SHORT QUESTIONS

Note: Long Questions are given after some lines below;

1. What is an object? http://shiningstudy.com/
   - Object is an instance of class.
   - "**object**" refers to a particular instance of a class where the **object** can be a combination of variables, functions, and data structures.
2. What is class, write syntax how to declare class?
   - Class is a template.
   - When you define a class, you define a blueprint for a data type
   - A class definition starts with the keyword **class** followed by the class name
   - the class body, enclosed by a pair of curly braces.
   - A class definition must be followed either by a semicolon or a list of declarations. For example, we defined the Box data type using the keyword **class** as follows:

```
class Box

{

   public:

      double length;   // Length of a box

      double breadth;  // Breadth of a box

      double height;   // Height of a box

};
```

3. What is constructor?
   This constructor function is declared just like a regular member function, but with a name that matches the class name and without any return type; not even void
   Or
   Constructors (**C++**) A **constructor** is a kind of member function that initializes an instance of its class. A **constructor** has the same name as the class and no return value. A **constructor** can have any number of parameters and a class may have any number of overloaded **constructors**

4. What is destructor?
   Destructors (**C++**) "**Destructor**" functions are the inverse of constructor functions. They are called when objects are destroyed (deallocated). Designate a function as a class's **destructor** by preceding the class name with a tilde (~)
5. What is constructor overloading?
   Like any other function, a constructor can also be overloaded with different versions taking different parameters: with a different number of parameters and/or parameters of different types. The compiler will automatically call the one whose parameters match the arguments:

```
// overloading class constructors
#include <iostream>
using namespace std;

class Rectangle {
    int width, height;
  public:
    Rectangle ();
    Rectangle (int,int);
    int area (void) {return (width*height);}
};

Rectangle::Rectangle () {
  width = 5;
  height = 5;
}

Rectangle::Rectangle (int a, int b) {
  width = a;
  height = b;
}

int main () {
  Rectangle rect (3,4);
  Rectangle rectb;
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

6. Can we pass class object into another class?

   there are two way of passing class object(it is what you are asking) as a function argument i) Either pass the copy of object to the function, in this way if there is any change done by the function in the object will not be reflected in the original object

ii) Pass the base address of the object as a argument to the function.In thsi method if there are any changes done in the object by the calling function, they will be reflected in the orignal object too.

Example

```
#include <iostream>
using namespace std;
class rational
{
private:
        int num;
        int dnum;
public:
        rational():num(1),dnum(1)
        {}
        void get ()
        {
                cout<<"enter numerator";
                cin>>num;
                cout<<"enter denomenator";
                cin>>dnum;
        }
        void print ()
        {
                cout<<num<<"/"<<dnum<<endl;
        }
        void multi(rational r1,rational r2)
        {
                num=r1.num*r2.num;
                dnum=r1.dnum*r2.dnum;
        }
};
void main ()
{
        rational r1,r2,r3;
        r1.get();
        r2.get();
        r3.multi(r1,r2);

        r3.print();

}
```

7. Unary increment , decrement

The increment operator ++ adds 1 to its operand, and the decrement operator -- subtracts 1 from its operand. Thus:

```
x = x+1;

is the same as
```

```
x++;
```

And similarly:

```
x = x-1;

is the same as

x--;
```

Both the increment and decrement operators can either precede (prefix) or follow (postfix) the operand.

8. Virtual member function
   In object-oriented programming, a **virtual function** or **virtual method** is a function or method whose behavior can be overridden within an inheriting class by a function with the same signature. This concept is an important part of the polymorphism portion of object-oriented programming

Purpose:

In object-oriented programming, when a derived class inherits from a base class, an object of the derived class may be referred to via a pointer or reference of the base class type instead of the derived class type. If there are base class methods overridden by the derived class, the method actually called by such a reference or pointer can be bound either 'early' (by the compiler), according to the declared type of the pointer or reference, or 'late' (i.e., by the runtime system of the language), according to the actual type of the object referred to.

Virtual functions are resolved 'late'. If the function in question is 'virtual' in the base class, the most-derived class's implementation of the function is called according to the actual type of the object referred to, regardless of the declared type of the pointer or reference. If it is not 'virtual', the method is resolved 'early' and the function called is selected according to the declared type of the pointer or reference.

Virtual functions allow a program to call methods that don't necessarily even exist at the moment the code is compiled.

In C++, *virtual methods* are declared by prepending the `virtual` keyword to the function's declaration in the base class.

Virtual Functions:

If you want to execute the member function of derived class then, you can declare display( ) in the base class virtual which makes that function existing in appearance only but, you can't call that function. In order to make a function virtual, you have to add keyword **virtual** in front of a function.

