Disclaimer: I DO NOT claim or guarantee whether the answer written here is 100% accurate. If you feel something is not accurate or out-of-order, it will be a great help, if you let me know. You know how to find me.

Note: Those question which are crossed (i.e. strike-through) refer to the repeated question which has been solved/answered already.

Tips: Don't try to read this pdf using mobile Its pointless and time wasting activity. Use your laptop. Read it in 16:9 full-screen format. Use **Chrome Browser** to read properly.

Author: Pranav Bhattarai

2019 Spring

1 a) Differentiate between Procedural oriented and Object-Oriented programming. Explain different types of OO concepts.

1 b) What do you mean by friend function? Write a program to add private data of two different classes using friend function.

2 a) Define constructor and destructor. Explain the constructor overloading with an example.

2 b) Write down the difference between function overloading and overriding with proper examples.

3 a) Write a C++ program to create a class "Furniture" which reads and displays Furniture related information. Create another class "Chair" which is derived from "Furniture" class, it class reads and displays Chair related information. Create another class "Table" which is also derived from the "Furniture" class and this class reads and displays information related to "Tables". Read Chair and Table class information and displays those information.

3 b) What do mean by operator overloading? Write a program to overload unary minus operator in C++.

4 a) Write a program with class which has hours and minutes as data members. Use conversion routine to convert data of class to seconds.

4 b) What do mean by dynamic memory allocation and de-allocation? Explain, how to allocate and de-allocate memory at run time.
Dynamic memory allocation refers to performing memory allocation manually by programmer. Dynamically allocated memory is allocated on Heap and non-static and local variables get memory allocated on Stack

In C++, there are two operators available for the dynamic memory allocation and de-allocation; the new operator for allocation and delete for the de-allocation.

5 a) What is static binding? How do you achieve static and dynamic binding? Explain with examples.
5 b) Explain the need of virtual function. Write a program to implement run time polymorphism in C++.
6 a) What do you mean by exception? Why is it necessary to handle exception? Explain with examples.
6 b) Define template. Explain template function overloading with examples.
7. Write short notes on **any two**:
a) Multi level inheritance
b) Static data member
c) Class Template

2018 Spring

1 a) Why do you need OOP? Explain any five features of OOP.
1 b) What is difference between class and object? Explain different access specifier with suitable examples.
2 a) Why do you need constructor and destructor in a program? Explain different access specifier with suitable examples.
2 b) Create a class **complex** with two data types (real, imag). Provide the method of adding and multiplying two complex numbers passed as arguments to those functions and returning the new complex number.
3 a) What do you mean by inheritance? What are the different types of inheritance? Explain in brief.
3 b) Create a class **Employee** with data members *NAME*, *IDNUM* & *ADDRESS*. Create another class **MANAGER** with data members *TITLE* and *SALARY*. Create another class **AUTHOR** with data members *BOOK_NAME* and *PRICE*. Inherit EMPLOYEE class to MANAGER and AUTHOR class. Use GETDATA() in every class as member function to get the required data and PUTDATA() to show every data members.

4 a) What do you mean by data type conversion? Explain the conversion from basic type to class type.
4 b) Create a class **String** and overload the operator **+** to concatenate two strings using the statement s3=s1+s2, where s1, s2, s3 are objects of type **String**.
5 a) Explain how dynamic objects are created and destroyed using **new** and **delete** operator.
5 b) What is polymorphism? Explain different types of polymorphism you studied in C++ with an example.
6 a) What are the advantage of generic programming? Explain using a function template with an example.
6 b) What do you mean by exception handling? Explain the meaning of throwing an exception, try block and catch block with a suitable example.
7. Write short notes on **any two**:
a) **Static data member**
Static data members are class members that are declared using the static keyword. There is only one copy of the static data member in the class, even if there are many class objects. This is because all the objects share the static data member. The static data member is always initialized to zero when the first class object is created.

The syntax of the static data members is given as follows –

```
static data_type data_member_name;
```

For example:

```
static int phoneNumber;
```

A program to demonstrate static data member usage is written below:

b) **This pointer**
c) **Friend function**
A friend function, that is a "friend" of a given class, is a function that is given the same access as methods to private and protected data. A friend function is declared by the class that is granting access, so friend functions are part of the class interface, like methods.

