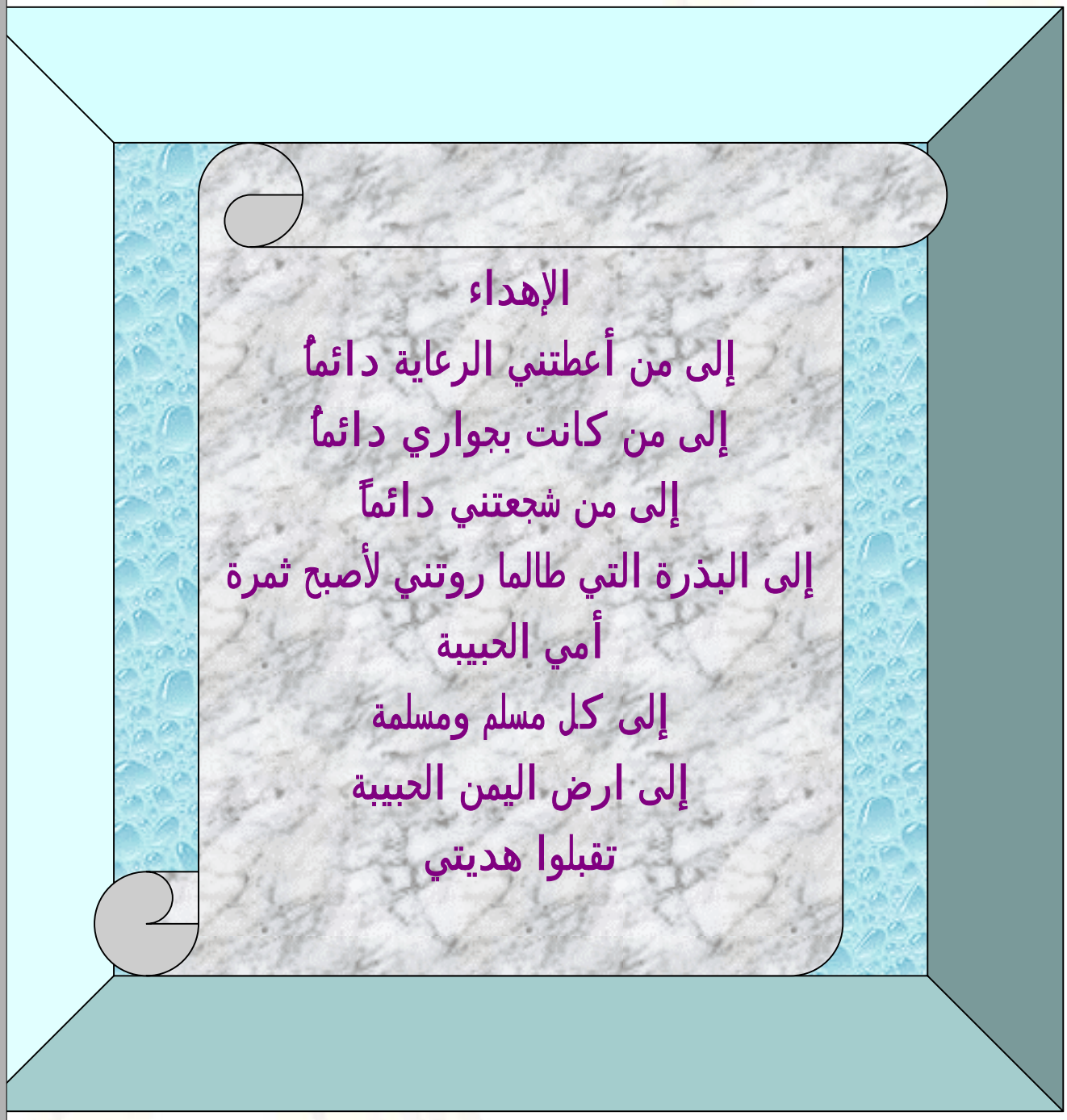
aldopaee@hotmail.com

بسم الله الرحمن الرحيم

الحمد لله رب العالمين الواحد الأحد الفرد الصمد الذي لم يلد ولم يولد والذي علم بالقلم علم الإنسان ما لم يعلم ونصلي ونسلم على المبعوث رحمة للعالمين محمد ابن عبد الله الصادق الآمين وعلى صحابته الأخيار ومن بعدة آلي يوم الذين أما بعد

تعد لغة c++ من اكتر اللغات انتشاراً واستخداماً كونها مناسبة ومفضلة في كثير من التطبيقات المهمة إذ نستطيع أن نقول أنها وسط تطوير البرمجيات و إنتاجها وهي ما تزال تحتل موقع الصدارة مقارنة مع غيرها من لغات البرمجة إذ تعتبر هي لغة إلام ونظرا لهذه الأهمية وعدم وجود مراجع عربية كافية بالغرض فقد قمت بتأليف هذا الكتاب الذي قد حاولت تلخيص الفكرة لإخواني الطلاب والمحبين لهذه اللغة من خلال دراسي الجامعية ومن محبتي لهذه اللغة راجياً من الله ان يستفيدوا إخواني المسلمين في مشرق الأرض ومغربة وإعطائي دعوة صالحة في ظهر الغيب من إخواني الذين قرئوا هذا الكتاب راجياً من الله إن يكون هذا الكتاب يفي بالغرض وان ينال إعجابكم ونأسف عن أي خطأ او تقصير فالكمال لله عز وجل وصلى الله على محمد وعلى آلة وصحبة أجمعين وأخر دعوانا إن الحمد لله رب العالمين ولا عدوان إلى على الظالمين والمشركين من اليوم إلى يوم الذين .

aldopaee@hotmail.com



aldopaee@hotmail.com

لا يحق لأحد بيع هذا الكتاب فهو مجاني

ملاحظه

- يوجد بعض البرامج مكتوبة بلغة السي وذلك لتذكير بها من النسيان.
- الخطوة الوحيدة لتطوير هذا الكتاب أبداء إي ملاحظة أو استفسار نحو هذا الكتاب أو اكتشاف آي خطأ الرجاء أبلاغيي بذالك على البريد الالكتروني
- إلى كل طالب في قسم الحاسوب أي سؤال عجز عن حلة يرجى أرسالة للبريد الالكتروني وانأ سأوافيه الحل إنشاء الله وسأكون تحت الخدمة دائماً..... وشكراً

السيرة الذاتية

الاسم: عمار محمد عيسى الدبعي

الجنسية: يمني العمر: ٢١ سنة محل الإقامة: الجمهورية اليمن

aldopaee@hotmail.com

لغة C++

المحتويت

المؤثرات العلاقية و المنطقية

مؤثر العنوان (Address of Operator)

