Part1: Virtual methods

A virtual function (also known as virtual methods) is a member function that is declared within a base class and is re-defined (overridden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class’s version of the method.

Consider the following simple program showing the runtime behavior of virtual functions.

#include <iostream>

using namespace std;

class base {

public:

virtual void print()

{ cout << "print base class\n"; }

void show()

{ cout << "show base class\n"; }

};

class derived : public base {

public:

void print()

{ cout << "print derived class\n"; }

void show()

{ cout << "show derived class\n"; }

};

int main()

{

base\* bptr;

derived d;

bptr = &d;

bptr->print();

bptr->show();

return 0;

}

Output:

|  |
| --- |
|  |

Part 2: pure virtual function (abstract class)

Sometimes implementation of all function cannot be provided in a base class because we don’t know the implementation. Such a class is called abstract class. For example, let Shape be a base class. We cannot provide implementation of function draw() in Shape, but we know every derived class must have implementation of draw(). Similarly an Animal class doesn’t have implementation of move() (assuming that all animals move), but all animals must know how to move. We cannot create objects of abstract classes.

A pure virtual function is declared by assigning 0 in declaration. See the following example.

#include<iostream>

using namespace std;

class Base

{

int x;

public:

virtual void fun() = 0;

int getX() { return x; }

};

// This class inherits from Base and implements fun()

class Derived : public Base

{

int y;

public:

void fun() { cout << "fun() called"; }

};

int main(void)

{

Derived d;

d.fun();

return 0;

}

Output:

|  |
| --- |
|  |

In the main function, create an object of base class, what is the output??!!

Part 3: Virtual Destructor

Deleting a derived class object using a pointer of base class type that has a non-virtual destructor results in undefined behavior. To correct this situation, the base class should be defined with a virtual destructor.   
For example, the following program results in undefined behavior.

#include <iostream>

using namespace std;

class base {

public:

base()

{

cout << "Constructing base\n";

}

~base()

{

cout << "Destructing base\n";

}

};

class derived : public base {

public:

derived()

{

cout << "Constructing derived\n";

}

~derived()

{

cout << "Destructing derived\n";

}

};

int main()

{

derived\* d = new derived();

base\* b = d;

delete b;

return 0;

}

Output:

|  |
| --- |
|  |

Making base class destructor virtual guarantees that the object of derived class is destructed properly, i.e., both base class and derived class destructors are called.

Now, update the destructor function in the base class by make it a virtual function, what is the output:

|  |
| --- |
|  |

Exercises:

1. Write a program to create a class shape with functions to find the area of the shapes and display the names of the shapes and other essential components of the class. Create derived classes circle, rectangle each having overriding functions area() and display(). Write a suitable program to illustrate virtual functions and virtual destructors.
2. Modify your solution in exercise 1 by convert shape class to abstract class.

That is the end of labsheet.. Good Luck