الفصل الثالث

البرامج الجزئية في C++

التوابع التي يعرفها المبرمج

تتيح لغة **C++** للمبرمج إمكان تعريف توابعه الخاصة. الشكل العام للتابع هو:

```
Function_type function_name( parameter_list)
{
Local_definitions;
Function_implementation;
}
```

- إذا كان التابع يعيد قيمة، يجب تحديد نمط هذه القيمة في **function_type**. وفي حال عدم وجود
  قيمة معادة نضع **void** في **function_type**.
- تسرد **parameter_list** موسطات التابع كل مع نمطه.
- إن **local_definitions** هي تعريفات للمتحولات المحلية المستخدمة داخل التابع، ولا يكون لهذه
  المتحولات أي معنى خارج التابع.
- تتألف **function_implementation** من التعليمات التي تقوم بالوظيفة المطلوبة من التابع.

مثال

التابع **distance** يأخذ كموسطي دخل إحداثيي نقطة **x,y** ويعيد بعد هذه النقطة عن مركز الإحداثيات.

```
float distance(float x, float y)
//Returns the distance of (x, y) from origin
    {
float dist;  //local variable
dist = sqrt(x * x + y * y) ;
return dist;
}
```

في هذا التابع قمنا باستخدام **متحول محلي local variable** يسمى **dist**.

التوابع التي ليس لها موسطات

يستخدم هذا النوع من التوابع استخداماً محدوداً، وهي تقوم عادة بوظيفة معينة دون أن تعيد أية نتيجة.

مثال

يقوم التابع التالي بإظهار انزياح مقداره ثلاثة أسطر، أي أنه يترك ثلاثة أسطر فارغة على الخرج ليبدأ الإظهار على السطر الرابع :

```
void skipthree(void)
//skips three lines on output
    {
cout << endl << endl << endl;
}
```

لاحظ نمط الخرج **void**، وعدم وجود موسطات دخل للتابع لذلك نضع **void** في **parameter_list**. كما أنه لا توجد **local_definitions**. ينصح بشدة اتباع القاعدة التي تقضي بكتابة تعليق في التابع يشرح وظيفة التابع كما في هذا التابع. ونظراً لعدم وجود نمط عودة، يجري استدعاء **call** التابع على النحو التالي:

```
Skipthree();
```

وبرغم عدم وجود موسطات دخل، فإنه **يجب** وضع القوسين ( ).

ينصح بالعودة إلى مقرر سابق للبرمجة لفهم كيفية عمل المترجم عند استدعاء البرنامج الرئيسي تابعاً ما.

ويجب أن يعرف أي تابع قبل التابع الرئيسي **main** الذي يقوم باستدعائه، لأن البرنامج الرئيسي يجب أن يعلم بوجوده وتعريفه وأنماط موسطاته عندما يصادف تعليمة استدعاء هذا التابع.

نرى في الجزء التالي من برنامج استدعاء التابع skipthree ترتيب تعريف التابع الرئيسي والتابع skipthree.

```
#include <iostream.h>
void skipthree(void)
//Function to skip three lines
{
cout << endl << endl << endl;
}

void main()
{
int …;
float …;
cout << “Title Line 1”;
skipthree();
cout << “Title Line 2”;
.
.
}
```

غير أن هذا الترتيب يقلل من فهم البرنامج، إذ أن البرنامج الرئيسي يحتوي على معلومات ذات أثر أكبر في إفهام القارئ وظيفة البرنامج من التوابع المختلفة، وعليه، يجدر به أن يأتي تعريفه أولاً. إلا أن هذا الحل يسبب عدم تعرف المترجم على التوابع عند استدعائها في البرنامج الرئيسي.

جرت معالجة هذه النقطة باستخدام ما يسمى نموذج التابع function prototypes. فنموذج التابع يحتوي على نمط العودة وأنماط موسطات الدخل. يوضع هذا النموذج قبل البرنامج الرئيسي الذي يستدعي هذا التابع. أما التعريف الكامل للتابع فيوضع بعد البرنامج الرئيسي، وقد يكون موجوداً في ملف منفصل سبقت ترجمته ويجري ربطه link مع البرنامج الرئيسي لاحقاً. يتطابق نموذج التابع مع ترويسته.

```
#include <iostream.h>

void skipthree(void);  // function prototype

void main()
{
int…..;
float…..;
cout << “Title Line 1”;
skipthree();
cout << “Title Line 2”;
.
.
}

// Now the function definition
void skipthree(void)
// Function to skip three lines
  {
cout << endl << endl << endl;
}
```

نموذج التابع skipthree

استدعاء التابع skipthree

مر معنا في الفصل السابق (راجع الفقرة: البرنامج الأول بلغة **C++** في الفصل الثاني تعرف **C++**) أن استخدام توابع من المكتبات يقتضي تضمين تعريفات هذه المكتبات في الملف عبر استخدام **include**. في الحقيقة تقوم عملية التضمين بوضع نماذج (ترويسات) التوابع في الملف

قبل البرنامج الرئيسي، بحيث يتعرف عليها المترجم عندما يجد استدعاءها، لأن هذه الملفات المضمنة تحتوي على الترويسات التي تقدمها المكتبة الموافقة.

للتعرف على مكتبة iostream.h راجع الملحق الأول في نهاية هذا الملف.

للتعرف على مكتبة iomanip.h راجع الملحق الثاني في نهاية هذا الملف.

