لأي استفسار او وجود تعديل الرجاء مراسلتي على البريد الالكتروني

الاسم : MLN4EVER

البريد الالكتروني : MLN4EVER9@hotmail.com

اهداء :

اهدي هذا الكتاب لوالدتى العزيزة ..... حفظها الله من كل مكروه وسوء وادام عليها الصحة والعافية .

وكذلك احب ان اهديه الى (منتديات الفريق العربي للبرمجة) .

مقدمة :

أتمنى ان يفيد هذا الكتاب جميع المهتمين بلغة ال C وال++C ويكون سهل الفهم ..... فهو يبدأ في لغة ال C من الصفر ثم ينتقل بعد ذلك الى لغة ال++C

(مع ذكر الفرق بينها)

فأرجو ان يحوز هذا الكتاب على رضاكم .

وفقني الله واياكم الى فعل الخير .....

ملاحظة : هذا الكتاب مجاني

مكتبة طارق
www.t9tt.com
أكبر مكتبة عربية مجانية على الانترنت

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 ال C 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

```
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

```
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

```
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 له فورمات 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

```
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

```
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 ال C باسماء 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

```
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

```
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.

أما لو قمنا 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

```
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

```
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 :

```
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);
}
```

المتغير	الفورمات	معنى المتغير مع هذه الفورمات	الخرج
A	%s	الحرفي نفسه	Hello again
*a	%c	أول رمز من الحرفي	H
A	%d	عنوان الذاكرة الموجود به أول رمز من الحرفي بالنظام العشري	122
A	%p	عنوان الذاكرة الموجود به أول رمز من الحرفي بالنظام السداسي عشري	7A
*a	%d	الكود أسكي الخاص بأول رمز من الحرفي	72

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.

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 :

```
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 :

```
#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:

```
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:

```
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 7/3

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

الرمز	المعنى	مثال	النتيجة
>	أكبر من	a>3	(false) أي "0"
<	أصغر من	a<b	(false) أي "0"
>=	أكبر من أو يساوي	a>=b	(true) أي "1"
<=	أصغر من أو يساوي	a<=3	(true) أي "1"
==	يساوي	a = = b	(true) أي "1"
!=	لايساوي	a ! = b	(false) أي "0"

المؤثرات المنطقية (Logical Operators) :

الرمز	المعنى	مثال	النتيجة
&&	AND " و"	10 > 8 && 9 > 7	1
||	OR " أو"	10 < 8 || 7 < 8	1
!	NOT " لا "	!(10 == 8)	1

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 ال C 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 (" %d",counter);) 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 النتاتج 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.

الفصل الأول مميزات لغة C++ عن لغة C :

تدعم لغة C++ أسلوب برمجة الكائنات الموجهة وبالإضافة لذلك تمتاز لغة C++ بالعديد من المزايا والتي سنتناولها فيما يلي بشيء من التفصيل.

المزيد من الحرية في الإعلان عن البيانات :

في لغة C يشترط الإعلان عن المتغيرات في مستهل البرنامج، وعند الحاجة لمتغير جديد لابد من الرجوع لأول البرنامج و الإعلان عنه.

ومع لغة C++ ينتفي هذا الشرط إذ يتمكن المبرمج من تعريف المتغيرات وقت الحاجة إليها وفي أي مكان.

ويمكن للمبرمج الاقتصاد في استخدام الذاكرة باستخدام هذه الميزة ولتوضيح ذلك فلننظر للمثال التالي

CODE

```
#include <stdio.h>
main()
{
int l;
scanf("%d",&l);
if (l>5)
 {
  int j;
  printf("Enter the second number  ");
  scanf("%d",j);
  printf(" the result is ",j*l);
 }
```

نلاحظ هنا أن المتغير " j " تم الإعلان عنه وقت الحاجة لاستخدامه فقط
موفرة بذلك ميزة رائعة، لأنة يمكن للمبرمج أن يقتصد في استخدام الذاكرة وذلك بتعريف المتغيرات التي يحتاجها عند تحقق شرط معين مثلا داخل هذا الشرط.
وبذلك لا نحجز للمتغير مكانا إلا عند الحاجة إليه فقط.

إعطاء قيم الابتدائية لمعاملات الدوال:
تسمح لغة C++ بإعطاء قيم ابتدائية لمعاملات الدوال عند تعريفها، وعندما يتم استدعاء الدالة في البرنامج بدون معاملات يتم استعمال القيم الابتدائية، أما إذا أعطى المبرمج قيما للمعاملات فإنها تستخدم بدلا من القيم الابتدائية.
ويسمح في هذه الحالة باستدعاء الدالة بأكثر من طريقة، والمثال التالي يلقي مزيدا من الضوء على هذه الميزة.

CODE

```
#include <iostream.h>
void Add(int a=5,int b=9){
cout << a+b;
};
main()
{
Add(4,6);
Add();
}
```

في هذا المثال عرفنا الدالة (Add()) والتي تجمع متغيرين من النوع الصحيح، والدالة تأخذ قيم المتغيرين عند استدعائها مثل عبارة الاستدعاء الأولى وفيها أخذ المتغير الأول القيمة (4) والثاني القيمة (6)
أما عبارة الاستدعاء الثانية فلم نعط للدالة قيما لمتغيراتها، وفي هذه الحالة تستخدم الدالة القيم الابتدائية المعلن عنها عند تعريف الدالة.

كتابة التعليقات:

سهلت لغة C++ عملية كتابة التعليقات حيث أصبح بالإمكان كتابة التعليقات بعد العلامة " // " ودون التقيد بعلامة في نهاية التعليق كما كان سابقا في لغة C حيث كان التعليق يكتب دوما بين العلامتين " /* " و " */ ".

ويجب مراعاة أن التعليق لو جاء في عدة أسطر لزم وضع العلامة " // " في أول كل سطر ، ومن الممكن في هذه الحالة وضع التعليق بين العلامتين المعتادتين لتلافي الخطأ عند نسيان وضع العلامة " // " في أول السطر.

القدرة على إنشاء واستخدام الفصائل:

وهذه الميزة من أهم المميزات والتي تجعل لغة C++ تدعم أسلوب برمجة الكائنات الموجهة ويتم إنشاء الفصيلة باستخدام الكلمة المحجوزة (class) وذلك تبعا للصورة العامة الموضحة بالشكل التالي

CODE