الفصل الثالث : اتخاذ القرارات

العبارة الشرطية البسيطة (if statement)

العبارة الشرطية الكاملة (if ……. else statement)

العبارة الشرطية المتدرجة (if-else- if Ladder)

الاختيار متعدد البدائل (switch statement)

مسائل على الباب

الفصل الرابع : الحلقات التكرارية

الحلقة for (for loop)

الحلقة while (while loop)

الحلقة التكرارية do-while

مسائل على الباب

الفصل الخامس: الدوال و الماكرو (Function & Macro)

أنواع الدوال

معاملات الدوال

معاملات الدالة الرئيسية (main function arguments)

المؤشرات

السجلات

الماكرو

مسائل على الباب

الفصل السادس : المصفوفات

aldopaee@hotmail.com

مصفوفات البعد الواحد

مصفوفات ذات بعدين

مسائل على الباب

الفصل السابع : الفصائل والكائنات (Classes & Objects)

درجة حماية أعضاء الفصيلة

دالة البناء

مصفوفة الكائنات

استعمال المؤشرات مع الكائنات

مسائل على الباب

aldopaee@hotmail.com

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.

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.

برمجة الكائنات الموجهة ولغة C++

تعتبر برمجة الكائنات الموجهة من أحدث أساليب البرمجة، وليست بالأسلوب الوحيد حيث سبقتها أسلوب عرف بالبرمجة المنهجية (Procedural programming) والتي تعتمد على الدوال كوحدات بناء للبرنامج .

إذ يتكون البرنامج من مجموعة من الدوال التي تؤدي كل منها وظيفة محددة، وتقوم الدالة الرئيسية باستدعاء تلك الدوال وتنظيم العمل بينها.

أما أسلوب برمجة الكائنات الموجهة فيعتمد الفصيلة (class) كوحدة بناء البرنامج، وتتكون الفصيلة من مجموعة من البيانات والدوال التي تعمل على هذه البيانات.

وسنتعرف في هذا الفصل على عناصر ومزايا برمجة الكائنات الموجهة في هذا الفصل.بالإضافة إلى بعض مزايا لغة C++والتي تعتمد أسلوب برمجة الكائنات الموجهة في بناء البرامج

.

عناصر برمجة الكائنات الموجهة :

الفصائل (classes):

الفصيلة كما ذكرنا ما هي إلا بناء يتكون من بعض البيانات بالإضافة إلى دوال تتعامل مع هذه البيانات،

والفصيلة هي تكوين يقترب كثيرا من الواقع ، إذ أننا نجد في الحياة العملية الكثير من

الأشياء والتي يمكن اعتبارها فصائلا.

وكمثال على ذلك يمكننا اعتبار الكتاب فصيلة، وبيانات فصيلة الكتاب عديدة مثل: اسم المؤلف، موضوع الكتاب، اسم الكتاب.....

أما دوال فصيلة الكتاب فهي مثلا قراءته، تأليفه، طباعته .....

وكما تصورنا الكتاب كفصيلة يمكننا أن نجد فصيلة لكل الأشياء الموجودة في الحياة الواقعية.

الكائنات (Objects):

الكائن هو صورة من الفصيلة يتعامل معها المبرمج، فكما نعرف متغيرات من النوع الصحيح مثلا يمكننا باستخدام أسلوب برمجة الكائنات الموجهة أن نعرف كائنات من فصائل موجودة لدينا لنتعامل معها.

ومع التقدم في البرمجة باستخدام برمجة الكائنات الموجهة ستصبح هذه المصطلحات أكثر وضوحا وأقرب للذهن.

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

```
#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.

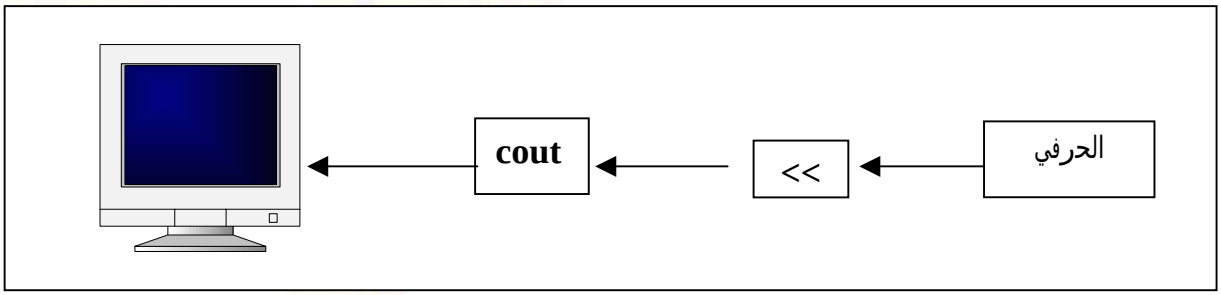
أما الكلمة المحجوزة (**void**) والتي تسبق اسم الدالة فتوضح أن الدالة ليس لها قيمة ترجع بها ، بخلاف بعض الدوال التي نخصص لها نوعا من البيانات بحيث ترجع قيمة من نوع هذا البيان.

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

```
// this is the first method        } 1

// this is the
// first method                    } 2

/* this is the second method */    } 3

/* this is
the second
method*/                           } 4
```

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++

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:

١- 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.

٢- يمكن أن يحتوي الاسم على أى حرف من الحروف الأبجدية (A-Z) سواء صغيرة كانت أم كبيرة، وأي رقم من الأرقام (٠-٩) كما يمكن أن تحتوي على علامة الشرطة السفلى (_) ولكن لا يجوز أن يبدأ الاسم برقم.

٣- 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.

٤- 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.

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 (b) 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

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(٢-١٣) برنامجا قمنا فية بالإعلان عن متغيرين و إختزان بيانين عدديين في كل منهما ، ثم نطبع محتويات هذين المتغيرين على الشاشة.

عبارة التخصيص (Assignment statment) :

إن العبارة

a=1000;

يمكن قرائتها على النحو التالي:

" خصص العدد ١٠٠٠ 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:

١- تخصيص قيمة عددية للمتغير "a"

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

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

ومن الملاحظ في هذا البرنامج أنه قد تم إعلان المتغيرين "b" ، "c" 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.

المتغيرات العددية الصحيحة (integer variables):

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

```
#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 " ٢٠ " في المتغير الأول، والعبارة المستخدمة لتخصيص قيمة المتغير الثاني ليست مباشرة، إذ يقوم المعالج بأداء عملية حسابية قبل تخصيص القيمة، حيث يجمع القيمة " ١٠ " 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

```
#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:

```
#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.