التوابع التي تعيد نتيجة

ترد التوابع على الأغلب في شكل تأخذ فيه موسطات دخل وتعيد نتيجة. لنأخذ على سبيل المثال النقطة التي نريد حساب بعدها عن مركز الإحداثيات انطلاقاً من إحداثييها.

```
float distance(float x, float y)
//Returns the distance of (x, y) from origin
{
float dist;  //local variable
dist = sqrt(x * x + y * y);
return dist;
}
```

إن نموذج التابع distance هو:

```
float distance(float, float); // function prototype
```

يمكن لنا كتابة ما يلي في البرنامج الرئيسي:

```
float a, b, c, d, x, y;
a = 3.0;
b = 4.4;
c = 5.1;
d = 2.6;


x = distance(a, b);
y = distance(c, d);


if (distance(4.1, 6.7) > distance(x, y))
cout << " Message 1 " << endl;
```

قد نجد في التابع أكثر من تعليمة return كما في المثال التالي الذي نحسب فيه الجذر التربيعي لعدد حقيقي:

```
float mysqrt(float x)
//  Function returns square root of x.
//If x is negative it returns zero.
 {
const float tol = 1.0e-7;  // 7 significant figures
float xold, xnew;          // local variables
if (x <= 0.0)
return 0.0;              // covers -ve and zero case
else
{
xold = x;                // x as first approx
xnew = 0.5 * (xold + x / xold); // better approx
while (fabs((xold-xnew)/xnew) > tol)
{
xold = xnew;
xnew = 0.5 * (xold + x / xold);
}
return xnew;  // must return float value
}
}  //  end mysqrt
```

ولكن إذا كان التابع من النمط void، أي أنه لا يوجد نمط معاد، فيجب ألا يكون هناك تعليمة return في جسم التابع، فإن وجدت، فسيصدر المترجم تحذيراً بهذا الشأن.

مثال: تابع الرفع إلى قوة

يعيد هذا التابع نتيجة رفع الموسط الأول إلى القوة الموافقة للموسط الثاني. يكون الموسط الثاني عدداً صحيحاً، ويمكن أن تكون هذه القوة 0 أو سالبة.

```
float power(float x, int n)
   {
float product = 1.0;
int absn;
int i;
if ( n == 0)
return 1.0;
else
    {
absn = int(fabs(n));
for (i = 1; i <= absn; i++)
product *= x;
if (n < 0)
return 1.0 // product;
else
return product;
        }
}    //end of power
```

يمكن استخدام التابع كما يلي:

```
float x, y, z;
int p;
cout << “Enter a float and an integer:”;
cin >> x >> p;
y = power(x, p);
z = power(x + y, 3);
```

الموسطات التي تمرر بالقيمة Call-by-value parameters

مر معك في مقرر سابق للبرمجة نوعان من تمرير الموسطات: تمرير بالقيمة وتمرير بالمرجع.

(راجع البرمجة1 الفقرة "تمرير الوسطاء (المحددات) parameter Passing").

في التمرير بالقيمة لا تتغير قيمة المتحول الفعلي الذي يحل محل الموسط الشكلي عند الاستدعاء.
وهكذا ، فلو أضفنا التعليمة:

```
n++;
```

إلى تعليمات التابع power مباشرة قبل قوس نهاية التابع، ثم نفذنا التعليمات:

```
p = 4;
y = power(x, p);
cout << p;
```

لوجدنا النتيجة هي 4 وليس 5، وذلك لأنه جرى تمرير الموسط بالقيمة passed by value.

ويعني هذا التمرير أنه عند الاستدعاء نأخذ نسخة من قيمة المتحول الفعلي ونتعامل معها داخل التابع، في حين تبقى قيمة المتحول الفعلي بعيدة عن التغيير.

نستخدم هذا النمط من التمرير عندما نرغب بتمرير المعلومات للتابع من خلال متحول معين دون إجراء أي تعديل عليه، وهذا ما يجنّب إحداث آثار جانبية.

الموسطات التي تمرر بالمرجع Call-by- reference parameters

لا يمكن إعادة قيمة جرى حسابها في تابع عبر الموسطات الممررة بالقيمة للأسباب التي ذكرناها في الفقرة السابقة، ولكن لو رغبنا بذلك فيجب أن نمرر للتابع عنوان الموسط لا قيمته. يستخدم التابع هذا العنوان للنفاذ إلى الموسط الفعلي وإجراء التعديل اللازم على قيمته بحيث يبقى هذا التعديل جاري المفعول بعد انتهاء استدعاء التابع. وبما أننا نمرر مرجعاً للموسط، فإنني نسمي هذا النمط بالتمرير بالمرجع **call-by-reference**. ولتمييز الموسط الذي نمرره بالمرجع عن ذلك الذي نمرره بالقيمة، نستخدم الرمز & الذي نضعه بعد نمط الموسط.

مثال

ليكن لدينا التابع التالي الذي يحل معادلة من الدرجة الثانية:

```
//  solves the quadratic equation a*x*x+b*x+c = 0.
//  If the roots are real then the roots are
//returned in two parameters root1 and root2 and
//  the function returns true, if they are complex
//then the function returns false.

bool quadsolve(float a,        // IN coefficient
                    float b,        // IN coefficient
                    float c,        // IN coefficient
                    float& root1,   // OUT root
                    float& root2)   // OUT root
 {
float disc;      // local variable
disc = b * b - 4 * a * c;
if (disc < 0.0)
return false;
else
        {
root1 = (-b + sqrt(disc))/(2 * a);
root2 = (-b - sqrt(disc))/(2 * a);
return true;
        }
  }
```

لاحظ أنه جرى التصريح عن الجذرين (وهما موسطي الخرج) كموسطين ممررين بالمرجع، في حين أنه جرى تمرير أمثال المعادلة بالقيمة لأنها موسطات دخل. ويكون شكل نموذج التابع كمايلي :

```
int quadsolve(float, float, float, float&, float&);
```

