دورة في السي للكاتب MLN4EVER من منتديات الفريق العربي للبرمجة

لـــغة الـــــسي

الفصل الأول : لغة C ... نظرة تاريخية و ملامح عامة

لغة C لغة متفردة في ملامحها ومنشآتها، وتتميز بأنها سلاح قوي للمبرمج، فهي تؤدي العديد مما لا تستطيع اللغات الأخرى – عالية المستوى- أن تؤديه كما تتيح للمبرمج التحكم بصورة أفضل في الكمبيوتر، ولذلك فإن لغة ال C قد أصبحت لغة العصر.

و على الرغم من أن لغة الC ليست جديدة فإنها لغة سريعة التطور ، حيث أبتكرها " دينيس ريتشي" في أوائل السبعينات وقدمها بالاشتراك مع " بريان كارينجان" في كتابهما (The C programming language)والذي يعد المرجع الأساسي في اللغة. ومنذ ذلك الحين واللغة في تطور مستمر.

وتطورت لغة C تطورا سريعا ليظهر منها الامتداد الذي يطلق عليه C++ وتتميز لغة C++ باعتمادها أساسا جديدا من طرق البرمجة وهو ما يطلق عليه (Object Oriented Programming).ومهدت لغة C++ الطريق لظهور لغة Visual C++ وهي الصورة الأحدث من اللغة والتي تعمل في بيئة الويندوز.

ونتيجة تزايد استخدام لغة C قامت مؤسسة القياسات الأمريكية في عام 1983
بعملية توحيد للهجات المختلفة التي كادت أن تنتشر للغة C فأصدرت اللغة القياسية التي يطلق عليها " ANSI C "وهي تحتوي على بعض الإضافات إلى اللغة الأصلية التي ابتكرها ريتشي.

ما هو البرنامج:
البرنامج اصطلاح يرمز لعدد محدد من الأوامر التي تعطى للكمبيوتر، بغرض تنفيذ مهمة محددة أو أداء وظيفة مطلوبة.

ومن أهم ملامح البرمجة بلغة C أن البرنامج ما هو إلا معمار دقيق التصميم يعتمد في بنائه على البلوكات الجاهزة التي تتكامل معا لتصنع البناء الضخم.و البلوك أو مايسمى بالدالة (function) ما هو إلا مجموعة من الأوامر متعلقة بجزء محدد من البرنامج، وتنتج البلوكات من تقسيم البرنامج إلى أجزاء أصغر لكل وظيفته التي يتم تحديدها بالأوامر التي تكتب في البلوك.

و استخدام البلوكات الجاهزة يوفر الوقت ولا سيما عندما نرغب في تطوير البرنامج أو إحداث تغيرات جذرية به. وليس هذا هو الحال مع لغة مثل بيسك حيث يبني المبرمج البناء كله من البداية، فإذا أراد المبرمج تعديل البرنامج فإنه يعيد كتابتة أو على الأقل يعيد كتابة أغلب أجزائة.

ونستطيع مع لغة C استخدام البلوكات الجاهزة الموجودة بمكتبات المبرمجين الآخرين، أو بناء مكتبة من الدوال للاستعانة بها وقت الحاجة.

وهناك خطوات مطلوبة لتنفيذ أي برنامج وهي:

1- كتابة البرنامج وحفظة على القرص باستخدام أحد برامج التحرير (Editors)

2- عملية الترجمة (compilation) وينتج عن هذه العملية البرنامج الهدف الذي يحمل عادة الامتداد " . "OBJ

3- عملية الربط بمكتبة اللغة (Linking) وينتج عن هذه العملية البرنامج التنفيذي الذي يحمل الامتداد "EXE". والبرنامج التنفيذي هو البرنامج الذي يتم تنفيذه بمجرد إدخال اسمه .

وهناك العديد من برامج الترجمة الشهيرة على الكمبيوتر مثل " Terbo C" أو "Quick C" وتلك البرامج تحتوي على بيئة مجمعة تشمل محررا لكتابة البرنامج، و قوائم ذات نوافذ بها أوامر الحفظ والترجمة و الربط و التنفيذ.

الفصل الثاني : مبادئ لغة ال : C

1- البرنامج الأول بلغة ال C

من أفضل الطرق للبدء بتعلم لغة جديدة النظر لأحد البرامج البسيطة المكتوبة بهذه اللغة ودراسة أجزاءه كل على حدة، ولنتخذ برنامجا متكاملا جاهزا للتنفيذ.

يوضح البرنامج التالي برنامجا صغيرا يطبع على الشاشة عند تشغيله العبارة
" Hello C"

CODE

```
#include <stdio.h>
main()
{
printf ( "Hello C");
}
```

إن البرنامج يعتمد أساسا على الدالة printf فهي المسئولة عن طباعة العبارة المطلوب طباعتها على الشاشة. وعندما تتقدم في اللغة ستجد أن لغة C مبنية من دوال مختلفة لكل وظيفتها المحددة، كما ذكرنا سابقا.

ولتؤدي الدالة printf المطلوب منها لا تستخدم بمفردها بل لابد أن تأتي بداخل الإطار الموضح بالشكل السابق حتى تتمكن من القيام

بعملها.

والإطار الذى يحوي البرنامج يبدأ بكلمة main يعقبها القوس الأيسر " { " والذي تتالى بعده عبارات البرنامج، ثم ينتهي بالقوس الأيمن " } ".

ويطلق على الجزء المحتوى بين القوسين " { } " اسم البلوك (block) .و البلوك الذي يبدأ بكلمة (main) يسمى بلوك البرنامج. وفي المثال السابق يتكون البرنامج من بلوك واحد هو بلوك البرنامج.

والسطر الأول من البرنامج والمحصور بين العلامتين " /* */" يسمى التعليق ويستخدم التعليق لكتابة الملاحظات على البرنامج، ومن المفيد دوما كتابة التعليقات لتسهيل مراجعة البرنامج .

وعند ترجمة هذا البرنامج فإن مترجم لغة C يتجاهل تماما كل ما يأتي بين هاتين العلامتين. ويجوز أن تضيف إلى البرنامج ما تشاء من الملاحظات وفي أي مكان من البرنامج وبأي عدد من السطور مادمت تبدأ وتنتهي بالعلامتين المميزتين" /* " ، " */ ".

أما السطر الثاني والذي يبدأ بالعلامة الخاصة " # " فيسمى بالتوجيه (Directive) وهو لا يمثل جزءا من منطق البرنامج ولكنه يستخدم لتوجيه المترجم أثناء الترجمة ، حيث يدله على مكان الملف " stdio.h " والذي يطلق عليه اسم ملف العناوين للدخل و الخرج أو (Standard Input Output header file)

ويجب الالتزام بسطور التوجيه لأن هناك دوال لابد لها من إستدعاء ملفات خاصه بها، وعندما نستخدم دالة دون استخدام سطر التوجية الخاص بها نحصل على خطأ من المترجم عند بداية الترجمة.

و هناك قواعد بسيطة لكتابة البرنامج بلغة C ولابد من مراعتها عند كتابة البرامج ومن هذه القواعد ما يمكن التسامح فيه فمثلا المسافات الخالية والسطور التي تفصل ما بين الكلمات والعبارات كلها اختيارية ويمكن الأستغناء عنها.