عبارات الدخل والخرج في C:

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**"

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:

١- ضرورة استخدام توصيف للفورمات بنفس الأسلوب المتبع مع الدالة **printf** وفي المثال السابق قد استخدمنا التوصيف " **%f** " الذي يناظر المتغير الحقيقي "**x**" أو " **y**".

٢- 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

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.

رسالة لتنبية مستخدم البرنامج :

من عيوب الدالة **cin** او **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** او **cout** 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.

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**):

```
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,b)**) 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**.

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 " ٠ "

إلى " ١٩" كالتالي:

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 ١١ 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.

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 ١٩ 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:

```
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 :

١- علامة السطر الجديد (\n).

٢- الرمز الصفري (NULL).

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

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

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.

عبارات الإدخال باستخدام **cin**

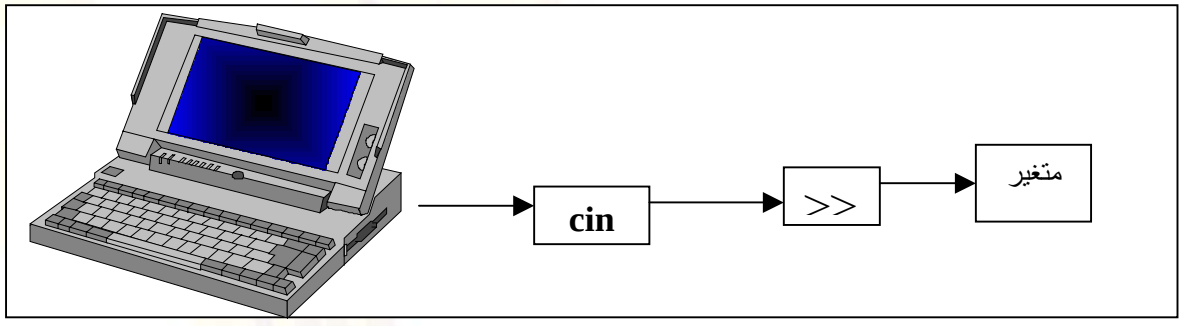
تعرفنا على العبارة المستخدمة في الإخراج ونتناول الآن العبارة التي تستخدم للإدخال.

والمثال التالي يوضح العبارة قيد الاستخدام

```
#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.

والشكل التالي يوضح استخدام عبارة الإدخال



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

المؤثرات (Operators):

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

وتنقسم المؤثرات إلى:

١- المؤثرات الحسابية (Arithmetic Operators)

٢- المؤثرات العلاقية. (Relational operators)

٣- المؤثرات المنطقية. (Logical operators)

المؤثرات الحسابية:

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

المؤثر	المعني
+	الجمع
-	الطرح
*	الضرب
/	القسمة

وبالإضافة لهذه المؤثرات توجد مؤثرات خاصة وهي الموضحة بالجدول التالي

المؤثر	المعني
%	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

الاستخدام	10%3
الناتج	1

مؤثر النقصان و مؤثر الزيادة:

يعتبر هذان المؤثران من أهم ملامح اللغة ، ويمكنان المبرمج من كتابة عبرات البرنامج باختصار شديد.

والأمثلة التالية توضح استخدام هذين المؤثرين

مثال : بدلا من استخدام العبارة الآتية لزيادة قيمة المتغير (a) بمقدار (1)

```
a = a + 1;
```

يمكننا استخدام مؤثر الزيادة مباشرة كالآتي

```
a++;
```

والعبارة الأخيرة تؤدي أيضا إلى زيادة المتغير ولكن يلاحظ أن العبارة أصبحت أكثر اختصارا.

مثال بدلا من العبارة التالية والتي تستخدم لإنقاص المتغير (a) بمقدار ١

```
a = a –1;
```

يمكننا الآن استخدام العبارة المختصرة

```
a--;
```

ونلاحظ أيضا أن استخدام مؤثر النقصان يؤدي إلى الاختصار.

ونلاحظ ايضاً انه يوجد فرق بين a++ و ++a في الحالة الاولى تزيد قيمة aبواحد بعد طباعتها و الحالة الثانية عكس

وهذا المثال يوضح ذلك

```
#include <iostream.h>
int main()
{
int myAge = 39;
int yourAge = 39;
cout << "I am: " << myAge << " years
old.\n";
cout << "You are: " << yourAge << " years
old\n";
myAge++;
++yourAge;
cout << "One year passes...\n";
cout << "I am: " << myAge << " years
old.\n";
cout << "You are: " << yourAge << " years
old\n";
cout << "Another year passes\n";
cout << "I am: " << myAge++ << " years
old.\n";

cout << "You are: " << ++yourAge << " years
old\n";
cout << "Let's print it again.\n";
cout << "I am: " << myAge << " years
old.\n";
cout << "You are: " << yourAge << " years
old\n";
return 0;
}
```

```
Output: I am      39 years old
You are   39 years old
One year passes
I am      40 years old
You are   40 years old
Another year passes
I am      40 years old
You are   41 years old
Let's print it again
I am      41 years old
You are   41 years old
```

المؤثرات العلاقية والمنطقية:

تستخدم المؤثرات العلاقية لبناء التعبيرات العلاقية المستخدمة في المقارنات مثل

- **x أكبر من 5 وتكتب للكمبيوتر بالصورة x>5.**
- **y أصغر من أو تساوي ٤٠ وتكتب y <= 40.**
- **x تساوي 15 وتكتب x = = 15.**

والجدول الآتي يوضح المؤثرات العلاقية المختلفة المستخدمة في لغة C++.

المؤثر	المعني
>	أكبر من
>=	أكبر من أو يساوي
<	أصغر من
<=	أصغر من أو يساوي
= =	يساوي
!=	لا يساوي

ومن الملاحظات الهامة أن مؤثر التساوي (= =)يختلف عن مؤثر التخصيص (=) في لغة C ++، وذلك على العكس من اللغات الأخرى والتي تستخدم المؤثر (=) للتساوي.

والمؤثرات المنطقية هي التي تستخدم لتحديد العلاقات المنطقية في العبارات المستخدمة، فمثلا لو أردنا تنفيذ شرط معين عندما يتحقق تساوي أحد المتغيرات وليكن (x) بالمقدار (4) و عدم تساوي متغير آخر وليكن (y) بالقيمة (9) فإننا نلجأ الى استخدام المؤثرات المنطقية (مع استخدام المؤثرات العلاقية طبعا) وتكون العبارة كما يلي

