الفصل الثاني

لغة ++C

هدف الفصل

يهدف هذا الفصل إلى تحصيل المعارف التالية:

- لمحة عن لغة **++C** وعلاقتها بلغة **C**
- البنية العامة لبرنامج بلغة **++C**
- البرمجة البنيوية بلغة **++C**
  - الأنماط في **++C**
  - التعليمات في **++C**
  - الأقنية في **++C**

مدخل

C++ هي لغة برمجة حديثة، لذا، فهي تعكس معظم الإضافات التي ظهرت في نظريات البرمجة. وهي من ثَم لغة غنية تعطي المبرمج إمكانات هائلة. يربط الكثير من الناس بين البرمجة الغرضية التوجه ولغة **C++**، و هذا خطأ. إذ يمكننا البرمجة بـ **C++** دون أي ذكر للأغراض، على عكس بعض اللغات الغرضية التوجه الصرفة، مثل جافا . وبهذا فإن **C++** لغة هجينة بما لذلك من حسنات وسيئات.

لمحة تاريخية

طور **Bjarne Stroustrup** لغة البرمجة **C++** في مختبرات **AT&T Bell** في بداية الثمانينيات. وكما يقول مبتكر اللغة، فإن **C++** تأخذ الكثير من **C**، إذ احتُفظ بلغة **C** كمجموعة جزئية من **C++** للتسهيلات التي تقدمها في المستويات القريبة من الآلة، وهذا مما يسمح ببرمجة نظم التشغيل. وهكذا، فقد قام **Stroustrup** عام **1979** بتطوير لغة "**C** المزودة بالصفوف". وقد أعيد تشكيل هذه اللغة ما بين **1983/1984** فنتجت لغة **C++** التي خضعت لبعض التحسينات الطفيفة فأصبحت **C++** النسخة **1.2** عام **1987** . انتشرت هذه اللغة انتشاراً واسعاً وأصبحت متوفرة على جميع بيئات التطوير، ثم طورت اللغة من النسخة **1.2** إلى النسخ **2.0** و **2.1** و **3.0**.

في عام **1990** نشر **Stroustrup** مؤلفه (**Annotated C++ Reference Manual " ARM"**) أي مرجع لغة **C++** الذي يعتبر الأساس في تقييس اللغة.

ولغة **C++** هي تطور للغة **C** في ثلاثة مسارات أساسية جديدة وهي:

- احتواؤها على تحسينات واضحة مقارنة بلغة **C** التقليدية.
- دعمها لبناء واستكمال أنماط جديدة.
- احتواؤها على تقنيات البرمجة الغرضية التوجه.

ولعل أفضل طريقة لتعلم اللغة هي البدء بكتابة البرنامج الأول بـ **C++**، ثم سنتناول بشيء من التفصيل أهم النقاط التي تخص لغة **C++** معتمدين على أن هذا المقرر ليس المقرر الأول في

البرمجة، وبالتالي فقد سبق لك أن تعرفت المفاهيم الأساسية (متحول، ثابت، تعليمة، ....) في مقرر سابق (راجع مقرر البرمجة1).

البرنامج الأول بلغة ++C

```
// my first program in C++
#include <iostream>
using namespace std;
int main ()
{
 cout << "Hello World!";
 return 0;
}
```

```
Hello World!
```

يتضمن المستطيل الموجود إلى يسارك البرنامج، في حين يتضمن المستطيل الموجود إلى يمينك نتيجة تنفيذ البرنامج بعدما تمت ترجمته. تختاف الطريقة المستخدمة لترجمة البرنامج باختلاف المترجم المستخدم.

إيضاح:

المترجم هو البرنامج الذي يتحقق من صحة برنامجك قواعدياً بلغة البرمجة التي كتب بها. يوجد عدة مترجمات للغة **++C** ، فهناك المترجم **Microsoft visual C++** وهناك **Visual Studio.Net** وهناك المترجم **GNU C++** الذي يعمل في البيئة يونكس.

إن هذا البرنامج مثال جيد لبرنامج يكتبه طالب مبتدئ بلغة **++C**، وينتج عنه ظهور العبارة

"**Hello World!** " على الشاشة، وبالرغم من بساطة البرنامج، فهو يتضمن المكونات الأساسية التي نجدها في جميع البرامج المكتوبة بلغة **++C**. وسنتفحص هذا البرنامج سطراً سطراً.

```
// my first program in C++

#include <iostream>

using namespace std;

int main ()
{
  cout << "Hello World!";
  return 0;
}
```

هذا سطر لكتابة تعليق أو ملاحظة.

جميع الأسطر التي تبدأ بالرمز // هي أسطر ملاحظات، ولا يكون لها أي تأثير على تصرف البرنامج. يستخدم المبرمج أسطر الملاحظات ليضيف شرحاً أو ملاحظة إلى أسطر الرماز في البرنامج.

