
FACE

Polymorphism

- As the word suggest, “ many forms”

- Allows the object to behave differently in different conditions

- Different types:

  i. Compile time Polymorphism – Static (or early) binding.

  ii. Runtime Polymorphism – Dynamic (or late) binding.

FACE

Binding

- Connecting the function call to the function body is called Binding.

- When it is done before the program is run, its called Early Binding or Static Binding or Compile-time Binding.

1) Compile time Polymorphism

- Function overloading
- Operator overloading

2) Run time Polymorphism

- Function overriding

  (using Virtual functions)

1) Compile time Polymorphism

- Connecting the function call to the function body is done before the execution

- Different types:
  i. Function overloading
  ii. Operator overloading

FACE

a) Function overloading

- Multiple functions with same names but different parameters.

- Different Ways:
    a) By changing number of Arguments.
    b) By having different types of argument.

```
1  #include <iostream>
2  using namespace std;
3  void print(int i)
4  {
5      cout<<"Integer: "<<i<<endl;
6  }
7  void print(double i)
8  {
9      cout<<"Double: "<<i<<endl;
10 }
11 void print(char* i)
12 {
13     cout<<"Character: "<<i<<endl;
14 }
15 int main()
16 {
17     print(10);
18     print(10.10);
19     print("Ten");
20     return 0;
21 }
22
```

OUTPUT

```
Integer: 10
Double: 10.1
Character: Ten
```

```
1  #include <iostream>
2  using namespace std;
3  void sum(int x, int y)
4  {
5      cout<<x+y<<endl;
6  }
7  void sum(int x, int y, int z)
8  {
9      cout<<x+y+z<<endl;
10 }
11 int main()
12 {
13     sum(10,20);
14     sum(10,20,30);
15     return 0;
16 }
17
18
19
20
21
22
```

OUTPUT

```
30
60
```

Constructor overloading:

- Multiple constructors with same names but different parameters.

Ways to overload a constructor:

- Same name as that of the class name and by changing number of Arguments.

FACE

```
1  class construct
2  {
3      public:
4      float area;
5      construct ()
6
7      {
8          cout<<"0-arg constructor"<<endl;
9          area = 0;
10     }
11     construct (int a, int b)
12
13     {
14         cout<<"1-arg constructor"<<endl;
15         area = a * b;
16     }
17     void disp()
18     {
19         cout<< area<< endl;
20     }
21 };
22
```

```
1  class construct
2  {
3  public:
4      float area;
5      construct ()
6      {
7          cout<<"0-arg constructor"<<endl;
8          area = 0;
9      }
10     construct (int a, int b)
11
12     {
13         cout<<"2-arg constructor"<<endl;
14         area = a * b;
15     }
16     void disp()
17     {
18         cout<< area<< endl;
19     }
20 };
21
22
```

```
1   int main()
2   {
3       construct o;
4       o.disp();
5       construct o2( 10, 20);
6       o2.disp();
7       return 1;
8   }
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

OUTPUT

```
0-arg constructor
0
2-arg constructor
200
```

```
#include<iostream>
using namespace std;
class Base1
{
     public:
     Base1()
     {
cout <<" Base1's constructor called"
endl;
     }
};
class Base2
{
     public:
     Base2()
     {
cout << "Base2's constructor called"
endl;
}
};


class Derived: public Base1, public
Base2
{
    public:
     Derived()
     {
     cout << "Derived's constructor
called" << endl;
     }
};

 int main()
{
   Derived d;
   return 0;
}
```

Question 1

Function overloading is also similar to which of the following?

A) Operator overloading

B) Constructor overloading

C) Destructor over loading

D) None of the mentioned

b) Operator overloading

- A single operator is overloaded to give user defined meaning to it

- There are some C++ operators which we can't overload:

  (.) dot

  (.*) dot-asterisk

  (::)

  (?:)

  (sizeof)

FACE

b) Operator overloading

```
class className
{
        ……
        public
                returnType operator symbol(arguments)
                {
                        ……..
                }
        ……
};
```

FACE

Rules for Operator overloading

1) Only built-in operators can be overloaded. New operators can not be created.

2) Arity of the operators cannot be changed.

3) Precedence and associativity of the operators cannot be changed.

4) Overloaded operators cannot have default arguments except the function call operator () which can have default arguments.

FACE

Rules for Operator overloading

5) Operators cannot be overloaded for built in types only. At least one operand must be used defined type.

6) Assignment (=), subscript ([]), function call (“()”), and member selection (->) operators must be defined as member functions

7) Except the operators specified in point 6, all other operators can be either member functions or a non member functions.

8) Some operators like (assignment)=, (address)& and comma (,) are by default overloaded.