```
/* Example to demonstrate the working of virtual function in C++ programming. */

#include <iostream>
using namespace std;
class B
{
    public:
     virtual void display()      /* Virtual function */
         { cout<<"Content of base class.\n"; }
};

class D1 : public B
{
    public:
       void display()
         { cout<<"Content of first derived class.\n"; }
};

class D2 : public B
{
    public:
       void display()
         { cout<<"Content of second derived class.\n"; }
};

int main()
{
    B *b;
    D1 d1;
    D2 d2;

/* b->display();  // You cannot use this code here because the function of base class is virtual. */

    b = &d1;
    b->display();   /* calls display() of class derived D1 */
    b = &d2;
    b->display();   /* calls display() of class derived D2 */
    return 0;
}
```

Output

```
Content of first derived class.
Content of second derived class.
```

After the function of base class is made virtual, code `b->display()` will call

the `display()` of the derived class depending upon the content of pointer.In this program, `display()` function of two different classes are called with same code which is one of the example of polymorphism in C++ programming using virtual functions.

C++ Abstract class and Pure virtual Function

In C++ programming, sometimes inheritance is used only for the better visualization of data and you do not need to create any object of base class. For example: If you want to calculate area of different objects like: circle and square then, you can inherit these classes from a shape because it helps to visualize the problem but, you do not need to create any object of *shape*. In such case, you can declare *shape* as a abstract class. If you try to create object of a abstract class, compiler shows error.

Declaration of a Abstract Class

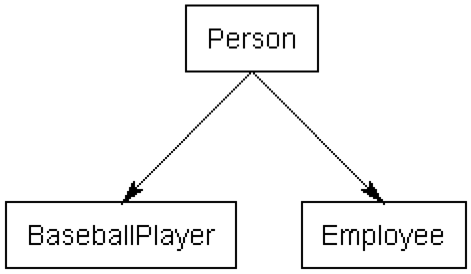
If expression **`=0`** is added to a virtual function then, that function is becomes pure virtual function. Note that, adding `=0` to virtual function does not assign value, it simply indicates the virtual function is a pure function. If a base class contains at least one virtual function then, that class is known as abstract class.

Example to Demonstrate the Use of Abstract class

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

class Shape                   /* Abstract class */
{
    protected:
       float l;
    public:
       void get_data()          /* Note: this function is not virtual. */
       {
           cin>>l;
       }

       virtual float area() = 0; /* Pure virtual function */
};

class Square : public Shape
{
   public:
      float area()
      {   return l*l;  }
};

class Circle : public Shape
{
   public:
      float area()
      { return 3.14*l*l; }
};

int main()
{
   Square s;
   Circle c;
   cout<<"Enter length to calculate area of a square: ";
   s.get_data();
   cout<<"Area of square: "<<s.area();
   cout<<"\nEnter radius to calcuate area of a circle:";
   c.get_data();
   cout<<"Area of circle: "<<c.area();
   return 0;
}
```

In this program, pure virtual function `virtual float area() = 0;` is defined inside class *Shape*, so this class is an abstract class and you cannot create object of class *Shape*

9. What is inheritance
   Classes in **C++** can be extended, creating new classes which retain characteristics of the base class. This process, known as**inheritance**, involves a base class and a derived class: The derived class inherits the members of the base class, on top of which it can add its own members.



10. What is multi and multiple inheritance
11. What is meant by level of inheritance

Levels of Inheritance

In C++ programming, a class be can derived from a derived class which is known as multilevel inhertiance. For example:

```
class A

{ .... ... .... };

class B : public A

{ .... ... .... };

class C : public B

{ .... ... .... };
```

In this example, class *B* is derived from class *A* and class *C* is derived from derived class *B*.