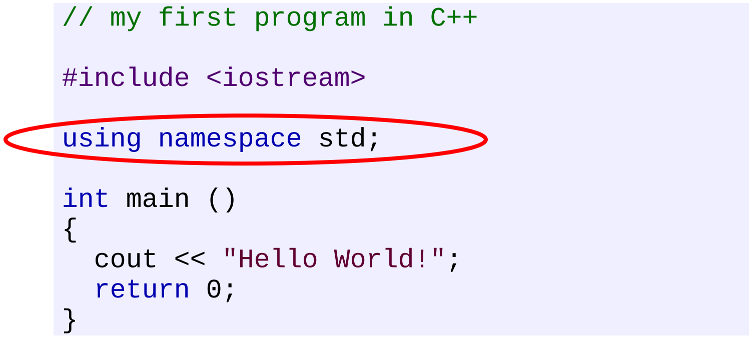
```
// my first program in C++

#include <iostream>

using namespace std;

int main ()
{
  cout << "Hello World!";
  return 0;
}
```

جميع الأسطر التي تبدأ بالرمز # تتضمن توجيهات للمترجم. وهي لا تعتبر تعليمات بالمعنى المعروف للتعليمة. في برنامجنا هذا، يطلب هذا التوجيه

#include <iostream>

من المترجم أن يضمن الملف القياسي iostream الذي يحتوي على التصريحات الخاصة بمكتبة الدخل والخرج القياسية في C++. وقد جرى تضمينها لأننا سنستخدمها لاحقاً في البرنامج

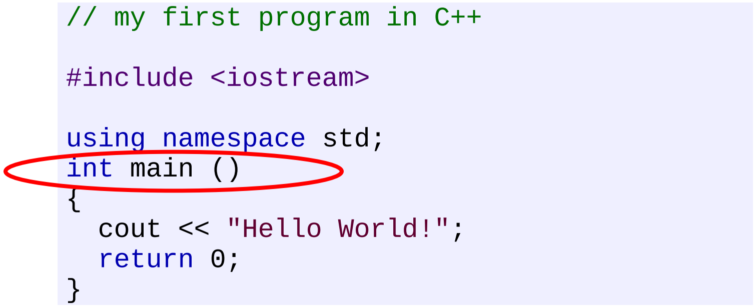
إن جميع العناصر التابعة للمكتبة القياسية في C++ يصرح عنها فيما يسمى فضاء الأسماء الذي يحمل الاسم std. وعليه، فإن هذه العبارة تدل على أننا سنستخدم هذه العناصر. غالباً ما نرى هذا السطر في برامج C++ التي تستخدم المكتبة القياسية.



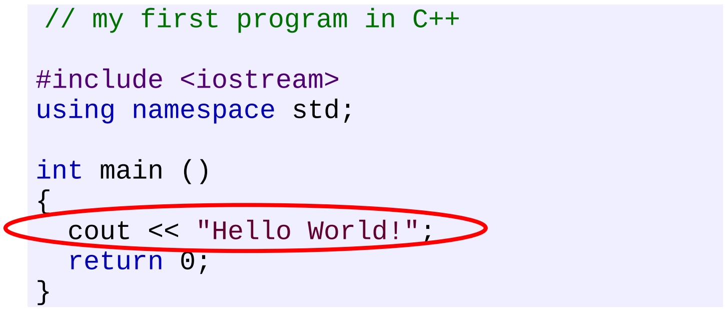
```
// my first program in C++

#include <iostream>

using namespace std;

int main ()
{
  cout << "Hello World!";
  return 0;
}
```

يدل هذا السطر على بدء تعريف التابع الرئيسي (البرنامج). ويعتبر التابع الرئيسي نقطة البدء التي يجري تنفيذ البرنامج اعتباراً منها بقطع النظر عن مكان وجودها في الرماز، فقد توجد توابع أخرى بأسماء أخرى قبل أو بعد التابع الرئيسي، لكن التنفيذ يبدأ من الأسطر الموجودة فيه. وهكذا يجب أن يحتوي كل برنامج C++ على تابع رئيسي.

نلاحظ هنا وجود زوج من الأقواس ( ) بعد كلمة main ، والسبب في ذلك يعود إلى أننا نصرح عن تابع وليس عن متحول، والقوسان يفيدان في عملية التمييز هذه. أحياناً توضع لائحة من الموسطات بين القوسين. وفوراً بعد هذين القوسين نجد جسم التابع الرئيسي محصوراً بين زوج من الأقواس الكبيرة { } أي التعليمات التي تنفذ عند تنفيذ هذا التابع.



```
// my first program in C++

#include <iostream>

using namespace std;
int main ()
{
  cout << "Hello World!";
  return 0;
}
```

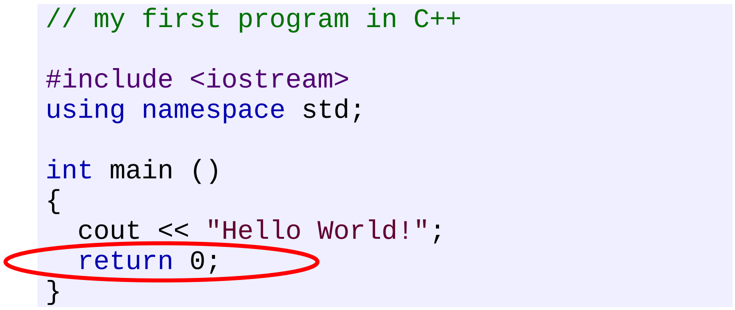


```
 // my first program in C++

#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World!";
  return 0;
}
```

يمثل هذا السطر تعليمة من تعليمات لغة C++ . والتعليمة هي صيغة بسيطة أو مركبة يؤدي تنفيذها إلى الحصول على ناتج ما. في الواقع، تقوم هذه التعليمة بالعملية الوحيدة في برنامجنا وهي الحصول على إظهار على الشاشة.

تعتبر cout قناة الخرج القياسية في C++، ويعني هذا السطر أننا نضع متتالية من المحارف (في برنامجنا الجملة Hello World!) في قناة الخرج القياسية والتي تكون عادة هي الشاشة. يحتوي الملف iostream الموجود في فضاء الأسماء std على تصريح كلمة cout، ولهذا السبب وجدت الأسطر السابقة.

