الجامعة التقنية الشمالية / المعهد التقني نينوى

قسم أنظمة الحاسوب / المستوى الاول

المبادئ الأساسية للغة C++

تقسيم لغات البرمجة
تقسم لغات البرمجة إلى ثلاث مستويات

١-لغات ذات مستوى الأعلى
HIGH LEVEL LANGUAGE (H.L.L)

- هذا النوع من اللغات لا يتطلب من المبرمج سوى معرفة كيف تعمل ايعازات اللغة بدون الحاجة إلى معرفة كيف يتعامل مع الذاكرة أو مع المعالج الدقيق (MICRO PROCESSOR) مثل لغة بيسك وفورتران وباسكال PASCAL , FORTRAN, BASIC

٢-لغات ذات المستويات الدنيا
LOW LEVEL LANGUAGE (L.L.L)

- هذا النوع من اللغات يتعامل مع السجلات (REGISTERS) ، ويتطلب من المبرمج معرفة الايعازات الخاصة بالذاكرة وكيفية التعامل مع المعالج الدقيق (MICRO PROCESSOR) مثل لغة الاسمبلي ولغة الماكنة ASSEMBLY LANGUAGE & MACHINE LANGUAGE .

٣-لغات ذات المستوى المتوسط
MIDDLE LEVEL LANGUAGE(M.L.L)

- هذا النوع من اللغات تمتلك بعض من خواص كلا النوعين أعلاه ، أي إنها تمتلك القدرة على التعامل مع مواقع الذاكرة ولها خواص اللغات العليا مثل لغة C و C + + .

تطورات سريعة وأصبحت C++ و C تطورت لغة من أكثر لغات البرمجة شيوعاً وإقبالا وذلك لسببين رئيسيين :

أولا : مرونة اللغة ، حيث تعطي المبرمج حرية الاختيار على مستوى البرامجيات SOFTWARE أو الأجهزة HARDWARE .

ثانياً : كتب نظام التشغيل يونكس UNIX بلغة C .

مميزات لغة C :-

من بين سائر لغات البرمجة بالمميزات التالية : C تمتاز لغة

- لغة صغيرة : إذا تستخدم عدداً من الكلمات المحجوزة KEYWORDS اقل من تلك التي PASCAL . تستعملها لغة مشهورة مثل باسكال ، وهذا أمر مرغوب في عمليات البرمجة
- بنيوية STRUCTURAL : حيث يبنى البرنامج من مجموعة من جمل التحكم التي تهدف إلى إعطاء البرنامج صورة متماسكة وسهلة التداول.
- لغة غنية بالدوال FUNCTION : C توفر لغة للمبرمج عدداً من دوال الإخراج والإدخال والدوال الرياضية والحسابية ودوال معالجة النصوص ، الأمر الذي يسهل عملية بناء البرنامج بهذه الدوال وهي جاهزة للاستخدام.

- لغة جامعة : لغة واقعة بين اللغات الراقية مثل باسكال وفورتران وبيسك وكوبل وغيرها ، واللغات الواطئة MACHINE LANGUAGE ولغة الآلة ASSEMBLY LANGUAGEمثل لغة التجميع (اسمبلي) . إذا أنها تستعمل تعليمات (ايعازات) اللغات الراقية ، كما تتعامل مع الأدوات الدقيقة على مستوى الآلة (BITWISE OPERATORS).
- UNIXهي لغة نظام : للحاسبات المتطورة ، ويعتبر هذا مصدر قوة للغة .
- لا تعتمد على الأجهزة بصورة كبيرة : يعني انه يمكن نقل البرمجيات المكتوبة بها بين أجهزة الحاسبات المختلفة .

- لغة معتمدة قياسياً STANDARD (ANSI) : حيث اعتمدت في أصلها على النمط القياسي AMERICAN NATIONAL STANDARD INSTITUTE).
- لغة سريعة UNIX : حيث تعتبر سريعة التنفيذ في حالة كون نظام التشغيل هو نظام يونكس .
- ذات نظام توزيع حركي اقتصادي للذاكرة : حيث تمتلك عددا من الدوال الخاصة بالتوزيع الحركي (MEMORY DYNAMIC ALLOCATION) ، وبذلك يتم استغلال جميع الاقتصادي للذاكرة (OPTIMUM) الشواغر للذاكرة بشكل ممتاز.
- لغة مرنة FLEXIBLE : توفر عدد كبير من التعليمات القوية التي تعطي المبرمج حرية الاختيار والحركة .

الإعلان (Declaration) في لغة C و C + +

يقصد بالإعلان هو تمثيل الثوابت والمتغيرات في اللغة وكيفية حجز مواقع لها في الذاكرة وتقسم إلى الأنواع التالية :

Variables المتغيرات		Constants الثوابت	
المتغيرات العامة Global	المتغيرات المحلية Local	الثوابت العامة Global	الثوابت المحلية Local
int float double char	int float double char	#define const	#define const

الأسماء التعريفية Identifiers :

تطلق الأسماء التعريفية على المتغيرات والدوال والمؤشرات والعناوين .. الخ ويتراوح طول الاسم من حرف واحد إلى ٣١ حرف (رمز) وينبغي إن يكون الرمز الأول من الاسم حرف أبجدي أو شارحة سفلى (Under score) ويمكن أن تكون بقية الرموز مؤلفة من حروف أبجدية أو أرقام.

الأسماء التالية مقبولة :

A , x , ali , _ y , Ahmad , System , system

الأسماء التالية غير مقبولة :

7-up , #x , ma!a , &y

الثوابت Constants :

- الثوابت العددية : Numeric Constants

ا - الأعداد الصحيحة : Integers تكتب الأعداد بدون فاصلة عشرية وتصنف حسب طولها .

وتمثل بطريقتين :-

الطريقة الأولى :

```
# define  x  100
# define y  -5
```

الطريقة الثانية :

```
const   int  x = 50 ;
```

ومن الممكن كتابة

```
const   z = 120 ;
```

ب- الأعداد الحقيقية Floating – Point Constants :

وهي الثوابت العددية التي تستعمل الفاصلة العشرية

وتمثل بطريقتين :-

الطريقة الأولى :

```
# define   pie  3.14
# define   t -7.457
```

الطريقة الثانية :

```
const  int  y = 58 ;
const  float  m = 63.75 ;
const  n = 120 ;
```

ملاحظة: تمثل الأعداد الآسية E-formatted constants كما في الأمثلة التالية :

1.35e -3 = 0.00135 , 2.0e4 = 20000

٢. الثوابت الرمزية : String constants

هي سلسلة الحرف والأرقام والرموز الخاصة التي توضع بين علامتي الاقتباس أو مايسمى بالحاصرات العلوية المزدوجة (" ") أو الفردية (' ')

مثل :

'k' , "ali" , "ahmad"

المتغيرات Variables:

إن المعلومات المراد تخزينها تكون عادة مختلفة القيم ، مثل القيم العددية الصحيحة أو الحقيقية أو الرمزية لذا ينبغي في بداية البرنامج أن نعلن عن أنواع المتغيرات التي نريد استعمالها في البرنامج .وتقسم المتغيرات إلى قسمين من المتغيرات حسب موقع الإعلان عنها ووظائفها .

- المتغيرات العامة : Global Variables

إن المتغيرات العامة تكون لها قيمتها المعروفة في البرنامج من أوله إلى أخره وينشأ هذا المتغير خارج حدود الدوال .

```
int radius ;
main ( )
{
  radius = 10;
  func1( )
    {
      float  area ;
      area = 3.142 * radius * radius ; /* مرتبط بالتعريف العام */
    }
   func2 ( )
    {
      int radius ; /* غير مرتبط بالتعريف العام */
      ……..
      ……..
    }
```

ملاحظة : لا ينبغي استعمال متغيرات عامة لا لزوم لها للأسباب التالية :-

١. إسراف في استخدام المخازن .

