

Mathematical Software Programming (02635)

Lecture 12 — November 27, 2025

Instructor: Martin S. Andersen

Fall 2025



This week

Topics

- ▶ Introduction to object-oriented programming and C++

Learning objectives

- ▶ Describe and use basic object-oriented programming concepts such as classes and objects

What is C++?

- ▶ General-purpose programming language that is derived from C
- ▶ Development started by Bjarne Stroustrup in the late 1970s
- ▶ Renamed from *C with Classes* to *C++* in 1983
- ▶ Adds *object-oriented* abstractions
- ▶ Adds *namespaces* and scope-resolution operator ::
- ▶ Allows generic programming via *templates*
- ▶ Error handling via *exceptions*
- ▶ Most (but not all) C code is valid C++ code
- ▶ Some C++ innovations have been integrated in C
- ▶ Use of macros is discouraged in C++
- ▶ The C++ language is more complicated than the C language

Hello World (v1)

```
// hello.cpp
#include <iostream>

int main(int argc, const char *argv[]) {
    std::cout << "Hello 02635!" << std::endl;

    return 0;
}
```

Compiling (g++, clang++, c++) and running the program:

```
$ g++ hello.cpp -Wall -std=c++11 -o Hello
$ ./Hello
```

Makefile

Use CXX and CXXFLAGS instead of CC and CFLAGS

```
CXX=g++                      # C++ compiler
CPPFLAGS=                      # preprocessor flags
CXXFLAGS=-Wall -std=c++11      # C++ compiler flags
LDFLAGS=                        # linker flags
LDLIBS=                         # library flags

hello: hello.cpp               # This is an implicit rule

.PHONY: clean
clean:
    -$(RM) hello
```

Hello World (v2)

```
// hello.cpp
#include <iostream>
using namespace std;

int main(int argc, const char *argv[]) {
    cout << "Hello 02635!" << endl;

    return 0;
}
```

Input/output

- ▶ Standard Input/Output Streams Library: #include <iostream>
- ▶ Standard input stream: std::cin
- ▶ Standard output stream: std::cout
- ▶ Standard output stream for errors: std::cerr

Example

```
int i;
std::cout << "Enter an integer: ";
std::cin >> i;
std::cout << "You entered " << i << "!\n";
```

What happens if the user enters a string?

Structures in C++

- ▶ A struct member can be a function (aka a *method*)
- ▶ A struct member can be *public* (default) or *private*

Example

```
struct point {  
public:  
    double x;  
    double y;  
    double distance(point& p) { // call-by-reference  
        return hypot(x-p.x,y-p.y);  
    }  
};
```

Example

```
#include <iostream>
#include <cmath>

struct point {
    double x; double y;
    double distance(point& p) {
        return hypot(x-p.x,y-p.y);
    }
};

int main(int argc, const char *argv[]) {
    point P1,P2;
    P1.x = 1; P1.y = 3; P2.x = 2; P2.y = 4;
    std::cout << P1.distance(P2) << std::endl;
    return 0;
}
```

So what are classes and objects?

- ▶ A *class* is an abstract data type
- ▶ A *class* is essentially a C++ *struct* with *privacy* by default
- ▶ An *object* is an *instance* of a class
- ▶ An *object* is also sometimes called a *class instance* or a *class object*

Example

```
#include <iostream>
class rectangle {
public:
    double x; double y;
    double area() { return x*y; }
};
int main(int argc, const char *argv[]) {
    rectangle R; R.x = 1.0; R.y = 2.0;
    std::cout << "Area: " << R.area() << std::endl;
};
```

Example: the string class

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

int main(int argc, const char *argv[]) {
    string s1, s2;    // declare string objects s1 and s2
    s1 = "Hello";
    cout << "Enter your name: ";
    cin >> s2;
    cout << s1 << " " << s2 << "!\n"
        << "Your name is " << s2.length()
        << " characters long.\n";
    return 0;
}
```

Documentation: <http://www.cplusplus.com/reference/string/string/>

Dynamic allocation in C++

Keywords: new, delete

