.

13. Як можна реверсувати рядок? Що при цьому відбувається з кінцевим символом ‘\0’?

10.5 Тести для самоконтролю

1) У що потрібно укласти послідовність символів для того, щоб вона стала рядком у С?

А) у одинарні лапки

Б) у подвійні лапки

В) у круглі дужки

Г) у квадратні дужки

2) Що є ознакою кінця рядка?

А) символ `‘\n’`

Б) символ `‘\t’`

В) символ `‘\0’`

Г) символ пробілу

3) Як у С представляється рядок?

А) як елемент типу `char`

Б) як множина елементів типу `char`

В) не можна ніяк представити

Г) як одновимірний масив елементів типу `char`

4) Як у С представляється масив рядків?

А) як двовимірний масив символів

Б) як одновимірний масив символів

В) як двовимірний масив цілих елементів

Г) немає правильної відповіді

5) Який заголовний файл містить описи функцій для роботи з рядками?

А) `string.h`

Б) `rows.h`

В) `string_func.h`

Г) `stdio.h`

6) Які переваги функції `gets()` перед функцією `scanf()` ?

А) дозволяє вводити рядки більшої довжини

Б) дозволяє вводити рядки, що містять прописні та рядкові букви

В) дозволяє вводити рядки з пробілами

Г) немає правильної відповіді

7) Чим відрізняється функція `strncpy` від функції `strcpy`?

А) копіює останні `n` символів одного рядка в інший

Б) копіює перші `n` символів одного рядка в інший

В) застосовується в більш новій версії Visual Studio

Г) виконується швидше

8) Чим відрізняється функція `stricmp` від функції `strcmp`?

А) порівнює перші `n` символів одного рядка з іншим

Б) порівнює рядки без різниці між буквами верхнього і нижнього регістрів

В) порівнює останні `n` символів одного рядка з іншим

Г) використовує для роботи іншу бібліотеку

9) Що робить функція `strlen`?

А) визначає довжину рядка з урахуванням кінцевого символу

Б) визначає максимальний індекс у рядку

В) визначає кількість прописних букв у рядку

Г) визначає довжину рядка без урахування кінцевого символу

10) Чи можна використовувати функцію `strstr` замість функції `strrchr`?

А) так, якщо привести символ, який шукається, до вигляду односимвольного рядка

Б) так, у будь-якому випадку

В) ні, ніколи

Г) так, якщо символ, який шукається, є константою

РОЗДІЛ 11 СТРУКТУРИ І ОБ'ЄДНАННЯ

11.1 Загальні положення

Структурою називається спеціальний тип даних, що є досить гнучким для представлення різноманітних даних. Крім того, він дозволяє програмісту створювати нові типи.

Структура – набір змінних різних типів, що утворюють єдиний об'єкт. З використанням структур зручно описувати об'єкти реальнго світу, що мають властивості різних типів (цілі, дійсні, масиви й ін.). Елементами структури можуть бути дані будь-якого типу, включаючи і інші структури, а також покажчик на ту ж саму структуру.

11.2 Описи структур. Звертання до полів. Масиви структур

Структури в мові С – це похідні типи даних, вони створюються з об'єктів інших типів.

Перш ніж зі структурою можна буде працювати, необхідно описати її, для чого необхідно визначити так називаний структурний шаблон:

```
struct struct_tag
{
     type1 item1;
     type2 item2;
     ...
     typeN itemN;
};
```

Ключове слово `struct` визначає, що все, що стоїть за ним є структурою. Далі йде ім'я типу структури `struct_tag`, називаної також тегом. Воно ідентифікує структуру для того, щоб можна було створювати перемінні цього типу. Саме "зміст" структури (її елементи) укладено у фігурні дужки.

Однак шаблон у представленому виді не створює змінної. Для її визначення необхідно записати

```
struct struct_type structVariable;
```

У цьому описі `struct struct_type` грає ту ж роль, що `int` чи `float` у відповідних описах.

Ще одним способом створення змінної структурного типу є сполучення опису шаблона і визначення змінних:

```
struct struct_tag
{
     type1 item1;
     type2 item2;
     …
     typeN itemN;
} structVariable1, structVariable2;
```

Якщо не передбачається створення додаткових змінних, то ім'я типу структури `struct_tag` можна опустити:

```
struct
{
     type1 item1;
     type2 item2;
     …
     typeN itemN;
} structVariable1, structVariable2;
```

Можливе створення і покажчика на структуру:

```
struct struct_tag *ptrStruct;
```

При визначенні змінної компілятор виділяє пам'ять для кожного елемента структури, що поєднується під ім'ям цієї змінної:

```
struct accept
{
char * parol[6];
char * login[5];
};
accept user1, users[10];
```

Тут `accept` є теговим ім'ям, чи ім'ям типу. Змінні, оголошені усередині фігурних дужок, є елементами структури. Елементи однієї структури повинні мати унікальні імена, але дві різні структури можуть містити однакові імена, і це не викликає ніяких конфліктів. Створено одиночний об'єкт `user1` і 10 об'єктів (масив) `users` типу `accept`. Того ж результату можна було б досягнути іншими способами:

```
struct accept
```

```
{
     char* parol;
     char* login;
} user1,users[10];
struct
{
     char* parol;
     char* login;
} user1,users[10];
```

Як і інші типи даних, структури можна ініціалізувати у місці визначення. Наприклад,

```
struct accept user1 = {“Admin”, “1234”};
```

Звертатися до елементів структури можна, вказавши ім'я структурної змінної, ім'я елемента і розділивши ці імена операцією . (крапка).

```
printf(“%s”, user1.parol);
printf(“%s”, user1.login);
```

Крім того, дуже часто використовуються покажчики на структури, тому для доступу до її полів була введена коротка форма запису. Якщо p – покажчик на структуру, то p -> <поле структури> дозволяє звернутися до зазначеного полю структурної змінної. Наприклад,

```
struct Struct_type
{
     int a,b;
};
Struct_type * ptr;
ptr->a=5; ptr->b=2*ptr->a;
```

Покажчики на структуру зазвичай використовуються в наступних випадках:

- доступ до структур, розміщених у динамічній пам'яті;
- створення складних структур даних – списків, дерев;
- передача структур в якості параметрів у функції.

Інформація, що міститься в одній структурі, може бути привласнена іншій структурі того ж типу за допомогою одиночного оператора присвоювання, тобто не потрібно привласнювати значення кожного поля по окремості. Наприклад,

```
users[0] = user1;
```

Приклад. Створити дві структури типу "книга", про кожну з яких відомі такі дані: назва, автор, рік видання. Вивести дані про книгу, у якої найбільш пізній рік видання.

```
#include <stdio.h>

void main()
{
     struct book
     {
          char name[30];
          char author[25];
          int year;
     };
     book b1,b2;
     printf("\nInput information for book 1: ");
     scanf("%s", b1.name);
     scanf("%s", b1.author);
     scanf("%d", b1.year);
     printf("\nInput information for book 1: ");
     scanf("%s", b2.name);
     scanf("%s", b2.author);
     scanf("%d", b2.year);
     if (b1.year>b2.year)
     printf("%s %s %d", b1.name, b1.author, b1.year);
     else
          if (b2.year>b1.year)
               printf("%s %s %d", b2.name, b2.author, b2.year);
     else
          printf("Is equal");
}
```

Приклад. Завдання аналогічно попередньому, тільки використовується статичний масив елементів типа «книга».

```
#include <stdio.h>

void main()
{
```

```
     struct book
     {
          char name[30];
          char author[25];
          int year;
     };

     const int n=4;
     book A[n];
     int i, max=0, j;
     for (i=0; i<n; i++)
     {
          printf("\nInput information for book #%d", i+1);
          scanf("%s", A[i].name);
          scanf("%s", A[i].author);
          scanf("%d", A[i].year);
          if (A[i].year>max)
          {
               max=A[i].year;
               j=i;
          }
     }

     printf("%s %s %d", A[j].name, A[j].author, A[j].year);
}
```

11.3 Складні структури

Полем структури може бути інша структура (вкладена). У цьому випадку спочатку повинна бути описана вкладена структура, а потім основна:

```
struct struct_tag_inside
{
     type1 item1_inside;
     type2 item2_inside;
     ...
     typeN itemN_inside;
```

```
};
struct struct_tag
{
     type1 item1;
     type2 item2;
     ...
     struct_tag_inside item_inside
     ...
     typeN itemN;
} element;
```

У цьому випадку для доступу до поля item1_inside структури item_inside необхідно вказати:

```
element. item_inside. item1_inside
```

Приклад. Задати дані про банківського службовця, про якого відомо наступне: ПІБ, рік народження, домашня адреса, зарплата по місяцях за 12 місяців. Визначити середню зарплату службовця за рік. Використовувати тип даних – структура. Для ПІБ та домашньої адреси використовувати вкладені структури. Вивести інформацію про банківського службовця.

```
#include <stdio.h>

void main()
{
     struct FIO_type
     {
          char f[20],i[15],o[17]; //значення у дужках задаються
                                  // в залежності від вимог
     };
     struct ADDR_type
     {
          int index;
          char city[10],street[20];
          int building,flat;
     };

     struct WORKER_type
```

```
     {
          FIO_type fio;
          int year;
          ADDR_type addr;
          float salary[12];
     };

     WORKER_type worker;
     int j;
     float sum=0;
     printf("Input FIO: ");
     scanf("%s%s%s",worker.fio.f,worker.fio.i,worker.fio.o);
     printf("Input year: ");
     scanf("%d",&worker.year);
     printf("Input home address: index, city, street, building,
flat");

scanf("%d%s%s%d%d",&worker.addr.index,worker.addr.city,worker.addr.st
reet,&worker.addr.building,&worker.addr.flat);
     printf("Input salary for 12 months: ");
     for (j=0;j<12;j++)
     {
          scanf("%f",&worker.salary[j]);
          sum+=worker.salary[j];
     }

     printf("The worker %s %s %s have average salary =
%.2f",worker.fio.f,worker.fio.i,worker.fio.o,sum/12);

}
```

11.4 Об'єднання

Об'єднання – структурований тип даних. Синтаксис об'єднання ідентичний синтаксису оголошення структури, тільки замість ключового слова struct указується union.

Розходження полягає в тому, що кожному елементу об'єднання виділяється та ж сама область пам'яті. Розмір пам'яті, виділюваної під об'єднання, дорівнює максимальному числу байтів, необхідних для збереження самого довгого поля.

Приклад. Оголошення типу об'єднання.

```
union u1
{
     int I;
     double d;
};
```

Об'єднання дозволяють створювати масиви чи файли з елементами різного типу.

Приклад.

```
…
struct circle
{
int x,y,r;
};
struct rect
{
int x1, y1, x2, y2;
};
union prim
{
circle c; rect r;
};
struct geom
{
char type; prim x;
};
geom. mgeom[20];
…
mgeom[0].type=’c’;
mgeom[0].x.c.x=10;
…
mgeom[0].type=’r’;
```

```
mgeom[0].x.r.x1=100;
…
// ініціалізація графічного режима
for (int i=0;i<20;i++)
{
if (mgeom[i].type==’c’) circle (…);
if (mgeom[i].type==’r’) rectangle (…);
}
…
```

11.5 Контрольні питання

1. Який тип даних називається структурою? Чим він відрізняється від типів, що були відомі раніше?
2. Як визначити структурний шаблон? Що означає ключове слово struct?
3. Як визначити змінну або масив структурного типу? Наведіть приклади з використанням: тегового імені без змінних, змінних без тегового имені, змінних з теговим ім’ям.
4. Чи може структура містити одне поле, декілька полів, декілька однотипних полів, жодного поля?
5. Скільки виділяється пам’яті для зберігання змінної структурного типу?
6. Як можна ініціалізувати структури у місці визначення?
7. Як звертатися до елементів структури?
8. Чи можна поле однієї структури привласнити полю іншої структури? У якому разі? Як це зробити?
9. Чи можна цілком привласнити одну структуру іншій? У якому разі? Як це зробити?
10. Чи може структура бути полем іншої структури? Як це зробити? Як звертатися до елементів таких структур?
11. Чи може бути одновимірний чи багатовимірний масив полем іншої структури? Наведіть приклади.
12. Чи можна створювати масиви зі структур? Наведіть приклади.
13. Що таке об'єднання? Чим воно відрізняється від структури? Як оголосити об'єднання?

14. Чи можна з елементів типу об'єднання створювати масиви чи файли?

11.6 Тести для самоконтролю

1) Що можна назвати структурою?

А) групу елементів одного типу

Б) групу елементів, які займають одну і ту ж саму область в пам'яті

В) сукупність змінних, об'єднаних під одним ім'ям

Г) немає правильної відповіді

2) Яке слово є ключовим при оголошенні структур?

А) `struct`

Б) `stract`

В) `structure`

Г) немає ключового слова

3) Який обов'язковий символ потрібно ставити після опису структури?

А) ;

Б) ~

В) :

Г) /

4) Який оператор чи символ треба вказати для доступу до поля структури після вказання імені структури?

А) :

Б) =

В) ,

Г) .

5) Чи можна формувати масиви структур?

А) можна тільки статичні

Б) можна

В) можна тільки динамічні

Г) не можна

6) Який оператор чи символ треба вказати для доступу до поля структури після вказання покажчика на структуру?

А) .

Б) &

В) ->

Г) ^

7) Чи може одна структура бути привласнена іншій?

А) може, якщо співпадають типи

Б) не може

В) може, якщо структури не містять масивів

Г) може, якщо структури мають по одному полю

8) Що не може бути вкладено в структуру?

А) ціла змінна

Б) структура того ж типу

В) масив дійсних елементів

Г) покажчик на структуру того ж типу

9) Що треба зробити, щоб включити одну структуру у якості поля для іншій?

А) це не можна робити

Б) підключити заголовний файл `<struct.h>`

В) визначити зовнішню структуру, а потім вкладену

Г) визначити вкладену структуру, а потім зовнішню

10) У чому відмінність `union` від `struct`?

А) використовуються різні типи доступу до елементів

Б) може зберегати менше елементів

В) елементи можуть займати одну і ту ж саму область в пам'яті

Г) немає можливості динамічного виділення пам'яті

11) Чи можуть бути структури та об’єднання вкладені один в одного?

А) ніколи не можуть

Б) можуть

В) можна вкласти тільки структури в об’єднання

Г) можна вкласти тільки об’єднання в структури

РОЗДІЛ 12 ФУНКЦІЇ

12.1 Загальні положення: підпрограми

Підпрограми являють собою відносно самостійні фрагменти програми, оформлені особливим чином і постачені ім'ям. Згадування цього імені в тексті програми називається викликом підпрограми. Підпрограми являють собою інструмент, за допомогою якого будь-яка програма може бути розбита на ряд певною мірою незалежних одна від одної частин. Така розбивка необхідна з двох причин.

По-перше, цей засіб економії пам'яті: кожна підпрограма існує в програмі в єдиному екземплярі, тоді як звертатися до неї можна багаторазово з різних точок програми. При виклику підпрограми активізується послідовність утворюючих її операторів, а за допомогою переданих підпрограмі параметрів потрібним чином модифікується реалізований у неї алгоритм.

Друга причина полягає в застосуванні методики спадного проектування програм. У цьому випадку алгоритм представляється у вигляді послідовності щодо великих підпрограм, що реалізують більш-менш самостійні значущі частини алгоритму.

Підпрограми у свою чергу можуть розбиватися на менш великі підпрограми нижнього рівня і т.д. Послідовне структурування програми продовжується доки реалізовані підпрограмами алгоритми не стануть настільки простими, щоб їх можна було легко запрограмувати.

Виклик підпрограми здійснюється простим вказуванням імені підпрограми в операторі виклику підпрограми чи імені підпрограми у виразі. Як відомо, будь-яке ім'я в програмі має бути обов'язково описано перед тим, як воно з'явиться серед операторів, що виконуються. Щодо підпрограм це також не є винятком.

Для мови програмування С у ролі підпрограм виступають тільки функції. Функція – самостійна одиниця програми, спроектована для реалізації конкретної задачі. Поняття функції є основою програмування на С.

12.2 Опис функцій. Методика виділення функцій

Визначення функції має такий вид:

```
тип_ значення_що_повертається  ім'я_функції ( список_параметрів )
```

```
{
оператори
}
```

Формально схему визначення функції можна представити таким чином:

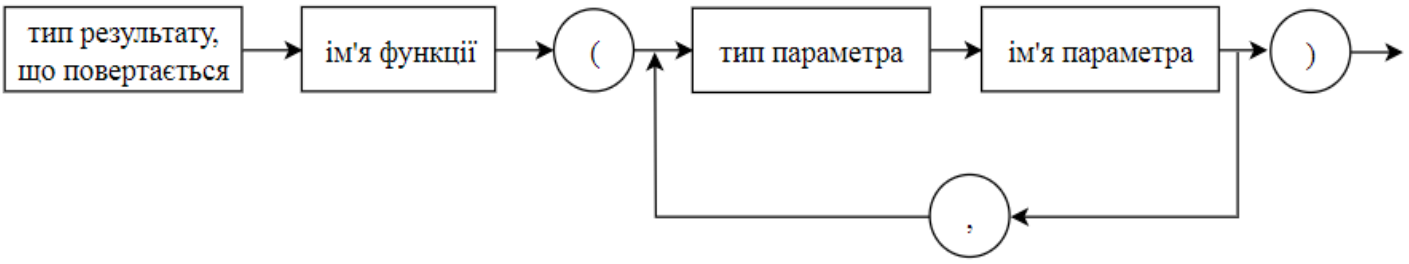


Рисунок 12.1 – Формальне представлення визначення функції

При створенні функції слід виконати два кроки: створити оголошення функції і визначення (називане також представленням) функції. Оголошення робить можливим доступ до функції – поміщає її в область видимості. У мові С усі функції глобальні. Обмежити область видимості функції одним файлом можна, указавши перед нею ключове слово static.

Як ім'я функції може використовуватися будь-який припустимий ідентифікатор. Перед ім'ям функції вказується тип результату, що повертається. Якщо тип заданий ключовим словом void, то це означає, що функція нічого не повертає. Якщо тип значення, що повертається, не зазначений, то компілятор буде припускати тип int.

Після імені функції в круглих дужках перелічуються всі її аргументи з указівкою типу кожного. Аргументи можуть бути відсутніми. Завершується оголошення функції крапкою з комою. Нижче приведені приклади декількох оголошень функцій:

```
long factorial (int number);
int swap (float * a, float * b);
void makeItem (int amount, char * name, float cost);
struct fruit insert();
```

Функція може повертати тільки одне значення. Якщо необхідно одержати більше оброблених функцією значень, то це можна зробити тільки через покажчики, передані як аргументи.

Після того як функція оголошена, її необхідно визначити, тобто записати код, що виконує покладену на функцію задачу.

У визначенні функції спочатку йде заголовок функції, тобто рядок, аналогічний оголошенню, за винятком того, що завершальна крапка з комою заміняється на фігурні дужки, у які укладене тіло функції. Тіло функції містить у собі дві "частини": область оголошення (визначення) локальних даних і область коду, що містить конструкції мови С.

Приклад програми з функцією, що визначає максимальне з трьох чисел.

```
#include <stdio.h>

int maximum (int, int, int); // прототип (оголошення) функції

void main()
{
     int a,b,c;
     printf("Enter three integers: ");
     scanf("%d%d%d", &a, &b, &c);
     printf("Maximum is: %d", maximum(a,b,c));
}

// Визначення функції maximum
int maximum (int x, int y, int z) {
     int max = x;
     if (y > max) max = y;
     if (z > max);
     return max;
}
```

У попередньому прикладі аргумент у функцію передавався за значенням. Коли змінна передається у функцію таким чином, то, насправді, функція одержує копію змінної. Техніка передачі змінної у функцію за значенням обмежена у використанні тією обставиною, що функція може повернути тільки одне значення.

Виклик за посиланням організований таким чином, що функція одержує адресу аргументу. Цей підхід вимагає менше пам'яті, чим виклик за значенням. Однак, коли ви використовуєте виклик за посиланням, змінна в програмі може бути фактично змінена. Крім того, функцією може бути повернуте більш ніж одне

значення. Для того, щоб передати функції аргумент за посиланням перед ідентифікатором параметра треба вказати оператор одержання адреси &.

Приклад. Увести з клавіатури число і перевернути в ньому цифри навпаки.

```
#include <stdio.h>

void invers(int &x);

void main()
{
     int number;
     scanf("%d", &number);
     invers(number);
     printf("\n%d", number);
}

void invers(int &x)
{
     int y=0, n;
     n = x%10;
     while (n!=0)
{
          y = y*10+n;
          x = (x-n)/10;
          n = x%10;
     }
     x = y;
}
```

У цьому прикладі функція `invers` приймає ціле число як аргумент і перевертає цифри в цьому числі навпаки. Число змінюється і виводиться з функції main. Це можливо завдяки тому, що передача аргументу здійснюється за посиланням.

Розглянемо іншу реалізацію цієї задачі.

```
#include <stdio.h>
void invers(int x);
```

```
void main()
{
     int old_number;
     scanf("%d", &old_number);
     invers(old_number);
}
void invers(int x)
{
     int y=0, n;
     n = x%10;
     while (n!=0)
{
          y = y*10+n;
          x = (x-n)/10;
          n = x%10;
     }
     printf("%d", x);
}
```

У цьому прикладі зміни, що виробляються над копією аргументу, ніяк не видні в головній функції `main`, тому вивод результату необхідно здійснювати з функції `invers`.

Якщо тип, що вертається функцією значення не `void`, то для повернення значення використовується оператор `return`. Цей же оператор здійснює вихід з функції в код, що викликав функцію. Тіло функції може містити декілька операторів `return`.

Приклад.

```
#include <stdio.h>
#include <math.h>

float root(float a);

void main()
{
     int x;
     scanf("%d", &x);
     if (root(x)==-1)
```

```
          printf("Кореня немає");
     else
          printf("%f", root(x));
}

float root(float a)
{
     if (a<0)
          return -1;
     else
          return sqrt(a);
}
```

Якщо функція повертає відмінне від void значення, вона може використовуватися в арифметичних і логічних вираженнях. Значення, що повертається нею, може бути введене на екран чи привласнено сумісної по типу змінної.

Наприклад, якщо є такий прототип функції

```
float func(int a,float * b,char * s);
```

то вона може бути використана таким чином:

```
float res = func(a1, b1, s1);
```

чи

```
if (func(a2, b2, s2) == 3.5)
printf(”This is a test”);
```

чи

```
printf(“%f”, func(a3,b3,s3));
```

Якщо при роботі підпрограми виникла необхідність завершити програму цілком, треба застосовувати функцію exit:

```
     void exit (int code);
```

яка потребує підключення заголовного файлу <stdlib.h>. Аргумент code вказує статус завершення роботи. При завершенні роботи в штатному порядку рекомендується вказувати значення статусу 0 або EXIT_SUCCESS. В іншому випадку рекомендується вказувати відмінне від нуля значення або EXIT_FAILURE.

У мові С всі змінні і функції повинні бути оголошені до їх використання. Оголошення змінної встановлює відповідність імені та атрибутів змінної, викликає виділення пам'яті для зберігання її значення. Місце розміщення змінної в

пам'яті програми задається за допомогою класу пам'яті (класу зберігання) змінної. Залежно від місця розміщення змінної в пам'яті визначається час життя і область видимості змінної.

Клас пам'яті задається специфікатором класу пам'яті. У мові С є 4 класу пам'яті:

- `Extern` (зовнішні змінні);
- `Auto` (автоматичні змінні);
- `Register` (реєстрові змінні);
- `Static` (статичні змінні).

Правила видимості мови – це правила, що керують тим, що бачить частина програми. Кожна функція мови С – це самостійний блок коду. Код функції є власністю функції, і до нього не можна одержати доступ за допомогою якого-небудь чи оператора іншої функції, крім виклику даної функції. Код, що утворює тіло функції, захований від іншої частини програми, іншими словами код і дані, визначені в одній функції, не можуть впливати на код і дані, визначені в іншій функції, оскільки дані функції мають різні області видимості.

Змінні, визначені у функціях, називаються локальними змінними. Локальні змінні створюються при вході у функцію і знищуються при виході з неї. Тому ці змінні не можуть містити значення між викликами функцій. Єдиним винятком з цього правила є змінні, оголошеним викликом static. Він змушує компілятор сприймати дану змінну як глобальну, але область видимості як і раніше обмежена функцією.

Час життя – це інтервал часу виконання програми, протягом якого програмний об'єкт (змінна або функція) існує, що визначається фактом виділення пам'яті. Час життя змінної може бути локальним або глобальним. Змінна з глобальним часом життя має виділену пам'ять і певне значення протягом усього часу виконання програми, починаючи з моменту оголошення цієї змінної. Змінна з локальним часом життя має виділену пам'ять і певне значення тільки під час виконання блоку, в якому ця змінна оголошена. При кожному вході в блок локальній змінній виділяється нова пам'ять, яка звільняється при виході з блоку. Область видимості – це частина тексту програми, в якій може бути використаний даний об'єкт. Об'єкт вважається видимим в блоці або в початковому файлі, якщо в цьому блоці або файлі відомі ім'я і тип об'єкта.

Об'єкт може бути видимим в межах блоку, поточного файлу або у всіх файлах, що утворюють програму. Час життя і область видимості об'єкта залежать від того, в якому місці оголошений об'єкт. Якщо об'єкт оголошений на внутрішньому рівні, тобто усередині деякого блоку, то за замовчуванням він має локальний час життя і локальну область видимості, тобто він існує, поки виконується блок, і його бачать в цьому блоці й у всіх внутрішніх блоках (локальна змінна). Якщо об'єкт оголошений на зовнішньому рівні, тобто поза всіх блоків, він має глобальний час життя і глобальну видимість. Такий об'єкт існує до завершення програми і його бачать від точки його оголошення до кінця даного вихідного файлу (глобальна змінна).

Змінити час життя змінної і область її видимості можна, якщо вказати для неї клас пам'яті. Всі змінні, які використовувалися до цих пір, були відомі тільки функціям в яких знаходилися.

У С є можливості зробити ряд змінних відомими відразу для кількох функцій і швидко обробити змінні шляхом переміщення їх значень в регістри і т. д. Ці та інші можливості реалізуються за допомогою привласнення змінним класу пам'яті. Службове слово, яке визначає клас пам'яті, ставиться в описах змінних перед службовим словом, що задає тип змінних:

```
static int x,result;
extern char c;
register int number;
```

Усі функції в С знаходяться на одному рівні видимості, тобто неможливо визначити функцію у функції.

Якщо функція використовує аргументи, вона повинна визначати змінні, що одержують значення аргументів. Ці змінні називаються формальними параметрами функції. Вони поводяться так само як і звичайні локальні змінні, тобто створюються при вході у функцію і знищуються при виході з неї.

Таким чином, в заголовку та в прототипі функції указуються її формальні параметри, які при виклику функції заміщаються фактичними. Тому типи даних формальних і фактичних параметрів повинні збігатись. Імена фактичних і формальних параметрів збігатись не зобов’язані, тому що описується функція один раз, а бути викликувана може бути декілька разів.

12.3 Передача масивів як аргументів

У мові С для передачі масиву як параметра укажіть його ім'я без усяких дужок. Наприклад, у фрагменті програми

```
int hourlyTemp[24];
modifyArray (hourlyTemp, 24)
```

оператор виклику функції передає масив і його розмір у функцію. Мова С автоматично передає масив у функцію шляхом імітації передачі параметра за посиланням – при цьому викликувані функції можуть змінювати значення елементів у вихідних масивах викликаючих функцій. Для того, щоб функція при звертанні до неї могла прийняти масив, у списку параметрів цієї функції повинно бути зазначено, що очікується передача масиву, наприклад:

```
void modifyArray (int b[], int size)
```

Прототип функції може виглядати так:

```
void modifyArray (int [], int)
```

Другий засіб прототипу функції з масивом-аргументом:

```
void modifyArray (int * b, int size)
```

Приклад. За допомогою функції обчислити добуток елементів одномірного масиву.

```
#include <stdio.h>

const int n=5;

int prod(int *br); // прототип функції

void main()
{
     int i, pr;
     int b[n];
     for (i=0; i<n; i++)
          scanf("%d", &b[i]);
     pr = prod(b); // виклик функції
     printf("\nproduct=", pr);
}
// опис функції
```

```
int prod(int *br)
{
     int i, p;
     p=1;
     for (i=0; i<n; i++)
          p *= br[i];
     return p;
}
```

При передачі параметрів-масивів С не обмежуються масивами базових типів. Можна визначити свій власний тип (наприклад, структуру) і масив з елементами цього типу передавати у функцію.

Приклад. Задана множина точок на площині. Визначити відстань між найбільш віддаленими точками.

```
#include <stdio.h>
#include <math.h>

const int n=3;

struct point
{
     int x,y;
};

float max(point * a);

void main()
{
     int i;
     point p[n];
     for (i=0;i<n;i++)
          scanf("%d%d", &p[i].x, &p[i].y);
     printf("\Max rasst = %f", max(p));
}

float max(point*a)
```

```
{
     float m=0;
     int i,j;
     for (i=0; i<n; i++)
          for (j=0; j<n; j++)
               if (i!=j && sqrt(pow(a[j].x-a[i].x, 2)
  + pow(a[j].y-a[i].y, 2))>m)
               {
                    m = sqrt(pow(a[j].x-a[i].x, 2)
    + pow(a[j].y-a[i].y, 2));
               }
     return m;
}
```

Інший варіант рішення цієї ж задачі:

```
#include <stdio.h>
#include <math.h>

const int n=3;

struct point
{
     int x,y;
};

float max(point s, point v);
void main() {
     float m=0,temp;
     int i,j;
     point p[n];
     for (i=0; i<n; i++)
          scanf("%d%d", &p[i].x, &p[i].y);
     for (i=0; i<n; i++)
          for (j=0; j<n; j++)
               if (i!=j)
               {
```

```
                    temp = max(p[i],p[j]);
                    if (temp>m)
                         m = temp;
               }
     printf("\nMaximum length = %f", m);
}

float max(point s,point v)
{
     return sqrt(pow(s.x-v.x,2) + pow(s.y-v.y,2));
}
```

Коли двовимірний масив використовується як аргумент функції, передається покажчик на перший елемент. Функція, що одержує двовимірний масив, повинна, як мінімум, визначати розмір першого виміру, оскільки компілятору необхідно знати довжину кожного рядка для коректної індексації масиву.