ولكن هناك من القواعد ما يجب الإلتزام به :

1-تكتب التوجيهات على سطر مستقل.
2- تستخدم الدوال (مثل printf) في تكوين عبارات البرنامج (staements) وتنتهي كل عبارة بفاصلة منقوطة. والفاصلة المنقوطة لاغنى عنها حتى لو كان البرنامج محتويا على عبارة واحدة، وأغلب الأخطاء التي نحصل عليها تكون نتيجة نسيان فاصلة منقوطة.
3- تتطلب بعض الكلمات الخاصة باللغة أن نعقبها بمسافة خالية على الأقل وإلا تعرضنا لرسالة خطأ من المترجم عند ترجمة البرنامج.
4- تكتب الكلمات المفتاحية للغة (key words) مثل أسماء الدوال (مثل printf) بالحروف الصغيرة (.(small letters

2-الطباعة على الشاشة

تستخدم الدالة **printf** لطابعة النصوص على الشاشة وهي كأي دالة أخرى تأتي متبوعة بقوسين نكتب بينهما النص المطلوب طباعته بين علامتي اقتباس.

وكل ما نكتبة بين علامتي الاقتباس يظهر كما هو على الشاشة ولذلك يصطلح على تسميته بالحرفي (string) .

والبرنامج الموضح في الشكل التالي يحتوي على عبارتين تستخدم في كل منهما الدالة **printf** لطباعة حرفي معين على الشاشة

CODE

```
#include <stdio.h>
main()
{
printf("Welcome ");
printf(" C Programmer");
}
```

وناتج البرنامج موضح بالشكل التالي

CODE

```
WelcomeC Programmer
```

ونلاحظ أن العبارتين طبعتا على الشاشة دون أي فاصل بينهما.

و لكننا حتما نريد الفصل بين العبارات المختلفة فمثلا ماذا لو أردنا الإنتقال لسطر جديد لتطبع العبارة الثانية على سطر مستقل؟

إن الإنتقال لسطر جديد يستلزم إضافة علامة خاصة إلى نهاية الحرفي الأول، وتسمى هذه العلامة بعلامة السطر الجديد (new line character) وتكتب كالآتي (\n)

ولنجرب إستخدام هذه العلامة وذلك كما هو موضح في الشكل التالى

CODE

```
#include <stdio.h>
main()
{
printf("Welcome \n");
printf(" C Programmer");
}
```

وعند تنفيذ هذا البرنامج نحصل على النتيجة التالية

CODE

```
Welcome
C Programmer
```

ومما يجب ملاحظته أن علامة السطر الجديد تكتب بداخل علامتي الاقتباس ولا تظهر على الشاشة كما هي !!!
وذلك لأن المترجم يفهم العلامات الخاصة على نحو ما وتعتبر أمرا من الأوامر يقوم بتنفيذها بالصورة المطلوبة.

ويمكن استخدام دالة الطباعة لتطبع على الشاشة محتويات بطاقة تحمل الاسم والعنوان كما هو موضح بالشكل التالي

CODE

```
Future Horizons Co.
81 emarat othman
NasrCity
Cairo
```

و البرنامج المستخدم لطباعة هذة البطاقة موضح بالمثال التالي

CODE

```
#include <stdio.h>
main()
{
```

```
printf("Future Horizons Co.\n");
printf("81 emarat othman \n");
printf("NasrCity\n");
printf("Cairo\n");
}
```

3- التعامل مع الاعداد

يمكن باستخدام عبارة الطباعة و الدالة printf أن نعرض الأرقام على الشاشة بل يمكننا أيضا أن نجري العمليات الحسابية المختلفة فتتولى الدالة printf تقييم التعبيرات الحسابية وطباعة النتيجة على الشاشة.ومن الملاحظ أن الأعداد لا تحتاج لعلامات اقتباس.

وفي لغة C يجب أن نفرق بين نوعين من الأعداد:

1-الأعداد الصحيحة (Integers)
2- الأعداد الحقيقية (Real numbers)

أما الأعداد الصحيحة فهي تلك الأعداد التي لا تحوي كسورا. بينما تحتوي الأعداد الحقيقية على علامة عشرية (بصرف النظر عن وجود كسر من عدمه).

فورمات الأعداد:

يلزم إخبار الكمبيوتر دائما عن نوع العدد باستخدام صيغة خاصة (فورمات) تأتي بداخل علامتي الأقتباس، لأن الكمبيوتر يتعامل مع كل نوعية من الأعداد بطريقة مختلفة تماما.

ولتوضيح استخدام الفورمات انظر الشكل

CODE

```
#include <stdio.h>
main()
{
printf("%d \n",130);
printf("%f\n",130.5);
}
```

وفي هذا البرنامج استخدمنا نوعين من الأعداد و لكل منهما لبفورمات الخاصة به فنجد أن رمز الفورمات المستخدم مع العدد الصحيح هو (d%)
والحرف (d) بهذا الرمز هو اختصار كلمة (decimal) بمعنى رقم عشري أي مكتوب بالنظام العشري.

أما رمز الفورمات المستخدم لطباعة العدد الحقيقي فهو (f%) والحرف (f) بهذا الرمز هو اختصار كلمة (floating point number) وهي الأعداد ذات العلامة العشرية.

وعند تنفيذ البرنامج السابق نحصل على النتيجة الموضحة بالشكل

CODE

```
130
130.5
```

ويجب على المبرمج تحري الدقة التامة عند التعامل مع الفورمات ، فلا نستخدم فورمات الأعداد الحقيقية مع الأعداد الصحيحة أو العكس. لأن الخطأ في الاستخدام ينتج عنه نتائج غير صحيحة.

التعبيرات الحسابية:
كما ذكرنا سابقا فأن دالة الطباعة يمكنها أيضا أن تجري العمليات الحسابية المختلفة وتطبع النتيجة على الشاشة.
وتستخدم المؤثرات الحسابية الموضحة ادناه لبناء التعبيرات الحسابية:

مؤثر الجمع +
مؤثر الطرح -
مؤثر الضرب *
مؤثر القسمة /

والمثال التالي يوضح استخدام المؤثرات الحسابية مع الدالة (printf)

CODE

```
#include <stdio.h>
main()
```

```
{
printf("%d\n",128*2);
printf("%f\n",128.0/2);
}
```

وعند تنفيذ البرنامج نحصل على الناتج الموضح بالشكل التالي

CODE

```
256
64.000000
```

4- استخدام المتغيرات

يقوم الكمبيوتر بتخزين البيانات التي يحتاجها في الذاكرة والمتغيرات ما هي إلا عناوين خانات في الذاكرة التي نحفظ فيها البيانات.
ولتسهيل الوصول للبيانات المختزنة يتم في لغات البرمجة عالية المستوى استبدال العناوين الرقمية بأسماء المتغيرات.

ويكفينا هنا - لو كنا مبتدئين في البرمجة- أن نتذكر دائما أن المتغير ما هو إلا اسم لأحد الأماكن التي تختزن فيها البيانات في الذاكرة.
وأسماء المتغيرات يصطلح عليها في لغة الC بأسماء البيانات (Identifiers)
وهناك قواعد محددة لاختيار أسماء البيانات وهي:

1-ألا يكون اسم البيان أحد الكلمات المحجوزة باللغة (Reserved words) أو الكلمات التي تحمل معنى خاصا مثل(main)
ويمكن التعرف على الكلمات المحجوزة باللغة من دفتر التشغيل المصاحب للمترجم.
2-يمكن أن يحتوي الاسم على أى حرف من الحروف الأبجدية (A-Z) سواء صغيرة كانت أم كبيرة، وأي رقم من الأرقام (0-9) كما يمكن أن تحتوي على علامة الشرطة السفلى (_) ولكن لا يجوز أن يبدأ الاسم برقم.
3-لا قيود على طول الاسم ، وتتيح هذه الميزة استخدام أسماء معبرة عن مضمونها، ومن الأفضل دائما استخدام الاسم المعبر عن محتوى المتغير لتسهيل عملية فحص البرنامج في حالة الخطأ من جهة، ولتسهيل عملية الإضافة والتعديل للبرنامج.
4-الحروف الكبيرة و الصغيرة ليست متكافئة في لغة C فمثلا اسم البيان(MY_NUMBER) يختلف عن الاسم (my_number)
وكلاهما يختلف عن الاسم (My_Number).