```
if ((x = = 4) && (y != 9))
```

والمؤثر (&&) هو مؤثر منطقي بمعنى (و) أي عند تحقق الشرط الأول والشرط الثاني معا.

والجدول التالي يوضح المؤثرات المنطقية المستخدمة في لغة C++

المؤثر	المعني
&&	المؤثر (و) يستخدم عند لضمان تحقق الشروط كلها معا
||	المؤثر (أو) يستخدم لضمان تحقق أحد الشروط على الأقل
!	مؤثر النفي يستخدم للتأكد من عدم تحقق شرط (أو مجموعة شروط)

مؤثر العنوان (Address of Operator):

لنلق نظرة على البرنامج التالي

```
#include <iostream.h>
void main()
```

```
{
      int a =11;

      cout << &a<<'\n';
      cout << a;
}
```

عند تنفيذ هذا البرنامج نحصل على الناتج التالي

0x8f4ffff4
11

و القيمة الثانية تمثل قيمة المتغير (**a**) ولكن القيمة الأولى تمثل عنوان المكان المخزن فيه قيمة المتغير في الذاكرة .

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;
```

العبارة الشرطية البسيطة

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

```
#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 ٥٠)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 ٥٠ ؟؟

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

```
#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;
```

العبارة الشرطية الكاملة

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

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

```
        statement 2;
                .
            .
        statement m;
}
```

العبارة الشرطية الكاملة باستخدام بلوكات للنتائج

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.

```
#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

```
if  ( condition –1)
      statement –1;
else if ( condition-2)
      statement-2;
else if( condition-3)
      statement-3;
……….
else
      statement-n;
```

العبارة الشرطية المتدرجة

الاختيار متعدد البدائل (**switch statement**)

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

```
switch (variable)
{
      case value1;
            statement 1;
            break;
      case value2;
            statement 2;
            break;
      case value 3;
```

```
        statement 3;
        break;
    ..............
    default:
        statement;
}
```

الاختيار متعدد البدائل

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.

مسائل على الباب

*برنامج لمعرفة العدد المدخل هل هو زوجي ام فردي

```
#include<iostream.h>
Main(){int a;
Cin>>a;
If (a%2=0)
Cout<<"zojee";
Else
Cout<<"frdee";
}
```

*برنامج لاستخراج الاعداد الفردية من ١-١٠ ومجموعهم

```
#include<iostream.h>
Main(){int a=1,b=0;
Q:If (a%2!=0){cout<<a;b+=a;}
a++;
if (a<=10)goto q;
cout<<"\n"<<b;
}
```

*برنامج لمعرفة العدد هل هو اولي ام لا من بين مجموعة من الاعداد ؟

*برنامج يجمع اخر عددين مدخلين من لوحة المفاتيح ؟

*برنامج يجمع الاعداد السالبة والاعداد الموجبة من بين عشرة اعداد ؟

*برنامج يطبع تقدير الطالب للدرجة المدخلة ؟

+++++>هذة التمارين على القارى.

aldopaee@hotmail.com

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

١- الحلقة for (for loop).

٢- الحلقة while (while loop).

٣- الحلقة do…. while (do-while loop).

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)
```

شكل ٦-١ الصورة العامة للحلقة for

و هذه العناصر هي:

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

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

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

والشكل التالي (شكل ٦-٢) يوضح برنامجا قمنا فيه باستخدام الحلقة for :

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

```
{
    int counter ;
    for ( counter=1;counter<=20;counter++)
        cout<<counter;
}
```

برنامج يوضح استخدام الحلقة for

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 ٢٠.

أما العبارة الثالثة فهي تحدد الخطوة، وفي هذا البرنامج يزاد العداد بمقدار ١ كل مرة تنفذ فيها الحلقة.

والبرنامج السابق ينتج عنه طباعة الأرقام من ١ إلى ٢٠.

ملاحظات:

١- 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.

٢- 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

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

الصورة العامة للحلقة while

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 ١ إلى ٢٠

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

مثال لاستخدام الحلقة while

من المثال السابق يمكننا استخلاص النتائج التالية عن الحلقة **while**:

١- تخصيص القيمة الابتدائية للعداد تتم خارج الحلقة **while**.

٢- 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**

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

الصورة العامة للحلقة do-while

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.

مسائل على الباب

*برنامج يطبع اكبر قيمة واصغر قيمة

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

main()
{int i,a,s,d,f;
```

```
a=s=0;
clrscr();
cin>>a;
d=f=a;
for ( i=2; i<=6; i++){
if(d<a){
d=a;}
if(f>a){
f=a;}
cin>>a;
}
cout<<d<<"  "<<f;
getch();
}
```

* برنامج لايجاد مضروب العدد

```
#include<iostream.h>
#include<conio.h>
main(){
clrscr();
int a,s;long d;d=1;
cin>>a;
for(s=1;s<=a;s++){
d*=a;cout<<d<<"\n";
}
getch();
}
```

* برنامج لإيجاد عوامل العدد

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
main(){
clrscr();
int a,s,d;
```

```
cin>>a;
for(s=1;s<=a;s++){
d=floor(a/s)*s;
if(a!=d){goto h;}
cout<<s<<"\n";
h:
}
getch();

}
```

*برنامج لإيجاد القاسم المشترك ؟

* // // // المضاعف ؟

* // يطبع الارقام التي تقبل القسمة على ٣ و٥ ؟

*// يطبع ويجمع السلسلة الثالية ١,١,٢,٣,٥,٨,١٣,٢١,٣٣ ؟

* // يعرف هل العدد المدخل متساوي المراتب ١١١ يطبع نعم /١٠١ يطبع لا؟

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:

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

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

٣- 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 ١)يوضح استخدام دالة عرفها المبرمج

```
#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<<'*';
}
```

مثال (١)

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 :

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

٢- 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:

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

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

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

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

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

والمثال التالي (@ مثال ٢) يوضح دالة أعداد حقيقية وكيفية استخدامها.

```
#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 );
}
```

مثال ٢

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 ٢

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 ١

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

مثال٣ يوضح برنامجا فيه الدالة الرئيسية تم استدعائها بمعاملاتها

```
#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];
}
```

مثال ٣

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

عنصر المصفوفة	القيمة المختزنة
argv[0]	Prog1
argv[1]	First
argv[2]	Second

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
```

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++**.

```
#include "stdio.h"

int main ()
{
int *px;
int a;

01: px = &a; /* 'px' will point on 'a'
*/

02: *px = 10; /* Changes on 'px' will
effect on 'a' */

03: printf("px = %d \n\n", *px);
04: printf("a = %d \n\n", a); /* 'a' and
'px' will have the same value that is 10
*/


05: a = 20; /* Changes on 'a' will
effect on 'px' */

06: printf("px = %d \n\n", *px);
07: printf("a = %d \n\n", a); /* 'a' and
```

```
'px' will have the same value that is 20
*/

return 0;
}
```

لاحظ عزيزي أنه في السطر ١ جعلنا px يؤشر على a أي أن px سيحتوي على عنوان a .

