المحاضرة الثامنة

دوال التحويل و التحميل الزائد للدوال

دوال التحويل Conversion Functions

تعريف دالة التحويل :

هي عبارة عن دالة تحمل بشكل زائد ووظيفتها التحويل من نوع معين إلى نوع آخر .

حيث يوجد لدينا نوعان من البيانات :

1. الأنواع الأساسية : مثل : **int , float , char , double ....**
2. الأنواع المعرفة : وهي الكائنات باعتبار أنها معرفة من قبل المستخدم

و توجد ثلاثة أنواع من التحويلات هي :-

1. التحويل من نوع أساسي إلى نوع معرف من قبل المستخدم .

مثلاً : إذا كان لدينا متغير صحيح وليكن **x** وكائن **m** من صنف معين إذا كتبنا العبارة التالية :

m = x ;

فهي تعني تحويل نوع أساسي وهو **x** إلى نوع معرف من قبل المستخدم وهو **m**

مثال : هل يمكن تنفيذ البرنامج التالي :

```
class    myclass
{
        ................
        ................
        ................
};
int int main ( )
{
myclass  c1 ;
int   x = 10 ;
*   -c1  =   x ;
return 0;
}
```

☑	لا يمكن تنفيذ البرنامج أعلاه لأن العبارة (*) خاطئة لأنها غير معرفة ولا يمكن التحويل بين عدد صحيح وكائن ولجعل العبارة السابقة صحيحة يجب تعريف الطريقة التي سيستخدمها المترجم لإجراء هذه العملية . يمكن إنجاز هذه العملية باستخدام دالة بناء تستقبل وسيطة واحدة .

مثال : برنامج يقوم بالتحويل من نوع أساسي إلى نوع معرّف

```
#include<iostream.h>
class  Lmeters
{
private :
int   meter;
public :
Lmeters ( )
{
meter  = 0 ;
}
** Lmeters (int  n )
{
meter  = n ;
}

void display ( )
{
cout<< meter << "\n" ;
}
}   ;
int main ( )
{
Lmeters m1 ;
m1.display ( );
int   x ;
cout << "Enter x  \n " ;
cin >> x ;
m1 = x ;
m1.display () ;
return 0;
}
```

☑	لاحظ العبارة **m1 = x** أسندنا فيها نوع أساسي (**int**) إلى نوع معرف (كائن) في هذا المثال ينظر المترجم إلى الصنف فإذا وجد دالة بناء تستدعي وسيطة واحدة وهي الدالة المشار إليها بـ ** تعتبر التعليمة صحيحة حيث تعتبر العبارة (**m1= x**) هي استدعاء دالة بناء ذات وسيطة واحدة أي بمعنى آخر أن : **m1= x ≡ m1 (x)** إذا لم يجد المترجم دالة بناء بوسيطة واحدة تعتبر العبارة **m1= x** خاطئة وتظهر رسالة خطأ

2. التحويل من نوع معرف من قبل المستخدم إلى نوع أساسي

مثلاً إذا كان لدينا متغير حقيقي وليكن **y** وكائن **n** من صنف معين إذا كتبنا العبارة التالية **y = n** .

فهي تعني تحويل نوع معرف من قبل المستخدم **n** إلى نوع أساسي وهو **y**

مثال : .هل يمكن تنفيذ البرنامج التالي

```
class     myclass
{
        ................
        ................
        ................
};

int main ( )
{
myclass  c1;
float   x = 12.5;
* - x  =   c1 ;
return 0;
}
```

☑	لا يمكن تنفيذ البرنامج أعلاه لأن العبارة (*) خاطئة لأنها غير معرفة ولا يمكن التحويل بين نوع معرف (كائن) ونوع أساسي (عدد حقيقي) ولجعل العبارة السابقة صحيحة نقوم بتحميل النوع الأساسي (**float**) تحميلاً زائداً حتى يتم تحويل النوع المعرف **c1** إلى النوع الأساسي **x**