الإعلان عن المتغيرات:

ليتمكن المستخدم من استخدام المتغيرات التي يريدها يتطلب البرنامج المكتوب بلغة C الإعلان المسبق عن أسمائها ونوعياتها في مستهل البرنامج .

وتصنف المتغيرات بحسب البيانات التي يمكن أن تختزن فيها فهناك المتغيرات الصحيحة (أي التي تصلح لإختزان الأعداد الصحيحة) و هناك المتغيرات الحقيقية (أي التي تختزن الأعداد الحقيقية)، ومع تقدمنا في اللغة سنتعرف على نوعيات أخرى من المتغيرات.

والشكل التالي يوضح برنامجا قمنا فيه بالإعلان عن المتغيرات

CODE

```
#include <stdio.h>
main()
{
/* variable declaration*/
int a;
float b;
/*Display output */
printf("%d\n",a);
printf("%f\n",b);
}
```

وكما نرى في البرنامج أنه قد تم الإعلان عن متغيرين الأول (a) وهو من النوع الصحيح (integer) وقد استخدمنا الكلمة int للإعلان عنه.

وأما المتغير الثاني (فهو يختزن الأعداد الحقيقية (Real) وقد استخدمنا معه الكلمة float للإعلان عنه.

وكما ذكرنا سابقا، نلاحظ أن عبارة الإعلان تنتهي بفاصلة منقوطة كسائر عبارات البرنامج، كما أنه يلزم ترك مسافة خالية على الأقل بعد كل من الكلمات المحجوزة (int أو float)

وبعد ذلك تقوم بقية البرنامج بطباعة محتوى المتغيرات a,b ولأننا لم نخزن في هذين المتغيرين أية بيانات فإن ما نحصل علية ليس إلا بعض المخلفات الموجودة في الذاكرة، وهي بلا معنى على الإطلاق والشكل التالي يوضح مثالا لهذة المخلفات كنتيجة لتشغيل البرنامج

CODE

```
22348
476.950
```

تخزين البيانات في المتغيرات (Assignment):

في البرنامج السابق لاحظنا أنه لابد من أن تختزن عددا ما في المتغير العددي الذي أعلنا عنه ويتم ذلك باستخدام عبارة التخصيص (assignment statement) ويوضح الشكل التالي برنامجا قمنا فية بالإعلان عن متغيرين و إختزان بيانين عدديين في كل منهما ، ثم نطبع محتويات هذين المتغيرين على الشاشة.

CODE

```
#include <stdio.h>
main()
{
/* variable declaration*/
int a;
float b;
/* Assignment */
a=1000;
b=796.5;
/*Display output */
printf("%d\n",a);
printf("%f\n",b);
}
```

وعند تنفيذ هذا البرنامج نحصل على النتيجة الموضحة بالشكل

CODE

```
1000
796.5
```

عبارة التخصيص (Assignment statment) :

إن العبارة

a=1000;

يمكن قرائتها على النحو التالي:

" خصص العدد 1000 للمتغير a "

ومن الجائز أن نخصص متغيرا لمتغير آخر ، ومعنى ذلك أننا نضع نسخة من المغير الأول في المتغير الثاني.

أمالو فمنا بتخصيص تعبير حسابي يحتوي على متغيرات وقيم عددية لمتغير ما فأن البرنامج في هذه الحالة يجري عملية تقييم للتعبير الحسابي ويضع قيمته النهائية في المتغير المقصود.

ويوضح المثال التالي ثلاث عمليات تخصيص كالآتي:

1- تخصيص قيمة عددية للمتغير " "a

2- قسمة محتويات المتغير " a" على 2 وتخصيص الناتج للمتغير " "b

3- جمع محتويات كل من " b " ، "a" وتخصيصها للمتغير " c".

CODE

```
#include<stdio.h>
main()
{
int a;
float b,c;

a=1024;
b=a/2.0;
c= b+a;

printf("The result is   %f\n",c);

}
```

ومن الملاحظ في هذا البرنامج أنه قد تم إعلان المتغيرين " c " ، "b" في عبارة واحدة وقمنا بإستخدام علامة الفاصلة للفصل بينهما.

ونتيجة البرنامج النهائية هي طباعة محتويات المتغير " c"

التخصيص المتعدد:

يمكننا في لغة C أن نخصص قيمة ما لأكثر من متغير في نفس العبارة كالآتي:

;a = b = c = 24

تخصيص قيم ابتدائية المتغيرات:

يمكن أيضا شحن المتغير بقيمة إبتدائية أثناء الأعلان عنه كالآتي:

float a = 5.6 ;

ونقوم بشحن المتغيرات بقيمة ابتدائية عند الإعلان عنها لضمان تنظيف وعاء المتغير من مخلفات الذاكرة.

5-التحكم في الفورمات

عند تنفيذ البرامج السابقة رأينا أن الأعداد الحقيقية تظهر على الشاشة متبوعة بعدد من الأصفار.
ومما لا شك فيه أننا في التطبيقات الحقيقية قد نرغب في تعديل هذه الصورة لتحتوي على رقمين عشريين أو رقم واحد، كما أننا قد لا نرغب في مشاهدة الأصفار الزائدة على الإطلاق. فالأفضل أن نرى العدد

25.0000000

مكتوبا بالصورة

25

ويتم ذلك باستخدام علامات خاصة لتعديل مواصفات الفورمات يطلق عليها علامات تعديل الفورمات (format modifiers)
والشكل التالي يوضح طرقا مختلفة لطباعة الرقم الحقيقي 25

الوصف | التأثير على شكل الناتج | النتيجة

%.0f | حذف جميع الأصفار الزائدة | 25
%.3f | إظهار ثلاث أصفار فقط بجوار العلامة | 25.000

ومن هذا الجدول نلاحظ أن الرقم السابق للحرف f يتحكم في عدد الأصفار التي تظهر على يمين العلامة العشرية.

والمثال التالي يوضح برنامجا لطباعة العدد 75 بصور مختلفة

CODE

```
#include <stdio.h>
main()
{
float x;
x=75;
printf("%.0f\n",x);
```

```
printf("%.1f\n",x);
printf("%.2f\n",x);
}
```

وعند تنفيذه نحصل على الناتج الموضح بالشكل

CODE

```
75
75.0
75.00
```

والآن ماذا لو كان العدد المطلوب طباعته محتويا على كسر عشري مثل

25.8756

واستخدمنا تعديلا في الفورمات لطباعته ؟ إن ما يحدث في هذه الحالة هو تقريب العدد إلى عدد من الخانات العشرية بحسب الرقم المستخدم في الفورمات

ويمكنك تجربة البرنامج التالي لطباعة قيمة الكسر ¾ في صور مختلفة وبدرجات مختلفة من التقريب.

