1. & 2. Introducing, Making & Using Objects

Hu Sikang
skhu@163.com

School of Computer
Beijing Institute of Technology

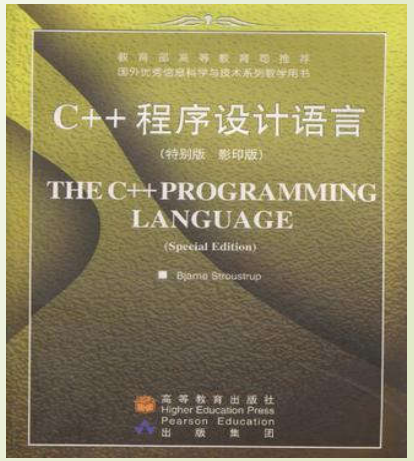
The history of C++

- C++:

  the father of C++ --- Bjarne Stroustrup, bell lab.

http://www.stroustrup.com/

- C++ is a better C, but it's not a pure OOP.
- C# is a better C++. It's pure OOP.

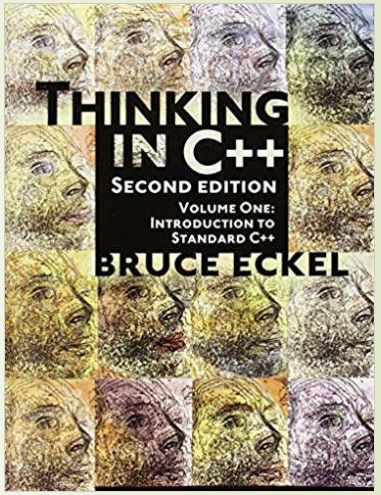


The C++ Programming Language(4th edition) -- Bjarne Stroustrup

The history of C++

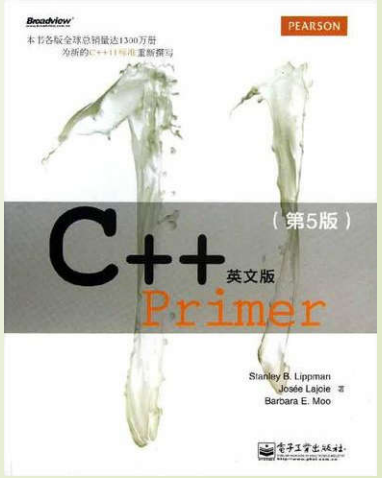
- Thinking in C++ (2rd edition)