```
class class_name{
private:
private data and functions
public :
public data and functions
}
```

حيث يعطى اسم الفصيلة بعد الكلمة المحجوزة (class)ويتوالى بعد ذلك تعريف البيانات والدوال.

تحديد درجة حماية البيانات :

تتيح لغة C++ تحديد درجات لحماية البيانات وذلك على مستوى الفصيلة، وتتحدد درجة الحماية باستخدام الكلمات (public , private , protected) ويوضح الجدول التالي درجات الحماية

المختلفة

محدد الحماية	متاح لنفس الفصيلة	متاح للفصائل المشتقة	متاح للكائنات من فصائل أخرى
public	نعم	نعم	نعم
protected	نعم	نعم	لا
private	نعم	لا	لا

و بالتقدم في البرمجة سنألف استخدام محددات الحماية، وسنتعرض لها بشيء من التفصيل عند الحديث عن الفصائل والكائنات.

دوال البناء والهدم (constructors and destructors):

كما ذكرنا سابقا فالفصيلة تتكون من بيانات و دوال تتعامل مع هذه البيانات، وتتيح لغة C++ للمبرمج أن ينشئ دالتين خاصتين تسمى إحداهما دالة البناء (constructor) وهي دالة تنفذ تلقائيا عند الإعلان عن كائن من هذه الفصيلة. وتظهر فائدة هذه الدالة عندما نرغب في تخصيص قيم ابتدائية لبيانات الفصيلة.

أما الدالة الأخرى فهي دالة الهدم (destructor) وتنفذ تلقائيا عند انتهاء استخدام الفصيلة وتستخدم هذه الدالة لتحرير أجزاء من الذاكرة كنا نستخدمها أثناء استعمال الفصيلة ولم نعد بحاجة إليها، أو لتنفيذ سطور معينة عند الانتهاء من استخدام الفصيلة.

ودالة البناء تحمل نفس اسم الفصيلة، فمثلا لو كان اسم الفصيلة (Ball) كانت دالة البناء تحمل الاسم (Ball).

أما دالة الهدم فتأتي بنفس اسم الفصيلة مسبوقا بالعلامة(~) فللفصيلة السابقة دالة الهدم تحمل الاسم (~Ball).

التوريث (Inheritance) :

من أقوى خصائص برمجة الكائنات الموجهة خاصية التوريث. ونعني هنا توريث فصيلة إلى فصيلة

أخرى.

وهنا ترث الفصيلة المشتقة (derived class) من الفصيلة الأساسية (parent class) كل بياناتها ودوالها ويمكن التعديل بعد ذلك في خصائص الفصيلة المشتقة لتناسب الاحتياجات الجديدة، بإضافة المزيد من البيانات والدوال.

وبذلك نجد أن برمجة الكائنات الموجهة تعفي المبرمج من إعادة بناء البرامج من الصفر بل يعتمد على ما سبق لإنجاز البرامج الجديدة، فتمكنه من استخدام الفصائل السابقة و عمل فصائل جديدة للاستفادة منها مستقبلا.

الدوال الصديقة(friend functions):

عندما تعلن فصيلة عن دالة صديقة أو عدة دوال صديقة فإنها تسمح لهذه الدوال باستعمال البيانات الأعضاء فيها ولا تسمح لغير هذه الدوال بذلك.

وكذلك الحال عندما تعلن فصيلة عن فصيلة صديقة، فإنها تسمح لجميع دوال الفصيلة الصديقة باستخدام بيانات الفصيلة الأساسية.

وسيأتي الحديث بالتفصيل عن الدوال الصديقة في فصل الفصائل والكائنات.

الفصل الثاني: أساسيات البرمجة بلغة C++ :

توجد لكل لغة أساسياتها التي ينبغي الإلمام بها قبل كتابة البرامج بواسطتها، وهذا الفصل يوضح هذه الأساسيات مثل: هيكل البرنامج، المتغيرات، الإدخال والإخراج.
وبجانب هذا يلمس الباب العديد من مزايا اللغة من كتابة التعليقات، والعمليات الحسابية، وتحويل البيانات. وغيرها من المزايا.

البناء الأساسي للبرنامج:

لنلق نظرة متعمقة على البرنامج التالي

CODE

```
#include <iostream.h>
void main()
{
cout << " Every age has its own language . . .";
}
```

وبغض النظر عن صغر حجمه فإنه يوضح البناء الأساسي للبرنامج في لغة C++ ويتضح ذلك عندما نتناوله بالتفصيل كما يلي.

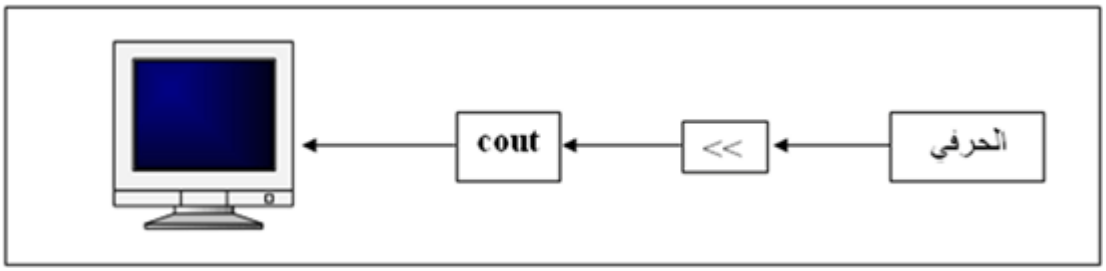
الدوال:

الدوال تشكل البلوكات الأساسية لبناء البرنامج ، ويتكون البرنامج هنا من دالة واحدة وهي الدالة الرئيسية (main()) والدوال في بناء برمجة الكائنات الموجهة قد تكون أعضاء في فصائل محددة أو تكون مستقلة بذاتها ، والدالة الرئيسية دالة مستقلة بذاتها حيث لا تنتمي لأي فصيلة.

والدالة لها اسمها ويليه قوسين توضع بينهما معاملات الدالة، ونلاحظ أن الدالة الرئيسية في هذا المثال ليس لها معاملات.
أما الكلمة المحجوزة (void) والتي تسبق اسم الدالة فتوضح أن الدالة ليس لها قيمة ترجع بها ، بخلاف بعض الدوال التي نخصص لها نوعا من البيانات بحيث ترجع قيمة من نوع هذا البيان.

وعبارات الدالة نفسها تحاط بقوسين خاصين" } " ، " { " يسميان بقوسي البلوكات.
و الدالة الرئيسية هي أول ما ينفذه الكمبيوتر عند تنفيذ البرنامج.

عبارات الإخراج
في البرنامج السابق نلاحظ أننا قد استخدمنا عبارة لطباعة الحرفيات، وتختلف هذه العبارة عن العبارات التي تعودنا عليها عند استخدام اللغات الأخرى فهنا لم نستخدم دالة خاصة لتطبع الخرج على الشاشة، بل قمنا بكتابة الحرفي بين علامتي تنصيص واستخدمنا الكلمة المحجوزة (cout) والمعامل (>>) والعبارة التي استخدمناها للطباعة يفهمها الكمبيوتر بكما هو موضح بالشكل التالي



موجهات ما قبل المعالجة(preprocessor directives):
العبارة التي بدأ بها البرنامج (#include <iostream.h>) ليست في الواقع جزءا منه بل هي إعلان عن ملف يحتوي على تعريفات العديد من الدوال التي نحتاجها أثناء البرمجة.
وتبدأ العبارة بما يسمى بموجه قبل المعالجة وهو الرمز (#) والأمر الذي يليه موجه للمعالج مباشرة وهناك جزء من المعالج يتعامل مع مثل هذة الأوامر.
ويقوم بتنفيذ الأوامر الصادرة إلية لتتم عملية المعالجة اعتمادا على المعلومات التي وفرها للمعالج.

التعليقات (Comments):

عند كتابة برنامج بأية لغة يستحب كتابة التعليقات لتوضيح العبارات المكونة للبرنامج.
والمبرمج الذكي يحرص دوما على كتابة كل ما يمكنه من تعليقات على برنامجه ليسهل عليه تصحيحه أو استخدام بعض أجزاءه إن دعت الحاجة لذلك.

وتسمح لغة C++ بكتابة التعليقات بطريقتين تسهلان على المبرمج وضع ما يشاء من التعليقات على البرنامج .

والطريقة الأولى هي كتابة التعليق بعد العلامة " // " حيث يتجاهل المترجم السطر الذي يلي هذه العلامة.
ولكن لو تجاوز التعليق السطر لزم إضافة المزيد من الرموز " //" أمام كل سطر من التعليقات.
وللاستغناء عن الحاجة لكتابة العلامة " // " أمام كل سطر يمكن للمبرمج أن يستخدم الطريقة الثانية وهي كتابة التعليق بين العلامتين " /*" و " */" ويسمح في هذة الحالة كتابة التعليق على أكثر من سطر دون التسبب في الخطأ، طالما كان التعليق بين العلامتين المذكورتين.
والمثال التالي يوضح كيفية استخدام الطريقتين

CODE