يجري استدعاء التابع في البرنامج الرئيسي على النحو التالي:

```
float c1, c2, c3;
float r1, r2;
.
.
if (quadsolve(c1, c2, c3, r1, r2))
cout << “Roots are “ << r1 << “ and “ << r2 << endl;
else
cout << “Complex Roots” << endl;
```

تدريب:

يطلب كتابة برنامج يقوم بمعالجة فاتورة تحتوي المعلومات التالية:

n عدد البنود الموجودة في الفاتورة

يتميز كل بند بالمعلومات التالية: رمز للبند(على ست خانات)، الكمية، سعر القطعة الواحدة(بالليرات وأجزائها العشرية)

وهكذا يمكن على سبيل المثال إعطاء الدخل التالي للبرنامج :

```
3
161432 5 6 50
543289 10 2 25
876234 2 10 75
```

للدلالة على وجود 3 بنود في الفاتورة، أولها رمزه هو 161432 كميته 5 وسعر الواحدة منه 6 ليرات و50 قرشاً.

والمطلوب كتابة برنامج بلغة **C++** يقرأ تاريخ اليوم، ثم يقرأ المعلومات الخاصة بفاتورة كما في المثال أعلاه، ويقرأ تاريخ اليوم، ثم يظهر السعر الكلي لكل بند، وسعر المجموع.

يكون شكل الخرج كما يلي:

```
Invoice date: 10/6/96
 Item    quantity  unit price  total price

161432       5         6.50        32.50
543289      10         2.25        22.50
876234       2        10.75        21.50

                          Total    76.50
```

يمكن أن نكتب خوارزمية الحل على النحو التالي:

إعطاء القيم الابتدائية initialise

قراءة التاريخ

قراءة عدد البنود n

إظهار الترويسة

من أجل n بند كرر

{

قراءة المعلومات المتعلقة بالبند

حساب السعر الإجمالي للبند

الإضافة للسعر الإجمالي للفاتورة

كتابة سطر في الفاتورة خاص بالبند

إظهار السعر الإجمالي للفاتورة

{

لو افترضنا أنه يوجد لدينا أربعة مبرمجين لكتابة الحل، فيمكن أن نقسم الحل إلى أربعة مهام منفصلة نراها واضحة داخل الحلقة، ويمكن إعطاء كل مهمة منها لمبرمج. ولعل أفضل طريقة لإنجاح تقسيم المهمات هو الطلب من كل مبرمج أن يكتب تابعاً للمهمة الموكلة إليه.

وتحدد مهمة كل مبرمج بتوصيف دقيق كما يلي:

لنأخذ مثلاً مهمة قراءة المعلومات المتعلقة بالبند. توصف المهمة كما يلي:

```
Function name: dataentry
Operation:     Enter a record
Description:   Enters four integers from the current
               input line and returns their values.
Parameters:    Output parameter  int  itemno
               Output parameter  int  quantity
               Output parameter  int  unitpounds
               Output parameter  int  unitpence
```

وبالمثل توصف المهمة حساب السعر الإجمالي للبند كما يلي:

```
Function name : calccost
Operation     : Calculates the cost for a single item.
Description   : Given the unit price of an item in
                pounds and pence and the quantity of
                the item calculates the total cost in
                pounds and pence.
Parameters    : Input parameter int quantity
                input parameter int unitpounds
                input parameter int unitpence
                output parameter int totalpound
                output parameter int totalpence
```

وبالمثل توصف المهمة الإضافة للسعر الإجمالي للفاتورة كما يلي:

```
Function name : acctotal
Operation     : Accumulates the total cost of invoice
Description   : Given current total invoice cost and
                the total cost of an invoice item
                calculates the new total invoice cost.
Parameters    : input parameter int totalpound
                input parameter int totalpence
```

input & output parameter int invpound
input & output parameter int invpence

وبالمثل توصف المهمة كتابة سطر في الفاتورة خاص بالبند كما يلي:

Function name : writeline

Operation : Writes a line of the invoice.

Description : Given the item reference number, the quantity, the unit price and total price of an item outputs a line of the invoice.

Parameters : input parameter int itemno
input parameter int quantity
input parameter int unitpounds
input parameter int unitpence
input parameter int totalpound
input parameter int totalpence

وهكذا نحصل على البرنامج التالي الذي يقوم باستدعاء التوابع التي تنفذ هذه المهمات على النحو التالي:

```
void main()
{
    int i,          // control variable
        n,          // number of items
        itemno,     // item reference number
        quantity,   // quantity of item
unitpounds,
unitpence,  // unit item price
totalpound,
totalpence, // total item price
invpound,
invpence;   // total invoice price
//  initialise
invpound = 0; // total value of invoice has to be
invpence = 0; // set to zero initially
//  Enter number of items
cout << “Enter number of items on invoice:”;
cin  >> n;
//  Headings
cout << “ Item   quantity  unit price  total price”
<<endl << endl;
//For n items
for (i=1; i<=n; i++)
     {
dataentry(itemno, quantity, unitpounds, unitpence);
calccost(quantity, unitpounds, unitpence, totalpound,
totalpence);
acctotal(totalpound, totalpence, invpound, invpence);
writeline(itemno, quantity, unitpounds, unitpence,
totalpound, totalpence);
   }
//write total line
cout << “                          Total     “
<< invpound
 << “.”
<<invpence << endl;
}
```

