Object-Oriented Programming Using C++

Overloading

Indranil Saha

Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

Overloading

- Specify more than one definition for a function name or an operator in the same scope
- Both declarations have different arguments and obviously different definition (implementation)
- When you call an overloaded function or operator, the compiler determines the most appropriate definition to use, by comparing the argument types you have used to call the function or operator with the parameter types specified in the definitions
- The process of selecting the most appropriate overloaded function or operator is called overload resolution

Example: Function Overloading

Print Function Overloading

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

class printData {
   public:
      void print(int i) {
        cout << "Printing int: " << i << endl;
      }
      void print(double  f) {
        cout << "Printing float: " << f << endl;
      }
      void print(char* c) {
        cout << "Printing character: " << c << endl;
      }
};

int main(void) {
   printData pd;

   // Call print to print integer
   pd.print(5);

   // Call print to print float
   pd.print(500.263);

   // Call print to print character
   pd.print("Hello C++");

   return 0;
}
```

Example: Function Overloading

Output

```
Printing int: 5
Printing float: 500.263
Printing character: Hello C++
```

Operator Overloading

- You can redefine or overload most of the built-in operators available in C++.
- A programmer can use operators with user-defined types as well.
- Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined.
- Like any other function, an overloaded operator has a return type and a parameter list.

Operator Overloading as Member and Non-Member Functions

Overloaded Operator as Member Function

```
Box operator+(const Box&);
```

Overloaded Operator as Non-Member Function

```
Box operator+(const Box&, const Box&);
```

Example: Operator Overloading

Operator Overloading for the Box class

```
class Box {
   public:
      double getVolume(void) {
         return length * breadth * height;
      }
      void setLength( double len ) {
         length = len;
      }
      void setBreadth( double bre ) {
         breadth = bre;
      }
      void setHeight( double hei ) {
         height = hei;
      }

      // Overload + operator to add two Box objects.
      Box operator+(const Box& b) {
         Box box;
         box.length = this->length + b.length;
         box.breadth = this->breadth + b.breadth;
         box.height = this->height + b.height;
         return box;
      }

   private:
      double length;      // Length of a box
      double breadth;     // Breadth of a box
      double height;      // Height of a box
};
```

Example: Operator Overloading

Operator Overloading for the Box class

```
// Main function for the program
int main() {
   Box Box1;                // Declare Box1 of type Box
   Box Box2;                // Declare Box2 of type Box
   Box Box3;                // Declare Box3 of type Box
   double volume = 0.0;     // Store the volume of a box here

   // box 1 specification
   Box1.setLength(6.0);
   Box1.setBreadth(7.0);
   Box1.setHeight(5.0);
   // box 2 specification
   Box2.setLength(12.0);
   Box2.setBreadth(13.0);
   Box2.setHeight(10.0);

   // volume of box 1
   volume = Box1.getVolume();
   cout << "Volume of Box1 : " << volume <<endl;
   // volume of box 2
   volume = Box2.getVolume();
   cout << "Volume of Box2 : " << volume <<endl;
   // Add two object as follows:
   Box3 = Box1 + Box2;
   // volume of box 3
   volume = Box3.getVolume();
   cout << "Volume of Box3 : " << volume <<endl;

   return 0;
}
```

Example: Operator Overloading

Output

```
Volume of Box1 : 210
Volume of Box2 : 1560
Volume of Box3 : 5400
```

Operators That Can Be Overloaded

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

Operators That Can Not Be Overloaded

::	.*	.	?:

Object-Oriented Programming Using C++

Overloading

Indranil Saha

Department of Computer Science and Engineering
Indian Institute of Technology Kanpur