BATCH

RECURSION

C AND C++

PROGRAMMING MASTERCLASS

AWESH ISLAM
BUET, CSE

C++ Class-07

SHAROARE HOSAN EMON
BME , BUET

আমাদের সবগুলো ক্লাস দেখার জন্য ভিজিট করো

https://www.hsccrackers.com/

SCAN ME

Virtual Functions

Pointers of Base class

```cpp
#include<iostream>
using namespace std;
class base{
    int x;
public:
    void set_x(int i) { x = i; }
    int get_x() { return x; }
};
class derived:public base{
    int y;
public:
    void set_y(int i) { y = i; }
    int get_y() { return  y; }
};
int main(){
    base *p;
    base b_ob;
    derived d_ob;

    p = &b_ob;
    p->set_x(10);
    cout<< "Base object x: "<<p->get_x()<<endl;
    p = &d_ob;
    p->set_x(99);
    // p->set_y() //Cannot be accessed
    d_ob.set_y(100);
    cout<<"Derived object x: "<<p->get_x()<<endl;
    cout<<"Derived object y: "<<d_ob.get_y()<<endl;

}
```

Pointers of Base class

1) Pointers of Base class can point object of derived class without any error.
2) But pointer to derived object will only have access to the inherited base class members.
3) The reverse is not true derived class pointer cannot point base class object.
4) If we increment the pointer it will indicate the next base class even if it is pointed to derived class.

Introduction to virtual functions

Why we need virtual functions?
Virtual function is a kind of functin which is declared within base class and redefined by a derived class.
Virtual functions implement “one interface, multiple methods” philosophy that underlies polymorphism.
It supports runtime polymorphism.

What happens without Virtual

```cpp
#include<iostream>
using namespace std;
class base{
    int x;
public:
    void set_x(int i) { x = i; }
    int get_x() { return x; }
    void display() {cout<<"Displaying Base class..."<<endl;}
};
class derived:public base{
    int y;
public:
    void set_y(int i) { y = i; }
    int get_y() { return  y; }
    void display() {cout<<"Displaying Derived class..."<<endl;}
};
int main(){
    base *p;
    derived d_ob;
    p = &d_ob;
    p->display();
}
```

```
Displaying Base class...
Program ended with exit code: 0
```

What happens with Virtual Function

```cpp
#include<iostream>
using namespace std;
class base{
    int x;
public:
    void set_x(int i) { x = i; }
    int get_x() { return x; }
    virtual void display() {cout<<"Displaying Base class..."<<endl;}
};
class derived:public base{
    int y;
public:
    void set_y(int i) { y = i; }
    int get_y() { return  y; }
    void display() {cout<<"Displaying Derived class..."<<endl;}
};
int main(){
    base *p;
    derived d_ob;
    p = &d_ob;
    p->display();
}
```

```
Displaying Derived class...
Program ended with exit code: 0
```

Overloading vs Overriding

	Function Overloading	Function Overriding
Inheritance	Can occur without Inheritance	Overriding of functions occurs when one class is inherited from another class.
Parameters	Overloaded functions must differ in either number of parameters or type of parameters should differ.	In overriding, function signatures must be same.
Scope of functions	Overloaded functions are in same scope.	Overridden functions are in different scopes.
Behaviour/ Use	Overloading is used to have same name functions which behave differently.	Overriding is needed when derived class function has to do a different job than the base class function.

If not overridden

```cpp
#include<iostream>
using namespace std;
class base{
    int x;
public:
    void set_x(int i) { x = i; }
    int get_x() { return x; }
    virtual void display() {cout<<"Displaying Base class..."<<endl;}
};
class derived:public base{
    int y;
public:
    void set_y(int i) { y = i; }
    int get_y() { return  y; }
};
int main(){
    base *p;
    derived d_ob;
    p = &d_ob;
    p->display();
}

```

```
Displaying Base class...
Program ended with exit code: 0
```

Runtime Polymorphism

```cpp
#include<iostream>
#include<cstdlib>
#include<ctime>
using namespace std;
class base{
    int x;
public:
    void set_x(int i) { x = i; }
    int get_x() { return x; }
    virtual void display() {cout<<"Displaying Base class..."<<endl;}
};
class derived:public base{
    int y;
public:
    void display() { cout<<"Displaying derived class..."<<endl;}
};
class derived2:public base{
    int z;
public:
    void display() { cout<<"Displaying derived2 class..."<<endl;}
};
int main(){
    base *p;
    derived d_ob;
    derived2 d2_ob;
    time_t t;
    srand((unsigned)(time(&t)));
    int j = rand();
    if(j % 2) p = &d2_ob;
    else p = &d_ob;
    p->display();
}
```