A friend function is declared by the class that is granting access, so friend functions are part of the class interface, like methods. Friend functions allow alternative syntax to use objects, for instance f(x) instead of x.f(), or g(x,y) instead of x.g(y). Friend functions have the same implications on encapsulation as methods.

A similar concept is that of friend class.

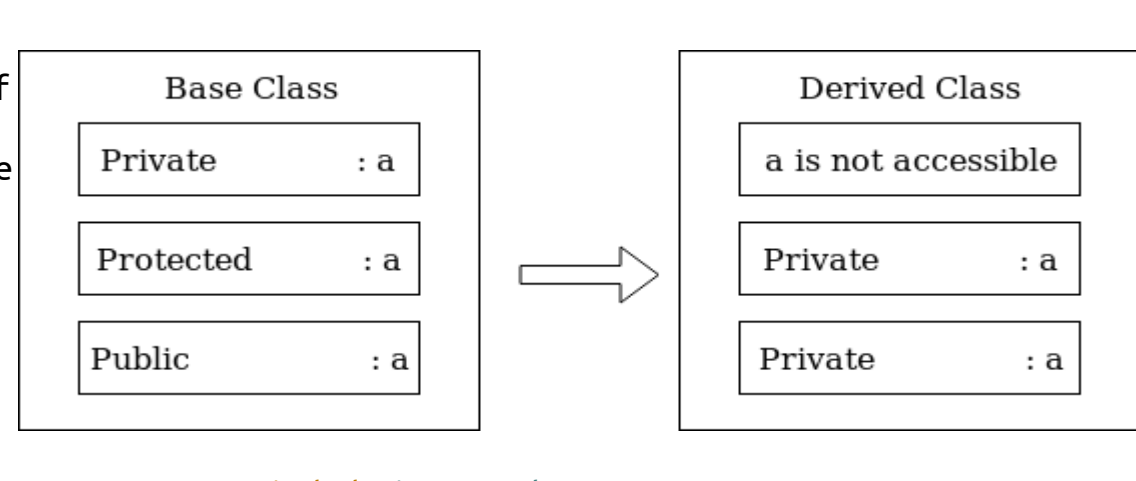
This approach may be used in friendly function when a function needs to access private data in objects from two different classes. This may be accomplished in two similar ways:
- a function of global or name-space scope may be declared as friend of both classes
- a member function of one class may be declared as friend of another one.

2017 Spring

1 a) What are the limitation of procedural oriented language? Why OOP is dominant over procedural language?
The limitation of procedural oriented language are:
i) There is no access specifier.
ii) Adding new data and function is not easy.
iii) It does not have any proper way for hiding data. So it is less secure.
iv) Overloading is not possible.

OPP is dominant over procedural language because of following reasons:
i) Adding new data and function is easy because of the use of objects and classes.
ii) It provides data hiding features like encapsulation, which procedural programming lacks.
iii) It is based on *real world* which makes it easy to understand and makes it very beginner friendly.

1 b) Describe access specifier used in C++ with appropriate example.
C++ access specifiers are used for determining or setting the boundary for the availability of class members (**data members** and **member functions**) beyond that class.
The class members are grouped into three sections i.e. **private**, **protected** and **public**. These keywords are called access specifiers which define the accessibility or visibility level of class members. By default the class members are private. So if the visibility labels are missing then by default all the class members are private.



In C++, there are three access specifiers:
i) **Public** - members are accessible from outside the class.

ii) **Private** - members cannot be accessed (or viewed) from outside the class.
If private access specifier is used while creating a class, then the public and protected data members of the base class become the private member of the derived class and private member of base class remains private.

In this case, the members of the base class can be used only within the derived class and cannot be accessed through the object of derived class whereas they can be accessed by creating a function in the derived class.

iii) Protected - members cannot be accessed from outside the class, however, they can be accessed in inherited classes.

```
#include<iosteram.h>
#include<iostream>
using namespace std;

class Base{
    private:
    int x;
    protected:
    int y;
    public:
    int z;

    Base(){
        x =5;
        y=10;
        z=15;
    }
};

class Derived : Base{
    public:
    void showData(){
        //cout<<x;          //throws an error
        cout<<y<<endl;
        cout<<z<<endl;
}
};

int main(){
    Derived object;
    object.showData();
}
```

A program to demonstrate Public, Private and Protected access specifier

2 a) What is constructor and destructor? Describe constructor overloading with possible example.
A **constructor** in C++ is a special method that is automatically called when an object of a class is created. The constructor has the same name as the class, it is always public, and it does not have any return value.

