بسم الله الرحمن الرحيم

Object Oriented Programming

البرمجة الشيئية

عدد الساعات: ٢ نظري + ٢ عملي

المتطلبات السابقة: ٢١١حسب

أ/المادة: م. نجلاء حسن

Lecture 7

-التحميل الزائد للمعاملات overloading operators

التحميل الزائد (الاستنساخ) للمعاملات Operator Overloading

تطلق عملية التحميل الزائد للمعاملات على اعطاء معاملات C++ العادية مثل + و* و =< و =+ بعض العمليات الاضافية عند تطبيقها على انواع البيانات المعرفة بواسطة المستخدم (كالكائنات) . أي يتم استنساخ المعامل للحصول على معامل جديد قادر على تنفيذ نفس وظيفته العادية ولكن مع انواع بيانات مركبة.

فائدة التحميل الزائد للمؤثرات :-

استخدام الطريقة أو الأسلوب المستخدم في التعامل مع الأنواع المعرفة مسبقاً (الأنواع الأساسية) في التعامل مع الأنواع المعرفة من قبل المستخدم (الكائنات) مثلاً : يمكن أن نقوم بتطبيق المؤثر الرياضي (+) [مؤثر الجمع] على عددين صحيحين أو حقيقيين (أنوع أساسية) ولكن لا يمكن أن نطبق هذا المؤثر على كائنين(أنواع معرفة) إلا بعد تحميل المؤثر (+) تحميلاً زائداً .

شروط التحميل الزائد للمؤثرات :-

1ـ لا ينبغي تغيير وظيفة المؤثر عند التحميل مثلاً إذا قمنا بتحميل المؤثر (+) تحميلاً زائداً فلا ينبغي أن تستخدمه في أجراء عملية أخرى مثل (- أو * ..... الخ) .

2ـ لا ينبغي تغيير أولوية المؤثر عند التحميل .

- يتم استخدام الكلمة الاساسية operator لتحميل المعامل كي يعمل مع انواع البيانات المعرفة من قبل المستخدم .

الصيغة العامة لدالة التحميل الزائد للمعاملات:

```
datatype  operator operatorSymbol ( Parameters)

    {
      //statements
    }
```

حيث:

datatype: القيمة الراجعة من الدالة (غالباً ما ترجع كائناً تابعاً للفئة التي تعمل على كائناتها ولكن يمكن أن يكون return_type من أي نوع آخر.)

operator: وهي كلمة مفتاحية Keyword

operatorSymbol: وهو رمز المعامل (... , * , +)

Parameters :وهى لائحة الوسيطات الممررة إلى الدالة

statements :تعليمات الدالة.

١- تحميل المعاملات الاحادية:Uniary Operator

المعامل الاحادي : هو المعامل الذي يعمل على عنصر (متغير) واحد فقط وهي:

- معامل التزايد: (++) Increment
- معامل التناقص (__) Decrement

مثال ١ :

