



Module 09

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outline

Operators &
Functions

Operator
Overloading

Advantages and
Disadvantages

Examples

String

Enum

Operator
Overloading
Rules

Summary

Module 09: Programming in C++

Operator Overloading

Instructors: Abir Das and Sourangshu Bhattacharya

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

{abir, sourangshu}@cse.iitkgp.ac.in

Slides taken from NPTEL course on Programming in Modern C++

by **Prof. Partha Pratim Das**



Module Objectives

Module 09

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outline

Operators &
Functions

Operator
Overloading

Advantages and
Disadvantages

Examples

String

Enum

Operator
Overloading
Rules

Summary

- Understand the Operator Overloading



Module Outline

Module 09

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outline

Operators &
Functions

Operator
Overloading

Advantages and
Disadvantages

Examples

String

Enum

Operator
Overloading
Rules

Summary

- Basic Differences between Operators & Functions
- Operator Overloading
- Examples of Operator Overloading
 - `operator+` for String & Enum
- Operator Overloading Rules



Operator & Function

Module 09

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outline

Operators &
Functions

Operator
Overloading

Advantages and
Disadvantages

Examples

String

Enum

Operator
Overloading
Rules

Summary

- What is the difference between an *operator* & a *function*?

```
unsigned int Multiply(unsigned x, unsigned y) {  
    int prod = 0;  
    while (y-- > 0) prod += x;  
    return prod;  
}
```

```
int main() {  
    unsigned int a = 2, b = 3;  
  
    // Computed by '*' operator  
    unsigned int c = a * b;           // c is 6  
  
    // Computed by Multiply function  
    unsigned int d = Multiply(a, b); // d is 6  
  
    return 0;  
}
```

- Same computation by an operator and a function



Difference between Operator & Functions

Module 09

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outline

Operators &
Functions

Operator
Overloading

Advantages and
Disadvantages

Examples

String

Enum

Operator
Overloading
Rules

Summary

Operator

- Usually written in **infix** notation - at times in **prefix** or **postfix**

- Examples:

```
// Operator in-between operands
```

```
Infix: a + b; a ? b : c;
```

```
// Operator before operands
```

```
Prefix: ++a;
```

```
// Operator after operands
```

```
Postfix: a++;
```

- Operates on one or more operands, typically up to 3 (Unary, Binary or Ternary)
- Produces **one result**
- Order of operations is decided by **precedence** and **associativity**
- Operators are pre-defined

Function

- Always written in **prefix** notation

- Examples:

```
// Operator before operands
```

```
Prefix: max(a, b);
```

```
qsort(int[], int, int,  
void (*)(void*, void*));
```

- Operates on zero or more arguments
- Produces **up to one result**
- Order of application is decided by **depth of nesting**
- Functions can be defined as needed



Operator Functions in C++

Module 09

Instructors: Abir Das and Sourangshu Bhattacharya

Objectives & Outline

Operators & Functions

Operator Overloading

Advantages and Disadvantages

Examples

String

Enum

Operator Overloading Rules

Summary

- C++ introduces a new keyword: `operator`
- Every operator is associated with an operator function that defines its behavior

Operator Expression	Operator Function
<code>a + b</code>	<code>operator+(a, b)</code>
<code>a = b</code>	<code>operator=(a, b)</code>
<code>c = a + b</code>	<code>operator=(c, operator+(a, b))</code>

- Operator functions are *implicit for predefined operators of built-in types* and *cannot be redefined*
- An operator function may have a signature as:

```
MyType a, b; // An enum or struct
```



```
MyType operator+(MyType, MyType); // Operator function
```



```
a + b // Calls operator+(a, b)
```
- C++ allows users to define an operator function and overload it



Operator Overloading

Module 09

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outline

Operators &
Functions

Operator
Overloading

Advantages and
Disadvantages

Examples

String

Enum

Operator
Overloading
Rules

Summary

- **Operator Overloading** (also called *ad hoc polymorphism*), is a specific case of *polymorphism*, where different operators have different implementations depending on their arguments
- Operator overloading is generally defined by a programming language, For example, in C (and in C++), for `operator/`, we have:

Integer Division	Floating Point Division
<pre>int i = 5, j = 2; int k = i / j; // k = 2</pre>	<pre>double i = 5, j = 2; double k = i / j; // k = 2.5</pre>

- C does not allow programmers to overload its operators
- C++ allows programmers to overload its operators by using operator functions



Operator Overloading: Advantages and Disadvantages

Module 09

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outline

Operators &
Functions

Operator
Overloading

Advantages and
Disadvantages

Examples

String

Enum

Operator
Overloading
Rules

Summary

- **Advantages:**

- Operator overloading is *syntactic sugar*, and is used because it allows programming using notation nearer to the target domain
- It also allows user-defined types a similar level of syntactic support as types built into a language
- It is common in scientific computing, where it allows computing representations of mathematical objects to be manipulated with the same syntax as on paper
- For example, if we build a `Complex` type in C and `a`, `b` and `c` are variables of `Complex` type, we need to code an expression

$$a + b * c$$

using functions to add and multiply Complex value as

$$\text{Add}(a, \text{Multiply}(b, c))$$

which is clumsy and non-intuitive

- Using operator overloading we can write the expression with operators without having to use the functions



Operator Overloading: Advantages and Disadvantages

Module 09

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outline

Operators &
Functions

Operator
Overloading

Advantages and
Disadvantages

Examples

String

Enum

Operator
Overloading
Rules

Summary

• Disadvantages

- Operator overloading allows programmers to *reassign the semantics of operators* depending on the types of their operands. For example, for `int a, b`, an expression `a << b` shifts the bits in the variable `a` left by `b`, whereas `cout << a << b` outputs values of `a` and `b` to standard output (`cout`)
- As operator overloading allows the programmer to change the usual semantics of an operator, it is a good practice to use operator overloading with care to maintain the *Semantic Congruity*
- With operator overloading certain rules from mathematics can be *wrongly expected* or *unintentionally assumed*. For example, the commutativity of `operator+` (that is, `a + b == b + a`) is not preserved when we overload it to mean *string concatenation* as

`"run" + "time" = "runtime" ≠ "timerun" = "time" + "run"`

Of course, mathematics too has such deviations as multiplication is commutative for real and complex numbers but not commutative in matrix multiplication



Program 09.01: String Concatenation

Module 09

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outline

Operators &
Functions

Operator
Overloading

Advantages and
Disadvantages

Examples

String

Enum

Operator
Overloading
Rules

Summary

Concatenation by string functions

```
#include <iostream>
#include <cstring>
using namespace std;
typedef struct _String { char *str; } String;
int main() { String fName, lName, name;
    fName.str = strdup("Partha ");
    lName.str = strdup("Das" );
    name.str = (char *) malloc( // Allocation
        strlen(fName.str) +
        strlen(lName.str) + 1);
    strcpy(name.str, fName.str);
    strcat(name.str, lName.str);
    cout << "First Name: " <<
        fName.str << endl;
    cout << "Last Name: " <<
        lName.str << endl;
    cout << "Full Name: " <<
        name.str << endl;
}
```

First Name: Partha
Last Name: Das
Full Name: Partha Das

Concatenation operator

```
#include <iostream>
#include <cstring>
using namespace std;
typedef struct _String { char *str; } String;
String operator+(const String& s1, const String& s2) {
    String s;
    s.str = (char *) malloc(strlen(s1.str) +
        strlen(s2.str) + 1); // Allocation
    strcpy(s.str, s1.str); strcat(s.str, s2.str);
    return s;
}
int main() { String fName, lName, name;
    fName.str = strdup("Partha ");
    lName.str = strdup("Das");
    name = fName + lName; // Overloaded operator +
    cout << "First Name: " << fName.str << endl;
    cout << "Last Name: " << lName.str << endl;
    cout << "Full Name: " << name.str << endl;
}
```