Destructor is a special member function that is executed automatically when an object is destroyed that has been created by the constructor. C++ destructors are used to de-allocate the memory that has been allocated for the object by the constructor.

The structure of constructor and destructor in syntax is like this:

```
class class_name
{
   public:
     class_name();   //constructor.
    ~class_name();   //destructor.
}
```

Unlike constructor, a destructor neither takes any arguments nor does it returns value. And destructor can't be overloaded.
Note: Remember that more than one destructor can't be used in a program. Only single destructor is allowed.

Constructor Overloading refers to having more than one constructor defined in a class. In this condition, every constructor has same name as class but they differ in terms of either number of arguments or the data-types of the arguments or the both.

A very simple program of constructor overloading is written where there are two constructors have been defined. The first one is invoked when no arguments is passed in `A obj`. The second one is invoked when we pass one integer value as an argument as the constructor has one integer parameter.

```
#include<iostream>
using namespace std;
class A{
  public:
  int x;

  A() {               //constructor with no argument
    x=0;
  }

  A(int b) {          //constructor with an argument
    x=2*b;
  }

  void display(){
    cout<<"\n Value of x is: "<<x<<endl;
  }
};

int main(){
  A obj;              //First constructor is called using a object named "obj"
  A obj2(8);          //Second constructor is called using an object named "obj2"
  obj.display();
  obj2.display();
}
```

2 b) What is advantage of using inline function? Demonstrate with example.
Advantages of inline function are:-
i) It does not require function calling overhead.
ii) It also save overhead of variables push/pop on the stack, while function calling.
iii) It also save overhead of return call from a function.
iv) It increases locality of reference by utilizing instruction cache.
v) After in-lining compiler can also apply intra-procedural optimization if specified. This is the most important one, in this way compiler can now focus on dead code elimination, can give more stress on branch prediction, induction variable elimination etc.

The syntax goes like this:

```
inline data_type function_name(arguments_list);
```

An program to demonstrate the advantage of inline function is given below:

In this program, we created an inline function named **Max**, its functionality is transferred in the main function without transferring the control out of main function, which makes the execution time to decrease. This speeds up the process.

Note: even if we not put use inline function, the program works fine. But the only difference that is while using normal function, the control is passed out outside of the main function, and calculate the value and return to main function. But this doesn't happen while using inline function.

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

inline int Max(int x, int y) {                //incline function
  return (x > y) ? x : y;
}

int main() {
  cout << "Maximum number is: " << Max(20,10) << endl;   //inline code is inserted here. I.e control
  cout << "Maximum number is: " << Max(40,30) << endl;   // is not passed outside of the main
function.
  return 0;
}
```

//Output
Maximum number is 20
Maximum number is 40

3 a) Explain types of inheritance in details.
C++ supports five types of inheritance. They are:

i. Single inheritance
If a single class is derived from one base class then it is called single inheritance.
In C++ single inheritance base and derived class exhibit one to one relation.

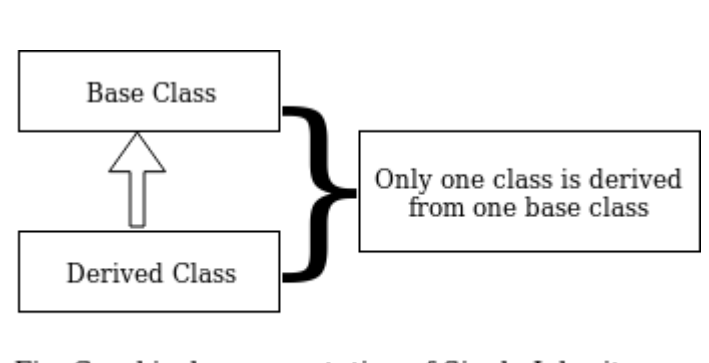


Fig: Graphical representation of Single Inheritance

As shown in the figure, in C++ single inheritance only one class can be derived from the base class.