Example to Demonstrate the Multilevel Inheritance

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

class A
{
    public:
      void display()
      {
          cout<<"Base class content.";
      }
};

class B : public A
{

};

class C : public B
{

};

int main()
{
    C c;
    c.display();
    return 0;
}
```

Output

```
Base class content.
```

Explanation of Program

In this program, class *B* is derived from *A* and *C* is derived from *B*. An object of class *C* is defined in`main()` function. When the `display()` function is called, `display()` in class *A* is executed because there is no `display()` function in *C* and *B*. The program first looks for `display()` in class C first but, can't find it. Then, looks in *B* because *C* is derived from *B*. Again it can't find it. And finally looks it in*A* and executes the codes inside that function.

If there was `display()` function in *C* too, then it would have override `display()` in *A* because ofmember function overriding.

Multiple Inheritance in C++

In C++ programming, a class can be derived from more than one parents. For example: A class*Rectangle* is derived from base classes *Area* and *Circle*.

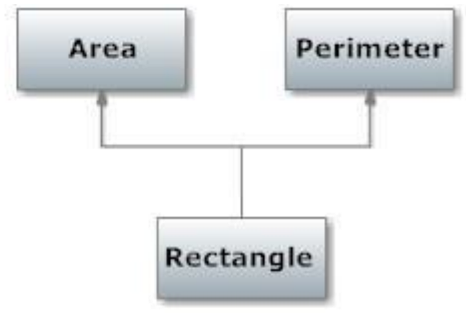


Figure: Multiple Inheritance Example

Source Code to Implement Multiple Inheritance in C++ Programming

This program calculates the area and perimeter of an rectangle but, to perform this program, multiple inheritance is used.

```
#include <iostream>
using namespace std;
class Area
{
  public:
    float area_calc(float l,float b)
    {
        return l*b;
    }
};
```

```
class Perimeter
{
  public:
    float peri_calc(float l,float b)
    {
        return 2*(l+b);
    }
};

/* Rectangle class is derived from classes Area and Perimeter. */
class Rectangle : private Area, private Perimeter
{
    private:
        float length, breadth;
    public:
       Rectangle() : length(0.0), breadth(0.0) { }
       void get_data( )
       {
           cout<<"Enter length: ";
           cin>>length;
           cout<<"Enter breadth: ";
           cin>>breadth;
       }

       float area_calc()
       {
       /* Calls area_calc() of class Area and returns it. */

           return Area::area_calc(length,breadth);
       }

       float peri_calc()
       {
       /* Calls peri_calc() function of class Perimeter and returns it. */

           return Perimeter::peri_calc(length,breadth);
       }
};

int main()
{
    Rectangle r;
    r.get_data();
    cout<<"Area = "<<r.area_calc();
    cout<<"\nPerimeter = "<<r.peri_calc();
    return 0;
}
```

Output

```
Enter length: 5.1

Enter breadth: 2.3

Area = 11.73
```

```
Perimeter = 14.8
```

LONG QUESTIONS

C++ Practice Programs for Exams (UOS)

Question # 1: Create a class called **time** that has separate int member data for hours, minutes, and seconds. One constructor should initialize this data to 0, and another should initialize it to fixed values. Another member function should display it, in 11:59:59 format. Write a program to validate the following expression in the main () function.

t4 = t1 += t2 + t3;

Where t1, t2, t3 and t4 are the instances of the **time** class, objects (should they be const?) and one that isn't initialized. Make appropriate member functions const.

Answer # 1:

```
#include <iostream>
using namespace std;
class Time{
public:
int hours, minutes, seconds;
Time (){
hours=minutes=seconds=0;
}//Time()
Time(int hours, int minutes, int seconds){
this->hours=hours; this->minutes=minutes; this->seconds=seconds;
}//Time(int hours, int minutes, int seconds)
Time operator+(const Time& t){
    Time t2;
    t2.hours = this->hours + t.hours;
    t2.minutes = this->minutes+t.minutes;
    t2.seconds = this->seconds+t.seconds;
return t2;
}//overloaded + operator
Time operator+=(const Time& t){
    Time t2;
    t2.hours = this->hours + t.hours;
    t2.minutes = this->minutes+t.minutes;
    t2.seconds = this->seconds+t.seconds;
return t2;
}//overloaded +=
void show(){
cout<<hours<<":"<<minutes<<":"<<seconds<<endl;
}//show()
};//end class Time
```