CODE

```
#include <stdio.h>
main()
{
printf("%.0f\n",3.0/4.0);
printf("%.1f\n",3.0/4.0);
printf("%.2f\n",3.0/4.0);

}
```

وناتج هذا المثال هو الموضح بالشكل

CODE

```
1
0.8
0.75
```

6- متغير الرمز (char variable) :

ذكرنا فيما سبق أننا سنلتقي مع أنواع أخرى من المتغيرات، والآن بعد أن تعرفنا على المتغيرات العددية نتعرف على نوع آخر من المتغيرات وهو ما يصلح لتخزين رمز واحد (character) ويطلق على هذا النوع من المتغيرات الاسم (char).

و الرموز التي يمكن تخزينها في هذا النوع من المتغيرات فهي قد تكون رموزا موجودة في جدول الكود آسكي (ASCII code table) وهو جدول يحتوي الرموز المعتمدة من هيئة المواصفات القياسية الأمريكية، ويضم جميع الحروف والأرقام والعلامات الخاصة وعلامات التحكم والأرقام الكودية المناظرة لكل منها.

و للإعلان عن متغير رمز بالاسم "a" مثلا نستخدم العبارة التالية:

```
char a;
```

ولتخصيص رمز ما لهذا المتغير فإننا نضعه بين علامتي اقتباس مفردتين كالآتي

```
a = 'Z'
```

و بهذا التخصيص أصبح متغير الرمز "a " محتويا على الحرف " Z"، ولطباعة محتويات المتغير الرمز نحتاج إلى توصيف جديد للفورمات وهو التوصيف" %c"

,هذا التوصيف يحتوي على الحرف الأول من كلمة character وهو مخصص لطباعة الرموز.

والمثال التالي يوضح برنامجا قمنا فيه بالإعلان عن متغير رمز بالاسم " first_letter"
ثم خصصنا لهذا المتغير الحرف " A" ثم طبعنا محتويات المتغير باستخدام التوصيف %c.
وعند تنفيذ هذا البرنامج فإنه يطبع على الشاشة الحرف A.

CODE

```
#include <stdio.h>
main()
```

```
{
char first_letter;
first_letter = 'A';
printf("%c\n",first_letter);
}
```

ومن أهم خصائص متغير الرمز أننا نستطيع أن نطبعه بطريقتين مختلفتين:

1- باستخدام الفورمات c.%

2- باستخدام الفورمات d%.

في الحالة الأولى كما رأينا في المثال السابق فأن الرمز المختزن هو الذي يظهر على الشاشة.

أما لو استخدمنا الفورمات d% فإن رقم الكود آسكي المناظر للرمز هو الذي يظهر على الشاشة.

والمثال التالي يوضح استخدام نوعي موصفات الفورمات مع متغير الرمز

CODE

```
#include <stdio.h>
main()
{
char first_letter;
first_letter = 'A';
printf("%c\n",first_letter);
printf("%d\n",first_letter);

}
```

6- تخزين الحرفيات والمؤشرات(String & Pointer) في لغة C:

إلى الآن تعلمنا كيفية التعامل مع المتغير الرمز ، والمتغيرات العددية.

و سنتعلم الآن نوعا جديدا من المتغيرات وهو المتغير الحرفي (string) ولا ريب أن كل المبرمجين الذين سبق لهم التعامل مع لغات أخرى مثل البيسك قد تعودوا على إستخدام هذا النوع من المتغيرات...

ولكن لغة C لاتحتوي متغيرا من هذا النوع بل تختزن الحرفيات بطرية خاصة كرموز متتابعة.

وأحدى الطرق المستخدمة لتخزين الحرفيات هي استخدام نوع خاص من المتغيرات يسمى المؤشر (pointer)، الذي يشير إلى أول رمز في الحرفي المختزن في الذاكرة كما يتم تمييز نهاية الحرفي برمز خاص ، وبذلك يمكن الأستدلال على أولة و آخرة.

المؤشرات (pointers):
المؤشر متغير كسائر المتغيرات ولكنه يختلف عنها فيما يختزنه من بيانات، فالمؤشر لا يختزن البيانات العادية مثل الأرقام أو الرموز. ولكنة يختزن فقط عناوين الذاكرة، ومن هنا جاء اسمة كمؤشر لأنة يشير مباشرة إلى أحد خانات الذاكرة.

وتختلف طريقة الأعلان عن المؤشر بحسب البيان المخزون فيه، فإذا كان المؤشر يشير إلى عدد صحيح مثلا فيسمى في هذة الحالة (مؤشر إلى عدد صحيح) ويعلن عنه بعبارة كالعبارة التاليه:
int *a;

أما لو كان يشير إلى رمز من الرموز فيسمى في هذه الحالة مؤشر إلى رمز أو (character pointer) ويعلن عنه بعبارة كالعبارة التاليه:

char * a
ونلاحظ أنه في كلتا الحالتين فأن " a" هو اسم المؤشر الذي اخترناه وهو يأتي مسبوقا بالعلامة " * " التي تدل على كونه مؤشرا. أما نوع المؤشر فهو يتم تحديدة وفقا لنوع البيان المشار إلية فقد يكون عددا صحيحا (int) أو حقيقيا (real) أو رمزا (char) وهي الأنواع الثلاثة التي عرفناها في لغة C

والمثال التالي يوضح كيفية تخصيص متغير حرفي وطباعته على الشاشة، ونلاحظ أنه لطباعة الحرفي نقوم بطباعة المؤشر الذي يشير إليه مع استخدام توصيف جديد للفورمات وهو (%s)

CODE

```
#include <stdio.h>
main()
{
char *a;
a = "Welcome C programmer";
printf("%s\n",a);
}
```

وناتج البرنامج هو الموضح بالشكل التالي

CODE

```
Welcome C programmer
```

وعند الأعلان عن مؤشر بالعبارة

char *a;

فأن هذا يؤدي إلى خلق الآتي:

1- المؤشر " a " الذي يشير إلى أول حرف من الحرفي.

2- المتغير" *a " الذي يحتوي على أول حرف من الحرفي.

من المهم لمن كان جديدا على لغة C أن يحاول التدقيق في مفهوم المؤشرات فهي أداة قوية تساعد المبرمج على إنجاز مهام كثيرة في أقل وقت ممكن، ولكنها في نفس الوقت تمثل مصدرا للأخطاء ما لم تستخدم بصورة مناسبة.

والمثال التالي يساعدنا على تعميق مفهوم المؤشر، فهو يبدأ بإعلان عن متغير رمز " a " ثم يختزن فيه الحرفي " Hello again "، ويطبع محتويات العديد من المتغيرات المتعلقة بالحرفي.

CODE

```
#include <stdio.h>
main()
{
char *a;
a = "Hello again";
printf("%s\n",a);
printf("%c\n",*a);
printf("%d\n",a);
printf("%p\n",a);
printf("%d\n",*a);
}
```

الفصل الثالث : الإدخال و الإخراج (I/O)

حتى الآن قمنا بالطباعة على الشاشة باستخدام الدالة **printf** لطباعة الخرج وفقا لصيغة محددة (فورمات). و لكن قد يحتاج المبرمج لإدخال البيانات في وقت تنفيذ البرنامج ويستلزم ذلك استخدام دوال لإدخال البيانات، وهو ما سنتعرض له الآن بشيء من التفصيل.

