## Массивы. Одномерные массивы. Обработка числовых одномерных массивов.

Дата: 12.11.2021г.

Цель занятия: формирование умений обрабатывать числовые одномерные массивы на языке С++.

Задачи занятия:

- познакомить учащихся с одномерными массивами и формировать умения применять полученные знания при составлении программ на языке С++;
- развитие алгоритмического и логического мышления;
- формирование умений индивидуальной и коллективной работы.

Тип занятия: изучения нового материала

### Ход занятия

1. **Организационный этап.**
2. **Этап постановки целей и задач.**
3. **Информационный этап:**

На сегодняшнем занятии мы изучим цикл for, а также научимся работать с массивами чисел, которые позволяют хранить в себе последовательность.

**Сокращённая запись арифметических операций**

При написании программ часто требуется прибавить что-либо к текущему значению переменной. Например, при написании циклов while мы на каждом шаге прибавляли к счётчику единицу. Если название переменной длинное, то писать его несколько раз в одном арифметическом выражении неэкономно и неудобно. Чтобы избежать этого, была придумана сокращенная запись арифметических операций.

Например, мы хотим прибавить к переменной x единицу. Мы уже умеем записывать это как x = x + 1. То же самое можно написать в сокращенной записи как x += 1. Знак += означает, что к значению переменной, которая находится слева, будет прибавлен результат вычисления арифметического выражения, который находится справа. По аналогии с += существуют операции -=, *=, /=. Они определены и для целых, и для вещественных чисел. Для целых чисел также определена операция %= для подсчёта остатка.

Поскольку задача прибавить или вычесть единицу возникает очень часто, то для её записи существует еще более короткая форма. Чтобы увеличить значение переменной x на единицу, можно написать x++. Если же мы хотим вычесть единицу, то писать нужно x−−. Именно в честь этой операции язык называется С++.

**Цикл for**

В языке С++ for является более короткой записью цикла while, которую удобно применять в некоторых случаях. Допустим, нам нужно вывести все числа от 1 до 100. Это несложно сделать с помощью цикла while:

```
int i;
i = 1;
while (i <= 100) {
    cout << i << " ";
    i++;
}
```

Однако намного проще сделать это с помощью цикла for:

```
int i;
for (i = 1; i <= 100; i++) {
    cout << i << " ";
}
```

После слова for в скобках пишутся три выражения:

1. Начальное значение счетчика (то, что выполняется непосредственно перед while).
2. Условие, при истинности которого выполняется цикл (абсолютно то же самое, что и в цикле while).

3. Изменение счетчика (то, что делается самой последней операцией в цикле while).
Выражения разделяются точкой с запятой. В принципе, в качестве выражений может выступать что угодно: первое из них выполнится один раз до начала цикла, второе будет проверяться на каждом шаге, а третье — выполняться каждый раз после того, как выполнены все действия в теле цикла.

**Векторы**

На прошлых занятиях мы уже научились обрабатывать последовательности, при этом не запоминая их целиком. Но такое возможно не во всех задачах. Например, на практических занятиях мы научились упорядочивать два и даже три числа по возрастанию. Если бы нас попросили написать программу, упорядочивающую сто тысяч чисел, то у нас на это ушло бы довольно много времени. Поэтому было бы здорово научиться хранить последовательности элементов и работать с ними в независимости от их количества.
В языке C++ есть несколько способов хранить последовательности чисел, и мы рассмотрим только один из них — vector. Он назван в честь математического термина «вектор», который обозначает занумерованную последовательность объектов.
Возвращаясь к идее «коробочек» для хранения чисел, вектор можно представить как выложенные в ряд коробочки, на каждой — одинаковое название и порядковый номер (он также называется индекс). Нумерация коробочек идёт с нуля и без пропусков.
Чтобы создать вектор, нам потребуется подключить библиотеку vector, написав в начале программы:

```
#include <vector>
```

Рассмотрим пример: задано число N — количество элементов последовательности, а затем N целых чисел $A_i$ — это члены последовательности. Необходимо развернуть и вывести эту последовательность чисел.

```
int n;
cin >> n;
vector <int> a(n);
//считывание
for (int i = 0; i < n; i++) {
    cin >> a[i];
}
//обработка и вывод
for (int i = n - 1; i >= 0; i--) {
    cout << a[i] << " ";
}
```

Рассмотрим новый код подробнее. Строка vector <int> a(n); создаёт новый вектор. Сначала пишется слово vector, затем тип элементов вектора в угловых скобках (int или double), затем имя вектора и в круглых скобках — количество элементов в нём. Считать вектор целиком в C++ нельзя, поэтому мы будем считывать очередное число и класть его в элемент вектора с соответствующим номером.
Чтобы обратиться к отдельным элементам вектора, нужно указать номер в квадратных скобках после его имени. Внутри скобок допускается любое арифметическое выражение, а с отдельным элементом вектора можно обращаться как с обычной переменной. Если указать в скобках номер, который превышает размер вектора (или отрицательное число), то программа скомпилируется, но будет работать неправильно или вовсе сломается. Это связано с тем, что такое обращение попадает в область памяти, которая не относится к нашему вектору.