```
int main(){
    Time t1;
    Time t2(2,2,2);
    Time t3(4,5,6);
    Time t4 = t1 += t2 + t3;
t4.show();
}//end main()
```

<u>Question # 2:</u>Create a class **<u>student</u>** which shows the ***degree***(declared as c- string)**,** of the student and **<u>person</u>** class which has the data member ***name***(declared as c- string**)**and ***id***(integer)**.** By using multiple inheritances derive another class **<u>instructor</u>** which has its own data member ***salary*** (as double). Write the appropriate constructor, member functions to input data by using keyboard and display data for the **<u>instructor</u>**.

<u>Answer # 2:</u>

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

class Student{
public:
string degree;
};//end class student
class Person{
public:
int id;
string name;
};//end class Person
class Instructor :Student,Person{
public:
double salary;
Instructor(){}
Instructor(int id, string name, string degree, double salary){
this->id = id; this->name = name; this->degree = degree;
     this->salary = salary;
    }//Constructor
voidshowData(){
cout<<"ID:"<<id<<endl;
cout<<"NAME:"<<name<<endl;
cout<<"DEGREE:"<<degree<<endl;
cout<<"SALARY:"<<salary<<endl;
```

```
    }//void show()
Void inputData(){
cout<<"Enter id: "; cin>>id;
cout<<"Enter name: "; getline(cin,name);
cout<<"Enter degree: "; getline(cin,degree);
cout<<"Enter salary: "; cin>>salary;
    }//inputData()
};//end class Instructor
int main(){
    Instructor ali;
     ali.inputData();
     ali.showData();
return 0;
}
```

or

```
#include <iostream>
using namespace std;
class Student{
public:
char degree[10];
};
class Person{
public:
int id;
char name[30];
};
class Instructor :Student,Person{
public:
double salary;
Instructor(){}
void show(){
     cout<<"ID:"<<id<<endl;
     cout<<"NAME:"<<name<<endl;
     cout<<"DEGREE:"<<degree<<endl;
     cout<<"SALARY:"<<salary<<endl;
    }
void inputData(){
     cout<<"Enter id: "; cin>>id;
     cout<<"Enter name: "; cin.getline(name,30);
     cout<<"Enter degree: "; cin.getline(degree,10);
     cout<<"Enter salary: "; cin>>salary;
    }
};

int main(){
    Instructor ali;
     ali.inputData();
     ali.show();
     return 0;
}
```

Question # 3:A publishing company that markets both book and audiocassette versions of its works. Create a class called **publication** that stores the ***title***(a string or c-string) and ***price*** (type float) of a **publication**. From this class derive two classes: **book**, which has a page ***count*** (type int) and **tape**: which has a playing time in ***minutes*** (type float). Each of the three classes should have a ***getdata()*** function to get its data from the user at the keyboard, and a ***putdata()*** function to display the data. Add a member function ***Oversize ()*** to the **book** and **tape classes**. Let's say that a book with more than 500 pages, or a tape with a playing time longer than *90 minutes*, is considered *oversize*. You can access this function from main () and display the string ***"Oversize"*** for oversized books and tapes when you display their data.
Write a ***main ()*** program that creates an array of pointers to **publication**. In a loop, ask the user for data about a particular **book** or **tape**.

Answer #3:
```
#include <iostream>
#include <string>
using namespace std;
class publication{
    protected:
    string title;
    float price;
    public:
    virtual void getdata(){
    }
    virtual void putdata(){
     cout<<"Title: "<<title<<endl;
     cout<<"Price: "<<price<<endl;
    }
};
class book : public publication{
    public:
    int count;
    void getdata(){
          cout<<"Enter book title : ";
          cin>>title;
          cout<<"Enter book price : ";
          cin>>price;
          cout<<"Enter page count : ";
          cin>>count;
    }
    void putdata(){
        publication::putdata();
        cout<<"Pages :"<<count<<endl;
        Oversize();
    }
    void Oversize(){
        if(count>500)
          cout<<"Oversize"<<endl;
    }
};
class tape : public publication{
    public:
    float minutes;
    void getdata(){
     cout<<"Enter tape title : ";
     cin>>title;
     cout<<"Enter tape price : ";
     cin>>price;
     cout<<"Enter tape minutes : ";
     cin>>minutes;
    }
    void putdata(){
        publication::putdata();
        cout<<"Time : "<<minutes<<endl;
        Oversize();
    }
    void Oversize(){
        if(minutes>90)
          cout<<"Oversize"<<endl;
    }
};
int main() {
     publication *p[5];
     book completeRef;
     book oopInCpp;
     book htmlIntro;
     tape briefHistoryOfTime;
     tape whyWomenCannotReadMaps;
     p[0] = &completeRef;
     p[1] = &oopInCpp;
     p[2] = &briefHistoryOfTime;
     p[3] = &htmlIntro;
     p[4] = &whyWomenCannotReadMaps;

     for(int i=0; i<5; i++){
         p[i]->getdata();
     }
     for(int i=0; i<5; i++){
         p[i]->putdata();
     }

     return 0;
}
```

Question # 4: Write down a class person (name) inherit class student (marks) and teacher (publications) from it. Write appropriate constructors, print() and get() functions for each class. Then write down main function that takes choice from user what type of object he wants to create. The program at the end should display the names of students and teachers that are outstanding. (A student is outstanding if he gets 90% marks and teacher having publication greater than 10).