انشاء صنف counter يعمل على زيادة قيمة الكائن (C1) باستخدام المعامل ++

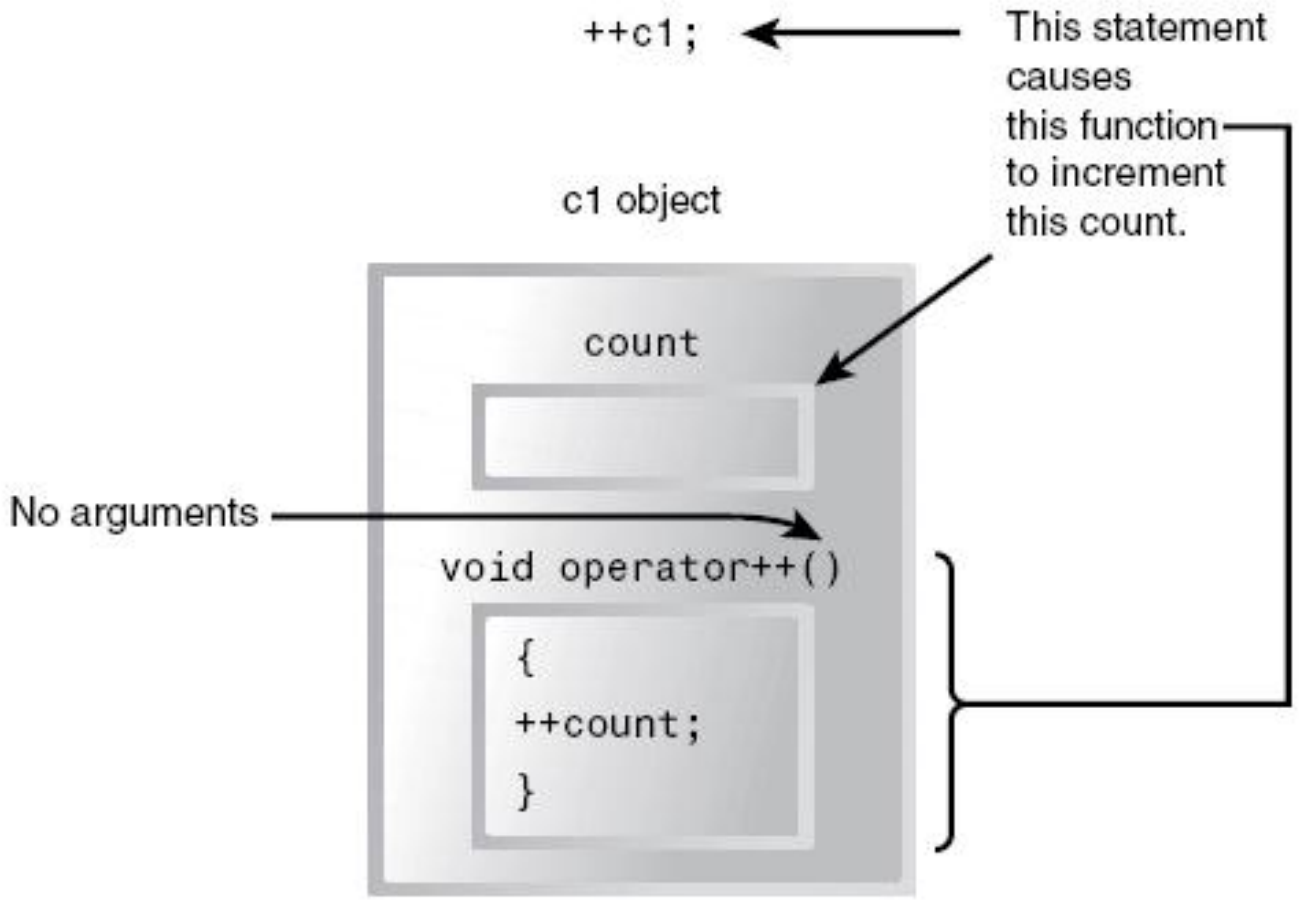
```
++c1;
```

```
#include<iostream.h>
class counter
{
private:
int count;
public:
counter ():count(0){}\\دالة بناء
int get_count()
{return  count;}

void operator++()// (prefix) زيادة قبلية
{ ++ count;}

};
int main()
{
 counter C1;//اشتقاق كائن
 cout <<C1.get_count()<<"\n";//طباعة
 ++C1;  //  زيادة قيمة الكائن
cout <<C1.get_count();//طباعة قيمة الكائن بعد الزيادة
return 0;
}
```

دالة التحميل الزائد


++c1;
This statement causes this function to increment this count.
c1 object
count
No arguments
void operator++()
{
++count;
}

ارجاع دالة التحميل الزائد للقيم:

```
class counter
{
private:
int count;
public:
counter ():count(0){}
int get_count(){return count;}

counter operator++ ( )
{ ++ count;
  counter  temp; //(temporary) انشاء كائن مؤقت
  ) temp.count=count;
  return temp;}
};
void main()
{
 counter C1,C2;
 cout <<C1.get_count( )<<"\n";
 cout <<C2.get_count( )<<"\n";
 ++C1;
C2=++C1;
cout <<C1.get_count( )<<"\n";
cout <<C2.get_count( );}
```

- اذا استخدامنا العبارة C2=++C1 داخل البرنامج السابق سيعترض المترجم لأننا حددنا اثناء تعريف دالة التحميل الزائد لا تقوم بارجاع أي بيانات ، وبالتالي لا يمكن استخدام المعامل ++ في عبارة تخصيص وانما يستخدم مع الكائن فقط .
- لكي نتمكن من استخدامه مع عبارات التخصيص يجب اعادة تعريف الدالة لكي ترجع قيمة. والقيمة عبارة عن كائن .(يتم اسنادها الى متغير(كائن) على يسار معامل الاسناد =)

في المثال السابق : تقوم دالة التحميل الزائد operator++ بزيادة عنصر البيانات count كما في البرنامج الأول ثم انشاء كائن جديد من النوع counter باسم **temp** لاستخدامه كقيمة مرتجعة من الدالة. و تخصيص عنصر البيانات count به بالقيمة count الجديدة واخيرا ارجاع الكائن الي البرنامج .وبالتالي يقوم التعبير C1++ بارجاع قيمة وبالتالي يمكن استخدامه في عبارات التخصيص كالعبارة : C2=++C1;