٢. يجعل الدالة اقل تعميماً .

٣. حدوث غموض في البرنامج .

٢.المتغيرات الموضعية : Local Variables

تسمى المتغيرات التي يتم الإعلان عنها داخل الدالة بالموضعية أو تسمى أحيانا بالمتغيرات الذاتية (Automatic Variables) حيث تستعمل الكلمة المحجوزة (auto) للإعلان عن هذه الصورة .
وتسمى المتغيرات الموضعية بهذا الاسم لأنها تكون غير معّرفة خارج الدالة المعرفة ضمنها. ان المتغير الموضعي ينشأ له قيمة وكيان عند ابتداء تنفيذ الدالة التي ينتمي إليها فقط ، وينتهي عند الانتهاء من تنفيذ تلك الدالة .

مثال :

```
func1( )
{
float  y ;
y = 12.5 ;
}
```

```
func2( )
{
  float   y ;
   y = 10.5 ;
}
```

```
func3( )
{
    int   y ;
   y = 2  ;
}
```

إن قيم y في كل من الدوال الثلاثة في نفس البرنامج ليس بينهم أي علاقة .

وتأخذ المتغيرات الأنواع الشائعة الاستخدام التالية (int , float), اضافة الى الانواع التي تذكر التفصيل الدقيق للنوع (double , char , والتي تسمى الأنواع المعدلة (Type modifiers) وهي (short,long,signed,unsigned) والتي تستخدم مع النوع (int) وتستخدم (signed,unsigned) مع النوع (char) وتستخدم (long) مع النوع (double).وكما موضح في الجداول التالية .

الأنواع الشائعة الاستخدام

Type النوع	طول النوع بالبايت Sizeof( )	Range المدى
char	1	127…..-128
int	2	32767…..-32768
float	4	3.4E38…..3.4E-38
double	8	1.7E308….1.7E-308
void	الخالية	لا تأخذ أي مدى

الأنواع المعدلة أو مع التفصيل الدقيق
Type Modifiers

Type النوع	Length الطول	Range المدى
unsigned int	2	0…..65565
signed int	2	32767…. -32768
short int	2	32767….. -32768
long int	4	2147483647….-2147483648
singned char	1	127….. -128
unsigned char	1	0…..65565
long double	10	25-خانة من الدقة

التعابير الحسابية Expressions:

التعبير الحسابي هو مجموعة من العمليات الحسابية الخاصة بالإعداد والثوابت والمتغيرات والدوال الحسابية ، مثال: x*y وتعني xy

جملة التعيين : Statement Assignment

وتأخذ الصيغة العامة variable_name = expression

شروط جملة التعيين :

- يوضع اسم المتغير في الطرف الأيسر .
- لا يجوز أن يكون الطرف الأيسر ثابتاً أو دالة أو عملية حسابية .
- يمكن للطرف الأيمن أن يكون ثابتاً أو دالة أو تعبير حسابي .
- تنتهي الجملة بفارزة منقوطة (;) .

```
a = 10 ;
y = a * x + b ;
x = sin(36.8);
i = i + 1;
x = x + 2 ;
y = y * 4 ;
k = k /6 ;
c = (a = 3 ) + ( b = 4 ) ;
a = b = c = 0 ;
z = ( y = x + 5 ) / x ; تعني   y = x + 5  ,  z = y /x
```

تعين قيمة ابتدائية للمتغير : Variable Initialization

ممكن تعيين قيمة أولية للمتغير عند الإعلان عنه مباشرة .

مثال :

```
int x = 10;
float  sum = 0 ;
char  name = 'A' ;
double pie = 3.1415922654 ;
```

الأدوات المستعملة في اللغة Operators :-

١. الأدوات الحسابية Arithmetic Operation :

الأداة operator	الوظيفة Function	النوع
-	كإشارة سالبة	أحادية Unary
-	للطرح	ثنائية Binary
+	الجمع	ثنائية Binary
*	الضرب	ثنائية Binary
/	القسمة	ثنائية Binary
%	باقي القسمة	ثنائية Binary
- -	للنقصان	أحادية Unary
+ +	للزيادة	أحادية Unary

Increment and Decrement -: الزيادة والنقصان

+ + لزيادة القيم بمقدار واحد .

- - إنقاص القيم بمقدار واحد .

مثال : a++ أو ++a وتعني a = a+1

a-- أو --a وتعني a = a – 1

ملاحظة : هناك فرق بين a++ و ++a

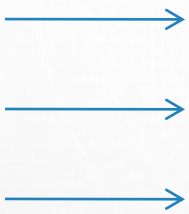
تزداد قبل استخراج قيمة التعبير . a هنا ++a

الحالية قبل الزيادة a في التعبير يستعمل قيمة aa++

بمقدار1 .

مثال :

```
int x , y , z ;
x = y = z = 0 ;
x=2    y=1    z=1
x = y++ + z++ ;
z=2
x = y-- - --z ;
```



```
x = ++y + ++z ;
x=2    y=2
x=1    y=1    z=1
```

* من الممكن إعادة جمل التعيين وكتابة التعابير على النحو التالي :

```
x+=2 ;          x = x + 2;
y*=4 ;          y = y * 4;
z*=(5+x) ;      z = z * (5+x);
b/=(3+y) ;      b = b/(3 + y) ;
```

- من الممكن إعادة جمل التعيين وكتابة التعابير على النحو التالي :

x+=2 ;	x = x + 2 ;
y*=4 ;	y = y * 4;
z*=(5+x) ;	z = z * (5+x);
b/=(3+y) ;	b = b/(3 + y) ;

أولويات الأدوات الحسابية : Precedence of Arithmetic Operation

الأداة	الأولوية	
++ --	1	التي تأتي قبل العدد (على يسار العدد)
-	2	الإشارة السالبة قبل العدد(على يسار العدد)
* / %	3	
+ -	4	
=	5	
++ --	6	التي تأتي بعد العدد (على يمين العدد)

ملاحظة : الأقواس تأخذ الأولوية في التنفيذ وقبل الزيادة والنقصان .

الأدوات العلائقية والمنطقية Relational and logical operators:

تكون نتيجة إجراء عمليات المقارنة المنطقية هي إما نعم (true) أو (false) ويرمز لنعم بالقيمة (1) أو أي قيمة غير الصفر .

ويرمز للا بالقيمة (0)

الأدوات العلائقية Relational Operators:

الأداة	المعنى
>	اكبر من
> =	اكبر من أو يساوي
<	اصغر من
< =	اصغر من أو يساوي
= =	يساوي
=!	لا يساوي

الأدوات المنطقية Logical Operators :

الأداة	المعنى
&&	AND أي و
||	OR أي أو
!	NOT للنفي (أداة أحادية) (unary)

أولويات الأدوات العلائقية والمنطقية :

الأولوية	الأداة
1	!
2	> > = < < =
3	== !=
4	&&
5	||

الأدوات الدقيقة Bitwise Operators.

تستخدم الأدوات الدقيقة على مستوى وحدة التخزين [Bit] [Binary Digit] وتستعمل مع المعطيات الصحيحة int والرمزية char ولا تستعمل مع غيرها من المعطيات .

الجدول التالي يبين الأدوات الدقيقة ووظيفة كل منها وأسبقيتها .