Answer # 4:

```
#include <iostream>
#include <string>
using namespace std;
class person{
    public:
    string name;
    person(){}
    person(string n){
        name = n;
    }
    virtual void get(){}
    virtual void print(){}

};
class student : public person{
    public:
    int marks;
    student(){}
    student(string n, int m){
        name = n; marks = m;
    }
    void get(){
     cout<<"Enter student name: "; cin>>name;
     cout<<"Enter marks: ";  cin>>marks;
    }
    void print(){
        if(marks>=90){//print the out standings only
          cout<<"Name: "<<name<<endl;
     cout<<"Marks: "<<marks<<endl;
        }
    }
};
class teacher : public person{
    public:
    int publications;
    teacher(){}
    teacher(string name, int publications){
        this->name = name; this->publications = publications;
    }
    void get(){
     cout<<"Enter teacher name: "; cin>>name;
     cout<<"Enter number of publications: "; cin>>publications;
    }
    void print(){
        if(publications>10){//print the out standings only
cout<<"Name: "<<name<<endl;
cout<<"Pubs: "<<publications<<endl;
        }
    }
};
int main() {
     person *p[6];
     char option;
     for(int i=0; i<6; i++){
     cout<<"What do you want to creat? (t=teacher, s=student): ";
cin>>option;
         if(option=='t'){
             p[i] = new teacher;
             p[i]->get();
         }else {//s or any other character
             p[i] = new student;
             p[i]->get();
         }
     }
     for(int i=0; i<6; i++)
        p[i]->print();
     return 0;
}
```

Question # 5: Write a program that creates an **abstract class Child**. The **Child** class contains data members that hold the ***name*** (string) and ***age*** (float) of a child. The Child class also contains

GetData () and ShowData () member function and one pure virtual **member function GetGender (). Create** two subclasses**Male** and**Female** inherited from child. The **Male** and **Female** subclasses contain additional data member ***gender*** (string). The subclass overrides the **base class member functions**. Write a ***main ()*** that **demonstrates polymorphism** in action.

Answer #5:

```
#include<iostream>

using namespace std;

class Child{

        public:

        string name; float age;

        void GetData(){

                cout<<"Enter name :";

                cin>>name;

                cout<<"Enter age :";

                cin>>age;

        }

        virtual void ShowData(){

        }

        virtual void GetGender()=0;

};

class male : public Child {

        public:

        string gender;

        void ShowData(){

                cout<<"Name : "<< name << "\n"<<"Age : "<<age<<"\n"<<"Gender : "<<gender<<endl;

        }

        void GetGender(){
                cout<<"Enter gender ";
                cin>>gender;
        }
};
class Female : public Child{
        public:
                string gender;
        void ShowData(){
                cout<<"Name : "<< name << "\n"<<"Age : "<<age<<"\n"<<"Gender : "<<gender<<endl;
        }
        void GetGender(){
                cout<<"Enter gender ";
                cin>>gender;
        }
};
int main(){
        male m;
        Female f;
        Child *c;
        c = &m;
        c->GetData();
        c->GetGender();
        c->ShowData();
        c = &f;
        c->GetData();
        c->GetGender();

        c->ShowData();

        return 0;

}
```

------------------OR------------------

```
#include<iostream>

using namespace std;

class Child{

        public:

                void virtual GetGender () =0;

        string name; float age;

        void virtual GetData() =0;

        void virtual ShowData() =0;

};

class Male: public Child

{

        public:

                string Gender;

                void GetData(){

                        cout<<"Enter name :";

                        cin>>name;

                        cout<<"Enter Age";

                        cin>>age;
```

```
		}
		void GetGender(){
			cout<<"Enter Gender :";
			cin>>Gender;
		}
		void ShowData(){
			cout<<"Name is  :"<<name<<"\n"<<"Gender :"<<Gender<<"\n"<<"Age is
:"<<age<<endl;
		}
};
class Female: public Child
{
	public:
		string Gender;
		void GetData(){
			cout<<"Enter name :";
			cin>>name;
			cout<<"Enter Age";
			cin>>age;
		}
		void GetGender(){
			cout<<"Enter Gender :";
			cin>>Gender;
```

```
                }

                void ShowData(){

                        cout<<"Name is  :"<<name<<"\n"<<"Gender :"<<Gender<<"\n"<<"Age is
:"<<age<<endl;

                }

};

int main(){

        Child *c;

        Male m; Female f;

        c = & m;

        c->GetData();

        c->GetGender();

        c->ShowData();

        c = & f;

        c->GetData();

        c->GetGender();

        c->ShowData();

        return 0 ;

}
```

Question # 6: Create a **Job** class that holds a ***Job ID*** number and the ***cost of the Job***. Create a **JobException** class that holds ***Job Ecost*** and ***an error message***.When the user enters Job data, if the Job fee is below ***$250***, then create a JobException object and throw it. Write a **main()**function that declares a Job objects. Save the file as your **name**.