يمكن لدالة التحميل الزائد أن ترجع كائن باستخدام:

- كائن مؤقت باسم Temporary object: كما في المثال السابق

```
counter operator++ ( )
{ ++ count;
  counter  temp; //(temporary)انشاء كائن  مؤقت باسم
 temp.count=count;
  return temp;}
```

- كائن مؤقت من غير اسم Names less object :

```
counter operator++ ( )
{ ++ count;
  return counter  (count); //كائن من غير اسم
}
```

تقوم العبارة بإنشاء كائن مؤقت بدون اسم مع تخصيص القيمة الى هذا الكائن. ولكن تحتاج الى دالة بناء تحتوي على معامل واحد.

```
Counter (int c):count(c){}
```

- كتابة احد الطريقتين في البرنامج السابق تعطي نفس النتيجة.;

- يمكن استخدام نفس الأفكار السابقة في استنساخ معامل الانقاص (--)

٢- تحميل المعاملات الثنائية Binary Operator:

المعامل الثنائي هو المعامل الذي يعمل على عنصرين (متغيرين) مثل المعاملات الحسابية مثل الجمع (+)ومعاملات المقارنة: != , == , <= , >= , < , >

ومعاملات التخصيص الحسابية += , -= , *= , /= , %=

استنساخ المعاملات الحسابية:

مثال: استنساخ معامل الجمع الحسابي +:

صممي برنامج بلغة C++ يحتوى على:

-صنف Distance الذي يمثل المسافة بالنظام الانجليزي في صورة قدم feet وبوصة inches .

يقوم البرنامج باضافة المسافات (كائنات الصنف Distance) للحصول على مسافة جديدة (كائن) .أي تحميل المعامل +تحميل زائد

الحل

```
#include<iostream.h>
class Distance
{
private:
int feet;
float inches;
public:
Distance ():feet(0), inches(0.0){}
Distance (int f, float in):feet(f), inches(in){}
void getdist()
{cout<<"enter feet:"<<endl;   cin>>feet;
 cout<<"enter inches:"<<endl;  cin>>inches; }
 void showdist()const
{cout<< feet<<"-"<<inches<<endl; }
Distance operator+(Distance  )const;
};
Distance Distance ::operator+(Distance
d2)const
{ int f=feet+d2.feet;
  float i=inches+d2.inches;
      if(i>=12.0)
        {
             i-=12.0;
             f++;
         }
        return Distance(f,i);}
```

```
void main()
{
 Distance dist1,dist3,dist4;
 dist1.getdist();
 Distance dist2(11,6.25);
 dist3 = dist1+  dist2;
 dist4 = dist1+  dist2+dist3 ;
 cout <<"dist1=";  dist1.showdist();
 cout <<"dist2=";  dist2.showdist();
 cout <<"dist3=";  dist3.showdist();
 cout <<"dist4=";  dist4.showdist();
 }
```

```
(Inactive C:\TCWI
enter feet:
10
enter inches:
6.5
dist1=10:6.5
dist2=11:6.25
dist3=22:0.75
dist4=44:1.5
```