```
// this is the first method

// this is the
// first method

/* this is the second method */

/* this is
the second
method*/
```

في المرة الأولى استخدمنا الطريقة الأولى ولم يتجاوز التعليق السطر فلم نستخدم سوى علامة تعليق واحدة. أما في المرة الثانية تجاوز التعليق السطر فلزم علينا استخدام علامة تعليق ثانية .

وفي المرة الرابعة استخدمنا الطريقة الثانية لكتابة التعليقات ومع ان التعليق تجاوز السطر فلم نستخدم علامة تعليق جديدة لكل سطر بل اكتفينا بوجود العلامة " /*" في بداية التعليق والعلامة " */" في نهايته.

المتغيرات في لغة C++ (تحدثنا مسبقا عن هذا الموضوع في السي ولكن باختصار)

تمثل المتغيرات الجزء الأهم من أي لغة، والمتغيرات ليست إلا أسماء رمزية لأوعية اختزان البيانات في الذاكرة. وحسب أنواع البيانات المختزنة تنقسم المتغيرات إلى أنواع عديدة، ويخضع اختيار أسماء المتغيرات لقواعد هي:

1- ألا يكون اسم المتغير 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 المتغير (MY_NUMBER) يختلف عن الاسم (my_number) وكلاهما يختلف عن الاسم (My_Number).

وسنتناول هنا بالشرح بعض الأنواع الأساسية من المتغيرات التي لا غنى للمبرمج عنها.

المتغيرات العددية الصحيحة (integer variables):

لنتعرف على كيفية تعريف المتغيرات العددية الصحيحة نلقي نظرة على البرنامج التالي

CODE

```
#include <iostream.h>
void main()
{
int var1;      //define var1
int var2;  //define var2

var1=20;       //assign value to var1
var2=var1+10;

cout<< “var1+10 is ”; //output text
cout<<var2;   // output value of var2

}
```

قمنا في هذا البرنامج بتعريف متغيرين من النوع الصحيح بالاسمين " var1" و " var2" .
ولتعريف المتغير نستخدم الكلمة المحجوزة " int" وهي اختصار " integer" أو عدد صحيح، متبوعة باسم المتغير والذي يتبع القواعد المحددة السابق ذكرها لاختيار أسماء المتغيرات.

ونلاحظ في هذا البرنامج أن المتغيرات تم تعريفها في أول البرنامج وليس هذا شرطا في لغة C++ إذ تتيح لنا إمكانية تعريف المتغيرات وقت الحاجة في أي مكان نشاء.

وبعد عبارة الإعلان عن المتغيرين ننتقل إلى عبارة أخرى وهي عبارة تخصيص القيم للمتغيرات ، حيث نخزن قيما فعلية في الأماكن التي حجزناها سالفا .
وفي هذا البرنامج نخزن القيمة " 20" في المتغير الأول، والعبارة المستخدمة لتخصيص قيمة المتغير الثاني ليست مباشرة، إذ يقوم المعالج بأداء عملية حسابية قبل تخصيص القيمة، حيث يجمع القيمة " 10" على المتغير الأول.
ولإخراج قيمة المتغيرات على الشاشة نستخدم العبارتين الأخيرتين .

المتغيرات الرمزية (char variables):

المتغير الرمزي هو المتغير الذي يسمح بتخزين رمز فيه، والرمز في لغة الكمبيوتر هو كل ما يرد في جدول الكود آسكي والذي يحدد الرموز التي يمكن للكمبيوتر التعامل معها.

والرموز تحتوي الحروف الكبيرة والصغيرة والأعداد بالإضافة إلى العديد من رموز التحكم.

ولتعريف متغير رمزي نستخدم العبارة

```
char variable_name;
```

حيث (variable_name) هو اسم المتغير الرمزي، ويخضع أيضا للقواعد العامة لتسمية المتغيرات.

وعند تخصيص قيمة لمتغير رمزي نستخدم علامتي اقتباس مفردتين كما بالعبارة التالية

```
variable = ‘A’;
```

وهذه العبارة تخصص الرمز (A) للمتغير (variable)

تتابعات الهروب (escape sequences) :

من إمكانيات لغة C++ استخدام بعض رموز الحروف لأداء مهام خاصة ولنلق نظرة على البرنامج التالي :

CODE

```
#include <iostream.h>
main()
{
char var1='A';
char var2='\t';
char var3='\n';

cout << var1;
cout << var2;
cout << var1;
cout << var3;