مثال : برنامج يقوم بالتحويل من نوع معرّف إلى نوع أساسي :

```
#include<iostream.h>
class  Lmeters
{
private :
float   meter;
public :
   Lmeters ( )
{
meter  = 0.0  ;
}
void display ( )
{
cout<< meter << "\n" ;
}
operator float ()
```

```
{
return meter ;
}
}   ;
int main ( )
{
Lmeters   m1 ;
m1.display ( ) ;
float  x ;
cout << "Enter x \n " ;
cin >> x ;
x = float ( m1 ) ;  // x = m1 ;
cout<< x << " \n " ;
return 0;
}
```

☑	باستخدام العبارة **x = float (m1)** تكون قد حولنا **m1** من النوع **Lmeters** إلى النوع **float** عن طريق دالة التحميل الزائد **operator float –( )**

3. التحويل من نوع معرف من قبل المستخدم إلى نوع آخر معرف من قبل المستخدم

مثلاً إذا كان لدينا صنفان الصنف الأول **class1** وبه الكائن **c1** والصنف الثاني **class2** وبه الكائن **c2** .

إذا كتبنا العبارة (*) **c2 = c 1**

فهي تعني تحويل **c1** إلى **c2** وكلاهما كائنين معرفان من قبل المستخدم ولكن من صنفين مختلفين . فهذه العبارة تعتبر خاطئة .

☑	**ملاحظة مهمة :** إذا كان الكائنين **c1** و **c2** من نفس الصنف فالعبارة **c2 = c1** تعتبر صحيحة وتعني نسخ محتويات الكائن **c1** في الكائن **c2** . ولجعل العبارة (*) صحيحة **c2 = c1** كائن من صنف المصدر كائن من صنف المستودع

نستخدم إحدى الطرق التالية :

1. وجود دالة بناء ناسخة في الصنف المستودع تستقبل من صنف المصدر .
2. تحميل الصنف المحول إليه (صنف المستودع) تحميلاً زائداً وتكون دالة التحميل الزائد موجودة في (صنف المصدر) .

مثال : تحويل نوع معرف من نوع معرف باستخدام دالة بناء ناسخة حيث يقوم البرنامج بتحويل الأقدام والبوصات إلى أمتار .

```
#include<iostream.h>
class  Lfeets
{
private :
int feets , inches ;
public :
   Lfeets  ( )  // default constructor                    دالة بناء افتراضية
{
feets = 0;
inches = 0 ;
}
//constructor function two arguments
//  دالة بناء تستقبل وسيطتين
Lfeets ( int f , int i )
{
feets  =  f  ;
inches  = i  ;
}
void display ( )
{
cout << feets << ". " << inches  << " \n" ;
}
int  get_feets (    )
{
return feets ;
}
int get_inches ( )
{
return inches ;
}
}  ;
class Lmeters
{
```

```
private :
float meter ;
public :
Lmeters (   )
{
meter = 0.0 ;
}
void display (   )
{
cout << meter <<" \n " ;
}
// تعريف دالة بناء ناسخة تقوم بتحويل الأقدام والبوصات إلى أمتار
Lmeters (Lfeets  lf )
{
float i ;
// تحويل الأقدام والبوصات إلى سنتمترات
i=( lf.get_feets( ) * 30.48+ lf .get_inches()* 2.5 ) ;
// تحويل السنتمترات إلى أمتار .
i = i/100;
meter = i ;
}
} ;
int main ( )
{
int       i  , f ;
cout <<"Enter feets \n" ;
cin >> f ;
cout <<"Enter inches \n" ;
cin >> i   ;
Lfeets  f1 ;
Lfeets f2 ( f, i)  ;
f2.display ( ) ;
Lmeters  m1 ;
m1.display ( ) ;
m1=f1;
m1.display ( );
m1= f2;
m1.display ( ) ;
return 0;
}
```

إذا كانت f= 24 ، i = 10

مخرج البرنامج هو :

```
24.10
0.0
0.0
7.5652
```

☑	في صنف الأقدام **Lfeets** أضفنا دالتي **get-feets ( )** و **get_inches( )** للحصول على الأقدام والبوصات حتى نصل عن طريق صنف الأمتار **L meters** إلى الخصائص المحلية في صنف الأقدام **feets** و **inches** .

