Ministry of Higher Education and Scientific Research

Al-Mustaqbal University College

Computer Engineering Techniques Department

C/C++ programming language

First year

Lecture (2)

Prepared By

MSc. Zaid Ibrahim Rasool

2020-2021

1.8.2 العمليات العلائقية (Relational Operators)

وتستخدم هذه العوامل لغرض المقارنة, وهي ست عمليات تستخدم على اي زوج من العناصر ويكون ناتجها اما صحيح(true)يعني اي قيمة ما عدى(0)او خطا(false) يعني تكون القيمة(0). ويمكن تمثيل ذلك من خلال الجدول التالي :

المؤثر	معناه	مثال	النتيجة
= =	Equal to	7= =5	False (0)
!=	Not equal to	3!=2	True (1)
<	Less than	4<9	True (1)
< =	Less than or equal to	7< =6	False (0)
>	Greater than	-2>4	False (0)
> =	Greater than or equal to	12> =10	True (1)

1.8.3 العمليات المنطقية (Logical Operators)

وهي عبارة عن عمليات ينتج عنها اما قيمة صحيح true او قيمة خطا false ويكثر استخدام التعابير المنطقية في الجمل الشرطية كمثال على ذلك
التعبير المنطقي : x==y هو أما true او false .
والتعبير المنطقي : matrix> 100 هو أما true او false .
الجدول التالي يبين لنا الادوات المنطقية :

العمليات	معناه
&&	And
||	Or
!	negating or anti-thesis

جدول الصدق: جدول الصدق:

a	b	a && b	a||b	!a
True	True	True	True	False
True	False	False	True	
False	True	False	True	True
False	False	False	False	

Example

Suppose that int a=b=3;

فان التعبير a<3 النتيجة تكون false اي 0

التعبير a<=3 النتيجة تكون true اي 1

التعبير a>b النتيجة تكون false اي 0

التعبير a!=b النتيجة false اي 0

التعبير a=b النتيجة تكون true اي 1

1.8.4 العمليات المركبة (compound operators)

وهي استخدام المساوات مع العمليات الاخرى

(+ =, - =, * =, / =, % =, > = ,< = ,& =, | =)

تتميز لغة C++ باستخدام العمليات الحسابية مع اشارة التخصيص (=) تحت اسم العمليات المركبة.

فمثلا التعبير التالي X=X+9 تعني اضافة قيمة للمتغير القديم X الموجود في الطرف الايمن, ثم تخصيص هذه القيمة للمتغير الجديد الموجود في الطرف الايسر وهو X .

فتصبح طريقة العملية المركبة =+ وكما يلي :

X+=9

الجدول التالي يوضح استخدام المساواة المركبة:

التعبير	المكافئ له
value += increase;	value = value + increase;
a -= 5;	a = a - 5;
a /= b;	a = a / b;
price *= units + 1;	price = price * (units + 1);

1.8.5 الفاصلة (,) كأداة (The Comma operator)

وهي عبارة عن ادات ثنائية (binary) تحتل الاسبقية الاخيرة في سلم الاسبقيات من الادوات المختلفة , صيغتها العامة :

Expretion1, Expretion2

تسلسل العمليات يكون على النحو التالي :

1- نستخرج قيمة التعبير الاول الذي هو على يسار الفاصلة ثم تستند الى التعبير الثاني الذي هو على اليمين .
2- نستخرج قيمة التعبير الثاني الذي هو على يمين الفاصلة كقيمة نهائية لكامل التعبير.

Example

A= (B = 2 , B+1);

في هذا المثال يقوم ال (Compiler) بالعمل على يمين الفاصلة كما هو متعارف عليه, يقوم باسناد القيمة (2) الى المتغير (B) (يبدا اولا بالتعبير الذي على يسار الفاصلة).

في هذه الحالة فان قيمة (B) هي 2 ومنها يستخرج القيمة النهائية للتعبير B+1 لتكون النتيجة هي 3 وهي تمثل نتيجة التعبيرين **B = 2 , B+1** والتي في النهاية سوف تستند الى القيمة (A)

.

1.9 عمليات الزيادة والنقصان (increment and decrement operators)

في بعض من التطبيقات هنالك استعمال للعدادات لاغراض محددة وهي عادة تبدا بالرقم (0) او اي رقم اخر وتزداد بمقدار واحد او اكثر في كل مرة وتكتب كما يلي :

count = count + 1;

ونضرا لان هذا العامل واسع الانتشار لذلك تميزت لغة C++ بتوفير عامل مفرد للاختصار لهذا الغرض وهو (++) لاغراض الزيادة بمقدار واحد وعامل (--) لاغراض النقصان بمقدار واحد , حيث يستخدم هذا العامل بطريقتين ياما ان يسبق المتغير مثلا:

(Z++) او ان يلي المتغير (++Z) وهما ليسا متشابهين فكل منهما لهما المعنى الخاص, فعندما يسبق المتغير عامل الزيادة فان المتغير تزداد قيمته بمقدار واحد ثم يستخدم اما اذا جاء عامل الزيادة بعد المتغير فان المتغير يستخدم حسب قيمته الحالية وبعدها يزداد بمقدار واحد.

