



كلية الهندسة – قسم المعلوماتية

مقرر برمجة 2

إ. د. علي سليمان

محاضرات الأسبوع العاشرة

Templates

الفصل الاول 2023-2024

1. مقدمة.
 2. تعريف التابع العمومي، الصنف العمومي.
 3. إنشاء قالب تابع.
 4. القيام بالاستخدامات المتعددة لقوالب التوابع.
 5. قيود استخدام التابع العمومية.
 6. إنشاء قوالب الأصناف وتوظيفها في حل العديد من المسائل.
1. Introduction.
 2. Definition of Generic Functions, Generic class.
 3. Create a Generic template.
 4. Make multiple uses of function templates.
 5. constraint general method.
 6. Create class templates and use them to solve many problems.

المحاضرة من المراجع :

[1]- Deitel & Deitel, C++ How to Program, Pearson; 10th Edition (February 29, 2016)

[2]- د.علي سليمان, البرمجة غرضية التوجه في لغة C++ 2009-2010

- إن القوالب هي واحدة من المزايا الأكثر تعقيداً وقوة في لغة C++.
- على الرغم من أنها ليست جزءاً من المحددات الأصلية للغة C++، إلا أنها أضيفت إليها منذ عدة سنوات وتم دعمها من قبل جميع مترجمات لغة C++ الحديثة .
- يمكنك باستخدام القوالب أن تقوم بإنشاء أصناف وتوابع عمومية **generic functions and classes**.
- التابع أو الصنف العمومي – نوع البيانات الذي يستخدمه التابع أو الصنف محددًا كبارامتر.
- يمكن استخدام تابع واحد أو صنف واحد مع أنواع مختلفة من أنماط البيانات دون الحاجة لكتابة نسخة محددة بشكل صريح لكل نوع بيانات.

Generic Functions

يعرف التابع العمومي مجموعة عامة من العمليات التي يمكن أن تطبق على أنواع مختلفة من البيانات. يتم تمرير نوع البيانات الذي سيستخدمه التابع إلى هذا التابع كبارامتر. وبالتالي فإنه في تابع عمومي، يتم استخدام

DataStructures Templates. إجرائية عامة وحيدة مع طيف واسع من أنواع البيانات.

وكما تعلم، فإن العديد من الخوارزميات هي نفسها منطقياً بغض النظر عن نوع البيانات التي تتعامل معه. على سبيل المثال، خوارزمية الفرز السريع quicksort algorithm هي نفسها سواء طبقت على مصفوفة من الأعداد الصحيحة أو مصفوفة من الأعداد ذات الفاصلة العائمة، وليس هناك أي اختلاف سوى نوع البيانات المراد فرزها.

يُنشأك لتابع عمومي، تستطيع تعريف طبيعة الخوارزمية بشكل منفصل عن نوع البيانات. بمجرد أن تقوم بذلك، سيقوم المترجم تلقائياً بتوليد الشيفرة المناسبة من أجل نوع البيانات المستخدم فعلياً لتنفيذ التابع. بكلام آخر، عندما تقوم بإنشاء تابع عمومي، فإنك تنشئ تابعاً يستطيع أن يقوم بتحميل نفسه بشكل زائد تلقائياً.

يتم إنشاء التابع العمومي باستخدام الكلمة المفتاحية `template`، وكما يشير معنى الكلمة الطبيعي، فهي تستخدم لإنشاء " قالب " أو إطار عمل `framework` يصف ما يقوم به التابع، تاركاً للمترجم أن يقوم بملء التفاصيل اللازمة.

Generic Functions

التوابع العمومية 2

يأخذ تعريف قالب التابع الشكل العام التالي :

```
template <class Ttype> ret-type func-name (parameter list)
{
// body of function
}
```

- حيث : **Ttype** هي مقبض استبدال لنوع البيانات المستخدم من قبل التابع، وهذا الاسم يمكن أن يستخدم في تعريف التابع، إلا أنه مجرد مقبض استبدال يقوم المترجم تلقائياً باستبداله بنوع البيانات الفعلي عندما يقوم بإنشاء نسخة محددة من التابع.
- على الرغم من أن استخدام الكلمة المفتاحية **class** لتحديد النمط العمومي في التصريح عن القالب، إلا أنك تستطيع استخدام الكلمة المفتاحية **typename** أيضاً.
- المثال التالي بإنشاء تابع عمومي يقوم بمبادلة قيمتي المتحولين الذين استدعي من أجلهما. وبما أن العملية العامة لمبادلة قيمتين مستقلة عن نوع البيانات للمتحولين، فإن إنشاء تابع عمومي للقيام بذلك هو الخيار الأفضل:

التوابع العمومية 3

ملف تعريف القالب `template` :

```
// Templates190.cpp : main project file.  
#include "stdafx.h"  
#include <iostream>  
using namespace std;  
//function template.  
template <class X> void swapargs(X &a, X &b)  
{  
    X temp;  
    temp = a;  
    a = b;  
    b = temp;  
}  
int main()
```

التوابع العمومية 4

```
{   int i=10, j=20;
    double x=10.1, y=23.3;
    char a='x', b='z';
    cout << "Original i, j: " << i << ' ' << j << "\n";
    cout << "Original x, y: " << x << ' ' << y << "\n";
    cout << "Original a, b: " << a << ' ' << b << "\n";
    swapargs(i, j); // swap integers
    swapargs(x, y); // swap floats
    swapargs(a, b); // swap chars
    cout << "Swapped i, j: " << i << ' ' << j << "\n";
    cout << "Swapped x, y: " << x << ' ' << y << "\n";
    cout << "Swapped a, b: " << a << ' ' << b << "\n";
    system("pause"); return 0;
}
```

التوابع العمومية 5

لنتمعن قليلاً في المثال السابق. يقوم السطر :

```
template <class X> void swapargs(X &a, X &b)
```

بإخبار المترجم بأمرين : أننا نقوم بإنشاء قالب وأن هذه بداية تعريف عمومي، حيث X هي نمط عمومي تم استخدامه كمقبض استبدال.

تم التصريح بعد الجزء `template` عن التابع `swapargs()` حيث تم استخدام X كنمط بيانات للقيم التي ستبادل.

تم في التابع `main()` استدعاء التابع `swapargs()` باستخدام ثلاثة أنماط بيانات مختلفة `int`، `double` و `char`. وبما أن `swapargs()` هو تابع عمومي، فإن المترجم يقوم تلقائياً بإنشاء ثلاث نسخ منه، الأولى ستقوم بمبادلة قيم صحيحة، الثانية ستقوم بمبادلة قيم ذات فاصلة عائمة والثالثة ستقوم بمبادلة قيم حرفية.

فيما يلي بعض المصطلحات الهامة المتعلقة بالقوالب، يدعى التابع العمومي (أي تعريف التابع الذي يسبق بالكلمة `template`) باسم قالب التابع `template function`. لدى قيام المترجم بإنشاء نسخة محددة من

ذلك التابع، يقال عندها أنه قد أنشأ تخصيصاً `specialization` ويدعى أيضاً تابع مولد `generated`

`function`. يقال عن عملية إنشاء تابع بأنها إنشاء مثل له `instantiating`، بمعنى آخر يقال عن التابع المنشأ على أنه مثل محدد من قالب التابع.

التوابع العمومية 6

يمكن كتابة التصريح السابق على سطرين منفصلين كما يلي :

```
template <class X>  
void swapargs(X &a, X &b)
```

وهذه هي الصيغة الأكثر استخداماً، بشرط عدم ورود أي تعليمات بينهما كما في الشكل الخاطئ التالي :

```
template <class X>  
int i; //this is an error  
void swapargs(X &a, X &b)
```

9-1-1- تابع يستخدم نمطين عموميين

A Function with Two Generic Types

يمكن تعريف أكثر من نوع بيانات عمومي في العبارة `template` باستخدام لائحة من الأنماط تفصل بينها فواصل. يقوم البرنامج التالي على سبيل المثال بإنشاء قالب تابع يحوي نوعين عموميين :

التوابع العمومية 7

```
#include "stdafx.h"
#include <iostream>
using namespace std;
template <class type1, class type2>
void myfunc(type1 x, type2 y)
{    cout << x << ' ' << y << '\n';}

int main()
{
    myfunc(10, "I like C++");
    myfunc(98.6, 19L);
system("pause");    return 0;
} // end main
```

التوابع العمومية 8

تم في المثال السابق استبدال مقابض الأنماط `type1` و `type2` من قبل المترجم بأنماط البيانات `int` و `char*` في الاستدعاء الأول، وأنماط البيانات `double` و `long` في الاستدعاء الثاني.