}
```

في هذا البرنامج نعلن عن ثلاثة متغيرات من النوع الرمزي ونخصص الرمز (A) للمتغير الأول.
أما المتغيرين الثاني و الثالث فنخصص لكل منهما رمز جديد مكون من علامة الشرطة المائلة العكسية (back slash) والتي تعني للمترجم أن الرمز الذي يليها ليس رمزا عاديا بل يحمل دلالة خاصة، والرمز الذي يلي علامة الشرطة المائلة العكسية يقوم بأداء عملية خاصة ، فمثلا إذا جاء بعد علامة الشرطة المائلة العكسية الرمز (n) كانت النتيجة الانتقال لسطر جديد.
ولو جاء الحرف (t) كانت النتيجة طباعة عدد من المسافات الفارغة و مماثلة للتي تنتج من استخدام المفتاح (tab).

وهناك العديد من تتابعات الهروب والتي نلخصها في الجدول التالي

تتابع الهروب	المعنى أو المفتاح المناظر
\a	إطلاق صفارة الجهاز مرة واحدة
\b	العودة للخلف مسافة رمز واحد (مثل استعمال المفتاح backspace)
\n	الانتقال لسطر جديد (مماثل لاستعمال المفتاح Enter)
\t	الانتقال للأمام مسافة عدة رموز (مماثل لاستخدام المفتاح tab)
\\	طباعة علامة الشرطة المائلة العكسية (\ )
\"	طباعة علامة اقتباس مزدوجة
\'	طباعة علامة اقتباس مفردة

المتغيرات العددية العشرية(floating point variables)

تعرفنا في جزء سابق على المتغيرات العددية الصحيحة، ونعني بالصحيحة تلك التي لا تحتوي على كسور أي لا توجد بها علامة عشرية.
والآن نتعرف على المتغيرات العددية العشرية أو كما تسميها بعض الكتب المتغيرات الحقيقية وهي المتغيرات التي تسمح لنا بالتعامل مع الأعداد التي تحوي الكسور أو العلامات العشرية، ومن هنا جاء اسمها.

ولتعريف متغيرات من النوع الحقيقي نلقي نظرة على البرنامج التالي:

CODE

```
#include <iostream.h>
void main()
{
float var1;      //define var1
float var2;  //define var2

var1= 50.79;       //assign value to var1
var2= var1 + 56.9;

cout<< “var1+ 56.9 is ”; //output text
cout<<var2;   // output value of var2
}
```

وتعريف المتغيرات الحقيقية لا يختلف عن المتغيرات الأخرى إذ يتم بنفس الطريقة وباستخدام الكلمة المحجوزة (float) وهي اختصار لكلمة (floating point) والتي تعني علامة عشرية، وهي ما يميز الأعداد الحقيقية.

ويتم التعامل مع الأعداد الحقيقية بنفس طريقة التعامل مع المتغيرات العددية الصحيحة.

وتجب ملاحظة أنه لو خصصنا ناتج عملية حسابية تحتوي على متغيرات حقيقية و أخرى صحيحة لابد من أن يكون الناتج مخصصا لمتغير من النوع الحقيقي، وإلا حصلنا على أخطاء عند التنفيذ.

المؤشرات:

فكرة المؤشرات قد تبدو للوهلة الأولى صعبة ولكن مع الفهم الجيد يصبح استعمال المؤشرات في غاية السهولة.

والفكرة الأساسية هي أن ذاكرة الكمبيوتر مقسمة إلى أماكن لتخزين البيانات المختلفة ولكل مكان من هذه الأماكن عنوانه الخاص، وهذا العنوان يفهمه الكمبيوتر بصورته العددية (أي أن هذه العناوين ما هي إلا أعداد).

والبرنامج عندما يعلن عن متغير من نوع معين فإن الكمبيوتر يحجز مكانا له في الذاكرة. وبالتالي

يكون لكل متغير من متغيرات البرنامج عنوانه الخاص.

والمؤشر هو متغير يحمل العنوان، ويمكننا تعريف مؤشرات لكل أنواع المتغيرات في لغة C++ .

ولتعريف مؤشر ما يذكر نوعه أولا ثم اسم المتغير مسبوقا بالعلامة (*)

وذلك كما في العبارة

float *ptr;

وفي هذه العبارة قمنا بتعريف مؤشر لعدد حقيقي، واسم المؤشر هو ptr.

ويمكننا بنفس الطريقة تعريف مؤشرات لكل أنواع البيانات التي توجد في لغة C++.

عبارات الإدخال باستخدام cin

تعرفنا على العبارة المستخدمة في الإخراج ونتناول الآن العبارة التي تستخدم للإدخال. والمثال التالي يوضح العبارة قيد الاستخدام

CODE

```
#include<iostream>
void main()
{
int ftemp;
cout << " Enter temperature in Fahrenheit: ";
cin >> ftemp;
int ctemp= (ftemp-32) * 5/9;
cout<<"The temperature in Celsius is :  " <<ctemp<<"\n";
}
```

والإدخال في هذا البرنامج يتم بالعبارة التي تحوي الكلمة المحجوزة (cin) ويليها المؤثر (>>) ثم اسم المتغير الذي سنحتفظ فيه بالقيمة المدخلة.

وتنتظر عبارة الإدخال المستخدم ليضغط على الرمز المراد إدخاله متبوعا بالمفتاح (Enter) ليضع القيمة في المتغير المحجوز سابقا.

السجلات (Structures):

السجل عبارة عن مجموعة مترابطة من البيانات كما في المصفوفات ولكن السجل يحتوي بيانات مختلفة الأنواع وليست من نوع واحد كما في المصفوفة.

والسجل يتكون من عدة حقول (fields) تحوي البيانات المختلفة ويستخدم السجل لتخزين بيانات مترابطة متكررة، كما في قاعدة البيانات حيث تتكون قاعدة البيانات من سجلات بكل سجل منها نفس الحقول، ولكن قيم تلك الحقول تختلف من سجل لآخر.

والإعلان عن السجل يتم كما هو موضح بالشكل التالي

CODE

```
struct structure_name
{
 type   field1;
 type   field2;
 …
};
```

حيث (stucture_name) هو اسم السجل وبداخل السجل تتوالى الحقول المختلفة الأنواع (field1, field2,…..) ولكل حقل نوعه الخاص.

وبتعريفنا للسجل يمكننا بعد ذلك تعريف متغيرات من نوع هذا السجل لاستخدامها في البرنامج حسب الحاجة

ويتم تعريف المتغيرات من السجل كما هو موضح بالشكل التالي الذي يوضح تعريف متغير (var1) من نوع السجل (structure1)

CODE

```
struct structure1
{
 type   field1;
 type   field2;
 …
}  var1;
```

ويمكننا تعريف أي عدد من المتغيرات من نوع هذا السجل كما يتطلب البرنامج

والآن كيف نتعامل مع السجلات؟؟

إننا نحتاج مثلا لتخزين قيمة معينة في أحد الحقول، وفي هذه الحالة نستخدم المؤثر (.) والمثال التالي يوضح عمل سجل باسم (Student) وتخصيص اسم (Mohammed) لحقل الاسم (name)

CODE