في السطر ٢ جلعا غيرنا px و جعلناها تساوي ١٠ و لكن في الحقيقة نحن لم نغير px و لكننا قمنا بتغيير العنوان الذي يوجد بداخل px و المعنى الحرفي لهذا السطر هو:

" إذهب إلى العنوان الموجود داخل px و إجعل هذا العنوان يحتوي على القيمة ١٠ " و نحن نعلم أن هذا العنوان هو عنوان a (من السطر ١) إذا سوف يغيير هذا السطر المتغير a و سنلاحظ هذا التغيير في السطرين ٣ و ٤ في السطر ٣ قلنا للدالة printf إطبعي محتوى العنوان الموجود داخل px و في السطر ٤ قلنا لها إطبعي قيمة المتغير a و سنلاحظ أن هذين القيمتين هما ١٠ إذاً سيطبع على الشاشة القيمة ١٠.

أما في السطر ٥ أسندنا للمتغير a القيمة ٢٠.

و في السطر ٦ قلنا للدالة printf إطبعي القيمة الموجوده في العنوان الموجود داخل px و نحن نعرف أن العنوان الموجود داخل px هو عنوان a (من السطر ١)، ثم في السطر ٧ طبعنا قيمة a و سيبطع على الشاشة القيمة ٢٠ مرتين، مره من المؤشر px و مرة أخرى من المتغير a .

طبعاً هذا المثال سهل و لكن الذي يفهم هذا المثال سيفهم ٩٠٪ من موضوع المؤشرات، و هذا الموضوع ليس صعب كما يعتقده الكثيرون :).

الآن قد تعرفنا على كيفية جعل المؤشر يؤشر على متغير و على كيفية حجز عنوان في الذاكرة للمؤشر و كيفية إسناد القيم للمؤشر و لكن بقي علينا أن نتعرف على كيفية إسناد المؤشرات لبعضها البعض و على كيفية جعل المؤشر يؤشر على مؤشر آخر.

aldopaee@hotmail.com

ففرضاً لو كان لدينا التعريق التالي:

```
int *p1, *p2;
int a = 5;
```

و جعلنا p1 يؤشر على a بكتابة هذا السطر:

```
p1 = &a;
```

(أي جعلنا p1 يحوي عنوان a)

الآن نريد أن نجعل p2 يؤشر على p1 كيف سيكون ذلك ؟!؟

الحل ببساطة نجعل في p2 عنوان p1 الذي هو في الأصل عنوان a، إذاً يكون السطر الذي يجعل p2 يؤشر على P1 هو:

```
p2 = p1;
```

أي عنوان p2 يساوي عنوان p1 .

الآن لو طبعاً قيم:

```
*p1, *p2, a
```

المؤشرات و المصفوفات:

المؤشرات و المصفوفات في لغة سي و سي++ متقاربين جداً، لذلك تجدون في أغلب الكتب أنهما يكونان في فصل واحد، و في الحقيقة كمبايلر السي و السي++ يجعل إسم المصفوفة مؤشر إلى أول عنصر فيها، لذلك الجملتين التاليتين متماثله تماماً:

```
r = x[1];  تساوي  r = *(x+1);
```

و لأن اسم المصفوفة مؤشر إلى أول عنصر بداخالها فإنه من الممكن إسناد مؤشر إلى المصفوفة عن طريق هذه الجملة:

```
ptr = x;
```

أو:

```
ptr = &x[0];
```

هاتين الطريقتين لهما نفس الفاعليه و لكن الفرق هو انه في الأول عاملنا المصفوفه X كمؤشر إلى أول عنصر و لكن في الطريقة الثانية عاملنا المصفوفه X كمصفوفه أي أن X[0] هو متغير عادي مثل أي متغير.

و لنأخذ هذا المثال:

```
#include "stdio.h"

int main ()
{
int x[] = {0, 1, 2, 3, 4, 5, 6, 7, 8,
9};
int i;

printf("Arrays As Pointers:\n");
for( i = 0 ; i < 10 ; i++)
{
printf("*(x + %d) = %d\n", i, *(x + i));
}

printf("\n\nArrays As Arrays:\n");
for( i = 0 ; i < 10 ; i++)
{
printf("x[%d] = %d\n", i, x[i]);
}
```

```
printf("\n\n\n");

return 0;
}
```

في هذا المثال يتبين لنا كيف إستخدمنا المصفوفات كمؤشرات تاره و كمصفوفات تارة أخرى و أن لهما نفس النتائج.

مصفوفة المؤشرات:

المصفوفات من الممكن أن تكون مؤشر إلى نوع ما بحيث كل عنصر من المصفوفة يصبح مؤشر إلى نوع معين.

من أكثر الأمثله لهذا التركيب هو مصوفة مؤشرات إلى char أي كل عنصر عبارة عن string أي كالتالي:

```
char *names[4];
```

أي أن المصوفه name تحتوي على أربع عناصر كل منها هو string .

لنأخذ مثال بارسم على ذلك:

لنفرض انه لدينا هذا التعريف:

```
char *name[4] = { "Talal", "Abdullah",
"Thamer", "Mohammad" };

name[0] ===> "Talal"

name[1] ===> "Abdullah"

name[2] ===> "Thamer"

name[3] ===> "Mohammad"
```

سيكون تمثيل العناصر كما هو مبين أعلاه.

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.

كيفية تعريف السجل في لغة سي:

أولاً لابد ان نعلم ان كلمة struct كلمة محجوزه في لغة سي و سي++ ، و نستطيع تعريف السجل كالتالي:

```
struct (إسم السجل
{
أعضاء السجل
};
```

– طبعاً هذه الطريقة هي أحدا الطرق التي تستطيع تعريف السجل بها.

فلو اردنا ان نعرف سجل إسمه data و يحتوي على إسم من نوع char* و العمر من نوع int

إذا سيكون التعريف كالتالي:

```
struct data
{
   char namr[30];
   int age;
```

```
};
```

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)

```
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)

```
#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).

مصفوفة السجلات :

لقد علمنا ان السجل نوع كأي نوع من انواع البيانات، لذلك من الممكن ان يكون السجل مصفوفة ايضاً و الطريقة سهله جداً كالتالي:

```
structure_name var[NUM] ;
```

فلو اخذنا السجل :

```
typedef struct
{
        char name[30];
        int age;
}data;
```

و اردنا ان نعرف مصفوفة من نوع data يسكون كالتالي:

```
data student[100] ;
```

طبعاً العدد ١٠٠ إختياري .