تذكره : عندما تقوم بإنشاء قالب تابع، فأنت في الأساس تقوم بالسماح للمترجم بتوليد نسخ مختلفة عديدة من ذلك التابع، وذلك بحسب ما تتطلب طرق الاستدعاء المختلفة لذلك التابع من قبل البرنامج.

9-1-2- التحميل الزائد الصريح للتابع العمومي

Explicitly Overloading a Generic Function

على الرغم من أن التابع العمومي يقوم بتحميل نفسه حسب الحاجة، إلا أن بإمكانك تحميله صراحة بشكل زائد، وهذا ما يدعى عادة بالتخصيص الصريح `explicit specialization`.

إذا قمت بالتحميل الزائد لتابع عمومي، فإن ذلك التابع يعيد تعريف التابع العمومي المرتبط بتلك النسخة. لندرس هذه النسخة المعدلة من برنامج تبادل القيم الذي قمنا بدراسته فيما سبق :

```
// Overriding a template function.
```

```
#include "stdafx.h"
```

```
#include <iostream>
```

```
using namespace std;
```

التوابع العمومية 9

```
template <class X> void swapargs(X &a, X &b)
{
    X temp;
    temp = a;
    a = b;
    b = temp;
    cout << "Inside template swapargs.\n";
}
// This overrides the generic version of swapargs() for ints.
void swapargs(int &a, int &b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
    cout << "Inside swapargs int specialization.\n";
}
```

```
int main()
{
    int i=10, j=20;
    double x=10.1, y=23.3;
    char a='x', b='z';
    cout << "Original i, j: " << i << ' ' << j << '\n';
    cout << "Original x, y: " << x << ' ' << y << '\n';
    cout << "Original a, b: " << a << ' ' << b << '\n';
    swapargs(i, j); //calls explicitly overloaded swapargs()
    swapargs(x, y); // calls generic swapargs()
    swapargs(a, b); // calls generic swapargs()
    cout << "Swapped i, j: " << i << ' ' << j << '\n';
    cout << "Swapped x, y: " << x << ' ' << y << '\n';
    cout << "Swapped a, b: " << a << ' ' << b << '\n';
    system("pause");
    return 0; } // end main
```

التوابع العمومية 11

يعطي هذا البرنامج على خرجه :

```
"C:\templates\Debug\templates.exe"  
Original i, j: 10 20  
Original x, y: 10.1 23.3  
Original a, b: x z  
Inside swapargs int specialization.  
Inside template swapargs.  
Inside template swapargs.  
Swapped i, j: 20 10  
Swapped x, y: 23.3 10.1  
Swapped a, b: z x  
Press any key to continue
```

التوابع العمومية 12

- كما تبين التعليقات الموضوعية ضمن البرنامج، فإنه عندما يتم إجراء الاستدعاء `swapargs(i,j)` فإنه ينفذ النسخة المحملة صراحة بشكل زائد من التابع `swapargs()` المعرفة في البرنامج، وبالتالي فإن المترجم لن يولد هذه النسخة من التابع العمومي `swapargs()` وذلك لأن التابع العمومي تم إعادة تعريفه من خلال التحميل الزائد الصريح.
- تم مؤخراً إضافة أسلوب جديد للتعبير عن التخصيص الصريح لتابع. هذه الطريقة الجديدة تستخدم الكلمة المفتاحية `template`.
- على سبيل المثال، يمكن كتابة التابع `swapargs()` المحمل بشكل زائد في المثال السابق باستخدام الأسلوب الجديد كما يلي :

```
// Use new-style specialization syntax.
```

```
template<> void swapargs<int>(int &a, int &b)  
{  
    int temp; temp = a; a = b; b = temp;  
    cout << "Inside swapargs int specialization.\n";}
```

التوابع العمومية 13

استخدمت الصيغة الجديدة التعليمة < > template للإشارة للتخصيص. وقد تم تحديد نوع البيانات الذي أنشئ التخصيص من أجله داخل قوسي زاوية من الشكل < > بعد اسم التابع. لقد تم استخدام هذه الصيغة نفسها لتخصيص أي نوع من التوابع العمومية.