Приклад. З використанням функції, що приймає як аргумент двовимірний масив цілих чисел, сформувати одномірний масив з максимальних елементів рядків.

```
#include <stdio.h>

const int n = 4;
void f(int a[n][n], int b[n]);

void main()
{
     int i, j;
     int a[n][n], b[n];
     printf("Input array:");
     for (i=0; i<n; i++)
          for (j=0; j<n;j++)
               scanf("%d", &a[i][j]);
     f(a,b);
     printf("\nYour array\n");
     for (i=0; i<n; i++)
```

```
          printf("%d\t", b[i]);
}

void f(int a[n][n],int b[n])
{
     int i, j;
     for (i=0; i<n; i++)
     {
          b[i] =a [i][0];
          for (j=1; j<n; j++)
               if(a[i][j] > b[i])
                    b[i] = a[i][j];
     }
}
```

12.4 Рекомендації з використання функції у програмі

При створенні програми дуже важливим є рішення про те, які саме дії повинна виконувати функція. З одного боку, вона не повинна бути дуже «маленькою» по обсягу дій, тому що тоді доведеться створювати чимало функцій для реалізації якогось нескладного завдання. З іншого боку, вона не повинна самостійно виконувати усе завдання цілком, тому що у цьому разі її не можна буде використовувати для рішення аналогічного, але трохи іншого завдання.

Розглянемо чотири приклади, які виконують одне і те ж завдання, але у кожному прикладі для функції відводяться різні дії.

Приклад. У одновимірному масиві розміром N підрахувати кількість елементів, що дорівнюють введеному користувачем значенню k. Варіант без функцій.

```
#include <stdio.h>

void main()
{
     const int N=10;
     int mas[N],i,k,count=0;
```

```
     printf("Input k: ");
     scanf("%d",&k);

     printf("Input %d elements: ",N);
     for (i=0;i<N;i++)
     {
          scanf("%d",&mas[i]);
          if (mas[i]==k) count++;
     }
     printf("Result=%d\n",count);
}
```

Приклад. У одновимірному масиві розміром N підрахувати кількість елементів, що дорівнюють введеному користувачем значенню k. Варіант з функцією – усю работу виконує функція.

```
#include <stdio.h>

void func();
void main()
{
     func();
}

void func()
{
const int N=10;
     int mas[N],i,k,count=0;

     printf("Input k: ");
     scanf("%d",&k);

     printf("Input %d elements: ",N);
     for (i=0;i<N;i++)
     {
          scanf("%d",&mas[i]);
          if (mas[i]==k) count++;
```

```
    }
    printf("Result=%d\n",count);
}
```

Приклад. У одновимірному масиві розміром N підрахувати кількість елементів, що дорівнюють введеному користувачем значенню k. Варіант з функцією – функції передається масив і значення k, функція обчислює і виводить на екран результат.

```
#include <stdio.h>

void func(int*,int,int);

void main()
{
    const int N=10;
    int mas[N],i,k;
    printf("Input k: ");
    scanf("%d",&k);

    printf("Input %d elements: ",N);
    for (i=0;i<N;i++)
        scanf("%d",&mas[i]);

    func(mas,N,k);
}

void func(int* mas,int N,int k)
{
    int i;
    int count=0; //локальна змінна
    for (i=0;i<N;i++)
        if (mas[i]==k) count++;
    printf("Result=%d\n",count);
}
```

Приклад. У одновимірному масиві розміром N підрахувати кількість елементів, що дорівнюють введеному користувачем значенню k. Варіант з функцією – функції передається масив і значення k, функція обчислює результат і повертає його у головну програму, яка виводить результат на екран. Найбільш універсальний варіант!

```
#include <stdio.h>

int func(int*,int,int);

void main()
{
     const int N=10;
     int mas[N],i,k,res;
     printf("Input k: ");
     scanf("%d",&k);

     printf("Input %d elements: ",N);
     for (i=0;i<N;i++)
          scanf("%d",&mas[i]);

     res=func(mas,N,k);

     printf("Result=%d\n",res);
}

int func(int* mas,int N,int k)
{
     int i;
     int count=0;
     for (i=0;i<N;i++)
          if (mas[i]==k) count++;
     return count;
}
```

12.5 Параметри за замовчуванням

Існують задачи, які вимогають різної кількості параметрів залежно від деяких обставин. У цьому випадку можна надати ряду параметрів функції початкових значень, які залишаються при виклику функції чи змінюються фактичними параметрами. Список формальних параметрів такої функції має такий вигляд:

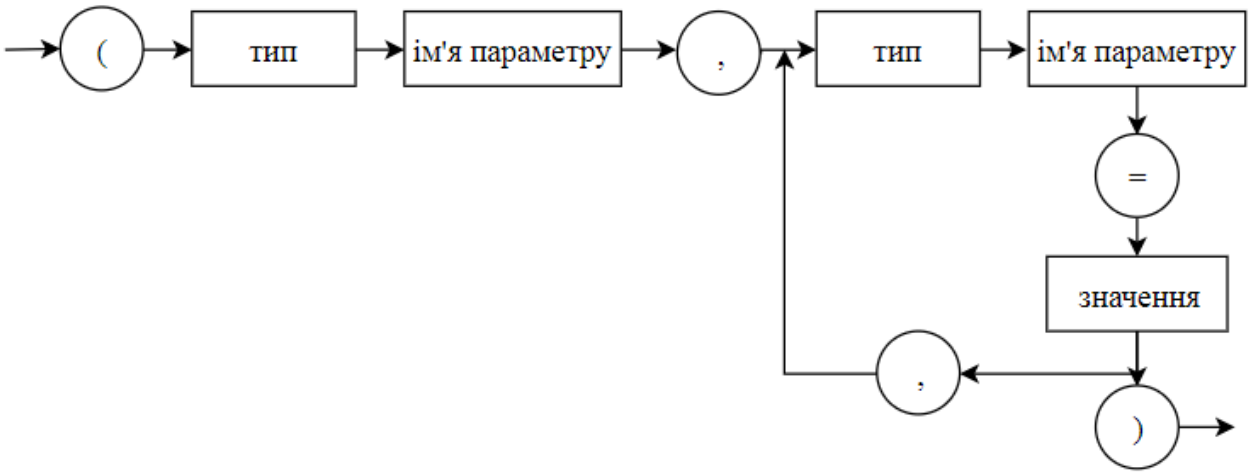


Рисунок 12.3 – Загальне представлення формальних параметрів функції

Список повинен мати принаймні один параметр без значення за замовчуванням (перший параметр).

Приклад. Створити функцію яка відшукує найбільше число серед 2, 3, 4 або 5 дійсних чисел.

```
#include <stdio.h>

float max( float x1, float x2, float x3=-3.4e+38,
float x4=-3.4e+38, float x5=-3.4e+38)
{
     float xm;
     if(x1 > x2)
          xm = x1;
     else
          xm = x2;
```

```
     if(x3 > xm)
          xm = x3;
     if(x4 > xm)
          xm = x4;
     if(x5 > xm)
          xm = x5;
     return xm;
}

void main()
{
     double a, b, c, d, e;
     printf("\n");
     scanf("%d%d%d%d%d", &a, &b, &c, &d, &e);
     printf("\n max(a,b): %f", max(a, b));
     printf("\n max(a,b,c): %f", max(a, b, c));
     printf("\n max(a,b,c,d): %f", max(a, b, c, d));
     printf("\n max(a,b,c,d,e): %f", max(a, b, c, d, e));
}
```

12.6 Покажчики на функції

Покажчики на функції можна використовувати як аргументи у разі виклика іншої функції, зберігати у масиві, застосовувати у операторі-виразі.

Приклад. Створити програму, в який можна надавати покажчику функції значення певної функції в операторі-виразі.

```
#include<stdio.h>
#include<string.h>
int (*funct)(const char*, const char*);
void main( )
{
     char *s1="abs", *s2="abcd";
     funct = strcmp;
     int i = (*funct)(s1,s2);
     printf("\ni = ", i);
}
```

Приклад. Скласти програму для обчислення одного дійсного кореня методом половинного ділення для двох функцій `f1(x)` та `f2(x)`.

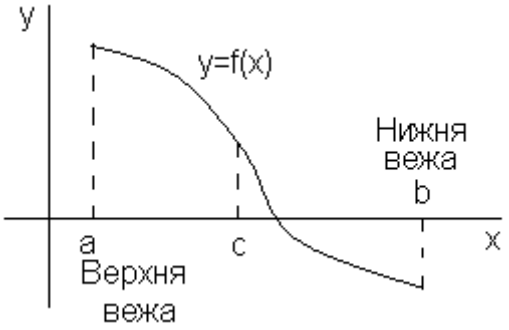


Рисунок 12.4 – Метод поділу навпіл

Зміст алгоритму полягає в тому, що треба ділити відрізок на дві рівні частини та визначати, де змінюється знак функції. Якщо знак змінюється зліва, то процес продовжується зліва і навпаки. Процес завершується якщо досягнута задана точність обчислення. Проблема у тому, що функція обчислення кореня має працювати з різними функціями певного виду, які будуть прийматися як фактичні параметри.

```
#include<stdio.h>
#include<math.h>
typedef float(*pf)(float);
float root(pf, float, float, float);
float f1(float);
float f2(float);

void main()
{
     float a, b, eps;
     printf("\n a, b, eps:");
     scanf("%d%d%d", &a, &b, &eps);
     printf("\n x1 = ", root(f1, a, b, eps));
     printf("\n x2 = ", root(f2, a, b, eps));
}

float root(pf f, float a1, float b1, float pogr)
{
```

```
     float c;
     do
     {
          c=(a1+b1)/2;
          if(f(a1)*f(c)>0)
               a1=c;
          else
               b1=c;
     }
     while(fabs(f)>=pogr);
     return c;
}

float f1(float x)
{    return 4-x*x; }

float f2(float x)
{    return 9-x*x; }
```

12.7 Рекурсивні функції

Рекурсивна функція – це функція, що викликає саму себе чи безпосередньо, чи побічно через іншу функцію. Розглянемо класичний приклад – обчислення факторіала.

Програма вводить із клавіатури ціле число n і виводить на екран значення n!, що обчислюється за допомогою рекурсивної функції FACT. Фрагмент програми:

```
#include <stdio.h>
#include <stdlib.h>
long factorial (long x);
void main()
{
     long result, number;
     printf("Enter an integer: ");
     scanf("%li", &number);
```

```
    result = factorial(number);
    printf("!%li = %li\n", number, result);
    system("pause");
}

//рекурсивне визначення функції factorial
long factorial (long x)
{
    if (x == 0) return 1;
    else return x*factorial(x-1);
}
```

12.8 Контрольні питання

1. Що називається підпрограмою? Які є підпрограми у мові програмування С?
2. Для чого рекомендується використовувати підпрограми? Як за допомогою підпрограм економлять пам’ять? Як застосування підпрограм відповідає методиці спадного проектування програм?
3. Як у програмі викликати підпрограму? Які у мові С існують правила завдання імен функцій?
4. Як указати виклик функції на схемі алгоритму? Як на схемі алгоритму показати реалізацію цієї функції?
5. Як у програмі створити оголошення функції та її визначення? Наведіть два варіанти: визначення власної функції до функції main(), визначення після функції main().
6. Що повертає функція як результат своєї роботи? Що можна робити з цим результатом? Скільки значень може повернути функція? Як використовується оператор `return`? Скільки операторів return може мати функція у своєї реалізації?
7. Аргументи функції: для чого використовуються, є обов’язковими чи ні, як вказується тип аргументу?
8. Аргументи функції: передача за значенням, передача за посиланням. Поясніть відмінності кожного з видів передач аргументів. Як за допомогою аргументів функція може повернути більш ніж одне значення?

9. Локальні та глобальні змінні: місце визначення, правила видимості, час життя. Чи буде у програмі конфлікт, якщо глобальна та локальна змінні мають однакові імена? Якщо ні, як програма буде з ними працювати?

10. Що таке формальні та фактичні параметри? Де вони використовуються? Які вимоги накладаються на формальні та фактичні параметри функції?

11. Як передавати масив у якості аргумента в функцію? Чи змінюються фактично значення елементів масиву, якщо їх змінює функція? Наведіть приклади.

12. Чи можна передавати в функцію структуру чи масив структур? Наведіть приклади.

13. Що називають параметрами за замовчуванням? У якому разі їх доцільно використовувати? Наведіть приклади.

14. Наведіть формальну схему для списку формальних параметрів за замовчуванням. Скільки у цьому списку повинно бути параметрів без значення за замовчуванням?

15. Що називають покажчиками на функції? Як їх можна використовувати? Наведіть приклади.

16. Які функції називають рекурсивними? Чи мають вони відмінний від інших функцій синтаксис? Наведіть приклади використання рекурсивних функцій.

12.9 Тести для самоконтролю

1) Що називається функцією?

А) сукупність операторів мови

Б) сукупність змінних програми

В) самостійний фрагмент програми

Г) весь вміст файлу .cpp

2) Що не можна розглядати як перевагу при використанні функцій?

А) економію пам'яті

Б) простоту реалізації алгоритмів

В) застосування методики спадного проектування програм

Г) організацію бібліотек функцій

3) Як можна викликати підпрограму?

А) вказати ім’я підпрограми у виразі у будь-якому разі

Б) вказати ім’я підпрограми у виразі, якщо тип значення, що повертає підпрограма, відповідає типу операнда у виразі

В) вказати ім’я підпрограми у функції printf у будь-якому разі

Г) немає правильної відповіді

4) Що робить прототип функції?

А) оголошує функцію

Б) дає команду на запуск функції

В) виділяє пам'ять для функції

Г) описує функцію

5) Де в коді прописуються прототипи?

А) в довільному місці

Б) тільки на початку програми

В) в будь-якому місці до опису самої функції та функції `main()`

Г) в будь-якому місці до функції `main()`

6) Яке призначення оператору `return`?

А) призупинення виконання функції

Б) завершення програми

В) звільнення невикористаної пам’яті

Г) вихід та/або повернення певного значення з функції

7) Де застосовуються формальні параметри?

А) при виклику функції

Б) при описі функції та у прототипі

В) тільки у прототипі

Г) немає правильної відповіді

8) Де застосовуються фактичні параметри?

А) таких параметрів немає

Б) при описі реалізації функції

В) в заголовку прототипа функції

Г) при виклику функції

9) Як може функція повернути більш ніж одне значення?

А) через параметри, що передаються за посиланням

Б) використовуючи декілька операторів `return`

В) ніяк не може

Г) немає правильної відповіді

10) Який вказується тип значення, що повертається, для функції, яка нічого не повертає?

А) 0

Б) `NULL`

В) `void`

Г) не вказується ніякий тип

11) Як передається массив у якості аргументу функції?

А) за значенням

Б) за посиланням

В) передавати не можна

Г) немає правильної відповіді

12) Чи можна в функцію передавати багатовимірний масив?

А) не можна

Б) можна, якщо тип елементів є базовим

В) можна

Г) можна тільки двовимірний

13) Що називається параметрами за замовчуванням?

А) перші три параметри, які передаються в функцію

Б) усі фактичні параметри

В) усі формальні параметри

Г) параметри, що мають початкові значення, які можна залишити чи змінити

14) Яка функція називається рекурсивною?

А) така, що викладає сама себе

Б) така, що не має прототипу

В) така, що нічого не повертає

Г) така, що не викливається з функції `main()`

15) Яка функція застосовується для успішного завершення програми?

А) `return 0;`

Б) `exit(0);`

В) `return NULL;`

Г) `exit(1);`

РОЗДІЛ 13 ПОСИЛАННЯ

13.1 Загальні положення

Посилання у мові програмування С дозволяє створити псевдонім (або друге ім'я) для ваших змінних у программі. По суті, посилання – це неявний покажчик. Воно використовується для передачі параметрів функції, для повернення значення функції і в якості відносних змінних. Далі розглянемо кожен з цих варіантів.

Для оголошення посилання всередині програми вкажіть знак амперсанда (&) безпосередньо після типу параметра. Оголошуючи посилання, ви повинні відразу ж привласнити змінну, для якої це посилання буде псевдонімом:

```
int & someRef = someInt;
```

Оголошене таким чином посилання можна використовувати в якості відносної змінної, і всі дії виконуються з ним будуть, по-суті, виконуватися з адресатом. Важливо відзначити, що посилання не можна перепризначити. Намагаючись зробити це, ви всього лише привласните адресату нове значення. Також не існує нульових або пустих посилань.

13.2 Передача аргументів функції як посилань

Мова програмування С передбачає два способи передачі аргументів у функцію: за значенням і за посиланням. Якщо аргумент передається за значенням, функція отримує їх копію. Це призводить до витрат часу і пам'яті, що вимагаються на створення копії. Передача аргументу за посиланням дозволяє функції оперувати самими змінними, без створення копій. Також ця можливість дозволяє виконувати більше одного повернення з функції. Також існує два способи передачі аргументів за посиланням: покажчиком і посиланням (будьте обережні з термінологією!). Наступна програма показує механізм передачі змінних, як посилань:

```
/* програма обміну чисел*/
#include <stdio.h>
void swap(int & x, int & y);
int main()
{
     int x = 5, y = 10;
     printf ("Main. Before swap: %d %d\n", x, y);
```

```
     swap(x, y);
     printf ("Main. After swap: %d %d\n", x, y);
     return 0;
}

void swap(int & x, int & y)
{
     int temp;
     printf ("Swap. Before swap: %d %d\n", x, y);
     temp = x;
     x = y;
     y = temp;
     printf ("Swap. After swap: %d %d\n", x, y);
}
```

Проведемо паралель між покажчиками та посиланнями. Основне призначення покажчика – це організація динамічних об'єктів, тобто таких, розмір яких може змінюватися (збільшуватися чи зменшуватися). Посилання призначені для організації прямого доступу до того чи іншого об'єкту. Головна відмінність полягає у внутрішньому механізмі роботи. Покажчики посилаються на ділянку в пам'яті, використовуючи його адресу. А посилання посилаються на об'єкт за його ім’ям (теж свого роду адреса).

Якщо немає необхідності змінити передане значення посилальної змінної, але потрібно виграти у швидкості, використовуйте специфікатор `const` в оголошенні параметрів функцій. Тільки так і можна захистити дані від випадкового їх зміни або повної втрати. Наприклад,

`int sum_by_reference (const int & reference)` /* функція приймає аргумент за посиланням. Кваліфікатор `const` не дає змінити переданий аргумент усередині функції */

Приклад. Розробити три функції, аргументи в яких будуть передаватися за значенням, за посиланням та через покажчики.

```
#include <stdio.h>
int sum_by_value(int );// підсумовування за значенням
int sum_by_reference(int &);// підсумовування за посиланням
int sum_by_pointer(int *); // підсумовування за покажчиком
```

```
int main()
{
    int value = 10;
    printf("sum_by_value     = %d\n",sum_by_value(value));
    printf("value = %d\n",value);//значення змінної не змінилося
    printf("sum_by_reference = %d\n",sum_by_reference(value));
    printf("value = %d\n ",value); // значення змінної змінилося
    printf("sum_by_pointer = %d\n",sum_by_pointer(&value));
    printf("value = %d\n",value); // значення змінної змінилося
system("pause");
    return 0;
}

int sum_by_value(int value)// функція приймає аргумент за значенням
{
    value += value;
    return value;
}

int sum_by_reference(int &reference) // функція приймає аргумент за
посиланням
{
    reference += reference;
    return reference;
}

int sum_by_pointer(int *ptrvalue) // функція приймає аргумент через
покажчик
{
    *ptrvalue += *ptrvalue;// арифметика с покажчиком
    return *ptrvalue;
}
```

Результати роботи програми:

```
sum_by_value     = 20
```

```
value = 10
sum_by_reference = 20
value = 20
sum_by_pointer     = 40
value = 40
Для продовження натисніть будь-яку клавішу . . .
```

13.3 Повернення посилання

Функція може повертати посилання на об'єкт. Таким чином, виклик функції може стояти в лівій частині оператора присвоювання. Розглянемо наступний приклад:

```
/* программа обміну чисел*/
#include <stdio.h>
char & replace(int i);
char s[80] = "Hello Everyone!";
int main()
{
     replace(5) = 'X';
     printf ("%s", s);
     return 0;
}
char & replace (int i)
{    return s[i]; }
```

Програма вставляє символ X замість пропусків між словами «Hello» і «everyone», тобто виводить на екран рядок «HelloXeveryone». Для цього спочатку функція `replace()` повертає посилання на символ, що входить в рядок `s`, індекс якого заданий змінною `i`. Далі у функції `main()` за допомогою цього посилання відповідному елементу рядка присвоюється символ X.

13.4 Контрольні питання

1. Що називається посиланням? Для чого вони використовуються?

2. Як оголосити посилання? Чи існують нульові, або пусті, посилання? Наведіть приклади.

3. Які існують способи передачі аргументів за посиланням? Наведіть приклади.

4. Як функція може повертати посилання на об'єкт? Наведіть приклади.

13.5 Тести для самоконтролю

1) Що називається посиланням?

А) це особливий тип даних, що є прихованою формою покажчика

Б) це спосіб підключення заголовних файлів

В) це форма параметрів програми

Г) немає правильної відповіді

2) Який знак вказується для оголошення посилання?

А) *

Б) ->

В) &&

Г) &

3) Яке значення прийме змінна `value` після виконання наступного фрагмента програми:

```
int value = 15;
int &reference = value;
reference+=20;
```

А) 15

Б) 35

В) 20

Г) значення не визначено

4) У чому різниця в призначенні між покажчиками та посиланнями?

А) основне призначення покажчика – це організація динамічних об'єктів, розмір, яких може змінюватися, а посилання призначені для організації прямого доступу до об'єкту

Б) обидва призначені для організації прямого доступу до об'єкту

В) обидва призначені для організації динамічних об'єктів

Г) немає правильної відповіді

5) У чому полягає відмінність внутрішніх механізмів роботи у покажчиків та посилань?

А) відмінностей немає

Б) відмінності залежать від реалізації

В) покажчики посилаються на ділянку в пам'яті, використовуючи його адресу, а посилання посилаються на об'єкт за його ім’ям

Г) посилання посилаються на ділянку в пам'яті, використовуючи його адресу, а покажчики посилаються на об'єкт за його ім’ям

6) Який специфікатор забезпечує незмінність значення в посилальної змінної?

А) `static`

Б) `const`

В) `register`

Г) такого специфікатора немає

7) Якщо функція приймає аргумент за посиланням, як потрібно звертатись до нього у реалізації функції?

А) вказуючи & перед його ім’ям

Б) вказуючи * перед його ім’ям

В) звернення неможливо

Г) вказуючи його ім’я

8) Яке присвоювання є допустимим після виконання наступного фрагменту програми?

```
int i;
int * pi;
int * const cp = &i;
int ci = 7;
int * pci;
const int v=10;
ci = v;
```

```
А) ci=0.5;
Б) cp=&ci;
В) *pci=*3;
Г) i=ci;
```

РОЗДІЛ 14: ЛОКАЛІЗАЦІЯ КОНСОЛЬНИХ ПРОГРАМ

14.1 Загальні положення

При розробці консольних програм виникає проблема їх локалізації, викликана тим, що в середовищах розробки та виконання програм використовуються різні кодові сторінки з різними кодами для кириличних символів [25].

Символи з кодами 0x00 – 0x7F у всіх кодових сторінках однакові і утворюють набір символів ASCII (American Standard Code for Information Interchange – американський стандартний код для обміну інформацією). Символи з кодами 0x80 – 0xFF використовуються для кодування символів національних алфавітів. Код символу дорівнює сумі шістнадцяткових номерів рядка і стовпчика.

Консольні програми виконуються в текстових вікнах, де використовується кодова сторінка CP866. У цих сторінках для кириличних символів виділені різні діапазони кодів, тому програма, створена із застосуванням кодової сторінки 1251, буде виводити замість кириличних символів символи кодової сторінки 866, що мають ті ж коди, що й російські букви в сторінці 1251.

Однобайтові схеми кодування дозволяють працювати тільки з 256 символами, що набагато менше кількості використовуваних символів, тому були розвинені багатобайтові схеми кодування, щоб знаки могли бути представлені в 8-бітових, 16-бітових, 24-бітових, або 32-бітових послідовностях. Стандарт Unicode (Юнікод) для представлення даних задає однозначну відповідність символів кодами – елементам кодового простору, представляє невід'ємні цілі числа, які називаються кодовими точками. Наприклад, цифрі 0 відповідає кодова точка U+0030. У зразках кодових сторінок, наведених на рисунках вище, коди символів в кодуванні Юнікод вказані під знаками, наприклад, для російських букв відведений діапазон від U+0410 (заголовна А) до U+044F (мала літера я). Коди букв утворюють послідовність, зростаючу в алфавітному порядку. Вийнятки становлять букви І та І (U+406, U+0456), Ї та ї (U+457, U+407), Є та є (U+404, U+454), Ґ та ґ (U+490, U+491), які розташовані поза загального алфавітного порядку. Доступні в середовищі Windows символи і їх кодові точки можна побачити за допомогою програми «Таблиця символів" (запускається командою Пуск, Усі програми, Стандартні, Службові, Таблиця символів).

Щоб консольна програма «заговорила» українською, потрібно забезпечити перетворення кодування символів до середовища виконання. Далі розглянуті засоби обліку національних особливостей (локалізації), наявні в мові С.

14.2 Засоби локалізації мови С

Для врахування особливостей, пов'язаних з країною і мовою, використовуються спеціальні середовища, звані локальними контекстами (*locale* – місце дії), які включають набір параметрів і функцій, що забезпечують підтримку національних та культурних стандартів.

Локальний контекст визначається рядком формату:

```
мова [ _зона [ . код ]]
```

Тут мова – позначення мови (наприклад, англійська, німецька, українська), а зона – країна, в якій використовується мова. Цей кваліфікатор дозволяє підтримувати національні стандарти для різних країн, що використовують одну мову. Кваліфікатор код визначає кодову сторінку. Так, український локальний контекст визначається як uk-UA. Ці рядки не стандартизовані, тому підтримка тих чи інших локальних контекстів залежить від реалізації.

Засоби мови С для роботи з локальними контекстами оголошені в заголовку *locale*. Налаштування програми на конкретний локальний контекст виконує функція:

```
char *setlocale( int category, const char *locale );
```

Аргумент `category` визначає категорію функцій, на які `setlocale` впливає:

`LC_ALL` – всі категорії

`LC_COLLATE` – порівняння і перетворення рядків

`LC_CTYPE` – обробка символів

`LC_MONETARY` – форматування грошових сум

`LC_NUMERIC` – форматування чисел

`LC_TIME` – форматування часу

Аргумент `locale` є покажчиком на рядок, що задає ім'я локального контексту. Якщо `locale` вказує на порожній рядок, використовується локальний контекст, який використовує кодову сторінку, одержувану з операційної системи. Якщо `locale` дорівнює нулю (`NULL`), діючий локальний контекст не змінюється. При успішному завершенні роботи функція `setlocale` повертає покажчик на

статично визначений рядок з описом локального контексту або нульовий покажчик при невдачі.

Можливі кілька варіантів виклику `setlocale`, наприклад:

`setlocale (LC_ALL, "uk-UA")` – налаштування всіх функцій на Україну;

`setlocale (LC_CTYPE, "uk-UA")` – налаштування функцій обробки символів на Україну;

`setlocale (LC_CTYPE, ".1251")` – налаштування функцій обробки символів на кодову сторінку `1251`.

Приклад. Використання функції `setlocale`. Програма виводить кириличне слово, задане безпосередньо в програмі і введене з консолі (клавіатури) при локальному контексті за замовчуванням, локальному контексті з налаштуваннями з ОС і явно заданими локальними контекстами, які використовують кодові сторінки `866` і `1251`.