```
#include<iostream>
using namespace std;
class BaseClass{                                   //One base class
  public:
  int a;
      void askData(){
          cout<<"Enter value of a: "; cin>>a;
      }
};

class DerivedClass : public BaseClass {     //base class being inherited by another class.
  int b;
  public:
      void askNextData(){
          cout<<"Enter value of b: "; cin>>b;
      }
      void nowCalculate(){
          cout<<"Multiplication : "<<a*b<<endl;
      }
};

int main(){
  DerivedClass obj;                          //Object is created under derived class
  obj.askData();
  obj.askNextData();
  obj.nowCalculate();
}
```

In this program class derive is publicly derived from the base class base. So the class derive inherits all the protected and public members of base class base i.e the protected and the public members of base class are accessible from class derive.

ii. Multilevel inheritance
If a class is derived from another derived class then it is called multilevel inheritance. So in multilevel inheritance, a class has more than one parent class. To simplify this I have made an diagram and wrote a program which can clarify this.

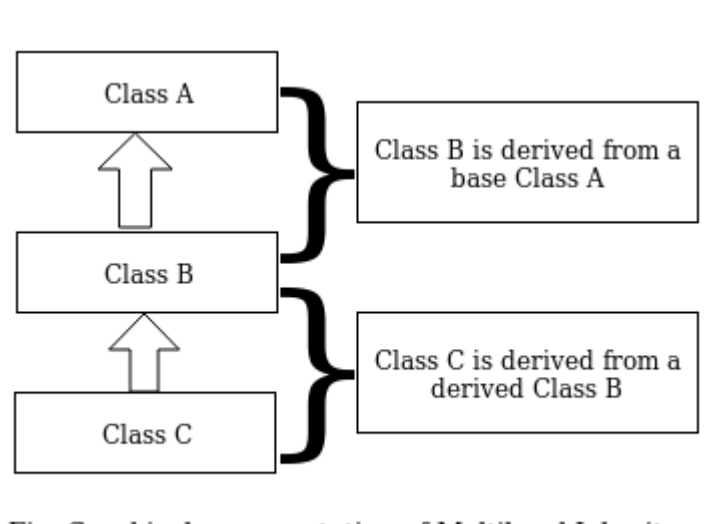


Fig: Graphical representation of Multilevel Inheritance

```
#include<iostream>
using namespace std;
class A{
  public:
  int a =8;
  }
};

class B : public A{    //base class being inherited
  public:
   int b;
   void getB(){
   cout<<"Enter b value: ";cin>>b;
   }
};

class C : public B{    //derive class is being inherited
    public:
    void addThem(){
       cout<<a+b<<endl;
   }
};

int main(){
   B obj1;
   C obj2;
   obj1.getB();
   obj2.addThem();
}
```

iii. Hierarchical inheritance
When several classes are derived from common base class it is called hierarchical inheritance. In this inheritance, the feature of the base class is inherited onto more than one sub-class. For example, a car is a common class from which Audi, Ferrari, Mercedes, etc can be derived.

Following block diagram highlights its concept:

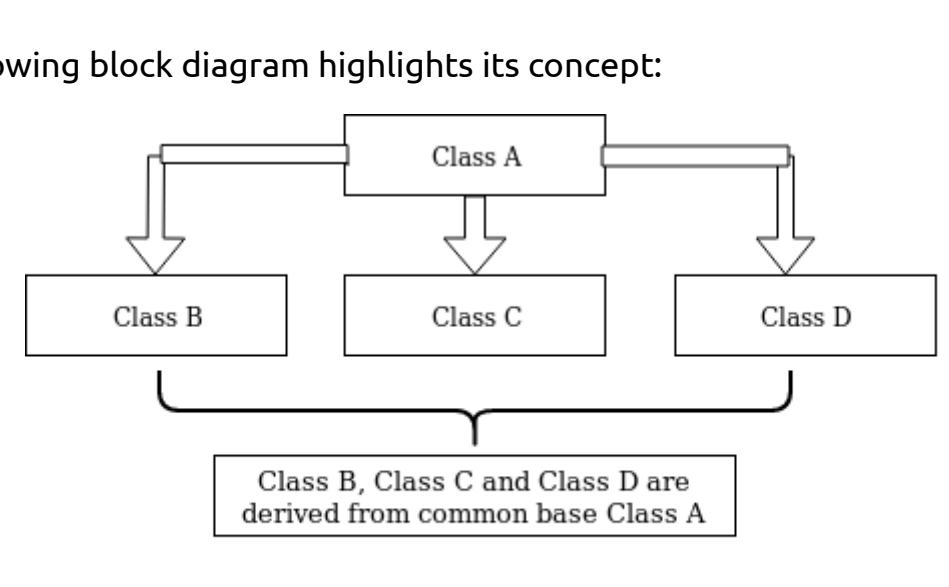
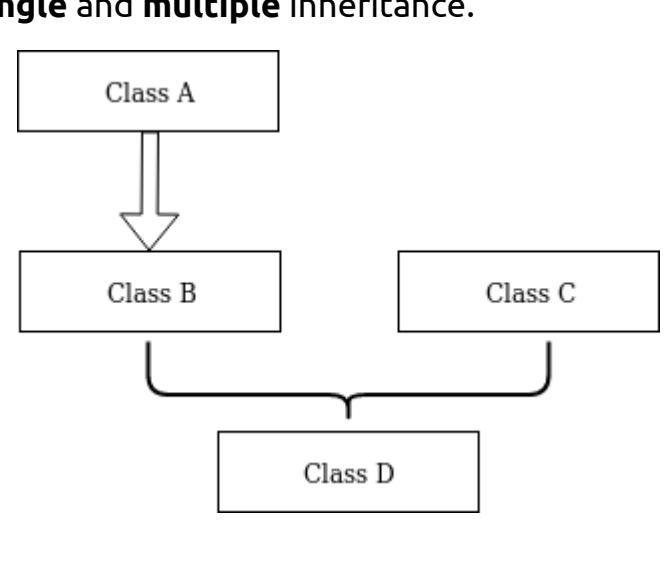


Fig: Graphical representation of Hierarchical Inheritance