أما الدالة المناظرة للدالة **printf**، والمخصصه لإدخال البيانات وفقا لصيغة محددة، فهي الدالة **scanf** ، ويعتبر الحرف **" f"** الذي تنتهي به كل من الدالتين هو الحرف الأول من كلمة **"format "**

والمثال التالي يوضح كيفية استخدام الدالة **scanf** لإدخال البيانات.

CODE

```
#include <stdio.h>
main()
{
float x,y,z;
scanf (“%f”,&x);
scanf (“%f”,&y);
z=x+y;
printf(“the sum of the numbers you entered is : %.2f\n”,z);
}
```

يبدأ البرنامج بالإعلان عن ثلاثة متغيرات من النوع الحقيقي **" x,y,z "** ثم يتم استقبال قيمة المتغير **" x"** من لوحة الأزرار بموجب العبارة :

CODE

```
scanf ("%f",&x)
```

ثم يتم استقبال المتغير الثاني**"y"** بعبارة مماثلة ثم يتم جمع المتغيرين **" x,y "** وتخصيص الناتج للمتغير **" z"**
وفي النهاية نطبع قيمة المتغير **" z "** المحتوي على المجموع.

عند تشغيل البرنامج سوف ينتظر إدخال قيمة المتغير **"x"** فإذا أدخلنا العدد المطلوب وأتبعنا ذلك بالضغط على الزر **Enter** ، فإن البرنامج يتوقف مرة أخرى منتظرا إدخال قيمة المتغير **" y "** متبوعة بالضغط على الزر **Enter** وعندئذ يوافينا البرنامج بالنتيجة.

والآن فلننظر بتفحص لإحدى العبارات التي تحتوي على الدالة scanf فنلاحظ ما يلي:

1- ضرورة استخدام توصيف للفورمات بنفس الأسلوب المتبع مع الدالة printf وفي المثال السابق قد استخدمنا التوصيف " %f " الذي يناظر المتغير الحقيقي "x" أو " ."y"
2- لم تستخدم الدالة المتغير " x" أو "y" صراحة بل استخدمت صورة محورة منه وهي (x&) ، وهذه الصورة الجديدة تسمى مؤشر العنوان (address operator) وهي عبارة عن عنوان المتغير لا المتغير نفسه. أما المؤثر الجديد & فيسمى مؤثر العنوان إلى (address-of operator)

إدخال أكثر من قيمة متغير واحد بنفس العبارة:
تماما كما مع الدالة printf يمكننا مع الدالة scanf استخدام عبارة واحدة ودالة واحدة لاستقبال قيم عدة متغيرات كما في المثال التالي

CODE

```
#include <stdio.h>
main()
{
float x,y,z;
scanf (“%f%f”,&x,&y);
z=x+y;
printf(“the sum of the numbers you entered is : %.2f\n”,z);
}
```

نلاحظ أن الجزء الخاص بالفورمات (والواقع بين علامتي الاقتباس) يحتوي على توصيفين للفورمات " %f% f " بنفس عدد المتغيرات التي تأتي مفصولة عن بعضها البعض باستخدام الفاصلة " , " (أنظر العبارة المحتوية على الدالة scanf)

ومن الملاحظات الهامه أن ترتيب الفورمات في الدالة scanf يجب أن يكون بنفس ترتيب المتغيرات التي سيتم إدخالها. وهذه الملاحظة غير واضحة في المثال السابق نظرا لأن كلا المتغيرين المراد إدخالهما من نفس النوع.

الفصل بين المدخلات:
في المثال السابق كانت المتغيرات تدخل كل على حدة متبوعا بالضغط على الزر Enter ، ولكن ماذا لو أردنا إدخال المتغيرين في سطر واحد؟؟؟

المثال التالي يوضح الطريقة الجديدة لإدخال المتغيرين في سطر واحد ويتم الفصل بينهما بفاصلة ، ويتم ذلك بكتابة الفاصلة في البرنامج نفسه كفاصل بين توصيفات الفورمات.

CODE

```
#include <stdio.h>
main()
{
int x;
float y,z;
scanf (“%d,%f”,&x,&y);
z=x+y;
printf(“the sum of  %d  and %f  is : %.2f\n”,x,y,z);
}
```

رسالة لتنبية مستخدم البرنامج :

من عيوب الدالة **scanf** أنها لا يمكن استخدامها لطباعة أي نص على الشاشة كما مع دوال الدخل في لغة مثل البيسك . وهذا معناه ضرورة الاستعانة بدالة الطباعة

printf إذا أردنا أن نطبع على الشاشة رسالة تنبه المستخدم إلى أن البرنامج ينتظر إدخال بيان مثل:

Please Enter the number

في المثال التالي نرى صورة محسنة لإدخال قيمتي متغيرين مع طباعة الرسائل اللازمة لتنبية المستخدم.

CODE

```
#include <stdio.h>
main()
{
float x,y,z;
printf(“Enter the first number : ”);
scanf (“%f”,&x);
printf(“Enter the second number : ”);
scanf (“%f”,&y);
z=x+y;
printf(“the sum of the numbers you entered is : %.2f\n”,z);
}
```

ملاحظة هامة:

لا يوصى باستخدام الدالة scanf لاستقبال الحرفيات من لوحة المفاتيح، حيث يتطلب الأمر احتياطات كثيرة . ولاستقبال الحرفيات من لوحة المفاتيح توجد طرق أفضل سيأتي الحديث عنها.

طرق جديدة للتعامل مع الحرفيات:

لقد رأينا من قبل كيف يمكننا تخزين الحرفي بالاستعانة بالمؤشرات حيث يشير المؤشر إلى الرمز الأول من الحرفي المختزن في الذاكرة . هذا من ناحية بداية الحرفي . أما من ناحية نهاية الحرفي فإن البرنامج من تلقاء نفسه يضيف إلى مؤخرة الحرفي الرمز الصفري (NULL character) وهو الرمز رقم صفر في جدول الكود آسكي.

ويفيد هذا الرمز في تمييز مؤخرة الحرفي و بالتالي في تحديد طوله لتسهيل التعامل معه قراءة وكتابة ومعالجة بالطرق المختلفة.

وفي الواقع أن هذه الطريقة برغم ما تحتويه من تفصيلات فنية دقيقة لكنها أفضل من الطرق المستخدمة في اللغات الأخرى التي تتوفر بها المتغيرات الحرفية (string variables)، فمع هذه الطريقة في لغة C لا توجد أية قيود على طول الحرفي المستخدم.

وهنا سنتناول طريقة أخرى لتمثيل الحرفيات وهي مصفوفة الرموز (character arrays) ومن اسم هذه الطريقة يتضح أنه يتم حجز خانات الذاكرة اللازمة للحرفي مقدما.

الأعلان عن مصفوفة الرموز:

لننشئ مصفوفة من الرموز فإننا نبدأ بالإعلان عنها في بداية البرنامج . ويشمل الإعلان اسم المصفوفة وسعتها (size) أي الحد الأقصى لعدد الرموز بها .

فمثلا الجملة التالية يتم فيها الإعلان عن مصفوفة رموز بالاسم (employee_name):

CODE

```
char employee_name[20];
```

في هذا الإعلان يتم حجز عشرين خانة في الذاكرة تتسع كل منها لرمز واحد ، كما تخصص الخانة الأخيرة للرمز الصفري(NULL).

