

# Overloading and Templates

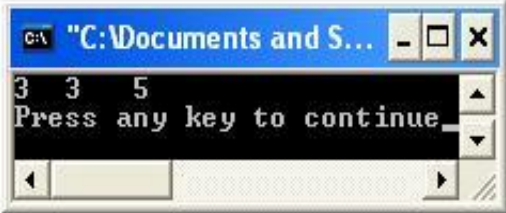
## Pointer this

المؤشر `this` هو مؤشر معرف مسبقاً يتم انشائه تلقائياً عند تعريف اوبجكت بحيث يقوم بالتأشير على الاوبجكت الذي تم انشائه ..

this is reserved word

```
class A
{
public:
    int x;
    int y;
    void fun() {cout<<x<<" "<<(*this).x<<" "<<this->y<<endl;}
};

void main()
{
    A obj1;
    obj1.x=3;
    obj1.y=5;
    obj1.fun();
}
```



كما نشاهد بالمثال تم الوصول الى الاعضاء `x` and `y` من خلال البوينتر `this` داخل الفنكشن `fun` . وتم اخراج نتائج مشابهة .. الهدف من هذا المثال اقناعك بأن المؤشر `this` هو مؤشر مخفي وله استخدامات بهذا الشايتتر سنتعرف اليها فيما بعد .

```

class A
{
    int x;
public:
    A(){x=5;}
    A fun(){x=x+3; return *this;}
    void print(){cout<<x<<endl;}
};

void main()
{
    A obj1, obj2;
    obj1=obj2.fun();
    obj1.print();
    obj2.print();
}

```

في هذا المثال ستكون المخرجات :-

8

8

عند تعريف الـ obj1 and obj2 تم اعطاء x قيمة ابتدائية تساوي 5 حسب الكونستركتور ومن ثم تم استدعاء الفنكشن fun واسناد القيمة التي يقوم بارجاعها الى الـ obj1

الفنكشن fun يقوم بزيادة قيمة x بمقدار 3 .. ولكن كيف يمكنني ارجاع الـ obj1 ؟

الجواب من خلال المؤشر this حيث سيقوم بارجاع الـ obj1 الذي تم الاتصال منه caller

object وهو obj2 بعد تعديل قيمة x بزيادة 3 لتصبح قيمتها 8 . والان عملية المساواة

ستقوم بنسخ محتوى obj2 الى obj1 . وتصبح قيمة x في كلاهما تساوي 8

بإختصار المؤشر this هو مؤشر معرف ضمناً داخل الكلاس .

# Friend Functions of Classes


الفنكشن الصديق :- هو فنكشن يملك الصلاحية بالوصول الى الـ private member يكتب الـ prototype الخاص فيه داخل الفنكشن مع اضافة كلمة friend في بدايته ويكتب التعريف الخاص به خارج الكلاس كما انه لا يوجد اي مشكلة في كتابة الـ prototype سواء كان داخل منطقة الـ public or private or protected لان الفريند فنكشن ليس عضو بالكلاس وعند استدعائه لسنا بحاجة الى caller object ويعامل معاملة الفنكشن العادي .

```
class A
{
    int x;
public:
    int y;
    void set(int a, int b) {x=a; y=b;}
    void fun1() {cout<<x<<" "<<y<<endl;}
    friend void fun2(A );
};

void main()
{
    A obj1;
    obj1.set(3,5);
    obj1.fun1();
    fun2(obj1);
}

void fun2(A obj)
{
    cout<<obj.x<<" "<<obj.y<<endl;
}

void fun3(A obj)
{
    // cout<<obj.x<<" "<<obj.y<<endl;    x is private
}
```



كما نلاحظ في هذا المثال fun2 هو فنكشن صديق للكلاس A بالتالي يستطيع الوصول الى الاعضاء المصنفة على انها private .. يكتب الـ header داخل الكلاس مع اضافة كلمة friend ومن ثم فاصلة منقوطة .. كما نلاحظ ان fun 3 لا يستطيع الوصول الى الـ private لانه غير صديق للكلاس .

```

class A
{
    int x;
public:
    A(){x=5;}
    void fun(){cout<<x<<" Member function"<<endl;}
    friend void fun(A);
};

void fun(A obj)
{    cout<<obj.x<<" Non member function"<<endl;}

void main()
{
    A obj1;
    obj1.fun();
    fun(obj1);
}

