مقرر برمجة ٢
الجلسة التاسعة عملي

- تجاوز الطرق والتوابع الظاهرية

:Functions Overriding

```
#include <iostream>
using namespace std;
class BaseClass {
public:
void disp()
{cout<<"Function of Parent Class";}};
class DerivedClass: public BaseClass{
public:
void disp()
{cout<<"Function of Child Class";}};
int main() {
DerivedClass obj ;
obj.disp();
return 0;}
```

يتم استدعاء التابع من الصنف الأساس **overridden**
function باستخدام احدى الطريقتين المبينتين:

```
#include <iostream>
using namespace std;
class BaseClass {
public:
BaseClass(){cout <<"the base class constructor "<<endl;}
void disp() {cout<<"Function of Parent Class"<<endl;}};
class DerivedClass: public BaseClass{
public:
DerivedClass(){cout <<"the derived class constructor"<<endl;}
void disp(){cout<<"Function of Child Class"<<endl;
BaseClass::disp();}};
int main() {
DerivedClass obj ; obj.disp();
return 0;}
```

```
#include <iostream>
using namespace std;
class BaseClass {
public:
BaseClass(){cout << "the base class constructor "<<endl;}
void disp() {cout<<"Function of Parent Class"<<endl;}};
class DerivedClass: public BaseClass{
public:
DerivedClass(){cout << "the derived class constructor "<<endl;}
void disp() {cout<<"Function of Child Class"<<endl;}};
int main(){
DerivedClass obj ; obj.disp();
obj.BaseClass::disp();
return 0;}
```

التوابع الظاهرية:

```
#include <iostream>
using namespace std;
class Base {
public:
void print() {
cout << "Base Function" << endl;}};
class Derived : public Base {
public:
void print(){
cout << "Derived Function" << endl;}};
int main() {
Derived derived1;
// pointer of Base type that points to
derived1
Base* base1 = &derived1;
// calls member function of base class
base1->print();
return 0;}
```

التوابع الظاهرية virtual functions:

```
#include <iostream>
using namespace std;
class Base {
public:
virtual void print() {
cout << "Base Function" << endl; }};
class Derived : public Base {
public:
void print() {
cout << "Derived Function" << endl;
}};
int main() {
Derived derived1;
// pointer of Base type that points to
derived1
Base* base1 = &derived1;
// calls member function of Derived
class
base1->print();
return 0;}
```

```
class Base {
    public:
      virtual void print() {
          // code
      }
};

class Derived : public Base {
    public:
      void print() {
          // code
      }
};

int main() {
    Derived derived1;
    Base* base1 = &derived1;

    base1->print();

    return 0;
}
```

print() of Derived class is called because print() of Base class is virtual

التوابع الظاهرية virtual functions:

```
#include <iostream>
using namespace std;
class Shape{
protected: double width , height;
public:
Shape(int a = 0, int b = 0)
{width= a;height= b; }
virtual double getArea() {
cout<< "Area can not be calculated..."
<< "Shape must be specified first ";
return 0;}};
class Rectangle: public Shape{
public:
Rectangle(double w,double h):Shape(w,h)
{width =w;height=h;
cout << "the constructor"<<endl; }
double getArea() {
cout<< "Rectangle area : ";
return width* height; }};
```

```
class Triangle: public Shape{
public:
Triangle(double w, double h):Shape ( w, h)
{width =w;height=h;
cout << "the constructor"<<endl;}
double getArea() {
cout<< "Triangle area : ";
return(width* height/ 2); }};
int main() {
Rectangle rect(4,5);
Triangle triA(4,5);
Shape*p1 = &rect;
Shape*p2 = &triA;
cout<< p1->getArea() << endl;
cout<< p2->getArea() << endl;
return 0 ;}
```

لاحظ استخدام الصيغة الموسعة للباني في الأصناف المشتقة

احذف الكلمة المفتاحية virtual كيف سيصبح الخرج

الوراثة المتعددة:

```
#include <iostream>
using namespace std;
class A {
int a; // we can make it as protected
public:
void seta(int x){a = x;}
int geta() {return a;}};
class B {
int b; // we can make it as protected
public:
void setb(int y){b = y;}
int getb() {return b;}};
class C : public A, public B {
public:
void print()
{cout<<"Addition of two numbers "<<
geta()+getb();}};
```

```
int main() {
C obj;
obj.seta(4);
obj.setb(9);
obj.print();
return 0;}
```

الوراثة المتعددة:

```
//C inherits A and B :both
#include <iostream>
using namespace std;
class A {
public:
A(){cout<<"Constructor of A class"<<endl;}
~A(){cout<<"deConstructor of A class"<<endl;}};
class B {
public:
B(){cout<<"Constructor of B class"<<endl;}
~B(){cout<<"deConstructor of B class"<<endl;}};
class C: public A, public B {
public:
C(){cout<<"Constructor of C class"<<endl;}
~C(){cout<<"deConstructor of C class"<<endl;}};
int main() {
//Creating object of class C
C obj;
return 0;}
```

جَامعة
المَنارة
MANARA UNIVERSITY

انتهت الجلسة