إن التخصيص الصريح للقوالب يتيح لك تصميم نسخة من تابع عمومي بما يتوافق مع وضعية وحيدة، وربما يكون لذلك فائدة في بعض حالات الرغبة في زيادة الفعالية الناجمة عن استخدام نوع واحد من البيانات.

وكقاعدة عامة، إذا أردت استخدام نسخ مختلفة من تابع ما من أجل أنواع بيانات مختلفة فإن من الأفضل استخدام التوابع المحملة بشكل زائد بدلاً من القوالب.

Overloading a Function Template

بالإضافة إلى إنشاء نسخ محملة بشكل زائد بشكل صريح، فإنه من الممكن أيضاً القيام بالتحميل الزائد للمحدد template بحد ذاته. ويتم ذلك ببساطة بإنشاء نسخة أخرى من القالب تختلف جميع الأخرى في لائحة البارامترات الخاصة بها.
على سبيل المثال :

```
// Overload a function template declaration.  
#include "stdafx.h"  
#include <iostream>  
using namespace std;  
// First version of f() template.
```

```
template <class X> void f(X a)
{cout << "Inside f(X a)\n"; }
// Second version of f() template.
template <class X, class Y> void f(X a, Y b)
{cout << "Inside f(X a, Y b)\n"; }
int main()
{
    f(10); // calls f(X)
    f(10, 20); // calls f(X, Y)
    system("pause");return 0;
} // end main
```

تم في هذا المثال التحميل الزائد لقالب التابع f() لقبول بارمتر واحد أو بارامترين.

4-1-9- استخدام البارامترات القياسية مع قوالب التوابع

Using Standard Parameters with Template Functions

يمكن استخدام البارامترات القياسية إلى جانب البارامترات ذات الأنماط العمومية في قالب تابع. وتعمل هذه البارامترات غير العمومية تماماً كما تعمل في أي تابع آخر، على سبيل المثال :

```
// Using standard parameters in a template function.
```

```
#include "stdafx.h"
```

```
#include <iostream>
```

```
using namespace std;
```

```
const int TABWIDTH = 8;
```

```
// Display data at specified tab position.
```

```
template<class X> void tabOut(X data, int tab)
```

```
{
```

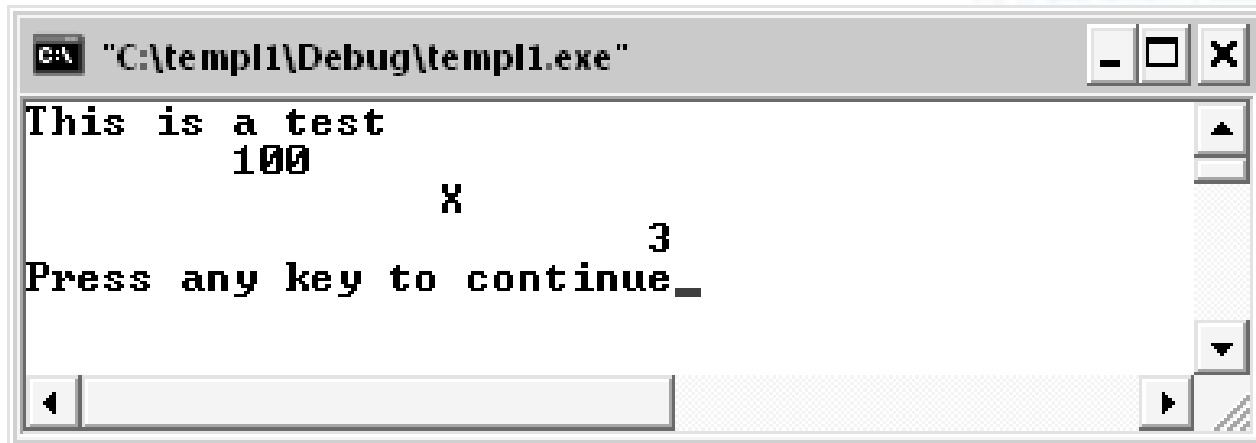
التوابع العمومية 16

```

for(; tab; tab--)
for(int i=0; i<TABWIDTH; i++) cout << ' ';
cout << data << "\n";}
int main()
{
tabOut("This is a test", 0);      tabOut(100, 1);
tabOut('X', 2);                  tabOut(10/3, 3);
system("pause");                  return 0;}// end main

```

وفيما يلي شكل الخرج الناتج عن تنفيذ البرنامج :



```

C:\temp1\Debug\templ1.exe
This is a test
      100
           X
                   3
Press any key to continue_

```

يقوم التابع `tabOut()` في هذا البرنامج بعرض وسيطه الأول على البعد المحدد من خلال وسيطه الثاني. بما أن الوسيط الأول هو من نوع عمومي، فإن التابع `tabOut()` يمكن استخدامه لعرض قيم من أي نوع.