لاحظ أن التعليمة تنتهي بفاصلة منقوطة، وهنا نشير إلى ضرورة وضع هذه الفاصلة المنقوطة في نهاية كل تعليمة في C++.



```
// my first program in C++

#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World!";
  return 0;
}
```

تؤدي تعليمة return إلى انتهاء البرنامج ، وقد نجد بعد هذه الكلمة رمازاً. في مثالنا، ادينا 0 وهو يفسر بأنه قد تم تنفيذ البرنامج دون أخطاء. وغالباً ما نجد هذا السطر بهذا الشكل في جميع برامج C++.

لعلك لاحظت في هذا البرنامج أن هناك بعض الأسطر التي لا تقوم بأفعال عند التنفيذ. فهناك أسطر للتعليقات (وهي الأسطر التي تبدأ بـ //)، وأسطر لتوجيه المترجم (وهي الأسطر التي تبدأ بـ #)، وأسطر للتصريح عن التوابع (في حالتنا هذه عن التابع الرئيسي). أما أسطر التعليمات (مثل تعليمة **cout**) فتكون موجودة داخل كتلة التابع الرئيسي بين القوسين الكبيرين { }.

أسلوب كتابة البرنامج

نلاحظ أنه قد تم تنظيم البرنامج السابق في أسطر متتالية بحيث تكون قراءته سهلة وواضحة، ولكن لا يوجد في **C++** قواعد صارمة لفصل التعليمات في عدة أسطر. فمثلاً، بدلاً من كتابة البرنامج على النحو التالي:

```
int main ()
{
  cout << " Hello World ";
  return 0;
}
```

يمكن أن نكتب:

```
int main () { cout << "Hello World"; return 0; }
```

ويرجع ذلك إلى أن التعليمات تنتهي بفاصلة منقوطة. وعليه يمكننا أن نكتب عدة تعليمات في سطر واحد أو نكتب تعليمة واحدة في عدة أسطر.

لنضيف سطراً جديداً إلى البرنامج:

```
// my second program in C++
#include <iostream>
using namespace std;
int main ()
{
  cout << "Hello World! ";
  cout << "I'm a C++ program";
  return 0;
}
```

```
Hello World! I'm a C++ program
```

في هذا البرنامج كتبنا تعليمتي إظهار. وكما ذكرنا آنفاً فإن كتابة كل تعليمة على سطر تهدف إلى توضيح القراءة فقط.

غير أنه من الممكن أن نكتب البرنامج على النحو التالي دون أن يؤثر ذلك على صحته:

```
int main () { cout << " Hello World! "; cout << " I'm a C++ program "; return 0; }
```

كما أننا يمكن أن نكتبه على النحو التالي إذا اعتقدنا أن ذلك أوضح للقارئ:

```

int main ()
{
  cout <<
    "Hello World!";
  cout
    << "I'm a C++ program";
  return 0;
}
```

لابد من الإشارة إلى أننا نستثني التوجيهات التي تخص المترجم، أي تلك التي تبدأ بالرمز#، من إمكان كتابتها في أكثر من سطر، لأنها ليست تعليمات بل هي توجيهات للمترجم يعالجها بطريقة مختلفة ولا تولد بذاتها أي رماز، كما أنها لا تنتهي بفاصلة منقوطة.

ملاحظة حول التعليقات

توجد طريقتان لكتابة التعليقات:

```
// line comment
/* block comment */
```

الطريقة الأولى تعرف بالتعليقات السطرية **line comment** حيث يتجاهل المترجم جميع الكلمات الواردة بعد الرمز // وحتى نهاية السطر.

الطريقة الثانية تعرف بالتعليقات الكتابة **line comment** حيث يتجاهل المترجم جميع الكلمات الواردة بعد الرمز/* وحتى الظهور الأول للرمز */ ولا يوجد ما يمنع كون هذه التعليقات في عدة أسطر.

ولإيضاح الفكرة سنضيف تعليق كتلي لبرنامجنا الثاني:

```
/* my second program in C++
   with more comments */
#include <iostream>
using namespace std;
int main ()
{
  cout  <<  "Hello  World!  ";          //
prints Hello World!
  cout   <<  "I'm   a   C++   program";  //
prints I'm a C++ program
  return 0;
}
```

```
Hello World! I'm a C++ program
```