```

---

في هذا المثال تطبيق على الي حكيناه فوق ما في اي مشكلة 2 fun هو فنكشن

صديق وتم كتابة ال prototype الخاص فيه في منطقة ال private ولم ياتر

على النتائج لانو حكيتم بنقدر نكتبه باي مكان وهو ليس عضو بالكلاس وانما

صديق له يمكنه الوصول الى الاعضاء الخاصة

---

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

void fun2(A obj)
{cout<<obj.x<<" Non member function and friend"<<endl;}

class A
{   friend void fun2(A);
    int x;
public:
    A(){x=5;}
    void fun1(){cout<<x<<" Member function"<<endl;}
};

void main()
{
    A obj1;
    obj1.fun1();
    fun2(obj1);
}
```

هنا error والسبب تعريف الفنكشن قبل الكلاس

---

```

class A
{
    friend void fun1();
    int x;
};

class B: public A
{
    friend void fun2();
    int y;
};

void fun1()
{
    B obj;
    obj.x=3;
    //obj.y=4;    illegal
}

void fun2()
{
    B obj;
    //obj.x=3;    illegal
    obj.y=4;
}

void main()
{
    fun1();
    fun2();
}

```

في حالة الوراثة فإن الفريند فنكشن لا يورث لانه حكيما انه مش عضو .. بالتالي كل فريند

فنكشن حتى بعد الوراثة لا يستطيع الوصول سوى للاعضاء الخاص بالكلاس المعروف به يعني

يكون صديق لكلاس الي تعرف فيه فقط والمثال السابق يوضح الفكرة

<=====

# Templates

وهو انشاء نوع بيانات ياخذ صفة القالب بحيث ادراج اي نوع بيانات معرف مسبقا يقوم مكان  
نوع ال template .

مثال :-

```
template <class T>
class A
{ T X;
  T Y;
public :
  T fun1 ()
  {return x+y;}
};
```

في ال main عند تعريف اوبجكت لابد من تحديد نوع البيانات الذي سيحل محل ال T .

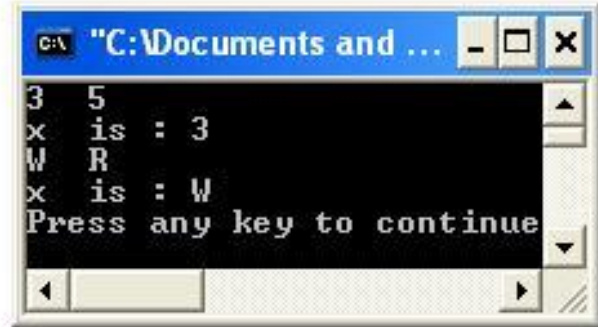
```
void main ()
{A <int> obj1;
  T=int هيك بصير كل
  A<char> obj2;
  T=cahr هيك بصير كل
  and so on ....}
```

```

template<class T>
class A
{
    T x;
    T y;
public:
    void set(T a, T b) {x=a; y=b;}
    void print(){cout<<x<<" "<<y<<endl;}
    T getX();
};
template<class T>
T A <T>::getX()
{
    return x;
}
void main()
{
    A<int> obj;
    obj.set(3,5);
    obj.print();
    cout<<"x is : "<<obj.getX()<<endl;

    A<char> obj2;
    obj2.set('W','R');
    obj2.print();
    cout<<"x is : "<<obj2.getX()<<endl;
}

```



مثال يوضح الفكرة لكن يجب التنبيه على كتابة تعريف الفونكشن خارج الكلاس getX من خلال اتباع الخطوات كما في الشكل

اولاً يتم كتابة <class T> template

ومن ثم نوع الفونكشن سواء كان T or char or int or void . بعد ذلك اسم الكلاس ومن ثم بين اشارتين اكبر واصغر يتم كتابة اسم ال template وحسب المثال اسمه T ومن ثم نقطتين رئيسيتين واسم الفونكشن .

هذا كل ما في ال template بسيط جداً المهم حفظ ال syntax .



# Operator Overloading

عملية اعادة تعريف المعاملات مثل +, -, %, <, >, >> وجميع المعاملات الاخرى للتعامل معها بشكل مباشر عن طريق الاوبجكت ..

فيما سبق كان هناك عملية واحدة مسموح بها على الاوبجكت بشكل مباشر وهي obj1=obj2 عملية الاسناد . اليوم سنتعلم سوياً طريقة السماح لجميع العمليات (حسب الحاجة) التي من الممكن ان تكون متاحة للعمل بها بشكل مباشر على الاوبجكت ..

ضروري نعرف انه جميع المعاملات "operator" هي عبارة عن فنكشن معرفة مسبقاً بلغة سي بلس بلس والي رح نعمله هسا over load يعني اعادة تعريف لهذا الفنكشن ...

example :-

```
class square {
    int length,width;

    public :
    square(int x,int y) { length=x; width=y; }
    square operator +(square para) {
        square A(0,0);
        A.length=length+para.length;
        A.width=width+para.width;
        return A; }
    void print()
    {cout<<"Length= "<<length<<" Width= "<<width<<endl;}
};
```

الآن خلونا نعرف شو الي صار وشو يعني الفونكشن الي اسمه operator .