التوابع العمومية 17

9-1-5- القيود على التوابع العمومية

Generic Function Restrictions

إن التوابع العمومية مشابهة للتوابع المحملة بشكل زائد ما عدا أنها أكثر تقييداً. عندما يتم التحميل الزائد للتوابع فقد يكون لديك أفعال مختلفة لتحقيقها ضمن جسم كل تابع، بينما التابع العام يجب أن يسلك السلوك العام ذاته من أجل جميع النسخ، والذي يمكن أن يتغير هو فقط نوع البيانات.

لندرس التوابع المحملة بشكل زائد في البرنامج التالي. هذه التوابع لا يمكن استبدالها بتابع عمومي لأنها لاتنفذ الشيء ذاته :

```
#include "stdafx.h"  
#include <iostream>  
#include <cmath>  
using namespace std;  
void myfunc(int i) {cout << "value is: " << i << "\n";}
```

```
void myfunc(double d)
{
    double intpart;           double fracpart;
    fracpart = modf(d, &intpart);
    cout<< "Fractional part: " << fracpart;cout << "\n";
    cout << "Integer part: " << intpart;
}
int main()
{
    myfunc(1);               myfunc(12.2);
    system("pause");        return 0;
} // end main
```

/* modf() function breaks the given argument into two parts, one is integer and the other one is fractional.*/

تعتبر التوابع العمومية واحدة من المزايا الأكثر فائدة في لغة C++. فهي يمكن أن تستخدم جميع أنواع المسائل، فأينما احتجنا لتابع يعرف خوارزمية قابلة للتعميم فإنه بالإمكان تعريفها في قالب تابع. وبمجرد قيامنا بذلك في إمكاننا استخدامها مع أي نوع بيانات دون الحاجة لذكره.

قبل الانتقال إلى الأصناف العمومية **generic classes** سنقوم بعرض مثالين لتطبيق التوابع العمومية، هذان المثالان يوضحان مقدار السهولة في الاستفادة من هذه الميزة الرائعة للغة C++.

9-2-1- عملية فرز عمومية

A Generic Sort

إن عملية الفرز هي من النمط المثالي للعمليات التي صممت من أجلها التوابع العمومية. وخوارزمية الفرز – إلى حد بعيد – هي نفسها بغض النظر عن نوع البيانات المراد فرزها.

يبين البرنامج التالي هذه العملية من خلال إنشاء عملية فرز فقاعي **bubble sort** عمومية. على الرغم من أن الفرز الفقاعي هي من خوارزميات الفرز الضعيفة إلا أنها واضحة وسهلة الفهم، يقوم التابع **bubble** بفرز أي نوع من المصفوفات، ويتم استدعاؤه بتمرير مؤشر إلى العنصر الأول في المصفوفة وعدد العناصر في المصفوفة:

```
// A Generic bubble sort.  
#include "stdafx.h"  
#include <iostream>
```

Applying Generic Functions



2-9- تطبيق التوابع العمومية

```
using namespace std;
template <class X> void bubble(
    X *items, // pointer to array to be sorted
    int count) // number of items in array
{
    register int a, b;          X t;
    for(a=1; a<count; a++)
        for(b=count-1; b>=a; b--)
            if(items[b-1] > items[b]) {
                // exchange elements
                t = items[b-1];
                items[b-1] = items[b];
                items[b] = t;
            }
}
```


Applying Generic Functions



2-9- تطبيق التوابع العمومية

```
int main()
{
    int iarray[7] = {7, 5, 4, 3, 9, 8, 6};
    double darray[5] = {4.3, 2.5, -0.9, 100.2, 3.0};
    int i;
    cout << "Here is unsorted integer array: ";
    for(i=0; i<7; i++)
        cout << iarray[i] << ' ';
    cout << endl;
    cout << "Here is unsorted double array: ";
    for(i=0; i<5; i++)
        cout << darray[i] << ' ';
    cout << endl;
}
```