```
#include<iostream.h>
struct Student
{
 char* name;
 int number;
};
main()
{
Student Sdt1;
Std1.name=”Mohammed”;
Cout << Std1.name;
}
```

وعند تنفيذ البرنامج تقوم العبارة الأخيرة بطباعة الاسم " Mohammed" وهو الذي قمنا بتخزينه في الحقل (name) من المتغير (Std1).

الفصل الثالث : اتخاذ القرارات

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 C++ كيفية تغيير تسلسل التنفيذ تبعا للشروط الموضوعة.

العبارة الشرطية البسيطة (if statement):

تكوين العبارة الشرطية البسيطة كما هو موضح بالشكل التالي

```
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 <iostream.h>
main()
{
float sum;
cout<< " Enter the sum ";
cin >> sum;

if(sum>50)
cout<<" 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 <iostream.h>
main()
{
float sum;
cout<< " Enter the sum ";
cin >> sum;

if(sum>50)
 cout<<" The student had passed";
else
 cout<<" The student had failed";

}
```

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

```
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 التالي

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 نفس البرنامج السابق بعد تعديل عبارات النتائج لتصبح بلوكات، وذلك ليتمكن البرنامج من إعطاء تقرير بالنجاح أو الرسوب متضمنا النسبة المئوية في حالة النجاح أو رسالة تفيد بأنه لا يمكن احتساب النسبة المئوية لطالب راسب.

CODE

```
#include <iostream.h>
main()
{
float sum;
cout<< " Enter the sum ";
cin >> sum;

if(sum>50)
 {
  cout<<" The student had passed";
  cout<< " His points are "<< sum/100;
 }
else
 {
cout<<" The student had failed";
cout<<" No points are calculated for failed student !!";
 }
}
```

العبارة الشرطية المتدرجة (if-else- if Ladder):

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 الرابع: 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 كما هو موضح بالشكل التالي

```
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 <iostream.h>
main()
{
int counter;
for ( counter=1;counter<=20;counter++)
 cout<<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 (cout << counter;) 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 <iostream.h>
main()
{
int counter=1;
while ( counter <=20 )
{
 cout<< counter;
counter++;
}
}
```

من المثال السابق يمكننا استخلاص النتاتج 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.

الفصل الخامس: الدوال و الماكرو (Function & Macro) :

معنى الدالة:

الدالة هي مجموعة من الأوامر المحددة التي تعطى للكمبيوتر وغالبا ما تكون هذه الأوامر مرتبطة بأداء وظيفة محددة.

والدوال تمنح اللغة بعض المزايا مثل:

1- توفر في حجم البرنامج، حيث نستعيض عن تكرار عدد السطور التي تؤدي مهمة الدالة بإعادة استدعاء الدالة فقط.

2- توفر مكتبة دائمة للمبرمج، حيث يمكن الاحتفاظ بالدوال وإعادة استخدامها حين الحاجة دون كتابتها من البداية.

3- يؤدي استخدام الدوال الى زيادة وضوح البرنامج وتسهيل عملية تصحيحه، حيث يبدو البرنامج مع استخدام الدوال مقسما إلى أجزاء محددة واضحة أو ما يسمى بالبلوكات.

وسنتناول الآن طريقة استخدام الدوال في البرامج، ليتم استخدام الدالة يجب أولا الإعلان عنها، وبعد عملية الإعلان عن الدالة يمكننا استخدامها بواسطة ما يسمى باستدعاء الدالة، ولابد من كتابة التعليمات التي تؤديها الدالة فيما يعرف باسم تعريف الدالة.

والمثال التالي مثال (1) يوضح استخدام دالة عرفها المبرمج

CODE

```
#include <iostream.h>

void DrawLine();   (1)
void main()
{
cout << " This is the output of the function :  " << '\n';
DrawLine();   (2)
}

void DrawLine()
{
for (I=1;I<=40;I++) (3)
 cout<<'*';
}
```

وفي هذا المثال استخدمنا الدالة المسماة (DrawLine)والتي صممناها لرسم سطر من العلامة (*)

وفي السطر المشار إلية برقم (1) في البرنامج السابق قمنا بالإعلان عن الدالة أو (function declaration) وهو مجرد ذكر اسم الدالة وأنواع المتغيرات التي تأخذها ونوع القيمة التي تعيدها.

وفي هذا المثال لا تأخذ الدالة أية متغيرات وهو الموضح بالقوسين الفارغين بعد اسم الدالة مباشرة، ولا تعيد الدالة قيما أيضا وهو الموضح بالكلمة (void) والتي تسبق اسم الدالة.

أما السطر المشار إلية برقم (2) ففيه قمنا باستدعاء الدالة أو (function calling) والمراد منه توجيه الأمر للكمبيوتر بتنفيذ مضمون الدالة.

ومجموعة السطور المشار إليها بالرقم (3) هي تعريف الدالة أو (function definition)، وتعريف الدالة يتم بين قوسي بلوكات " { " و " } " ويتضمن التعليمات المطلوب من الدالة تنفيذها، وهنا تقوم الدالة بتنفيذ طباعة العلامة " *" أربعين مرة متتالية على نفس السطر مما يشكل الخرج

المطلوب من الدالة.

ومن أهم الملاحظات التي يجب وضعها دائما في الاعتبار :

1- يمكن أن يأتي تعريف الدالة قبل استدعائها، وفي هذه الحالة لا حاجة بنا للإعلان عن الدالة في سطر مستقل.

2- لا يمكن بأية حال أن يتم استدعاء الدالة قبل الإعلان عنها أو تعريفها.

أنواع الدوال:

تصنف الدوال تبعا للقيمة التي تعيدها، وتبعا لذلك نجد الأنواع التالية:

1- دوال أعداد صحيحة (int functions) وهي التي تعيد بيانا من النوع العددي الصحيح integer)).

-2 دوال أعداد حقيقية (float functions) والقيمة المعادة في هذه الحالة تكون من النوع الحقيقي . ((float

-3دوال حرفيات (string functions) وتعيد بيانا من النوع الحرفي وهو سلسلة من الرموز .

4- دوال الرموز (char functions) وتعيد بيانا من النوع الرمزي (.(char

-5 دوال لا تعيد قيما (void function) ولا تعيد قيما من أي نوع

مثال)2):

CODE

```
#include<iostream.h>
float sum(float x, float y)
{
float result;
result = x + y;
return result;  (1)
};
void main()
{
cout << sum( 4.5 , 8.9 );
}
```

نلاحظ أننا قمنا بالإعلان عن الدالة (sum) وتعريفها قبل الدالة الرئيسية (main)
وتأخذ الدالة (sum) متغيرين من النوع الحقيقي وتقوم بجمعهما وتعيد الناتج في صورة عدد حقيقي.

