**Mid Semester Examination – 2019**

**School of Computer Engineering**

**Kalinga Institute of Industrial Technology (KIIT) Deemed to be University**

**Subject: Object Oriented Programming , Code :IT-2005**

Time: 1.5 Hrs Full Marks: 20

*(Answer any Four Questions including Question No. 1)*

**1. (a). What is a reference variable? What is its major use? [1]**

Ans. A reference variable is an alias, that is, another name for an already existing variable. Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.
You would use it when you wanted to just pass the value around instead of copying the same variable into memory at a different location. It's not a big deal for small programs to copy, but big chunks of memory being copied can slow down programs a lot.
Reference variable is nothing but a wrapper around a pointer with some additional and some limited functionality to provide security in the program.

**(b) When will you make a function inline? Why? [1]**

Ans. Large functions leads to larger executables, which significantly impairs performance regardless of the faster execution that results from the calling overhead. so when the function instructions/code are small and called very often we can declare a function inline.

**(c) How can you resolve the ambiguity arising due to multiple inheritance in C++? [1]**

Ans. The most obvious problem with multiple inheritance occurs during function overriding. Suppose, two base classes have a same function which is not overridden in derived class. If you try to call the function using the object of the derived class, compiler shows error. It's because compiler doesn't know which function to call. For example

```
class base1
{
  public:
    void someFunction( )
     { .... ... .... }
};
class base2
{
   void someFunction( )
     { .... ... .... }
};
class derived : public base1, public base2
{

};
int main()
{
   derived obj;
   obj.someFunction() // Error!
}
```

This problem can be solved using scope resolution function to specify which function to class either base1 or base2

```
int main()
{
   obj.base1::someFunction( );  // Function of base1 class is called
   obj.base2::someFunction();   // Function of base2 class is called.
}
```

**(d) Find output of the program assuming no compilation errors.** **[1]**

```
#include<iostream>
 using namespace std;
 class Demo{
};
int main(){
Demo d1;
cout<<sizeof(d1);
return 0;
}
```

Ans. 1

**(e)Find output/errors of the program.** **[1]**

```
#include<iostream>
using namespace std;
int fun (int x, int y)
{
return x + y ;
}
double fun (int x, int y)
{
return x * y ;
}
int main()
{
cout<<fun(5 , 10);
return 0;
}
```

Ans. Compiler error: call of overloaded ‘fun(int,int)’ is ambiguous.
In function overloading function name should be same and number of arguments should be different.

---

**2(a) What is a Class? How does it accomplish data hiding? Define and implement a student class with data members and member functions, that you feel most essential. [2.5]**

Ans. **Class:** A class is a template to create objects. classes are the framework for objects, they describe which properties, or attributes they have, and provide functions or methods for the purposes of accessing or modifying these attributes. These accessor methods, laid out by the class, allow you to see certain attributes of the object, without worrying about the internal workings of the object OR the class. This is encapsulation and makes programming a lot easier.

**Data hiding:** It is a technique used in object-oriented programming. It is the one most important OOP mechanism. It hides the details of the class from outside of the class. Methods and variables declared private cannot be access outside of the class. Those can be only accessed internally, it means that they are hidden from the outside:

```
class student
{
        private:
                char name[20],regd[10],branch[10];
                        int sem;
        public:
                void input();
                void display();
                };
void student::input()
{
        cout<<"Enter Name:";
        cin>>name;
        cout<<"Enter Regdno.:";
        cin>>regd;
        cout<<"Enter Branch:";
        cin>>branch;
        cout<<"Enter Sem:";
        cin>>sem;
}
void student::display()
{
        cout<<"\nName:"<<name;
        cout<<"\nRegdno.:"<<regd;
        cout<<"\nBranch:"<<branch;
        cout<<"\nSem:"<<sem;
}
int main()
{
        student s;
        s.input();
        s.display();
}
```

**(b) Can we pass class objects as function arguments? Explain with the help of an example. [2.5]**

Ans: The objects of a class can be passed as arguments to member functions as well as nonmember functions either by value or by reference. When an object is passed by value, a copy of the actual object is created inside the function. This copy is destroyed when the function terminates. Moreover, any changes made to the copy of the object inside the function are not reflected in the actual object. On the other hand, in pass by reference, only a reference to that object (not the entire object) is passed to the function. Thus, the changes made to the object within the function are also reflected in the actual object.
Whenever an object of a class is passed to a member function of the same class, its data members can be accessed inside the function using the object name and the dot operator.