Answer # 6:

```
#include <iostream>
#include <exception>
using namespace std;
class Job{
    public:
    int JobID;
    float CostOfJob;
};
class JobException: public exception{
    public:
    float JobECost;
    char* errorMessage = "Job cost must be greater then 250$";
    virtual const char* what() const throw(){
    return errorMessage;
  }
};

int main () {
  try{
     Job j;
     cout<<"Enter job ID: ";  cin>>j.JobID;
     cout<<"Enter Cost of Job: "; cin>>j.CostOfJob;
     if(j.CostOfJob<250){
        throw JobException();
     }
  }
  catch (exception& e){
     cout<<e.what() << '\n';
     }
  return 0;
}
```

Question # 7: Develop a class **BSCS** that has data members ***year(type int), Classstrength (type int)***and ***college(type String).***The class also has a ***Constructor initialize these data members*** and an ***Input*** method. Create **three objects of this class** and store them into a **binary file "BSCS.bin".**

Answer # 7:

```
#include <iostream>
#include<string>
using namespace std;
class BSCS{
    public:
    int year;
    int classStrength;
    string college;
    BSCS(int year, int classStrength, string college){
        this->year = year; this->classStrength = classStrength;
        this->college = college;
    }
    void inputData(){
     cout<"Enter year: "; cin>>year;
     cout<<"Enter class strength"; cin>>classStrength;
     cout<<"Enter college"; cin>>college;
    }

};

int main () {
  BSCS o1(2015,30,"Ithub");
  BSCS o2(2014,7,"ILM");
  BSCS o3(0,0," ");
  o3.inputData();
ifstream in("BSCS.bin".c_str());
  in>>o1;in>>o2;in>>o3;
  //if you want to use read:
  //in.read(reinterpret_cast<const char*>(&o1),sizeof(o1));
in.close();
  return 0;
}
```

Question # 8: Create a function called **swaps ()** that interchanges the values of the two arguments sent to it. (*You will probably want to pass these arguments by reference*.) Make the *function into a template*, so it can be used with all numerical data types (***char***, ***int***, ***float***, and ***so on***). Write a main () program to exercise the function with several types. Write a **main()** program to exercise the function with several types

Answer # 8:

```
#include <iostream>
using namespace std;
//template function to swap values
template <class type> void swap(type *a, type *b){
    type temp = *a;
    *a = *b;
    *b = temp;
}//template function swap
int main(){
    int i1 = 5, i2 = 10;
     cout<< "i1= "<<i1<<" i2= "<<i2<<endl;
     swap<int>(&i1,&i2);//pass by reference
     cout<< "After swaping\ni1= "<<i1<<" i2= "<<i2<<endl;
     cout<<"________________________"<<endl;
     double d1 = 10.01, d2 = 30.3;
     cout<< "d1= "<<d1<<" d2= "<<d2<<endl;
     swap<double>(&d1,&d2);
     cout<< "After swaping\nd1= "<<d1<<" d2= "<<d2<<endl;
     cout<<"________________________"<<endl;
     char c1='3', c2= '4';
     cout<< "c1= "<<c1<<" c2= "<<c2<<endl;
     swap<char>(&c1,&c2);
     cout<< "After swaping\nc1= "<<d1<<" c2= "<<c2<<endl;
    return 0;
}//end main()
Note: code was written and tested with online c++ compiler
```

http://www.cpp.sh/

Question # 9:Create an interface BuildInterface include two abstract methods *GetData()* and *PutData()*.Create a Building class contains fields for square footage (SqurFeets as float); building stories(BStories as int) and implements all BuildInterface. Create two subclasses Houseclass contain additional field for number of bedrooms(int) and Schoolclass contain additional field fornumber of classrooms (int). All subclasses contain appropriate constructors and overridesuperclass methods (GetData and PutData). Write a main() that creates both a Building (School or House) and display their fields.