**Virtual Functions:**

Virtual functions are member functions whose behavior can be overridden in derived classes. As opposed to non-virtual functions, the overridden behavior is preserved even if there is no compile-time information about the actual type of the class. If a derived class is handled using pointer or reference to the base class, a call to an overridden virtual function would invoke the behavior defined in the derived class.

#include <iostream>

using namespace std;

class Polygon {

protected:

int width, height;

public:

void set\_values (int a, int b)

{ width=a; height=b; }

virtual int area ()

{ return 0; }

};

class Rectangle: public Polygon {

public:

int area ()

{ return width \* height; }

};

class Triangle: public Polygon {

public:

int area ()

{ return (width \* height / 2); }

};

int main () {

Rectangle rect;

Triangle trgl;

Polygon poly;

Polygon \* ppoly1 = &rect;

Polygon \* ppoly2 = &trgl;

Polygon \* ppoly3 = &poly;

ppoly1->set\_values (4,5);

ppoly2->set\_values (4,5);

ppoly3->set\_values (4,5);

cout << ppoly1->area() << '\n';

cout << ppoly2->area() << '\n';

cout << ppoly3->area() << '\n';

return 0;

}

// Virtual Functions

#include <iostream>

using namespace std;

class Weapon {

public:

virtual void features()

{ cout << "Loading weapon features.\n"; }

};

class Bomb : public Weapon {

public:

void features() {

this->Weapon::features();

cout << "Loading bomb features.\n";

}

};

class Gun : public Weapon {

public:

void features() {

this->Weapon::features();

cout << "Loading gun features.\n";

}

};

class Loader {

public:

void loadFeatures(Weapon \*weapon) {

weapon->features();

}

};

int main() {

Loader \*l = new Loader;

Weapon \*w;

Bomb b;

Gun g;

w = &b;

l->loadFeatures(w);

w = &g;

l->loadFeatures(w);

return 0;

}

Abstract classes and pure Virtual fuctions :

Abstract classes act as expressions of general concepts from which more specific classes can be derived. You cannot create an object of an abstract class type; however, you can use pointers and references to abstract class types.

A class that contains at least one pure virtual function is considered an abstract class. Classes derived from the abstract class must implement the pure virtual function or they, too, are abstract classes.

#include <iostream>

using namespace std;

// Abstract class

class Shape

{

protected:

float l;

public:

void getData()

{

cin >> l;

}

virtual float calculateArea() = 0;

};

class Square : public Shape

{

public:

float calculateArea()

{ return l\*l; }

};

class Circle : public Shape

{

public:

float calculateArea()

{ return 3.14\*l\*l; }

};

int main()

{

Square s;

Circle c;

cout << "Enter length to calculate the area of a square: ";

s.getData();

cout<<"Area of square: " << s.calculateArea();

cout<<"\nEnter radius to calculate the area of a circle: ";

c.getData();

cout << "Area of circle: " << c.calculateArea();

return 0;

}