الكلمات المحجوزة في لغة C++

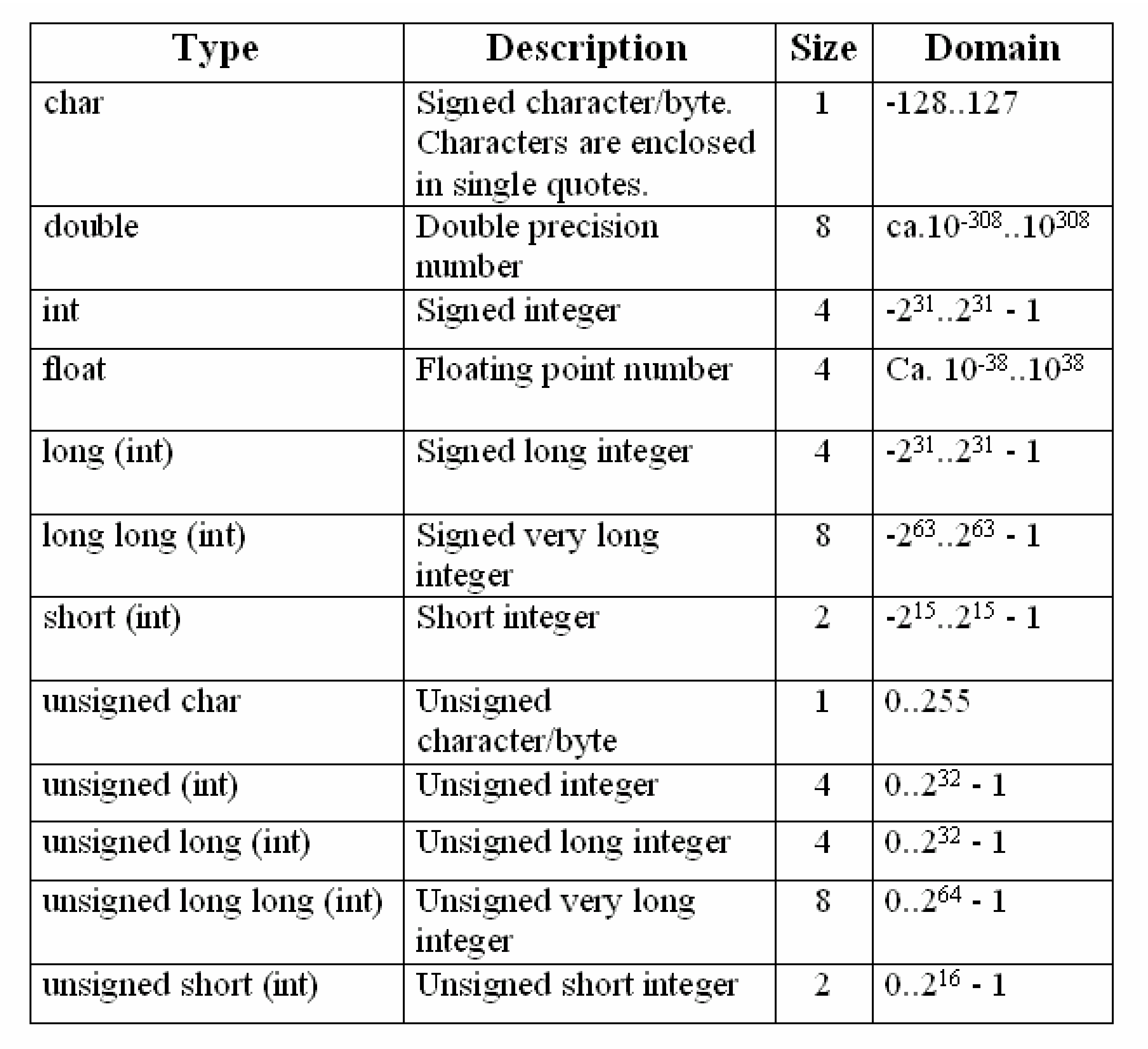
تستخدم لغة **C++** مجموعة من الكلمات المحجوزة **reserved words** نوردها في الجدول التالي:

and	and_eq	asm	auto	bitand
bitor	bool	break	case	catch
char	class	const	const_cast	continue
default	delete	do	double	dynamic_cast
else	enum	explicit	export	extern
false	float	for	friend	goto
if	inline	int	long	mutable
namespace	new	not	not_eq	operator
or	or_eq	private	protected	public
register	reinterpret_cast	return	short	signed
sizeof	static	static_cast	struct	switch
template	this	throw	true	try
typedef	typeid	typename	union	unsigned
using	virtual	void	volatile	wchar_t
while	xor	xor_eq		

المتحولات والأنماط في C++

كما رأيت في مقرر سابق للبرمجة (راجع مقرر البرمجة1)، فإن البرامج تتضمن متحولات يجري تداولها والتعامل معها في البرنامج لتخزين المعطيات ونتائج الحسابات والعمليات. وكما في الكثير من لغات البرمجة، فإنه يجب التصريح عن أي متحول قبل استخدامه.

توجد في **C++** أربعة أنماط معرفة في أصل اللغة هي: **int**، **float** ، **bool** ، **char** ، وقد اشتقت منها مجموعة من الأنماط الأخرى .



Type	Description	Size	Domain
char	Signed character/byte. Characters are enclosed in single quotes.	1	-128..127
double	Double precision number	8	ca.$10^{-308}..10^{308}$
int	Signed integer	4	$-2^{31}..2^{31} - 1$
float	Floating point number	4	Ca. $10^{-38}..10^{38}$
long (int)	Signed long integer	4	$-2^{31}..2^{31} - 1$
long long (int)	Signed very long integer	8	$-2^{63}..2^{63} - 1$
short (int)	Short integer	2	$-2^{15}..2^{15} - 1$
unsigned char	Unsigned character/byte	1	0..255
unsigned (int)	Unsigned integer	4	$0..2^{32} - 1$
unsigned long (int)	Unsigned long integer	4	$0..2^{32} - 1$
unsigned long long (int)	Unsigned very long integer	8	$0..2^{64} - 1$
unsigned short (int)	Unsigned short integer	2	$0..2^{16} - 1$

وهكذا يمكن أن نكتب التصريحات التالية:

```
int i, j, count;
float sum, product;
char ch;
bool passed_exam;
double wave_length;
unsigned char color;
long seconds;
```

ويمكن إعطاء قيمة ابتدائية للمتحول عند التصريح:

```
int i, j, count = 0;
float sum = 0.0, product;
char ch = '7';
bool passed_exam = false;
double wave_length=0.00000879;
unsigned char color=120;
long seconds=54087996;
```

التصريح عن الثوابت

في البرامج، نحتاج أحياناً إلى التصريح عن الثوابت. نصرح عن الثوابت في **C++** على النحو التالي:

```
    const type constant_identifier = value ;
```

مثال

```
const int days_in_year = 365;
const float VatRate = 17.5;
const int days_in_year = 365,
          days_in_leap_year = 366;
```

ولا يسمح بمحاولة أي تعليمة في البرنامج أن تغير قيمة الثابت.