لاحظ هنا أن الفائدة العظمى من التوابع هي أننا يمكن أن نأخذ توصيف أي من المهمات الواردة سابقاً، فننفذها بكتابة تابع يحقق هذه المهمة، ونختبره على حده. فمثلاً، لو أخذنا التابع calccost فإننا ننفذه على النحو التالي:

```
void calccost(int q, int ul, int up, int& totl, int& totp)
//  Calculates the quantity q times the unit cost in
//  pounds and pence in ul and up and places the
//result in pounds and pence in totl and totp
{
int p;
p = q * up;
totp = p % 100;
totl = q * ul + p/100;
}
```

لاختبار هذا التابع على نحو مستقل عن البرنامج، يجب أن نكتب **برنامج سواق** **driver program.**

إيضاح:

البرنامج السواق هو برنامج يسمح بأن يقوم المبرمج بإدخال قيم لموسطات التابع المراد اختباره ، ثم يظهر النتيجة الموافقة.

فعلى سبيل المثال يمكن أن نكتب البرنامج السواق التالي:

```
// Driver program to test calccost
#include <iostream.h>
// function prototype
void calccost (int, int, int, int&, int&);


void main()
{
int quant, unitl, unitp, totall, totalp;
// stop on negative quantity
cout << “Enter quantity:”;
cin >> quant;
while (quant >= 0)
     {
cout << “Enter unit cost (pounds pence:) “;
cin >> unitl >> unitp;
calccost(quant, unitl, unitp, totall, totalp);
cout << endl
<< quant << “ times”
<< unitl << “ pounds”
<<unitp << “ pence”
<<”is “
<< totall << “ pounds”
<< totalp << “ pence “;
cout << endl << “ Enter quantity:”;
cin >> quant;
}
}
// function definition here
```

عند الاختبار يجب أن تحاول عطاء قيم دخل توافق جميع الحالات. فمثلاً إذا أدخلنا معطيات دخل لهذا التابع بحيث نسحب السعر الإجمالي لـ 7 قطع من بند سعر القطعة منه هو 1.10 لا يختبر حالات وجود حمل في الجزء العشري من المبلغ بحيث يضاف إلى الجزء الصحيح. ينصح باختبار السعر الإجمالي لـ 6 قطع يبلغ سعر الواحدة منها 2.74 مثلاً. نقصد هنا أن نقول أن اختيار معطيات الدخل يستلزم عناية خاصة.

التحميل الزائد للتوابع

يسمح في البرنامج نفسه في C++ أن يجري تعريف تابعين أو أكثر لهما الاسم نفسه. يفترض هذا ضمناً أن هذه التوابع تختلف فيما يتعلق بقائمة موسطاتها الشكلية formal parameters. وتقع على عاتق المترجم مهمة تحديد التابع الواجب استدعاؤه بناء على الموسطات الفعلية effective parameters المستخدمة عند استدعاء التابع.

يسمح هذا الإمكان بتسهيل قراءة البرامج، إذ لا يطالَب المبرمج في كل مرة باختراع اسم جديد لتابع يقوم بعمل مطابق لتابع آخر، لكن على معطيات من نمط مختلف. إن إمكان زيادة تحميل التوابع يسمح بإعطاء صفة تعددية الأشكال للعمل، يحدد الشكل حسب نمط الموسطات وعددها عند الاستدعاء.

لنأخذ مثلاً الحالة التي نريد فيها صنع تابع لإظهار القيمة الممررة في الموسط. لو كانت اللغة لا تسمح باستخدام الاسم ذاته للتوابع كما في العديد من لغات البرمجة، لوجب أن نكتب التوابع التالية:

```
void displayint (int i  )
{
cout<<”integer “<<i<<endl;
}


void displaydouble (double i  )
{
cout<<”double “ <<i<<endl;
}
```

تقدم لغة C++ إمكان استخدام الاسم نفسه، فنكتب البرنامج التالي:

```
#include <iostream.h>
void display (int i)
{
cout<<”integer “<<i<<endl;
}

void display (double i  )
{
cout<<”double “ <<i<<endl;
}

int main()
{
display(7);
display(3.5);
return 0;
}
```

يعطي تنفيذ البرنامج:

```
integer 7
double 3.5
```

يتبين لنا هنا أن المترجم قام في كلتا الحالتين بالاختيار الصحيح للتابع الواجب استدعاؤه حسب نمط الموسطات الممررة للتابع. ولكن يجب الانتباه طبعاً إلى عدم التسبب في حالات التباس لا يستطيع فيها المترجم الاختيار بدلاً عن المبرمج.

مثال

```
#include <iostream.h>
void display (int i )
{cout<<”integer “<<i<<endl;}

void display (double i )
{cout<<”double “ <<i<<endl;}

int main(){
long int i = 1;
display(7);
display(3.5);
display(i);  // Error … compilation is ambiguous!!
return 0;
}
```

في هذا المثال، يقع المترجم في حيرة، إذ أنه لا يعرف: أيحول المتحول i ذو القيمة 1 والنمط long ضمنياً إلى int أم إلى double؟ علماً بأن كلتي الحالتين ممكنتان. يجب إذن استعمال زيادة التحميل مع الحذر من الوقوع في مثل هذه الحالات.

المعاملات الافتراضية

عندما يكتب المبرمج توابع تستخدم موسطات، يمكن أن يكون لبعض هذه الموسطات قيم اعتيادية. في هذه الحالة، يمكن تعريف تابع بمحددات ذات قيم افتراضية. يسرع هذا الأمر تنفيذ استدعاء التابع لأن هذه المحددات لا تحتاج إلى أن تُنسخ. لهذا السبب، جعلت المحددات التي تأخذ قيماً افتراضية على اليمين في نموذج التابع.