```
square operator +(square para) {  
    square A(0,0);  
    A.length=length+para.length;  
    A.width=width+para.width;  
    return A; }
```

لاعادة تعريف اي operator :-

1- يتم كتابة prototype كالتالي :- اول اشي نوع هذا الفونكشن وفي المثال السابق نوعه من نفس نوع الكلاس لانه رح يرجعلي حاصل جمع ل two object ويحطهم باوبجكت ثالث ويرجعلي اياه .

2- اسم الفونكشن دائماً يكون operator مضاف اليه رمز المعامل المراد عمل اعادة تعريف اليه .

3- يتم تحديد الباراميتر وبما انه رح اجمع 2 object وبما انه الفونكشن member بالتالي انا بحاجة ل بارميتر واحد الي هو على يمين المعامل .. وهسا بحكيكو كيف <sup>٨٨</sup> .

4- يتم انشاء اوبجكت مؤقت لحفظ النتيجة فيه ومن ثم ارجاعه بجملة return .

الفونكشن عندما يكون member فمن المستحيل استدعائه الا بواسطة اوبجكت يدعى caller object . وال caller object هو الاوبجكت الذي يكون على يسار المعامل بينما الاوبجكت الذي على يمين المعامل يتم ارساله ك باراميتر

```
void main()  
{ square obj1(2,3),obj2(1,2),obj3(0,0);  
    obj3=obj1+obj2;  
    obj3.print(); }
```

خلونا نركز على السطر الثالث بال main هاي العملية كانت غير مسموحة قبل ما نعمل over load ويتعامل معها المترجم compiler ... obj3=obj1.operator+(obj2)  
ويجدر التنبيه على ان ال caller object لديه صلاحية وصول مباشرة الى member private لذا انا لما كنت اجمع داخل الفونكشن length +para.length ..

لو بدنا نعمل للناقص .. ما رح يختلف اشي ..

```
square operator -(square para) {  
    square A(0,0);  
    A.length=length-para.length;  
    A.width=width-para.width;  
    return A; }
```

طيب للضرب ...

```
square operator *(square para) {  
    square A(0,0);  
    A.length=length*para.length;  
    A.width=width*para.width;  
    return A; }
```

طيب للقسمة ...

```
square operator /(square para) {  
    square A(0,0);  
    A.length=length/para.length;  
    A.width=width/para.width;  
    return A; }
```

طيب لباقي القسمة ....

```
square operator %(square para) {  
    square A(0,0);  
    A.length=length%para.length;  
    A.width=width%para.width;  
    return A; }
```

وهكذا لاي معامل رياضي .. ضروري نفهم فقط انه بعدل عملية ال over load بقدر اجري  
هاي العملية بشكل مباشر على الاوبجكت .

---

المعاملات ذات العلاقة relational operator

هذه المعاملات تقوم بارجاع قيمة true or false مثلاً ..

$5 < 4 = \text{false}$

الان كيف بدى اطبق هاي العلاقة نفسها على 2 object ,,

```
bool operator >(square para)  
{bool result= length>para.length && width>para.width};  
return result; }
```

or

```
bool operator >(square para)  
{  
    return (length>para.length && width>para.width);  
}
```

الآن باجي ل main ..