ولشحن هذه المصفوفة بأحد الحرفيات، فإن دالة خاصة تستخدم لهذا الغرض وهي الدالة (strcpy (a ,) حيث " a " هو اسم

مصفوفة الرموز، و "b" هو الحرفي المراد تخزينه في المصفوفة.

والمثال التالي يوضح الإعلان عن مصفوفة رموز بالاسم " a" تتسع لعشرين رمزا ثم ننسخ إلى عناصرها الحرفي " Hello again " وفي النهاية نطبع محتويات المصفوفة باستخدام دالة الطباعة printf مع استخدام الفورمات المناسبة للحرفيات %s.

CODE

```
#include <stdio.h>
#include <string.h>
main()
{
char a[20];
strcpy(a,"Hello again");
printf(" %s\n",a);
}
```

ومن الملاحظ في هذا البرنامج ظهور توجيه جديد هو :

CODE

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

إن هذا التوجيه يصبح لازما عند استخدام الدالة strcpy حيث أن الملف " string.h" هو الملف الذي يحتوي على تعريف الدالة " strcpy " وبقية دوال الحرفيات، ويطلق على هذا الملف اسم ملف العناوين للحرفيات " string header file"

والآن سنتناول طريقة عمل البرنامج بشيء من التفصيل، ولنبدأ بدالة الطباعة printf . فعندما تتعامل مع مصفوفة الرموز " a " فغنها تقرأ و تطبع عناصر المصفوفة واحدا بعد الآخر حتى تصادف الرمز الصفري فتتوقف.

أما عن طريقة تخزين الرموز في المصفوفة فهناك نقاط جديرة باهتمامنا .

إننا عندما نعلن عن المصفوفة " a[20]" فإن عناصر المصفوفة تأخذ الأرقام المسلسلة من " 0 " إلى " 19" كالتالي:

CODE

```
a[0], a[1],……..,a[19]
```

ولا يشترط عندما نخصص أحد الحرفيات لهذه المصفوفة أن نشغل جميع العناصر (الخانات) ففي المثال السبق مثلا عدد رموز الحرفي

كانت 11 حرفا و استخدم العنصر الثاني عشر من المصفوفة لتخزين الرمز الصفري.

طرق مختلفة لإدخال الحرفيات:

ذكرنا من قبل أنه لا يوصى باستخدام الدالة scanf لإدخال الحرفيات من لوحة المفاتيح .والآن سنستعرض البدائل المختلفة التي تتيحها اللغة لإدخال الحرفيات.

الدالة gets :
يعتبر اسم الدالة اختصارا للعبارة " get string " وهي تقوم بقراءة الحرفي المدخل من لوحة المفاتيح ، وتضيف إليه الرمز الصفري (NULL) ثم تقوم بتخصيصه للمتغير المطلوب و الذي يستخدم كدليل للدالة. وصيغة الدالة كالآتي:
gets(a);
حيث " a " مصفوفة الرموز.

والمثال التالي يوضح استخدام هذه الدالة.

CODE

```
#include <stdio.h>
main()
{
char employee_name[20];
gets(employee_name);
printf(" Employee: %s\n",employee_name);
}
```

وعندما يبدأ البرنامج سوف ينتظر منك إدخال الحرفي المطلوب وهو اسم الموظف " employee name " ثم يخصصه لمصفوفة الرموز المكونة من عشرين عنصرا. وفي النهاية يطبع البرنامج الاسم على الشاشة كتأكيد لتمام الاستلام و الحفظ.

ويمكننا هنا إدخال الاسم محتويا على مسافات خالية وذلك على العكس من الدالة scanf التي تعتبر المسافة الخالية مماثلة للضغط على المفتاح Enter.
ولكن هناك قيد على الحرفي المدخل إذ يجب مراعاة ألا يزيد طوله عن الحجم المحجوز للمصفوفة مع العلم بأن المترجم يستغل خانة من المصفوفة لتخزين الرمز الصفري. ففي هذا المثال لا يمكن إدخال أكثر من 19 رمز فقط.

الدالة fgets :

تستخدم هذه الدالة لقراءة حرفي من ملف أو جهاز للدخل (input device). ويتم تعريف الملف (أو جهاز الإدخال) ضمن صيغة الدالة نفسها كالتالي:

CODE

```
fgets( a, n, stdin );
```

حيث " a " مصفوفة رموز

و " n " الحد الأقصى للرموز المدخلة.

و " stdin" اسم جهاز الدخل القياسي (لوحة المفاتيح)

ويمكن بالطبع استبدال جهاز الدخل القياسي stdin باجهزة أخرى حسب الموقف و لكننا في الوقت الحالي سوف نكتفي بلوحة المفاتيح كجهاز للدخل .

عند استخدام هذه الدالة في إدخال الحرفيات فإنها تضيف إلى مؤخرة الحرفي كلا من :

1- علامة السطر الجديد (\n).

2- الرمز الصفري (NULL).

ولذلك فإنه مع هذه الدالة لابد وأن نخصص عنصرين في المصفوفة لهذين الرمزين .

والمثال التالي يوضح استخدام هذه الدالة

CODE

```
#include <stdio.h>
main()
{
char employee_name[20+2];
fgets(employee_name,22,stdin);
printf(" Employee: %s\n",employee_name);
}
```

طرق مختلفة لطباعة الحرفيات:

سنتناول الآن بعضا من دوال الخرج التي تصلح لطباعة الحرفيات بطريقة مبسطة.

الدالة puts:

اسم هذه الدالة إختصار للعبارة " put string " وهي الدالة المقابلة لدالة الدخل gets وصيغة هذه الدالة كالآتي:

CODE

```
puts ( a);
```

حيث a ثابت حرفي ، أو مصفوفة رموز.

والمثال التالي يوضح استخدام هذه الدالة لطباعة رسالة لتنبيه المستخدم قبل استخدام الدالة gets لاستقبال البيان

CODE

```
#include <stdio.h>
main()
{
char employee_name[20+1];
puts("Enter employee_name:  ");
gets(employee_name);
puts(employee_name);
}
```

وعند تنفيذ البرنامج نلاحظ أن الاسم المدخل قد جاء على سطر مستقل بعد رسالة التنبيه . وذلك لأن الدالة puts عندما تطبع حرفيا على الشاشة تطبع في مؤخرته علامة السطر الجديد " \n"

الدالة fputs:

هذه الدالة هي المناظرة للدالة fgets فهي تستخدم لإرسال الخرج إلى ملف أو جهاز الخرج المذكور اسمه ضمن بارامترات الدالة.

وصيغة الدالة كالآتي:

CODE

```
fputs( a, stdout );
```

حيث a مصفوفة رموز أو ثابت حرفي.
و " stdout " اسم جهاز الخرج القياسي وهو جهاز الشاشة.

ومن الطبيعي استبدال جهاز الشاشة كما يتطلب التطبيق.

والدالة fputs تختلف عن puts في أنها لا تطبع علامة السطر الجديد في نهاية الحرفي.

الفصل الرابع : المؤثرات

إن لغة C – كأي لغة أخرى – تتعامل مع التعبيرات، وتتكون التعبيرات من الثوابت و المتغيرات المرتبطة ببعضها البعض بواسطة المؤثرات.