و نحن في السابق أخذنا نوع student من السجل data و سيكون سجل واحد و لكن هنا سيتضح اهمية السجلات فعندما عرفنا student كمصفوفة من نوع data أصبح كأنه لدينا

١٠٠ طالب و كل عنصر في المصفوفة عباره عن سجل بحد ذاته.

و للوصول إلى محتويات السجل نتبع الطريقة التاليه :

student[indix].name & student[indix].age ...

و غالباً تستخدم مصفوفة السجلات إذا كان العدد محدداً أما إذا كان العدد غير محدد نستخدم طريقة من طريق الـ Data Structure منها اللنك لست درسنا القادم.

السجلات و المؤشرات :

و نعيد و نكرر انه بعد تعريف السجل يصبح نوع كأي نوع آخر من انواع البيانات، إذا يمكن للسجل ان يكون مؤشر (Pointer) و العمليه كالتالي:

```
typedef struct
{
	char name[30];
	int age;
}data;
```

و سنعرف مؤشر للسجل كالتاالي :

```
data *s ;
```

فالنأخذ البرنامج التالي للتوضيح :

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

```
typedef struct
{
	char name[30];
	int age;
}data;

int main()
{
	data *s, std;

	s = &std;	// Assign std to s

	strcpy(std.name,"Talal");
	std.age = 20;

	printf("std.name = %s, std.age =
%d\n\n",std.name, std.age);
	printf("s->name = %s, s->age = %d\n\n",s-
>name, s->age);

return 0;
}
```

طبعاً نلاحظ الآن ظهور العلامة '->' بدل من النقطة عند إستخدام المتغير S ؟! لماذا ؟

الجواب : لأنه مؤشر لسجل و مؤشر السجل يستعمل في لغة السي و السي++ هذه العلامة بدلاً من العلامة '.' ، و هذا من الاختلافات التي تميز لغة السي و السي++ عن باقي اللغات مثل الجافا و الدلفي فهي لا تفرق إذا كان مؤشر أو لا .

إذا قاعدة في لغة سي و سي++ هي إنه عند إستخدام مؤشر لسجل نستخدم -> بدلاً من '.'

طبعاً هناك طريقة أخرى و هي هكذا :

s->name بدل (*s).name

طبعاً العلامة '->' أسهل :).

- **السجلات و الدوال :**

عند إستخدام السجلات مع الدوال إما أن يكون السجل مرسل للدالة أو إما ان يكون معاد من الدالة و إما ان يكون مستخدم في ضمن الدالة .

الحالة الأخيره معروفة و عملنا عليها في السابق داخل الدالة **main** و الـ **main** دالة اصلاً.

أما الحالتين الأولى و الثانيه فسنتطرق لها الآن.

- أولاً السجل معامل من معاملات الدالة :

أي أن نرسل السجل للدالة و الدالة تقوم بالعمليات على هذا السجل مثلاً: طباعة، معالجة، ... إلخ

و لنأخذ هذا المثال و نشرحة بعد قرائة المثال جيداً:

```
//-------------------------------------------------------

#include<stdio.h>
#include <conio.h>
//-------------------------------------------------------
typedef struct
{
        char name[30];
        int age;
```

```
}data;
//-------------------------------------------------------
void display(data r);
//-------------------------------------------------------
main()
{
	data std;

	strcpy(std.name,"Talal");
	std.age = 20;

	display(std);
}

//-------------------------------------------------------
void display(data r)
{
	printf("(r.name) = %s,\n(r.age) =
%d\n\n",r.name, r.age);
}

//-------------------------------------------------------
```

و في هذا المثال لقد كتبنا رأس الدالة كالتالي:

```
void display(data r) ;
```

أي أنه يوجد دالة إسمها display تستقبل السجل r من نوع data ولا تقوم بإرجاع شئ.

و عند إستدعاينا الدالة و بعد إعطائها القيم كالتالي:

```
display( std ) ;
```

ارسلنا لها السجل كاملاً لتسقبله و تطبعه في جسم الدالة display .

و لنأخذ مثالاً آخر لإعطاء قيم السجل في الدالة و طبعاتها في الـ main :

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

typedef struct
{
        char name[30];
        int age;
}data;

void assign(data *r);

main()
{
        data std;

        assign(&std);
        printf("std.name = %s,\nstd.age =
%d\n\n",std.name, std.age);
}

void assign(data *r)
{
        strcpy(r->name,"Talal");
```

```
        r->age = 20;
}
```

و في هذا المثال كتبنا رأس الدالة (التعريف) هكذا :

```
void assign(data *r) ;
```

و جعلنا r كمؤشر لأن قيمة r ستتغير (نحن نريد ذلك) لإعطائها القيم.

و قمنا بإرسال السجل كالتالي :

```
assign( &std ) ;
```

لأن الدالة assign تستقبل مؤشر للسجل لذلك نرسل لها عنوان السجل و ليس السجل نفسه.

و داخل الدالة assign إستخدمنا name<-r و age<-r لأن r في الدالة مؤشر (و مع المؤشرات نستخدم -> بدلاً من '.') .

<u>- ثانياً إرجاع سجل من الدالة :</u>

أي أن الدالة تقوم بإرجاع السجل عند الانتهاء من عملها و نستطيع تغيير البرنامج السابق ليرجع السجل بدلاً من إرسال السجل كعنوان و إستقباله كمؤشر.

سيتغير البرنامج ليصبح هكذا :

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

typedef struct
```

```
{
        char name[30];
        int age;
}data;

data assign(void);

main()
{
        data std;

        std = assign();
        printf("std.name = %s,\nstd.age =
%d\n\n",std.name, std.age);
}

data assign(void)
{
        data r;
        strcpy(r.name,"Talal");
        r.age = 20;
    return r;
}
```

و هنا عرفنا الدالة كالتالي :

```
data assign(void) ;
```

أي أن الدالة assign لا تستقبل شئ و القيمة المرجعة من الدالة هي عباره عن سجل من نوع data .

و قمنا بإستدعا الدالة هكذا :

```
std = assign() ;
```

أي أن القيمة المرجعة من الدالة ستوضع قيمتها في السجل std .

و في جسم الدالة assign عرفنا المتغير r من نوع سجل data و أعطينا لها قيم و قمنا بإرجاع هذا السجل من الدالة عن طريق الامر

```
return r ;
```

• إسناد السجلات :

نستطيع ان نسند سجلين لبعضهما البعض لكن شريطة أن يكونا من نفس النوع .