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

int main()
{
     char program[] = "Програма";
     char console[20];
     printf("Default locale is\t%s\n", setlocale(LC_ALL, NULL));

     printf("Enter the Ukrainian word: ");
     gets(console);
     puts(program);
     puts(console);

     printf("OS locale is\t %s \n", setlocale(LC_ALL, ""));
     puts(program);
     puts(console);

     printf("DOS locale is\t %s \n",setlocale(LC_CTYPE,".866"));
     puts(program);
     puts(console);
```

```
    printf("Locale: \t %s \n", setlocale(LC_CTYPE, ".1251"));
    puts(program);

    printf("Locale: \t %s \n", setlocale(LC_CTYPE, ".866"));
    puts(console);

    system("pause");
    return 0;
}
```

Припустимо, що при виконанні програми з консолі вводилося слово «Консоль». У середовищі Visual Studio отримуємо:

```
Default locale is        C
Enter the Ukrainian word: Консоль
╧ЁюуЁрьр
Консоль
OS locale is     Ukrainian_Ukraine.1251
Програма
?R-бR<м
DOS locale is    Ukrainian_Ukraine.866
╧ЁюуЁрьр
Консоль
Locale:          Ukrainian_Ukraine.1251
Програма
Locale:          Ukrainian_Ukraine.866
Консоль
Для продовження натисніть будь-яку клавішу . . .
```

Видно, що за замовчуванням встановлюється стандартний контекст C, при дії якого українські літери з програми не виводяться, а введені з консолі виводяться. Виклик `setlocale (LC_ALL, "")` встановлює локальний контекст з налаштуваннями ОС, де передбачено використання кодової сторінки `1251`, в якому кириличні літери з програми виводяться правильно, а введені з консолі – ні. Оскільки в консольному вікні використовується кодова сторінка `866`, для правильного виведення букв, введених з консольного вікна, потрібно змінювати локальний контекст на кодову сторінку `866` викликом

`setlocale (LC_CTYPE, ".866")`, але при цьому літери, задані в програмі, виводяться неправильно. Таким чином, при виведенні українських букв, заданих безпосередньо в програмі, варто встановлювати кодову таблицю `1251`, а перед виведенням кіриличного тексту, введеного з клавіатури, необхідно встановити кодову таблицю `866`.

14.3 Нестандартні функції налаштування консолі

Крім локалізації, можливість якої закладена в стандарті мови С, існують нестандартні засоби [26]. У Windows є функції для управління консольним вікном DOS, для доступу до яких в програму слід включити заголовний файл `windows.h`.

Функція

```
UINT GetConsoleCP();
```

витягує вхідну кодову сторінку, що використовується консоллю, асоційовану з запущеним процесом (програмою) для перетворення клавіатурного введення у відповідні символьні значення. Значення, що повертається, є номером кодової сторінки. Тип значення, що повертається `UINT` визначений у заголовному файлі `windef.h` інструкцією

```
typedef unsigned int UINT;
```

Функція

```
BOOL SetConsoleCP(UINT wCodePageID);
```

встановлює кодову сторінку, яка використовується консоллю, що асоціюється з запущеним процесом, при введенні символів з клавіатури.

Параметр `wCodePageID` є ідентифікатором встановлюваної кодової сторінки. При успішному завершенні повертається нульове значення. Якщо робота функції невдала, повертається нуль.

Функція

```
UINT GetConsoleOutputCP();
```

отримує вихідну кодову сторінку, яка використовується консоллю, що асоційована з викликаним процесом для перекладу символьних значень, записуваних різними функціями виводу в зображення символів, що показуються в консольному вікні. Повертає номер кодової сторінки.

Функція

```
BOOL SetConsoleOutputCP (UINT wCodePageID);
```

встановлює вихідну кодову сторінку, яка використовується консоллю, що асоційована з викликаним процесом. Параметр `wCodePageID`
є ідентифікатором встановлюваної кодової сторінки. При успішному завершенні повертається нульове значення.

Приклад. Налаштування консолі. Програма використовує функції для управління кодовими сторінками при введенні з консолі і при виведенні на консоль.

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main()
{
     printf("GetConsoleCP() = \t %d \n", GetConsoleCP());
     printf("GetConsoleOutputCP()=\t%d n", GetConsoleCP());

     // Кодова сторінка 1251 для вводу з консолі
     SetConsoleCP(1251);

     // Кодова сторінка 1251 для виводу з консолі
     SetConsoleOutputCP(1251);

     char name[100];
     printf("Введіть ваше ім'я: ");
     gets(name);
     printf("Вітаю, %s \n", name);

     system("pause");
     return 0;
}
```

У програмі спочатку отримуються вхідні і вихідні кодові сторінки, потім для введення і виведення встановлюється кодова сторінка `1251`, яка забезпечує коректне введення і виведення російських букв. Далі наведені результати роботи програми.

```
GetConsoleCP() =         866
```

```
GetConsoleOutputCP() =   866
Введіть ваше ім'я: Віктор
Вітаю, Віктор
Для продовження натисніть будь-яку клавішу . . .
```

Видно, що за замовчуванням для введення з консолі і виведення на консоль встановлюється кодова сторінка 866. Для узгодження середовища розробки і консолі слід встановити для консольного введення і виведення кодову сторінку 1251. При використанні функцій налаштування консолі російські літери будуть відображатися правильно, якщо для консольного вікна обраний шрифт Lucida Console. Для вибору шрифту потрібно клацнути правою кнопкою миші по заголовку вікна, виконати команду Властивості, а потім на вкладці Шрифт вибрати потрібний шрифт.

14.4 Пряме перетворення символів

Для перетворення кодування символів з кодової сторінки, що діє в середовищі розробки, в кодову сторінку, що використовується в середовищі виконання, можна вставити в програму заголовний файл windows.h і використовувати функцію:

```
BOOL CharToOem(LPCTSTR lpszSrc, LPSTR lpszDst);
```

Тут lpszSrc – покажчик на рядок у початковому вигляді, lpszDst – покажчик на буфер для перетвореного рядка.

Функція CharToOem існує у двох варіантах: для однобайтових (ANSI) і для широких символів. Якщо функція CharToOem використовується як ANSI функція, рядок може бути переведений на місці установкою для параметрів lpszDst і lpszSrc однакового значення. Однак це не може бути зроблено, якщо CharToOem використовується як функція для обробки широких символів.

Як аргументи функції CharToOem можна передавати звичайні рядки символів з нульовим байтом на кінці. Значення, що повертається функцією CharToOem, завжди НЕ нуль, за винятком випадку, коли при використанні версії функції для широких символів їй помилково передаються однакові адреси для lpszSrc і lpszDst. У цьому випадку функція повертає нуль.

Приклад. Використання `CharToOem`. Для зручності використання написана допоміжна функція `Ua`, яка повертає покажчик на рядок, перетворений в кодування `OEM`.

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

char* Ua(char* s)
{
     static char buf[500];
     CharToOem(s, buf);
     return buf;
}

int main()
{
     printf(Ua("Як Вас звати?\t"));
     char name[100];
     gets(name);
     printf(Ua("Вітаю,\t"));
     printf(name);

     system("pause");
     return 0;
}
```

Приклад роботи програми:

```
Як Вас звати?    Віктор
Вітаю,    Віктор Для продовження натисніть будь-яку клавішу . . .
```

При введенні рядка інструкцією

```
gets(name);
```

використовується кодова сторінка `866` консольного вікна, тому при зворотному виведенні на консоль рядка `name` його не потрібно перетворювати для правильного відображення букв.

Таким чином, і закладена стандартами мов С (а також С++) можливість русифікації консольних програм, і нестандартні засоби локалізації, реалізовані в

ОС Windows, забезпечують локалізацію і працюють однаково гарно. Вибір засобу локалізації залишається за програмістом, але використання функції `CharToOem` видається менш зручним, так як цю функцію доводиться викликати при виведенні кожного створеного в програмі рядка з кириличними символами. Більш зручною представляється локалізація, реалізована шляхом налагодження консолі з використанням функцій `SetConsoleCP` і `SetConsoleOutputCP`, яка була продемонстрована в прикладі вище.

14.5 Контрольні питання

1. Чим зумовлені проблеми локалізації у консольних програмах?
2. Охарактеризуйте набір символів `ASCII`.
3. Чим відрізняються кодові сторінки `Windows-1251` та `CP866`?
4. Що представляє собою стандарт `Unicode`? Як згідно з цим стандартом кодуються символи?
5. Що називається локальний контекстом? З чого він складається?
6. Опишіть заголовок `locale` та функцію `setlocale`. Як за допомогою `setlocale` встановити локалізацію?
7. Наведіть нестандартні функції налаштування консолі.
8. Як функція `CharToOem` використовується для перекодування символів?

14.6 Тести для самоконтролю

1) З чим пов’язані проблеми локалізації консольних програм?

А) з необхідністю чітко визначати пам’ять, яку вони займають

Б) з використанням різних кодових сторінок в середовищах розробки та виконання програм

В) з незнанням таблиці `ASCII` символів

Г) проблем не існує

2) Яка кодова сторінка використовується в текстових вікнах консольних програм?

А) `Windows-1251`

Б) `CP866`

В) `ASCII`

Г) ніяка не використовується

3) Що називається локальним контекстом?

А) фрагмент програми з локальними змінними

Б) папку, у якій розміщується програмний проект

В) спеціальне середовище для підтримки національних стандартів

Г) немає правильної відповіді

4) Яка функція у С налаштовує програму на конкретний локальний контекст?

А) `setlocale`

Б) `locale`

В) `context`

Г) `LC_ALL`

5) Як працюють Windows-функції `GetConsoleCP/SetConsoleCP`?

А) завершують усі консольні процеси

Б) перевіряють чи відповідають введені символи стандарту Unicode

В) управляють кодовими сторінками, що використовується консоллю

Г) немає правильної відповіді

6) Який заголовний файл потрібно підключити для застосування функцій, що управляють консольним вікном DOS?

А) `console.h`

Б) `dos.h`

В) `locale.h`

Г) `windows.h`

7) Який шрифт потрібно обрати в консолі для правильного відображення російських літер?

А) `Times New Roman`

Б) `Tahoma`

В) `Lucida Console`

Г) `Arial`

8) Як працює функція `CharToOem`?

А) перетворює кодування символів з кодової сторінки, що діє в середовищі розробки, в кодову сторінку, що використовується в середовищі виконання

Б) перетворює кодування символів з кодової сторінки, що діє в середовищі виконання, в кодову сторінку, що використовується в середовищі розробки

В) її однократне використання забезпечує русифікації усієї програми

Г) немає правильної відповіді

РОЗДІЛ 15 РОБОТА З ФАЙЛАМИ

15.1 Загальні положення

Зберігання даних у змінних і в масивах є тимчасовим: всі ці дані втрачаються при завершенні роботи програми. Для постійного зберігання великих обсягів даних використовуються файли. Комп'ютери зберігають дані на пристроях вторинної пам'яті, головним чином дискових пристроях. У цьому розділі ми пояснимо, як за допомогою написаних на С програм створювати, оновлювати і обробляти файли даних.

Файли даних – обов'язковий елемент практично будь-якої програмної системи. Вони використовуються як вихідні дані для розв’язання різних задач, а також можуть бути результатами роботи програм. Особливість цього структурованого типу даних у тому, що інформація звичайно знаходиться на дисках, а не в оперативній пам'яті. Тому розміри файлів можуть бути практично необмеженими. Оскільки доступ до дисків виконується значно повільніше ніж до оперативної пам'яті, звичайно при роботі з файлами в пам'яті створюються буфери, у які копіюється з диска або накопичується в результаті роботи програми інформація. Ця інформація при вичерпанні або заповненні буфера обновляється або записується на диск. Якщо робота з файлом даних завершується за допомогою спеціальної команди, то дані з буфера виводу записуються у файл. Якщо такої команди нема, а завершується робота програми, то буфер знищується без збереження даних у файлі.

Прикладна програма, що працює з файлами, не управляє процесом обміну даних з файлом. Після кожного читання або запису покажчик файла (адреса, за якою читається або записується інформація) переміщається на новий запис – порцію даних, котрими здійснюється обмін. Навіть із коротких зауважень про файли випливає, що це повинен бути спеціальний і досить складний тип даних.

Файлова система мови С складається з декількох взаємопов'язаних функцій. Для їх використання необхідний заголовний файл stdio.h, в якому визначені самі функції, макроси `NULL, EOF, FOPEN_MAX, SEEK_SET, SEEK_CUR SEEK_END`, а також визначено тип `FILE` [27].

У мові С будь-який файл розглядається як потік байтів. Кінцем будь-якого файлу є спеціальний символ. При відкритті файлу, йому ставиться певний потік і повертається покажчик на структуру `FILE` (так само іменують значення, що

повертається – дескриптор файлу). Тобто доступ до елементів файлу буде відбуватися за допомогою цього покажчика. Структура `FILE` призначена для зберігання атрибутів (параметрів) файлів (покажчика поточної позиції файлу, ознаки кінця файлу, прапорів індикації помилок, відомостей про буферизацію та ін.)

15.2 Основні функції для роботи з файлами

Оголошення потоку – змінної-покажчика на структуру типу `FILE`, в якій зберігатимуться атрибути файлу:

```
FILE *fl; //*fl - показчик на файл.
```

Для *відкриття файлів* використовується функція fopen, яка приймає два параметри. Перший – це ім'я файлу, другий – режим роботи з файлом, в якому він буде відкритий. Ця функція повертає покажчик на структуру `FILE` (дескриптор файлу), який необхідно присвоїти створеному нами вказівником.

Існують різні режими відкриття файлу:

1. Режим запису даних `"w"` – покажчик поточної позиції встановлюється на початок файлу, якщо зазначений у функції `fopen()` файл не існує, то він створюється. Необхідно пам'ятати, що відкриття існуючого файлу в режимі `"w"` призводить до знищення його старого змісту.
2. Режим читання даних `"r"` – можливий тільки для створеного раніше файлу, при цьому покажчик поточної позиції встановлюється на початок файлу. Якщо відкривається на читання файл не існує, функція fopen() повертає порожній покажчик зі значенням `NULL`.
3. Режим додавання даних `"a"` – покажчик поточної позиції встановлюється на кінець файлу. Дані, раніше поміщені у файл, залишаються без змін. Якщо вказується неіснуючий файл, то він створюється заново.

Також можна вказати додаткові умови режиму відкриття файлу:

`"b"` – двійковий потік;

`"t"` – текстовий потік;

`"+"` – оновлення файлу.

Приклад

```
fl = fopen("My_file1","r+b");
```

```
/* перевірка показчика на NULL,
   якщо файл не можна відкрити*/
if ((file = fopen("1.txt","wt")) == NULL)
     printf (“Error opening the file”);
```

У мові С розрізняють два способи *введення-виводу* у файл – форматований і неформатований. При форматованому введенні зчитується одна або кілька змінних певного типу, при неформатованому – зчитується задана кількість байт. Функціями неформатованого введення-виведення є `fwrite()` і `fread()`, використовувані для читання і запису відповідно. Як аргумент їм передається адреса записуваної або зчитуваної величини, розмір одного екземпляру, кількість зчитувальних величин та ім'я логічного файлу. Вони визначається як

```
int fwrite(const char * array, size_t size, size_t count, FILE *
stream);
int fread(const char * array, size_t size, size_t count, FILE *
stream);
```

Для *форматованого введення-виводу* у файл використовуються функції `fscanf()` і `fprintf()` аналогічні відповідним функціям форматованого введення-виведення, проте виконують це стосовно до файлу, відповідно їх першим параметром є покажчик на файл.

```
int  fprintf (FILE * fp, const char * theString, …);
int  fscanf (FILE * fp, const char * theString, …);
```

Також, існує набір функцій, призначених для запису і зчитування з файлу рядків і символів – функції `fgetc()` і `fputc()` дозволяють відповідно здійснити введення-виведення символу, а функції `fgets()` і `fputs()` дозволяють відповідно здійснити введення-виведення рядка. Визначаються вони так:

```
int fgetc(FILE *fp);
int fputc(int ch, FILE *fp); // ch – символ, в файл записуєтьтся
молодший байт
char * fgets(char * str, int length, FILE *fp);
int fputs(const char * str, FILE *fp);
```

У випадку помилки даними функціями повертається константа `EOF`.

`int fseek (FILE * fp, long int offset, int beginning)` – встановлює курсор на заданий байт файлу. Призначення цієї функції - підтримувати операції введення/виводу з довільним доступом.

Параметр `offset` вказує кількість байтів, на яку слід перемістити курсор файлу від точки, заданої параметром `beginning`, який задається макросами `SEEK_SET` (початок файлу), `SEEK_CUR` (поточна позиція), `SEEK_END` (кінець файлу). Повернення нульового значення свідчить про успішне виконання функції `fseek()`, а ненульового – про виникнення збою.

`int ftell (FILE * fp)` – повертає поточну позицію курсору.

`int feof (FILE * fp)` – повертає істинне значення, якщо досягнуто кінця файлу.

`int ferror (FILE * fp)` – повертає істинне значення, якщо сталася помилка.

`void rewind (FILE * fp)` – встановлює курсор на початок файлу.

`int remove (const char * fileName)` – стирає файл.

`int fflush (FILE * fp)` – очищує потік.

Для *закриття файлу* використовується функція `fclose()` – вона закриває потік, відкритий раніше функцією `fopen()`. Вона записує всі дані, що залишилися в буфері, у файл і закриває його, використовуючи команди операційної системи. Помилка, що виникла при закритті файлу, може породити безліч проблем, починаючи з втрати даних і руйнування файлів і закінчуючи непередбачуваними наслідками для програми. Крім того, функція `fclose()` звільняє керуючий блок файлу, пов'язаного з потоком, дозволяючи використовувати цей блок повторно. Операційна система обмежує кількість одночасно відкритих файлів, тому перш ніж відкрити один файл, слід закрити інший.

Прототип функції `fclose()` виглядає наступним чином:

```
int  fclose(FILE * fp)
```

Якщо функція повернула нульовий покажчик, значить, файл відкритий успішно, значення `EOF` є ознакою помилки.

Макрос `assert()` використовується для перевірки на нерівність нулю змінної. Якщо значення змінної дорівнює нулю, макрос `assert`, визначений у заголовку `<assert.h>`, записує в потік `stderr` інформацію про помилку і припиняє виконання програми. В іншому випадку макрос `assert()` не виконує

ніяких дій. Хоча точний зміст повідомлень залежить від конкретної реалізації, більшість компіляторів використовують наступний шаблон:

```
Assertion failed:  file <filename>, line <linenumber>
```

Макрос `assert()` зазвичай використовується для верифікації програми, а вираз складається так, щоб він приймав значення `true`, тільки якщо жодних помилок не сталося.

Макрос `assert()` не обов'язково видаляти з програми після відладки, оскільки, якщо в програмі визначено макрос `NDEBUG`, макрос `assert()` просто ігнорується.

Приклад. В даній програмі у бінарний файл буде записаний і збережений випадковий набір чисел, потім ці числа будуть записані в текстовий файл.

```
#include <stdio.h>
#include <assert.h>
int main(void)
{
     FILE * binaryFile;
     binaryFile = fopen("file.bin", "wb"); // створення бінарного
файлу
     int numbOfItems;
     char boofer;
     printf ("Input symbols: ");
     while ((boofer=getchar())!=EOF)//зчитуємо файли до EOF
     fputc(boofer, binaryFile);
     fclose(binaryFile); // закриваємо і зберігаємо файл
     if ((binaryFile = fopen("file.bin", "rb")) == NULL)
{ // виконуємо перевірку відкриття файлу
          printf ("Error opening the file");
          return 1;
     }
     char charArray[255] = { }; // створюємо масив символів,
ініціалізуємо його порожнім символом
     int i;
     // зчитуємо символи з файлу. Умовою виходу є повернення fread
() символу EOF
```

```
     for ( i = 0; fread ( &charArray[i], sizeof(char), 1,
binaryFile); i++);
     printf ("%s", charArray);
     FILE * textFile = fopen ("file.txt", "w"); // створюємо новий
текстовий файл
     assert(textFile); //Перевіряємо файл на відкриття за допомогою
макросу assert
     fprintf(textFile, "%s", charArray);// записуємо туди зчитані з
бінарного файлу символи
     fcloseall();
     return 0;
}
```

Приклад. Створити файл test.txt, що розміщується в корневій папці диску D. Зчитати з клавіатури рядок і вмістити його у файл. Потім дані з файлу зчитати, поки не буде досягнутий кінець файлу. Зчитані дані, їх коди і звіт про роботу вивести в консоль.

```
#include < stdio.h >  //Для printf, getc, fopen, fclose, feof
#include <locale.h>
int main (void)
{   setlocale(LC_ALL,"rus");
   // Змінна, в яку буде поміщений покажчик
   // на створений потік даних
   FILE *mf;
   // Змінна, в яку по черзі будуть поміщатися зчитувальні байти
   int sym;
   char str[100];
   printf("Введіть рядок для заповнення файлу test: ");
   gets(str);
   // Відкриття файлу з режимом доступу "тільки читання"
   // і прив'язка до нього потоку даних
   printf ("Відкриття файлу: ");
   mf=fopen ("d:\\test.txt","w");
   fputs(str,mf);
   fclose(mf);
```

```
mf = fopen ("d:\\test.txt","r");
// Перевірка відкриття файлу
if (mf == NULL) {printf ("помилка\n"); return -1;}
else printf ("виконано \n");
printf ("Коди зчитаних символів:\n");
// Читання (побайтно) даних з файлу в нескінченному циклі
while (1)
{
   // Читання одного байта з файлу
   sym = fgetc (mf);

   // Перевірка на кінець файлу або помилку читання
   if (sym == EOF)
   {
      // Перевіряємо що саме сталося: скінчився файл
      // або це помилка читання
      if (feof (mf) != 0)
      {
       //Якщо файл закінчився, виводимо повідомлення про
       //завершення читання і виходимо з нескінченного циклу
         printf ("\nЧитання файлу завершено\n");
         break;
      }
      else
      {
       //Якщо при читанні сталася помилка, виводимо
       // повідомлення і виходимо з нескінченного циклу
         printf ("\nПомилка читання з файлу\n");
         break;
      }
   }
   // Якщо файл не закінчився, і не було помилки читання
   // виводимо код зчитаного символу на екран
   printf ("\nкод %d, символ %c",sym,sym);
}
// Закриваємо файл
printf ("Закриття файлу: ");
```

```
   if ( fclose (mf) == EOF) printf ("помилка\n");
   else printf ("выконано\n");
   return 0;
}
```

Приклад. Сформувати масив з даними виду: прізвище студента, номер курсу. Занести ці дані в бінарний файл. Зчитати дані з бінарного файлу, вибрати студентів, які навчаються на заданому курсі. Інформацію про цих студентів занести в текстовий файл, використовуючи функції форматованого виводу у файл.

```
#include <stdio.h>
int main()
{
     struct student
     {
          char surname[20];
          int course;
     };
     const int n=5;
     student s[n];
     int i;
     for (i=0;i<n;i++)
     {
     printf("Input surname and course for %d student: ",i+1);
     scanf("%s%d",s[i].surname,&s[i].course);
     }
     FILE* f1;
     f1=fopen("d:\\f1.dat","w+");
     if (f1==NULL)
     {
          printf("Error1");
          return 1;
     }
     fwrite(s,sizeof(s),1,f1);// запис масиву в бінарний файл
     fseek(f1,0,0);          // переводимо курсор в початок файлу
     fread(s,sizeof(s),1,f1);// читання масиву з файлу
     FILE* f2;
```

```
     f2=fopen("d:\\f2.dat","w"); // створюємо текстовий файл
     if (f2==NULL)
     {
          printf("Error2");
          return 2;
     }
     int c;
     printf("Input number of course: ");
     scanf("%d",&c);
     for (i=0;i<n;i++)
          if (s[i].course==c)
                    // запис обраних студентів в текстовий файл
               fprintf(f2,"%20s%4d\n",s[i].surname,s[i].course);
     fcloseall();
     return 0;
}
```

Приклад. Сформувати файл даних з довільним числом записів про прилад, про який відомо: назва приладу, габарити приладу (довжина, висота, ширина), вага, вартість, джерело живлення (напруга, струм, частота). Організувати вивід у формі таблиці всіх записів файлу та записів, що відповідають завданню. Завдання: скласти список з L найбільш дорогих приладів (L задається користувачем).

```
#include <stdio.h>
#include <conio.h>
int main () {
      FILE *f;
      struct dim
      {
           int length;
           int width;
           int height;
      };
      struct source
      {
           float voltage;
```

```
      float amperage;
      int frequency;
 };
 struct tovar
 {
      char name[20];
      dim dimentions;
      float weight;
      float price;
      source sour;
 };
 tovar t;
 f=fopen("d:\\Tovar.dat", "w");
if(f==NULL)
{
     printf("Error");
     return 3;
}
 int i, n;
 printf("Input number of goods-  ");
 scanf("%d", &n);
 for(i=1; i<=n; i++)
 {
    printf("Input the name of the good -  ");
    scanf("%s", t.name);
    printf("Input the dimensions of the good:\nLength - ");
    scanf("%d", &t.dimentions.length);
    printf("Width -  ");
    scanf("%d", &t.dimentions.width);
    printf("Height -  ");
    scanf("%d", &t.dimentions.height);
    printf("Input the weight of the good - ");
    scanf("%f", &t.weight);
    printf("Input the price of the good - ");
    scanf("%f", &t.price);

    printf("Input the source of electricity:\nVoltage - ");
```

```
        scanf("%f", &t.sour.voltage);
        printf("Amperage -  ");
        scanf("%f", &t.sour.amperage);
        printf("Frequency -  ");
        scanf("%d", &t.sour.frequency);
        fwrite(&t,sizeof(t), 1, f);
     }
    fclose(f);
    f=fopen("d:\\Tovar.dat", "r");
    if(f==NULL)
    {
         printf("Error");
         return 3;
    }
    i=1;
    while( fread(&t, sizeof(t), 1, f))
    {
     printf("%d  %s %d %d %d %f %f %f %f %d\n", i, t.name,
     t.dimentions.length, t.dimentions.width, t.dimentions.height,
     t.weight, t.price, t.sour.voltage, t.sour.amperage,
     t.sour.frequency);
     i++;
    }
    fclose(f);
    return 0;
getch();
 }
```

15.3 Контрольні питання

1. Для чого використовуються файли? Де вони можуть зберігатися?
2. Який заголовний файл потрібен для роботи з файлами?
3. До чого доступ здійснюється швидше: до файлів на дисках чи до інформації в оперативної пам'яті?
4. Що називається буферами вводу-виводу? Як вони працюють? Хто управляє їх роботою?

5. Що називається структурою `FILE`? Які дані вона містить?

6. Як оголосити потік – змінної-покажчика на структуру типу `FILE`?

7. Як відкрити файл? Наведіть приклади для відкриття файлів у різних режимах. Як завдати повне ім'я для файлу, що відкривається?

8. Які існують додаткові умови режиму відкриття файлу?

9. Форматований і неформатований ввід-вивод у файл: чим відрізняються, які функції використовуються?

10. Які параметри потребують функції `fwrite` та `fread`? Що вони повертають як результат своєї роботи? Наведіть приклади.

11. Які параметри потребують функції `fprintf` та `fscanf`? Що вони повертають як результат своєї роботи? Наведіть приклади.

12. Як працюють спеціальні функції, призначені для вводу-виводу одиночного символу та рядка? Що вони повертають як результат своєї роботи? Наведіть приклади.

13. За допомогою якої функції можна здійснювати операції введення/виводу з довільним доступом? Якими способами можна перевести курсор на початок файлу? Як можна повернути поточну позицію курсора? Наведіть приклади.

14. Яка функція дозволяє перевірити досягнення кінця файлу? Для чого це можна використовувати? Наведіть приклади.

15. Як можна закрити файл? Що при цьому відбувається на системному рівні?

16. Для чого використовується макрос `assert()`? У якому вигляді він видає результат перевірки?

15.4 Тести для самоконтролю

1) Яке основне призначення файлів?

А) збереження інформації між сеансами роботи однієї програми

Б) використання як буфера

В) короткострокове зберігання текстової інформації

Г) довгострокове зберігання будь-якої інформації

2) Яка функція для відкриття файлів описана в бібліотеці `stdio`?

А) `open()`

Б) `fopen()`

В) `openfile()`

Г) `stdfile()`

3) Яких ключів для відкриття файлів не існує?

А) `c+`

Б) `r+`

В) `w+`

Г) `a+`

4) Який ключ для відкриття файлу дозволяє доповнити його в кінці?

А) `w`

Б) `w+`

В) `a+`

Г) `r+`

5) Які аргументи не приймає функція `fseek()`?

А) `SEEK_SET`

Б) `SEEK_CUR`

В) `SEEK_STR`

Г) `SEEK_END`

6) Який ключ відкриває файл в бінарному вигляді?

А) `bin`

Б) `b`

В) `t`

Г) `bt`

7) Що повертає функція `fread()`?

А) число дійсно прочитаних об'єктів

Б) число непрочитаних об'єктів

В) прочитані символи з файлу

Г) поточну позицію курсора

8) Яка функція використовується для форматованого виводу у файл?

А) `fputs()`

Б) `fprintf()`

В) `cout`

Г) `fputc()`

9) Як встановити покажчик на початок файлу?

А) за допомогою функції `rewind()`

Б) за допомогою функції `fseek(f,0,0)`, де `f` – файлова змінна

В) закрити файл і знов відкрити

Г) усі відповіді вірні

10) Що робить функція `feof()`?

А) переводить покажчик на кінець файлу

Б) повертає істинне значення, якщо відкриття файлу неможливо

В) повертає істинне значення, якщо досягнуто кінця файлу

Г) немає правильної відповіді

11) Що робить функція `fclose()`?

А) закриває потік, відкритий раніше функцією `fopen()`

Б) записує всі дані, що залишилися в буфері, у файл

В) звільняє керуючий блок файлу, пов'язаного з потоком

Г) усі відповіді вірні

12) Для чого використовується макрос `assert()`?

А) для перевірки на нерівність нулю змінної

Б) для порівняння рядків

В) для звільнення пам’яті, що зайнята файлом

Г) немає такого макроса

РОЗДІЛ 16 КОМАНДИ ПРЕПРОЦЕСОРА

У програму С можна включати різні інструкції, що регламентують роботу компілятора. Ці інструкції називаються директивами препроцесора. Хоча вони не є частиною мови С, їх застосування дозволяє розширити можливості програм. Також нами будуть розглянуті макроси препроцесора.

При кожному запуску компілятора спочатку запускається препроцесор, який шукає команди препроцесора, при виконанні будь-якої у текст вихідного коду вносяться деякі зміни, в результаті чого створюється новий файл вихідного коду. Цей новий файл є тимчасовим. Він і компілюється у виконуваний файл.

Всі директиви препроцесора починаються зі знака #. Крім того, кожна директива повинна розташовуватися в окремому рядку. Оскільки директиви не є частиною С, вони не закінчуються символом ';'.

16.1 Директива #define

Директива `#define` визначає ідентифікатор і послідовність символів, яка замінює його в тексті програми. Ви вже стикалися з даною директивою, оголошуючи константи. Наприклад:

```
#define MAX_SIZE 256
```

16.2 Макроси

Директиву `#define` можна також використовувати для створення макросів. Макрос – це лексема, створена за допомогою директиви `#define`. Він приймає параметри подібно звичайній функції. Препроцесор заміняє рядок підстановки будь-яким заданим параметром. Наприклад:

```
#define MAX(x,y) ( (x) > (y)  ? (x) : (y) )
```

Зверніть увагу, що в визначенні макросу відкриваюча кругла дужка для списку параметрів повинна негайно йти за іменем макросу, інакше буде виконана просто текстова заміна. Також рекомендується використовувати навколо параметрів круглі дужки, щоб уникнути небажаних побічних ефектів при передачі у макрос складних значень.

Макрос має бути визначений в одному рядку.

Вирази, що підставляються, зазвичай працюють швидше функцій, оскільки не витрачається час на саме звернення до функції, однак відсутність макросу у вихідному коді, використовуваному компілятором для тестування, може ускладнити налагодження програми. Також у макросах не підтримується контроль за відповідністю типів даних.

16.3 Директива #include

Директива `#include` змушує компілятор зчитати і підставити у вихідний текст програми файл із заданим ім'ям. Це ім'я укладається в подвійні лапки або кутові дужки.