Use of Virtual Function

```cpp
#include<iostream>
using namespace std;
class shape{
    double dim1,dim2;
public:
    void set_dimensions(double d1,double d2){
        dim1 = d1;
        dim2 = d2;
    }
    void get_dimensions(double &d1,double &d2){
        d1 = dim1;
        d2 = dim2;
    }
    virtual double get_area(){
        cout<<"You have to override this function..."<<endl;
        return 0.0;
    }
};
class rectangle:public shape{
public:
    double get_area(){
        double d1,d2;
        get_dimensions(d1,d2);
        return d1*d2;
    }
};
class triangle:public shape{
public:
    double get_area(){
        double d1,d2;
        get_dimensions(d1,d2);
        return 0.5*d1*d2;
    }
};

```

```cpp
int main(){
    shape *p;
    rectangle r;
    triangle t;
    r.set_dimensions(10.5, 20.3);
    t.set_dimensions(5.1, 4.13);
    p = &r;
    cout<<"Area of rectangle is: "<<p->get_area()<<endl;
    p = &t;
    cout<<"Area of trigangle is: "<<p->get_area()<<endl;
}
```

```
Area of rectangle is: 213.15
Area of trigangle is: 10.5315
Program ended with exit code: 0
```

Pure Virtual Function

```cpp
#include<iostream>
using namespace std;
class shape{
    double dim1,dim2;
public:
    void set_dimensions(double d1,double d2){
        dim1 = d1;
        dim2 = d2;
    }
    void get_dimensions(double &d1,double &d2){
        d1 = dim1;
        d2 = dim2;
    }
    virtual double get_area() = 0;
};
class rectangle:public shape{
public:
    double get_area(){
        double d1,d2;
        get_dimensions(d1,d2);
        return d1*d2;
    }
};
class triangle:public shape{
public:
    double get_area(){
        double d1,d2;
        get_dimensions(d1,d2);
        return 0.5*d1*d2;
    }
};
```

Abstract Classes

1) Abstract function contains at least one function for which no body exists.
2) Abstract classes are incomplete
3) They are not intended nor able to stand alone
4) You can create a pointer to an abstract class

Applying Polymorphism

Pure abstract class list:

```
1  #include<iostream>
2  using namespace std;
3  class list{
4      list *head;
5      list *tail;
6      list *next;
7      int num;
8      list() { head = tail = next = NULL;}
9      virtual void store(int i) = 0;
10     virtual void retreive() = 0;
11 };
12
```

Applying Polymorphism

Queue

```
13 class queue:public list{
14 public:
15     void store(int i){
16         list *item;
17         item = new queue;
18         if(!item){
19             cout<<"Allocation Error\n";
20             exit(1);
21         }
22         item->num = i;
23         if(tail) tail->next = item;
24         tail = item;
25         item->next = NULL;
26         if(!head) head = tail;
27     }
28     int retreive(){
29         int i;
30         list *p;
31         if(!head){
32             cout<<"List is empty\n";
33             return 0;
34         }
35         i = head->num;
36         p = head;
37         head = head->next;
38         delete p;
39         return i;
40     }
41 };
```

Applying Polymorphism

Stack

```
42  class stack:public list{
43  public:
44      void store(int i){
45          list *item;
46          item = new stack;
47          if(!item){
48              cout<<"Allocation Error\n";
49              exit(1);
50          }
51          item->num = i;
52          if(head) item->next = head;
53          head = item;
54          if(!tail) tail = head;
55      }
56      int retreive(){
57          int i;
58          list *p;
59          if(!head){
60              cout<<"List is empty\n";
61              return 0;
62          }
63          i = head->num;
64          p = head;
65          head = head->next;
66          delete p;
67          return i;
68      }
69  };
```

Applying Polymorphism

Main

```
73  int main(){
74      list *p;
75      stack s_ob;
76      queue q_ob;
77      char ch;
78      for(int i = 0; i < 10;i++){
79          cout<<"Stack or Queue? (S/Q): ";
80          cin>>ch;
81          if(ch == 'Q') p = &q_ob;
82          else p = &s_ob;
83          p->store(i);
84      }
85      for(;;){
86          cout<<"Stack or Queue? (S/Q): ";
87          cin>>ch;
88          if(ch == 'Q') p = &q_ob;
89          else p = &s_ob;
90          cout<<p->retreive()<<endl;
91      }
92  }
```