```
1  #include <iostream>
2  using namespace std;
3
4  class Sample
5  {
6      private:
7        int num;
8      public:
9        Sample():num(8){}
10       void operator ++()
11       {
12           num=num+2;
13           cout<<"The Count is: "<<num;
14       }
15 };
16
17
18
19
20
21
22
```

comment/pseudo code/output

```
This is similar to:
Sample()
{
        num = 8;
}
```

```
1  #include <iostream>
2  using namespace std;
3  class Sample
4  {
5      private:
6        int num;
7      public:
8        Sample():num(8){}
9        void operator ++()
10       {
11           num=num+2;
12           cout<<"The Count is: "<<num;
13       }
14 };
15 int main()
16 {
17     Sample S;
18     ++S;
19     return 0;
20 }
21
22
```

OUTPUT

```
The Count is: 10
```

Operator overloading

Example Program for complex subtraction using operator overloading

```
1   #include <iostream>
2   using namespace std;
3   class Complex
4   {
5       private:
6         float real;
7         float imag;
8       public:
9          Complex(): real(0), imag(0){ }
10         void input()
11         {
12             cout << "Enter real and imaginary parts respectively: ";
13             cin >> real;
14             cin >> imag;
15         }
16         Complex operator - (Complex c2)
17         {
18             Complex temp;
19             temp.real = real - c2.real;
20             temp.imag = imag - c2.imag;
21             return temp;
22         }
```

```
1      void output()
2      {
3         if(imag < 0)
4              cout << "Output Complex number: "<< real << imag << "i";
5         else
6              cout << "Output Complex number: " << real << "+" << imag << "i";
7      }
8   };
9
10  int main()
11  {
12      Complex c1, c2, result;
13      cout<<"Enter first complex number:\n";
14      c1.input();
15      cout<<"Enter second complex number:\n";
16      c2.input();
17      result = c1 - c2;
18      result.output();
19      return 0;
20  }
21
22
```

OUTPUT

```
Enter first complex number:
Enter real and imaginary parts respectively: 5 8
Enter second complex number:
Enter real and imaginary parts respectively: 2 5
Output Complex number: 3+3i
```

Question 1

Which of the following operators are overloaded by default by the compiler in every user defined classes even if user has not written?

A) Comparison Operator (==)

B) Assignment Operator (=)

Question 2

Which of the following operator(s) cannot be overloaded?

A) . (Member Access or Dot operator)

B) ?: (Ternary or Conditional Operator)

C) :: (Scope Resolution Operator)

D) All of the above

Can main() be overloaded in C++?

```
#include <iostream>
using namespace std;
int main(int a)
{
	cout << a << "\n";
	return 0;
}
int main(char *a)
{
	cout << a << endl;
	return 0;
}
```

```
int main(int a, int b)
{
	cout << a << " " << b;
	return 0;
}
int main()
{
	main(3);
	main("C++");
	main(9, 6);
	return 0;
}
```

FACE

Can main() be overloaded in C++?

- The above program fails in compilation and produce warnings and errors
- To overload main() function in C++, it is necessary to use class and declare the main as member function
- main is not a keyword, we can declare a variable whose name is main

FACE

Can main() be overloaded in C++?

```
#include <iostream>
int main()
{
      int main = 10;
      std::cout << main;
      return 0;
}
```

Output:

10

FACE

Can main() be overloaded in C++?