**Метод push_back**

Рассмотрим еще одну похожую задачу: нужно считать последовательность и вывести задом наперед только положительные элементы. Задачу можно решить несколькими способами — например, считать всю последовательность и при выводе печатать только положительные элементы. Мы научимся решать задачу другим способом, запоминая только положительные элементы на этапе считывания данных.

```
int n;
cin >> n;
vector <int> a;
//считывание
for (int i = 0; i < n; i++) {
      int temp;
      cin >> temp;
      if (temp > 0) {
            a.push_back(temp);
      }
}
//обработка и вывод
for (i = a.size() - 1; i >= 0; i--) {
      cout << a[i] << " ";
}
```

В этой программе при создании вектора мы не указывали количество элементов в нём — он создался пустым.

Добавление делается с помощью метода push_back, при этом в скобках указано то, что и требуется добавить. Метод — это почти то же самое, что функция, применяемая к объекту. Сначала указывается имя объекта (в нашем случае это вектор a), затем ставится точка, пишется имя метода и в скобках указываются параметры. Аналогично push_back используется и метод size, который не принимает параметров и возвращает количество элементов в векторе.

**Поиск минимума в последовательности**

Часто возникает задача найти минимум (или максимум) в последовательности из N чисел. Эта задача уже встречалась нам. Рассмотрим такую задачу: есть последовательность чисел, в ней нужно поменять местами то число, что стоит в начале последовательности, и самое маленькое число.

Большинство простых программ — это запись тех действий, которые произвёл бы человек для решения той же задачи. Что будет делать человек, если его попросят найти минимум в последовательности из одного числа? Очевидно, он назовёт это число. Для двух чисел задача тоже решается несложно — достаточно выбрать меньшее из них. Как эта задача решается больше чем для двух чисел? В каждый момент времени человек помнит наименьшее из уже названных чисел и сравнивает его со следующим в ряду. Если очередное число оказалось меньше, чем запомненное минимальное, то человек запоминает вместо старого числа новое. Таким образом, в любой момент человек может сказать, какое из уже названных чисел было минимальным. В нашей задаче нужно помнить не само число, а его номер в последовательности. После нахождения номера минимального числа, нужно просто поменять местами этот элемент с элементом с индексом 0.

```
int n;
cin >> n;
vector <int> a;
//считывание
for (int i = 0; i < n; i++) {
    int temp;
    cin >> temp;
    a.push_back(temp);
}
//обработка
int num_min = 0; //номер минимального элемента
for (int i = 0; i < n; i++) {
    if (a[i] < a[num_min]) {
```

```
            num_min = i;
        }
    }
    //обмен значений элементов a[0] и a[num_min]
    int temp;
    temp = a[0];
    a[0] = a[num_min];
    a[num_min] = temp;
    //вывод
    for (auto now : a) {
        cout << now << " ";
    }
```

В этом решении есть два интересных момента. Во-первых, это обмен значений двух переменных. Он делается через третью, вспомогательную переменную. Во-вторых, это ещё более короткая запись при выводе всех значений вектора. Цикл for (auto now : a) будет поочередно подставлять в переменную now все значения из вектора a. Здесь auto – это тип переменной, «автоматический». Поскольку вектор состоит из целых чисел, то переменная now автоматически будет определена как целое число.

**Сортировка массива**

Рассмотрим теперь задачу сортировки массива. В ней нужно расположить числа, записанные в массиве, в порядке неубывания (это то же самое, что порядок возрастания, только в нём могут быть одинаковые элементы). Для решения этой задачи воспользуемся решением предыдущей. Прошлая программа умела находить минимальный элемент и ставить его в начало массива. Таким образом, в начале массива появилась отсортированная часть, состоящая пока что из одного элемента. Если повторить эту операцию (поиск минимума и перестановка) для всех элементов, кроме начального, то у нас получится отсортированная часть массива, состоящая уже из двух элементов. Повторив эту операцию N раз, мы получим упорядоченный массив.

```
int n;
cin >> n;
vector <int> a;
//считывание
for (int i = 0; i < n; i++) {
    int temp;
    cin >> temp;
    a.push_back(temp);
}
//обработка
for (int j = 0; j < n; j++) { //j - начиная с какого номера ищем наименьший
    int num_min = j; //номер минимального элемента
    for (int i = j; i < n; i++) { //ищем только в еще не упорядоченной части
        if (a[i] < a[num_min]) {
            num_min = i;
        }
    }
    //обмен значений элементов a[j] и a[num_min]
    int temp;
    temp = a[j];
    a[j] = a[num_min];
    a[num_min] = temp;
}
```

```
//вывод
for (auto now : a) {
    cout << now << " ";
}
```

Такой метод сортировки называется «сортировка выбором минимума».

4. **Перерыв на физкультминутку.**
5. **Практическое применение полученных знаний:**
   **Работа на сайте Stepik.org (раздел 1.4. Условный оператор)**
6. **Подведение итогов. Рефлексия**