```
#include<iostream>
using namespace std;
class BaseClass{
  public:
  float a;
  void dataLinxu(){    //One base class
    cout<<"Enter value of A: "; cin>>a;
  }
};

class deriveClass1 : public BaseClass {    //First derived class inheriting base class
  public:
  int b;
  void add(){
    cout<<"Enter value of B: "; cin>>b;
    cout<<"A + B = "<<a+b<<endl;
  }
};

class deriveClass2 : public BaseClass{    //Second dervied class inheriting base class
  public:
  int c;
  void sub(){
    cout<<"Enter value of C: "; cin>>c;
    cout<<"A - C: "<<a-c<<endl;
  }
};

int main(){
  deriveClass1 Object2;     //derive class is being made object
  deriveClass2 Object3;     //derive class is being made object
  Object2.dataLinxu();
  Object2.add();
  Object3.dataLinxu();
  Object3.sub();
}
```

iv. Hybrid inheritance
The inheritance in which the derivation of a class involves more than one form of any inheritance is called hybrid inheritance. Basically C++ hybrid inheritance is combination of two or more types of inheritance. It can also be called multi path inheritance.

Following block diagram highlights the concept of hybrid inheritance which involves **single** and **multiple** inheritance.



According to above block diagram, I have created a simple simple program to demonstrate this.

```
#include<iostream>
using namepsace std;
class A{
        public:
                int x;
};
class B: public A{
        public:
                Child(){
                x =5;
        }
};
class C{
        public:
                int y;
};
class D : public C, public B{
        public:
        int added;
        void sum() {
                y =6;
                added=x+y;
                cout<<added;
        }
};
int main(){
                D obj;
                obj.sum();
}
```

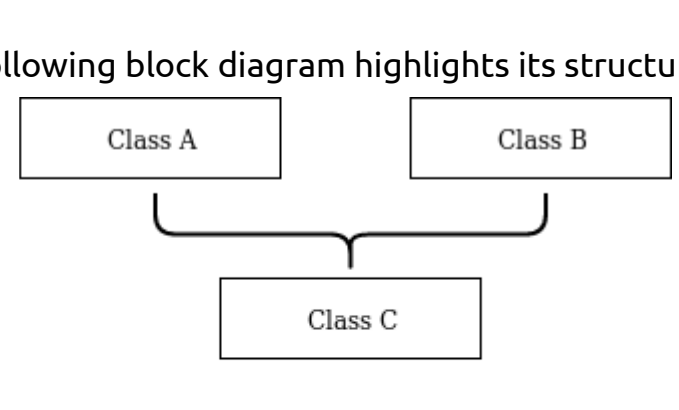
v. Multiple Inheritance
If a class is derived from two or more base classes then it is called multiple inheritance. In C++ multiple inheritance a derived class has more than one base class.
You might me wondering how multilevel and multiple is different when they sounds so similar. The answer is, in multilevel inheritance, we have multiple parent classes whereas in in multiple inheritance we have multiple base classes.

To put it in simple words, in multilevel inheritance, a class is derived from a class which is also derived from another base class. And these levels of inheritance can be extended. On the contrary, in multiple inheritance, a class is derived from two different base classes.

For example
Multilevel inheritance : Inheritance of characters by a child from father and father inheriting characters from his father (grandfather).
Multiple inheritance : Inheritance of characters by a child from mother and father.