However, the data members of the calling object can be directly accessed inside the function without using the object name and the dot operator.

To understand how objects are passed and accessed within a member function, consider this example.

```
class Test {
public:
   int a;

   // This function will take
   // an object as an argument
   void add(Test t)
   {
      a = a + t.a;
   }
};

// Driver Code
int main()
{

   // Create objects
   Test t1,t2;

   // Values are initialized for both objects
   t1.a = 50;
   t2.a = 100;

   cout << "Initial Values \n";
   cout << "Value of object 1: " << t1.a
       << "\n& object 2: " << t2.a
       << "\n\n";

   // Passing object as an argument
   // to function add()
   t2.add(t1);

   // Changed values after passing
   // object as argument
   cout << "New values \n";
   cout << "Value of object 1: " << t1.a   << "\n& object 2: " << t2.a<< "\n\n";

   return 0;
}
```

Output

```
Initial Values
Value of object 1: 50
& object 2: 100

New values
Value of object 1: 50
& object 2: 150
```

**3. (a) Write a program to count the number of objects created using static member function. [2.5]**

Ans.

```
#include <iostream>
using namespace std;
class test {
        int code;
        static int count;

public:
        void setcode(void)
        {
                code = ++count;
        }
        void showcode(void)
        {
                cout << "object number :" << code << "\n";
        }
        static void showcount(void)
        {
                cout << "count:" << count << "\n";
        }
};
int test::count;
int main()
{
        test t1, t2;
        t1.setcode();
        t2.setcode();

        test::showcount();

        test t3;
        t3.setcode();

        test::showcount();
        t1.showcode();
        t2.showcode();
        t3.showcode();
        return 0;
}
```

**(b) What is function overloading? How do we achieve function overloading? On what basis, the compiler distinguishes between a set of overloading functions having the same name? [2.5]**

**Answer:**

- Function overloading is a programming concept that allows programmers to define two or more functions with the same name and in the same scope, but having different signature.
- The signature can differ in the number or in the type of arguments, or in both.
- In other words, functions having different number or type (or both) of parameters are known as overloaded functions.

- To overload a function name, It is required to declare and define all the functions with the same name but different signatures, separately before using the functions.
- **<u>Example:</u>**
  - ✓ **Function Overloading: Different Number of Arguments**

```
int SUM(int, int); // Function Prototype to add two integers
int SUM(int, int, int); // Function Prototype to add three integers
-------
-------
// Function definition to add two integer numbers
int SUM(int x, int y) { return x+y;}
// Function definition to add three integer numbers
int SUM(int x, int y, int z) { return x+y+z;}
-------
-------
```

    The function call SUM(2, 3); will return 5 and SUM(2, 3, 4) will returm 9.

  - ✓ **Function Overloading: Different Datatype of Arguments**

```
int SUM(int, int); // Function Prototype to add two integers
float SUM(float, float); // Function Prototype to add two float numbers
-------
-------
// Function definition to add two integer numbers
int SUM(int x, int y) { return x+y;}
 // Function definition to add two float numbers
float SUM(float x, float y) { return x+y;}
-------
-------
```

    The function call SUM(2, 3); will return 5 and SUM(2.5,7.5) will returm 10.