```
void main()
{
    square obj1(2,3),obj2(1,1);
    if(obj1>obj2)
    cout<<"obj1 is biiger"<<endl;
}
```

جملة ال if كيف رح تشتغل .. مثل ما حكينا الي ع يسارها caller object والي على يمينها parameter بروح بقارن ال length تبع ال caller وال para.length تبع obj2 ورح يرجعلي true . ويطبع الجملة التي تلي ال if .

طيب معامل الاصغر كيف يشتغله معقول نفس الاشئ ؟

```
bool operator <(square para)
{
    return (length<para.length && width<para.width);
}
```

طيب المعامل ==

```
bool operator ==(square para)
{
    return (length==para.length && width==para.width);
}
```

وهكذا على كل المعاملات ذات العلاقة مثل اكبر او يساوي ولايساوي !=

كل هذا الشرح السابق والامثلة توضح عمل ال overload في حال كان عضو بالكلاس ..  
ولكن هل يمكن كتابة هذه الفنكشن ك فنكشن nonmember ... الجواب نعم ..  
لو ركزتوا بالي حكيته انا فوق .. اول اشي بحدد المعادلة الي بدني اشتغلها مثلا .. السؤال طلب  
مني جمع او ضرب او قسمة او اي شي ل2 اوبجكت .

ob1+ob2 ;

خلينا نركز شوي هون :- لطالما العنصر الذي على يسار المعامل هو اوبجكت انا بقدر اكتبه  
بطريقتين member مثل ما تعلمنا و non member بس شو رح يفرق ؟

non member :- ليس بحاجة الى caller object بالتالي يتحول الي على يمين المعامل  
والي على يسار المعامل الى باراميتر .. بس شوي شوي في مشكلة .. كيف بدني اوصل  
للاعضاء البرايفت وانا مش عضو بالكلاس ... آه صح خليناها هاي من خلال friend  
... function

اذن يمكن لل overload operator ان يكون non member function بالتالي يجب ان  
يكون فريند وياخذ 2 parameter ..

```
friend rectangleType operator+(const rectangleType&,  
                                const rectangleType&);
```

يوضع هذا السطر داخل الكلاس كما تعلمنا مسبقا طريقة كتابة الفريند فنكشن وتعريفه يوضع  
خارج الكلاس كما التالي :-

```
rectangleType operator+(const rectangleType& firstRect,  
                        const rectangleType& secondRect)  
{  
    rectangleType tempRect;  
  
    tempRect.length = firstRect.length + secondRect.length;  
    tempRect.width = firstRect.width + secondRect.width;  
  
    return tempRect;  
}
```

طبعا كل الامثلة الماضية تحول بنفس الطريقة ٨٨

في حال ما كان عندي اعضاء private ف أنا لست بحاجة الى فريند فنكشن

## logic operator

ال and وال or بقدر عملهم over load

```
bool operator &&(square para)
{
    return (length&&para.length && width&&para.width);
}
```

or operator

```
bool operator ||(square para)
{
    return (length || para.length || width||para.width);
}
```

what the output :-

```
#include <iostream>
using namespace std;
class A
{
    int x, y;
public:
    A(){x=4; y=10;}
    void print(){cout<<x<<"      "<<y<<endl;}
    A operator+ (int a){A obj; obj.x=x+a; obj.y=y+a; return obj;}
    friend A operator- (int, A);
};
A operator- (int a, A obj)
{
    A temp;
    temp.x=obj.x-a;
    temp.y=obj.y-a;
    return temp;
}
void main()
{
    A obj1, obj2, obj3;
    obj2=obj1+20;
    //obj3=obj1-1; error
    obj3=1 - obj1;
    obj1.print();
    obj2.print();
    obj3.print();
}
```



الآن نيجي لـ cin and cout كيف رح نبرمجهم ؟

احنا رح نبرمج المعاملات الخاصة فيهم << and >> ,,

خلينا نكتب الجملة ونحلها ..

```
cout<<obj;
```

الي على يسار المعامل اوبجكت من كلاس ostream والي على يمينه اوبجكت من نوع كلاس انا عامله بس انا ما عندي اي صلاحية لافوت على الكلاس ostream واضيف عليه فنكشن بالتالي رح اعمل nonmember ويكون صديق للكلاس الي انا عامله وبوخد 2 باراميتير الاول الي هو cin او cout والثاني اوبجكت من نوع الكلاس الي انا عامله ..

بس خلينا نعرف:-

ال cout من كلاس اسمه ostream

وال cin من كلاس اسمه istream

```
friend ostream& operator << (ostream &C, square &OOP)
```

هاي الجملة بنكتبها داخل الكلاس square . وخلصنا نلاحظ انو بعد ال ostream في اشارة & ضروري نخطها ..وانا بفضل كما في المثال تحطو ريفرنس على كلشي .