```
#include “stdio.h”
#include <stdio.h>
```

Лапки і дужки визначають спосіб пошуку файлів на диску. Якщо ім'я файлу міститься у кутових дужках він знаходиться в каталозі, зазначеному компілятором. Якщо ім'я файлу укладено в лапки, як правило, його пошук ведеться в робочому каталозі. Більшість програмістів укладають в кутові дужки імена заголовних файлів, а лапки залишають для файлів, визначених користувачем.

16.4 Директиви умовної компіляції

Ймовірно, найбільш поширеними директивами умовної компіляції є директиви `#if, #else, #elif, #endif`. Ці директиви дозволяють вибирати компільовані частини залежно від значення костантного вираження.

Директива `#if` має наступний вигляд:

```
#if вираз
    послідовність операторів
#endif
```

Якщо значення константного виразу істино, код, укладений між директивами `#if` і `#endif` компілюється. В іншому випадку ця частина програми ігнорується. Директива `#endif` відзначає кінець блоку директиви `#if`.

Директива `#else` ідентична оператору `else`, що є частиною мови С: вона визначає альтернативу директиві `#if`

```
#if вираз
```

```
    оператори
#else
    оператори
#endif
```

Директива `#elif` означає «`else if`» і створює ланцюжок `if-else-if`, дозволяючи виконувати багатоваріантну умовну компіляцію. Якщо вираз істинно, компілюється пов'язаний з ним блок коду, а інші вирази `#elif` не обчислюються. В іншому випадку обчислюється наступний блок. Загальний вигляд директиви `#elif` такий:

```
#if вираз
    оператори
#elif вираз1
    оператори
#elif вираз2
    оператори
.
.
.
#elif виразN
    оператори
#endif
```

Інший спосіб умовної компіляції заснований на застосуванні директив `#ifdef` і `#ifndef`. Загальний вигляд директиви `#ifdef` такий:

```
#ifdef macros_name
    оператори
#endif
```

Якщо ім'я макросу раніше визначено за допомогою директиви `#define`, компілюється відповідний блок коду.

Директива `#ifndef` має наступний вигляд:

```
#ifndef macros_name
    оператори
#endif
```

Якщо ім'я макросу не було визначено за допомогою директиви `#define`, компілюється відповідний блок коду.

Директива `#undef` видаляє визначення зазначеного імені макросів. Іншими словами, ця директива робить макрос невизначеним. Вона визначається як
`#undef macros_name`

Директива `#pragma` залежить від реалізації. Деталі й допустимі опції описуються у документації компілятора.

Прероцессор має 2 оператора `#` і `##`. Ці оператори використовуються в директиві `#define`. Оператор `#` перетворює свій аргумент в рядок, укладений в лапки, оператор `##` конкатенує («склеює») дві лексеми. У більшості випадків без цих операторів можна обійтися, вони призначені для обробки деяких особливих ситуацій.

16.5 Контрольні питання

1. Як називаються інструкції, які регламентують роботу компілятора?
2. Чому директиви препроцесора не закінчуються символом «`;`»?
3. Що робить директива `#define` ?
4. Що таке макрос? Як він пов’язан з директивою `#define` ?
5. Чому макроси працюють швидше функцій?
6. Чи підтримується у макросах контроль за відповідністю типів даних?
7. Що робить директива `#include` ? Що визначають лапки і дужки, у які укладено ім’я файлу?
8. Що означаю директива `#if` ? Наведіть приклад використання.
9. Як працює директива `#else` ? Якому оператору мови С вона ідентична?
10. У чому є принцип роботи директиви `#elif` ? Який ланцюжок вона створює? Наведіть приклади використання.
11. Директиви `#ifdef` та `#ifndef`: призначення, загальний вигляд, приклади.

16.6 Тести для самоконтролю

1) Що роблять директиви препроцесора?

А) регламентують роботу компілятора

Б) застосовуються тільки для підключення заголовних файлів

В) застосовуються тільки для оголошення констант

Г) немає правильної відповіді

2) З якого знака починаються директиви препроцесора?

А) `<`

Б) `&`

В) `$`

Г) `#`

3) Що робить директива `#define`?

А) визначає ідентифікатор і послідовність символів, яка замінює його в тексті програми

Б) визначає новий тип даних

В) підключає статичну бібліотеку

Г) немає правильної відповіді

4) Що називається макросом?

А) лексема без параметрів

Б) лексема, яка приймає параметри подібно звичайній функції

В) те ж саме, що і функція

Г) у мові С макроси не використовуються

5) Що робить директива `#include`?

А) перевіряє на відповідність типи даних

Б) перейменовує файл

В) зчитує і підставляє у вихідний текст програми файл із заданим ім'ям

Г) виключає з тексту програми коментарі

6) Як працює директива `#if`?

А) код, укладений між директивами `#if` і `#endif` компілюється

Б) якщо значення константного виразу є помилковим, код, укладений між директивами `#if` і `#endif` компілюється

В) код, що передує `#if`, компілюється

Г) якщо значення константного виразу істино, код, укладений між директивами `#if` і `#endif` компілюється

7) Який ланцюжок створює директива `#elif`?

А) `if-else-if`

Б) `if-else`

В) `if`

Г) `if-else-while`

8) Що робить директива `#ifdef`?

А) компілюється відповідний блок коду

Б) код не компілюється

В) якщо ім'я макросу раніше визначено за допомогою директиви `#define`, компілюється відповідний блок коду

Г) якщо ім'я макросу не було визначено за допомогою директиви `#define`, компілюється відповідний блок коду

9) Що робить директива `#undef`?

А) робить макрос визначеним

Б) видаляє визначення зазначеного імені макросів

В) видаляє програму з пам’яті

Г) немає правильної відповіді

10) Що робить оператор `##`?

А) порівнює дві лексеми

Б) привласнює одну лексему іншій

В) перевіряє помилки умовної компіляції

Г) конкатенує дві лексеми

РОЗДІЛ 17 БІТОВІ ОПЕРАЦІЇ

17.1 Операції з бітами даних

Іноді, щоб відстежувати стан змінних буває зручно встановлювати для них прапори. Для прапора можна використовувати змінні типу `bool`, але якщо у вас багато ознак і для вас важливо економити ресурси комп'ютера, зручніше для встановлення прапорів використовувати окремі біти значення в двійковому форматі. Кожен байт має вісім бітів, тому чотирьохбайтова змінна типу `long` може представляти 32 окремих прапора. Якщо значення біта дорівнює 1, то говорять, що прапор встановлений, а якщо 0 – то скинутий. Іншими словами, щоб встановити прапор, потрібно певному біту присвоїти значення 1, а щоб скинути прапор – значення 0. Встановлювати і скидати прапори можна, змінюючи значення змінної типу `long`, але такий підхід не раціональний і може ввести в оману. У мові С передбачені побітові оператори для роботи з бітами даних. Вони схожі, але в той же час відрізняються від логічних операторів, тому багато програмістів-початківців їх плутають. До побітових операторів відносяться:

Логічне І (AND)

Для позначення оператора побітового І (&) використовується одиночний амперсанд. При виконанні операції побітового І з двома бітами результат дорівнює 1, якщо обидва рівні 1 і 0, якщо хоча б один біт (або відразу обидва) дорівнює 0.

Логічне АБО (OR)

При виконанні операції побітового АБО (|) з двома бітами результат дорівнює 1, якщо хоча б один з операндів дорівнює 1.

Виключне АБО (XOR)

При виконанні операції виключного АБО (^) з двома бітами результат дорівнює 1, якщо мають різне значення.

Зсув вліво

Оператор зсуву вліво (<<) зсуває байти в лівому операнді на число біт, задане у правому операнді; праворуч заповнюється нулями.

Зсув вправо

Оператор зсуву вправо (>>) зсуває біти в лівому операнді вправо на число біт, задане у правому операнді; метод заповнення ліворуч залежить від типу даних і конкретної системи.

Доповнення до одиниці

Оператор доповнення до одиниці (~) скидає кожний встановлений біт і встановлює кожен скинутий біт в числі. Якщо поточне значення дорівнює 10100011, до доповнення його до одиниці буде мати вигляд 0101 1100.

Порозрядне присвоювання

Кожна поразрядна операція (виключаючи операцію доповнення) має відповідні знаки операції привласнення. Ці знаки операції порозрядного привласнення (`& =, | =, ^ =, << =, >> =`), використовуються точно так само, як знаки арифметичного присвоювання.

17.2 Встановлення бітів

Якщо ви хочете встановити або скинути конкретний прапор, слід використовувати операцію маскування. Якщо в програмі для встановлення прапорів використовується 4-байтова змінна і потрібно встановити прапор, пов'язані з восьмим бітом цієї змінної, слід виконає операцію побітового АБО для цієї змінної і числа 128, яке називають маскою.

Маска – це ціле значення, визначені біти якого встановлені рівними одиниці. Маски використовуються для того, щоб заховати деякі біти числа. Чому 128? Тому що 128 – це 1000 0000 в двійковій системі числення, таким чином, можна сказати, що число 128 визначає значення восьмого розряду. Аналогічно, можемо визначити маску за допомогою зсуву вліво, записавши це як 1 << 7 (1000 0000). Оператор зсуву вліво зсуває величину 1 з біта молодшого розряду (крайнього праворуч) в біт старшого розряду (крайній ліворуч), і заповнює нулями біти праворуч. Незалежно від поточного значення цього розряду в 4-байтовой змінної (встановлений він або скинутий), при виконанні операції АБО з числом 128 цей біт буде встановлений, а всі інші біти збережуть свої значення. Застосування до числа у двійковому форматі 1010 0110 0010 0110 операції АБО з числом 128 виглядає наступним чином:

```
1010 0110 0011 0110 //восьмий біт скинутий
|         0000 0000 1000 0000 // АБО 128
1010 0110 1010 0110 //8-й біт встановлений
```

Хочеться звернути вашу увагу на деякі речі. По-перше, як правило, біти читають справа наліво. По-друге, значення 128 містить всі нулі, за винятком восьмого біта, тобто того розряду, який ви хочете встановити. По-третє, у вихідному числі 1010 0110 1010 0110 операцією АБО змінюється тільки восьмий

біт. Якби він у цьому значенні був встановлений ще до виконання операції АБО то значення взагалі не змінилося б.

Якщо потрібно скинути восьмий біт, можна використовувати побітову операцію І з доповненням числа 128 до одиниці. Доповнення числа 128 до одиниці – це таке число, яке виходить, якщо в двійковому форматі представити число 128 (1000 0000), а потім встановити в ньому кожен скинутий і скинути кожний встановлений біт (0111 1111). При виконанні побітовій операції І з цими числами вихідне число не змінюється, за винятком восьмого розряду, який скидається в нуль.

```
1010 0110 1010 0110 //8-й біт встановлений
&  1111 1111 0111 1111 // ~128 (доповнення до одиниці числа 128)
1010 0110 0010 0110 // 8-й біт скинутий
```

Щоб до кінця зрозуміти метод рішення, зробіть самостійно всі математичні операції. Щоразу, коли обидва біти рівні 1, запишіть результат рівним 1. Якщо який-небудь біт дорівнює 0, запишіть у відповіді 0. Порівняйте відповідь з вихідним числом. Воно повинно бути без змін, за винятком восьмого біта, який в результаті цієї операції побітового І буде скинутий.

Нарешті, якщо ви хочете інвертувати восьмий біт незалежно від його попереднього стану, використовуйте операцію виключає АБО для цього числа і числа 128. Отже:

```
1010 0110 1010 0110 //число
^         0000 0000 1000 0000 //128
1010 0110 0010 0110 //8-й біт інвертован
^         0000 0000 1000 0000 // 128
1010 0110 1010 0110 //8-й біт інвертован знова
```

Операції зсуву також можна використовувати для ефективного множення або ділення на ступінь двійки.

`number<<n` перемножає `number` на 2 в `n`-му ступені

`number>>n` ділить `number` на 2 в `n`-му ступені, якщо число невід'ємне.

Приклад.

```
#include <stdio.h>
#include <stdlib.h>
void ShowBinary(int val) {
```

```
	int i = 0;
	printf("Decimal: %d\t", val);
	printf ("Binary: ");
	for(i = 7;i>=0;--i) //робимо побітовий зсув для кожного
//розряду і виконуємо операцію логічнгого множення з одиницею
		printf("%d",(val >> i) & 1); // для виводу значення
					// лише останнього біту
	printf("\n");
}
int main() {
	const short X = 240;
	const short Y = 129;
	short Z;
	printf ("X: ");
	ShowBinary(X);
	printf("Y: ");
	ShowBinary(Y);
	Z = X & Y;
	printf ("\nZ = X & Y: \n");
	ShowBinary(Z);
	Z = X | Y;
	printf ("\nZ = X | Y: \n");
	ShowBinary(Z);
	Z = X >> 3;
	printf ("\nZ = X >> 3: \n");
	ShowBinary(Z);
	system("pause");
	return 0;
}
```

17.3 Контрольні питання

1. Чому дорівнює значення біту, якщо прапор встановлений? А якщо прапор скинутий?
2. Як називаються оператори для роботи з бітами даних?
3. Як працює «логічне І»? Наведіть таблицю істинності.

4. Як працює «логічне АБО»? Наведіть таблицю істинності.

5. Як працює «виключне АБО»? Наведіть таблицю істинності.

6. Наведіть оператори зсуву вліво та вправо.

7. Як працює оператор доповнення до одиниці? Наведіть приклади.

8. Які існують операції порозрядного присвоювання? Наведіть приклади.

9. Як виконується робота з масками? Що потрібно зробити, щоб встановити 2 та 5 біти? Щоб скинути 4 біт?

10. Як за допомогою бітових зсувів виконувати множення на 2? Наведіть приклади.

11. Як за допомогою бітових зсувів виконувати ділення на 2? Наведіть приклади.

17.4 Тести для самоконтролю

1) Що необхідно зробити для встановлення та скидання прапору?

А) для встановлення дорівняти байт 1, для скидання – 0

Б) для встановлення дорівняти байт 0, для скидання – 1

В) для встановлення дорівняти біт 1, для скидання – 0

Г) для встановлення дорівняти біт 0, для скидання – 1

2) Оберіть вірне твердження:

А) `1 AND 1 AND 0 = 1`

Б) `0 AND 1 AND 0 = 1`

В) `1 AND 1 AND 1 = 0`

Г) `1 AND 1 AND 0 = 0`

3) Оберіть вірне твердження:

А) `(1 OR 0) AND 0 = 1`

Б) `1 OR (0 AND 1) = 0`

В) `1 OR (1 AND 0) = 1`

Г) `0 OR (1 AND 1) = 0`

4) Оберіть вірне твердження:

А) `(0 ^ 1) OR 0 = 0`

Б) `1 AND (1 ^ 0) = 1`

В) `1 OR (0 ^ 0) = 0`

Г) `0 AND (1 ^ 1) = 1`

5) Що робить оператор << ?

А) виконує зсув вліво на задане число біт

Б) виконує зсув вліво на 1 біт

В) виконує зсув вправо на задане число біт

Г) виконує зсув вправо на 1 біт

6) Що робить оператор ~ ?

А) обнуляє всі біти числа

Б) встановлює в 1 всі біти числа

В) такого оператора у С немає

Г) інвертує кожний біт числа

7) Що називається маскою?

А) ціле значення, що використовується для зміни окремих бітів

Б) значення типу bool

В) ціле значення, всі парні біти якого встановлені в 1

Г) ціле значення, всі непарні біти якого встановлені в 0

8) Як з числа 00110101 отримати 10111101?

А) зробити 00110101 `AND` 10001000

Б) зробити 00110101 `OR` 10001000

В) зробити 00110101 `XOR` 10001000

Г) зробити 00110101 `OR` 11001010

9) Як з числа 10101110 отримати 00001110?

А) зробити 10101110 `XOR` 01011110

Б) зробити 10101110 `AND` 01011110

В) зробити 10101110 `OR` 11110000

Г) зробити 10101110 `AND` 11110000

10) Як за допомогою оператора зсуву можна помножити число на 8?

А) зробити зсув на 1 біт вправо

Б) зробити зсув на 3 біти вправо

В) зробити зсув на 3 біти вліво

Г) зробити зсув на 1 біт вліво

РОЗДІЛ 18 ДИНАМІЧНІ СТРУКТУРИ ДАНИХ

Якщо до початку роботи з даними неможливо визначити, скільки пам'яті потрібно для їх зберігання, пам'ять слід розподіляти під час виконання програми в міру необхідності окремими блоками. Блоки зв'язуються один з одним за допомогою покажчиків. Такий спосіб організації даних називається динамічною структурою даних, оскільки вона розміщується в динамічній пам'яті і її розмір змінюється під час виконання програми [28].

З динамічних структур у програмах найчастіше використовуються лінійні списки, стеки і черги. Вони розрізняються способами зв'язку окремих елементів і допустимими операціями. Динамічна структура, на відміну від масиву або запису, може займати несуміжні ділянки оперативної пам'яті.

Елемент будь-якої динамічної структури складається з двох частин: інформаційної, заради зберігання якої і створюється структура, і покажчиків, що забезпечують зв'язок елементів один з одним.

18.1 Зв'язані списки

У зв'язаному списку кожний елемент пов'язаний з наступним і, можливо, з попереднім. У першому випадку список називається однозв'язний, у другому – двозв'язним. Якщо останній елемент зв'язати покажчиком з першим, вийде кільцевий список [29].

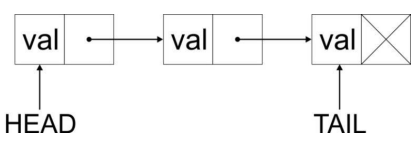
Над списками можна виконувати такі основні операції:

1. Початкове формування списку (створення першого елемента).
2. Додавання елемента в кінець списку.
3. Читання елемента із заданою властивістю.
4. Вставка елемента в задане місце списку (до або після елемента із заданою властивістю).
5. Видалення елемента із заданою властивістю.
6. Упорядкування списку.

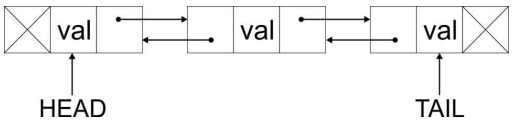
Для роботи зі списком у програмі потрібно визначити покажчик на його початок (HEAD). Щоб полегшити додавання нових елементів в кінець списку, можна також завести покажчик на кінець списку (TAIL).

Лінійні списки можна схематично зобразити таким чином:

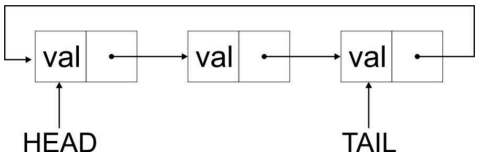
Однозв’язний список:



Двозв’язаний список:



Кільцевий список:



Розглянемо приклад програми для роботи з простим однозв'язним списком:

```
struct Node {
     int value;
     Node* next;
};
Node *HEAD, *TAIL;
void createList(int val);
void addToEnd(int val);
void addAfter(int ind, int val);
void deleteAtInd(int ind);
Node readAtInd(int ind);
void sort();
```

Тут ми створюємо структуру `Node` – вузол списку, яка містить деяке ціле число і покажчик на наступний вузол. Також створюються покажчики на початок і кінець списку і функції для виконання над списком таких операцій: створення списку, додавання вузла в кінець списку, додавання вузла після вузла із заданим індексом, видалення вузла з заданим індексом і читання вузла із заданим індексом. Розглянемо кожну функцію:

Створення списку:

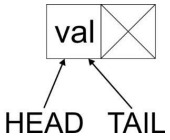
```
void createList(int val) {
     HEAD = (Node*) malloc(sizeof(Node));

     HEAD->value = val;
```

```
    HEAD->next = NULL;
    TAIL = HEAD;
}
```

Як аргумент функція отримує дані, які будуть зберігається в першому елементі списку. Спочатку виділяється пам'ять під новий вузол і покажчик на неї присвоюється вказівником на початок списку (HEAD). Цьому вузлу присвоюємо дані, які він буде зберігати. Далі вказівником на наступний вузол присвоюємо значення NULL, а вказівником TAIL присвоюємо значення вузла HEAD, так як наш список поки що складається тільки з одного вузла, який є одночасно початком і кінцем списку.

Схематичний рисунок:

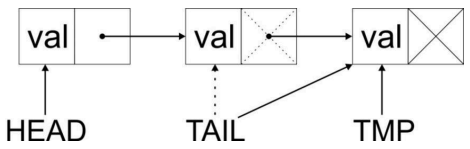


Додавання вузла в кінець:

```
void addToEnd(int val) {
    Node *tmp = (Node*) malloc(sizeof(Node));
    tmp->value=val;
    tmp->next = NULL;
    TAIL->next = tmp;
    TAIL = tmp;
}
```

Як аргумент функція отримує дані, які будуть зберігатися в новому елементі списку. Для початку створимо новий вузол в пам'яті і присвоїмо йому необхідні дані. Покажчику на наступний елемент присвоюється значення NULL, тому що новий елемент буде останнім у списку. В даний момент TAIL вказує на елемент, який буде в списку передпостаннім після додавання нашого нового елемента, тому його вказівником next присвоюємо наш новий вузол. І в кінці присвоюємо вказівнику TAIL новий вузол.

Схематичний рисунок:



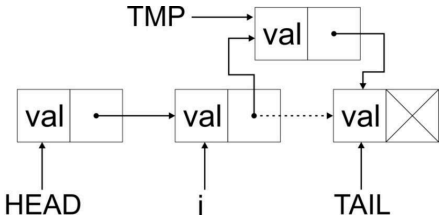
Додавання вузла після вузла з заданим індексом:

```
void addAfter(int ind, int val) {
```

```
    Node* i = HEAD;
    int j = 0;
    while(i != NULL) {
        j++;
        if(j == ind) {
            Node *tmp = (Node*) malloc(sizeof(Node));
            tmp->value = val;
            tmp->next = i->next;
            i->next = tmp;
        }
        else {
            i = i->next;
        }
    }
}
```

Як аргумент функція отримує індекс вузла, після якого необхідно вставити новий і дані, які будуть зберігатися в новому елементі списку. Для початку створимо покажчик на вузол і, за допомогою якого здійснимо прохід по списку та присвоїмо йому початковогий вузол, а також лічильник. У циклі while проходимо список (поки і не отримає значення NULL, тобто поки не досягнемо кінця списку). Якщо значення лічильника дорівнює заданому індексу, створюємо новий вузол, вказівнику на наступний якого присвоїмо елемент, наступний елемент із заданим індексом (i). А вказівнику наступного цього елемента відповідно присвоюємо новий вузол.

Схематичний рисунок:



Видалення елемента з заданим індексом:

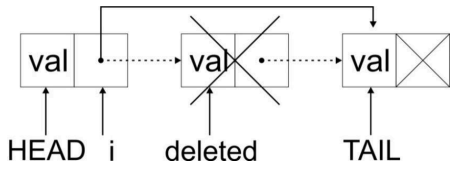
```
void deleteAtInd(int ind) {
    Node* i = HEAD;
    int j = 0;

    while(i != NULL) {
```

```
            j++;
            if(j == ind-1) {
                  Node *deleted = i->next;
                  i->next = deleted->next;
                  free(deleted);
            }
            else {
                  i = i->next;
            }
      }
}
```

Як аргумент функція отримує індекс вузла, який потрібно видалити. Прохід по списку здійснюється аналогічно вищеописаній функції, однак нам необхідно знайти елемент, що стоїть перед тим, що буде видалено. Далі отримуємо покажчик на вузол, що видаляється і покажчику next попереднього вузла присвоюємо вузол наступний за тим, що видаляється. Потім звільняємо пам'ять за вказівником на вузол, що видаляється.

Схематичний рисунок:



Читання елемента з заданим індексом:

```
Node* readAtInd(int ind) {
      Node* i = HEAD;
      int j = 0;
      while(i != NULL) {
            j++;
            if(j == ind) {
                  return i;
            }
            else {
                  i = i->next;
            }
      }
      return NULL; }
```

Як аргумент функція отримує індекс вузла, який потрібно отримати. Функція повертає покажчик на необхідний вузол. Прохід по списку здійснюється аналогічно розглянутим вище функціям. Після знаходження необхідного елемента повертаємо покажчик на нього. Якщо ж пройдено весь список, а потрібний вузол не знайдено, повернемо NULL.

Сортування списку:

```
void sort() {
     Node* i = HEAD;
     int n = 0;
     while(i != NULL) {
          n++;
          i=i->next;
     }
     int tmp;
     for(int j=0; j<n; j++) {
          i = HEAD;
          for(int k=0; k<n; k++) {
               if(i->value > i->next->value) {
                    tmp = i->value;
                    i->value = i->next->value;
                    i->next->value = tmp;
               }
               i = i->next;
          }
     }
}
```

Функія обчислює кількість елементів списку у циклі while() і виконує просте сортування бульбашкою: порівнюють значення сусідніх вузлів списку і в разі необхідності вузли міняють місцями. Крім чисел, сортувати можна і рядки за алфавітом за зростанням чи спаданням.

Приклад. Виконати сортування рядків бінарним методом у порядку зростання. Рядки зберігати у списку. Розробити окремі функції для створення списку та для сортування.

```
#include <stdlib.h>
#include <stdio.h>
```

```
#include <string.h>
struct List
{
    char ** mas;
    int count;
};
struct List * createList(char* str) {
     int i,j=0,k,begin=0;
     struct List * list = (struct List*) malloc(sizeof(struct
List*));
     list->count=0;
     for(i=0;i<strlen(str);++i)
          if(str[i]=='\\')
               list->count++;
     list->mas = (char**)malloc(list->count);
     for(i=0;i<list->count;++i) {
          while(str[j]!='\\')
               j++;
          list->mas[i]=(char*)malloc((j-begin+1)*sizeof(char));
          for(k=0;begin<j;++begin,k++)
               list->mas[i][k] = str[begin];
         list->mas[i][k]='\0';
          begin=j+1;
          j=j+1;
     }
     return list;
}

void bin_paste_sort(char**mas,int count,int flag) {
 int left=0,right=0,sred=0,i,j;
  for (i=1; i<count; i++)  {
     char * temp = (char*)malloc(strlen(mas[i])*sizeof(char));
     strcpy(temp,mas[i]);
      left=i;
      right=0;
         while (left > right) {
```

```
          sred=(left+right) / 2 ;
            if (strcmp(mas[sred],temp)<0)
               left=sred;
            else
               right=sred+1;
           }
          for (j=i-1; j>=left; j--)
          strcpy(mas[j+1],mas[j]);
          strcpy(mas[left],temp);
  }
       if(flag==0)
          for(i=0,j=count-1;i<count/2;j--,i++){
          char*temp=(char*)malloc(sizeof(char)*strlen(mas[j]));
               strcpy(temp,mas[j]);
               mas[j]=(char*)malloc(sizeof(char)*strlen(mas[i]));
               strcpy(mas[j],mas[i]);
               mas[i]=(char*)malloc(sizeof(char)*strlen(temp));
               strcpy(mas[i],temp);
          }
}

void main() {
char * str= "zzz\\optreasdpasd\\lllohgfrty\\xfdsad\\
             yyhhf\\asdcc\\aa\\bbbrr\\hgdfsds\\";
     struct List * list = createList(str);
     int i;
     bin_paste_sort(list->mas,list->count,0);
     for(i=0;i<list->count;++i) {
          printf("%s\n",list->mas[i]);
     }
     system("pause");
}
```

Робота з двозв'язним і кільцевими списками здійснюється аналогічно, проте слід пам'ятати, що при роботі з двозв'язним списком необхідно зберігати і змінювати також покажчик на попередній вузол prev, а в кільцевому списку

покажчик next останнього вузла `TAIL` повинен вказувати на початковий вузол `HEAD`, а не мати значення `NULL`.

18.2 Багатозв'язні списки

Багатозв'язні списки являють собою динамічні структури даних, в основу яких покладено одно- або двозв'язні списки, в яких є додаткові зв'язки між вузлами [30]. Найчастіше, такі зв'язки проводяться між далеко віддаленими вузлами, наприклад, позначають категорії даних.

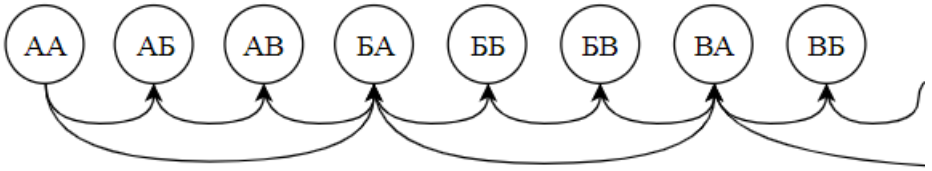


Рисунок 18.1 – Приклад багатозв'язного списку

Перехід між вузлами АА і БА може виконаний за додатковою зв'язку в обхід ланок АБ і АВ. Через такий характер переміщення ці списки іноді називають скіп-списками (skip – перестрибувати). А при характері розміщення даних, подібному показаному на цьому малюнку, такі списки називають словниковими (іноді просто словниками, але термін "словник" може використовуватися в теорії структур даних у різних значеннях).

18.3 Стек

Стек – динамічна структура даних, що представляє із себе упорядкований набір елементів, в якій додавання нових елементів і видалення існуючих проводиться з одного кінця, який називають вершиною стека [31].

За визначенням, елементи витягуються з стека в порядку, зворотному їх додаванню в цю структуру, тобто діє принцип "останній прийшов – перший пішов" (`LIFO` – last in - first out).

Найбільш наочним прикладом організації стека служить дитяча пірамідка, де додавання і зняття кілець здійснюється саме згідно з визначенням стека.

Стек можна організувати на базі будь-якої структури даних, де можливо зберігання декількох однотипних елементів і де можна реалізувати визначення стека: лінійний масив, односпрямований або двонаправлений список і т.п. У

нашому випадку найбільш відповідним для реалізації стека є односпрямований список, причому як вершини стека виберемо початок цього списку.
Виділимо типові операції над стеком і його елементами:
1. Додавання елемента в стек.
2. Видалення елемента зі стека.
3. Перевірка, чи порожній стек.
4. Перегляд елемента в вершині стека без видалення.
5. Очищення стека.

Розглянемо приклад програми з реалізацією стека і декількох основних операцій:

```
#include <stdlib.h>
#include <stdio.h>
struct StackNode
{
     int value;
     Node* next;
};
StackNode *TOS;
void createStack();
void stack_push(int val);
StackNode* stack_pop();
bool stack_isEmpty();
```

Структура вузла стека аналогчна розглянутому раніше списку. Замість покажчика на початок і кінець списку створимо покажчик на вершину стека `TOS`.
Розглянемо кожну операцію:

Створення стека:

```
Void createStack() {
     TOS = NULL;
}
```

Створюємо пустий стек – `TOS` присвюємо `NULL`.

Додавання елемента в стек:

```
void stack_push(int val) {
```

```
    Node *tmp = (Node*) malloc(sizeof(Node));
    tmp->value=val;
    tmp->next = TOS;
    TOS = tmp;
}
```

Створимо новий вузол з необхідними даними. Покажчик next цього вузла повинен вказувати на поточну вершину стека, потім перевизначаємо покажчик вершини так, щоб він вказував на нову вершину.

Видалення елемента зі стека:

```
StackNode* stack_pop() {
    if(isEmpty()) {
        printf("Error! Stack is empty. Cannot pop.");
        return NULL;
    }
    Node* popped = TOS;
    TOS = TOS->next;
    return popped;
}
```

Якщо стек порожній, виводимо повідомлення про помилку та повертаємо `NULL`. Інакше, зберігаємо покажчик на поточну вершину, перевизначаємо покажчик вершини так, щоб він вказував на елемент, наступний за поточної вершиною. Потім повертаємо вузол, видалений зі стека.

Перевірка, чи порожній стек:

```
bool stack_isEmpty() {
    return TOS==NULL ? true : false;
}
```

Функція повертає true, якщо показчик на вершину стека – `NULL`. Інакше, повертається `false`.

18.4 Черга

Черга – це інформаційна структура, в якій для додавання елементів доступний тільки один кінець, званий хвостом, а для видалення – інший, званий

головою, тобто діє принцип (перший прийшов – перший пішов (FIFO – first in - first out). Чергу, як і стек, реалізуємо за допомогою однозв’язного списку [32].

Виділимо типові операції над чергами:

1. Додавання елемента в чергу (приміщення у хвіст).
2. Видалення елемента з черги (видалення з голови).
3. Перевірка, порожня черга.
4. Очищення черги.

Розглянемо приклад реалізації черги і деяких основних операцій:

```
#include <stdlib.h>
#include <stdio.h>
struct QueeNode
{
     int value;
     Node* next;
};
QueeNode *HEAD, *TAIL;
void createQuee();
void quee_push(int val);
QueeNode* quee_pop ();
bool quee_isEmpty();
```

Структура вузла аналогічна однозв'язному списку і стеку. Створюємо покажчики на голову і хвіст черги.

Розглянемо кожну операцію:

Створення черги:

```
void createQuee(int val)
{
     TAIL = HEAD = NULL;
}
```

Створюємо пусту чергу: `HEAD` і `TAIL` присвоюємо `NULL`.

Додавання елемента в хвіст черги:

```
void quee_push(int val) {
```

```
     QueeNode *tmp = (Node*) malloc(sizeof(Node));
     tmp->value=val;
     tmp->next = NULL;
     TAIL->next = tmp;
     TAIL = TMP;
}
```

Функція аналогічна додаванню елемнта в кінец списка.

Видалення елемента з голови черги:

```
QueeNode* quee_pop() {
     if(isEmpty()) {
          printf("Error! Quee is empty. Cannot pop.");
          return NULL;
     }
     QueeNode* popped = HEAD;
     HEAD = HEAD->next;
     return popped;
}
```

Функція аналогічна витяганню елемента з вершини стека.

Перевірка, чи порожня черга:

```
bool quee_isEmpty() {
     return HEAD==NULL ? true : false;
}
```

Функція аналогічна функції перевірки на порожність стека.

18.5 Бінарні дерева

Зв'язані списки не охоплюють весь спектр можливих представлень даних. Наприклад, з їх допомогою важко описати ієрархічні структури, подібні каталогам і файлам або зберігання інформації генеалогічного древа. Для цього краще підходить модель, відома як бінарні дерева. Графічно бінарні дерева можна зобразити як послідовність об'єктів, кожен з яких може бути пов'язаний з двома наступними (рис. 18.2).

Кожен об'єкт бінарного дерева має два покажчика: на «ліву» і «праву» вершини. Самий верхній рівень називається коренем дерева. Якщо покажчики об'єкта left і right дорівнюють NULL, то він називається листом дерева.

Наприклад, при описі структури каталогів за допомогою бінарного дерева можна скористатися наступним правилом. Перехід по «лівим» вказівниками означатиме список файлів, а перехід по «правим» – список каталогів. Наприклад, для опису простої структури каталогів, бінарне дерево буде мати вигляд, представлений на рис. 18.3.

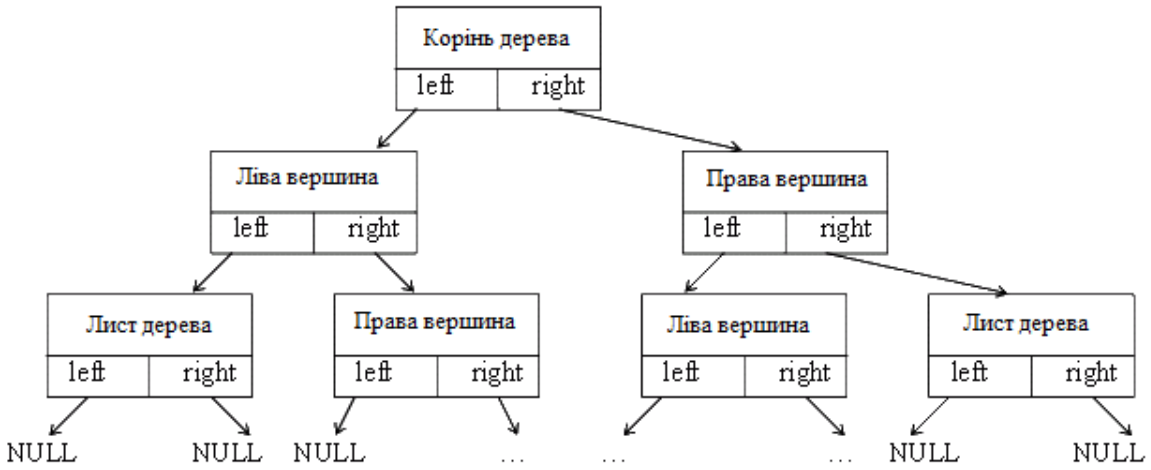


Рисунок 18.2 – Графічна інтерпрітація бінарного дерева

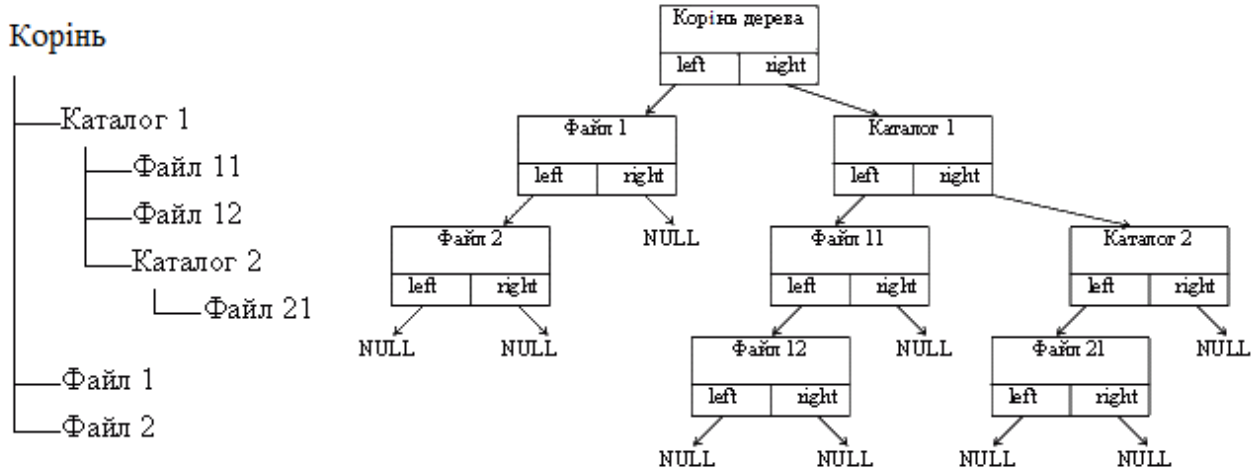


Рисунок 18.3 – Структура каталогів

Обхід дерев – послідовна обробка (перегляд, зміна тощо) всіх вузлів дерева, при якому кожен вузол обробляється строго один раз.

Залежно від траєкторій виділяють два типи обходу:

– горизонтальний (вшир);

– вертикальний (вглиб).

Горизонтальний обхід передбачає обхід дерева за рівнями (level-ordered) – спочатку обробляються всі вузли поточного рівня, після чого здійснюється перехід на нижній рівень.

При вертикальному обході порядок обробки поточного вузла і вузлів його правого і лівого піддерев варіює і за цією ознакою виділяють три варіанти вертикального обходу:

– прямий (префіксний, pre-ordered): вершина – ліве піддерево – праве піддерево;

– зворотний (інфіксне, in-ordered): ліве піддерево – вершина – праве піддерево;

– кінцевий (постфіксний, post-ordered): ліве піддерево – праве піддерево – вершина.

Розглянемо приклад реалізації дерева, основних операцій над ним і різних варіантів обходу:

```
struct TreeNode {
     int value;
     Node* left;
     Node* right;
};

enum Side {LEFT, RIGHT};
Node *root;
void createTree(int val);
TreeNode* createNode(int val);
void addNode(int val, TreeNode* parent, Side side);
void passLevelOrder(TreeNode* top);
void passPreOrder(TreeNode* top);
void passInOrder(TreeNode* top);
void passPostOrder(TreeNode* top);
```

Стуктура вузла дерева: збурежувані дані та показчики на лівий і правий дочірній вузли. Перерахування Side використовується у функції додавання вузла для визначення лівим чи правим буде доданий вузол. Також зберігаємо показчик на корінь дерева.

Розглянемо кожну функцію:

Створення дерева:

```
void createTree(int val) {
     root = createNode(val);
}
```

Створюємо новий вузол і присвоюємо його root.

Створення вузла:

```
TreeNode* createNode(int val) {
     TreeNode *newNode = (TreeNode*) malloc(sizeof(TreeNode));
     newNode->value = val;
     newNode->left = NULL;
     newNode->right = NULL;
     return newNode;
}
```

Виділяємо пам’ять під новий вузол, зберігаємо там необхідні дані, показчикам на `left` і `right` присвоюємо `NULL` та повертаємо показчик на створений вузол із функції.

Додавання вузла:

```
void addNode(TreeNode* newNode ,TreeNode* parent, Side side) {
     if(parent == NULL) {
          printf("Error! Parent cannot be a NULL-pointer.");
          return;
     }
     if(side == LEFT) {
          if(parent->left != NULL) {
printf("Error! Parent left child node is already occupied.");
               return;
          }
          parent->left = newNode;
     }
```

```
     else {
          if(parent->right != NULL) {
printf("Error! Parent right child node is already occupied.");
               return;
          }
          parent->right = newNode;
     }
}
```

Функція приймає показчик на вузол, що буде доданий до дерева, показчик на батьківський вузол, до якого додавати вузол та тип доданого дочірнього вузла (лівий чи правий). Якщо в якості батьківського вузла переданий NULL-показчик, чи дочірній вузол потрібного типу вже існує, видаємо повідомлення про помилка и виходимо з функції. В іншому випадку, присвоюємо відповідному показчику parent новий дочірній вузол.

Горизонтальний обхід:

```
void passLevelOrder(TreeNode* top) {
     QueeNode *HEAD, *TAIL;
     createQuee();
     do {

          /*...Необхідні операції над вузлом top...*/

          if(top->left != NULL) {
               quee_push(top->left);    }
          if(top->right != NULL) {
               quee_push(top->right);   }
          if(!quee_isEmpty()) {
               top = quee_pop();   }
     } while(!quee_isEmpty());
}
```

Обробляємо перший у черзі вузол, за наявності дочірніх вузлів заносимо їх в кінець черги. Переходимо до наступної ітерації.

Вертикальний прямий обхід:

```
void passPreOrder(TreeNode* top) {
```

```
     StackNode* TOS;
     createStack();

     while (top != NULL || !stack_isEmpty()) {
          if (!stack_isEmpty()) {
               top = stack_pop();
           }
        while (top != NULL) {
            /*...Необхідні операції над вузлом top...*/
            if (top->right != NULL) {
                    stack_push(top->right);   }
            top = top->left;
        }
    }
}
```

Обробляємо поточний вузол, за наявності правого піддерева додаємо його в стек для подальшої обробки. Переходимо до вузла лівого піддерева. Якщо лівого вузла немає, переходимо до верхнього вузла з стека.

Вертикальний зворотний обхід:

```
void passInOrder(TreeNode* top) {
     StackNode* TOS;
     createStack();
     while(top != NULL || !stack_isEmpty()) {
          if(!stack_isEmpty()) {
               top = stack_pop();
               /*...Необхідні операції над вузлом top...*/
               if (top->right != NULL) {
                    top = top->right;
               }
            else    {
                top = NULL;  }
          }
        while (top != NULL) {
            stack_push(top);
            top = top->left; }
```

```
     }
}
```

З поточного вузла «спускаємося» до самого нижнього лівого вузла, додаючи в стек всі відвідані вузли. Обробляємо верхній вузол з стека. Якщо в поточному вузлі мається праве піддерево, починаємо наступну ітерацію з правого вузла. Якщо правого вузла немає, пропускаємо крок зі спуском і переходимо до обробки наступного вузла з стека.

Вертикальний кінцевий обхід:

```
void passPostOrder(TreeNode* top) {
     StackNode* TOS;
     createStack();
     while (top != NULL || !stack_isEmpty()) {
          if(!stack_isEmpty()) {
               top = stack_pop();
            if (!stack_isEmpty() && top->right ==
stack_lastElement())    {
                    top = stack_pop();     }
               else {/*...Необхідні операції над вузлом top...*/
                 top = NULL;  }
         }
         while (top != NULL) {
               stack_push(top);
             if(top->right != NULL) {
                    stack_push(top->right);
                    stack_push(top);
               }
            top = top->left;  }
     }
}
```

Тут ситуація ускладнюється – на відміну від зворотного обходу, крім порядку спуску потрібно знати оброблено чи вже праве піддерево. Одним з варіантів вирішення є внесення в кожен екземпляр вузла прапора, який би зберігав відповідну інформацію (тут не розглядається). Іншим підходом є «кодування» безпосередньо в черговості стека – при спуску, якщо у чергового вузла пізніше потрібно буде обробити ще праве піддерево, в стек вноситься послідовність

«батько, правий вузол, батько». Таким чином, при обробці вузлів з стека ми зможемо визначити, чи потрібно нам обробляти праве піддерево.

18.6 Контрольні питання

1. Що називається динамічними структурами даних? Які структури є найпоширеними?
2. З чого складається елемент будь-якої динамічної структури?
3. Яку динамічну структуру називають зв'язаний списком? Які операції можна виконувати над зв'язаними списками? Які потрібні для цього покажчики?
4. Наведіть схеми та фрагменти програми для створення, перегляду, роздруку та пошуку заданого елемента у лінійному однозв'язаному списку.
5. Наведіть схеми та фрагменти програми для додавання елементу в початок лінійного однозв'язаного списку, додавання після заданого за значенням елемента, додавання перед заданим за значенням елементом, додавання перед заданим за індексом (номеру) елементом, додавання після заданого за індексом (номеру) елемента, додавання в кінець списку. Потрібно перевіряти наявність у списку заданих елементів.
6. Наведіть схеми та фрагменти програми для видалення елементу з початку лінійного однозв'язаного списку, видалення після заданого за значенням елемента, видалення перед заданим за значенням елементом, видалення перед заданим за індексом (номеру) елементом, видалення після заданого за індексом (номеру) елемента, видалення елемента з кінця списку. Потрібно перевіряти наявність у списку заданих елементів.
7. Які потрібні покажчики для роботи з кільцевими одно- та двозв'язним списками? Які операції можна виконувати над такими списками?
8. Виконайте усі завдання з пунктів 4, 5 та 6 для кільцевого однозв'язаного списку, для лінійного двозв'язного та для кільцевого двозв'язного списків.
9. Продемонструйте операцію сортування елементів однозв'язаного та двозв'язного списків.
10. Який принцип роботи стека? Що означає `LIFO`?
11. Які є типові операції над стеком і його елементами?
12. Наведіть схеми та фрагменти програми для створювання стека, додавання елемента в стек, видалення елемента зі стека. Для чого виконувати

перевірку чи є стек порожнім? Для чого виконувати перевірку чи не перевищує розмір стека заданого значення?

13. Який принцип роботи черги? Що означає `FIFO`?

14. Які є типові операції над чергою та її елементами?

15. Наведіть схеми та фрагменти програми для створювання черги, додавання елемента в хвіст черги, видалення елементи з голови черги, перевірці, чи порожня черга.

16. Що називається бінарним деревом? Який вид структур зручно описувати бінарними деревами? Наведіть графічну інтерпрітацію бінарного дерева.

17. Що називають коренем, вершиною, листом дерева?

18. Які існують типи обходу дерев? Чим вони відрізняються?

19. Які є варіанти вертикального обходу дерев?

20. Наведіть фрагменти програм для створення дерева, створення вузла, додавання вузла, горизонтального обходу дерева, вертикального прямого обходу, вертикального зворотнього обходу, вертикального кінцевого обходу дерева.

18.7 Тести для самоконтролю

1) Що з наступного не є назвою динамічної структури у С?

А) черга

Б) лінійний список

В) вектор

Г) стек

2) Який список називається двозв'язним?

А) такий, у якого два поля даних

Б) такий, у якого один покажчик вказує на попередній елемент, а другий – на наступний

В) такий, у якого є два покажчика

Г) такий, елементи якого вказуються на інший список

3) Який треба визначити покажчик для роботи з лінійним однозв'язним списком?

А) на його початок

Б) на його кінець

В) на його серединний елемент

Г) покажчики не потрібні

4) Що робить наступний фрагмент програми:

```
a = (Node*) malloc(sizeof(Node));
a->value = val;
a->next = NULL;
```

А) в існуючий лінійний однозв’язний список додається черговий елемент

Б) в існуючий лінійній двозв’язний список додається черговий елемент

В) створюється лінійній двозв’язний список, що містить один елемент

Г) створюється лінійний однозв’язний список, що містить один елемент

5) Куди вказують покажчики у кільцевому двозв’язаному списку?

А) покажчик першого елемента – на останній, покажчик останнього елемента нікуди не вказує

Б) покажчик останнього елемента – на перший, покажчик першого – на останній

В) покажчик останнього елемента – на перший, покажчик першого елемента нікуди не вказує

Г) обидва покажчики дорівнюють `NULL`

6) Що покладено в основу багатозв'язних списків?

А) одно- або двозв'язні списки

Б) масиви

В) прості змінні

Г) файли

7) Як виконується додавання та видалення елементів зі стеку?

А) додавання нових елементів проводиться з одного кінця, видалення – з іншого

Б) додавання та видалення елементів можна проводити з будь-якого кінця

В) додавання елементів проводиться у вершину стека, видалення – безпосередньо з вказанням порядкового номеру елемента

Г) додавання нових елементів і видалення існуючих проводиться з одного кінця

8) Який принцип роботи черги?

А) LIFO

Б) FILO

В) FIFO

Г) LILO

9) Яка модель даних називається бінарним деревом?

А) це послідовність об'єктів, кожен з яких може бути пов'язаний з двома наступними

Б) це послідовність об'єктів, кожен з яких може бути пов'язаний з одним наступним

В) те ж саме, що і багатозв’язний список

Г) немає правильної відповіді

10) Що з наступного не є варіантом вертикального обходу бінарного дерева?

А) прямий

Б) підпорядкований

В) зворотний

Г) кінцевий

РОЗДІЛ 19. СТВОРЕННЯ ТА ВИКОРИСТАННЯ БІБЛІОТЕК

Бібліотека в програмуванні (від англ. library) – збірка підпрограм або об'єктів, що використовуються для розробки програмного забезпечення (ПО).

З появою багатозадачних операційних систем (а може бути і раніше) виникла потреба використовувати той самий код в різних програмах. Як варіант можна було включати вихідний файл з необхідними функціями в кожну програму, але від цього зростали витрати оперативної пам'яті, та й розмір самої програми теж зростав. Тоді було вирішено робити бінарний код, на зразок самих програм, але на відміну від них код мав кілька точок входу і при цьому не міг самостійно запускатися. Назвали все це динамічними бібліотеками, так як вони зберігали вже відкомпільовані функції, які можна було використовувати в декількох програмах одночасно.

Іноді все ж було потрібно використовувати відкомпільований фрагмент програми саме в виконуваному файлі (зазвичай для зменшення часу компіляції, або коли не хотіли передавати код програми у відкритому вигляді). Такі бібліотеки отримали назву статичних. Зараз же практично всі операційні системи використовують бібліотеки. Також у бібліотеках поширюється безліч серйозних програмних продуктів, що дозволяють програмісту без особливих проблем вирішувати якісь завдання (наприклад, бібліотеки графічних і фізичних движків, бібліотеки роботи зі звуком та ін).

19.1 Динамічні бібліотеки

Динамічна бібліотека – це частина основної програми, яка завантажується в ОС за запитом працюючої програми в ході її виконання (Run-time), тобто динамічно (Dynamic Link Library, DLL в Windows). Той самий набір функцій (підпрограм) може бути використаний відразу в декількох працюючих програмах, через що вони мають ще одну назву – бібліотеки загального користування (Shared Library). Якщо динамічна бібліотека завантажена в адресний простір самої ОС (System Library), то єдина копія може бути використана безліччю працюючих з нею програм, що позитивно позначається на ступені використання ресурсу ОЗУ. Динамічні бібліотеки можуть містити в собі як критичні для роботи програми частини, так і додаткові функції. Додатковим плюсом такого підходу є те, що

динамічна бібліотека може бути використана в якості плагіна (Plug-ins), що розширює функціональність програми. Мінусом є те, що у випадку, якщо модуль, який містить в собі критичну частину, відсутній, програма не зможе продовжити роботу.

Динамічні бібліотеки зберігаються зазвичай у визначеному місці і мають стандартне розширення. Наприклад, файли .library в логічному томі Libs: у AmigaOS; в Microsoft Windows і OS/2 файли бібліотек загального користування мають розширення .dll; в UNIX-подібних ОС – зазвичай .so; в Mac OS – .dylib.

При написанні програми програмісту досить вказати транслятору (компілятору або інтерпретатору) мови програмування, що слід підключити потрібну бібліотеку і використовувати функцію з неї. Ні вихідний текст, ні виконуваний код функції до складу програми на даному етапі не входить.

19.2 Статичні бібліотеки

Статичні бібліотеки можуть бути представлені у вигляді вихідного тексту, що підключається програмістом до своєї програми на етапі написання (наприклад, для мови Fortran існує величезна кількість бібліотек для вирішення різних завдань саме в початкових текстах), або у вигляді об'єктних файлів, що приєднуються (лінкуються) до виконуваної програмі на етапі компіляції (у Microsoft Windows такі файли мають розширення .lib, в UNIX-подібних ОС – зазвичай .a). В результаті програма включає в себе всі необхідні функції, що робить її автономною, але збільшує розмір. Без статичних бібліотек об'єктних модулів (файлів) неможливе використання більшості сучасних компілючих мов і систем програмування: Fortran, Pascal, C, C++ та інших.

19.3 Створення і використання статичної бібліотеки в Visual Studio

Статичні бібліотеки є хорошим способом повторного використання коду. Замість того щоб кожного разу реалізовувати ті самі ж підпрограми в кожному створюваному додатку, їх можна створити один раз і потім викликати з додатків для забезпечення відповідної функціональності.

Створення проекту статичної бібліотеки в Visual Studio 2013:

1. У меню FILE (Файл) виберіть пункт New project… (Новий Проект).

2. У вузлі Visual C + + області Templates (Шаблони) виберіть «Win32».

3. Виберіть Win32 Console Application (Консольне програма Win32).

4. Виберіть ім'я для проекту, наприклад StaticLib, і введіть його в полі «Name» («Ім'я»). Виберіть ім'я для рішення, наприклад StaticLibrary, і введіть його в полі «Solution name» («Ім'я рішення»).

5. Для запуску майстра додатків Win32 натисніть кнопку "ОК". На сторінці «Overview» («Загальні відомості») діалогового вікна «Win32 Application Wizard» («Майстер програм Win32») натисніть кнопку «Next» («Далі»).

6. На сторінці «Application Settings» («Параметри програми») діалогового вікна «Майстер додатків Win32» в полі «Application type» («Тип програми») виберіть пункт «Static library» («Статична бібліотека»).

7. На сторінці «Application Settings» («Параметри програми») діалогового вікна «Майстер додатків Win32» в полі «Additional options» («Додаткові параметри») зніміть прапорець «Precompiled header» («Попередньо скомпільований заголовок»).

8. Щоб створити проект, натисніть кнопку «Finish» («Готово»).

Додавання структури в статичну бібліотеку:

1. У меню PROJECT (Проект) виберіть команду «Add New Item…» («Додати новий елемент»). Відкриється діалогове вікно Додавання нового елемента. У вузлі Visual C++ виберіть пункт «Code» («Код»). Виберіть пункт «Header File»(«Заголовний файл»). Виберіть ім'я для файлу заголовка, наприклад StatLib.h, і натисніть кнопку «Add» («Додати»). Відобразиться порожній файл.

Як приклад додайте структуру з ім'ям Tree і дві найпростіші функції по роботі зі структурами – пошук максимального значення і сортування за зростанням. Для цього додайте у файл `StatLib.h` наступний код:

```
//StatLib.h

struct Tree
{
     char name[20]; //назва дерева
     double age;    //вік
     int leavesCount;//кількість листків
};
```

```
Tree max(Tree tree[], int count); //функція для пошуку найстаршого
дерева
void sort(Tree tree[], int count);//функция сортування дерев
// за зростанням числа листків на кожному з них
```

2. Для створення вихідного файлу бібліотеки в меню PROJECT (Проект) виберіть команду «Add New Item…» («Додати новий елемент»). Відкриється діалогове вікно Додавання нового елемента. У вузлі Visual C++ виберіть пункт «Code» («Код»). Виберіть пункт «C++ File» («Файл C++»). Виберіть ім'я для вихідного файлу, наприклад `StatLib.cpp`, і натисніть кнопку «Add» («Додати»). Відобразиться порожній файл.

3. Реалізуйте функції у вихідному файлі. Для цього додайте у файл `StatLib.cpp` наступний код:

```
//StatLib.cpp
#include"StatLib.h"
Tree max(Tree tree[], int count)
{ //функція для пошуку найстаршого дерева
     Tree max = tree[0]; //присвоюємо тимчасовій змінній
//значення першої комірки масиву
     for(int i=1; i<count; i++)     //цикл від 2 значення до кінця
          if(tree[i].age > max.age){ //якщо поточне деревао має вік
  //більше ніж вік дерева в тимчасовій змінній max
          max = tree[i];//зберігаємо у тимчасовій змінній дерево
          }
     return MAX;     //повертаємо значення тимчасової змінної
}

void sort(Tree tree[],int count){  // сортування дерев за зростанням
     //числа листків на кожному з них
     Tree buffer;
     for(int i=0; i<count; i++){    //повторювати стільки разів,
                                    //скільки комірок у масиві
          for(int j=0; j<count-1; j++){ //прохід по массиву від
          //першого до передпостаннього
          if(tree[j].leavesCount > tree[j+1]. leavesCount)   {
//якщо у поточного дерева більше листків
//ніж у наступного – міняємо їх місцями
```

```
            buffer = tree[j];
            tree[j] = tree[j+1];
            tree[j+1] = buffer; }
            }
      }
}
```

4. Щоб вбудувати проект в статистичну бібліотеку, виберіть у меню PROJECT (Проект) пункт «Properties» («Властивості»). У лівій області в полі «Configuration properties» («Властивості конфігурації») виберіть «General» («Загальні»). У правій області в полі «Configuration type» («Тип конфігурації») виберіть «Static Library (.lib)» («Статична бібліотека (.lib)») (якщо вона там ще не вибрана). Натисніть кнопку «ОК» для збереження змін.

5. Скомпілюйте статичну бібліотеку, вибравши команду «Build solution» («Побудувати рішення») в меню «BUILD» («Побудова»). У результаті буде створена статична бібліотека, яка може використовуватися іншими програмами.

Створення консольної програми, що посилається на статичну бібліотеку:

1. Щоб створити програму, яка буде використовувати створену раніше статичну бібліотеку і посилатися на неї, в меню меню FILE (Файл) виберіть пункт New project… (Новий Проект).

2. У вузлі Visual C + + області Templates (Шаблони) виберіть «Win32».

3. Виберіть Win32 Console Application (Консольне програма Win32).

4. Виберіть ім'я для проекту, наприклад Using_static_library, і введіть його в полі «Name» («Ім'я»). У списку, поряд з полем «Solution» («Рішення») виберіть пункт «Add to solution» («Додати до рішення»). Після цього новий проект буде додано в те ж рішення, що і статична бібліотека.

5. Для запуску майстра додатків Win32 натисніть кнопку "ОК". На сторінці «Overview» («Загальні відомості») діалогового вікна «Win32 Application Wizard» («Майстер програм Win32») натисніть кнопку «Next» («Далі»).

6. На сторінці «Application Settings» («Параметри програми») діалогового вікна «Майстер додатків Win32» в полі «Application type» («Тип програми») виберіть пункт «Console application» («Консольна програма»).

7. На сторінці «Application Settings» («Параметри програми») діалогового вікна «Майстер додатків Win32» в полі «Additional options» («Додаткові

параметри») зніміть прапорець «Precompiled header» («Попередньо скомпільований заголовок»).

8. Щоб створити проект, натисніть кнопку «Finish» («Готово»).

Використання функціональних можливостей статичної бібліотеки у програмі:

1. Після створення консольної програми майстер створить порожню програму. Ім'я вихідного файлу буде збігатися з ім'ям, вибраним раніше для проекту. У цьому прикладі він має ім'я Using_static_library.cpp.

2. Для використання розроблених функцій з статичної бібліотеки необхідно послатися на цю бібліотеку. Для цього в меню PROJECT (Проект) виберіть пункт «References» («Посилання»). У діалоговому вікні «Property Pages» («Сторінки властивостей») виберіть команду «Add new reference» («Додати нове посилання»).

3. З'явиться діалогове вікно «Add Reference» («Додавання посилання»). На вкладці «Projects» («Проекти») перераховуються всі проекти поточного рішення і бібліотеки, на які можна створити посилання. Виберіть проект StaticLib і натисніть кнопку "ОК".

4. Для створення посилання на файл заголовка StatLib.h необхідно змінити шлях до каталогів включення. Для цього в діалоговому вікні ««Property Pages» («Сторінки властивостей») послідовно розгорніть вузли «Configuration properties» («Властивості конфігурації»), «C/C++», а потім виберіть «General» («Загальні»). У полі значення властивості «Additional include directories» («Додаткові каталоги включення») введіть шлях до каталогу StaticLibrary або знайдіть цей каталог. Щоб знайти шлях до каталогу, у списку значень властивостей клацніть «Edit» («Змінити»). У текстовому полі діалогового вікна «Додаткові каталоги включення файл» виберіть порожній рядок і натисніть кнопку з трьома крапками (...) наприкінці рядка. У діалоговому вікні «Вибір каталогу» виберіть каталог StaticLib і натисніть кнопку «Вибір папки», щоб зберегти обрані значення і закрити діалогове вікно. У діалоговому вікні «Додаткові каталоги включення файл» натисніть кнопку «ОК».

5. Тепер змінні структури Tree можна використовувати в додатку. Для цього замініть вміст файлу Using_static_library.cpp наступним кодом.