الإدخال والإخراج في C++

تستخدم لغة **C++** قناة الدخل **(input stream)** **cin** للإدخال وقناة الخرج **(output stream)** **cout** للإخراج.

- **الإدخال باستخدام cin:**

مثال

```
cin >> number;
cin >> n1 >> n2;

             لإدخال قيمتي المتحولين n1 و n2
```

ويستدل على نهاية الإدخال عندما يضغط المستخدم الزر **enter**، وفي حال وجود عدة متحولات يطلب إدخالها، يمكن الفصل بين القيم المدخلة بالفراغ أو **enter**.

```
int count, n;
float value;
cin >> count >> value >> n;
```

قد يدخل المستثمر المعطيات على النحو التالي:

```
     23  -65.1 3
```

أو على النحو التالي:

```
23
-65.1        3
```

تستخدم لغة **C++** قناة الدخل **(input stream)** **cin** للإدخال وقناة الخرج **(output stream)** **cout** للإخراج.

- **الإخراج باستخدام cout:**

مثال

```
cout << count;
cout <<"Hello there" << endl;    لإخراج الجملة الموضوعة بين علامتي اقتباس
```

لاحظ هنا استخدام **endl**. نستخدم **endl** عندما نريد أن ننتقل للسطر جديد بعد تعليمة الإخراج تلك.

تقويم :

ما هو خرج التعليمات التالية على الشاشة إذا كان دخل المستخدم هو **6.51** ثم **3.24**:

```
float length, breadth;
cout << "Enter the length and breadth: ";
cin >> length >> breadth;
cout << endl << "The length is " << length;
cout << endl << "The breadth is " << breadth << endl;
```

الإجابة:

```
The length is 6.51
The breadth is 3.24
```

تدريب:

اكتب برنامجاً يقرأ أربعة قيم يدخلها المستخدم، ثم يطبعها كل واحدة على سطر بين علامتي اقتباس بسيطة(مثلاً ’ **7** ‘).

التعليمات في لغة ++C

إيضاح

يفترض هذا المقرر أنك قد تعرضت لمعاني ودلالات أنواع التعليمات المختلفة في مقرر سابق. وهدف هذا المقرر تعريفك شكل التعليمة في لغة **++C** وليس إعادة شرح المفاهيم الأساسية. يمكنك العودة إلى مقرر البرمجة1 إن احتجت إلى مراجعة المفاهيم.

1. تعليمات الإسناد (راجع مقرر البرمجة1)

يستخدم الرمز= للدلالة على الإسناد. والشكل العام لتعليمة الإسناد البسيطة هو:

```
result = expression ;
```

مثال

```
            average = (a + b)/2;
```

يمكن في الصيغ الحسابية الموجودة إلى يمين الرمز = أن نجد الرموز التالية:

+ للجمع

- للطرح

* للضرب

/ للقسمة

% باقي القسمة (modulus)

مثال

```
i = 3;
sum = 0.0;
perimeter = 2.0 * (length + breadth);
ratio = (a + b)/(c + d);
```

إن نمط المعاملات المستخدمة داخل الصيغة هام جداً لتحديد النتيجة، وهنا تراعى القواعد التالية:

- إذا كان المعاملان من النمط **int** تكون النتيجة من النمط **int** كذلك.

- إذا كان أحد المعاملين أوكلاهما من النمط **float** تكون النتيجة من النمط **float** كذلك.
- إذا كانت نتيجة الصيغة من النمط **int** وكان المتحول إلى يسار معامل الإسناد من النمط **float**، عندها يجري تحويل القيمة الناتجة من **int** إلى **float** قبل خزنها في المتحول إلى يسار معامل الإسناد.
- إذا كانت نتيجة الصيغة من النمط **float** وكان المتحول إلى يسار معامل الإسناد من النمط **int**، عندها يجري تحويل القيمة الناتجة من **float** إلى **int** قبل خزنها في المتحول إلى يسار معامل الإسناد، وهذا يعني ضياع الجزء العشري الموجود في القيمة الأصلية. ويجري هذا التحويل بقطع الجزء العشري دون إجراء أي عملية تدوير.

حسب القاعدة الأخيرة، فإن الإسناد قد يؤدي إلى فقد الدقة ما لم تراعى الأنماط على نحو دقيق.

تقويم:

ما محتوى **i** بعد تنفيذ التعليمات التالية:

```
int i;
i = 3.5 ;
```

الجواب:

3

تواجهنا المشكلة ذاتها عند استعمال معامل القسمة، حيث تقول القاعدة:

- ناتج قسمة عددين من النمط **int** يكون من النمط **int** كذلك. ويجري القطع عندما يكون حاصل القسمة عدداً عشرياً موجباً. أما في حالة الناتج السالب، فلا تحدد اللغة ما الذي يحدث. ومن هنا يجب عليك الانتباه لئلا تفقد الدقة!

مثال

```
i = 1/7 ;
```

تؤدي إلى وضع القيمة 0 في **i**.

أما معامل % لباقي القسمة فهو يعمل على نحو صحيح عندما يكون العددان موجبان، أما عندما يكون أحدهما سالباً فلا يمكن التكهن بسلوك المترجم.

