Object-oriented programming CS10003

Lecturer: Do Nguyen Kha

Contents

- C++ Operators
- C++ Operator Overloading
- Static Member
- Const Method
- Singleton Pattern
- C++ Standard Template Library (STL)
- Team Project: Milestone 1

C++ Operators

Operators are symbols that perform operations on variables and values. For example, + is an operator used for addition, while - is an operator used for subtraction.

Operators in C++ can be classified into 6 types:

1. Arithmetic Operators
2. Assignment Operators
3. Relational Operators
4. Logical Operators
5. Bitwise Operators
6. Other Operators

Arithmetic Operators

+	Addition
−	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

Assignment Operators

```
=   a = b;      a = b;

+=  a += b;     a = a + b;

-=  a -= b;     a = a - b;

*=  a *= b;     a = a * b;

/=  a /= b;     a = a / b;

%=  a %= b;     a = a % b;
```

Relational Operators

Operator	Meaning	Example
==	Is Equal To	
!=	Not Equal To	
>	Greater Than	
<	Less Than	
>=	Greater Than or Equal To	
<=	Less Than or Equal To	

Logical Operators

- `&&` Logical AND
- `||` Logical OR.
- `!` Logical NOT.

Bitwise Operators

`&`	Binary AND
`|`	Binary OR
`^`	Binary XOR
`~`	Binary One's Complement
`<<`	Binary Shift Left
`>>`	Binary Shift Right

Other Operators

`sizeof` returns the size of data type

`&` represents memory address of the operand

`.` accesses members of struct variables or class objects

`->` used with pointers to access the class or struct variables

`<<` prints the output value

`>>` gets the input value

`new, delete, [], ()...`

C++ Operator Overloading

Customizes the C++ operators for operands of user-defined types

- The operators `::` (scope resolution), `.` (member access), `.*` (member access through pointer to member), and `?:` (ternary conditional) cannot be overloaded
- New operators such as `**`, `<>`, or `&|` cannot be created
- It is not possible to change the precedence, grouping, or number of operands of operators

… reference: https://en.cppreference.com/w/cpp/language/operators

Assignment operator

The canonical copy-assignment operator is expected to be safe on self-assignment, and to return the lhs by reference:

```
if (this == &other)

    return *this;
```

Increment Operator

```
struct X {
    // prefix increment
    X& operator++() {
        // actual increment takes place here
        return *this; // return new value by reference
    }

    // postfix increment
    X operator++(int) {
        X old = *this; // copy old value
        operator++();  // prefix increment
        return old;    // return old value
    }
};
```

Function Call Operator

```
// An object of this type represents a linear function of one variable a * x + b.
struct Linear {
    double a, b;
    double operator()(double x) const {
        return a * x + b;
    }
};
int main() {
    Linear f{2, 1};  // Represents function 2x + 1.
    Linear g{-1, 0}; // Represents function -x.
    double f_0 = f(0); // f and g are objects that can be used like a function.
    double f_1 = f(1);
    double g_0 = g(0);
}
```

Static Member

Define class members static using `static` keyword. It means no matter how many objects of the class are created, there is only one copy of the static member.

Static Member

```
class Test {
  static int count;
  public:
    static int Show();
    Test(){count++;}
    ~Test(){count--;};
};

int Test::Show(){
  return count;
}

int Test::count = 0; // Initialize static member

void main(){
  Test a, b;
  cout << Test::Show();
}
```

Const Method

Constant member functions are those functions that are denied to change the values of the data members. To make a function constant, the keyword `const` is appended to the function prototype and also to the function definition header.

```
return_type functionName() const;
```

Const Method

A const object can be created by prefixing the `const` keyword to the object declaration. Any attempt to change the data member of const objects results in a compile-time error

Singleton Pattern

Singleton design pattern is a software design principle that is used to restrict the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system.

Singleton Pattern

- Using `private`/`protected` modifier to prevent the call of construct outside of the class
- Define a `getInstance` static public method to access the single instance of the class

Singleton Pattern

```
class Counter {
private:
   static Singleton *instance;
   int data;
   Singleton() { count = 0; }
public:
   static Singleton *getInstance() {
      if (!instance)
         instance = new Singleton();
      return instance;
   }
   int getCount() {
      return this->count;
   }
   void increase() {
      this->count++;
   }
};
Singleton *Singleton::instance = NULL; //Initialize pointer to NULL
```

Singleton Pattern

```
int main() {
    Counter *c = Counter::getInstance();
    cout << c->getCount() << endl;
    c->increase();
    cout << c->getCount() << endl;
    return 0;
}
```

C++ Standard Template Library (STL)

The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators. It is a generalized library and so, its components are parameterized. Working knowledge of template classes is a prerequisite for working with STL.

Team Project: Milestone 1

- Using GDI plus: https://learn.microsoft.com/en-us/windows/win32/gdiplus/-gdiplus-gdi-start
- Using RapidXLM: https://rapidxml.sourceforge.net/manual.html