Applying Generic Functions

```
bubble(iarray, 7);  
bubble(darray, 5);  
cout << "Here is sorted integer array: ";  
for(i=0; i<7; i++)  
    cout << iarray[i] << ' ';  
cout << endl;  
cout << "Here is sorted double array: ";  
for(i=0; i<5; i++)  
    cout << darray[i] << ' ';  
cout << endl;  
system("pause");    return 0;  
} // end main
```

Applying Generic Functions

2-9- تطبيق التوابع العمومية

لخرج الناجم عن البرنامج السابق :

```
C:\templ1\Debug\templ1.exe
Here is unsorted integer array: 7 5 4 3 9 8 6
Here is unsorted double array: 4.3 2.5 -0.9 100.2 3
Here is sorted integer array: 3 4 5 6 7 8 9
Here is sorted double array: -0.9 2.5 3 4.3 100.2
Press any key to continue_
```

كما ترى، فقد قام البرنامج السابق بإنشاء مصفوفتين، واحدة من الأعداد الصحيحة والثانية من النوع **double** ومن ثم يقوم بفرزهما. وبما أن التابع **bubble()** هو قالب تابع فإنه يتم تحميله تلقائياً بشكل زائد ليتوافق مع النوعين المختلفين من البيانات.

Compacting an Array

2-2-9- ضغط مصفوفة

تابع آخر يدعى `compact()` يستفيد من تعريفه على شكل قالب. يقوم هذا التابع بضغط عناصر المصفوفة، فعند الرغبة بحذف عناصر من منتصف مصفوفة ومن ثم نقل العناصر الباقية بحيث تكون المواقع غير المستخدمة في نهاية المصفوفة.

إن هذا الأسلوب من العمل هو نفسه من أجل جميع أنواع المصفوفات لأنه مستقل عن نوع البيانات التي يؤثر فيها.

تم استدعاء التابع العمومي `compact()` في البرنامج التالي - باستخدام مؤشر إلى العنصر الأول في المصفوفة وعدد عناصر المصفوفة وفهرس البداية والنهاية للعناصر المراد حذفها.

```
// A Generic array compaction function.  
#include <iostream>  
using namespace std;  
template <class X> void compact(  
    X *items, // pointer to array to be compacted
```

Applying Generic Functions



2-9- تطبيق التوابع العمومية

```
int count, // number of items in array
int start, // starting index of compacted region
int end) // ending index of compacted region
{
    register int i;
    for(i=end+1; i<count; i++, start++)
        items[start] = items[i];
    /* For the sake of illustration, the remainder of
    the array will be zeroed. */
    for( ; start<count; start++) items[start] = (X) 0;
}
int main()
{
    int nums[7] = {0, 1, 2, 3, 4, 5, 6};
```

Applying Generic Functions



2-9- تطبيق التوابع العمومية

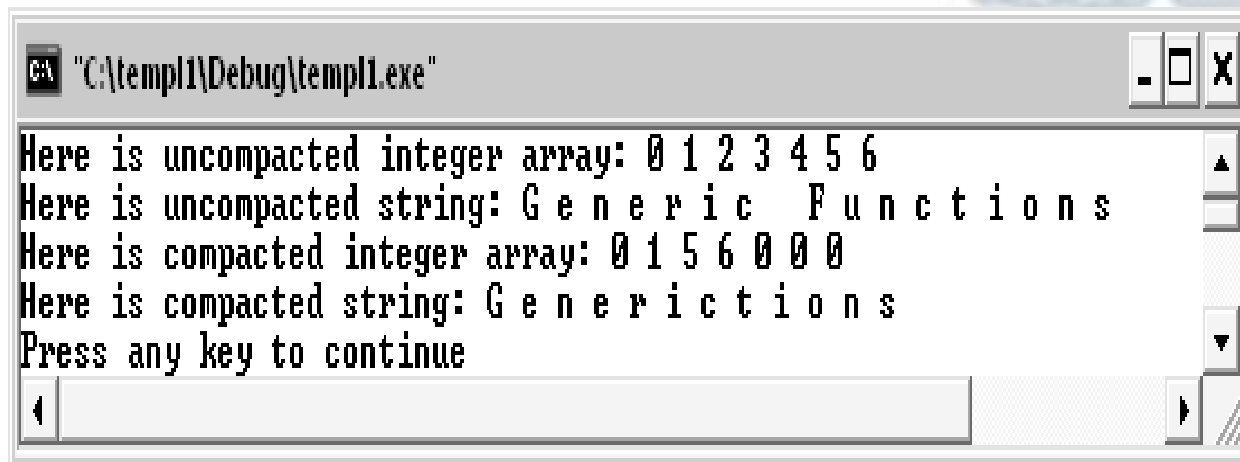
```
char str[18] = "Generic Functions";
int i;
cout << "Here is uncompact integer array: ";
for(i=0; i<7; i++)
    cout << nums[i] << ' ';
cout << endl;
cout << "Here is uncompact string: ";
for(i=0; i<18; i++)
    cout << str[i] << ' ';
cout << endl;
compact(nums, 7, 2, 4);
compact(str, 18, 6, 10);
cout << "Here is compact integer array: ";
```