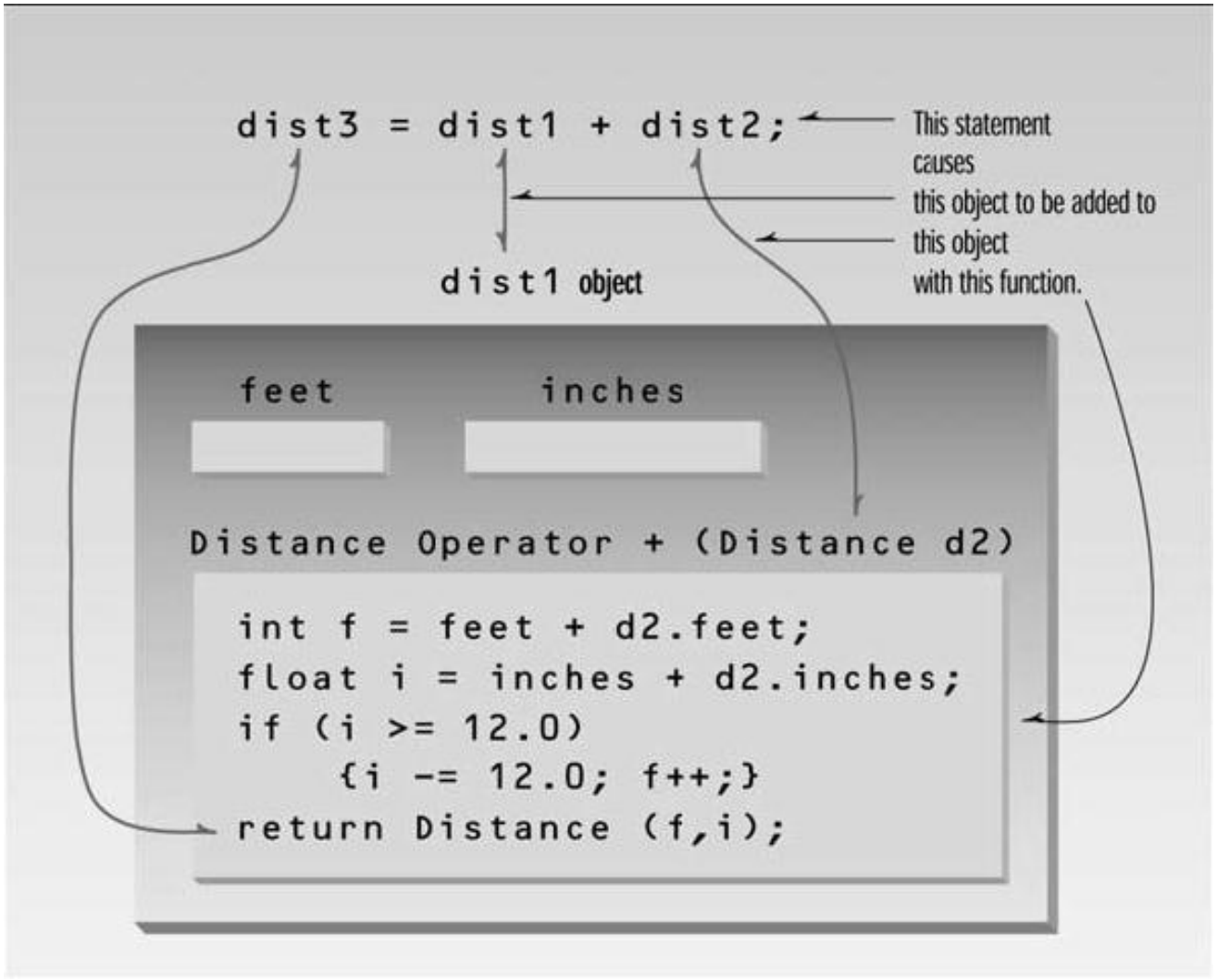
في المثال السابق :

دالة التحميل الزائد: **Distance operator+(Distance)const;**

تقوم بارجاع قيمة من النوع Distance وتأخذ معامل واحد من النوع Distance.

ما هو معامل (وسيط) الدالة dist1 أم dist2 ؟ وهل لا تحتاج الدالة الى معاملين حتى تتمكن من جمع الكائنين؟؟

(على الرغم من أنها تقوم بتحميل عامل الجمع + الثنائي الذي يعمل على قيمتين (dist3 = dist1+ dist2;


dist3 = dist1 + dist2;
This statement causes this object to be added to this object with this function.
dist1 object
feet
inches
Distance Operator + (Distance d2)
int f = feet + d2.feet;
float i = inches + d2.inches;
if (i >= 12.0)
{i -= 12.0; f++;}
return Distance (f,i);

- لتعميم القاعدة ، يجب أن يحتوى المعامل المستنسخ (دالة التحميل)على عدد من الوسائط اقل من العناصر التي يطبق عليها بمقدار واحد، حيث يكون احد هذه العناصر الكائن الذي ينتمي اليه المعامل. وهذا سبب عدم احتواء المعاملات الاحادية المستنسخة على اية وسائط.
- يمكن تعريف الدالة التي تعمل على تحميل عامل بشكل زائد في صنف ما كعضو في الصنف (اذا مررنا اليها معامل واحد)

مثل : Distance operator+(Distance d);

و **كدالة صديقة** للصنف (يتم تمرير وسيطتين) مثال:

friend Distance operator* (const **Distance & d1,** const **Distance &d2**);

- التمرير يكون بالقيمة أو بالإشارة & والأفضلية للتمرير بالإشارة و تكون من النوع const.

& , Const اذا لم نضعهم لم يسبب أي مشكلة.

- يمكن استخدام نفس الأفكار السابقة في استنساخ بقية المعاملات الحسابية حتى تتمكن من ضرب وقسمة وطرح قيم كائنات الصنف Distance كما لو كانت متغيرات عادية.