مثال

```
int i;
i=34 % 10;
cout<<i;    //i=4
```

في حين لا ندري ما قيمة نتيجة التعليمة التالية:

```
int i;
i=10 % -7;
cout<<i;    //i=3 or -4
```

وكل ما يمكننا أن نضمنه أن المترجم يراعي المساواة التالية:

```
i % j = i - (i / j) * j
```

التعليمات في لغة ++C، أولويات المعاملات

تراعى الأولويات في حساب الصيغ على النحو التالي:

( )

* / %

+ -

=

التعليمات في لغة ++C، التحويل بين الأنماط

رأينا في الفقرة السابقة (راجع الفقرة تعليمة الإسناد) أن ناتج قسمة عددين من النمط **int** يكون من النمط **int** كذلك، فكيف نفعل للحصول على الجواب الصحيح؟ يجب أن نجبر المترجم على حساب الناتج كقيمة من النمط **float**. فإذا كان أحد المعاملين ثابتاً، يمكننا تحويله إلى عدد ثابت من نمط **float** بإضافة .0 إلى اليمين.

مثال

```
int i;
float x=1.0/i;
```

أما إذا كان المعاملان متحولين صحيحين، فلابد من تحويل أحدهما على الأقل إلى النمط **float** وذلك باستخدام عملية القسر **cast** التالية:

```
f = float(i)/float(n);
```

هنا نستخدم الكلمة **float** الدالة على النمط كمعامل لإعطاء تمثيل بالفاصلة العائمة للمتحول أو للصيغة الموضوعة بين قوسين.

على غرار هذا المعامل لدينا المعامل **int(x)** الذي يعيد قيمة **x** كقيمة صحيحة و**char(y)** الذي يعيد المحرف الموافق للقيمة **y** في ترميز **ASCII**.

تقويم:

تحويل درجة الحرارة:

في التقويم التالي يطلب منك كتابة برنامج يقرأ عدداً صحيحاً يدخله المستخدم، ويمثل درجة الحرارة درجة بالفهرنهايت، ثم يقوم بتحويلها إلى قيمتها بالسنتيغراد.

الإجابة:

```
// Convert Fahrenheit to Centigrade
// Enters a Fahrenheit value from the user,
// converts it to centigrade and outputs
// the result.

#include <iostream.h>

void main()
{
  const float mult = 5.0/9.0;  // 5/9 returns zero
                               // integer division
  const int sub = 32;
  float fahr, cent;
  cout << "Enter Fahrenheit temperature: ";
  cin >> fahr;
  cent = (fahr - sub) * mult;
  cout << "Centigrade equivalent of " << fahr
       << " is " << cent << endl;
}
```

ملاحظة: تستخدم بعض المترجمات ترويسة المكتبة التي تضمن في الملف بوضع iostream.h بدلاً من iostream

التعليمات في لغة C++، معاملا الزيادة increment والإنقاص decrement

يكثر في جميع البرامج ورود شكلين من التعليمات هما:

```
n = n + 1;
n = n – 1;
```

يوجد في **C++** معاملان هما: ++ للزيادة و **--** للإنقاص، بحيث يمكننا أن نكتب :

n++; بدلاً من **n = n + 1;** و

n - -; بدلاً من **n = n - 1;**

يسمى هذان المعاملان قي هذه الحالة معاملي الزيادة اللحقية **postincrement** والإنقاص اللحقي **postdecrement** لأنهما وردا بعد المتحول.

يوجد كذلك معاملان مماثلان هما معاملا الزيادة السبقية **preincrement** والإنقاص السبقي **predecrement** ويستعملان على النحو التالي:

n++; بدلاً من **n = n + 1 ;** و

n - -; بدلاً من **n = n - 1 ;**

لا فرق بين استعمال أي من معاملي الزيادة السبقي أو اللحقي إذا كان الهدف هو زيادة 1 للمتحول **n**. وكذلك الحال بالنسبة لمعاملي الإنقاص، غير أن لكل منهما أثره الجانبي الذي يميزه عن الآخر. لتوضيح هذه الفكرة لنأخذ المثال التالي:

```
i = n++;
```

ما قيمة **i** بعد التنفيذ؟ هل هي قيمة **n** قبل زيادة **1** أم القيمة الجديدة بعد الزيادة؟

تقول القاعدة أن معاملي الزيادة اللحقية **postincrement** والإنقاص اللحقي **postdecrement** يعطيان قيمتهما قبل زيادة أو إنقاص المتحول، في حين أن معاملي الزيادة السبقية **preincrement** والإنقاص السبقي **predecrement** ينفذان زيادة أو إنقاص المتحول أولاً ثم يعطيان القيمة الناتجة عن ذلك. على سبيل المثال، إذا كانت قيمة **n** هي **5** يكون لدينا بعد تنفيذ التعليمة التالية:

```
i = n++;
```

قيمة **i** هي **5** وقيمة **n** هي **6**. في حين يكون لدينا بعد تنفيذ التعليمة التالية:

```
i = ++n;
```

قيمة **i** هي **5** وقيمة **n** هي **6**.

التعليمات في لغة C++، تعليمات مختصرة للإسناد

توجد في **C++** تعليما ت تنفذ التعليمة الحسابية المطلوبة والإسناد معاً. فمثلاً في حالة الجمع، وبدلاً من كتابة

```
sum = sum + x;
```

يمكن أن نكتب:

```
sum += x;
```

يبين المثال التالي استعمال التعليمات المختصرة للعمليات الحسابية +، *، /، %:

```
total += value; or total = total + value;
prod *= 10;     or prod = prod * 10;
x /= y + 1;     or x = x/(y + 1);
n %= 2;         or n = n % 2;
```

التعليمات في لغة C++، الصيغ المنطقية

إيضاح:

الصيغة المنطقية هي صيغة تعطي صح أو خطأ كنتيجة عند تقويمها

تستعمل المعاملات التالية في الصيغ المنطقية:

< أصغر من

> أكبر من

<=   أصغر من أو يساوي

>=   أكبر من أو يساوي

== يساوي

!= لا يساوي

2. التعليمات الشرطية

- التعليمة الشرطية **if statement**:

تأخذ التعليمة الشرطية في **C++** الشكل :

```
if (condition)
  statement
```

حيث يكون الشرط **condition** إما متحولاً منطقياً أو صيغة منطقية (راجع الفقرة السابقة).