- If the number and type of arguments passed to two or more  functions are same, even though the return type is different, the compiler will throw error as in C++ function overloading does not depend on return type unlike Ada and Swift language.

int SUM(int, int);

float SUplied in the function call. If a suitable function is found, that function is called. "Suitable" in this context means either: An exact match was found or A trivial conversion was performed or An integral promotion was performed or A standard conversion to the desired argument type exists.M(int, int);

The above functions are not overloaded as they are same in all respects except return type.

Overloaded functions are selected for the best match of function declarations in the current scope to the arguments sup

**4 (a) Write a program using friend function to swap the private data of two classes assuming each class to contain one private integer data member and associated member functions for inputting and displaying the data. [2.5]**

Ans.

```
#include<iostream>
using namespace std;
class second;
class first
{
        int  x;
        public:
        void input()
        {
        cin>>x;
        }
        void output()
        {
        cout<<x;
        }
friend void swap (first &,second &);
};
class second
{
        int  y;
        public:
        void input()
        {
        cin>>y;
        }
        void output()
        {
        cout<<y;
        }
friend void swap (first &,second &);
};
void swap(first &p, second &q)
{
int temp=p.x;
p.x=q.y;
q.y=temp;
}
int main()
{ first ob1;
second ob2;
ob1.input();
ob1.output();
ob2.input();
ob2.output();
swap(ob1,ob2);
ob1.output();
ob2.output();
return 0;
}
```

**(b) What do you mean by friend function and friend class? Explain the use of friend function with a suitable example. [2.5]**

**Answer:**

The Friend function and friend class are the techniques used to access the private members of a class by using friend keyword. The common difference between friend function and friend class is that when friend function is used the private class members can be accessed but in friend class, only the names of the friend class is accessed not the private members of the class.

| ASIS FOR COMPARISON | FRIEND FUNCTION | FRIEND CLASS |
|---|---|---|
| Basic | It is a function used with a friend keyword to grant a non-member function access to the private members of a class. | It is a class used with a friend keyword to access the private members of another class. |
| Forward declaration | Must be used. | Not mandatory. |
| Use | A friend function can be used in some situation of operator overloading. | A friend class can be used when a class is created on the top of another class. |

**Friend Function:**
A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.

**Friend Class:**
Friend Class A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class. For example a LinkedList class may be allowed to access private members of Node.

Example of Friend Function:

```
#include <iostream>
 using namespace std;
 class Box {
   double width;

   public:
     friend void printWidth( Box box );
     void setWidth( double wid );};
// Member function definitionvoid Box::setWidth( double wid ) {
   width = wid;}
// Note: printWidth() is not a member function of any class.void printWidth( Box box ) {
   /* Because printWidth() is a friend of Box, it can
   directly access any member of this class */
   cout << "Width of box : " << box.width <<endl;}
 // Main function for the programint main() {
   Box box;

   // set box width without member function
   box.setWidth(10.0);

   // Use friend function to print the wdith.
   printWidth( box );

   return 0;}
```

**5 (a) Write a program to add two complex numbers using constructor.** **[2.5]**
Answer:
#include <iostream>

```
using namespace std;

class complex
{
      float real,imag;
  public:
      complex()
      {
        real=imag=0;
      }
      complex(float r, float i)
      {
         real=r;imag=i;
      }
      complex add(complex ob)
      {
         ob.real = real + ob.real;
         ob.imag = imag + ob.imag;
         return ob;
      }
      void display()
      {
          cout<<real<<"j"<<imag;
      }
};

int main()
{
   float a, b;
   cout<<"\nEnter real and imaginary parts of complex number1:";
   cin>>a>>b;
   complex c1(a,b);
   cout<<"\nEnter real and imaginary parts of complex number2:";
   cin>>a>>b;
   complex c2(a,b);
   complex c3;
   c3=c1.add(c2);
   cout<<"\nComplex Number1:"; c1.display();
   cout<<"\nComplex Number2:"; c2.display();
   cout<<"\nAfter addition Complex Number3:"; c3.display();
   return 0;
}
```

**(b) Explain the characteristics of constructors and destructors with examples? [2.5]**
Ans.
**Constructor**

- A constructor is a member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) create. It is special member function of the class.