```
ostream& operator << (ostream &C, square &OOP)
```

```
{ C<<OOP.length<<endl;
```

```
cout<<OOP.width<<endl;
```

```
return c; //or return cout;
```

```
}
```

انا استخدمت مرة cout ومرة c الي عرفتها كا باراميتير عشان اورجيكو انو نفس الاشئ وبجملة الريتيرن نفس الاشئ



```
friend ostream& operator >> (istream &C, square &OOP)
```

هذه الجملة كالعادة تكتب بالكلاس لتعريف فنكشن صديق ...

```
istream& operator >> (istream &C, square &OOP)
{
    C>>OOP.length;
    cin>>OOP.width;
    return c; //or return cin;
}
```

ما اختلف اي اشي بس غيرنا بدل ال cout حطينا اشارة ال cin وبدل ال ostream حطينا istream ونفس الملاحظة استخدمت مرة cin ومرة c عشان اثبتلكو انو نفس الاشئ .

what the output :-

```

#include <iostream>
using namespace std;
class A
{
    int x, y;
public:
    A(){x=4; y=10;}
    void print(){cout<<x<<"      "<<y<<endl;}
    friend ostream &operator<< (ostream& o, A obj);
};
ostream &operator<< (ostream& o, A obj)
{
    o<<obj.x<<"      "<<obj.y<<endl;
    cout<<obj.x<<"      "<<obj.y<<endl;
    return o;
}
void main()
{
    A obj1;
    cout<<"PRINT FUNCTION"<<endl;
    obj1.print();
    cout<<"OPERATOR << "<<endl;
    cout<<obj1<<"The End"<<endl;
}

```

```

C:\WINDOWS\system32\cmd.e
PRINT FUNCTION
4      10
OPERATOR <<
4      10
4      10
The End
Press any key to continue

```

الان في حال طلب مني عمل overload للمعادلة التالية `obj+int`

في هذه الحالة بشوف على يسار الاوبجكت كما سبق بلاقي انو اوبجكت بالتالي بقدر اعمله  
nonmember والعنصر الي على اليمين رقم `int`

```

square operator +(int para) {
    square A(0,0);
    A.length=length+para;
    A.width=width+para;
    return A; }

```

وعلى هذا المنوال في حال كان الي على اليسار رقم والي على اليمين اوبجكت ..

`int +obj`

في هذه الحالة ما بقدر اعمله غير nonmember فنكشن .

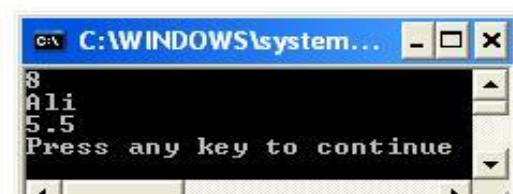
```
square operator +(int n, square para) {  
  
    square A(0,0);  
  
    A.length=n+para.length;  
  
    A.width=width+para.width;  
  
    return A; }
```

طبعاً ضروري يكون صديق هذا الفنكشن <sup>٨٨</sup>

# Function Overloading

هو أكثر من فنكشن يحمل نفس الاسم مع اختلاف الباراميتر ..

```
#include <iostream>  
#include<string>  
using namespace std;  
int fun(int a, int b)  
{  
    if (a>b)  
        return a;  
    else  
        return b;  
}  
  
double fun(double a, double b)  
{  
    if (a>b)  
        return a;  
    else  
        return b;  
}  
string fun(string a, string b)  
{  
    if (a>b)  
        return a;  
    else  
        return b;  
}  
  
void main()  
{  
    cout<<fun(8,5)<<endl;  
    cout<<fun("Ali","Ahmed")<<endl;  
    cout<<fun(3.4, 5.5)<<endl;  
}
```



كيف يمكنني اختصار هذا العدد من الفنكشن ؟

الحل استخدام ال template

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

template <class T>

T fun(T a, T b)
{
    if (a>b)
        return a;
    else
        return b;
}

void main()
{
    cout<<fun(8,5)<<endl;
    cout<<fun("Ali","Ahmed")<<endl;
    cout<<fun(3.4, 5.5)<<endl;
}
```

الى هنا يكون قد انتهى المشوار في مادة البرمجة الكينونية سائلاً الله عز وجل التوفيق للجميع  
وفي العلامات المرفعة كما ان لقائنا يتجدد ان شاء الله في مادة تراكيب البيانات data

structure

ما اصببت به فمن الله وما اخطأت به فمني ومن الشيطان اسال الله ان اكون قد وفقت في ايصال  
المعلومة ولا تنسوننا من صالح دعائكم والحمد لله الذي بحمده تتم الصالحات

اخوكم:- محمد المشرقي