وعملية إعادة الناتج من الدالة تتم في السطر المشار إلية بالرقم (1) وتتم باستخدام الكلمة المحجوزة (return) ويليها المتغير المراد إعادة قيمته

معاملات الدوال:
بعض الدوال تحتاج عند استدعائها إلى متغيرات مثل الدالة (sum) في المثال(2)
والمعاملات هي القيم التي تحتاجها الدالة لأداء مهمتها عليها ، في هذه الحالة جمع المعاملين.
وعلى العكس من ذلك توجد دوال لا تأخذ معاملات مثل الدالة (DrawLine) التي استخدمناها في المثال (1)

معاملات الدالة الرئيسية (main function arguments)
كل البرامج التي تعرضنا لها حتى الآن تستخدم الدالة الرئيسية (main) بدون معاملات أي تكون متبوعة بقوسين فارغين، وبعد معرفتنا بالدوال نتساءل ألا يمكن أن نستخدم الدالة الرئيسية بمعاملات؟

والجواب على هذا السؤال أنه يمكن بالفعل استخدام الدالة الرئيسية بمعاملات والمثال التالي مثال(3) يوضح برنامجا فيه الدالة الرئيسية تم استدعائها بمعاملاتها

CODE

```
#include < iostream.h >
main (int argc, char*argv[])
{
if(afgc!=3)
{
 cout<<" Arguments number error ….";
 exit(1);
}
cout<<"the first argument is"<<argv[1]<<'\n';
cout<<"the second argument is"<<srgv[2];
}
```

نلاحظ أن الدالة الرئيسية تستخدم معاملين هما (argc) وهو من النوع العددي الصحيح، ويستخدم لتخزين عدد المعاملات التي سيكتبها المستخدم عند استدعاء الدالة، والاسم (argc) اختصار لعدد المعاملات (argument counter)

أما المعامل الثاني فهو (argv) وهو عبارة عن مصفوفة حرفيات تختزن المعاملات التي يكتبها المستخدم عند استدعاء البرنامج.

وتكتب المعاملات الخاصة بالدالة الرئيسية عند استدعاء البرنامج فمثلا لو كان البرنامج السابق في صورته القابلة للتنفيذ محفوظا باسم (prog1.exe)

وكتبنا السطر الآتي لتنفيذه:

```
C:> prog1 First Second
```

فإن المعاملات تختزن في مصفوفة المعاملات بالشكل الموضح بالجدول أدناه

القيمة المختزنة : Prog1 *** عنصر المصفوفة : argv[0]

القيمة المختزنة : First **** عنصر المصفوفة : argv[1]

القيمة المختزنة : Second ** عنصر المصفوفة : argv[2]

ويقوم البرنامج بالتأكد من عدد المعاملات المعطاة فإذا كان غير ثلاثة طبع البرنامج رسالة خطأ.

و لو كان العدد مساويا لثلاثة (كما هو الحال في السطر المعطى بعالية) فإن البرنامج يطبع قيمة المعامل الأول .

ثم ينتقل لسطر جديد ليطبع المعامل الثاني.

ويكون خرج البرنامج كالآتي

```
the first argument is First
the second argument is Second
```

الماكرو:

الماكرو هو مجموعة أوامر مصممة لأداء غرض معين، والمثال التالي يوضح برنامجا استخدمنا فيه ماكرو لحساب مربع العدد

CODE

```
#include <iostream.h>
#define SQUARE(A) A*A
main()
{
int x;
cout<< " Please enter a number to calculate it's square ";
cin >> x;
cout << " The square of "<< x << "is :" << SQUARE(x);
}
```

والبرنامج هنا ينتظر من المستخدم إدخال قيمة عددية للحصول على مربعها،
ويحسب البرنامج قيمة مربع العدد باستخدام الماكرو المعلن في السطر الثاني

```
#define SQUARE(A) A*A
```

ولحساب القيمة يقوم البرنامج باستدعاء الماكرو

```
SQUARE(x);
```

والماكرو يشبه الدالة إلى حد ما وإن كان هناك اختلاف بينهما نتناوله الآن بالتفصيل.
يمر البرنامج بعدة مراحل قبل الحصول على النسخة القابلة للتنفيذ منه وهذه المراحل هي:
1- كتابة البرنامج وحفظة باستخدام أحد برامج التحرير (Editors) وتسمى هذه العملية بكتابة الكود (coding) ويحتفظ بالملف في هذه الحالة بالإمتداد " .cpp" ويسمى بالملف المصدر (source file).
2- عملية الترجمة (compilation) وينتج عن هذه العملية البرنامج الهدف الذي يحمل عادة الأمتداد " . "OBJ
3- عملية الربط بمكتبة اللغة (Linking) وينتج عن هذة العملية البرنامج التنفيذي الذي يحمل الأمتداد "EXE". والبرنامج التنفيذي هو البرنامج الذي يتم تنفيذه بمجرد إدخال اسمه .

والدالة بعد كتابتها في البرنامج تمر بمرحلة الترجمة إلى لغة الآلة ولا تنفذ إلا في مرحلة الربط، أما الماكرو وأثناء عملية الترجمة فيتم استبداله في كل سطر يتم استدعاؤه فيه بنتيجته النهائية ولا ينتظر مرحلة التنفيذ كالدالة.

ويمتاز الماكرو عن الدالة بالسرعة والسهولة في الكتابة بالإضافة لاستخدامه أنواعا محايدة من البيانات (فلم نشترط نوعا معينا من المتغيرات في تعريفنا للماكرو (SQUARE(A)) فهو لا يحتاج إلى تحديد النوع كما في الدوال.
وذلك بالإضافة إلى حصولنا على ملف تنفيذي أصغر في حالة استعمال الماكرو.

وبصفة عامة يوصى باستخدام الماكرو في العمليات القصيرة التي لا تتعدى سطرا واحدا.

الفصل السادس :المصفوفات

المصفوفة هي مجموعة من العناصر من نفس النوع، وتكون عناصر المصفوفة مرتبة بحيث يمكننا الوصول لأي عنصر نريده بتحديد ترتيبه في المصفوفة.

والمصفوفات تنقسم لنوعين فهناك المصفوفات ذات البعد الواحد، والمصفوفات ذات البعدين.

مصفوفات البعد الواحد

المصفوفة ذات البعد الواحد هي مجموعة من العناصر مرتبة بحيث يمكن الوصول إلى أى عنصر فيها باستخدام ترتيبه بالنسبة لأول عنصر في المصفوفة
وفي لغة C++ يأخذ أول عنصر الرقم صفر.
والشكل التالي يوضح مصفوفة ذات بعد واحد

```
A= [ 2  3   4   5   6]
```

وعناصر المصفوفة مرتبة بدءا من العنصر الأول والذي يأخذ الرقم صفر
ويكون العنصر الأول A[0] مساويا للقيمة 2.
وبالمثل يكون A[2] = 4 ، A[1] = 3 ، وهكذا...