والمؤثرات تنقسم إلى عدة أنواع هي:

1- المؤثرات الحسابية (Arithmetic Operators)

2- المؤثرات العلاقية (Relational Operators)

3- المؤثرات المنطقية (Logical Operators)

المؤثرات الحسابية (Arithmetic Operators) :

تتيح لغة C استخدام العديد من المؤثرات الحسابية، منها المؤثرات الأساسية والتي تقوم بالعمليات الحسابية الأساسية وهي الموضحة أدناه

+ (الجمع)

- (الطرح)

* (الضرب)

/ (القسمة)

وبالإضافة لهذه المؤثرات توجد مؤثرات خاصة بلغة C وهي الموضحة أدناه

% (باقي القسمة)

-- (النقصان)

++(الزيادة)

وسنتناول بشيء من التفصيل استخدام هذه المؤثرات الخاصة.

مؤثر باقي القسمة

الصورة العامة لاستخدام هذا المؤثر هي : x % y

ويكون الناتج هو باقي قسمة " x " على " y " ، والشكل التالي يوضح استخدام المؤثر والناتج

CODE

```
7%3
```

ويكون الناتج لهذه العملية هو "1 " وهو باقي القسمة للعددين 3/7

مؤثرات الزيادة والنقصان (Decrement & Increment) :

من مزايا لغة ال C انها تستعمل الأداتين الحسابيتين ++ و – لزيادة القيم بمقدار 1أو انقاصها بمقدار 1 والمثال التالي يوضح طريقة الاستعمال :

CODE

```
X++;
++X;
```

ومعناه اضافة قيمة 1 الى X ويمكن كتابته بصورة مكافئة على النحو التالي :

CODE

```
X=X+1;
```

وبالطريقة نفسها يمكن انقاص 1 من قيمة X على النحو التالي :

CODE

```
--X;
X--;
```

وهو يكافيء الصورة :

CODE

```
X=X-1;
```

لكن هناك فرقا في سرعة التنفيذ , فالتعبير ++X اسرع من التعبير X=X+1

وهذه هي الفائدة من جراء استخدام مثل هذه الأدوات

المؤثرات العلاقية (Relational Operators) :

يرجع اسم المؤثرات العلاقية الى العمليات المختصة بالقيم التي بينها علاقات وهو اجراء عمليات مقارنة بين كميات حسابية او رمزية , وتكون نتيجة منطقية وهي اما نعم (true) أو لا (false)

وفي لغة السي تعامل النتيجة (false) على انها صفر " 0 " وتأخذ النتيجة (true) أية قيمة غير الصفر والمشهور أنها " 1 " .
ويبين الشكل التالي المؤثرات العلاقية :
نفرض ان : int a=b=3

المؤثرات المنطقية (Logical Operators) :

الفصل الخامس : اتخاذ القرار

تعرضنا حتى الآن لبرامج متتالية الأوامر، حيث ينفذ الكمبيوتر العبارات الموجودة في البرنامج بالترتيب الذي وردت به .

ولكن في الحياة العملية نحتاج لاتخاذ بعض القرارات تبعا لشروط معينة، ومن هنا ظهرت الحاجة لوجود طرق لجعل البرنامج قادرا على تغيير تسلسل تنفيذ التعليمات تبعا للشروط المطلوبة.

وسنتعرض هنا لطرق اتخاذ القرار في لغة الC وكيفية تغيير تسلسل التنفيذ تبعا للشروط الموضوعة.

العبارة الشرطية البسيطة (if statement):

تكوين العبارة الشرطية البسيطة كما هو موضح بالشكل

CODE

```
if ( condition )
statement;
```

حيث (condition) هو الشرط و (statement) هو القرار المراد اتخاذه عند تحقق الشرط المعطى.
وعندما ترغب في تنفيذ أكثر من عبارة بتحقق الشرط نستبدل العبارة التي تمثل القرار المراد اتخاذه ببلوك به العبارات المراد تنفيذها.
ولتوضيح استخدام العبارة الشرطية البسيطة أنظر البرنامج التالي

CODE

```
#include <stdio.h>
main()
{
float sum;
printf("\n Enter the Sum : ");
scanf("%f",sum);
if (sum >50)
 printf ("\n The student had passed");
}
```

وفي هذا البرنامج يطبع الكمبيوتر رسالة ليسأل المستخدم عن مجموع الطالب وبعد ذلك يقوم بمقارنتها بالشرط اللازم للتأكد من النجاح (وهو تجاوز المجموع 50)فإذا تحقق الشرط يطبع الكمبيوتر رسالة للمستخدم يعلمه أن الطالب ناجح،

العبارة الشرطية الكاملة (if ..... else statement)

إن اتخاذ القرارات في الحياة العملية ليست بالسهولة التي ذكرت في البرنامج السابق، إذ نحتاج في معظم الأحيان لاتخاذ اجراء تبعا لشرط معين، واتخاذ إجراء آخر إذا لم يتحقق هذا الشرط.

لو نظرنا للبرنامج السابق لوجدنا سؤالا ملحا : ماذا لو كان مجموع الطالب أقل من 50 ؟؟
الاجابة على هذا السؤال هي أن الطالب يكون راسبا. ولكن البرنامج لا يتضمن أمرا بإعطاء حالة الرسوب، لأننا استخدمنا عبارة الشرط البسيطة والتي تستجيب لشرط واحد.
وسنتعرض الآن لعبارة مركبة كما في البرنامج التالي:

CODE

```
#include <stdio.h>
main()
{
float sum;
printf("\n Enter the Sum : ");
scanf("%f",sum);
if (sum >50)
 printf ("\n The student had passed");
else
 printf("\n The student had failed");
}
```

وفي هذا البرنامج استخدمنا العبارة الشرطية الكاملة والتي تأتي على الصورة الموضحة بالشكل التالي

CODE

```
if ( condition)
statement-1;
else
statement-2;
```

حيث أن (condition) هو الشرط
و (statement -1) هي عبارة النتيجة الأصلية.
و (statement -2) هي عبارة النتيجة البديلة.

ومنطق اتخاذ القرار هنا هو : " لو تحقق الشرط يقوم الكمبيوتر بتنفيذ عبارة النتيجة الأصلية أما لو لم يتحقق الشرط فيقوم الكمبيوتر بتنفيذ عبارة النتيجة البديلة"

وهكذا -باستخدام العبارة الشرطية الكاملة - تمكننا من اتخاذ القرار لحالتين متضادتين ، والآن ماذا لو كانت النتيجة الأصلية و النتيجة البديلة تتضمنان أكثر من أمر للكمبيوتر؟
في هذه الحالة نحتاج إلى احتواء عبارات النتيجة الأصلية بين قوسين من أقواس البلوكات، وهو الموضح بالشكل

CODE

```
if ( condition)
{
statement 1;
statement 2;
                              .
                .
statement n;
}
else
{
statement 1;
statement 2;
                              .
                .
statement m;
```

```
}
```

نلاحظ أن عبارة النتيجة تم استبدالها ببلوك النتيجة، والمثال التالي هو البرنامج السابق بعد تعديل عبارات النتائج لتصبح بلوكات، وذلك ليتمكن البرنامج من إعطاء تقرير بالنجاح أو الرسوب متضمنا النسبة المئوية باعتبار المجموع الكلي 1000 في حالة النجاح أو رسالة تفيد بأنه لا يمكن احتساب النسبة المئوية لطالب راسب.

CODE

```
#include <stdio.h>
main()
{
float sum;
printf("\n Enter the Sum : ");
scanf("%f",sum);
if (sum >50)
 {
  printf ("\n The student had passed");
  printf("\n The percentage is : %f",(sum/1000)*100)
 }
else
 {
printf("\n The student had failed");
printf("\ There is no percentage for failed student !");
 }
}
```

لو افترضنا انه قد طلب منك - كمبرمج - عمل برنامج يمكنه احتساب التقديرات اعتمادا على مجموع الطالب، في هذه الحالة نستخدم عبارة شرطية أيضا ولكن بها عدد من الشروط وعدد مناظر من النتائج. أو ما يطلق علية العبارة الشرطية المتدرجة.