- Constructor has same name as the class itself

- Constructors don’t have return type

- A constructor is automatically called when an object is created.
- If we do not specify a constructor, C++ compiler generates a default constructor for us (expects no parameters and has an empty body).
- It is of three types. i) Default Constructor, ii) Parameterised contructur iii) copy constructor

**Default Constructors:** Default constructor is the constructor which doesn't take any argument. It has no parameters.

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

class construct {
public:
	int a, b;

	// Default Constructor
	construct()
	{
		a = 10;
		b = 20;
	}
};

int main()
{
	// Default constructor called automatically
	// when the object is created
	construct c;
	cout << "a: " << c.a << endl
		<< "b: " << c.b;
	return 1;
}
```

**Parameterized Constructors:** It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor's body, use the parameters to initialize the object.

**Copy Constructor:** A copy constructor is a member function which initializes an object using another object of the same class. Detailed article on Copy Constructor.

**Destructor**
Destructor is a member function which destructs or deletes an object.

- A destructor function is called automatically when the object goes out of scope:
  (1) the function ends
  (2) the program ends
  (3) a block containing local variables ends
  (4) a delete operator is called
- Destructors have same name as the class preceded by a tilde (~)

- Destructors don’t take any argument and don’t return anything.

Example :

```
class A{
   public:
   // defining destructor for class
   ~A()
   {
      // statement
   }   };
```

**6 (a) Explain the advantages of inheritance and mention different types of inheritance in C++. Also justify the significance of protected access specifier in inheritance. [2.5]**

Ans: **Advantages of Inheritance**

The most frequent use of **inheritance** is for deriving classes using existing classes, which provides reusability which in turn reduce the development time of software.

The derived classes extend the properties of base classes to generate more dominant objects.

The same base classes can be used by a number of derived classes in class hierarchy.

Types of Inheritance in C++

Single **Inheritance**.

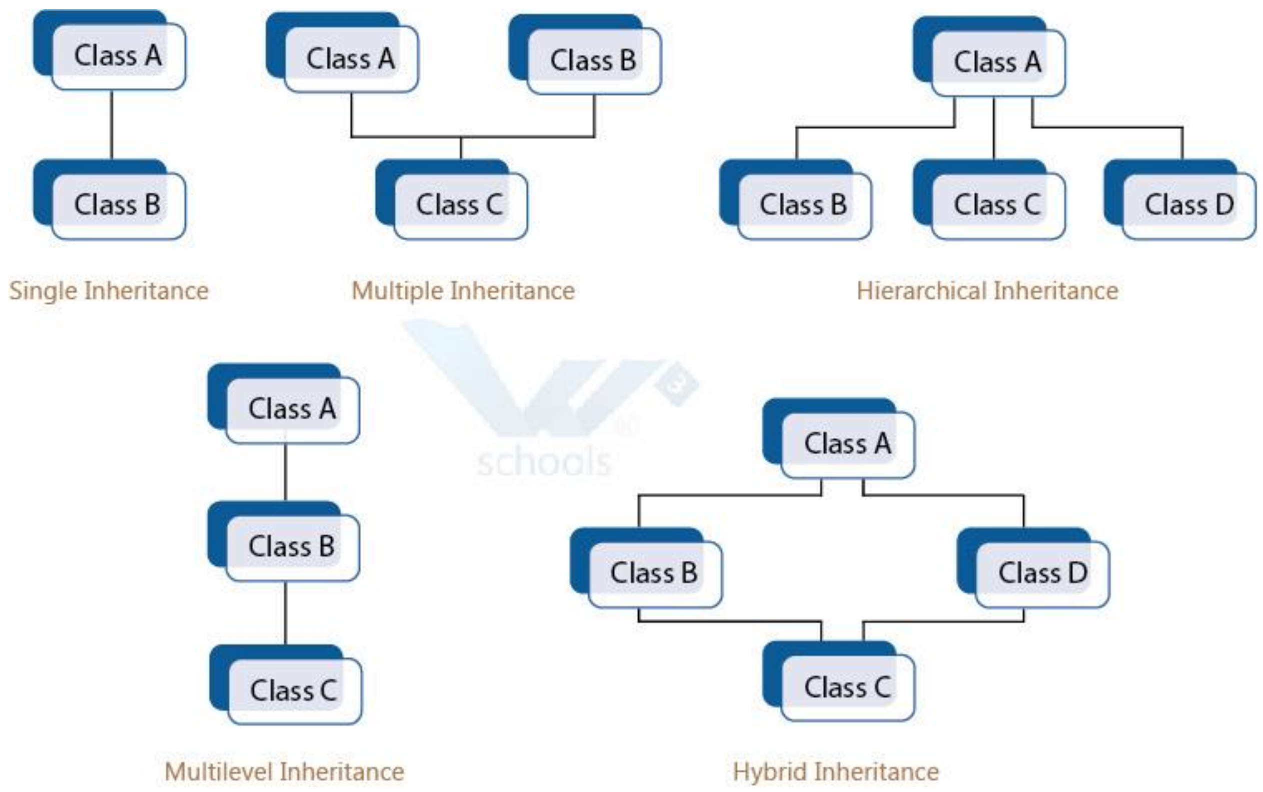
Multilevel **Inheritance**.

Multiple **Inheritance**.

Heirarchical **Inheritance**.

Hybrid **Inheritance**.

Multipath **Inheritance**.



The **protected** access specifier allows the class the member belongs to, friends, and derived classes to access the member. However, protected members are not accessible from outside the class. The public member of the super class becomes behave like protected in the child class.

With a protected attribute in a base class, derived classes can access that member directly. This means that if you later change anything about that protected attribute (the type, what the value means, etc…), you'll probably need to change both the base class AND all of the derived classes. Therefore, using the protected access specifier is most useful when you (or your team) are going to be the ones deriving from your own classes, and the number of derived classes is reasonable. That way, if you make a change to the implementation of the base class, and updates to the derived classes are necessary as a result, you can make the updates yourself (and have it not take forever, since the number of derived classes is limited).

**(b) Write a program to create a class "abc" that stores the first name, roll number and section of a student . Create another class "xyz" that stores the last name and 4 subject marks as its secure data members. Derive a class "studentdetails" from class "abc" and class "xyz". Use necessary member functions for inputting and displaying the full name, roll number, section, all subject marks along with average mark of a student through derived class object. [2.5]**

Ans.