الأداة	وظيفتها	الأسبقية
~	NOT (unary)	1
>>	إزاحة إلى اليمين	2
<<	إزاحة إلى اليسار	2
&	AND (و)	3
^	XOR (أو الاستثنائية)	4
|	OR (أو)	5

الإزاحة:-

```
x = x>>2 ;
```

يتم في هذا المثال إزاحة الـ Bits بمقدار خانتين إلى اليمين والخانات الفارغة نحو أقصى اليسار تملئ بصفر .

```
x = x<< 3 ;
```

يتم في هذا المثال إزاحة الـ Bits نحو اليسار بمقدار ثلاث خانات وتملئ الخانات الفارغة نحو أقصى اليمين بصفر .

ملخص أوليات الأدوات Summary Operator Precedence

رقم الأولية	الأداة	اسم وعمل الأداة
1	-> . [ ] ( )	الأقواس والنقطة أو السهم
2	sizeof( ) + + -- * & - ! ~	الأدوات الأحادية
3	% / *	الضرب أو القسمة أو باقي القسمة
4	- +	الجمع أو الطرح
5	<< >>	الإزاحة إلى اليمين أو إلى اليسار
6	>= > <= <	أدوات الأصغر والأكبر العلائقية
7	== =!	المساواة العلائقية أو عدم المساواة
8	&	(الواو) الدقيقة
9	^	(أو) الدقيقة أو الاستثنائية
10	|	(أو) الدقيقة
11	&&	(الواو) المنطقية
12	||	(أو) المنطقية
13	?	الأداة الشرطية
14	= = + = - = *= /=	أداة المساواة للمعادلات مع الأدوات الحسابية
15	-- + +	أدوات الزيادة والنقصان المتأخرة
16	,	الفاصلة بين التعابير الحسابية والمنطقية

? The conditional Operator أداة الشرط

وتتعامل مع ثلاث كميات

variable = expression 1 ? expression 2 : expression 3 ;

Example 1:

```
a = ( b < c ) ?  b : c ;
```

وتعني العبارة أعلاه :

```
if (b<c)
 a = b ;
 else
  a = c ;
```

Example 2:

```
a = (d<e) ? 300 : 400 ;
```

في حالة تحقق الشرط فان a = 300 وإذا لم يتحقق فان a = 400

الإدخال والإخراج في لغة Input & Output C + +-:

تستخدم جملة (scanf) وجملة (cin) لإدخال المتغيرات من لوحة المفاتيح إلى الذاكرة ، والصيغة العامة لكتابتها هي

```
scanf( "format  " , data to be entered ,) ;
cin>> var1>>var2>>…….>> varn ;
```

وتستخدم جملة (printf) وجملة (cout) لطباعة المتغيرات من الذاكرة إلى الشاشة . أو طباعة عبارات توضيحية أو ناتج تعبير حسابي ، والصيغة العامة هي :

```
printf("format " , printed material ) ;
cout << exp1<<exp2 ……<< exp n ;
```

ملاحظة :- في جملة القراءة (scanf) تسبق البيانات المراد إدخالها (data to be entered) علامة & التي تمثل عنوان المتغير في الذاكرة أما في حالة قراءة سلسلة حرفية فلا يحتاج أن نسبق المتغير بعلامة & لان السلسلة الحرفية تتعامل مع العنوان .

القرارات

العبارة if :

إن أسهل طريقة لاتخاذ قرار هي بواسطة العبارة if
تتألف العبارة if من الكلمة الأساسية if يليها تعبير اختبار بين أقواس ، ويتألف جسم القرار الذي يلي ذلك ، اما عبارة واحدة او عدة عبارات تحيطها أقواس كبيرة كما في المثال التالي :

```
if ( x > 100 )
statement 1  ;
```

تعبير الاختبار (condition)
عبارة واحدة

IF (SPEED <=55) تعبير الاختبار (CONDITION)

```
{
  STATEMENT1 ;
  STATEMENT2 ;
  STATEMENT3 ;
        .    ;
        .    ;
        .    ;
}
```

عدة عبارات

لا توجد فارزة منقوطة

العبارة : if .... else

في العبارة (if) البسيطة يحدث شي اذا كان الشرط صح ولكن اذا لم يكن الشرط صح لن يحدث شئ على الاطلاق . لنفرض اننا نريد حدوث شئ في الحالتين ، في حالة اذا كان الشرط صح يحدث شئ واذا لم يكن الشرط صح يحدث شئ آخر، لتحقيق ذلك نستعمل العبارة . **if .... else**

وكما في حالة (if) يمكن أن يتألف قسم (else) من عدة عبارات تحيطها أقواس كبيرة

Condition : تعبير الاختبار

يمكن أن تكون عبارة الاختبار معقدة مثل التعبير التالي :

نزيد يوم واحد إلى أيام شهر شباط في حالة السنة كبيسة ، ويحول إلى آذار إذا كانت السنة غير كبيسة .

```
if (day = = 28 && month = = 2 && year % 4 = = 0 &&
                                    year % 100 != 0 )
  day = 29 ;
  else
    {
       day = 1 ;
       month = 3 ;
     }
```

عبارات if ….else المتداخلة (Nested if …else):

```
if (age < 2)  cout<< "\ n Infant " ;
```
إذا كان العمر اصغر من سنتين أطبع (Infant)

```
else
if ( age < 18 ) cout << "\ n Child ";
```
وإذا لا إذا كان العمر اصغر من (18) أطبع (Child)

```
else   cout << " \ n Adult ;
       )
```
وإذا لا أطبع (Adult)

مثال أخر : إيجاد اكبر عدد من بين ثلاث أعداد .

```
# include<iostream.h>
main( )
 {
     int a ,b , c , max;
     cout << "Entre three integers : " ;
     cin >> a >> b >> c ;
      if ( a > b )
         if ( a > c ) max = a ;          / /  a > b and  a > c
         else max = c ;                   / / c > = b > b
     else
          if ( b > c ) max = b ;         / / b > = a and  b > c
          else  max  = c ;                / / c > = b > = a
   cout << " The maximum is " << max << endl ;
 }
```

في أول تنفيذ للبرنامج :

```
Enter three integers :  15   35   48
The maximum is  48
```

في التنفيذ الثاني :

```
Enter three integers : 95  65  35
The maximum is  95
```

في أول تنفيذ للبرنامج اختبار الشرط (a > b) غير صحيح لذلك يتم تنفيذ الشرط (b > c) الذي يلي ثاني else

وهو أيضا غير صحيح لذلك يتم تنفيذ ثالث else والتي تخصص c للمتغير max . في ثاني تنفيذ للبرنامج ، الشرط (a > b) يكون صحيحا وكذلك الشرط (a > c) أيضا يكون صحيحاً ، لذلك خصصت a للمتغير max .