فعلى سبيل المثال:

```
void init (int a , int b  = 0) ;      // 2nd argument = 0 by default
```

في هذه الحالة، init (3) و init (2,4) هي أشكال صحيحة لاستدعاء التابع المصرح عنه.

إن المثال التالي خاطئ لأن المحددات الافتراضية يجب أن تكون متوالية اعتباراً من أقصى اليمين نحو اليسار.

```
void incorrect (int a =3, int b, int c = 0) ; //error, b has no value
```

إذا جرى التصريح عن تابع مع محددات افتراضية، فلا يصح عند تعريف التابع تحديد هذه القيم الافتراضية مرة أخرى

```
void init (int a = 0) ;
//declaration of init
    .
    .
main()
{
    .
    .
}

void init (int a = 0) //definition of init ERROR
{/* body   */ }
```

```
void init (int a = 0) ;
// declaration of init
    .
    .
main()
{
    .
    .
}

void init (int a ) //OK
    { / * body * / }
```

بالطبع، يسمح بالتعريف الزائد للتابع ذي المحددات الافتراضية. في هذه الحالة يجب التنبه إلى عدم تحديد القيم الافتراضية مرة أخرى عند إعادة التعريف.

في المثال التالي نعرف التابع init الذي يأخذ عددين صحيحين كدخل، غير أنه يمكن أن يأخذ العدد الأول، وتكون القيمة الافتراضية للثاني 3. نقوم بعد ذلك بتعريف ثان للتابع نعطي فيه قيمة افتراضية للموسط الأول ولتكن 7.

```
#include <iostream.h>
void init(int,int=3);
void init(int a,int b) { cout<<a <<",”<<b;}
void init(int=7,int);  // overdefinition

int main(){
init(2,1); // displays 2,1
init(4);   // displays 4,3
init();    // displays 7,3
return 0;
}
```

عند التعريف الزائد للتابع init() هنا لم يكن يسمح بإعادة تعريف القيمة الافتراضية للمحدد الثاني. نلاحظ هنا أنه يجب ألا يذكر اسم المحدد الذي يأخذ قيمة افتراضية.

المصفوفات Arrays في C++

لا بد أنك قد سبق لك أن تعاملت مع بنية المعطيات المسماة مصفوفة في مقرر سابق للبرمجة (راجع فقرة "بنية المعطيات مصفوفة" في المقرر برمجة1)، وهي بنية تسمح بإعطاء اسم واحد لمجموعة من العناصر التي لها نفس النمط، ويمكن الوصول إلى كل عنصر منه باستخدام الدليل subscript /index الذي يدل على ترتيبه في المصفوفة، ويجب أن يكون هذا الدليل عدداً صحيحاً.

للتصريح عن المصفوفة في C++ نحدد نمط العناصر التي سنخزنها في المصفوفة، كما نعطي اسماً للمصفوفة، ونحدد بين قوسين متوسطين عدد العناصر فيها ويجب أن يكون هذا العدد صحيحاً.

مثال

يمكن تخزين المعطيات التي تمثل وسطي درجة الحرارة في العام في سوريا على مدى الأعوام الخمسين الماضية في مصفوفة على النحو التالي:

```
float annual_temp[50];
```

يجب أن يكون عدد العناصر محدداً في زمن الترجمة، لذا يفضل وضع القيمة الموافقة لطول المصفوفة في ثابت، بحيث نقوم في حال رغبنا بتغيير طول الشعاع بتعديل قيمة هذا الثابت فقط بدلاً من البحث داخل أسطر الرماز عن كل موقع تظهر فيه هذه القيمة واستبدالها.

```
const int NE = 50;
float annual_temp[NE];
```

ويجب عدم استخدام متحول لطول الشعاع لأن المترجم لا يعرف قيمة المتحول عند الترجمة كما في المثال:

```
int i;
cin>>i;
float annual_temp[i];
//Error
```

وبالمثل يعتبر التصريح التالي غير صحيح:

```
int i=50;
float
annual_temp[i];//Error
```

دليل المصفوفة

يأخذ العنصر الأول في المصفوفة في C++ الدليل ذا القيمة 0، في حين يأخذ العنصر الأخير القيمة الموافقة للطول -1، ويمكن وضع قيمة الدليل في متحول وإجراء عمليات عليه.

مثال

```
const int NE = 100;
N = 50;
int i, j, count[N];
float annual_temp[NE];
float sum, av1, av2;
.
for (i = 0; i < NE; i++)
cin >> annual_temp[i];
.
.
cin >> count[i];
count[i] = count[i] + 5;
count[i] += 12;
.
.
if (annual_temp[j] < 10.0)
cout << “It was cold this year”<<endl;
```

ولإيجاد وسطي درجات الحرارة لـ k عنصر مخزن في المصفوفة annual_temp اعتباراً من بدايتها (k<=NE) نكتب:

```
sum = 0.0;
for (i = NE - k; i < NE; i++)
  sum += annual_temp[i];
av2 = sum / k;
```

ملاحظة هامة:

لا تتحقق لغة C++ من كون قيمة الدليل التي يجري التعامل معها في تعليمة ما هي بين القيمة الدنيا والعظمى المصرح عنها. وعليه يمكن أن تخطئ وتكتب annual_temp[200]=10.8; (تعرف المصفوفة annual_temp كما يلي:

```
const int NE = 100;
float annual_temp[NE];
```

(

دون أن يعطي المترجم خطأ ولكن لا يمكن توقع نتيجة ذلك الأمر. فقد يحدث أن يحاول البرنامج الكتابة على موضع في الذاكرة يقع خارج فضاء الذاكرة المخصصة له، وعندها يتدخل برنامج التشغيل لإيقاف عمل البرنامج.