```
#include<iostream>
using namespace std;
class A{
        public:
                int x=5;
};
class B {
        public:
                int y=10;
};
class C: public A, public B{
   public:
     int sum;
     void add(){
       sum=x+y;
       cout<<sum;
     }
};
int main(){
                C obj;
                obj.add();
}
```

Following block diagram highlights its structure.



Multiple Inheritance

As shown in above block diagram, class C is derived from two base classes A and B.

3 b) What is inheritance? What is function overriding? Give example.
Inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. In such way, we can reuse, extend or modify the attributes and behaviors which are defined in other class.

```
class derived_class_name :: visibility-mode base_class_name
{
   // body of the derived class.
}
```

4 a) What is operator overloading? Describe unary operator overloading with example.

4 b) What is type casting? Explain the types of type casting used in C++.

5 a) Illustrate the use of the pointer with example.

5 b) What is pure virtual function? What is the use of virtual function in C++ programming language?

6 a) How exception is handled in C++? Illustrate with example.

6 b) What is generic function? Explain.

7. Write short notes on any two:
a) Compile time Vs Runtime exception handling
b) Template function overloading
Function Template is just like a normal function, but the only difference is while normal function can work only on one data type and a function template code can work on multiple data types.
Function templates are always more useful as we have to write the only program and it can work on all data types.

c) New and delete operator
In C++, there are two operators available for the dynamic memory allocation and de-allocation; the new operator for allocation and delete for the de-allocation.

Syntax to use new operator:

```
pointer-variable = new data-type;
```

Here, pointer-variable is the pointer of type data-type. Data-type could be any built-in data type including array or any user defined data types including structure and class.

An Example:

```
#include<iostream>
using namespace std;
struct MeroClass{
  MeroClass(){
    cout<<"MeroClass constructed \n";
  }
  ~MeroClass(){
    cout<<"MeroClass destructed \n";
  }
};
int main(){
  MeroClass *pointerHo;
  pointerHo = new MeroClass[2];
  delete[] pointerHo;
  return 0;
}
```

Output:

MeroClass constructed
MeroClass constructed
MeroClass destructed
MeroClass destructed

2017 Fall

1 a) Why do you need Object-Oriented Programming? Explain any five striking features of OOP.

1 b) What is the difference between class and object? Explain different access modifiers with suitable examples.

2 a) Write the difference between constructor and destructor? Explain difference types of constructor with examples.

2 b) Create a class **time** with required data members and member function to display the time format in HH:MM:SS after adding two time objects given by user and return new time object.

3 a) What do you mean by inheritance? What are the different types of inheritance, explain.

3 b) Create a class EMPLOYEE with data members NAME, IDNUM, ADDRESS. Create another class AUTHOR with data members BOOK_NAME and PRICE. Inherit EMPLOYEE class to MANAGER and AUTHOR class. Use GETDATA() in every class as member function to get the required data and PUTDATA() to show every data members.

4 a) What do you mean by data type conversion? Explain the conversion from class type to basic type.

4 b) Create a class **distance** and overload < to compare two distance object.

5 a) Explain how dynamic objects are created and destroyed using **new** and **delete** operators.

5 b) What is polymorphism? Explain different types of polymorphism you studied in C++ with example.

6 a) What do you mean by late binding? Write example to show late binding.

6 b) What do you mean by exception handling? Explain the meaning of throwing an exception, try block and catch block with a suitable example.

7. Write short notes on **any two:**
a) Function overriding

```
#include <stdio.h>
using namespace std;

void print(char const* myString) {
   printf("\n String %s\n", myString);
}

void print(int myInt) {
   printf("\n My int is %d \n", myInt);x`
}

int main() {
   print("Hello"); // Resolves to void print(const char*)
   print(15); // Resolves to void print(int)
}
```

b) This pointer
c) Friend function
d) Template