مثال آخر : هذا البرنامج يحول درجات الطالب إلى ما يكافئها من الحروف الأبجدية .

```
# include < iostream.h>
main( )
{
    int   score ;
     cout << "Enter the test score : " :
     cin<< score ;
     if  ( score > 100 ) cout  << " Error : score is out of range . " ;
     else   if  ( score > = 90 ) cout << 'A'    ;
     else   if  ( score > = 80 ) cout << 'B'    ;
     else   if  ( score > = 70 ) cout << 'C '   ;
     else   if  ( score > = 60 ) cout << ' D '  ;
     else   if  ( score > = 50 ) cout << ' E '  ;
     else   if  ( score > = 0 )   cout <<  ' F ' ;
     else  cout << " Error : score is out of range . " ;
}
```

جملة switch ….case :

وهي من جمل الاختبار أيضا ومبدأ عملها هو تنفيذ جملة او كتلة block من بين عدة جمل أو كتل blocks حسب فيمه المتغير في جملة switch والصيغة العامة هي :

```
switch ( var )
                {
case    value1  : statement 1 ;     / /  var = value1جملة او جمل تنفذ عندما
                  break ;
case    value2  : statement 2 ;    / /   var = value2جملة او جمل تنفذ عندما
                  break ;
 case   value3  : statement 3 ;     / /   =   =   =   =   =
                  break ;
                                .
                                .
 default   : last statement  ;     / / يتم تنفيذها عند عدم تنفيذ أي من العبارات أعلاه
                }
```

ملاحظات :-

- تستخدم switch حالة المساواة في الشرط ، بينما يمكن لجملة if الشرطية استخدام جميع علاقات المساواة وغير المساواة العلائقية منها اوالمنطقية .
- عند استعمال switch ينبغي ان تكون قيم ارقام الحالات (value) تختلف عن بعضها ، فليس منطقياً استعمال ثابتين مختلفين من اجل حالة واحدة .
- اذا كان الثابت (value) رمزيا أي character ، فانه يتحول ذاتيا الى القيمة الصحيحة المكافئة له .
- يعتبر استعمال break و default في بنية switch اختيارياً ويمكن حذفها كليهما .
- تعتبر كل حالة من الحالات في بنية switch مستقلة عن غيرها .
- عند وجود break في نهاية أية من حالات switch فان البرنامج يتوقف عندها ، ولا يصل التنفيذ حالات switch التالية بعد break عند أية حالة ، فيتوقف عندها ولا يستمر .

عبارة break :-

عند وضع عبارة break في جسم الحلقات (for , while , do..while) فان تنفيذ الحلقة يتوقف عند الوصول الى عبارة break ويخرج الى البرنامج الرئيسي .

عبارة continue :-

عمل continue هو تكرار تنفيذ الحلقة في هيكل (جسم) الحلقات (for , while) و تهمل الجمل التي بعدها ضمن الحلقة (loop) . مثال :- do…while ,

```
# include < iostream.h >
 main ( )
   {
    int   num ;
    for(int t = 0 ; t < 100 ; t+ + )
       {
          cin >> num  ;
          if ( num < 0 )
             continue  ;
          cout << num ;
          }        // end of for
    }             // end of main( )
```

في المثال أعلاه تعمل جملة continue على إهمال جملة cout التي بعدها في حالة العدد المدخل سالب و تقوم بتكرار الحلقة مرة أخرى .

الحلقــات:

لا يمكنك كتابة برامج مثيرة للاهتمام من دون الحلقات . فالبرنامج الخالي من الحلقات يفعل الأشياء مرة واحدة ثم ينتهي . إما البرنامج الذي يحتوي على حلقة فيستطيع فعل الأشياء قدر ما تدعو الحاجة .

هناك ثلاث أنواع من الحلقات في لغة C+ + :

- **الحلقة (العبارة) while :**

تتيح لك الحلقة while فعل شئ ما مراراً وتكراراً الى ان يتغير شرط ما . هذا الشرط هو شئ يمكن التعبير عنه بقيمة صح / خطأ . مثلاً قد تواصل الحلقة while الطلب من المستخدم إدخال حرف ، وتستمر الحلقة في التكرار إلى أن يكتب المستخدم الحرف q (اختصار لكلمة quit اي إنهاء) .

ادناه مثلاً عن حلقة while تتصرف هكذا:

```
while (ch != 'q')
{
cout << “Enter a character : “  ;
cin >> ch  ;
}
```

اذا لم يكتب المستخدم الحرف q ستستمر الحلقة في التكرار ، قد يبدو التفاعل كالتالي :

```
Enter a character : c
Enter a character : a
Enter a character : t
Enter a character : s
Enter a character : q
```

تعبير الاختبار

```
while  (n!=n )
statement  ;
```

لاحظ : لايوجد فارزة منقوطة

تعبير الاختبار

```
while  ( v2 > 45)
{
   statement ;
   statement ;
   statement ;
 }
```

لاحظ : لايوجد فارزة منقوطة

لاحظ : لايوجد فارزة منقوطة

تتألف الحلقة while من الكلمة الأساسية while يليها تعبير اختبار (يسمى أيضاً تعبير شرط أو شروط) محصور بين أقواس حاصرة (لأكن بدون فارزة منقوطة) . إذا كان جسم الحلقة يتألف من عبارة واحدة ، لا نحتاج إلى أقواس حاصرة كبيرة مثل :

```
while ( n <100 )
    n = n * 2 ;
```

(عبارة واحدة في جسم الحلقة لذا لا يحتاج إلى أقواس كبيرة)

إلى أن لا تعود هذه القيمة n ستستمر هذه الحلقة في مضاعفة قيمة المتغير)عندها تتوقف 100(أي عندما تصبح اكبر من أو تساوي 100اصغر من الحلقة .

- إذا كانت قيمة n تساوي 1 عند بداية الحلقة ، فكم ستكون هذه القيمة عند انتهاء الحلقة ؟

ستستمر القيمة في التسلسل 128,64,32,16,8,4,2,1حيث ستتوقف الحلقة (اي سينتقل التنفيذ الى العبارة التي تلي الحلقة)

ملاحظة :

لاحظ انه يتم فحص تعبير الاختبار قبل تنفيذ جسم الحلقة ، اذا كان الشرط خطأ عند دخول الحلقة ، فلن يتم تنفيذ جسم الحلقة ابداً . هذا الامر مناسب في بعض الحالات ، لاكنه يعني ان تكون حذراً ان تكون قيمة المتغير المستعمل في تعبير الاختبار مناسب قبل الدخول الى الحلقة . المتغير ch في المثال الاول لايجب ان تكون قيمته q عند دخول الحلقة ، والا لن يتم تنفيذ الحلقة ابداً . المتغير n في الحلقة الثانية يجب تمهيده عند قيمة اقل من 100 .

٢.الحلقة do :

الحلقة do (غالباً ما تسمى الحلقة do .. while)

تعمل هذه الحلقة مثل حلقة while ما عدا أنها تفحص تعبير الاختبار بعد تنفي جسم الحلقة . هذا الامر جيد عندما تريد القيام بشئ ما مرة واحدة على الاقل ، بغض النظر عن الحالة (صح | خطأ) الأولية للشرط .

مثال : تواصل الحلقة في المثال أدناه جمع رقمين يدخلهما المستخدم ، وعندما يدخل الرقم 0 للرقم الاول تتوقف الحلقة .

```
do
    {
cout << "\n Enter two numbers ( to quit, set first number to  0 ) :  " ;
        cin >> x >> y ;
        cout << " The sum is  = " << x + y ;
    } while  ( x != 0 ) ;
```

```
do
statement ;
while (ch != 'n' ) ;
```

لاحظ : لا توجد فارزة منقوطة

جسم الحلقة عبارة واحدة

لاحظ : توجد فارزة منقوطة

تعبير الاختبار

```
do
{
    statement ;
    statement ;
    statement ;
} while ( numb < 96 ) ;
```

جسم الحلقة من عدة عبارات

لاحظ : فاصلة منقوطة

تعبير الاختبار

(التركيب النحوي للحلقة do)

٣. الحلقة التكرارية for :

يتم التحكم في هذه الحلقة التكرارية بثلاث اجزاء منفصلة : القيمة الابتدائية initialization ، وشرط الاستمرار continuation condition والقيمة الجديدة update . ان الصورة العامة لجملة الحلقة التكرارية for هي :

for (initialization ; continuation condition ; update)

القيمة الابتدائية وشرط الاستمرار والقيمة الجديدة يمكن أن يكونوا فارغين أي بدون قيم .