مصفوفات ذات بعدين

المصفوفة ذات البعدين تحتوي على عناصر من نفس النوع، ولكنها مرتبة في صفوف و أعمدة .
وبالتالي تختلف طريقة الوصول للعناصر إذ يلزم لتحديد العنصر استخدام رقم الصف و رقم العمود و
الشكل التالي يوضح مصفوفة ذات بعدين

$$B=\begin{pmatrix} 12 & 23 & 15 \\ 25 & 35 & 89 \\ 90 & 80 & 16 \end{pmatrix}$$

وعناصر المصفوفة في هذه الحالة كما ذكرنا تحدد باستخدام رقمين رقم الصف ورقم العمود،
فالعنصر 12 يقع في العمود الأول والصف الأول أو بلغة الكمبيوتر
B[0][0]=12. لاحظ أن الترقيم في المصفوفة يبدأ بالرقم صفر دائما.

وبالمثل يمكن تحديد العناصر المختلفة ، ويذكر رقم الصف أولا ثم رقم العمود، والشكل التالي يوضح أمثلة لتحديد عناصر مختلفة من المصفوفة B

CODE

```
B[1][2] = 35

B[2][1] = 80

B[0][2] = 15

B[2][2] = 16
```

وسنتناول فيما يلي كيفية التعامل مع المصفوفات من خلال لغة C++ والإعلان عنها وتخصيص قيم للعناصر وطباعة العناصر وغيرها من اساليب معالجة المصفوفات.

مصفوفة البعد الواحد في لغة C++

المثال الموضح بالشكل التالي يوضح كيفية التعامل مع مصفوفة ذات بعد واحد بالاسم A

CODE

```
#include <iostream.h>
main()
{
int A[4];          // (1)

for(int I=0; I<4; I++)
{
cout<<" Please enter the value of the element A[" << I << "]";
cin >> A[I];
}

for(int I=0; I<4; I++)
{
cout<<" The value of the element A[" << I << "] is" << A[I];
}
}
```

ويقوم المثال بعد ذلك بعدة عمليات نتناولها بالتفصيل

السطر المشار إليه بالرقم(1) يعلن عن المصفوفة وعناصر المصفوفة من النوع العددي الصحيح (int) وعدد العناصر أربعة.

والإعلان عن المصفوفة كالإعلان عن المتغيرات العادية يذكر نوع المتغيرات أولا ثم اسم المصفوفة متبوعا بعدد العناصر بين قوسين مربعين.

والحلقة for الأولى تقوم بتعبئة المصفوفة بالبيانات التي يدخلها المستخدم واحدا بعد الآخر، ويلحظ أنه لا بد لنا من حلقة تكرارية لإدخال البيانات في المصفوفة.

أما الحلقة for الثانية فتقوم بعرض عناصر المصفوفة التي تم إدخالها عنصرا عنصرا.

المصفوفة ذات البعدين في لغة C++ :

في أحد الفصول الدراسية كانت نتائج ثلاثة طلاب كما هو موضح بالجدول التالي

	المادة الأولى	المادة الثانية	المادة الثالثة
الطالب الأول	68	52	70
الطالب الثاني	82	92	90
الطالب الثالث	85	85	83

والآن لو طلب منا برنامج يمكنه التعامل مع هذا الجزء من النتيجة، فإننا نحتاج بكل تأكيد لمصفوفة ذات بعدين والبرنامج التالي يوضح كيفية إنشاء مصفوفة ذات بعدين، وبعد ذلك يطلب من المستخدم إدخال البيانات الموضحة في الجدول ويقوم بتخزينها في عناصر المصفوفة المناظرة، وبعد ذلك يطبع البرنامج القيم المدخلة.

CODE

```
#include <iostream.h>
main()
{
float Degrees[3][3];    // Array declaration

// Array Element entry

for (int I=0; I<3; I++)
{
for(int J=0; J<3; J++)
{
            cout<<" Enter the result of subject " << I  << "for
student  "<< J;
            cin >> Degrees[I][J];
}
};
// Array elements display
for (int I=0; I<3; I++)
{
  for(int J=0; J<3; J++)
  {
cout<<" the result of subject " << I  << "for student  "<< J
<<"is";
cout<< Degrees[I][J];
   }
};
}
```

ويلاحظ استعمال حلقتين تكراريتين من النوع for لتخصيص البيانات للمصفوفة ولعرضها بعد التخصيص وكل من الحلقتين التكراريتين تتكونان من حلقة خارجية تقوم بزيادة عداد الأعمدة وحلقة الداخلية تقوم بزيادة عداد الصفوف.

وترمز الأعمدة هنا لرقم المادة بينما ترمز الصفوف لرقم الطالب.

الفصل السابع : الفصائل والكائنات (Classes & Objects)

سنتناول في هذا الفصل بشيء من التفصيل الفصائل والكائنات لنتعرف عن قرب على برمجة الكائنات الموجهة .

الفصيلة تتكون من بينات ودوال تتعامل مع هذه البيانات والشكل التالي (شكل 1) يوضح الصورة العامة للإعلان عن الفصيلة

CODE

```
class class_name{
private:
private data and functions
public :
public data and functions
}
```

والإعلان عن الفصيلة يتكون من:

أولا : الكلمة المحجوزة (class) يليها اسم الفصيلة (class_name) ويخضع اسم الفصيلة لقواعد عامة هي:

- ألا يكون اسم الفصيلة 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.

- 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.

-لا قيود على طول الاسم ، وتتيح هذه الميزة استخدام أسماء معبرة عن مضمونها، ومن الأفضل

دائما استخدام الاسم المعبر عن محتوى الفصيلة لتسهيل التعامل مع الفصائل.

-الحروف الكبيرة و الصغيرة ليست متكافئة في لغة C++ فمثلا اسم البيان(MY_CLASS) يختلف عن الاسم (my_class) وكلاهما يختلف عن الاسم (My_Class).

ثانيا: تحديد درجة الحماية ، ونبدأ عادة بدرجة الحماية الخاصة (private) وتلي هذه الكلمة البيانات و الدوال الخاصة بالفصيلة.

ثالثا: تحديد درجة حماية أخرى ،وفي هذا المثال استخدمنا الدرجة العامة (public) وسنتعرف على درجات الحماية بالتفصيل في وقت لاحق.

والمثال الموضح بالشكل التالي يوضح كيفية استخدام الفصيلة في برنامج.

CODE

```
01: #include <iostream.h>
02: class smallobj  // class name
03:{
04: private:
05:  int somedata; //class data
06: public:
07:  void setdata (int d); // member function to set data
08:   {somedata= d;}
09:  void showdata()  // mamber function to display data
10:   { cout << " \n Data is " << somedata;}
11:};

12: void main()
13:{
14: smallobj   s1,s2;    // define two objects of the class

15: s1.setdata(1096);  //call member function to set data
16: s2.setdata(200);

17: s1.showdata();
18: s2.showdata();
19:}
```

يبدأ البرنامج بالتوجيه في السطر الأول ، وفائدة هذا التوجيه إخبار المترجم بمكان الملف المحتوي على تعريفات الدوال الأساسية والتي سنستخدمها في البرنامج.