استنساخ معاملات المقارنة:

مثال: استنساخ معامل المقارنة «أقل من» <:

صممي برنامج بلغة C++ يحتوى على:

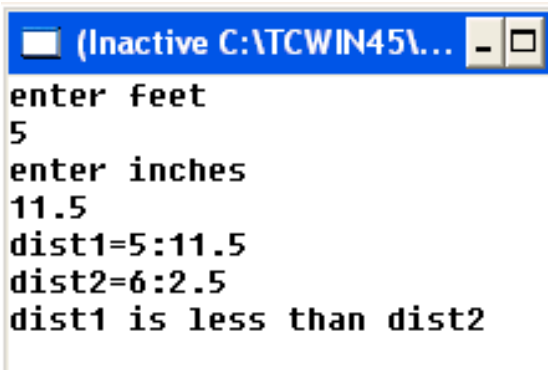
-صنف Distance الذي يمثل المسافة بالنظام الانجليزي في صورة قدم feet وبوصة inches .

قومي باستنساخ معامل المقارنة **«أقل من» < داخل الصنف Distance لمقارنة مسافتين (كائنين)**

الحل

```
# include<iostream.h>
class Distance
{
private:
int   feet;
float inches;
public:
Distance():feet(0),inches(0.0){ }
Distance(int f,float in):feet(f),inches(in){ }
//Distance(int f, float in){feet=f; inches=in;}
void getdist()
{
cout<<"enter feet \n";   cin>>feet;
cout<<"enter inches \n";  cin>>inches;
}
void showdist()
{cout<< feet<<":"<<inches<<endl; }
int  operator<(Distance );
 };
 int Distance::operator<(Distance d2)
{
  float  bf1  = feet+(inches/12);
  float  bf2  = d2.feet+(d2.inches/12);
  return (bf1<bf2)?1:0;}
```

```
int main()
{
 Distance dist1;
 dist1.getdist();
 Distance dist2(6,2.5);
 cout <<"dist1=";  dist1.showdist();
 cout <<"dist2=";  dist2.showdist();
 if(dist1<dist2)
 cout <<"dist1 is less than dist2\n";
  else
  cout <<"dist1 is greater than dist2\n";
  return 0;
 }
```



استنساخ معاملات التخصص الحسابية:

مثال: استنساخ معامل التخصيص =+ :

صممي برنامج بلغة C++ يحتوى على:

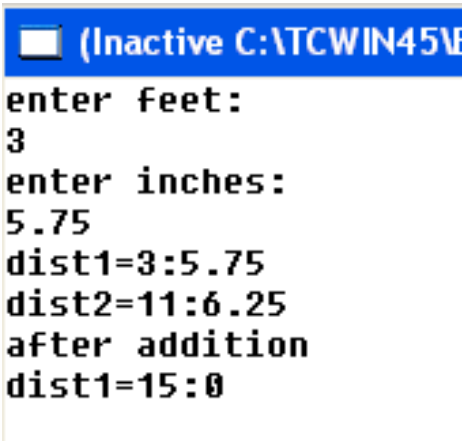
-صنف Distance الذي يمثل المسافة بالنظام الانجليزي في صورة قدم feet وبوصة inches .

قومي باستنساخ معامل التخصيص =+ والذي يقوم بجمع مسافتين (كائنين من نوع الصنف Distance) مع تخصيص الناتج للكائن الأول.(عمليتي الجمع والتخصيص في خطوه واحدة.)

الحل

```
#include<iostream.h>
class Distance
{
private:
int feet;
float inches;
public:
Distance ():feet(0), inches(0.0){}
Distance (int f, float in):feet(f), inches(in){}
void getdist()
{cout<<"enter feet:"<<endl;   cin>>feet;
 cout<<"enter inches:"<<endl;
cin>>inches; }
 void showdist()const
{cout<< feet<<":"<<inches<<endl; }
void operator+=(Distance);
};
void Distance ::operator+=(Distance d2)
{ feet+=d2.feet;
  inches+=d2.inches;
            if(inches>=12.0)
            {
              inches-=12.0;
              feet++;
              }
            }
```

```
 void main()
{
 Distance dist1;
 dist1.getdist();
 cout <<"dist1=";  dist1.showdist();
 Distance dist2(11,6.25);
 cout <<"dist2=";  dist2.showdist();
 dist1+= dist2;
 cout<<"after addition \n";
 cout <<"dist1=";  dist1.showdist();
 }
```



ما هو الفرق بين الدالة operator+=() والدالة operator+()؟؟

لا يمكن تحميل كل عوامل C++ بشكل زائد ، فمثلاً العوامل التالية لا يمكننا تحميلها :

:? (المعامل الشرطي)

:: (معامل النطاق)

. (معامل النقطة)

The End