مثال : مجموع مربعات الأعداد :

```
# include<iostream.h>
  main( )
    {
       int n ,sum = 0 ;
       cout << "Enter a positive integer  : " ;
        cin >>  n ;
       for ( int i =1 ; i < = n ; i + + )
            sum + = j* j ;
       cout << "The sum of the first " << n << " squares is "
                 << sum << end ;
     }
```

في هذا البرنامج القيمة الابتدائية هي int i = 1 وشرط الاستمرار هو i < = n والقيمة الجديدة هي i + + . من المعتاد ان يكون الاعلان عن متغير التحكم مع تخصيص القيمة الابتدائية له في الحلقة التكرارية for ،

على سبيل المثال متغير التحكم i في البرنامج السابق تم الإعلان عنه بداخل جزء التخصيص int i = 1 ، وهذه خاصية جميلة في لغة C++ . على أي حال بمجرد الإعلان عن متغير التحكم بهذه الطريقة فانه يجب أن لا يعاد الإعلان عنه في حلقة تكرارية for فيما بعد ، على سبيل المثال .

```
for  ( int  i = 0 ; i < 100 ; i + + )
    sum  + = i * i ;
for ( int i = 0 ; i < 100 ; i + + )        / / ERROR :
                                    ( i    has already been declared)
    cout << i * i * i  ;
```

نفس متغير التحكم يمكن أن يستخدم مرة أخرى بدون إعادة الإعلان عنه مرة أخرى

```
for ( i = 0 ; i < 100 ; i + + )          / / OK
    cout << i * i * i  ;
```

Arrays المصفوفات

المصفوفة عبارة عن سلسلة من البيانات المهيكلة التي تتكون من مجموعة من العناصر المتجانسة (ذات نوع واحد) ويتم ترقيمها بالتتابع 0, 1, 2, 3,..... هذه الأرقام تسمى الفهرس index او القيم الجانبية subscripts للصف . ان تعبير القيم الجانبية يتم كتابته كالتالي : a0 , a1 , a2 ,..... هذه الاقام الجانبية تحدد مكان العنصر في المصفوفة .

إذا كان اسم المصفوفة هو a ، فان a[0] هو اسم العنصر الموجود في المكان رقم صفر (اول مكان) و a[1] هو اسم العنصر الموجود في المكان رقم 1 ، وهكذا . وعليه اذا كانت المصفوفة تحتوي على n من العناصر فان أسماء هذه العناصر ستكون a[0] , a[1] , a[2] ,.....,a[n-1] .

ويمكن تصور المصفوفة كالأتي :

a	11.11	32.65	54.43	12.89	34.98
	0	1	2	3	4

هذا الرسم يبين مصفوفة أحادية اسمها a تتكون من خمسة عناصر (أعداد حقيقية من نوع float) .

الشكل أعلاه يمثل في الحقيقة جزء من الذاكرة الخاصة بالحاسبة لان أي مصفوفة يتم خزنها عادة بهذه الطريقة بحيث تكون كل عناصرها في تتابع حقيقي .

معالجة الصفوف :

ظاهريا جميع البرامج المفيدة تستخدم المصفوفات . من الأسباب التي تجعل المصفوفات لها فائدة هي إمكانية السماح لاسم واحد بفهرس متغير إن يستخدم بدلا من أسماء متعددة ، وهذا من السهل عمل أشياء كثيرة كان من الصعب جداً تحققها بدون استخدام المصفوفات .

مثال :-

هذا البرنامج يقرأ 4 أرقام ثم يقوم بطباعتها بترتيب عكسي لعملية قراءتهم .

```
# include < iostream.h>
main( )
 {
    double  a[4]  ;
     cout << " Enter  4  real  number  : \ n " ;
     for ( int  i = 1 ; i<=4 ; i++)
        {
          cout <<  i  << " :   "  ;
          cin >> a[i-1];
         }
      cout << " Here they are in reverse order : \n "  ;
      for ( i = 3 ; i > = 0 ; i -- )
      cout << " \ t  a[" << i << "] = " << a[i] << endl   ;
 }
```

أمر التعريف double a [4] يعرف a على أنها مصفوفة أحادية من 4 عناصر كلها من نوع double

الحلقة for الأولى تسمح للمستخدم بإدخال أرقام حقيقية في هذه العناصر الأربع . بعد ذلك تقوم الحلقة for الثانية بطباعة هذه الأرقام .

استخدام ثابت رمزي لتعريف ومعالجة صف :

```
main ( )
  {
     const  int SIZE = 4 ;
      double   a [ SIZE ] ;
      cout << " Enter  " <<  SIZE << " real number : \n " ;
      for (int i = 1 ; i << = SIZE  ; i + + )
           {
             cout << i << " :  " ;
             cin >> a [i-1] ;
            }
        cout << " Here they are in reverse order : \n " ;
        for ( i = SIZE – 1 ;  i > = 0 ;  i -- )
           cout << " \ t  a[ "  << i  << " ] = " << a [i] << endl ;
  }
```

الثابت الصحيح SIZE تم إعطاؤه القيمة الابتدائية 4 ، بعد ذلك تم استخدام هذا الثابت في الإعلان عن الصف a ، ومطالبة المستخدم بإدخال هذا الثابت وكذلك التحكم في الحلقة for . البرنامج يعمل بنفس الطريقة كما في البرنامج السابق .

إن الشكل العام للإعلان عن مصفوفة أحادية هو :

نوع عناصر المصفوفة اسم المصفوفة [عدد عناصر المصفوفة ;]

type array name [array-size] ;

حيث type هو نوع عناصر المصفوفة :

(int , double , float , char ….) .

في لغة C + + القياسية تتطلب ان يكون حجم الصف (array-size) عدد ثابت صحيح موجب .

إعطاء قيم ابتدائية للصف :

في لغة C + + ، أي مصفوفة يمكن تخصيص قيماً ابتدائية له باستخدام قائمة تخصيص كالتالي :

```
float  a[4] = { 32.6 , 76.3 , 21.1 , 46.2 } ;
```

ان القيم الموجودة في هذه القائمة يتم تخصيصها لعناصر الصف بنفس ترتيبها في القائمة .

في المثال التالي يبين كيفية تخصيص قيماً ابتدائية للمصفوفة :

```
main ( )
   {
     double  a [ 4 ] = { 32.6 , 76.3 , 21.1 , 46.2 ] ;
     for ( int i = 0 ; i < 4 ; i + + )
     cout << " a [ " << i << " ] = " << a[i] << endl ;
   }
```

سيكون الناتج (الإخراج) عند تنفيذ البرنامج كالتالي :

```
a[0] = 32.6
a[1] = 76.3
a[2] = 21.1
a[3] = 46.2
```

لاحظ ان قائمة القيم الابتدائية تحوي 4 عناصر ، وهو نفس الحجم المحدد في آمر إعلان المصفوفة .

ملاحظة : اذا كان للمصفوفة عدد من العناصر أكبر من العدد الموجود في قائمة تخصيص القيم الابتدائية ، فان العناصر المتبقية يتم وضعها أصفار .

مثال:

هذه المصفوفة لها 6 عناصر ، بينما قائمة تخصيص القيم الابتدائية تحوي عنصرين فقط :

```
main ( )
 {
    double  a [6] =  { 65.8 , 98.4 } ;
    for ( int i = 0 ; i < 6 ; i + + )
    cout << "a[" << i << "] = " << a[i ] << endl ;
  }
```

سيكون الناتج (الإخراج) كالأتي :

```
a [0] = 65.8
a [1] = 98.4
a [2] = 0.0
a [3] = 0.0
a [4] = 0.0
a [5] = 0.0
```

إذا كان الإعلان عن المصفوفة لا يحتوي تخصيص له فان جميع عناصر المصفوفة تأخذ قيماً غير متوقعة أو قيم عشوائية .