وقد يحاول الكتابة على موضع في الذاكرة يقع داخل فضاء الذاكرة المخصصة له فيعدل قيمة متحول في البرنامج أعطي له هذا الموضع، أو يكتب فوق الرماز المولد لبرنامجك. ويحصل أمر مماثل في حالة قراءة المعطيات المخزنة خارج المجال.

تقع مسؤولية حسن إدارة الدليل بحيث لا يتجاوز المجال المصرح عنه على المبرمج. ويسمى هذا التجاوز **طوفان الدليل Subscript Overflow.**

إعطاء المصفوفة قيمة ابتدائية

كما في حالة المتحولات البسيطة، يمكننا عند التصريح عن المصفوفة إعطاء عناصرها قيماً ابتدائية على النحو التالي:

```
int primes[] = {1, 2, 3, 5, 7, 11, 13};
```

لاحظ أننا هنا لم نحدد طول المصفوفة، إذ أن المترجم يمكنه استنتاج الطول من عدد القيم الابتدائية المعطاة وهو هنا 7.

وبافتراض أننا حددنا طول المصفوفة، فيمكن أن تحتوي قائمة القيم الابتدائية عدداً من العناصر مساوياً أو أقل من الطول. ففي المثال:

```
int primes[] = {1, 2, 3, 5, 7};
```

يحجز المترجم عشرة مواضع ويضع في الخمسة الأولى منها القيم الواردة في القائمة، ويكمل ما تبقى بـ 0.

سلاسل المحارف strings في C++

يمكن في C++ بناء سلسلة محارف والتعامل معها وكأنها عنصر واحد، وذلك باعتبارها مصفوفة عناصرها من النمط char. ويوضع عادة علامة في نهاية هذا النوع من المصفوفات للدلالة على نهايتها، وهذه العلامة هي محرف خاص يسمى **المحرف الصفري** **null character** وقيمته هي '\0' . وهكذا يكون طول كل سلسلة محارف مساوياً لعدد محارفها+1.

ونصرح عن متحول s1 كسلسلة محارف على النحو التالي:

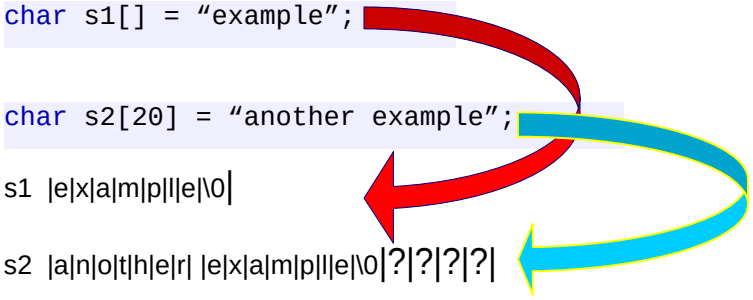
```
char s1[10];
```

تتسع s1 لتسعة محارف (+ المحرف الصفري).

مثال

```
char s1[] = "example";

char s2[20] = "another example";
```



سلاسل المحارف strings في C++، إخراج المحارف

كما هي الحال في جميع أنواع المتحولات، يكون الإخراج كما في المثال التالي:

```
cout << "The string s1 is " << s1 << endl;
```

فيظهر على الشاشة:

The string s1 is example

سلاسل المحارف strings في C++، إدخال المحارف

كما هي الحال في جميع أنواع المتحولات، يكون الإدخال كما في المثال التالي:

```
cin >> s1;
```

تنتهي قراءة متحول من نمط سلسلة محارف عندما يدخل المستخدم فراغاً أو enter أو tab.
ويضاف المحرف الصفري إلى آخر المتحول s1 .

بافتراض أن المتحول s1 كان يحتوي القيمة "example" قبل عملية القراءة

ثم أدخل المستخدم الكلمة first لوجدنا داخل المتحول s1 :

|f|i|r|s|t|\0|e|\0|

إيضاح:

بما أن طول الكلمة المدخلة first أقصر من طول السلسلة s1 تخزن الكلمة، ثم يوضع 0\ ويبقى باقي السلسلة على حاله قبل تنفيذ هذه التعليمة أي أنه يبقى فيه المحتوى السابق.

تقويم:

ماذا يحصل لو قرأنا كلمة طولها أكبر من طول المتحول الذي يجب أن نخزنها فيه؟

لقراءة عدة كلمات من الدخل باستخدام تعيمة cin يجب تخصيص عدة متحولات،كما في المثال التالي:

```
char first[12], last[12];
cout << “Enter your name (first last)";
cin >> first;
cin >> last;
cout << “The name entered was“
<<first “  “<<
<< last;
```

تدريب:

نريد كتابة برنامج ينسخ ملفاً يحتوي على أعداد صحيحة في ملف آخر. يدخل المستخدم اسم الملف الأصلي (ملف الدخل) واسم الملف الناتج (ملف الخرج) ثم ينسخ محتويات الملف الأصلي على الملف الناتج.

```
// Example program which copies a specified
// input file to a specified output file.
// It is assumed that the input file holds a
// sequence of integer values.

#include <iostream.h>
#include <fstream.h>

int main()
{
  ifstream ins;                  // declare input and output
  ofstream outs;                 // file streams
char infile[20], outfile[20];  // strings for file names
int I;
// ask user for file names
cout << “Enter input file name:”;
cin >> infile;
cout << “Enter output file name:”;
cin >> outfile;

//Associate file names with streams
ins.open(infile);
if (ins.fail())
{
cout << “Could not open file “ << infile <<” for input” << endl;
return 1; // exit with code 1 for failure
}
outs.open(outfile);
if (outs.fail())
{
cout << “Could not open file “ << outfile
<<” for output” << endl;
return 1; // exit with code 1 for failure
   }

//input from input file and copy to output file
ins >> I;
while (!ins.eof())
{
outs << i<<”  “;
```

```
ins >> i;
}
outs << endl;
//close files
ins.close();
outs.close();
return 0; //return success indication.
}
```