اما العامل (--) فتعمل بالطريقة نفسها التي تعمل بها عامل الزيادة اي قبل وبعد المتغير مع اختلاف ان استخدامها يقلل قيمة المتغير بمقدار واحد.

Example

Suppose that (b=7) (a=2) find (c) value in the following expression:

C = **a * ++b ;**

تكون قيمتها (16), حيث ان المترجم سيقوم بزيادة قيمة (b) لتصبح (8) ثم يعوض عنها في التعبير ويحسب نتيجة التعبير, اما قيمتها في التعبير التالي:

C = a* b++ ;

فتكون (14) حيث ان ال compiler يقوم باستخدام القيمة الحقيقية للمتغير (b) ثم يقوم بحساب نتيجة التعبير وبعد ذلك تتم زيادة قيمة المتغير (b) لتكون (8).

التعبير التالي :

C = a* --b ;

هنا قيمة (c) تكون (12) حيث يقوم ال compiler بانقاص قيمة (b) بواحد لتكون (6) ثم تعوض قيمتها في التعبير لايجاد قيمة (c).

اما في التعبير التالي :

C = a* b-- ;

تكون (14) تستخدم قيمة (b) الحقيقة (7) لايجاد قيمة (c)بعدها تقلل قيمة (b) لتكون قيمتها (6).

Example

A=3;

B=++A;

//B contains 4

//A contains 4

Example

A=3;

B=A++;

// B contains 3 // A contains 4

Example

B=C=3;

A=(++B+C++);

// B contains 4, C contains 3

// A contains 7

Example

B=C=3;

A=(B--+C--);

// B contains 3, C contains 3

// A contains 6

1.10 العمليات الدقيقة (Bitwise Operators)

تتميز لغة C++ باستخدامها ادوات دقيقة تعمل على مستوى وحدة التخزين الاولية (bit) وسميت هذه الادوات بالدقيقة لانها تتعامل مع البت بشكل مباشر وتستعمل هذه الادوات مع البيانات الصحيحة (int) والرمزية (char) فقط. ولاتستعمل مع غيرها من البيانات, الجدول التالي يبين لنا العمليات الدقيقة وعمل كل واحد منها:

العوامل الدقيقة	استخدامها	العمليات الرياضية المكافئة
&	تقوم بعملية (و) بين البتات	**AND**
|	تقوم بعملية (او) بين البتات	**OR**
^	تقوم بعملية (XOR) بين البتات	**XOR**
~	عكس قيمة البت	**NOT**
<<	ازاحة البتات الى اليسار	**SHL**
>>	ازاحة البتات الى اليمين	**SHR**

في ما يلي توضيح كل عملية من عمليات ال **Bitwise Operators**

1.10.1 اداة النفي (~) Bitwise Not

واحد من العمليات التي من الممكن ان تنجزها على البت تتمثل بعكس قيمته.عمل هذه الاداة هو ابدال قيمة الصفر(0) الى واحد(1) وقيمة الواحد(1) الى الصفر(1), الجدول التالي يبين عمل الاداة:

Bit	~Bit
1	0
0	1

مثال على ذلك نفرض عندنا قيمة x=8 ممثلة بالنظام العددي الثنائي (من 8بت)

قيمة x

0	0	0	1	1	0	1	0

قيمة ~x

1	1	1	0	0	1	0	1

معنا النفي (not) هو التضاد بين ال(1) وبين (0) في النظام العددي الثنائي, اذ تم النفي لقيمة x بالبت ليصبح ~x في جميع مكونات البت.

1.10.2 اداة المقارنة (&) Comparing Bits: The Bitwise AND Operator &

هذا العامل ياخذ قيمتين ويقوم بعملية المقارنة للبت في القيمة الاولى مع البت في القيمة الثانية, تكون النتيجة وفقا لجدول الصدق المبين ادناه:

Bit1	Bit2	Bit1 & Bit2
1	1	1
1	0	0
0	1	0
0	0	0

في المثال التالي توضيح لعملية المقارنة باستخدام (&) عند تمثيلها بالنظام العددي الثنائي :

العملية (x&y)

قيمة x بالنظام الثنائي

0	0	0	1	1	0	1	0

قيمة y بالنظام الثنائي

0	0	0	0	1	0	0	1

النتاج ل (x&y)

0	0	0	0	1	0	0	0

معنى ذلك ان كل عملية مقارنة بين اي قيمتين تعطي نتيجة (0) ما عدا عملية الجمع بين 1+1 تعطي نتيجة (1). لاحظ البرنامج التالي:

Example

```
#include <iostream.h>
main(){
const int N1 = 187;
const int N2 = 242;
cout<<N1<<"&"<<N2<<"="<< (N1 & N2)<<"\n\n";
}
```

مخرجات البرنامج:

```
187 & 242 = 178
```

من خلال البرنامج هذا البرنامج يتبين انه لدينا قيمتين الاولى هي 187 و الانية 242 , فان القيمة الثنائية للعدد العشري 187 هي (1011 1011) والقيمة الثنائية للعدد العشري 242 هي (0010 1111) سوف نقوم بمقارنة هاتين القيمتين بت بت , باستخدام عامل البتات ~ AND .