ملاحظة : عندما يتم تخصيص قيماً ابتدائية للمصفوفة فان الإعلان عن حجمها يمكن إهماله من أمر الإعلان . كما في المثال التالي :

```
double  a[4] = {22.2 , 33.3 , 44.4 , 55.5 }  ;
```

```
double  a[ ]  = { 22.2 ,33.3 , 44.4 , 55.5 }  ;
```

يكافئ التعريف

حيث إن حجم المصفوفة في الحالة الثانية سيحدد بعدد القيم الموجودة في قائمة تخصيص القيم الابتدائية .

المصفوفة ثنائية البعد : Two Dimensional Array :

المصفوفة ثنائية البعد نوع من البيانات المهيكلة التي تتكون من مجموعة من العناصر ضمن صفوف واعمدة ، يمكن الوصول الى اي عنصر من عناصر المصفوفة عن طريق رقم الصف ورقم العامود والذي يعتبر دليل (index) العنصر في المصفوفة ، والصيغة العامة لتعريف المصفوفة ذات البعدين هي كالاتي :

Type_specified array_name [row_size] [column_size] ;

Type_specified : يحدد نوع البيانات التي سوف تخزن داخل المصفوفة .

array_name : يحدد اسم المصفوفة .

row_size : عدد الصفوف في المصفوفة .

column_size : عدد الاعمدة في المصفوفة .

هناك طريقتين لادخال القيم الى المصفوفة :

هناك طريقتين لادخال القيم الى المصفوفة :

- اعطاء قيم اولية للمصفوفة عند تعريف المصفوفة كما في المثال التالي :

```
# include<iostream.h>
 main( )
  {
 int x[3][4] = { 12,14,25,36,47,58,69,85,74,96,52,41} ;
```

إعطاء قيم أولية للمصفوفة

وتتم طباعة المصفوفة كالتالي :

```
   for(int i = 0 ; i < 3 ; i+ +)
{
cout << endl   ;
for( int j = 0 ; j < 4 ; j + +) ;
cout << x [i][j] <<"\ n " ;

}
}
```

الحلقة الخارجية لقراءة الصفوف

فتح اقواس كبيرة الحلقة الخارجية لاحتوائها على اكثر من جملة

البداية بسطر جديد عند اكمال كل صف

الحلقة الداخلية لقراءة الاعمدة

طباعة عناصر المصفوفة وترك مسافة بين العناصر عند الطبع

- قراءة المصفوفة عند تنفيذ البرنامج وادخال القيم من لوحة المفاتيح كما في المثال التالي:

```
# include < iostream.h>
main ( )
{
 int  y [4][5] ;
 for(int  i = 0 ; i < 4 ; i + + )
 for ( int j = 0 ; j < 5 ; j + + )
 cin>> y [i][j] ;
```

وتتم طباعة المصفوفة كالتالي :

```
 for(int i = 0 ; i < 4 ; i+ +)
 {
  cout << endl ;
  for( int j = 0 ; j < 5 ; j + + ; )
   cout << y [i][j] <<"\ n " ;
 }
```

الحلقة الخارجية لقراءة الصفوف

فتح اقواس كبيرة الحلقة الخارجية لاحتوائها على اكتر من جملة

البداية بسطر جديد عند اكمال كل صف

الحلقة الداخلية لقراءة الأعمدة

طباعة عناصر المصفوفة وترك مسافة بين العناصر عند الطبع

الدوال Function :

هي عبارة عن برنامج فرعي يؤدي مهمة محددة أو جزء من المهمة الكاملة التي يؤديها البرنامج العام للدالة الرئيسية . إن الدالة الرئيسية في البرنامج في لغة C ++ هي main( ) .

الصيغة العامة لتعريف أي دالة :

```
return_type   function_name (parameters list  and its declaration )
      {
       declaration  of  local variable  ;

       body of the function  ;

       return (return_value)  ;
        }
```

return_type: تحدد نوع القيمة التي ترجعها الدالة عند استدعائها من خلال استخدام جملة

return (return_value) : والتي تحدد أي نوع من البيانات التي سيتم إرجاعه وإذا لم يحدد فان القيمة المرجعية من قبل الدالة تكون من نوع integer ، وإذا لم تحتوي الدالة على جملة return (return_value) يجب إن يكون نوع الإرجاع للدالة هو void (أي خالي).

parameter list : قائمة بأسماء المتغيرات وأنواعها (تعريفاتها) ، هذه المتغيرات سوف تستلم قيم عند استدعاء الدالة وتفصل هذه المتغيرات بواسطة الفارزة ، ويمكن عدم كتابة هذه المتغيرات اذا كانت الدالة لا تستل قيمة أي بالإمكان أن تبقى الأقواس فارغة .

body of function : يحتوي على الجمل التي تنفذ العملية المطلوبة من الدالة .

return (return_value) : كل الدوال ماعدا الدوال التي من نوع void تعمل على إرجاع قيمة وهذه القيمة تحدد بواسطة جملة return أما في حالة عدم احتواء الدالة على جملة return فيجب أن يكون قد عرف إرجاع الدالة من نوع void أي الدالة لا ترجع قيمة أو نكتب return (0) .

استدعاء الدالة:

استدعاء الدالة بالقيمة: اي إن البارومترات (parameters) تنقل (ترسل) بالقيمة . هذا يعني ان التعبير المستخدم في نداء الدالة يقدر أولا ثم تخصص القيمة الناتجة للبارومتر المقابل في قائمة بارومترات الدالة قبل بدء تنفيذ الدالة . على سبيل المثال في نداء الدالة cube(x) اذا كانت قيمة x هي 4 فان القيمة 4 تنقل إلى المتغير المحلي n قبل بدء تنفيذ الدالة . حيث إن القيمة 4 تستخدم فقط داخل الدالة فان المتغير x لا يتأثر بالدالة ، لذلك فان المتغير x هو بارومتر قابل للقراءة فقط read- only .

طريقة الإرسال تسمح باستخدام التعبيرات العامة بدلا من البارومتر الحقيقي في نداء الدالة . على سبيل المثال الدالة cube يمكن أيضا إن تستدعى ب cube(3) أو cube(2*x-3) او cube(2*sqrt(x)-cube(3)) .

في كل حالة من التعابير بين الأقواس تقدر بقيمة واحدة تم ترسل إلى الدالة .

استدعاء الدالة بالمرجع : طريقة الإرسال بالقيمة عادة هي المطلوبة في الدوال . فهي تجعل الدالة أكثر استقلالاً ومحفوظة من أي أخطاء غير مقصودة . في بعض الحالات تحتاج الدالة إن تغير قيمة البارومتر المنقول إليها ، هذا يمكن عمله بواسطة الإرسال بالمرجع reference .

لنقل البارومتر بالمرجع بدلا من النقل بالقيمة ببساطة الحق العلامة & بنوع البارومتر في قائمة بارومترات الدالة . هذا يجعل المتغير المحلي مؤشراً للبارومتر الحقيقي المنقول إليه . لذلك فان البارومتر الحقيقي قابل للقراءة والكتابة بدلاً من القراءة فقط .عندئذ أي تغيير للمتغير المحلي داخل الدالة سوف يسبب نفس التغيير للبارومتر الحقيقي الذي انتقل إليه .

لاحظ أن البارومترات المنقولة بالقيمة تسمى بارومترات ذات قيمة (Parameters value) والبارومترات المنقولة بالمرجع تسمى معاملات مرجعية (reference parameters).