Applying Generic Functions

2-9- تطبيق التوابع العمومية

```
for(i=0; i<7; i++)
    cout << nums[i] << ' ';    cout << endl;
cout << "Here is compacted string: ";
for(i=0; i<18; i++)
    cout << str[i] << ' ';    cout << endl;
return 0;}
```

يقوم هذا البرنامج بدمج نوعين مختلفين من المصفوفات، الأولى مصفوفة من الأعداد الصحيحة والأخرى مصفوفة سلاسل محرفية، ويكون خرج هذا البرنامج



```
"C:\templ1\Debug\templ1.exe"
Here is uncompactd integer array: 0 1 2 3 4 5 6
Here is uncompactd string: G e n e r i c   F u n c t i o n s
Here is compacted integer array: 0 1 5 6 0 0 0
Here is compacted string: G e n e r i c t i o n s
Press any key to continue
```

يمكن بالإضافة إلى تعريف التوابع العمومية أن نقوم بتعريف صنف عمومي. إن القيام بذلك يؤدي إلى إنشاء صنف يقوم بتعريف جميع الخوارزميات المستخدمة من قبل ذلك الصنف. في حين أن النوع الفعلي للبيانات التي ستتم معالجتها سيتم تحديده كبارامتر عند إنشاء أغراض من ذلك الصنف.

تكمّن فائدة الأَصْناف العمومية في الأَصْناف التي تستخدم منطقاً قابلاً للتعميم، على سبيل المثال، إن نفس الخوارزمية التي تتعامل مع رتل `queue` من الأعداد الصحيحة تعتبر صالحة من أجل رتل من المحارف، كما أن نفس الآلية التي تتعامل مع لائحة مترابطة `linked list` من العناوين البريدية ستتعامل مع لائحة مترابطة من قطع السيارات.

لدى القيام بتعريف صنف عمومي، فإن بإمكانه القيام بالعمل الذي تعرفه له كالتعامل مع رتل أو لائحة مترابطة من أي نوع من البيانات. وسيقوم المترجم تلقائياً بتوليد النمط الملائم للغرض، وذلك بحسب النمط الذي يتم تحديده عند إنشاء الغرض. تكون الصيغة العامة للتصريح عن الصنف العمومي من الشكل :

```
template <class Ttype> class class-name {
```

```
.  
. }  
}
```


Generic Classes

- حيث `Ttype` هي عبارة عن مقبض استبدال لاسم النوع، والذي سيتم تحديده عند تهيئة الصنف. ويمكن عند الضرورة - أن يتم تعريف أكثر من نوع بيانات عمومي باستخدام لائحة من الأنماط تفصل بينها فواصل. بمجرد أن يتم إنشاء صنف عمومي، يمكن إنشاء مثل `instance` محدد من ذلك الصنف باستخدام الصيغة العامة التالية :

```
class-name <type> ob;
```

حيث `type` هي اسم نمط البيانات الذي سيؤثر فيه الصنف. تكون التوابع الأعضاء بحد ذاتها توابع عمومية تلقائياً ولانحتاج لأن نستخدم العبارة `template` لتحديدها بشكل صريح. يتضمن البرنامج التالي الصنف `stack` معرّفاً كصنف عمومي، وبالتالي يستطيع تخزين أغراض من أي نوع. تم في هذا المثال إنشاء مكّس من المحارف ومكّس من القيم ذات الفاصلة العائمة (كمثال).

```
// This function demonstrates a generic stack.  
#include <iostream>  
using namespace std;  
const int SIZE = 10;  
// Create a generic stack class  
template <class StackType> class stack {
```