مثال : برنامج يقوم بتحويل الأقدام والبوصات إلى أمتار (نوع معرف إلى نوع معرف) بالتحميل الزائد :

```
#include<iostream.h>
class    Lmeters
{
private :
float  meter ;
public :
Lmeters ( )
{
meter  =  0.0  ;
}
void display ( )
 {
cout << meter << "\n" ;
}
};
class Lfeets
{
private :
int  feets , inches ;
public  :
Lfeets (  )
```

```
{
feets  =  0 ;
inches  = 0 ;
}
Lfeets ( int  f , int i )
{
feets  = f ;
inches  = i ;
}
void display (   )
{
cout <<  feets << "."  << inches << " \n"  ;
}
// تحميلاً زائداً  Lmeters  نحمل الصنف

// Lfeets  داخل الصنف
operator Lmeters (   )
{
f loat i ;
i = ( feets * 30.48 + inches * 2.5 ) ;
i  = i / 100 ;
return  Lmeters ( i ) ;
 }
} ;
int main ( )
{
int     f , i  ;
cout <<"Enter feets \n" ;
cin>>f ;
cout <<"Enter inches \n";
cin >> i ;
Lfeets f1 ,  f2( f,i)  ;
f1.display (  ) ;
f2.display ( ) ;
Lmeters m1 ;
m1.display ( ) ;
m1=f1;
m1.display ( );
m1= f2;
m1.display ( ) ;
return 0;
}
```

☑	1. لم نستخدم دالتي **get_feets ( )** و **get_ inches ( )** كما في دالة البناء الناسخة لان عملية تحويل الأقدام والبوصات إلى أمتار تمت داخل صنف الأقدام وبالتالي يتم الوصول إلى المتغيرات **feets** و **inches** مباشرة لأننا في نفس الصنف . 2. في دالة التحميل الزائد استخدمنا الطريقة الثالثة من طرق إرجاع كائن من دالة التحميل وهي استخدام كائن من غير اسم .

التحميل الزائد للدوال Functions Overloading

تعريف التحميل الزائد للدوال

هو إمكانية إنشاء أكثر من دالة تحمل نفس الاسم ولكنها قد تختلف في نوعية المعاملات **Parameters Types** التي تستخدمها الدالة أو في عدد المعاملات **Parameters Numbers** ولكل دالة من هذه الدوال الإجراء الخاص بها أي أنه قد تشترك الدوال في اسمها ولكنها قد تختلف في شكلها ومضمونها .

مثال :-

برنامج يقوم بإجراء عملية جمع يحتوي على عدد من الدوال تحمل نفس الاسم ولكنها تختلف في نوعية المعاملات وعددها .

```
#include<iostream.h>
class  summation
{
public :
int a, b, c ;
float  x, y ;
int sum ( int , int ) ;
float sum ( float , float ) ;
int sum ( int , int, int) ;
}  ;
int  summation :: sum ( int a , int b )
{
return a+ b ;
}
float  summation :: sum ( float x , float y )
{
return x + y ;
}
int  summation :: sum ( int a, int b, int c )
{
return  a + b + c ;
}
```

```
int main ( )
{
summation ob ;
cout << "Enter a , b , c \ n " ;
cin >> ob .a >> ob.b >> ob.c ;
cout <<"Enter x , y \n " ;
cin >> ob.x >> ob.y ;
cout << " sum1=”  << ob.sum ( ob.a  ,  ob.b ) << " \n"  ;
cout << " sum2=”  << ob.sum ( ob.x ,  ob.y ) << " \n" ;
cout << " sum3 =”<< ob.sum ( ob.a  ,  ob.b , ob.c )  << " \n" ;
return 0;
}
```