First Name: Partha
Last Name: Das
Full Name: Partha Das



Program 09.02: A new semantics for operator+

Module 09

Instructors: Abir Das and Sourangshu Bhattacharya

Objectives & Outline

Operators & Functions

Operator Overloading

Advantages and Disadvantages

Examples

String

Enum

Operator Overloading Rules

Summary

w/o Overloading +

```
#include <iostream>
using namespace std;
enum E { C0 = 0, C1 = 1, C2 = 2 };
```

```
int main() { E a = C1, b = C2;
            int x = -1;

            x = a + b; // operator + for int
            cout << x << endl;
        }
-----
3
```

- Implicitly converts `enum E` values to `int`
- Adds by `operator+` of `int`
- Result is outside `enum E` range

Overloading operator +

```
#include <iostream>
using namespace std;
enum E { C0 = 0, C1 = 1, C2 = 2 };
```

```
E operator+(const E& a, const E& b) { // Overloaded operator +
    unsigned int uia = a, uib = b;
    unsigned int t = (uia + uib) % 3; // Redefined addition
    return (E) t;
}

int main() { E a = C1, b = C2;
            int x = -1;

            x = a + b; // Overloaded operator + for enum E
            cout << x << endl;
        }
-----
0
```

- `operator +` is overloaded for `enum E`
- Result is a valid `enum E` value



Operator Overloading – Summary of Rules

Module 09

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outline

Operators &
Functions

Operator
Overloading

Advantages and
Disadvantages

Examples

String

Enum

Operator
Overloading
Rules

Summary

- **No new operator** such as `operators**` or `operators<>` can be defined for overloading
- **Intrinsic properties** of the overloaded operator *cannot be changed*
 - Preserves *arity*
 - Preserves *precedence*
 - Preserves *associativity*
- These operators can be overloaded:
`[] + - * / % ^ & | ~ ! = += -= *= /= %= ^= &= |=`
`<< >> >>= <<= == != < > <= >= && || ++ -- , ->* -> () []`
- For **unary prefix operators**, use: `MyType& operator++(MyType& s1)`
- For **unary postfix operators**, use: `MyType operator++(MyType& s1, int)`
- The `operators::` (*scope resolution*), `operator.` (*member access*), `operator.*` (*member access through pointer to member*), `operator sizeof`, and `operator?:` (*ternary conditional*) *cannot be overloaded*
- The overloads of `operators&&`, `operator||`, and `operator,` (*comma*) *lose their special properties: short-circuit evaluation and sequencing*



Overloading disallowed for

operator	Reason
dot (.)	It will raise question whether it is for <i>object reference</i> or <i>overloading</i>
Scope Resolution (::)	It performs a (compile time) <i>scope resolution</i> rather than an <i>expression evaluation</i>
Ternary (?:)	Overloading <code>expr1? expr2: expr3</code> would not guarantee that <i>only one of expr2 and expr3</i> was executed
sizeof	Operator <code>sizeof</code> cannot be overloaded because <i>built-in operations</i> , such as incrementing a pointer into an array <i>implicitly depends on it</i>
&& and	In evaluation, the <i>second operand is not evaluated</i> if the result can be deduced <i>solely by evaluating the first operand</i> . However, this evaluation is not possible for overloaded versions of these operators
Comma (,)	This operator guarantees that the <i>first operand</i> is evaluated <i>before</i> the <i>second operand</i> . However, if the comma operator is overloaded, its operand evaluation depends on C++'s function parameter mechanism, which does not guarantee the order of evaluation
Ampersand (&)	The address of an object of incomplete type can be taken, but if the complete type of that object is a class type that declares <code>operator&()</code> as a member function, then the behavior is undefined



Module Summary

Module 09

Instructors: Abir
Das and
Sourangshu
Bhattacharya

- Introduced operator overloading
- Explained the rules of operator overloading

Objectives &
Outline

Operators &
Functions

Operator
Overloading

Advantages and
Disadvantages

Examples

String

Enum

Operator
Overloading
Rules

Summary