```
double *x = new double[m];
/* do something with array */
delete[] x;

rectangle *Rp = new rectangle;
Rp->x = 1.0; Rp->y = 2.5;
double A = Rp->area();
delete Rp;

string *sp = new string("Hello!");
/* do something with string object */
delete sp;
```

Reference variables

- ▶ A *safer, less powerful*, alternative to pointers
- ▶ An *alias* for an existing variable
- ▶ Unlike a pointer, a reference cannot be NULL
- ▶ A reference must be initialized and cannot be changed

Example 1

```
#include<iostream>
void swap (int& a, int& b) {
    int c = a;
    a = b; b = c;
}
int main(void) {
    int j = 2, k = 3;
    swap(j, k);
    std::cout << j << " " << k;
    return 0;
}
```

Reference variables

Example 2

```
#include <iostream>

int& fun() {
    static int x = 10;
    return x;
}

int main(void) {
    std::cout << fun() << std::endl;
    fun() = 30;
    std::cout << fun() << std::endl;
    return 0;
}
```

What is the behavior of this program?

Constructors

- ▶ *Constructor* member function is called when initializing an object
- ▶ No return type (not even void)
- ▶ *Function overloading*: member functions with the same name

Example

```
class point {  
private:  
    double x;  double y;  
public:  
    point() { x = 0; y = 0; }  
    point(double new_x, double new_y) { x=new_x; y=new_y; }  
    double distance(point& p) {  
        return hypot(x-p.x,y-p.y);  
    }  
};
```

```
point P1 = point();  
point P2 = point(2,1);
```

The copy constructor and initialization lists

Initialize an object of some type with an object of same type

```
class point {  
private:  
    double x;  
    double y;  
public:  
    point(const point& pt) : x(pt.x), y(pt.y) {}  
    point() { x = 0; y = 0; }  
    point(double new_x, double new_y) { x=new_x; y=new_y; }  
    double distance(point& p) {  
        return hypot(x-p.x,y-p.y);  
    }  
};
```

```
point p1 = point(1.0,2.0); // C++11: point p1 {1.0,2.0};  
point p2 = point(p1);
```

Destructors

- ▶ A *destructor* member function is called when deleting an object
- ▶ No parameters and no return type (not even void)
- ▶ Only necessary if *default* destructor is not sufficient
- ▶ Typical use: release resources before deleting object

```
class MyClass {  
public:  
    MyClass(int size) : data(new double[size]) {};  
    MyClass(const MyClass& Obj) {  
        // ... copy constructor ...  
    }  
    ~MyClass() { delete [] data; }  
    void set(int i, double val) { data[i] = val; }  
    double get(int i) { return data[i]; }  
private:  
    double* data;  
};
```

Operator overloading

```
class vect {  
private:  
    int n;  
public:  
    double *x;  
    vect(int len) : n(len), x(new double[len]) {}  
    vect(const vect& v) { /* copy constructor */ };  
    ~vect() { delete[] x; }  
    void operator=(double val) { for (int i=0;i<n;i++) x[i] = val; }  
    void operator+=(vect& v1) { for (int i=0;i<n;i++) x[i] += v1.x[i]; }  
    void print() {  
        for (int i=0;i<n;i++)  
            std::cout << "x[" << i << "] = " << x[i] << "\n";  
    }  
};
```

Operator overloading

Example

```
#include <iostream>

int main(int argc, const char *argv[]) {
    vect v1 = vect(4);
    vect v2 = vect(4);

    v1 = 1.0;      // set all elements to 1.0
    v2 = 2.0;      // set all elements to 2.0
    v1 += v2;      // add v2 to v1
    v1.print();    // print v1

    return 0;
}
```

Today's exercises

1. C++ version of *Hello World!* (create a Makefile)
2. Input/output with std::cin and std::cout
3. Exercises 11-1, 11-2, and 11-3 in *Beginning C++*

```
class Integer {  
private:  
    int i;  
public:  
    Integer(); // Constructor  
    /* "setter", "getter", print, compare, add, subtract, multiply */  
};
```