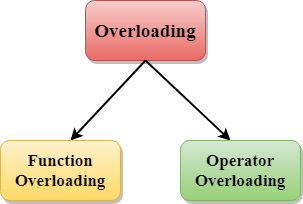
**Overloading in C++**

Types of overloading in C++ are:

1.Function overloading

2.Operator overloading



**Operator Overloading in C++**

An operator is a symbol that tells the compiler to perform specific task. Every operator have their own functionality to work with built-in data types. Class is user-defined data type and compiler doesn't understand how to use operators with user-defined data types. To use operators with user-defined data types, they need to be overload according to a programmer's requirement.

Operator overloading is a way of providing new implementation of existing operators to work with user-defined data types.

An operator can be overloaded by defining a function to it. The function for operator is declared by using the operator keyword followed by the operator.

**Syntax for C++ Operator Overloading**

To overload an operator, we use a special **operator** function. We define the function inside the class or structure whose objects/variables we want the overloaded operator to work with.

class className

{

... .. ...

public : returnType **operator symbol** (arguments)

{

... .. ...

}

... .. ...

};

**There are some rules for the operator overloading**.

1. Only built-in operators can be overloaded. If some operators are not present in C++, we cannot overload them.
2. The arity of the operators cannot be changed
3. The precedence of the operators remains same.
4. The overloaded operator cannot hold the default parameters except function call operator “( )”.
5. We cannot overload operators for built-in data types. At least one user defined data types must be there.

Overloadable/Non-overloadableOperators

Following is the list of operators which can be overloaded −

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| + | - | \* | / | % | ^ |
| & | | | ~ | ! | , | = |
| < | > | <= | >= | ++ | -- |
| << | >> | == | != | && | || |
| += | -= | /= | %= | ^= | &= |
| |= | \*= | <<= | >>= | [] | () |

Following is the list of operators, which can not be overloaded −

|  |  |  |  |
| --- | --- | --- | --- |
| :: | .\* | . | ?: |
| -> | ->\* | new | new [] | delete | delete [] |

**There are two types of operator overloading in C++**

1. Unary Operator Overloading

2. Binary Operator Overloading

Overloading Binary Operator

**Unary Operators Overloading in C++**

The operator which required only single operator to performed its operation is known as unary operator

Following example explain how minus (-) operator can be overloaded Exam

Example1:

class Data

{

private: int a,b;

public : Data()

{

a=10;

b=20;

cout<<”\n A=>”<<a;

cout<<”\n B=>”<<b;

}

void **operator** – ()

{

int x,y;

x=- a;

y=- b;

cout<<”\n X=>”<<x;

cout<<”\n Y=>”<<y;

}

}

void main()

{

clrscr();

Data d;

-d; //or d.operator – ( );

getch();

}

Output

A=>10

B=>20

X=>-10

Y=>-20

Binary Operators Overloading in C++

The operator which required only two operator to performed its operation is known as binary operator

Following example explain how minus (-) operator can be overloaded Exam

Example 2

class Data

{

private: int a,b;

public : Data()

{

a=10;

b=20;

cout<<”\n A=>”<<a;

cout<<”\n B=>”<<b;

}

void operator + ( Data &d2)

{

Data d3;

d3.a = a+d2.a;

d3.b = b+d2.b;

cout<<”\n A=>”<<d3.a;

cout<<”\n B=>”<<d3.b;

}

}

void main()

{

clrscr();

Data d1,d2;

d1+d2; //or d1.operator + ( d2 );

getch();

}

Output

A=>10

B=>20

A=>10

B=>20

A=>20

B=>40

**Exam no 1**

#include<iostream.h>

#include<conio.h>

/\* Unary operator overloading program \*/

class OpData1

{

private: int a,b;

public: OpData1()

{

a=10;

b=20;

cout<<"\n A=>"<<a;

cout<<"\n B=>"<<b;

}

void operator - ()

{

int x,y;

x=-a;

y=-b;

cout<<"\n X=>"<<x;

cout<<"\n Y=>"<<y;

}

};

void main()

{

clrscr();

OpData1 d;

d.operator -();

getch();

}

**Exam no2**

#include<iostream.h>

#include<conio.h>

/\* Unary operator overloading program \*/

class OpData2

{

private: int a,b;

public: OpData2()

{

a=10;

b=20;

cout<<"\n A=>"<<a;

cout<<"\n B=>"<<b;

}

void operator - ()

{

int x,y;

x=-a;

y=-b;

cout<<"\n X=>"<<x;

cout<<"\n Y=>"<<y;

}

void operator +()

{

cout<<"\n bit ballarpur";

}

};

void main()

{

clrscr();

OpData2 d;

d.operator -();

d.operator + ();

getch();

}

**Exam no 3**

#include<iostream.h>

#include<conio.h>

class OpData3

{

private: int a,b;

public: OpData3()

{

a=10;

b=20;

cout<<"\n A=>"<<a;

cout<<"\n B=>"<<b;

}

void operator - ()

{

int x,y;

x=-a;

y=-b;

cout<<"\n X=>"<<x;

cout<<"\n Y=>"<<y;

}

void operator +()

{

cout<<"\n bit ballarpur";

}

void operator \*()

{

int p=25,q=35;

cout<<"\n addition is =>"<<(p+q);

}

};

void main()

{

clrscr();

OpData3 d;

d.operator -();

// -d;

// d.operator \*();

\*d;

d.operator + ();

getch();

}

**Exam no 4**

#include<iostream.h>

#include<conio.h>

/\* Binary operator overloading program \*/

class OpData4

{

private: int a,b;

public: OpData4()

{

a=10;

b=20;

cout<<"\n A=>"<<a;

cout<<"\n B=>"<<b;

}

void operator+(OpData4 &d3)

{

OpData4 d4;

//d4.a = d1.a + d2.a;

//d4.a = d1.b + d2.b;

d4.a = a + d3.a;

d4.b = b + d3.b;

cout<<"\n A=>"<<d4.a;

cout<<"\n B=>"<<d4.b;

}

};

void main()

{

clrscr();

OpData4 d1,d2;

d1.operator +(d2); //OR d1+d2;

getch();

}

**Exam no 5**

#include<iostream.h>

#include<conio.h>

/\* Binary operator overloading program \*/

class OpData5

{

private: int real, img;

public: setdata( )

{

cout <<”\n Enter complex number real and imaginary”;

cin>>real>>img;

cout<<"\n the given complex number is =>”<<real <<"+J”<<img;

}

void operator+(OpData4 &d3)

{

OpData4 d4;

d4.real = real + d3.real;

d4.img = img + d3. img;

cout<<"\n The addition of given complex number is =>”<<d4.real <<"+J”<<d4.img;

}

};

void main( )

{

clrscr();

OpData4 d1,d2;

d1.setdata( );

d2.setdata( );

d1.operator +(d2); //OR d1+d2;

getch();

}