أما **statement** فهو إما تعليمة وحيدة تنتهي بفاصلة منقوطة، أو مجموعة من التعليمات. وفي حالة مجموعة من التعليمات نضع هذه التعليمات بين قوسين كبيرين.

مثال

```
if (x > 0.0)
   cout << "The value of x is positive;"
```

- التعليمة الشرطية الاختيارية **if_else statement**:

تأخذ التعليمة الشرطية الاختيارية في **C++** الشكل :

```
if ( condition )
statementT
else
statementF
```

حيث يكون الشرط **condition** إما متحولاً منطقياً أو صيغة منطقية.

أما **statementT** و **statementF** فتدل كل منهما إما على تعليمة وحيدة تنتهي بفاصلة منقوطة، أو مجموعة من التعليمات. وفي حالة مجموعة من التعليمات نضع هذه التعليمات بين قوسين كبيرين.

مثال

```
if (disc >= 0.0)
  cout << "Roots are real";
if (disc < 0.0 )
  cout << "Roots are complex";
```

تدريب:

1. اكتب برنامجاً بلغة **C++** يقرأ عددين صحيحين **x** و **y** ويظهر على الشاشة القيمة المطلقة للفرق بينهما. اختبر البرنامج بقيم مختلفة تغطي جميع الحالات الممكنة (الأول>الثاني، الأول>الثاني، حالة المساواة)

2. اكتب برنامجاً بلغة **C++** يقرأ ثلاثة أعداد صحيحة تمثل علامات الطالب في ثلاثة مقررات، ويظهر كخرج الكلمة **Pass** أو **Fail**.

يعتبر الطالب ناجحاً (**Pass**) إذا كان ناجحاً في جميع الامتحانات (أي حائزاً على علامة **40** وما فوق في كل مقرر)، أو كان حائزاً على علامة أدنى من **40** في مقرر واحد على الأكثر ولكن وسطي العلامات الثلاث **50** وما فوق.

◄ تعليمة **switch**

تأخذ التعليمة **switch** في **C++** الشكل :

```
     switch (   selector )
    {
    case label1: statement1;
        break;
    case label2: statement2;
        break;
        ...
    case labeln: statementn;
        break;
    default: statementd; // optional
        break;
    }
```

حيث يكون **selector** متحولاً من نمط **int** أو **char** أو تعبيراً نتيجته من أحد هذين النمطين.

و **labeli** من نفس نمط **selector**.

مثال

```
switch (i)
  {
    case 1 :  grade = 'A';
              break;
    case 2 :  grade = 'B';
              break;
    case 3 :  grade = 'c';
              break;
    default : cout << i
                   << " not in range";
              break;
  }
```

3. التعليمات التكرارية

- ◄ تعليمة **while**

تأخذ التعليمة **while** في **C++** الشكل :

```
while (  condition  )
         statement
```

حيث يكون الشرط **condition** إما متحولاً منطقياً أو صيغة منطقية.

أما **statement** فهو إما تعليمة وحيدة تنتهي بفاصلة منقوطة، أو مجموعة من التعليمات. وفي حالة مجموعة من التعليمات نضع هذه التعليمات بين قوسين كبيرين.

مثال

```
sum = 0.0;
cin >> x;
while (x > 0.0)
   {
    sum += x;
    cin >> x;
   }
```

◄ تعليمة **for**

تأخذ التعليمة **for** في **C++** الشكل:

```
for ( initialise ; test ; update )
statement
```

حيث يمثل **initialise** عملية إعطاء قيمة ابتدائية لمتحول التحكم بالحلقة

حيث يمثل **test** عملية تقويم الصيغة المنطقية

حيث يمثل **update** عملية تعديل قيمة متحول التحكم بالحلقة قبل إعادة تنفيذها ثانية

وعادة نستعيض عن كتابة مجموعة التعليمات التالية:

```
i = 1;
while (i <= 10)
  {
    cout << i << endl;
    i++;
  }
```

بالتعليمة:

```
for (i = 1; i <= 10; i++)
  {
    cout << i << endl;
  }
```

تدريب:

1. اكتب برنامجاً يقرأالعدد الصحيح n و يحسب مجموع السلسلة:

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots+\frac{1}{n}$$

عدل البرنامج بحيث يحسب مجموع السلسلة:

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots \frac{1}{n}$$

لاحظ أنه كلما كانت n أكبر كان المجموع أقرب إلى **0.7854** أي (π/**4**)

2. اكتب برنامجاً يقرأ مجموعة من الأعداد الصحيحة التي يبلغ عددها **n** (**n** عدد صحيح يدخله المستخدم) ويحسب عدد الأعداد الموجبة والسالبة الواردة في هذه المجموعة.

3. اكتب برنامجاً يقوم بإظهار جدول الضرب من الرتبة **n** حيث **n** قيمة يدخلها المستخدم وتكون أصغر من **10.** على سبيل المثال إذا أعطى المستخدم لـ **n** القيمة **4** يجب أن يظهر الجدول التالي:

```
   1    2    3    4
1  1    2    3    4
2  2    4    6    8
3  3    6    9   12
4  4    8   12   16
```

استخدم لهذا الغرض الحلقات المتداخلة وانتبه للإظهار بحيث يظهر الجدول واضحاً.

