Assignment - 17

PAGE No.
DATE / /

1) what is mean by function overloading?

→ function overloading is the one type of overlodding

- function overloading takes place within one class

- function overloading consider as compile time polymorphism

- In case of function overloading we can define multiple function in a class with same name & different prototype.

- function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters.

2) why function overloading is considered as Compile time polymorphism

→

- function overloading is considered compile time polymorphism because the compiler determines which overloaded function to use based on the arguments provided at compile time

method call decision is made before the program actually runs means method call decision takes in compile time.

Q3] what do you mean by name mangling name decoration

→ namemangling or name decoration

- when we compile the code the compiler change the name of every function with mangled name (modified name)
- when overload of the function all the names of the function are same but due to the concept of namemangling the compiler will change the function with new name as per pattern
- According to the ~~mang~~ namemangling there is no concept of overloading after the code gets compiled.

e.g.

int Addition (int no1, int no2)

PAGE No.
DATE / /

Namemangling

Addition @ 2 ii = Addition @ 2 i i

24] why return value is not considered as function overloading criteria

→ The compiler does not consider the return type while differentiating the overloaded method But you cannot declare two function with same name & different return types it will throw a compile-time error

25] why & what is the use of function overloading

→ function overloading allows you to create multiple functions with the same name but different parameter

enabling a single function name to perform different operations depending on the data types or number of arguments provided

Q6) what are the scenarios in which we cannot perform function overloading

→ following are the scenario ~~for~~ th in which we cannot perform function overloading

1) If the ~~class~~ function name is not same as the class name.

2) If all functions parameters are same

3) We cannot overload the Access specifier

4) If there is a different datatype for all the function.

Q7) what are the scenarios in which we can overload the function?

→ following are the scenario for the function overloading

1) name of all functions should be same as a class name

2) Datatype of the parameters of the function should be different

3) no-of parameters should be different.

4) we cannot overload function by changing its return value.

5) Predict the output of the below diagram

```
# include <iostream>
using namespace std;
class Demo
{
   public:
   void fun (int i)
      {
         cout << " first definition";
      }
      void fun (int i, int j)
      {
         cout << " second definitron";
      }
};

int main ()
{
    Demo obj ();
    obj . fun (10);
    obj . fun (10,20);
    return 0;
}
```

output:- First definition Second definition

eg) #include <iostream>
using namespace std;
class Demo
{
public:
void fun(int *p)
{
cout<<" First definition";
}
void fun (float *p)
{
cout<<" second definition";
}
void fun (int no)
{
cout<<" Third definitron";
}
};
int main()
{
int no = 10;
float f = 12.3;
Demo obj ();
obj.fun (no);
obj.fun (&no);
obj.fun (&f);

return 0;
}

output:-

Third definition first definition Second definition

10) Draw object layout & class diagra of below code snippets and explain its internal. Explain the type of inheritance in the below code

```
# include <iostream>
class Base
{
public:
 int i, j;
 static int k;

 Base()
  {
     i = 10;
     j = 20;
  }
 void fun ()
  {
   Cout << " Base fun";
  }

};
```

```
int Base::k = 12;

class Derived : public Base
{
  public:
   int x, y;

   Derived ()
     {
        x = 100;
        y = 200;
     }
   void gun ()
     {
        cout << " Derived Gun";
     }
};

int main ()
{
   Base bobj();
   Derived dobj();
   Cout << sizeof (bobj);
   Cout << sizeof (dobj);

   cout << bobj.i;
   Cout << bobj.j;
   cout << doby.i;
   cout << dobj.j;
   cout << boby.k;
   cout << boby.x;
```

```
baby.fun();
daby.fun();
daby.gun();

return 0;
}
```

In the above code class Base is the parent class having default constructor & fun() method & variable i, j and static variable k.

class Derived inherit the Base class having variable x and y & the constructor & gun() method.

In main function we create the object of Base and Derived class & Access all the functions of both class using object & also display the variables. & calculate the size of both object using sizeof() operator

Single level inheritance

In the above code single level inheritance is used in which class Derived inherit the properites of Base class.

object layout

bobj

100	10	Base::i
104	20	Base::j
108	12	Base::k
112		

dobj

100	10	Base::i
104	20	Base::j
108	12	Base::k
112	100	Derived::x
116	200	Derived::y
120		

class Diagram

Base class

charadterisitics
i, j, k
Behaviours
fun()

Derived class

characteristics
i, j, k, x, y
Behaviours
fun()
gun()