والشكل التالي يوضح التكوين العام للعبارة الشرطية المتدرجة

CODE

```
if  ( condition -1)
statement -1;
else if ( condition-2)
statement-2;
else if( condition-3)
```

```
statement-3;
……..
else
statement-n;
```

الاختيار متعدد البدائل (statement switch)

يعتبر الاختيار المتعدد البدائل بديلا للعبارة الشرطية المتدرجة التي تعرضنا لها سابقا، والواقع أن الاختيار المتعدد البدائل أعد خصيصا ليكون أسهل استخداما من العبارة الشرطية المتدرجة.ويتميز عنها بأنه أفضل توضيحا.

والشكل التالي يوضح الصورة العامة للاختيار متعدد البدائل

CODE

```
switch (variable)
{
case value1;
 statement 1;
 break;
case value2;
 statement 2;
 break;
case value 3;
 statement 3;
 break;
……..
default:
 statement;
}
```

وكما نرى فإن الاختيار المتعدد البدائل يبدأ بكلمة (switch) يليها متغير الاختيار والذي تحدد قيمته الاختيار الذي سيتم تنفيذه، ويلي ذلك قوس بلوك كبير يحتوي داخله بلوكات صغيرة كل منها يمثل اختيارا من البدائل المطروحة و كل بلوك من بلوكات البدائل يبدأ بكلمة (case) متبوعة بقيمة لمتغير الاختيار - والتي تمثل الشرط - وبعد ذلك تأتي عبارة النتيجة.

ويختتم بلوك البديل بكلمة (break) والغرض من هذه الكلمة هو منع الكمبيوتر من تنفيذ عبارة النتيجة التالية!!!
وقد تبدو هذه العبارة غريبة للوهلة الأولى ويتبادر للذهن سؤال ملح : ألم يتحقق الشرط الأول مثلا فماذا يدفع الكمبيوتر لتنفيذ بقية عبارات النتائج؟؟

والإجابة عن هذا السؤال هي أن عبارة الاختيار متعدد البدائل لا ترسل للكمبيوتر أمرا بالتوقف بعد تحقق أي شرط فيها، لذا لزم الاستعانة بكلمة (break)

وبعد نهاية بلوكات البدائل تأتي كلمة (default) متبوعة بعبارة أو بعبارات ينفذها الكمبيوتر في حالة عدم تحقق أي من الشروط السابقة.

الفصل السادس : الحلقات التكرارية

كثيرا ما نحتاج في البرامج إلى تكرار أمر موجه للكمبيوتر عددا من المرات، وتوفر لغة C عدة وسائل تمكن المبرمج من أداء هذا التكرار.

وعادة ما تسمى هذه الوسائل " الحلقات التكرارية "، ويوجد العديد من الحلقات التكرارية في لغة C سنتناول منها هنا

1- الحلقة for (for loop).

2- الحلقة while (while loop).

3- الحلقة do-while loop (do…. while).

وفيما يلي سنتناول كل حلقة بالدراسة من حيث الشكل العام و أسلوب الاستخدام وأمثلة توضيحية.

الحلقة for (for loop):

تستخدم الحلقة for لتكرار أمر معين (أو مجموعة من الأوامر) عددا من المرات وتحتاج الحلقة إلي ثلاث عناصر أساسية (انظر الشكل التالي)

CODE

```
for ( counter statement; condition; step)
```

و هذه العناصر هي:

1- العداد (counter) : وظيفة العداد هي تسجيل عدد مرات التكرار.

2- الشرط (condition): والشرط الذي يحدد نهاية التكرار إذ يظل التكرار قائما حتى ينتفي الشرط.

3- الخطوة(step) : وهي القيمة التي تحدد عدد مرات التكرار.

والشكل التالي يوضح برنامجا قمنا فيه باستخدام الحلقة for :

CODE

```
#include <stdio.h>
main()
{
int counter;
for ( counter=1;counter<=20;counter++)
 printf("%d",counter);
}
```

ومن البرنامج السابق نجد أن الحلقة for بدأت بكلمة (for) متبوعة بقوسين بينهما ثلاثة عبارات تفصل بينها علامة الفاصلة المنقوطة.

العبارة الأولى تخزن القيمة الابتدائية في العداد.

والعبارة الثانية هي الشرط وهنا الشرط أن قيمة العداد أقل من أو تساوي 20.

أما العبارة الثالثة فهي تحدد الخطوة، وفي هذا البرنامج يزاد العداد بمقدار 1 كل مرة تنفذ فيها الحلقة.

والبرنامج السابق ينتج عنه طباعة الأرقام من 1 إلى 20.

ملاحظات:

1- العبارات الثلاثة المكونة لحلقة for يجب أن تفصل عن بعضها بالفاصلة المنقوطة، وهذا الخطأ من الأخطاء الشهيرة جدا في عالم البرمجة لذا وجب توخي الحذر.

2- في حالة تكرار أكثر من أمر يتم استبدال العبارة التي تلي بداية الحلقة for (في المثال السابق هي العبارة (") printf ",counter(%d";;) ببلوك يحتوي العبارات المراد تنفيذها.

الحلقة while (while loop):

في هذه الحلقة التكرارية نحتاج إلى الشرط فقط وطالما كان هذا الشرط متحققا استمرت الحلقة في التكرار..

والصورة العامة للحلقة while موضحة بالشكل التالي

CODE

```
while ( conditon )
{
statement 1;
statement 2;
 .
 .
statement n;
}
```

حيث (condition) هو الشرط اللازم لأداء التكرار، والعبارات بداخل أقواس البلوكات هي العبارات المراد تكرارها.

والمثال الموضح بالشكل التالي يوضح استخدام الحلقة while لطباعة الأعداد من 1 إلى 20

CODE

```
#include <stdio.h>
main()
{
int counter=1;
while ( counter <=20 )
{
 printf("%d",counter);
counter++;
}
}
```

من المثال السابق يمكننا استخلاص النتائج التالية عن الحلقة :while

1- تخصيص القيمة الابتدائية للعداد تتم خارج الحلقة .while

2- زيادة العداد تتم داخل الحلقة while

الحلقة التكرارية do-while:

تختلف هذه الحلقة عن الحلقتين السابقتين في مكان كتابة الشرط ، حيث يكتب الشرط هنا بعد العبارات المطلوب تكرارها.

والشكل التالي يوضح الصورة العامة للحلقة do-while

CODE

```
do
{
statement 1;
statement 2;
 .
 .
statement n;
}
while ( conditon
```

وأهم ملاحظة على الحلقة التكرارية do-while أنها تنفذ العبارات المطلوب تكرارها مرة واحدة على الأقل حتى ولو كان الشرط غير متحقق !!!

وتفسير ذلك أن التحقق من الشرط يتم بعد التنفيذ وليس قبله كما في الحلقتين السابقتين.

ويلاحظ أنه عند تنفيذ البرنامج قد يختلف عنوان الذاكرة المطبوع. ورمز الفورمات " p% " هو رمز خاص بالمؤشرات ويؤدي إلى طباعة عنوان الذاكرة بالنظام السداسي عشري.