القنوات Streams

في ++C تقرأ المعطيات من القنوات وتكتب في القنوات. والقناة هي:

- سلسلة من المحارف
- متصلة بجهاز أو بملف (على القرص)
- تحمل اسماً محدداً

ولقد تعاملنا سابقاً مع القناتين **cin** و **cout** الموصولتين إلى لوحة المفاتيح والشاشة على الترتيب. وعند استخدام هاتين القناتين يجب تضمين المكتبة **iostream** في البرنامج.

إيضاح

تستخدم بعض المترجمات ترويسة المكتبة التي تضمن في الملف بوضع **iostream.h** بدلاً من **iostream**.

تسمح ++C بالتعامل مع قنوات أخرى، ويصرح عن هذه القنوات كما يصرح عن أي متحول آخر. ونجد أنه للتصريح عن قنوات لإدخال المعطيات فإننا يجب أن نستخدم النمط **ifstream**، في حين نستخدم النمط **ofstream** للقنوات التي نستخدمها لإخراج المعطيات.

بعد التصريح عن القناة يجب أن نصلها بملف خارجي.

وجب ألا ننسى أن نضمن المكتبة **fstream** للبرنامج.

إيضاح

تستخدم بعض المترجمات ترويسة المكتبة التي تضمن في الملف بوضع **fstream.h** بدلاً من **fstream**.

وصل القنوات بالملفات الخارجية

إن الشكل العام لتعليمة وصل قناة بملف هو:

```
streamname.open(filename);
```

ملاحظة:

Streamname: اسم القناة

Filename: اسم الملف الخارجي

مثال

نبدأ أولاً بالتصريح عن القناة كما في التعليمات التالية:

```
ifstream ins;  // input stream
ofstream outs; // output stream
```

لوصل الأقنية بالملفات الخارجية الموافقة لها نستخدم التعليمة **open**، وهي في الواقع تابعاً يستدعى لهذا الغرض. فمثلاً لوصل القناة **ins** بالملف **indata.dat** نكتب:

```
ins.open("indata.dat");
```

إيضاح :

استدعاء التوابع الموجودة ضمن الأغراض يتم بالرمز . كما سنرى لاحقاً

إذا كان الملف **indata.dat** غير موجود فستفشل عملية الربط. نختبر ما إذا فشلت هذه العملية باستخدام التابع **streamname.fail ()** الذي يعيد القيمة **true** إذا فشل التابع **open**. يجب إجراء هذا الاختبار بعد كل عملية **open**، كما في المثال التالي:

```
ifstream ins;
ins.open("indata.dat");
if (ins.fail())
  {
    cout << "Error opening file indata.dat"
         << endl;
    return 1;
  }
```

في البرنامج الرئيسي الذي كتبت داخله هذه التعليمات، يتم الخروج من البرنامج عند فشل عملية الفتح لأننا استخدمنا التعليمة **return**. تعيد هذه التعليمة **1** مما يدل نظام التشغيل على توقف البرنامج بحالة خطأ.

بعد أن يتم الوصل فإن جميع عمليات القراءة من القناة تعني القراءة من الملف والكتابة في القناة تعني الكتابة في الملف.

مثال

```
int main()
{
 ifstream ins;
 ofstream outs;
 ins.open("indata.dat");
 if (ins.fail())
   {
     cout << "Error opening indata.dat"
          << endl;
     return 1;
   }
 outs.open("results.dat");
 if (outs.fail())
   {
     cout << "Error opening results.dat"
          << endl;
     return 1;
   }
      .
      .
```

الآن أصبح تعاملنا مع الملفين يتم من خلال المتحولين **ins** و **outs** باستخدام المعاملين >> و<<.

وهكذا نقرأ من القناة **ins** قيمة المتحول **x**:

```
ins >> x;
```

ونخرج النتائج على القناة **outs** :

```
outs << "Result is " << count << endl;
```

اختبار بلوغ نهاية ملف end-of-file

نستخدم لهذا الغرض التابع **eof**. تكون نتيجة الاختبار **true** إذا حاولنا القراءة من ملف بلغ نهايته، وإلا تكون **false**.

مثال

لقراءة القيم الموجودة في ملف وحساب وسطي هذه القيم نكتب البرنامج التالي:

```
#include <iostream.h>
#include <fstream.h>
int main()
{
  int n;
  float x, sum, average;
  ifstream ins;         // input stream
  ofstream outs;        // output stream
  ins.open("indata.dat");

     // open files, exit program if fail
  if (ins.fail())
    {
      cout << "Can't open indata.dat" <<endl;
      return 1; //exit with code 1 for failure
    }
  outs.open("results.dat");
  if (outs.fail())
    {
      cout << "Can't open results.dat" << endl;
      return 1; //exit with code 1 for failure
    }

  // Initialise and let user know something is happening
  sum = 0.0;
  n = 0;
  cout << "Reading input file " << endl;
      // read from file, accumulate sum and output average
      // to output file.
  ins >> x;  // if file was empty then eof would now be true
  while (!ins.eof())
    {
      sum += x;
      n++;
      ins >> x;
    }
  average = sum / n;
  cout << "Writing results to file " << endl;
  outs << "The average of " << n << " numbers is "
       << average << endl;
  ins.close();     // Close all files - GOOD PRACTICE
  outs.close();
  return 0;        // indicate success
}
```

لاحظ أننا قمنا بإغلاق الملفين **ins** و **outs**. يفضل في البرامج إجراء الإغلاق قبل الخروج من البرنامج، وإلا فقد يحصل أحياناً أن نفقد بعض المعطيات التي أخرجناها على الملف.