النمط string في C++

يوجد في C++ نمط خاص بالسلاسل المحرفية يسمى string يمكّنك من التعامل مع أية سلسلة محرفية كما لو كنت تتعامل مع متحول بسيط، مما يسهل تداول السلاسل المحرفية. يتميز هذا النمط عن تعريف السلسلة المحرفية كمصفوفة من المحارف بأنه لا يلزم المبرمج بتحديد الطول عند التصريح، كما يسمح له بالتعامل مع المتحولات من هذا النمط بعمليات سهلة وبسيطة.

التصريح عن متحول من هذا النمط يجب كتابة: #include<string> ثم:

```
string s,s1="hi";
```

تجري قراءة المتحول من النمط string باستخدام cin،

!=	==	<	>

يمكن مقارنة متحولين من هذا النمط مباشرة باستخدام الرموز

والترتيب الموجود على هذا النمط هو الترتيب الأبجدي على مستوى الكلمة كاملة.

توجد توابع وعمليات خاصة للتعامل مع هذا النمط منها:

- التابع substr(b,l): وهو تابع يعطي كنتيجة السلسلة الجزئية الموجودة في السلسلة التي يستدعى معها هذا التابع اعتباراً من الدليل b وبطول مقداره L.

مثال

```
string s1="hello";
string s2=s1.substr(0,2);
cout<<s2;   //result:  he
```

ملاحظة

لاحظ هنا أن المتحول s1 الذي كنا نتوقع تمريره للتابع substr بين قوسين قد قام هو باستدعاء التابع substr باستخدام المؤثر النقطة . ذلك لأن s1 هو غرض وهذه هي طريقة استدعاء التوابع في الأغراض. سنفصل في فصل "الصفوف والأغراض" هذه الطريقة.

- التابع length: للحصول على طول سلسلة.

مثال

```
string s1=”hello”;
int le=s1.length();
cout<<le;  //result     5
```

- المؤثر [ ] : يمكن استخدام المؤثر [ ] للوصول إلى محرف محدد.

مثال

```
string s1=”hello”;
char c=s1[0];
cout<<c;  //result      h
```

- المؤثر + : يمكن استخدام المؤثر + لضم سلسلتين إلى بعضها.

مثال

```
string s1=”hello”;
string s2=” world”;
string s=s1+s2;
cout<<s;  //result      hello world
```

مثال

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
string s1,s2;
s1="hello";  //assignment
cout<<"Guess the hidden word, Enter a word"<<endl;
cin>>s2;
if (s1==s2)
   cout<<"You guess!"<<endl;
else cout<<"You miss it, I will give you the first two characters"<<endl;
string s=s1.substr(0,2);
cout<<s;
return 0
};
```

الملحق 1

تحتوي هذه المكتبة تعريفات لمجموعة من المؤثرات نذكر منها على سبيل المثال:

setfill(char) , setprecision(int), setw(int).

- setw : يسمح setw(int) بتحديد حجم الحقل المراد إدراج القيمة ضمنه. وفي هذه الحالة يكون التنسيق إلى اليمين وهو أمر عملي في حالة تمثيل الأعداد.

```
#include <iostream.h>
#include <iomanip.h>
int main() {
  int n = 64;
  cout << "In hexadecimal : " << hex << n << endl;
  cout << " In octal : " << oct << n << endl;
  cout << " In decimal : " << dec << n << endl;
  //  The same display right justified
  cout << setw(20) << "Hexadecimal : " << hex << setw(6) << n << endl;
  cout << setw(20) << "Octal : " << oct << setw(6) << n << endl;
  cout << setw(20) << "Decimal : " << dec << setw(6) << n << endl;

  return 0;
}
```

يعطي التنفيذ النتيجة التالية:

```
In hexadecimal :  40
 In octal : 100
 In decimal : 64
     Hexadecimal :     40
                 Octal :   100
                Decimal  :     64
```

يبدو هذا المؤثر مفيداً جداً عند تحصيل القيم. يقوم البرنامج التالي بإظهار النص المدخل على شكل كتل طولها LL-1 ينتهي كل منها بالمحرف "\0".

```
#include <iostream.h>
#include <iomanip.h>
const int LL = 10;  // maximum size of a line
int main() {
  char line[LL];
  while (cin >> setw(LL) >> line)
       cout << line << endl;
  return 0;
}
```

يعطي التنفيذ ما يلي:

```
abcdefghijklmnopqrstuvwxyz
abcdefghi
jklmnopqr
stuvwxyz
```

^Z

ملاحظة : ^Z هو المحرف الذي ينهي ملف الدخل في الأجهزة التي تعمل بـ Ms_Dos.

- setprecision : في الحالة الافتراضية، يجري إظهار أعداد الفاصلة العائمة على 6 أرقام معنوية. يمكن تعديل هذه الدقة باستعمال المعالج setprecision (int).

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

const double pi = 3.141592654;

int main() {
  cout << pi << endl;                    // 6 digit by default
  cout << setprecision(9) << pi << endl; // 9 digit
  cout << pi/2.0 << endl;                // we are still on  9 digit
  cout << setprecision(2) << pi << endl; // 2 digit
  return 0;
}
```