مثال: هذا المثال يبين الفرق بين الإرسال بالقيمة والإرسال بمرجع .

```
# include <iostream.h>
 void  faw(int  x ,int &y)
     {
        x = x*2 ;
        y = y*2 ;
      }
main( )
   {
   int a = 22 , b = 33 ;
   cout << " a = " << a <<  " b = " <<  b << endl ;
   faw(a,b)
   cout<< "a = " << a << " b = " << b << endl ;
   }
```

الدوال الرياضية :

تحتوي مكتبة C++ على مجموعة من الدوال المعرفة مسبقاً(جاهزة) وهي موجودة في ملف الرأس <math.h> وأدناه بعض من هذه الدوال الشائعة الاستخدام :

الدالة	الوصف	مثال
ceil (x)	أي تدويره نحو الأعلى x سقف الرقم	ceil(3.547) ترجع 4
floor(x)	أرضية الرقم x أي تدويره نحو الأسفل	ceil(3.547) ترجع 3
sin(x)	إيجاد جيب الزاوية x (بالتقدير الدائري)	sin(2) ترجع 0.90929
cos(x)	إيجاد جيب تمام الزاوية x (بالتقدير الدائري)	cos(2) ترجع - 0.4161
tan(x)	إيجاد ظل الزاوية x (بالتقدير الدائري)	tan(2) ترجع -2.185
pow(x,p)	حساب x للأس p	pow(2,3) ترجع 8
sqrt(x)	حساب الجذر ألتربيعي للعددx	sqrt (2) ترجع 1.4142
fabs(x)	القيمة المطلقة لـ x	fabs(-2) ترجع 2

السجلات Structure :

توفر لغة C++ إمكانية التعامل وتنظيم مجموعات من المتغيرات تسمى السجلات ، والسجل نوع من البيانات المهيكلة التي تحتوي على عناصر مختلفة الأنواع ضمن حقول متعددة ويتم التعامل مع هذه المجاميع من الحقول من خلال اسم المتغير الجماعي الذي يمثلها وتختلف هذه المجاميع عن المصفوفات انه يمكن خزن في داخل السجل اكتر من نوع واحد من البيانات ، وتستخدم بكثرة في إنشاء ومعالجة السجلات المتنوعة ، مثلا سجلات الطلبة ودرجاتهم ومعلوماتهم ، وسجلات العناوين والهواتف والبريد .

صيغة تكوين السجل:

```
structure_name    struct
    {
      type_filed      filed_name ;
      type_filed      filed_name ;
                .
                .
      } variable_name ;
```

structure_name. اسم السجل الذي يعامل معاملة نوع المتغيرات :

variable_name: اسم السجل الذي سوف نتعامل معه في البرنامج (أي المتغير الذي من نوع السجل المعلن عنه) ، ويمكن تعريفه في هذا الموضع او في أي مكان داخل البرنامج ، وفي حالة تعريفه داخا البرنامج أي داخل (main) يكون بالصيغة التالية .

```
struct   structure_name     variable_name ;
```

مثال : لتعريف سجل يحتوي على اسم الطالب وعمر الطالب ومعدله ، يتم كالأتي .

```
struct   student
        {
            char   name[20] ;
             int     age;
             float   average  ;
         }  st1 , st2 ,  st3 ;
```

في المثال اعلاه تم تعريف السجلات st1,st2,st3 من نوع student لثلاث طلاب .

ومن الممكن تعريف السجلات الثلاث في البرنامج الرئيسي كما في الصيغة التالية .

```
main( )
 {
  struct  student   st1 ,  st2 ,  st3 ;
```

يتم إعطاء القيم للمصفوفة بقراءتها بعبارة cin >> كما موضح ادناه .

```
cout<<"Enter the data of the first student";
cin>>st1.name>>st1.age>>st1.average ;
```
يتم قراءة معلومات الطالب الأول

```
cout << "Enter the data of the second student " ;
cin>>st2.name>>st2.age>>st2.average ;
```
يتم قراءة معلومات الطالب الثاني

```
cout<<"Enter the data of the third student " :
cin>>st3.name>>st3.age>>st3.average ;
```
يتم قراءة معلومات الطالب الثالث

//

طباعة معلومات الطالب الاول :

```
cout>>st1.name>>"   ">>st1.age>>"     " >>st1.average>>endl;
```
طباعة معلومات الطالب الثاني :

```
cout>>st2.name>>"   ">>st2.age>>"     " >>st2.average>>endl;
```
طباعة معلومات الطالب الثالث :

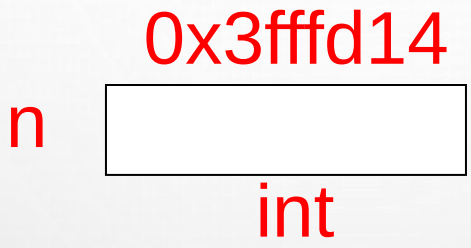
```
cout>>st2.name>>"   ">>st2.age>>"     " >>st2.average>>endl;
}
```

المؤشرات والمراجع Pointers and References:

عند الاعلان عن احد المتغيرات ترتبط به ثلاثة قرائن اساسية : اسمه ، نوعه وعنوانه بالذاكرة ، كمثال الاعلان التالي :

```
int n ;
```

. لنفترض أن n والعنوان بالذاكرة الذي يخزن به قيمة int ، والنوع n يرتبط الاسم كالتالي: n (في النظام الست عشري) عندها يمكن رؤية العنوان ox3fffd14



يمثل الصندوق مكان تخزين المتغير بالذاكرة .

اسم المتغير علي اليسار ، عنوان المتغير في الأعلى ، ونوع المتغير أسفل الصندوق . أذا كانت قيمة المتغير معروفة فتكون موضحة بداخل الصندوق :

يمكن التعامل مع قيمة المتغير بواسطة اسمه . فمثلاً يمكن طباعة قيمة المتغير n بالامر التالي :

```
cout << n ;
```

عنوان المتغير يمكن التعامل معه بعامل العنوان (address operator) & فمثلاً يمكن طباعة عنوان المتغير بالأمر التالي :

```
cout << &n
```

عامل العنوان & يعمل على اسم المتغير لينتج العنوان . إن لها أسبقية تنفيذ نفس أسبقية عامل النفي المنطقي ~ وعامل الزيادة المسبقة ++ .

مثال : طبع قيم المؤشر :

يبين هذا المثال كيف نطبع كلا من القيمة والعنوان لمتغير ،

```
main( )
 {
 int n = 33;
 cout << " n = " << n << endl ;  // print the value of  n
 cout << " &n = " << &n << end  // print the address of n
 }
```

سوف يكون الإخراج المتوقع لهذا البرنامج كالتالي :

```
  n = 33
 &n = ox3fffd14
```

يمكننا القول أن السطر الثاني من الاخراجات :

0x3fffd14

هو عنوان يسبقه العلامة 0x وهذا يعني أن العدد الذي يأتي بعد هذه العلامة هو عدد بالنظام السادس عشر . إن إظهار العنوان بهذه الطريقة ليست له فائدة كبيرة ، عامل العنوان له استعمالات أخرى أكثر أهمية في الدالة ، وذلك انه يرمز أو يشير إلى الثوابت المرجعية في إعلان الدالة ، هذا الاستخدام يقترب كثير من استخدام آخر وهو إعلان المتغيرات المرجعية .

المراجع References :

المراجع هو اسم مرادف لمتغير أخر . يعلن عنها باستخدام المعامل المرجعي & والذي يلحق بنوع المرجع .