	ثنائي								عشري
N1	1	0	1	1	1	0	1	1	187
N2	1	1	1	1	0	0	1	0	242
N1 & N2	1	0	1	1	0	0	1	0	178

في كثير من الاحيان يقوم المترجم (compiler) بانجاز هذه العملية واستخدام الناتج في البرنامج, وعرض النتيجة على شاشة الحاسوب. كما وضحنا في المثال السابق.

1.10.3 اداة الاختيار (|) Comparing Bit: Bitwise OR Operator

من الممكن ان تقوم بنوع اخر من المقارنة على البتات باستخدام عامل (|), حيث يقوم ال (compiler) بمقارنة البتات المتقابلة للقيمتين (x,y) فاذا كان على الاقل واحد من البتات يساوي 1 فان نتيجة المقارنة ستكون 1. نتيجة المقارنة ستكون 0 اذا كان البتان قيمتهما 0.

جدول الصدق لاداة (|)

Bit1	Bit2	Bit1 | Bit2
1	1	1
1	0	1
0	1	1
0	0	0

يمكن ملاحضة ذلك في المثال الاتي:

Example قيمة x

0	0	0	1	1	0	1	0

قيمة y

0	0	0	0	1	0	0	1

الناتج ل (x|y)

0	0	0	1	1	0	1	1

حيث الاختيار بين ال 0 و 1 هو 1 والاختيار بين 1 و 1 هو 1 وبين 0 و 0 هو 0.

1.10.4 اداة الاختيار الاستثنائي (^) Comparing Bits: The Bitwise-Exclusive XOR Operator

جدول الصدق لاداة XOR (^):

Bit1	Bit2	Bit2^Bit1
1	1	0
1	0	1
0	1	1
0	0	0

بنفس المثال السابق للتعرف على كيفية عمل الاداة :

Example قيمة **x**

0	0	0	1	1	0	1	0

قيمة y

0	0	0	0	1	0	0	1

الناتج ل (x^y)

0	0	0	1	0	0	1	1

لاحظ الاختيار بين 0 و 0 هو 0 , و الاختيار بين 1 و 0 هو 1, و الاختيار بين 1 و 1 هو 0, ان اداة الاختيار الاستثنائي ^ (XOR) تختلف عن ادات الاختيار | (OR) اذ تكون نتيجتها true عندما يكون احد طرفي الاختيار x,y هو true وماعدا ذلك فان النتيجة تكون دائما false.

لاتنسى بان قيمة true يقصد بيها القيمة =1 و false يقصد بالقيمة =0.

1.10.5 ادوات الازاحة (<< و >>) Bit Shift Operators

يقصد بالازاحة باتجاه اليمين او اتجاه اليسار حيث ينتج بازاحة قيمة المتغير الصحيح بالنظام الثنائي (بالبت) عدد من الخانات حسب الطلب, بحيث تملى عدد الخانات المفرغة من الجهة الموجبة اصفارا و من الجهة السالبة تملى احادا.

المثال التالي يوضح طريقة استعمال هذاه الاداة

Example

الجملة ; X>>2 عند تنفيذها على قيمة x وهي (23) بالنظام العشري, فان العملية تتم على النحو الاتي:

قيمة x

0	0	0	1	0	1	1	1

ناتج X>>2

0	0	0	0	0	1	0	1

تمت عملية الازاحة بمقدار (2بت) من جهة اليمين تصبح قيمتها :

5 بالنظام العشري, وهذا يعني ان 23>>2 تعطي النتيجة 5

لاحظ الخانتين الممفرغتين بسبب الازاحة الى اليمين قد اصبحت قيمها (0).

الاولوية في التنفيذ تتضح عند وجود تعبير به عدد من المؤثرات المختلفة, فاذا اردنا ان ياخذ التعبير مسار محدد لتقيمه يجب استخدام الاقواس وذلك حسب مايقتضي التعبير, اما الاولوية فتتم في الجدول التالي :

التسلسل	العمليات	الوصف	التنفيذ
1	++ -- ~ !	unary (prefix)	Right-to-left
2	<< >>	shift	Left-to-right
3	< <= > >=	relational	Left-to-right
4	!= ==	equality	Left-to-right
5	&	bitwise AND	Left-to-right
6	^	bitwise XOR	Left-to-right
7	|	bitwise OR	Left-to-right
8	&&	logical AND	Left-to-right
9	||	logical OR	Left-to-right
10	,	comma	Left-to-right