وعند التنفيذ نجد:

```
 3.14159
 3.14159265
 1.57079633
 3.1
```

setfill: يستعمل المؤثر setfill(char) عادةً مرتبطاً بـ setw(int)، ويملأ المساحة غير المستعملة بالمحرف المحدد.

```
#include <iostream.h>
#include <iomanip.h>
int main() {
  int n = 64;
  cout << "In hexadecimal : " << hex << n << endl;
  cout << "In octal : " << oct << n << endl;
  cout << "In decimal : " << dec << n << endl;
  // The same display right justified
  cout << setw(20) << "Octal : " << oct << setw(6) << n << endl;
  cout << setw(20) << "Decimal : " << dec << setw(6) << n << endl;

  cout << setw(20) << "In hexadecimal : ";
  cout << hex << setfill('.') << setw(6) << n << endl;
  cout << setfill(' ') << setw(20) << "In octal : ";
  cout << oct << setfill('.') << setw(6) << n << endl;
  cout << setfill(' ') << setw(20) << "In decimal : ";
  cout << dec << setfill('.') << setw(6) << n << endl;

  return 0;
}
```

يعطي التنفيذ النتيجة التالية:

```
In hexadecimal :  40
 In octal : 100
In decimal : 64
      In hexadecimal : ….40
                In  Octal : ...100
      In Decimal  : .…64
```

يمكن للقارئ المهتم أن يعود لدليل المستخدم للمترجم **C++** الذي يعمل عليه، للاطلاع على القائمة الشاملة للمؤشرات الموجودة.

الملحق 2

تعرف مكتبة iostream.h التي توجد مع كل نظام C++ نمطين من الدخل والخرج يسمى الأول istream (input stream) (أي قناة الدخل) والثاني ostream (output stream) (أي قناة الخرج).

تُنفّذ عملية الإخراج من قبل المبرمج باستعمال مؤثر الإدراج ">>" إذ يدرج قيمة في قناة الخرج.

أما عملية الإدخال فهي تنفّذ باستعمال مؤثر الاستخراج "<<". أي أننا نستخرج قيمة من قناة الدخل.

وهنالك ثلاثة متحولات قَبلِيةُ التعريف من النمط "قناة" وهي:

- cin، متحول من النمط istream مرتبط بملف الدخل المِقيس.
- cout، متحول من النمط ostream مرتبط بملف الخرج المِقيس.
- Cerr متحول من النمط ostream مرتبط بملف الأخطاء المِقيس ولا ترتبط به ذاكرة مؤقتة.

كما يوجد في هذه المكتبة تعريفات للمؤثرات التالية:

- المؤثر endl الذي يتسبب في إدراج المحرف "\n"
- المؤثرات dec, flush, hex, oct :

1. flush :يتسبب في كتابة الذاكرة المؤقتة المرتبطة بالقناة، ومن ثَم بمسحها وإفراغها. قد يكون هذا الأمر ضرورياً في عمليات التحصيل، إذ يجب أن يظهر النص الذي يوضح المطلوب قبل أن يقوم المؤثر بإدخال القيمة المطلوبة. فإن لم يستعمل flush، فلا نضمن هذا الأمر.

```
#include <iostream.h>
int main() {
  int numbe;
  cout << "Enter a number between 0 and 12 " << flush;

  cin >> nombre;
}
```

2. Hex يجعل النظام ينتقل للنمط الست عشري أثناء استخراج وإدراج الأعداد الصحيحة. يبقى هذا النظام فعالاً إلى أن يطلب المبرمج تغييره. وهناك

مؤثرات أخرى مشابهة هي oct, dec، غير أن الأول يعتمد الأساس الثماني، والثاني يعتمد الأساس العشري. في الحالة العامة الافتراضية يعمل النظام في النمط العشري.

```
#include <iostream.h>

int main() {
  int n;
  cout << "Enter an integer number " << flush;
  cin >> n;  // equivalent to cin >> dec >> n;
  cout << "This is the  number in hexadecimal : " << hex << n << endl;
  cout << " This is the  number in octal : " << oct << n << endl;

  cout << (++n) << endl; // we are still in octal mode!

  cout << " This is the  number in hexadecimal : " << hex << n << endl;
  // now we move to decimal mode
  cout << " This is the  number in decimal : " << dec << n << endl;

  cout << " Enter an integer number " << flush;
  cin >> hex >> n;  // input in en hexadecimal mode
  cout << " This is the  number in hexadecimal: " << hex << n << endl;
  cout << " This is the  number in decimal: " << dec << n << endl;

  cout << " Enter an integer number " << flush;
  cin >> n;  // still in hexadecimal mode
 cout << "" This is the  number in hexadecimal: " << hex << n << endl;
  cout << "" This is the  number in decimal: " << dec << n << endl;

  return 0;
}
```

يعطي تنفيذ البرنامج ما يلي:

```
Enter an integer number 45
This is the  number in hexadecimal : 2d
This is the  number in octal : 55
56
This is the  number in hexadecimal : 2
This is the  number in decimal : 46
Enter an integer number AB
This is the  number in hexadecimal: ab
This is the  number in decimal: 171
Enter an integer number D2
This is the  number in hexadecimal: d2
This is the  number in decimal: 210
```

- يسمح المؤثر ws (white space) أثناء القراءة من القناة باستخراج واستبعاد جميع المحارف "الفارغة" أي محارف الفراغ والجدولة والقفز إلى سطر جديد والقفز إلى

صفحة جديدة، وذلك اعتباراً من الموقع الحالي وحتى أول محرف غير فارغ.