Generic Classes



3-9- الأصناف العمومية

```
StackType stck[SIZE]; // holds the stack
int tos; // index of top-of-stack

public:
    stack() { tos = 0; } // initialize stack
    void push(StackType ob); // push object on stack
    StackType pop(); // pop object from stack
};
// Push an object.
template <class StackType> void stack<StackType>::push(StackType ob)
{
    if(tos==SIZE) {
        cout << "Stack is full.\n";
        return;
    }
    stck[tos] = ob;
    tos++;
}
```

Generic Classes



3-9- الأصناف العمومية

```
// Pop an object.
template <class StackType> StackType stack<StackType>::pop()
{
    if(tos==0) {
        cout << "Stack is empty.\n";
        return 0; // return null on empty stack
    }
    tos--;
    return stck[tos];}
int main()
{
    // Demonstrate character stacks.
    stack<char> s1, s2; // create two character stacks
    int i;
    s1.push('a');      s2.push('x');
    s1.push('b');      s2.push('y');
    s1.push('c');      s2.push('z');
```

```
for(i=0; i<3; i++) cout << "Pop s1: " << s1.pop() << "\n";  
for(i=0; i<3; i++) cout << "Pop s2: " << s2.pop() << "\n";  
// demonstrate double stacks
```

```
stack<double> ds1, ds2; // create two double stacks  
ds1.push(1.1);          ds2.push(2.2);  
ds1.push(3.3);          ds2.push(4.4);  
ds1.push(5.5);          ds2.push(6.6);  
for(i=0;i<3;i++) cout << "Pop ds1: " << ds1.pop() << "\n";  
for(i=0;i<3;i++) cout << "Pop ds2: " << ds2.pop() << "\n";  
return 0;}
```

كما تلاحظ، فإن التصريح عن الصنف العمومي شبيه بالتصريح عن التابع العمومي. ويكون نوع البيانات الفعلي المخزن في المكس عمومياً ضمن التصريح عن الصنف، ولا يتم تحديده إلى أن يتم إنشاء غرض من ذلك المكس.

Generic Classes

3-9- الأصناف العمومية

عندما يتم التصريح عن مثل محدد من الصنف `stack`، فإن المترجم يولد تلقائياً جميع التوابع والمتحولات الضرورية للتعامل مع البيانات الفعلية.

تم في المثال السابق التصريح عن نمطين مختلفين من المكذسات، مكذسان من الأعداد الصحيحة ومكذسان من الأعداد مضاعفة الدقة `double`، من خلال التصريحين التاليين :

```
stack<char> s1, s2; // create two character stacks
```

```
stack<double> ds1, ds2; // create two double stacks
```

لاحظ كيف تم تمرير نمط البيانات المرغوب داخل قوسين زاويين. بتغييرك لنوع البيانات المحدد عند إنشاء أغراض من الصنف `stack` يكون بإمكانك تغيير نمط البيانات المخزن في ذلك المكذس، فعلى سبيل المثال، يمكن من خلال التصريح التالي إنشاء مكذس آخر يقوم بتخزين مؤشرات إلى محارف :

```
stack<char *> chrptrQ;
```

كما يمكن إنشاء مكذسات لتخزين أنماط بيانات تقوم بإنشائها، على سبيل المثال، إذا أردنا استخدام السجل التالي لتخزين معلومات العنوان :

Generic Classes

```
struct addr
{
    char name[40];
    char street[40];
    char city[30];
    char state[3];
    char zip[12];
};
```

ومن ثم لاستخدام الصنف `stack` لإنشاء مكدس لتخزين أغراض من النمط `addr` يمكن استخدام تصريح من الشكل :

```
stack<addr> obj;
```

كما هو ملاحظ من خلال الصنف `stack` فإن التوابع والأصناف العمومية هي أدوات فعالة يمكن استخدامها لزيادة الفعالية البرمجية فهي تمكن المبرمج من تعريف الشكل العام لغرض يمكن أن يستخدم فيما بعد مع أي نوع بيانات.

انتهت تمارين الأسبوع العاشر