فلو أنشئنا السجل التالي :

```
typedef struct
{
char name[30];
int age;
}data;
```

و عرفنا منه متغيرين هكذا :

```
data a, b ;
```

و أعطينا المتغير a هذه القيم :

```
strcpy( a.name, "talal" ) ;
a.age = 20 ;
```

فبإمكاني ان اسند للمتغير b نفس محتويات المتغير a عن طريق هذه الجملة :

```
b = a ;
```

- إعطاء السجل أكثر من إسم أو إعطائه المتغيرات لحظة بناء السجل :

فلو كان لدينا السجل التالي :

```
typedef struct
{
char name[30];
int age;
}data, MyData ;
```

أستطيع أن اعرف المتغيرات سواء كان بـ data أو بـ MyData و كلها صحيحه.

فلو قلت :

```
MyData student ;
```

أو

data student ;

كانا سواء .

و هذا هو إعطاء السجل اكثر من إسم ، أما إعطاء السجل أكثر من متغير لحظة بناء السجل و بدون تحديد إسم للسجل يكون كالتالي :

```
struct
{
الاعضاء
}إسم المتغير ؛
```

فلو اردنا ان نعمل على ١٠٠ طالب فقط و متأكيدن أن العدد لن يزيد عن ١٠٠ طالب فالأفضل بناء السجل هكذا :

```
struct
{
    char name[30] ;
    int age ;
} student;
```

و هكذا يصبح student متغير و نقول :

student. name & student. age

طبعاً إلى الآن تعلمنا كيف ننشئ السجل بثلاثة طرق بقي الطريقة الرابعة و الاخيره و هي كالتالي :

struct (إسم السجل

}

الاعضاء

{(المتغيرات) ؛

أي نستطيع أن ننشئ سجل الطالب الذي تكرر علينا كثيراً بالطريقة الرابعه هكذا :

```
struct data
{
   char name[30] ;
   int age ;
} student;
```

هنا student سيكون متغير و data هو إسم السجل و هنا نستطيع في كل مرة نحتاج فيها لإنشاء سجل أن نشئ سجل بالطريقة :

```
struct data VAR ;
```

و إستعمال student كمتغير جاهز غير محتاج للتعيرف .

** نقطة أخيره :

في كل جزئ من أجزاء البرامج التي كتبتها و التعريفات و إنشاء المتغيرات في الدرس إستخدمت غالباً التعريف التالي :

```
typedef struct
{
```

```
    char name[30];
    int age;
}data;
```

و أنشئت المتغيرات كالتالي :

```
data VAR ;
```

ممكن تغييره إلى

```
struct data
{
char name[30] ;
int age ;
};
```

و لكن تعريف المتغير سيكون :

```
struct data VAR ;
```

الماكرو:

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

```
#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 الملون بالأحمر.

ولحساب القيمة يقوم البرنامج باستدعاء الماكرو (في الجزء الملون من آخر عبارة من البرنامج

)

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:

١- 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**).

٢- عملية الترجمة (**compilation**) وينتج عن هذه العملية البرنامج الهدف الذي يحمل عادة الأمتداد " **OBJ**" .

٣- 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.

مسائل على الباب

*دالة تاخد عددين من نوع انتجر وتعيد قيمة وتجمعهما

```
#include <iostream.h>
int addition (int a, int b)
{
int r;
r=a+b;
return (r);
}

int main ()
{  int z,b,n;
Cin>>b,n;
z = addition (b,n);
cout << "The result is " << z;
return 0;
}
```

*دالة لا تعيد قيمة من نوع فيود

```
#include <iostream.h>
void dummyfunction (void)
                    {
cout << "I'm a function!";
}
int main ()
{
dummyfunction ();
```

```
return 0;
}
```

*دالة لإيجاد المضروب من نوع انتجر

```
#include <iostream.h>
long factorial (long a)
{
if (a > 1)
return (a * factorial (a-1));
else
return (1);
}

int main ()
{
long l;
cout << "Type a number: ";
cin >> l;
cout << "!" << l << " = " << factorial (l);
return 0;
}
```

*دالة تعمل عمل الاس ؟

*// // السلسلة الثالية ١,٣,٧,١٥,٣١,٦٣,١٢٧ ؟

*// ترسم الاشكال التالية

```
*          *          *****

***        ***        ***

*****      *****      *

           ***

           *
```

*برنامج على المؤشرات

```
#include <iostream.h>
int main ()
```

```
{
int value1 = 5, value2 = 15;
int * mypointer;

mypointer = &value1;
*mypointer = 10;
mypointer = &value2;
*mypointer = 20;
cout << "value1==" << value1 << "/ value2==" <<
value2;
return 0;
}
```

value1==10 / value2==20

*ما ناتج البرنامج الاتي

```
#include <iostream.h>

int main ()
{
int numbers[5];
int * p;
p = numbers;  *p = 10;
p++;  *p = 20;
p = &numbers[2];  *p = 30;
p = numbers + 3;  *p = 40;
p = numbers;  *(p+4) = 50;
for (int n=0; n<5; n++)
cout << numbers[n] << ", ";
return 0;
}
```

10, 20, 30, 40, 50,

*اكتب دالة من نوع انتجر تعيد قيمة بواسطة مؤشر من نوع انتجر لايجاد المضروب للعدد المدخل ؟

* اكتب برنامج يقوم بعملية البحث عن احرف مكونة من سلسلة حرفية من نوع مؤشر ؟

aldopaee@hotmail.com

* اكتب برنامج رئيسي يحتوي على استدعاء للدوال التالية

– دالة تستقبل عدد من نوع صحيح مؤشر وتقوم بايجاد الاس لة

- دالة تستقبل عدد من نوع انتجر مؤشر وتقوم بايجاد عوامل العدد ؟

*برنامج للسجلات والدوال

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

struct movies_t {
char title [50];
int year;
} mine, yours;

void printmovie (movies_t movie);

int main ()
{
char buffer [50];

strcpy (mine.title, "2001 A Space Odyssey");
mine.year = 1968;

cout << "Enter title: ";
cin.getline (yours.title,50);
cout << "Enter year: ";
cin.getline (buffer,50);
yours.year = atoi (buffer);
```

```
Enter title: Alien
Enter year: 1979

My favourite movie is:
 2001 A Space Odyssey (1968)
And yours:
 Alien (1979)
```

aldopaee@hotmail.com

```
cout << "My favourite movie is:\n ";
printmovie (mine);
cout << "And yours:\n ";
printmovie (yours);
return 0;
}

void printmovie (movies_t movie)
{
cout << movie.title;
cout << " (" << movie.year << ")\n";
}
```

*متال للسجلات والمؤشرات

```
#inculde<iostream.h>
#include<string.h>
Typ struct{
Char name[30];
Int age;
}
Data;
Int main(){
Data*s,std;
S=&std;
Strcpy(std.name,''ammar aldopaee');
Std.age=20;
Cout<<std.name;
Cout<<std.age;
Cout<<s->name;
Return();
}
```


aldopaee@hotmail.com

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 ٢.