وتتابع السطور بعد ذلك كالآتي:

السطر 02 : تعريف فصيلة جديدة تحمل الاسم (smallobj) ويلاحظ التعليق المكتوب بعد العلامة

" //" ، وهذة الميزة لم تسمح بها لغة C .

السطرين 04 و 05: تعلنان عن بيان من النوع الصحيح.

السطر 06: يعلن درجة الحماية العامة، بمعنى أن ما سيأتي بعد ذلك سيكون عاما فيمكن للفصائل المشتقة أن تتعامل معه.

السطور من 07 إلى 10 فيهما تعريف الدالتين الوحيدتين في الفصيلة.
ويلاحظ في السطر 10 كيفية الطباعة على الشاشة وهو أسلوب جديد لم يكن مستعملا من قبل في لغة C. وسنتعرض للأساليب الجديدة في فصل مستقل.

وبداية من السطر 12 يبدأ البرنامج فعليا كالعادة بالدالة main() .

وفي السطر 14 نعرف كائنين من الفصيلة السابقة، ويلاحظ أن تعريف الكائنات يتم كتعريف المتغيرات العادية.

السطر 15 يستدعي الدالة (setdata()) للكائن الأول، وتجب ملاحظة طريقة الاستدعاء باستخدام المؤثر (.)، حيث يذكر اسم الكائن متبوعا بالمؤثر (.) ثم اسم الدالة المراد تنفيذها مع تخصيص قيم لمتغيرات الدالة.

السطرين 17 و 18 يتم فيهما استدعاء الدالة (showdata()) لكل من الكائنين (s1,s2).

درجة حماية أعضاء الفصيلة:

تعرضنا لعبارة " درجة حماية أعضاء الفصيلة " والآن نتناول هذه العبارة بشيء من التفصيل.

أن درجة الحماية هي تحديد مدى القدرة على التعامل مع أعضاء الفصيلة (البيانات و الدوال) والكلمات المستخدمة لتحديد درجة الحماية موضحة بالجدول التالي

الكلمة	المعنى
public	تعني أن البيانات التي تليها عامة ويمكن لأي دالة الوصول إليها و استعمالها.
protected	تفيد في حالة توريث الفصيلة، حيث يسمح للفصائل التي ورثت باستعمال أعضاء الفصيلة الأساسية
private	تعني أن البيانات خاصة بهذه الفصيلة ولا يمكن الوصول إليها إلا بواسطة دوال الفصيلة

دالة البناء :

ذكرنا سابقا أن دالة البناء ما هي إلا عضو من الفصيلة وتحمل نفس اسمها، وتنفذ هذه الدالة تلقائيا عند الإعلان عن كائن من الفصيلة.

ويمكننا أن نستفيد من هذه الدالة في تخصيص قيم لبعض بيانات الكائن عند الإعلان عنه.

و المثال التالي يوضح برنامجا قمنا فيه بالإعلان عن فصيلة ودالة البناء تقوم بعملية تخصيص القيم التي تأخذها لبيانات الفصيلة (a , b)

CODE

```
# include “iostream.h”
class MyClass
{
int a,b;
public:
MyClass (int i, int j)
{
a=i;
b=j;
}
void Show()
{
cout << a<< “              “<<b;
}
};
void main()
{
MyClass      C1(2,6);
C1.Show();
}
```

نلاحظ في هذا المثال أن الإعلان عن الكائن من الفصيلة (MyClass) لم يتم بالطريقة المعتادة حيث قمنا باستخدام قوسين بعد اسم الكائن وبينهما قيمتين عدديتين.

والإعلان هنا قام باستدعاء دالة البناء الخاصة بالفصيلة والتي بدورها تأخذ القيم المعطاة وتخصصها للبيانين (a , b) الموجودين بالفصيلة.

وعند استدعاء دالة (Show()) والتي تعرض قيم المتغيرين (a , b) نجد أنها تعرض القيم التي تم تخصيصها عند الإعلان عن الكائن (C1)

مصفوفة الكائنات:

المصفوفة هي مجموعة من العناصر من نفس النوع، وجرت العادة أن نعرف مصفوفات من أنواع المتغيرات المتاحة في اللغة.

و مع لغة C++ يتمكن المبرمج من الإعلان عن مصفوفة من الكائنات أيضا بنفس الكيفية التي يستخدمها للإعلان عن مصفوفة من المتغيرات العادية.

والمثال التالي يوضح كيفية الإعلان عن مصفوفة الكائنات

CODE

```
#include <iostream.h>
class Date
{ public:
int day,month,year;
 set_date(int d, int m, int y)
{day=d; month=m; yaer=y;}
};
main()
{
Date       date_array[3];
date_array[0].set_date(2,3,1990);
}
```

ونلاحظ من هذا المثال أننا أعلنا عن مصفوفة كائنات من نوع (Date) وهي الفصيلة التي أعلنا عنها قبل الدالة الرئيسية مباشرة.

ونتعامل مع عناصر مصفوفة الكائنات بطريقة مماثلة لتعاملنا مع عناصر المصفوفات الأخرى، والسطر الثاني يوضح مثلا كيفية استدعاء الدالة (set_date) للعنصر الأول من عناصر المصفوفة.

استعمال المؤشرات مع الكائنات

المؤشرات (pointers) في لغات البرمجة لها أهميتها القصوي (والتي قد لا يدركها المبتدئ) ونتيجة لهذة الأهمية ظهرت الحاجة لاستخدام المؤشرات مع الكائنات.

وتدعم لغة C++ استخدام المؤشرات مع الكائنات، ويتم الإعلان عن مؤشر لكائن ما وللإعلان عن مؤشر لكائن من الفصيلة (Date) الموضحة في المثال السابق نستخدم العبارة (Date *dptr;) كما بالمثال التالي

CODE

```
#include <iostream.h>
class Date
{ public:
int day,month,year;
 set_date(int d, int m, int y)
{day=d; month=m; yaer=y;}
};
main()
{
Date *dptr;
Date   date;
Dptr  -> day=3;
Date.day=4;
}
```

ويلاحظ أنه في هذا المثال قد تم الإعلان عن كائن (date) ومؤشر إلى كائن (dptr) ، ومعاملة كل منهما تختلف عن الآخر حيث نستخدم المؤثر (->) مع المؤشر للكائن للوصول إلى البيان (day) فيه بينما استخدمنا المؤثر (.) مع الكائن (date).

ويجب توخي الحرص دائما عند التعامل مع المؤشرات لتلافي الأخطاء التي يمكن أن تحدث.

تم بحمد الله