```
//Using_static_library.cpp
#include "StaticLib\StatLib.h"
#include <stdio.h>
```

```
void main() {
     Tree array [10];
     for(int i=0; i<10; i++) {
          printf(“Input name of tree”);
          scanf(“%s”, &array[i].name);
          printf(“Input age of tree:”);
          scanf(“%d”, &array[i].age);
          printf(“Input tree leaves count”);
          scanf(“%d”, &array[i].leavesCount);
     }
     do {
          printf(“1)Sort trees\n2)Search oldest\n3)View
all\n4)Exit”);
          char key;
          scanf(“%c”, &key);
          switch(key) {
               case'1': {
                    sort(array, 10);
                    break;
               }
               case'2': {
                    max(array, 10);
                    break;
               }
               case'3': {
                    for(int i=0;i<10;++i) {
                         printf(“Name of tree:%s\n”, array[i].name);
                         printf(“Age of tree:%d\n”, array[i].age);
                         printf(“Count of tree leaves:”,
   array[i].leavesCount);
                         printf(“\n-------------------\n”);
                    }
               }
               case'4': {
                    return;
               }
```

```
        }
    } while(true);
}
```

6. Побудуйте виконуваний файл, вибравши команду «Build solution» («Побудувати рішення») в меню «BUILD» («Побудова»).

Запуск програми:

1. Переконайтеся, що проект Using_static_library обраний як проект за замовчуванням. У вікні «Solution Explorer» («Оглядач рішень») виберіть проект Using_static_library, а потім у меню PROJECT (Проект) виберіть команду «Set as StartUp Project» («Призначити проектом, що запускається»).

2. Щоб запустити проект, в меню DEBUG (Відладка) виберіть команду «Start without debugging» («Запуск без відладки»).

19.4 Створення і використання динамічної бібліотеки в Visual Studio

Створимо бібліотеку динамічного компонування (DLL). Бібліотеки DLL є хорошим способом повторного використання коду. Замість того щоб кожного разу реалізовувати ті самі ж підпрограми в кожній створюваній програмі, їх можна створити один раз і потім викликати з додатків для забезпечення відповідної функціональності.

Створення проекту бібліотеки динамічного компонування (DLL) в Visual Studio 2013:

1. У меню FILE (Файл) виберіть пункт New project… (Новий Проект).

2. У вузлі Visual C + + області Templates (Шаблони) виберіть «Win32».

3. Виберіть Win32 Console Application (Консольне програма Win32).

4. Виберіть ім'я для проекту, наприклад MathFuncsDll, і введіть його в полі «Name» («Ім'я»). Виберіть ім'я для рішення, наприклад DynamicLibrary, і введіть його в полі «Solution name» («Ім'я рішення»).

5. Для запуску майстра додатків Win32 натисніть кнопку "ОК". На сторінці «Overview» («Загальні відомості») діалогового вікна «Win32 Application Wizard» («Майстер програм Win32») натисніть кнопку «Next» («Далі»).

6. На сторінці «Application Settings» («Параметри програми») діалогового вікна «Майстер додатків Win32» в полі «Application type» («Тип програми») виберіть пункт «DLL».

7. На сторінці «Application Settings» («Параметри програми») діалогового вікна «Майстер додатків Win32» в полі «Additional options» («Додаткові параметри») поставте прапорець «Empty project» («Порожній проект»).

8. Щоб створити проект, натисніть кнопку «Finish» («Готово»).

Додавання функцій в бібліотеку динамічного компонування:

1. У меню PROJECT (Проект) виберіть команду «Add New Item…» («Додати новий елемент»). Відкриється діалогове вікно Додавання нового елемента. У вузлі Visual C++ виберіть пункт «Code» («Код»). Виберіть пункт «Header File» («Заголовний файл»). Виберіть ім'я для файлу заголовка, наприклад MathFuncsDll.h, і натисніть кнопку «Add» («Додати»).

2. Додайте наступний код в початок файлу заголовка:

```
// MathFuncsDll.h
#ifdef MATHFUNCSDLL_EXPORTS
#define MATHFUNCSDLL_API __declspec(dllexport)
#else
#define MATHFUNCSDLL_API __declspec(dllimport)
#endif
```

3. Додайте прототипи функцій у заголовний файл:

```
MATHFUNCSDLL_API double add(double a, double b);
MATHFUNCSDLL_API double subtract(double a, double b);
MATHFUNCSDLL_API double multiply(double a, double b);
MATHFUNCSDLL_API double divide(double a, double b);
```

Коли символ `MATHFUNCSDLL_EXPORTS` визначений, символ `MATHFUNCSDLL_API` встановить модифікатор `__ declspec (dllexport)` в оголошеннях функції в цьому коді. Цей модифікатор дозволяє функції повинні експортуватися DLL таким чином, щоб він міг використовуватися іншими додатками. При `MATHFUNCSDLL_EXPORTS` не вказано, `MATHFUNCSDLL_API` визначає модифікатор `__ declspec (dllimport)` в оголошеннях функції-члена. Цей модифікатор дозволяє компілятору оптимізувати імпорт функції з бібліотеки DLL

для використання в інших додатках. Типово `MATHFUNCSDLL_EXPORTS` визначається при побудові проекту `MathFuncsDll`.

4. У проекті `MathFuncsDll` за допомогою Оглядача рішень в папці Вихідні файли відкрийте `MathFuncsDll.cpp`.

5. Реалізуйте функції у вихідному файлі. Код повинен виглядати наступним чином:

```
//MathFuncsDll.cpp
#include "MathFuncsDll.h"
double add(double a, double b) {
        return a + b;
}
double subtract(double a, double b) {
        return a - b;
}
double multiply(double a, double b) {
         return a * b;
}
double divide(double a, double b) {
        return a / b;
}
```

6. Скомпілюйте бібліотеку динамічного компонування (DLL), вибравши «BUILD» («Побудова») – «Build Solution» («Побудувати рішення») в рядку меню.

Створення консольної програми, що посилається на бібліотеку динамічного компонування:

1. Щоб створити програму, яка буде використовувати створену раніше динамічну бібліотеку і посилатися на неї, в меню FILE (Файл) виберіть пункт New project… (Новий Проект).

2. У вузлі Visual C + + області Templates (Шаблони) виберіть «Win32».

3. Виберіть Win32 Console Application (Консольне програма Win32).

4. Виберіть ім'я для проекту, наприклад MyExecRefsDll, і введіть його в полі «Name» («Ім'я»). У списку, поряд з полем «Solution» («Рішення») виберіть пункт «Add to solution» («Додати до рішення»). Після цього новий проект буде додано в те ж рішення, що і статична бібліотека.

5. Для запуску майстра додатків Win32 натисніть кнопку "ОК". На сторінці «Overview» («Загальні відомості») діалогового вікна «Win32 Application Wizard» («Майстер програм Win32») натисніть кнопку «Next» («Далі»).

6. На сторінці «Application Settings» («Параметри програми») діалогового вікна «Майстер додатків Win32» в полі «Application type» («Тип програми») виберіть пункт «Console application» («Консольна програма»).

7. На сторінці «Application Settings» («Параметри програми») діалогового вікна «Майстер додатків Win32» в полі «Additional options» («Додаткові параметри») зніміть прапорець «Precompiled header» («Попередньо скомпільований заголовок»).

8. Щоб створити проект, натисніть кнопку «Finish» («Готово»).

Використання функціональних можливостей динамічної бібліотеки у програмі :

1. По завершенні створення консольної програми буде створена порожня програма. Ім'я вихідного файлу буде збігатися з ім'ям, вибраним раніше для проекту. У цьому прикладі він має ім'я MyExecRefsDll.cpp.

2. Для використання математичних процедур з бібліотеки динамічного компонування необхідно послатися на цю бібліотеку. Для цього в оглядачі рішень виберіть проект MyExecRefsDll і потім виберіть «References» («Посилання...») у меню «PROJECT» («Проект»). У діалоговому вікні Сторінки властивостей розгорніть вузол «Common Properties» («Загальні властивості»), виберіть пункт «References» («Посилання») і натисніть кнопку «Add New Reference» («Додати нове посилання»).

3. З'явиться діалогове вікно «Add Reference» («Додавання посилання»). У цьому діалоговому вікні відображається список всіх бібліотек, на які можна посилатися. На вкладці «Project» («Проект») перераховуються всі проекти поточного рішення і включені в них бібліотеки. На вкладці Проекти виберіть MathFuncsDll. Потім натисніть кнопку ОК.

4. Для створення посилання на файл заголовка MathfuncsDll.h необхідно змінити шлях до каталогів включення. Для цього в діалоговому вікні ««Property Pages» («Сторінки властивостей») послідовно розгорніть вузли «Configuration properties» («Властивості конфігурації»), «C/C++», а потім виберіть «General» («Загальні»). У полі значення властивості «Additional include directories» («Додаткові каталоги включення») введіть шлях до каталогу MathFuncsDll або

знайдіть цей каталог. Щоб знайти шлях до каталогу, у списку значень властивостей клацніть «Edit» («Змінити»). У текстовому полі діалогового вікна «Додаткові каталоги включення файл» виберіть порожній рядок і натисніть кнопку з трьома крапками (...) наприкінці рядка. У діалоговому вікні «Вибір каталогу» виберіть каталог MathFuncsDll і натисніть кнопку «Вибір папки», щоб зберегти обрані значення і закрити діалогове вікно. У діалоговому вікні «Додаткові каталоги включення файл» натисніть кнопку «ОК».

5. Виконуваний файл не завантажує бібліотеки динамічного компонування під час виконання. Необхідно вказати системі місце для пошуку бібліотеки MathFuncsDll.dll. Це можна зробити за допомогою змінної середовища PATH. Для цього в діалоговому вікні Вікна властивостей розгорніть вузол «Configuration properties» («Властивості конфігурації»), а потім виберіть «Debugging» («Відладка»). У полі середу введіть наступний рядок:

```
PATH = <шлях_до_файлу_MathFuncsDll.dll>,
```

де замість <шлях_до_файлу_MathFuncsDll.dll> необхідно підставити фактичне розташування бібліотеки `MathFuncsDll.dll`. Натисніть кнопку ОК для збереження всіх змін.

6. Тепер функції динамічної бібліотеки можна використовувати в додатку. Замініть код у файлі MyExecRefsDll.cpp наступним кодом:

```
// MyExecRefsDll.cpp
// compile with: /EHsc /link MathFuncsDll.lib
#include “MathFuncsDll.h”
#include “stdio.h”
int main() {
     double a = 42.01;
     double b = 3.14;
     printf(“a + b = %lf\n”, add(a, b));
     printf(“a - b = %lf\n”, subtract(a, b));
     printf(“a * b = %lf\n”, multiply(a, b));
     printf(“a / b = %lf\n”, divide(a, b));
     return 0;
}
```

7. Побудуйте виконуваний файл, вибравши команду «Build Solution» («Побудувати рішення») в меню «BUILD» («Побудова»).

Запуск програми:

1. Переконайтеся, що проект MyExecRefsDll обраний як проект за замовчуванням. У вікні «Solution Explorer» («Оглядач рішень») виберіть проект MyExecRefsDll, а потім у меню PROJECT (Проект) виберіть команду «Set as StartUp Project» («Призначити проектом, що запускається»).

2. Щоб запустити проект, в меню DEBUG (Відладка) виберіть команду «Start without debugging» («Запуск без відладки»).

19.5 Контрольні питання

1. Що в програмуванні називають бібліотекою?
2. Чим статичні бібліотеки відрізняються від динамічних?
3. Як завантажуються в ОС динамічні бібліотеки? Що вони зазвичай містять?
4. Як підключаються до програми статичні бібліотеки? Що при цьому стає з розміром програми?
5. Як у Visual Studio C++ 2010 створити проект статичної бібліотеки?
6. Як додати структуру в статичну бібліотеку?
7. Як створити консольну програму, що посилається на статичну бібліотеку?
8. Як використати функціональні можливості статичної бібліотеки у програмі та запустити програму? Наведіть приклад.
9. Як створити проект бібліотеки динамічного компонування?
10. Що потрібно зробити для додавання функцій в бібліотеку динамічного компонування?
11. Як створити консольну програму, що посилається на динамічну бібліотеку?
12. Як використати функціональні можливості динамічної бібліотеки у програмі та запустити програму? Наведіть приклад.

19.6 Тести для самоконтролю

1) Що у програмуванні називають бібліотекою?

А) збірку підпрограм або об'єктів

Б) середу розробки програмного забезпечення

В) версію компілятора

Г) папку для зберігання проекту

2) Що є статичною бібліотекою?

А) відкомпільований фрагмент програми, який підключається у разі необхідності

Б) відкомпільований фрагмент програми, який міститься в самому виконуваному файлі

В) функції, що працюють тільки у старих версіях середи програмування

Г) немає правильної відповіді

3) Що є динамічною бібліотекою?

А) бінарний код, що має одну точку входу

Б) бінарний код, що має кілька точок входу

В) текстовий файл з описами функцій

Г) буфер пам’яті

4) Яке розширення має статична бібліотека у Visual Studio C++?

А) `h`

Б) `cpp`

В) `lib`

Г) `static`

5) Як у Visual Studio 2010 створити статичну бібліотеку?

А) обрати меню «Бібліотеки»

Б) у області «Шаблони» вибрати «Статична бібліотека»

В) почати ім'я для рішення з префіксу static_library

Г) на сторінці «Параметри програми» в полі «Тип програми» вибрати пункт «Статична бібліотека»

6) Як вбудувати створений проект в статистичну бібліотеку?

А) підключити заголовний файл <static.lib>

Б) це виходить за замовчуванням, спеціальних дій не треба

В) у меню «Проект» у пункті «Властивості» обрати тип конфігурації «Статична бібліотека»

Г) немає правильної відповіді

7) Яке розширення має динамічна бібліотека у Visual Studio C++?

А) dll

Б) lib

В) dynamic

Г) не має розширення

8) Як може використовуватись динамічна бібліотека?

А) по одному разу кожною програмою

Б) один раз

В) багато разів різними програмами

Г) немає правильної відповіді

9) Як додати функції в динамічну бібліотеку?

А) створити файл заголовку через Проект, Додавання нового елемента та відповідного .cpp-файла з реалізаціями функцій

Б) створити тільки .cpp-файл з реалізаціями функцій

В) створити тільки файл заголовку

Г) немає правильної відповіді

10) Як використовувати функціональні можливості динамічної бібліотеки?

А) розмістити її в тій же папці, що і проект

Б) звичайним викликом функцій бібліотеки

В) це залежить від семантики функцій

Г) посилатися на цю бібліотеку за допомогою вибору Посилання ... у меню Проект

ДОДАТОК А. РОБОТА З ІНТЕГРОВАНИМИ СЕРЕДОВИЩАМИ РОЗРОБКИ

DEV-C++

Dev-C++ — це повнофункціональне графічне інтегроване середовище розробки (англ. IDE, integrated development environment), тобто система програмних засобів, використовувана програмістами для розробки програмного забезпечення. Dev-C++ здатне створювати Windows або консольні C/C++ програми за допомогою системи компілятора MinGW. MinGW (Minimalist GNU для Windows) використовує GCC – GNU compiler collection (колекція компілятора GNU g++).

Редактор вихідного коду − текстовий редактор для створення і редагування вихідного коду програм. Він може бути окремим застосуванням, або вбудований в інтегроване середовище розробки.

Програмний проект може розглядатися як контейнер, який використовується для зберігання всіх елементів, необхідних для складання програми. Розглянемо основні режими роботи з проектом.

Створення проекту

1. Перейдіть до меню «File» (Файл) – «New» (Новий) – «Project...» (Проект).

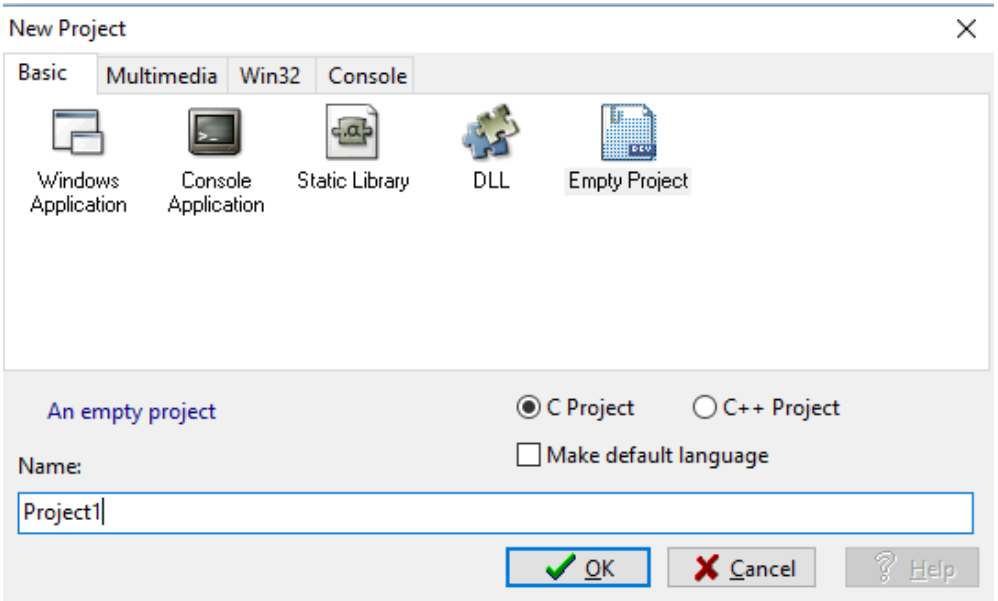


Рисунок А.1 – Вікно для створення проекту

2. Виберіть "Empty Project" та переконайтесь, що вибрано "C project". Тут ви також дасте назву своєму проекту. Ви можете надати своєму проекту будь-яке дійсне ім’я файлу, але пам’ятайте, що ім’я вашого проекту також буде іменем остаточного виконуваного файлу.

3. Натисніть “OK”.

4. Оберіть папку для збереження вашого проекту.

Відкриття створеного проекту

1. Перейдіть до меню "File"(Файл) і виберіть "Open"(Відкрити).

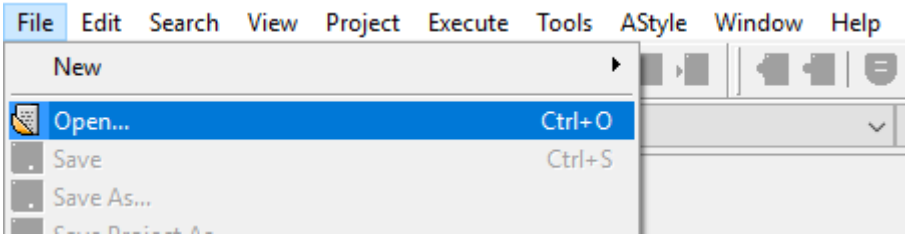


Рисунок А.2 – Вікно для відкриття проекту

2. Оберіть проектний файл з розширенням “.dev”

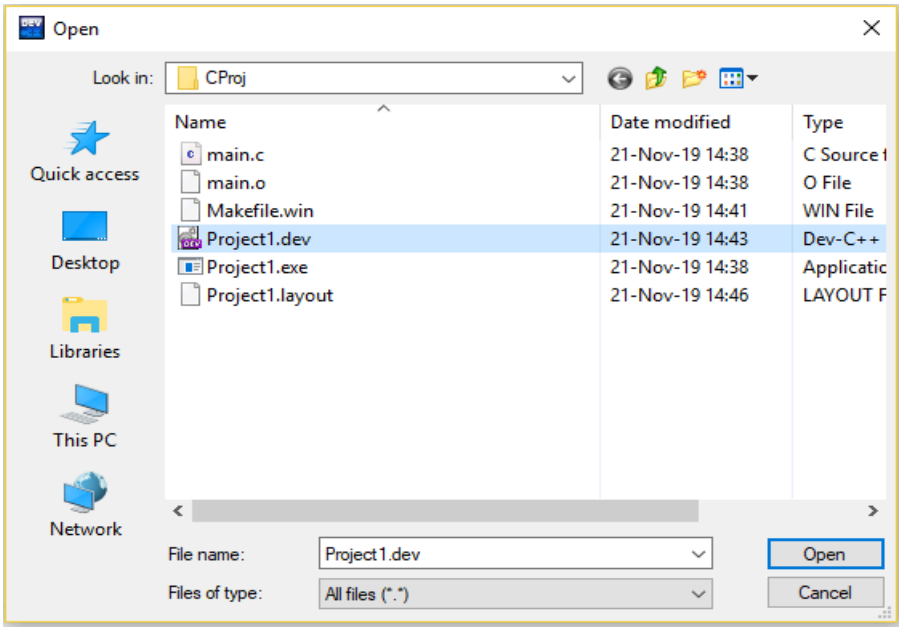


Рисунок А.3 – Вікно для вибору файлу проекту

Налагодження програми

Метою налагодження є усунення помилок у програмі. Спеціальні засоби налагодження спрямовані на усунення семантичних (смислових) помилок, при

яких проект може бути правильно скомпільований і побудований, тому що синтаксичні помилки (неправильне використання мовних конструкцій) легко усунути на етапі компіляції.

Існує два спеціальні засоби налагодження програми:

– використання покрокового режиму;

– використання точок зупину.

При використанні покрокового режиму команди виконуються послідовно одна за одною. З'являється можливість простежити шлях виконання програми і відстежити, наприклад, такі випадки:

– у місці розгалуження управління було передано не на ту гілку – потрібно проаналізувати умову;

– не був здійснений вхід в цикл – потрібно перевірити початкове значення змінної циклу і умова виходу;

– не було пройдено потрібне число ітерацій циклу – потрібно перевірити умову виходу з циклу і т.д.

Крім того, при покроковому налагоджені можна на кожному кроці переглядати значення змінних, і таким чином виявити місце помилки.

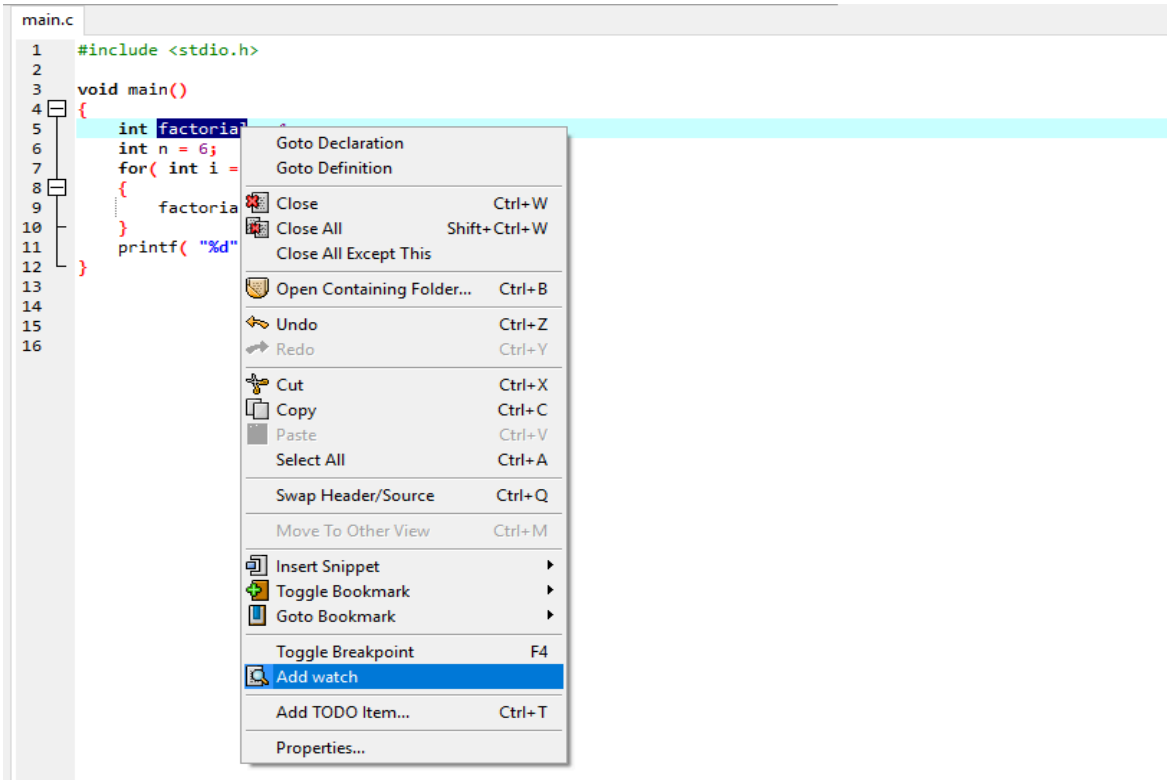


Рисунок А.4 – Вікно для перегляду змінних

Щоб налагоджувати програму в Dev-C++, оберіть змінну для відстежування. Обрана для відстежування змінна буде відображатись у панелі Debug. Для того щоб створити точки зупину, натисніть на номер рядка. Рядки, в яких стоять точки зупину, будуть виділені червоним кольором.

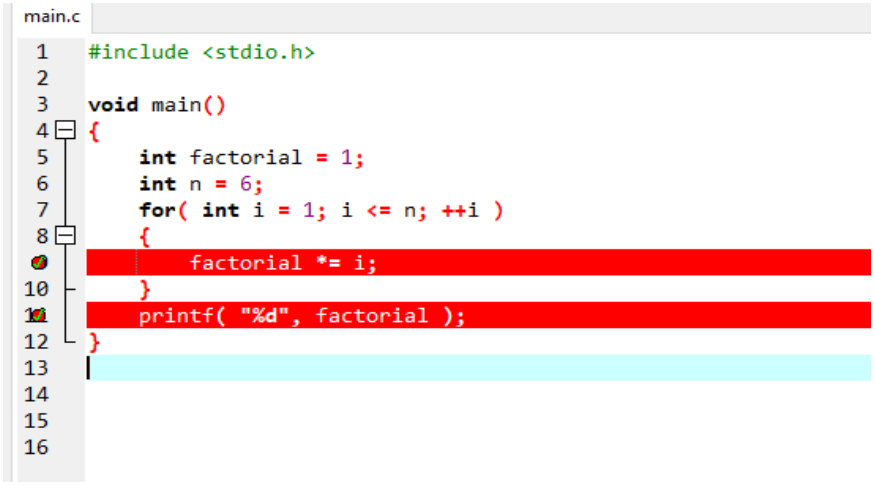


Рисунок А.5 – Вікно для створення точок зупину

Перейдіть у меню Debug та натисніть кнопку “Debug” щоб розпочати виконання програми. Програма призупиниться на першій точці зупину ПЕРЕД виконанням інструкції записаної у поточному рядку. Точка зупину, на якій зупинилась програма, буде виділена синім кольором. У лівій панелі Debug будуть відображені поточні значення для всіх відстежуваних змінних.

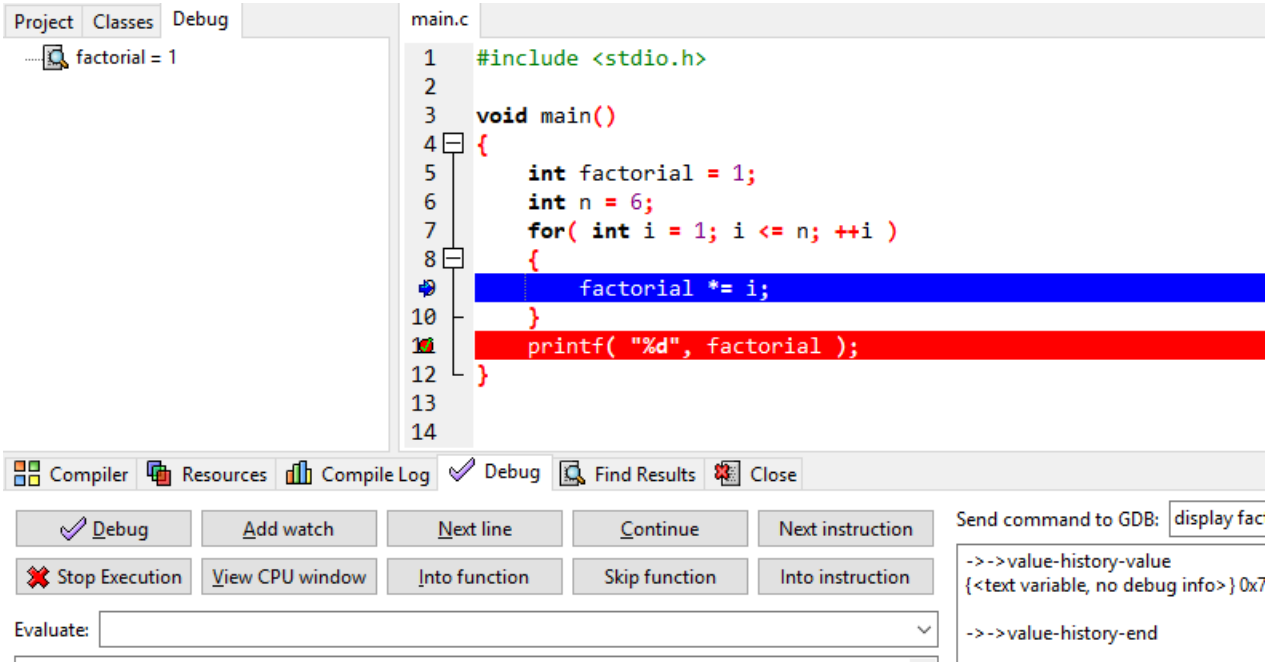


Рисунок А.6 – Вікно для відлагодження програми

Щоб перейти до наступних інструкцій у покроковому режимі, натисніть на кнопку “Next line”. Для переходу до наступної точки зупину натисніть кнопку “Continue”.

ECLIPSE

Eclipse — вільне модульне інтегроване середовище розробки програмного забезпечення. Розробляється і підтримується Eclipse Foundation і включає проекти, такі як платформа Eclipse, набір інструментів для програмістів, системи контролю версій, конструктори GUI тощо. Написаний в основному на Java, може бути використаний для розробки застосувань на Java і, за допомогою різних плагінів, на інших мовах програмування, включаючи Ada, C, C++, COBOL, Fortran, Perl, PHP, Python, R, Ruby (включно з каркасом Ruby on Rails), Scala, Clojure та Scheme.

Випущена на умовах Eclipse Public License, Eclipse є вільним програмним забезпеченням.

Початок роботи з Eclipse

Перше, що ми бачимо, після запуску програми – стартове вікно.

Рисунок А.7 – Стартове вікно Eclipse

Можна перейти в меню (1) «Tutorial: Import an existing project» (Навчальний посібник: Імпорт існуючого проекту) та отримати поради щодо імпорту існуючого проекту.

Меню (2) “Review IDE configuration settings” (Перегляд налаштування конфігурації IDE) дозволяє перейти у вікно налаштувань Eclipse.

Для швидкого створення проекту перейдіть в меню (3) «Create a new C/C++ project» (Створити новий C/C++ проект).

Щоб імпортувати існуючий проект з працюючим мейкфайлом, перейдіть в меню (4) «Import a project with a working Makefile» (Імпрот проекту з працюючим мейкфайлом).

Якщо ви хочете налаштувати відстежування Git проекту перейдіть в меню (5) «Checkout project from Git» (Відстеження проекту з Git).

Для того щоб імпортувати існуючий Eclipse проект з файлової системи або архіву перейдіть у меню (6) «Import existing projects» (Імпорт існуючого проекту).

Ви можете перейти в меню (7) «Overview» (Перегляд) та переглянути функції Eclipse середовища.

Ви можете перейти в меню (8) «Tutorials» (Посібник) та отримати поради щодо функцій Eclipse середовища.

У меню (9) «Samples» (Приклади) ви можете переглянути зразки проектів створених в Eclipse.

Якщо ви хочете переглянути новини про розробку середовища Eclipse, а також ознайомитись із новими інструментами, що пропонує Eclipse, перейдіть у меню (10) «What’s New» (Що нового?).

Створення проекту

Для кожної програми на C вам потрібно створити проект, що зберігає всі вихідні коди, виконувані файли та відповідні ресурси.

1. Виберіть меню «File» (Файл) – «New» (Новий) – «Project...» (Проект) (рис. А.8).
2. У діалоговому вікні «New Project» (Новий проект) оберіть тип C/C++ - C Project (Проект C) (рис. А.9).

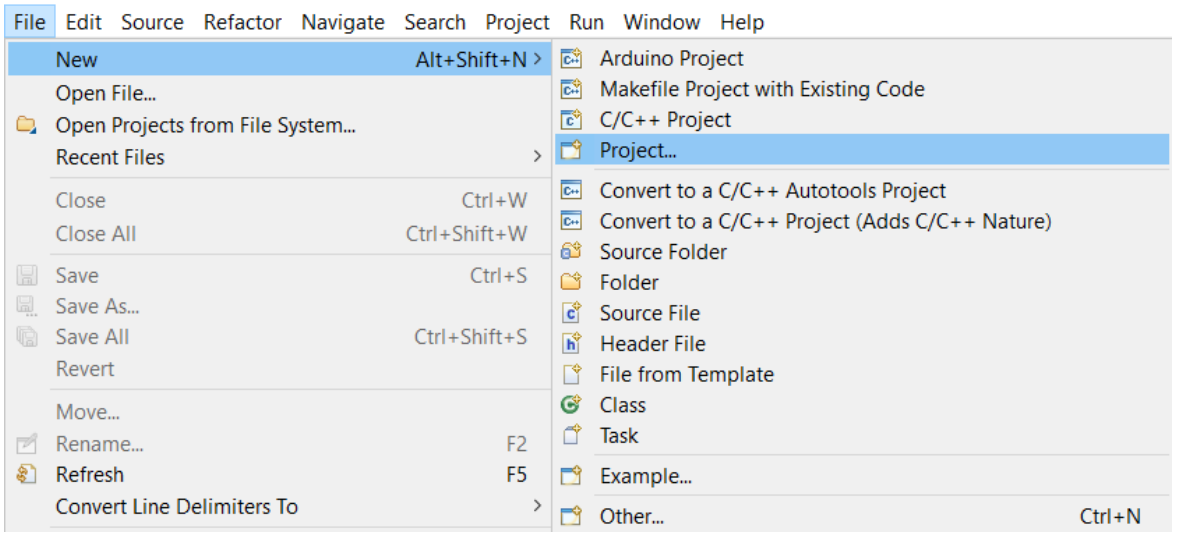


Рисунок А.8 – Створення проекту Eclipse

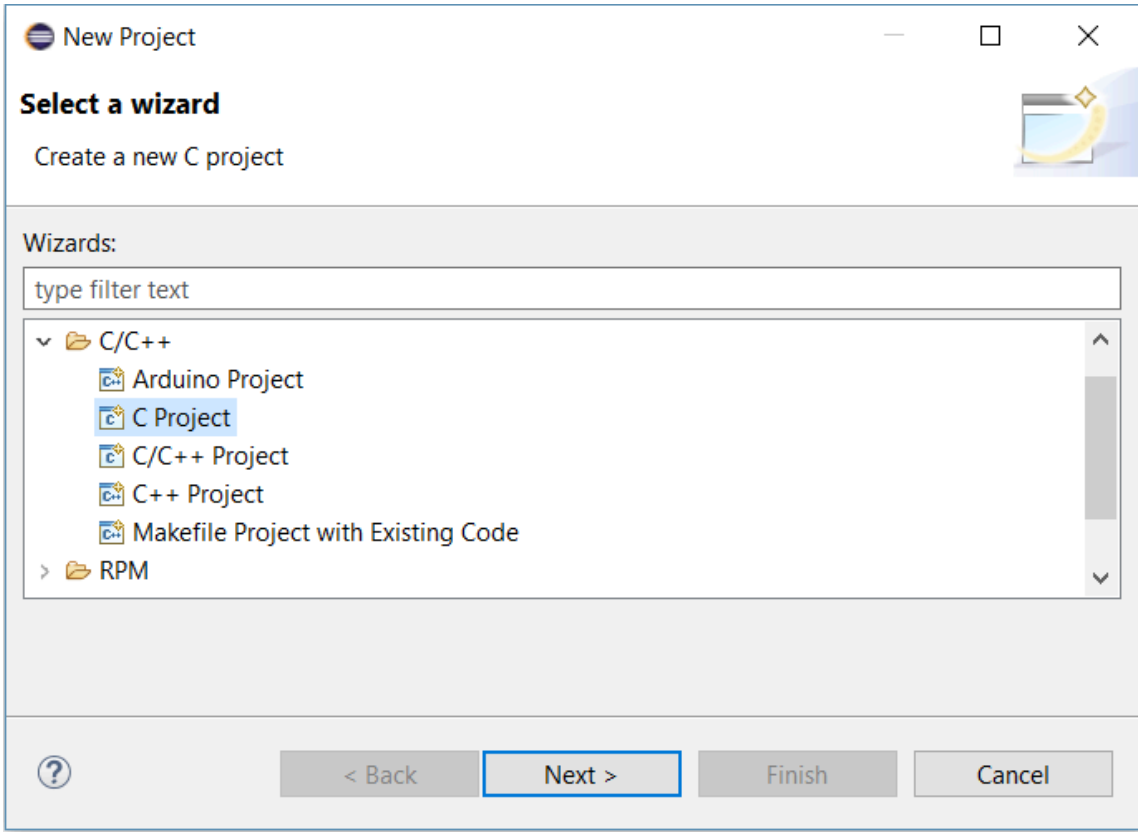


Рисунок А.9 – Вибір типу проекту Eclipse

3. У діалоговому вікні «C Project» (Проект С) потрібно задати параметри проекту (рис. А. 10):

– у поле «Project name» введіть ім’я проекту;

– у полі «Project Types» (Тип проекту) виберіть «Executable» (Виконуваний) – «Empty Project» (Пустий проект);

– у полі “Toolchains”(Інструменти) виберіть компілятор «MinGW GCC» та натисніть “Next” (Далі).

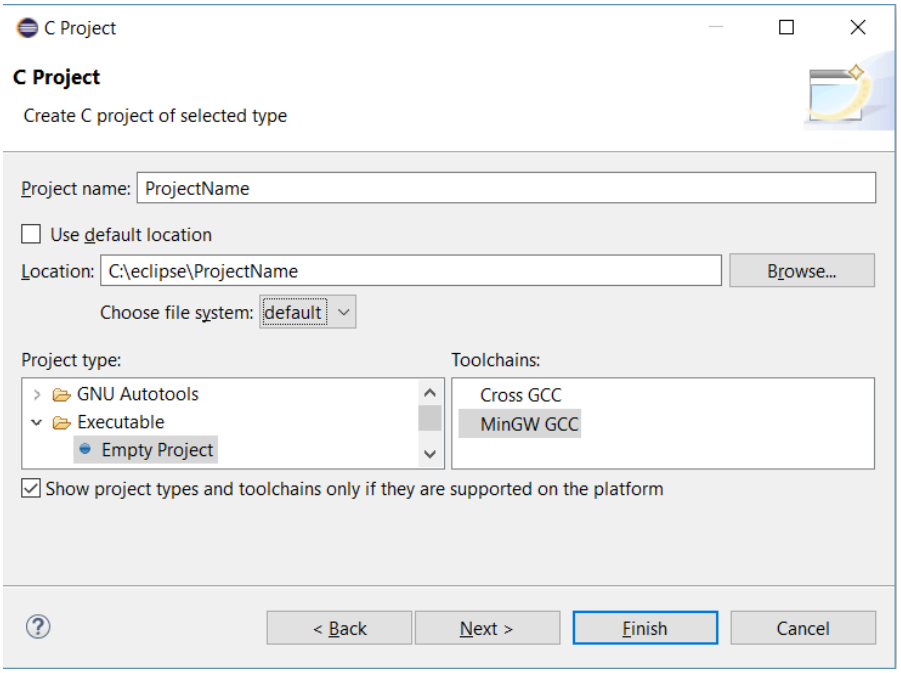


Рисунок А. 10 – Вікно вибору параметрів проекту

У діалоговому вікні "Select Configurations"(Вибір налаштувань) оберіть обидва варіанти “Debug” та “Release” (рис. А. 11). Натисніть кнопку “Finish”(Закінчити).

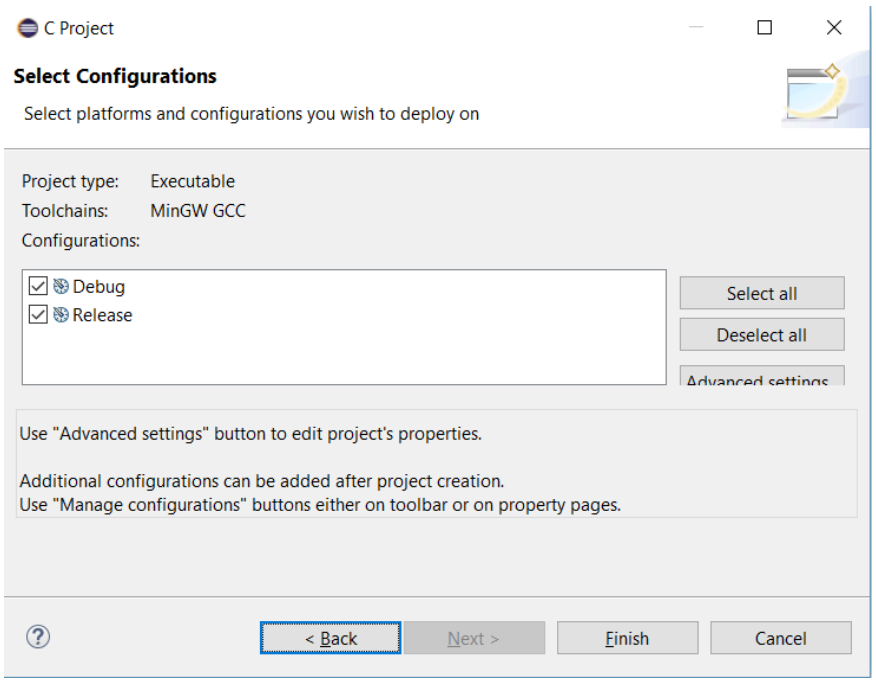


Рисунок А. 11 – Вибір конфігурації

4. У панелі "Project Explorer" (Провідник проектів) натисніть правою кнопкою на ваш проект - “New” (Новий) - “Source File” (Вихідний файл).

5. У діалоговому вікні “New Source File” введіть назву файлу з розширенням “.c” та оберіть шаблон “Default C source template” (Шаблон вихідного вайлу С за замовчуванням) (рис. А. 12).

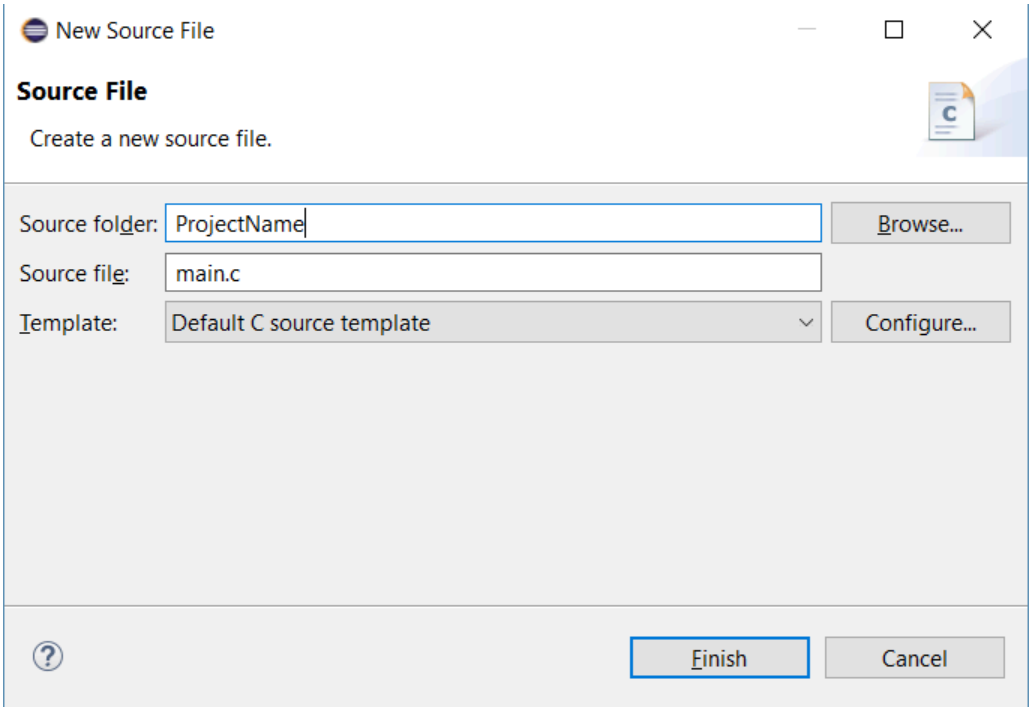


Рисунок А. 12 – Створення програмного файлу

На панелі редактора відкриється вихідний файл "main.c" (двічі клацніть на "main.c", щоб відкрити при необхідності).

Налагодження програми

Щоб налагодити програму в Eclipse встановіть точки зупину клацнувши по полю ліворуч від номера рядка на якому ви хочете встановити точку зупину

Натисніть Run – Debug (або F11) для того щоб перейти у режим налагодження.

Програма призупиниться на першій точці зупину ПЕРЕД виконанням інструкції записаної у поточному рядку. Точка зупину, на якій зупинилась програма буде виділена зеленим кольором. У правій панелі Debug будуть відображені поточні значення для змінних в області видимості (рис. А. 13).

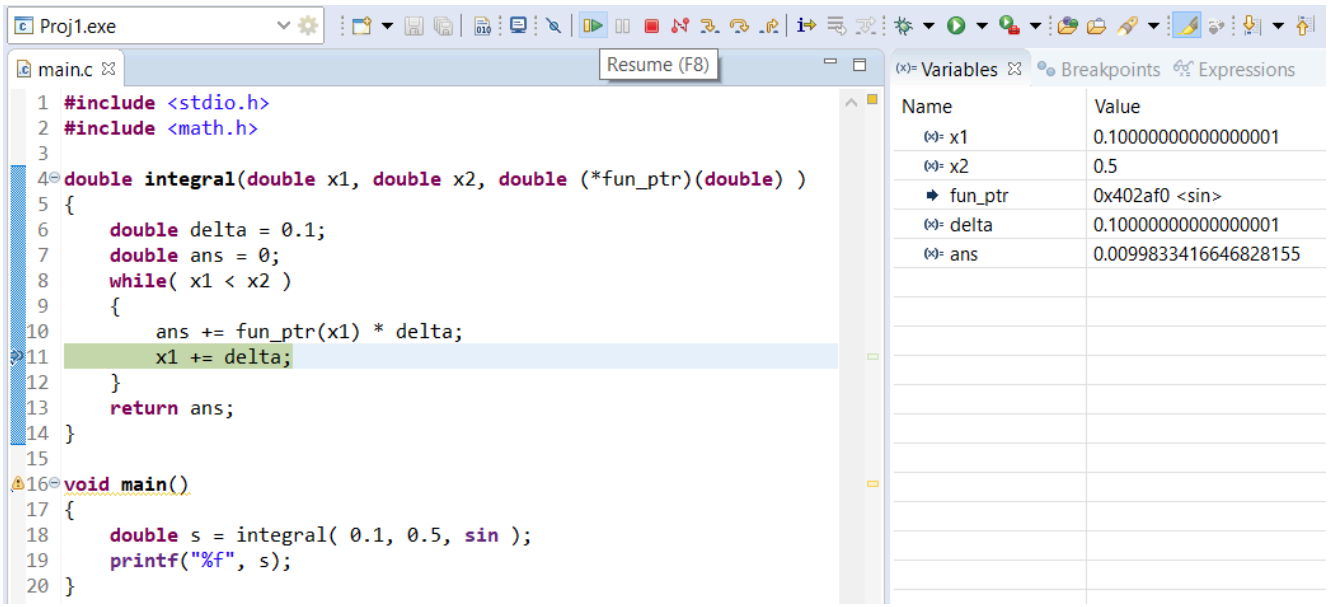


Рисунок А. 13 – Вікно від лагодження програми

Для переходу до наступної точки зупину натисніть Resume (або F8). Щоб перейти до наступної інструкції натисніть Step Into (або F5).

QT CREATOR

Qt Creator — інтегроване середовище розробки, призначене для створення крос-платформових застосувань з використанням бібліотеки Qt [33]. Підтримується розробка як класичних програм мовою C++, так і використання мови QML, для визначення сценаріїв, в якій використовується JavaScript, а структура і параметри елементів інтерфейсу задаються CSS-подібними блоками. Qt Creator може використовувати GCC або Microsoft VC++ як компілятор і GDB як налагоджувач. Для Windows версій бібліотека комплектується компілятором, заголовними і об'єктними файлами MinGW.

Qt Creator дозволяє використовувати будь який текстовий редактор, наприклад Emacs або Vim. В Qt вбудовані всі інструменти розробника, редактор з підсвічуванням і доповненням коду, налагоджувач (графічний інтерфейс для GNU Debugger, інструменту для налагоджування), а також реалізована підтримка систем контролю версій Perforce, SVN і Git.

Компоненти

До складу Qt включені інструменти розробника з графічним або консольним інтерфейсом. Серед них:

− assistant – графічний засіб для перегляду гіпертекстової документації за інструментарієм та бібліотекам Qt.

− designer – графічний засіб для створення і збірки призначених для користувача інтерфейсів на основі компонентів Qt.

− qmake – крос-платформений генератор Makefile.

− moc – компілятор метаоб'єктів (обробник розширень Qt для C ++).

− uic – компілятор призначений для користувальницьких інтерфейсів з файлів .ui, створених в Qt Designer.

− rcc – компілятор ресурсів з файлів .qrc.

− qtconfig – графічний засіб установки налаштувань для програм Qt.

− qtdemo – запуск прикладів і демонстраційних програм.

− linguist – засіб для локалізації програм.

− pixeltool – екранна лупа.

Qt поширюється за трьома ліцензіями:

− Qt Commercial — для розробки ПЗ ліцензією власника, що допускає модифікацію самої Qt без розкриття змін;

− GNU GPL — для розробки відкритого програмного забезпечення, що поширюється на умовах GNU GPL;

− GNU LGPL — для розробки ПЗ з ліцензією власника, але без внесення змін до Qt.

Початок роботи з Qt Creator

Перше, що ми бачимо, після запуску програми – стартове вікно (рис. А. 14).

Ви можете використовувати селектор режимів (1), щоб перейти на інший режим Qt Creator.

Ви можете використовувати селектор наборів (2), щоб вибрати комплект для запуску (3), налагодження (4) або побудови програми (5). Вивід з цих дій відображається на екранах виводу (7). Ви можете використовувати пошук (6) для перегляду проектів, файлів, класів, функцій, документації та файлових систем.

Створення проекту

Існує два варіанти створення проекту:

1. На стартовій сторінці натиснути пункт «New Project» («Новий проект...»).

2. Або через меню File - New File or Project (Файл – Новий файл або проект).

Після цього з’явиться вікно створення нового проекту, де можна обрати один з попередньо створених шаблонів проекту.

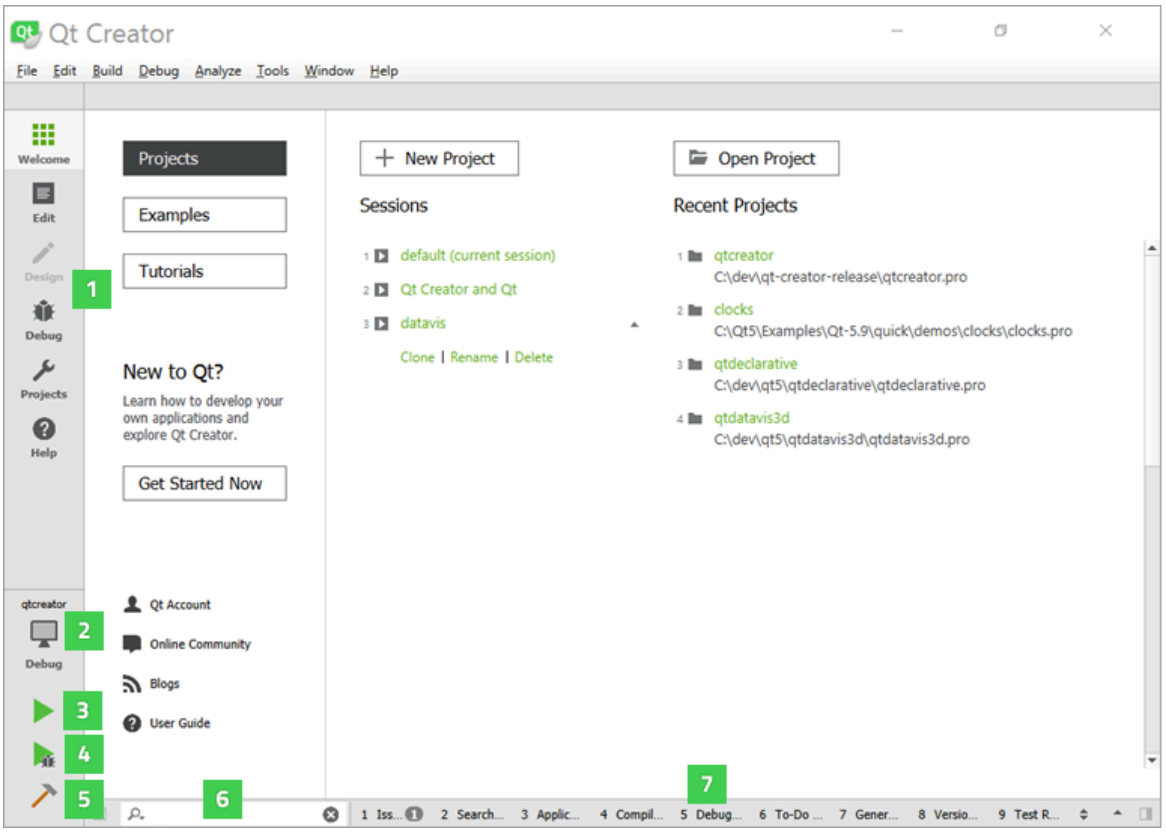


Рисунок А. 14 – Стартове вікно Qt Creator

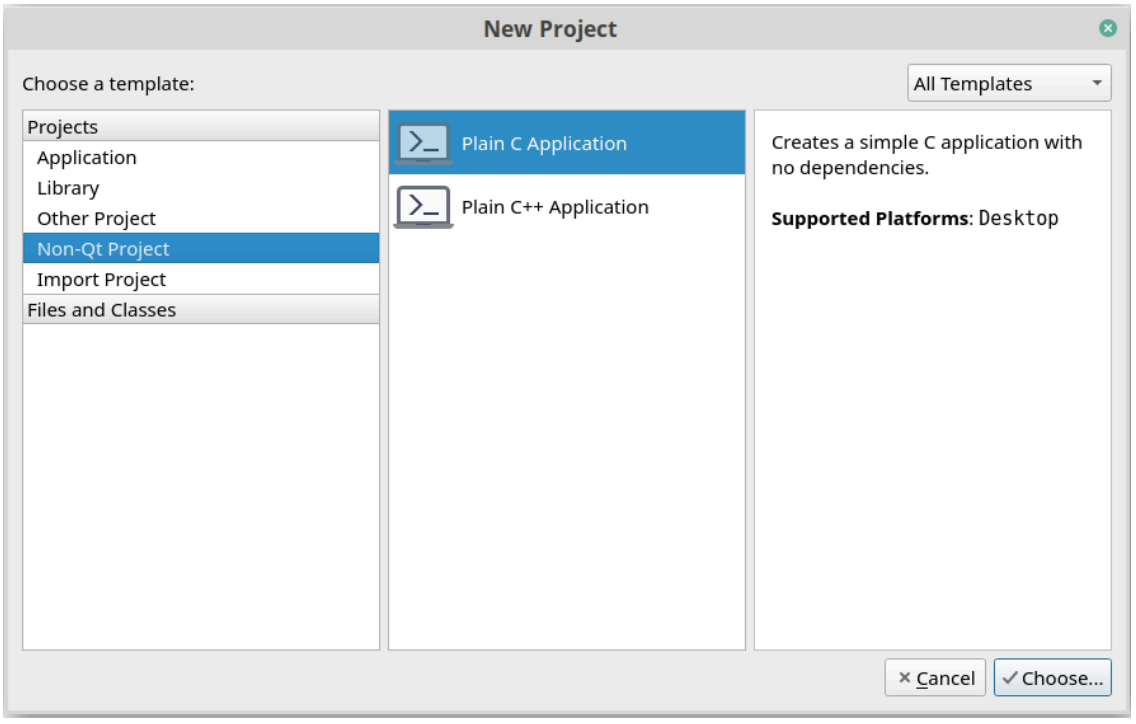


Рисунок А. 15 – Вікно створення проекту Qt Creator

Для написання нескладних програм:

1. Оберіть шаблон проекту “Non-Qt Project” (Проект без Qt).

2. Оберіть тип програми “Plain C Application”

3. На сторінці “Location” введіть ім’я проекту та місце розташування (рис. А. 16).

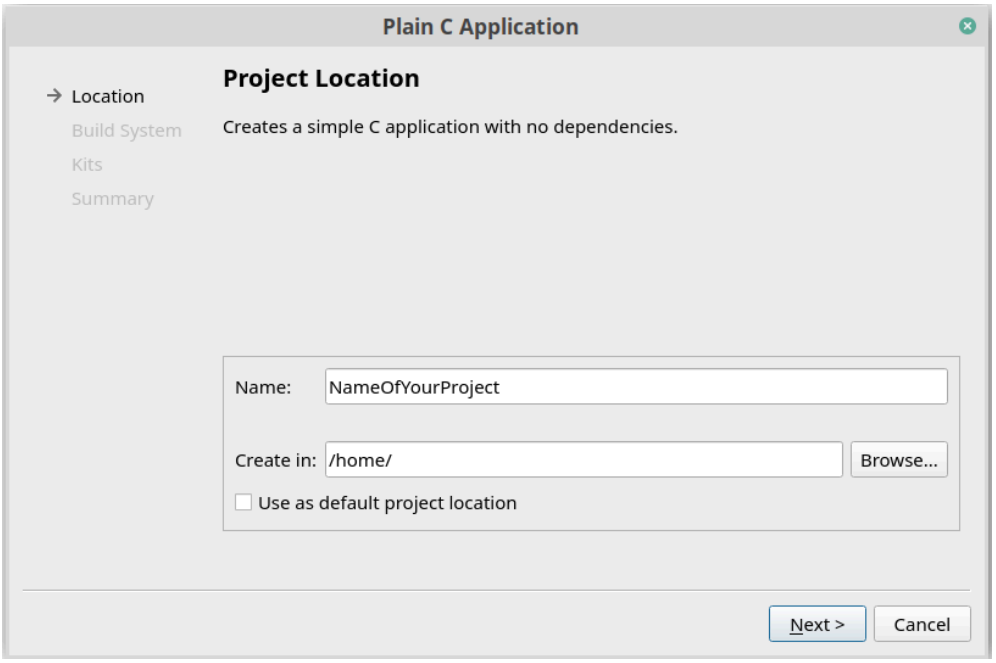


Рисунок А. 16 – Вікно створення проекту Qt Creator

4. На сторінці “Build system” оберіть систему збірки qmake (рис. А. 17).

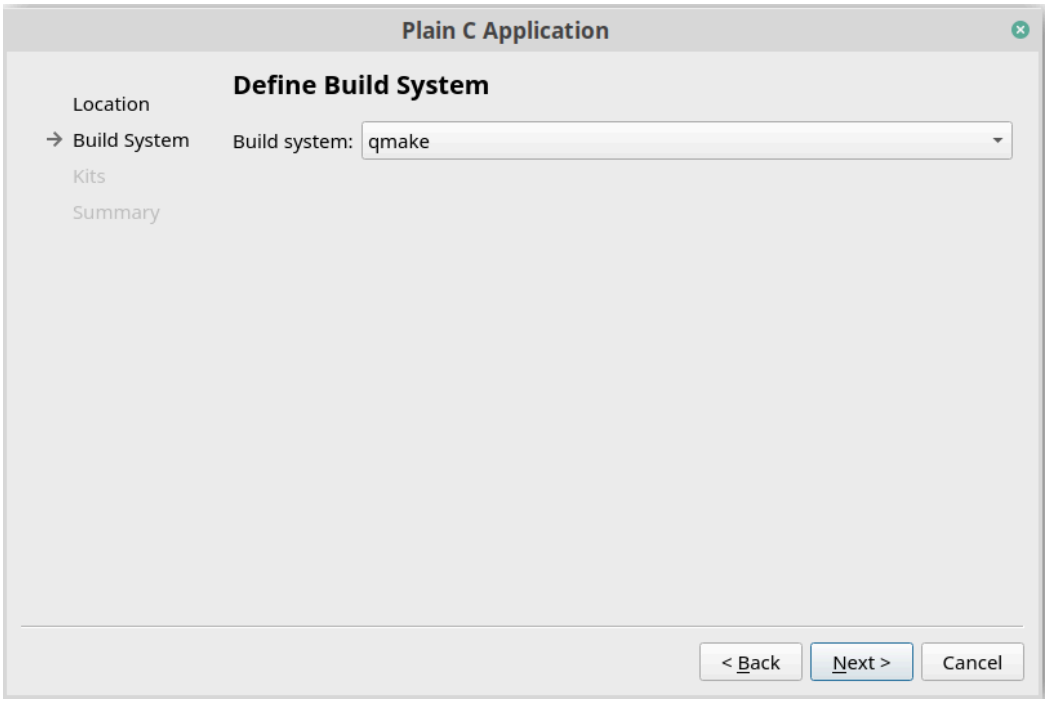


Рисунок А. 17 – Налаштування збірки проекту Qt Creator

5. На сторінці “Kits” оберіть варіант збірки (рис. А. 18)

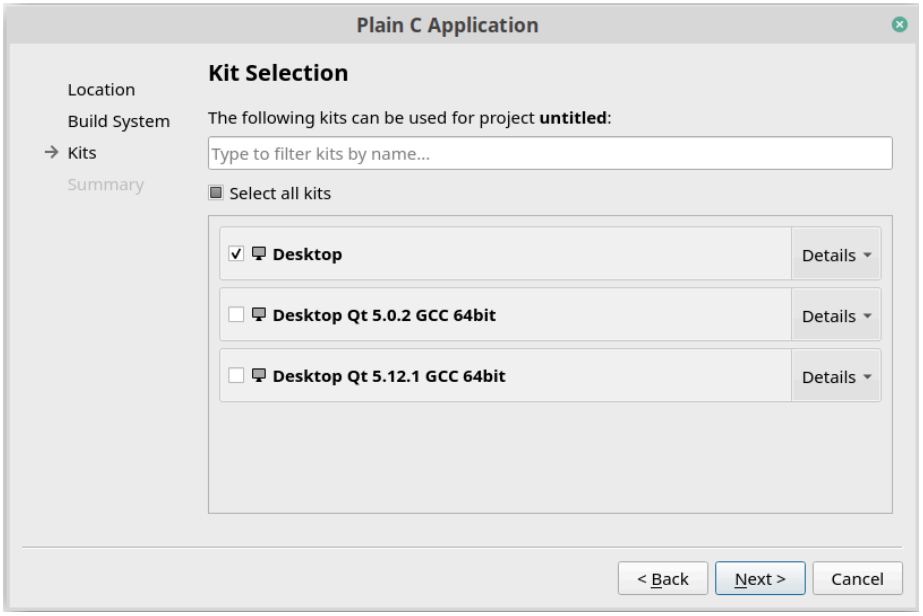


Рисунок А. 18 – Продовження налаштування збірки проекту Qt Creator

6. Натисніть кнопку “Finish”

Структура проекту

Після створення проекту будуть створенні файли:

- main.c — головний програмний файл .
- Ім’яПроекту.pro — файл з налаштуваннями побудови проекту (рис. А. 19).

Рисунок А. 19 – Структура проекту

Файли з розширенням “.c” будуть автоматично віднесенні до каталогу “Sources”. З розширенням “.h” – до каталогу “Headers”.

Запуск проекту

Сервіс для побудови та запуску програми є зрозумілим (рис. А. 20).

Рисунок А. 20 – Побудова та запуск проекту проекту

Натисніть на зображення “Build and Run Kit Selector” (1) або виберіть Build --> Open Build and Run Kit Selector для вибору комплекту збирання та запуску. Натисніть кнопку Run (2). Для того щоб призупинити виконання програми, натисніть кнопку Stop(3).

Налагодження програми

Щоб налагодити програму у Qt встановіть точки зупину клацнувши по полю ліворуч від номера рядка на якому ви хочете встановити точку зупину. Натисніть Debug – Start Debugging – Start Debugging of startup project (або F5) для того щоб перейти у режим налагодження (рис. А. 21).

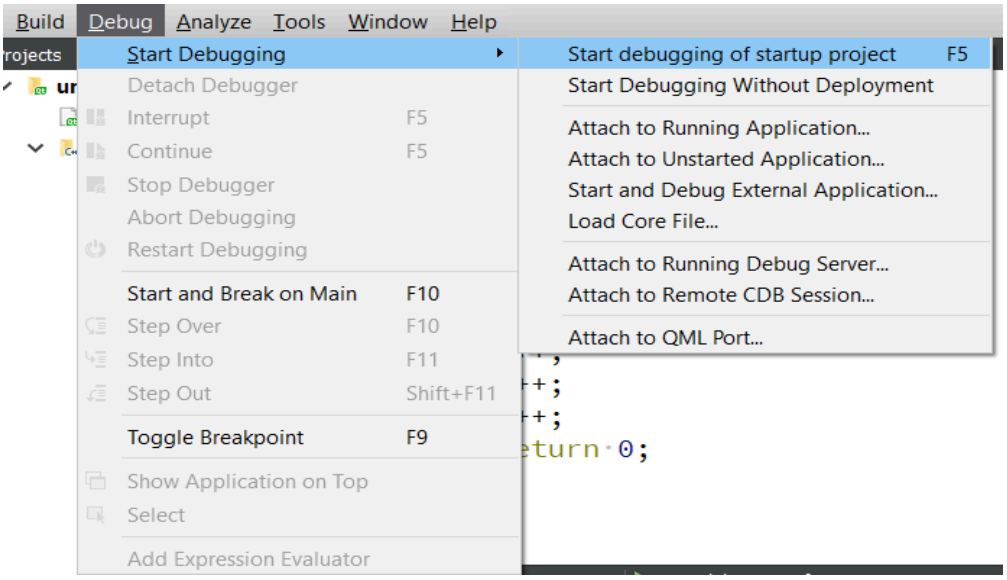


Рисунок А. 21 – Налагодження програми у Qt Creator

Для переходу до наступної точки зупину натисніть Continue (або F5). Щоб перейти до наступної інструкції натисніть Step Into (рис. А. 22).

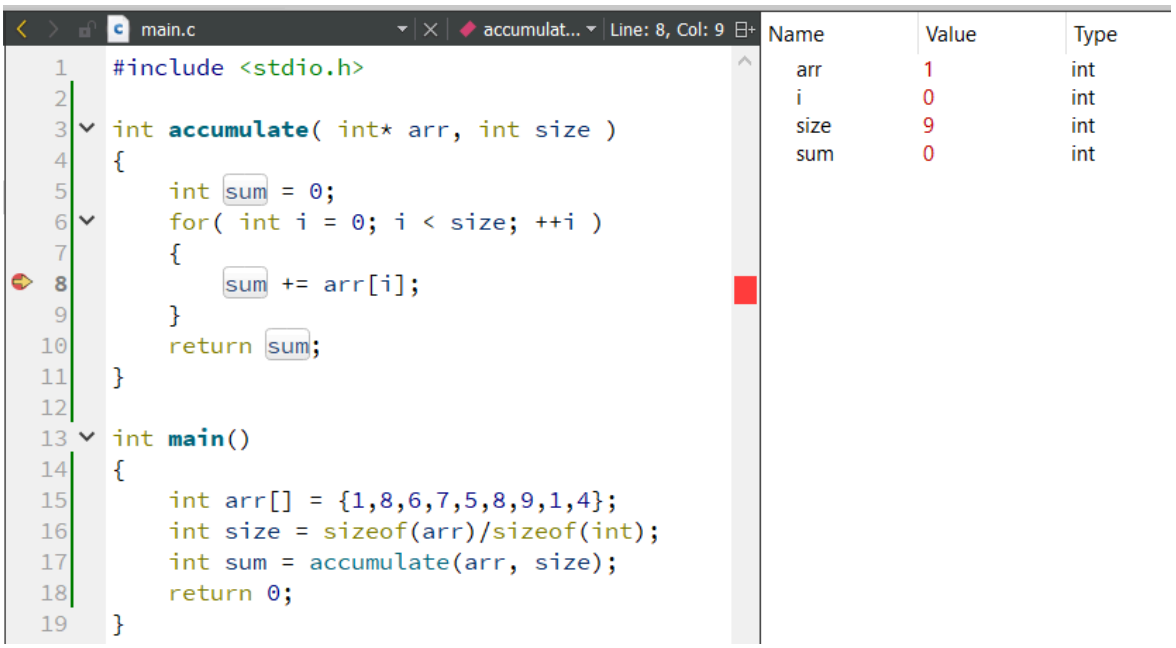


Рисунок А. 22 – Користування точками зупину

ДОДАТОК Б. ПРИНЦИПИ СТВОРЕННЯ КОНСОЛЬНИХ БАГАТОМОДУЛЬНИХ ПРОГРАМ

При розробці великої програми її зручно розділити на складові частини – підзадачі – і кожну частину помістити в окремий файл. Підзадача може розроблятися окремою групою розробників. Набір файлів, що утворюють програму, називається проектом. Для кожного проекту створюється спеціальний службовий файл – файл проекту, який зберігає інформацію про імена, типи, місцезнаходження файлів, що містять частини коду програми або дані. Ця інформація використовується при компіляції.

Файли, що входять в проект, можуть перебувати в будь-яких каталогах, проте зручніше зберігати їх всі разом в одному каталозі – каталозі проекту, який треба попередньо створити. Такий каталог створюється автоматично, якщо при розробці використовується середа Visual C++, однак у багатьох інших середовищах створення каталогу для зберігання файлів проекту виконує сам програміст.

Більшість сучасних інструментальних середовищ для розробки програм мовою C/C++ підтримують роботу над проектом. Для кожної консольної програми в Visual Studio C++ створюється один файл проекту. У ньому програмістом додаються файли вихідного коду та заголовні файли. Файл проекту генерується при виборі пункту меню FILE –> New project… (Файл –> Новий Проект…) (Ctrl + Shift + N). Також можна додати існуючий проект за допомогою FILE –> Add –> Existing project… (Файл / Додати / Існуючий проект…).

Далі в новий проект додаються файли вихідного коду з розширенням .cpp. Виконати це можна за допомогою команди меню FILE –> New file… (Ctrl + N) –> C++ File (Файл –> Новий Файл… –> Файл C++) або на панелі «Solution Explorer» («Оглядач рішень») клацнувши правою кнопкою миші на папці «Source files» («Файли вихідного коду») і вибравши Add –> New item… –> C++ File (Додати –> Новий елемент –> Файл C++). Ім'я файлу має бути унікальним. Два файла з одним і тим же ім'ям не можуть використовуватися одночасно в одному і тому ж проекті.

Модуль – функціонально закінчений фрагмент програми, оформлений у вигляді окремого файлу з вихідним кодом, призначений для використання в інших програмах.

У мові програмування C, заголовні файли – основний спосіб підключити до програми типи даних, структури, прототипи функцій, перелічувальні типи, і

макроси, використовувані в іншому модулі. Має за замовчуванням розширення .h. Щоб уникнути повторного включення одного й того ж коду, використовуються директиви #ifndef, #define, #endif. Заголовний файл в загальному випадку може містити будь-які конструкції мови програмування, але на практиці виконуваний код у заголовні файли поміщають. Наприклад, ідентифікатори, які є необхідність використовувати більш ніж в одному файлі, зручно описати в заголовному файлі, а потім його підключати у міру потреби. Також, за традицією, в заголовних файлах оголошують функції стандартної бібліотеки С.

При створенні програми з декількох модулів слід дотримуватися принципу незалежної компіляції модулів. Цей принцип полягає в тому, що при підготовці виконуваного файлу спочатку треба домогтися компіляції кожного модуля незалежно від інших модулів. Відсутність синтаксичних помилок у всіх модулях проекту – необхідна умова для створення виконуваного файлу, але недостатня, оскільки помилки можуть з'явитися при об'єднанні модулів, і поки вони не будуть виправлені, виконуваний файл не отримати.

Директива #include створює копію зазначеного файлу, яка включається в програму замість директиви. Існує дві форми використання директиви `#include`:

```
#include <filename>
#include “filename”
```

Різниця між ними полягає в тому, де предпроцессор буде шукати файли, які необхідно включити. Якщо ім'я укладено в лапки, предпроцессор шукає його в тому ж каталозі, що і компільований файл. Такий запис зазвичай використовують для включення визначених користувачем заголовних файлів, що містять загальні для окремих файлів програми оголошення. Прикладами таких оголошень є оголошення структур та об'єднань, констант і прототипів функцій. Якщо ім'я файлу укладено в кутові дужки (<i>), що використовуються для файлів стандартної бібліотеки, то пошук буде вестися в залежності від конкретної реалізації компілятора, звичайно в наперед визначених каталогах.

Розглянемо нескладний приклад. Припустимо, нам необхідно виконати обчислення площі та периметра еліпса. Для вирішення даної задачі скористаємося модулем для відділення загальної частини програми від основної. Створимо файл `Ellipse.h`, в який помістимо структуру еліпса і інтерфейс функцій, і файл `Ellipse.cpp`, в який помістимо реалізацію оголошених раніше функцій.

Наведемо листинги файлів.

```
//Ellipse.h
//Заголовний файл, в якому описується структура Ellipse і прототипи деяких функцій для роботи з нею, а також оголошується константа PI
//оголошення константи
const float PI = 3.14;
//оголошення структурного типа Ellipse
struct Ellipse {
     float axis1; //півосі
     float axis2; //еліпса
};
//оголошення прототипів функцій
//функція повертає Ellipse з заданими розмірами
Ellipse createEllipse (float a, float b);
//функція повертає площу переданого їй Ellipse
float area (Ellipse ell);
// функція повертає периметр переданого їй Ellipse
float perimeter (Ellipse ell);

//Ellipse.cpp
//У цьому файлі розміщується реалізація функцій, прототипи яких були оголошені в заголовному файлі Ellipse.h
#include "Ellipse.h"
Ellipse createEllipse (float a, float b) {
     Ellipse newEll;
     newEll.axis1 = a;
     newEll.axis2 = b;
     return newEll;
}
float area (Ellipse ell) {
     return PI*ell.axis1*ell.axis2; // обчислення площі еліпса
}

float perimeter (Ellipse ell) {
     return 4*((PI*ell.axis1*ell.axis2+
(ell.axis1-ell.axis2)*(ell.axis1-ell.axis2))/
(ell.axis1+ell.axis2));//формула обчислення периметра еліпса
}
```

```
//main.cpp
//Основний файл програми, в якому реалізується функція main
#include <stdio.h>
#include <stdlib.h>
#include "Ellipse.h"
int main() {
     float a, b;
     printf("Input ellipse semimajor and semiminor axes: ");
     scanf("%f%f", &a, &b);
     struct Ellipse theEllipse = createEllipse (a, b);
     printf("\nThe ellipse perimeter is: %.2f",
perimeter(theEllipse));
     printf("\nThe ellipse area is: %.2f\n", area(theEllipse));
     system("pause");
}
```

Розглянемо ще один приклад. Треба розробити інформаційну систему, яка дозволяє автоматизувати роботу користувача з книгами. О кожній з книг відомі такі дані: назва, автор, рік видання. Інформацію о книгах потрібно зберігати у бінарному файлі. Система повинна забезпечити можливість перегляду вмісту файла, додавання нових книг та видалення книг з файлу. Додатково треба розробити функції для пошуку книги у файлі за її назваю та для коригування року видання книги по заданій назві та старому року видання.

Аналіз завдання показує, що всю програму доцільно розбити на три .cpp-файла. У одному файлі будуть функції, що забезпечують роботу з файлом книг: перегляд, додавання, видалення. У другому – ті, що виконують додаткові дії: пошук, коригування. Третій файл, найважливіший, буде містити функцію main, з якої усі інші функції будуть викликатись. Також очевидно, що структурний тип, який буде використовуватись для роботи з даними про книги, потрібен для роботи усіх трьох файлів. Звісно, у цьому прикладі необхідно використовувати засоби умовної компіляції.

Наведемо текст програми. У файлі `book.h` наведено оголошення структури Book, там для запобігання множинного включення використовується `#ifndef`. У

файлі function.h наведені оголошення функцій, там використовується #pragma once.

```
// файл book.h
#ifndef BOOK_H
#define BOOK_H
struct Book {
     char name [50];
     char author [50];
     unsigned  year;
};
#endif // BOOK_H
// файл function.h
#pragma once
#include "book.h"
#include <stdio.h>
#include <stdlib.h>
/* Interface functions */
void PrintDB();
void AddBook();
void DeleteBook();
void FindBook();
void CorrectBook();
/* Controller functions */
Book * ReadDB(unsigned* size);
void WriteDB(Book * booksArray, int size);
// файл database_control_system.cpp
#include "functions.h"
#include <string.h>
void PrintDB() {
     unsigned* numberOfBooks = (unsigned*)malloc(sizeof(unsigned));
     *numberOfBooks = 0;
     Book * booksArray = ReadDB(numberOfBooks);
     printf("===Library===\n");
     for (unsigned i = 0; i < (*numberOfBooks); i++){
          if (strcmp(booksArray[i].name, "") != 0) {
               printf("\n Book %d \n", i + 1);
```

```
                printf("\t Name: \t\t %s \n", booksArray[i].name);
                printf("\tAuthor:\t %s \n", booksArray[i].author);
                printf("\t Year: \t\t %d \n", booksArray[i].year);
           }
     }
}

void AddBook() {
     char name [50];
     char author [50];
     unsigned year;
     unsigned newBooks;
     printf("===Adding a Book===\n\n");
     printf("Input number of books for adding: ");
     scanf("%d", &newBooks);
     unsigned* numberOfBooks = (unsigned*)malloc(sizeof(unsigned));
     Book * bookArray = ReadDB(numberOfBooks);
     bookArray = (Book*)realloc(bookArray, ((*numberOfBooks) +
newBooks)*sizeof(Book));
     for (unsigned i = 0; i < newBooks; i++) {
          getchar();//get rid of '\n' char left in input stream
          printf("\n Book %d \n", i+1);
          printf("Name: \t\t");
          gets(bookArray[(*numberOfBooks) + i].name);
          printf("Author: \t");
          gets(bookArray[(*numberOfBooks) + i].author);
          printf("Year: \t\t");
          scanf("%d", &bookArray[(*numberOfBooks) + i].year);
     }
     WriteDB(bookArray, (*numberOfBooks)+newBooks);    //rewrite
file
}
void FindBook() {
     char nameForSearch[50];
     unsigned* numberOfBooks = (unsigned*)malloc(sizeof(unsigned));
     Book * booksArray = ReadDB(numberOfBooks);
     printf("===Searching book===\n\n");
```

```
	printf("Input name of book: ");
	getchar();
	gets(nameForSearch);
	int booksCount = 0;
	for (unsigned i = 0; i < (*numberOfBooks); i++)   {
		if(strcmp(nameForSearch, booksArray[i].name) != 0)
			continue;
		printf("\n Book %d \n", i+1);
		printf("\t Name: \t\t %s \n", booksArray[i].name);
		printf("\t Author: \t %s \n", booksArray[i].author);
		printf("\t Year: \t\t %d \n", booksArray[i].year);
		booksCount++;
	}
	if (booksCount == 0) {
		printf("No such book found.\n");
	}
}
void DeleteBook() {
	char nameForDelete[50];
	unsigned yearForDelete;
	unsigned* numberOfBooks = (unsigned*)malloc(sizeof(unsigned));
	Book * booksArray = ReadDB(numberOfBooks);
	printf("Input name of book you want to delete: ");
	getchar();
	gets(nameForDelete);
	printf("Input year of book you want to delete: ");
	scanf("%d", &yearForDelete);
	for (unsigned i = 0; i < (*numberOfBooks); i++)   {
		//find book to delete
		if (strcmp(nameForDelete, booksArray[i].name) == 0 ||
yearForDelete == booksArray[i].year) {
			strcpy(booksArray[i].name, "");
			//rewrite file
			WriteDB(booksArray, (*numberOfBooks));
			return;
		}
	}
```

```
	printf("No such book found.\n");
}
void CorrectBook() {
	char nameForSearch[50];
	unsigned yearForSearch;
	unsigned* numberOfBooks = (unsigned*)malloc(sizeof(unsigned));
	Book * booksArray = ReadDB(numberOfBooks);
	printf("Input name of book you want to change: ");
	getchar();
	gets(nameForSearch);
	printf("Input year of book you want to change: ");
	scanf("%d", &yearForSearch);
	for (unsigned i = 0; i < (*numberOfBooks); i++){
		//find book to change
		if (strcmp(nameForSearch, booksArray[i].name) == 0 &&
yearForSearch == booksArray[i].year) {
			printf("Input new year: ");
			scanf("%d", &booksArray[i].year);
			WriteDB(booksArray, (*numberOfBooks));
			return;
		}
	}

	printf("No such book found.\n");
}

// файл additional_functions.cpp
#include "functions.h"
Book * ReadDB(unsigned* size) {
	FILE * database;
	if((database = fopen("database.bin", "rb")) == NULL){
		printf("Error opening file!\n");
		return 0;
	}
	Book buf;
	int arraySize = 0;
	// Counting the number of array`s elements
```

```
	while (fread(&buf, sizeof(Book), 1, database) != 0)
		arraySize++;
	// Putting the head on first byte
	fseek(database, 0, SEEK_SET);
	Book * booksArray = (Book*) malloc(arraySize * sizeof(Book));
	fread(booksArray, sizeof(Book), arraySize, database);
	fclose(database);
	*size = arraySize;
	return booksArray;
}
void WriteDB(Book * booksArray, int size) {
	FILE * database;
	if((database = fopen("database.bin", "w")) == NULL){
		printf("Error opening file!\n");
		return;
	}
	fwrite(booksArray, sizeof(Book), size, database);
	fclose(database);
}
// файл main.cpp
#include "functions.h"
int main() {
	int choice;
	//create file if it doesn't exist
	fopen("database.bin", "a");
	do   {
		system("CLS");
		printf("===Library Database===\n\n");
		printf("===Menu===\n");
		printf("1. Print database\n");
		printf("2. Add books\n");
		printf("3. Delete book\n");
		printf("4. Search for book\n");
		printf("5. Correct book's year\n");
		printf("0. Exit\n");
		scanf("%d", &choice);
		system("CLS");
```

```
        switch (choice)
        {
        case 1:
             PrintDB();
             break;
        case 2:
             AddBook();
             break;
        case 3:
             DeleteBook();
             break;
        case 4:
             FindBook();
             break;
        case 5:
             CorrectBook();
             break;
        case 0:
             return 0;
             break;
        default:
             printf("Incorrect input\n");
             break;
        }
        system("pause");
    } while(choice != 0);
    return -1;
}
```

ДОДАТОК В. ЗАГОЛОВОЧНІ ФАЙЛИ МОВИ С

З кожною функцією стандартної бібліотеки С пов'язаний свій заголовок. Відповідні заголовки використовуваних функцій повинні бути включені в програму за допомогою директиви #include. Заголовки виконують дві важливі функції. По-перше, багато функцій стандартної бібліотеки працюють з даними власного певного типу, до яких повинна мати доступ основна програма, що використовує ці функції. Ці типи даних задаються в заголовках, пов'язаних з кожною функцією.

Другою причиною включення заголовків є необхідність отримання прототипів бібліотечних функцій. Прототипи функцій дозволяють компілятору виробляти більш сувору перевірку типів. Хоча прототипи технічно є необов'язковими, вони необхідні для всіх практичних цілей.

Стандартом С для заголовків зарезервовані ідентифікатори, що починаються символом підкреслення, за яким слід символ підкреслення або заголовна буква. Заголовки – це, як правило, файли, але не завжди. Компілятор може зумовити вміст заголовка внутрішнім образом. Однак в практичних цілях вміст стандартних заголовків С знаходиться в файлах, імена яких збігаються з іменами самих заголовків.

Список стандартних заголовків, визначених Стандартом С89

`<assert.h>` – визначає макрос `assert()`
`<ctype.h>` – обробка символів
`<errno.h>` – видача повідомлень про помилки
`<float.h>` – задає межі значень з плаваючою точкою, що залежать від реалізації
`<limits.h>` – задає різні обмеження, що залежать від реалізації
`<locate.h>` – підтримує локалізацію
`<math.h>` – різні визначення, що використовуються математичною бібліотекою
`<setjmp.h>` – підтримує нелокальні переходи
`<signal.h>` – підтримує обробку сигналів
`<stdarg.h>` – підтримує списки вхідних параметрів функції зі змінним числом аргументів
`<stddef.h>` – визначає деякі найбільш часто використовувані константи
`<stdio.h>` – підтримує систему введення/виведення

<stdlib.h> – змішані оголошення

<string.h> – підтримує функції обробки рядків

<time.h> – підтримує функції, які звертаються до системного часу

Список стандартних заголовків, визначених Стандартом С99

<complex.h> – підтримує арифметичні операції з комплексними числами

<fenv.h> – надає доступ до прапорців стану обчислювача, що виконує операції з плаваючою точкою, а також доступ до інших сигналів цього обчислювача

<inttypes.h> – визначає стандартний, переносимий набір імен цілочисельних типів, підтримує функції, які працюють з цілими значеннями найбільшою розрядності

<iso646.h> – додано в 1995 році Поправкою 1; визначає макроси, які відповідають різним операторам, наприклад && і ^

<stdbool.h> – підтримує логічні типи даних; визначає макрос bool, що сприяє сумісності з мовою С++

<stdint.h> – задає стандартний переносимий набір імен цілочисельних типів; цей файл включений в заголовок <inttypes.h>

<tgmath.h> – визначає макроси для родового (абстрактного) типу чисел з плаваючою точкою

<wchar.h> – доданий в 1995 році Поправкою 1; підтримує функції обробки мультибайтних слів і двобайтових символів

<wctype.h> – доданий в 1995 році Поправкою 1; підтримує функції класифікації мультибайтних слів і двобайтових символів

ВІДПОВІДІ НА ТЕСТИ ДЛЯ САМОКОНТРОЛЮ

Розділ 1. 1-В, 2-А, 3-В, 4-Б, 5-Б, 6-Г, 7-А, 8-Г, 9-Б, 10-Г, 11-В, 12-В, 13-А, 14-Б, 15-В, 16-А.

Розділ 2. 1-В, 2-Б, 3-А, 4-Г, 5-А, 6-Б, 7-Г, 8-Б, 9-Г, 10-В, 11-А, 12-Б, 13-Г.

Розділ 3. 1-Б, 2-В, 3-А, 4-В, 5-Г, 6-А, 7-Б, 8-А, 9-В, 10-Г.

Розділ 4. 1-Г, 2-Б, 3-А, 4-Г, 5-Б, 6-А, 7-В, 8-В, 9-А, 10-Б, 11-Г, 12-В.

Розділ 5. 1-Г, 2-Б, 3-А, 4-Г, 5-В, 6-А, 7-Б, 8-А, 9-В, 10-Г, 11-А, 12-Б.

Розділ 6. 1-Г, 2-Б, 3-В, 4-В, 5-А, 6-Б, 7-А, 8-А, 9-В, 10-Б, 11-Г, 12-В, 13-В, 14-А, 15-Б, 16-А, 17-Г, 18-Б.

Розділ 7. 1-Б, 2-А, 3-Б, 4-Г, 5-В, 6-Г, 7-А, 8-Б, 9-А, 10-В, 11-Б, 12-Г.

Розділ 8. 1-В, 2-А, 3-Б, 4-А, 5-Г, 6-В, 7-Б, 8-А, 9-Г, 10-А, 11-А, 12-В, 13-Б, 14-Г, 15-Б.

Розділ 9. 1-А, 2-Г, 3-Б, 4-Б, 5-В, 6-А, 7-Г, 8-А.

Розділ 10. 1-Б, 2-В, 3-Г, 4-А, 5-А, 6-В, 7-Б, 8-Б, 9-Г, 10-А.

Розділ 11. 1-В, 2-А, 3-А, 4-Г, 5-Б, 6-В, 7-А, 8-Б, 9-Г, 10-В, 11-Б.

Розділ 12. 1-В, 2-Б, 3-Б, 4-А, 5-В, 6-Г, 7-Б, 8-Г, 9-А, 10-В, 11-Б, 12-В, 13-Г, 14-А, 15-Б.

Розділ 13. 1-А, 2-Г, 3-Б, 4-А, 5-В, 6-Б, 7-Г, 8-Г.

Розділ 14. 1-Б, 2-Б, 3-В, 4-А, 5-В, 6-Г, 7-В, 8-А.

Розділ 15. 1-Г, 2-Б, 3-А, 4-В, 5-В, 6-Б, 7-А, 8-Б, 9-Г, 10-В, 11-Г, 12-А.

Розділ 16. 1-А, 2-Г, 3-А, 4-Б, 5-В, 6-Г, 7-А, 8-В, 9-Б, 10-Г.

Розділ 17. 1-В, 2-Г, 3-В, 4-Б, 5-А, 6-Г, 7-А, 8-Б, 9-Б, 10-В.

Розділ 18. 1-В, 2-В, 3-А, 4-Г, 5-Б, 6-А, 7-Г, 8-В, 9-А 10-Б.

Розділ 19. 1-А, 2-Б, 3-Б, 4-В, 5-Г, 6-В, 7-А, 8-В, 9-А, 10-Г.

ПРЕДМЕТНИЙ ПОКАЖЧИК