وبالمثل يكون A[1] = 3 ، A[2] = 4 ، وهكذا...

مثال لجمع عناصر صفوفه احاديه

```
#include <iostream.h>

int billy [] = {16, 2, 77, 40, 12071};
int n, result=0;

int main ()
{
for ( n=0 ; n<5 ; n++ )
{
result += billy[n];
```

```
}
cout << result;
return 0;
}
12206
```

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

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

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

        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.

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

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

وعناصر المصفوفة في هذه الحالة كما ذكرنا تحدد باستخدام رقمين رقم الصف ورقم العمود، فالعنصر ١٢ 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 .

B[1][2] = 35

B[2][1] = 80

B[0][2] = 15

B[2][2] = 16

مثال

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.

```
#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.

مسائل على الباب

* برنامج يطبع الشكل الاتي

```
#include<iostream.h>
#include<conio.h>
main(){
clrscr();
int s,d,a,b,m,c,x;
cin>>s;x=s;
```

```
1
212
32123
4321234
```

```
for(d=1;d<=s;d++){
for(m=x;m>=1;m--){cout<<" ";}
x-=1;
if (d>=2){
for(c=d;c>=2;c--){
cout<<c;
}}
for(a=1;a<=d;a++){
cout<<a;}
cout<<"\n";
}
```

*برنامج يعكس اقطار المصفوفة

```
#include<iostream.h>
#include<conio.h>
main(){int a,b,i,j;
clrscr();a=4;
int
x[5][5]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
,21,22,23,24,25};
for(i=0;i<5;i++){
for(j=0;j<5;j++){
if(i==j){b=x[i][j];x[i][j]=x[i][a];x[i][a]=b;}
} a-=1;
}
for(i=0;i<5;i++){
for(j=0;j<5;j++){

cout<<x[i][j]<<" ";
}
cout<<"\n";
}
getch();
}
```

*يجمع العناصر الواقعة تحت القطر الرئيسي

```
#include<iostream.h>
#include<conio.h>
main(){int a,b,i,j;
clrscr();a=4;
int
x[5][5]={11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,
27,28,29,30,31,32,33,34,35};
for(i=0;i<5;i++){
for(j=0;j<5;j++){
if(i>j){b+=x[i][j];}
}
}
for(i=0;i<5;i++){
for(j=0;j<5;j++){

cout<<x[i][j]<<" ";
}
cout<<"\n";
}       cout<<b;
getch();
}
```

*اكتب برنامج يعكس مصفوفة احادية بدون استخدام اداة شرطية ؟

*اكتب برنامج يحول الاعداد الفردية الى جهه اليمين والزوجية الى اليسار ؟

*اكتب برنامج للبحت عن اسم شخص معين في مصفوفة مؤشرات من نوع تشار ؟

*اكتب برنامج لادخال قيم لمصفوفة ذات بعدين من نوع مؤشر ويقوم بطباعتها؟

aldopaee@hotmail.com

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 (شكل١) يوضح الصورة العامة للإعلان عن الفصيلة

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

شكل ١ الصورة العامة للإعلان عن الفصيلة

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 (شكل ٢) يوضح كيفية استخدام الفصيلة في برنامج.

```
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:}
```

شكل ٢

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 07إلى 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 و١٨ 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)

```
# 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

```
#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

```
#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.

اختبار تجريبي

١/أوجد المتسلسلات التالية

*أوجد هذه السلسلة بدون استخدام عداد مجمع للرقم ٤ مثل c+=4

1 5 9 13 17 21 25

* 81 20 4 36 3 17 9 5 1

* 72 56 42 30 20 12 6 2

* 127 63 31 15 7 3 1

* 36 30 24 18 6 3 1

٢/اكتب برنامج يستقبل عدد ويقوم بالأعمال التالية N=245496

مجموعهم	A=30
متوسط الأعداد	B=5
عددهم	C=6
أصغرهم	D=2
العدد المكرر	E=4
أكبرهم	H=9

٣/اكتب برنامج يطبع العنصر الاكتر تكرارا ؟

٤/كتب برنامج بواسطة دالة تأخذ قيمتين من نوع أتتجر وترد للبرنامج الرئيسي أربعة نواتج .....جمعهم.ضربهم.طرحهم.باقي القسمة ؟

٥/دالة تعمل على ترتيب مصفوفة وقت الإدخال ولا تقبل الأعداد المكررة ؟

٦/ دالة بواسطة مؤشر تدخل n من الأعداد وتطبع العناصر الغير متكررة ؟

٧/برنامج يبحث عن اكبر عنصر ويبدله مع أول عدد زوجي ؟

٨/مصفوفة ثنائية البعد تعرف بالدالة الرئيسية وتستدعيها دالة فرعية بواسطة مؤشر ليغير عناصر الصف الثاني فقط وتطبع المصفوفة ؟

aldopaee@hotmail.com

٩/بواسطة دالة واحدة فقط اعمل برنامج بإدخال عناصر لمصفوفة وتبديل الأعداد الفردية مكان الزوجية وطباعة المصفوفة ؟

١٠/ أنشئ الأشكال التالية

```
1111111111
1222222221
1233333321
1234444321
1234554321
1234444321
1233333321
1222222221
1111111111  n=10
```

```
1111111111111111
122222222222221
123333333321
123444321
1234321
12321
121
 1   n=4
```

```
*           *
**         **
***       ***
****     ****
*****   *****
**********
*****   *****
****     ****
***       ***
**         **
*           *
```

الخاتمة

إلى كل من انتفع بهدى الكتاب

إلى كل مسلم ومسلمة

إلى كل من يشهد أن لإله إلا الله محمد رسول الله

ارجوا الدعاء لي ولمن يصلي على النبي الأمي

صلوات الله وسلامه عليك يا حبيبي محمد ابن عبد لله

والحمد لله.

aldopaee@hotmail.com

المراجع
كيف تبرمج بلغة السي ++ .....د/صلاح الوه جي
المرجع الشامل في برمجة السي.....عوض منصور
بعض صفحات الويب

aldopaee@hotmail.com