

Assignment - 17

Q.1 What is mean by function overloading?

-
- overloading is a type of compile time polymorphism. (Early Binding)
 - Function overloading is concept in which we make same name function with different parameters.
 - Function overloading = Same name + Diffn. Prototype

◦ for eg -

```
class Demo {
```

```
    int Add (int A, int B) {
```

```
    } //
```

```
    int Add (int A, int B, int C) {
```

```
    } //
```

Q.2 Why Function overloading is considered as compile time polymorphism?

-
- Function overloading is considered as compile polymorphism because we write and modify name & parameter of function at the time of compilation.

- The memory will be allocated at the Run-time but calculation are made at the time of compilation.

3. what is use of function overloading?

-
- Function overloading is used when we want to perform one operation (consider Addition) but prototype is different / data type is different, then programmer doesn't need to remember each function name. He simply use concept of overloading.

4. what are the scenario in which we overload the function.

-
- Suppose we want to perform addition of two numbers & we do not have different parameters with different datatype. In this case we can overload the function.
- as,

```
class Demo {  
Example :   int add (int a, int b) { // }  
             int add (float a, float b) { // }  
             int add (int a, int b, int c) { // }  
};
```


Q5 what is mean by name mangling & name decoration?

→ To be write ..

Q6 why return value is not considered as function overloading criteria?

→ • The return value is not considered as function overloading criteria because function overloading can be determined by function's prototype like name & parameter list.

• for eg `int add (int a, int b)`

Q7 {
 return a+b;
}

`double[int] add (int a, int b)`

{
 return (double) (a+b);
}

Q8 - It generates ambiguity.

`add (5, 10);` // which function should
`add (5, 10);` // Ambiguity. called?

Q.7 what are the scenarios in which we cannot perform function overloading?

→

- Function overloading cannot be performed if ◦ function declarations that differ only in the return type. [Diffn. Retn.]

- Member function declarations with same name & same parameter list cannot be overloaded if any of them is a static member function.

- Parameter declarations that differ only in pointer * versus an array [] are equivalent that is the array declaration is adjusted to become a pointer declaration

```
int fun (int * ptr)
```

```
int fun (int ptr []) // Not allowed
```

- overloading ambiguity. [at time of overloading]

Q.8 Predict the output

```
class Demo
```

```
{ public:  
  void fun (int i)
```

```
{ cout << " First defination" << endl;
```

```
}
```



```

void fun (int i, int j)
{   cout << "Second Defination" << endl;
}
};

```

```

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

```

output : First defination
Second defination

Q.9 Predict output of Below code

→ output : Third defination
First defination
Second defination

Q.10 Draw object layout & class diagram of below code & explain its internal working in detail. Explain which type of inheritance is used in code snippet.

```

→ class Base
{
    public:
        int i, j;
        static int k;
}

Base()
{
    i = 10;
    j = 20;
}

void fun()
{
    cout << "Base";
}
};

```

<pre> int Base :: k = 10; class Derived : public Base { public: int x, y; Derived () { x = 50; y = 60; } void gun () { cout << "Derived gun"; } } </pre>	<pre> int main() { Base bobj; Derived dobj; cout << sizeof(bobj); cout << sizeof(dobj); cout << bobj.i; cout << bobj.j; cout << dobj.i; cout << dobj.j; cout << bobj.k; cout << bobj.x; bobj.fun(); dobj.fun(); dobj.gun(); return 0; } </pre>
---	--

output : Sizeof bobj = 8 10 10 11
 Size of dobj = 16 20 20 60

• Single level Inheritance is used above code.