مثال استخدام المراجع : في هذا المثال يتم الإعلان عن r على إنها مرجع n :

```
main( )
 {
    int  m = 33 ;
    int & r = n : // r is a reference  for n
    cout << " n = " << n << " , r = " << r << endl ;
    -- n ;
     cout << " n = " << n << " , r = " << r << endl ;
    r *=2 ;
    cout << " n = " << n << " , r = " << r << endl ;
 }
```

سيكون الإخراج كالتالي :

```
n = 33 , r = 33
n= 32 , r = 32
n= 64 , r = 64
```

في البرنامج أعلاه n و r هما اسمان مختلفان لنفس المتغير ودائماً لهم نفس القيمة . حيث إن إنقاص قيمة n يغير قيمة كل من n و r إلى 32 ، ومضاعفة r يزيد كل من n ، r إلى 64 .

إن القيمة 33 تخزن مرة واحدة فقط فان n , r هما أسماء رمزية لنفس المكان في الذاكرة ، وبما إن المرجع هو مرادف والمرادف لابد من شئ ينسب إليه بمعنى أن المرجع لابد من شئ يرجع إليه ، وخلاصة القول إن المرجع هي أسماء بديلة لمتغيرات أخرى .

ان العلامة & لها استخدامات عديدة في C + + عندما تتقدم اسم المتغير فانها تعيد عنوان المتغير . عند استعمالها بعد النوع في إعلان المتغير فإنها تعلن هذا المتغير على انه مرادف للمتغير التي حددت قيمته . وعند استعمالها بعد النوع في إعلان معاملات الدالة فإنها تعلن المعامل ليكون معامل مرجعي للمتغير المرسل لها ، كل هذه الاستخدامات تمثل اختلافات لنفس النمط : ان العلامة & تشير الى العنوان الذي تخزن فيه القيمة .

المؤشرات Pointer :

العامل المرجعي " & " يعيد عنوان المتغير في الذاكرة . لطباعة العنوان . يمكن أيضاً تخزين العنوان في متغير اخر . نوع المتغير الذي يخزن العنوان يسمى المؤشر . اذا كان المتغير من النوع int فان متغير المؤشر لابد ان يكون له نفس النوع أي مؤشر لرقم صحيح

int * ; ويرمز له " pointer to int "

مثال : قيم المؤشرات هي عناوين .

```
main ( )
{
int  n  =33 ;
int  * p = &n ;      // p   holds the address of   n
cout << " n = " << n << " , &n = " << & n << " , p = " << p << endl ;
cout << " & p = " << & p << endl ;
}
```

يكون إخراج البرنامج أعلاه كالتالي :

```
n = 33 , &n = ox3fffd14 , p = ox3fffd14
&p = ox3fffd10
```

ان متغير المؤشر p والتعبير &n لهما نفس النوع (مؤشر الى int) ونفس القيمة (0x3fffd14) . هذه القيمة مخزونة في المكان 0x3fffd10 بالذاكرة .

يطلق على المتغير p " مؤشر " لان قيمته تشير الى موضع قيمة اخرى . انه من النوع int لان القيمة التي يشير اليها هي int .

قيمة المؤشر هي عنوان ، هذا العنوان يعتمد على حالة الحاسبة الذي ينفذ عليها البرنامج . في معظم الحالات تكون القيمة الفعلية لهذا العنوان (مثل 0x3fffd14) غير مهمة للمبرمج .

غالبا سنحتاج الى استعمال المؤشر p وحده للحصول على القيمة التي يشير اليها ويطلق على ذلك " اعادة المرجعية للمؤشر " . وتتم ببساطة عن طريق تطبيق النجمة * كمؤشر على المؤشر .كما في المثال التالي .

Dereferencing a Pointer مثال: اعادة المرجعية لمؤشر

تشير p هنا الى الرقم الصحيح المسمى n ولذا p* و n لهما نفس القيمة

```
main ( )
    {
       int  n  = 33 ;
       int * p = &n ; // p  pointer to n
       cout << " *p = " << *p << endl ;
     }
```

إخراج البرنامج سيكون كالتالي : *p =
33

وهذا يبين ان p* هي مرادف لـ n .

عامل العنوان & وعامل اعادة المرجعية * هما عكس بعضهما n = *p

عندما p = &n ويمكن التعبير ايضاً عن ذلك بـ n = *&n أيضاً

p = &*p

"المرجعية" هي عكس" إعادة المرجعية " :

هنا p تشير الى الرقم الصحيح المسمى n و r هو مرجع تحددت قيمته بالمكان الذي تشير اليه p . ولهذا p هي مرجع لـ n و r إعادة مرجعية p . إذن r هي مرادف لـ n . بمعنى أنهما اسمان مختلفان لنفس القيمة 33

```
main( )
{
 int  n = 33 ;
 int * p = &n ;    // p  pointers to  n
 int & r = * p ;   //  r  is a reference for  n
 cout << " r = " << r << endl ;
 }
```

سيكون الإخراج: r = 33

الرسم البياني في لغة C :

إن دوال الرسم البياني توجد في الملف ألدليلي <graphics> ويتم استخدام الدالة initgraph( ) لغرض التعرف على مشغل الرسومات البيانية(نوع الشاشة) graphic drive وعلى نمط (طراز) الرسومات البيانية graphic mode وتحدد الدالة كذلك مسار تواجد برنامج التربو سي (tc) في الحاسبة . ويتم ذلك كما يلي :

```
include < graphics.h >
main ( )
{
int  gd = DETECT ;
```

يفتح الملف لغرض دوال الرسم

يقوم هذا السطر بتعريف نوع الشاشة

DETECT: الكلمة تعني أن تتحرى الدالة نوع الشاشة المستعملة في النظام

```
int  gm ;
initgraph( & gd , & gm , " c: \ \ tc " ) ;
```

يقوم هذا السطر بتعريف النمط (الطراز)

هذه الدالة تقوم بتهيئة شاشة الرسومات البيانية

قائمة بدوال الرسم الشائعة الاستخدام :

توجد هذه الدوال في الملف ألدليلي <graphics.h>

1. setcolor (x) ; x : قيمة عددية تمثل رقم اللون

هذه الدالة تقوم بتعيين اللون المستخدم في الرسم .

.....

2. setbkcolor (x); x : قيمة عددية تمثل رقم اللون

هذه الدالة تقوم بتعيين لون أرضية الرسم .

.....

3. putpixel (x ,y , color) ;

x : قيمة عددية تمثل المحور السيني .

y : قيمة عددية تمثل المحور الصادي .

color : قيمة عددية تمثل اللون .

هذه الدالة تقوم برسم نقطة في الموقع الذي إحداثياته (x ,y)ويكون لون النقطة بالقيمة العددية المتمثلة بـ (color) .

4. line (x1,y1,x2,y2) ;

هذه الدالة تقوم برسم خط يبدأ بالموقع (x1,y1) وينتهي بالموقع (x2 ,y2) . x و y تمثل الإحداثيات .

.....

5. circle (x ,y , r) ;

هذه الدالة ترسم دائرة يقع مركزها في النقطة التي إحداثياتها (x , y) ونصف قطرها طوله r (عدد صحيح) .

.....

6. rectangle (leftx , topy , rightx , bottomy);

تقوم هذه الدالة برسم مستطيل إحداثياته (leftx,topy) تمثل الزاوية العليا اليسرى من المستطيل ، والاحداثيات (rightx , bottomy) تمثل الزاوية السفلى اليمنى من المستطيل .

جدول بأرقام الألوان في شاشات الرسم ذات النمط المؤلف من (16) لون:-

العدد	اللون	العدد	اللون
0	أسود	8	رمادي غامق
1	ازرق غامق(نيلي)	9	ازرق فاتح
2	أخضر	10	اخضر فاتح
3	أزرق	11	سمائي(نيلي فاتح)
4	أحمر	12	وردي(احمر فاتح)
5	وردي(ارجواني فاتح)	13	اصفر فاتح
6	بني غامق	14	ارجواني
7	رمادي فاتح	15	ابيض