```
#include <iostream>
using namespace std;
#include<string.h>
class abc
{
protected:
char fname[10],section;          //char * can be taken for fname
int roll;
};
```

```
class xyz
{
protected:
char lname[10];               //char * can be taken for lname
int m1,m2,m3,m4;                              //array can be taken
};
class studentdetails:public abc,public xyz
{
float avg;
char fullname[20];                 //char * can be taken for fullname
public:
void input()
{
 cout<<"enter first name"; cin>>fname;
 cout<<"enter last name";  cin>>lname;
 cout<<"enter section";    cin>>section;
 cout<<"enter roll number"; cin>>roll;
 cout<<"enter 4 subject marks"; cin>>m1>>m2>>m3>>m4;
 avg= (m1+m2+m3+m4)/4.0;
 strcpy(fullname,fname);
 strcat(fullname , " ");                        //to add a space in between first name and last name
 strcat(fullname,lname);
 }
void output()                                   //output can be presented in different ways and
order
{
 cout<<"name of the student is  "<<fullname<<endl;
 cout<<"roll nuber is  "<< roll <<"  section is  "<<section<<endl;
 cout<<"4 subject marks are  " <<m1 <<"\t"<<m2<<"\t"<< m3<<"\t"<<m4<<endl;
 cout<<"average mark of the student is  "<<avg<<endl;
}
};
int main()
{
 studentdetails s;
s.input();
s.output();
}
```