```
#include <iostream>
using namespace std;
class Test
{
public:
	int main(int s)
	{
		cout << s << "\n";
		return 0;
	}
	int main(char *s)
	{
		cout << s << endl;
		return 0;
	}
```

```
	int main(int s ,int m)
	{
		cout << s << " " << m;
		return 0;
	}
};
int main()
{
	Test obj;
	obj.main(3);
	obj.main("I love C++");
	obj.main(1, 6);
	return 0;
}
```

Question 1

Which of the following permits function overloading on c++?

A) type

B) number of arguments

C) type & number of arguments

D) number of objects

Question 2

In which of the following we cannot overload the function?

A) return function

B) caller

C) called function

D) main function

Question 3

Function overloading is also similar to which of the following?

A) operator overloading

B) constructor overloading

C) destructor overloading

D) function overloading

Question 4

Overloaded functions are _____

A) Very long functions that can hardly run

B) One function containing another one or more functions inside it

C) Two or more functions with the same name but different number of parameters or type

D) Very long functions

Question 5

What will happen while using pass by reference?

A) The values of those variables are passed to the function so that it can manipulate them

B) The location of variable in memory is passed to the function so that it can use the same memory area for its processing

C) The function declaration should contain ampersand (& in its type declaration)

D) The function declaration should contain $

Question 6

What should be passed in parameters when function does not require any parameters?

A) void

B) blank space

C) both void & blank space

D) tab space

Question 7

What are the advantages of passing arguments by reference?

A) Changes to parameter values within the function also affect the original arguments

B) There is need to copy parameter values (i.e. less memory used)

C) There is no need to call constructors for parameters (i.e. faster)

D) All of the mentioned

Question 8

Which of the following in Object Oriented Programming is supported by Function overloading and default arguments features of C++.

A) Inheritance

B) Polymorphism

C) Encapsulation

D) None of the above

Question 9

What is the output?

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

A) 10

B) 0

C) 20

D) Compiler Error

Question 10

What is the output?

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

A) 5

B) 0

C) 10

D) Compiler Error

Question 11

Pick the other name of operator function.

A) function overloading

B) operator overloading

C) member overloading

D) object overloading

Question 12

Which of the following operators can’t be overloaded?

A) ::

B) +

C) –

D) []

Question 13

How to declare operator function?

A) operator sign

B) operator

C) name of the operator

D) name of the class

Question 14

What is the output?

```
#include <iostream>
using namespace std;
class myclass
{
    public:
    int i;
    myclass *operator->()
    { return this; }
};
int main()
{
    myclass ob;
    ob->i = 10;
    cout << ob.i << " " << ob-
>i;
    return 0;
}
```

Question 14

A) 10 10

B) 11 11

C) Error

D) Run time Error

Question 15

Which of the following statements is NOT valid about operator overloading?

A) Only existing operators can be overloaded

B) The overloaded operator must have at least one operand of its class type

C) The overloaded operators follow the syntax rules of the original operator

D) None of the mentioned

Question 16

Operator overloading is _____

A) making c++ operator works with objects

B) giving new meaning to existing operator

C) making the new operator

D) adding operation to the existing operators

Question 17

What is the output?

```
#include <iostream>
using namespace std;
ostream & operator<<(ostream & i, int n)
{ return i; }
int main() {
    cout << 5 << endl;
    cin.get();
    return 0;
}
```

A) 5

B) 6

C) Error

D) Run time Error

Question 18

What happens when objects s1 and s2 are added?

```
string s1 = "Hello";
string s2 = "World";
string s3 = (s1+s2).substr(5);
```

Question 18

A) Error because s1+s2 will result into string and no string has substr() function

B) Segmentation fault as two string cannot be added in C++

C) The statements runs perfectly

D) Run-time error

Question 19

What is operator overloading in C++?

A) Overriding the operator meaning by the user defined meaning for user defined data type

B) Redefining the way operator works for user defined types

C) Ability to provide the operators with some special meaning for user defined data type

D) All of the mentioned

Question 20

What is the syntax of overloading operator + for class A?

A) A operator+(argument_list){}

B) A operator[+](argument_list){}

C) int +(argument_list){}

D) int [+](argument_list){}

THANK YOU

FACE