  Bruce Eckel

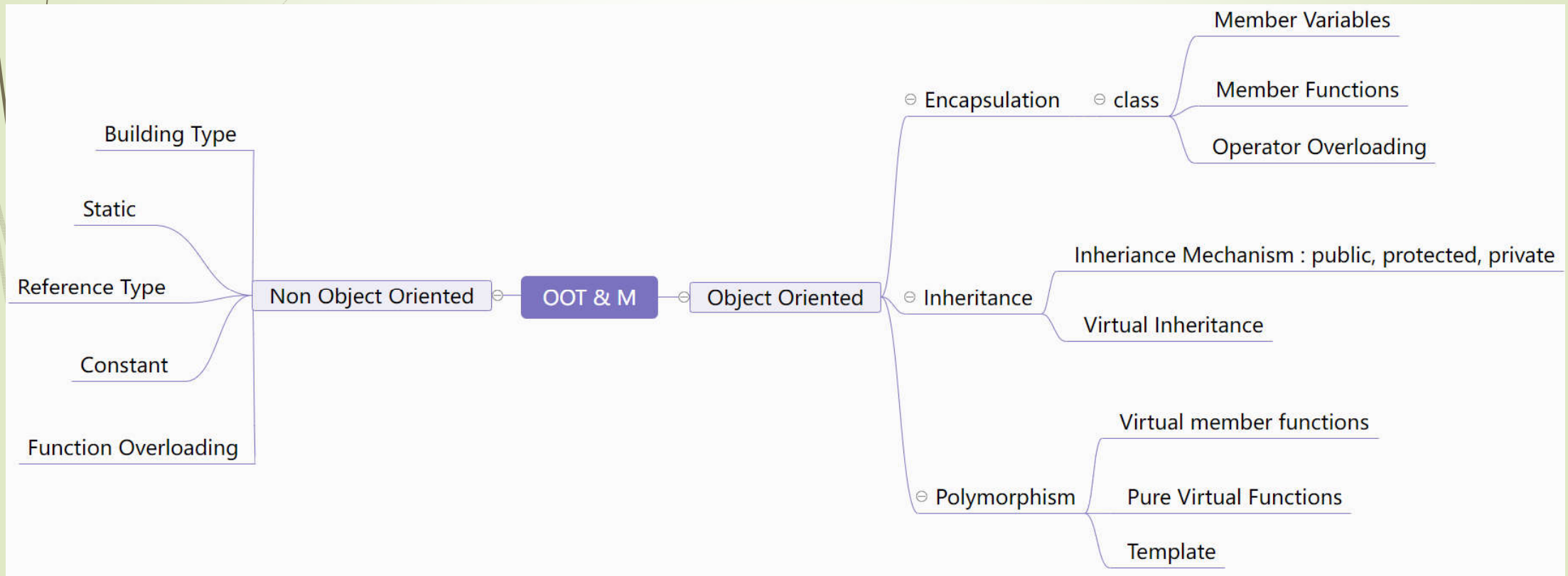


- C++ Primer(5th edition)

  Stanley B. Lippman, Josée Lajoie, Barbara E. Moo



Content

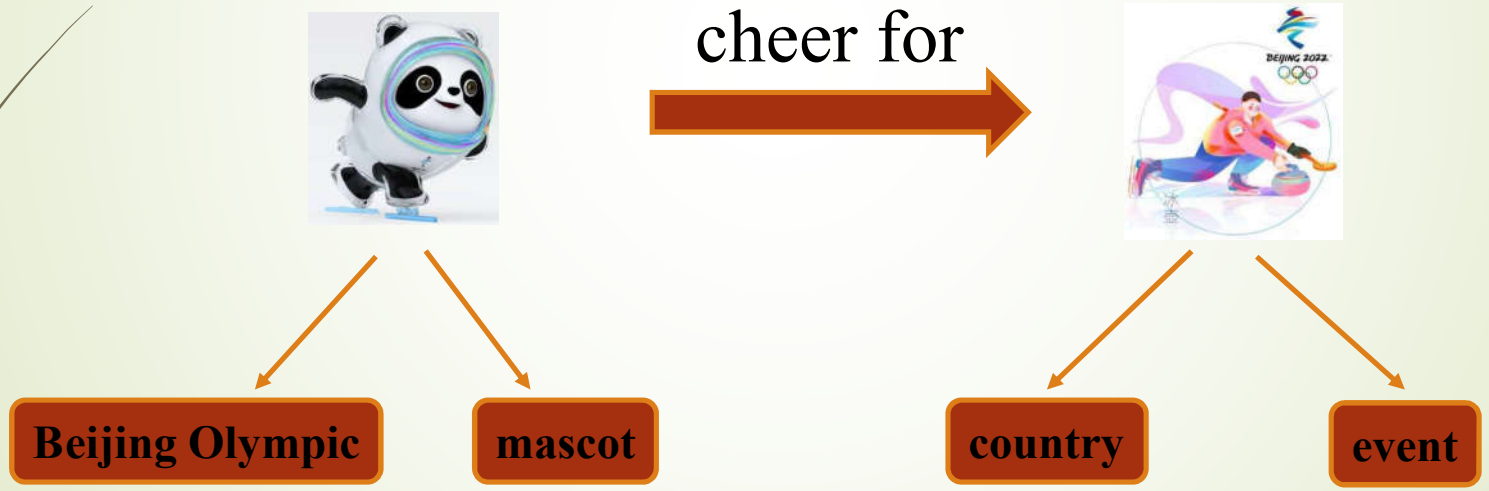

OOT & M
Non Object Oriented
Building Type
Static
Reference Type
Constant
Function Overloading
Object Oriented
Encapsulation
class
Member Variables
Member Functions
Operator Overloading
Inheritance
Inheriance Mechanism : public, protected, private
Virtual Inheritance
Polymorphism
Virtual member functions
Pure Virtual Functions
Template

Introducing, Making & Using Objects

- Class & Object
- The process of language translation
- Tools for separate compilation
- Your first C++ program
- More about iostreams
- Introducing strings
- Reading and writing files
- Introducing vector

1. Class & Object

Bing Dwen Dwen, the official mascots of the Beijing Olympic, will cheer for athletes from all over the world.



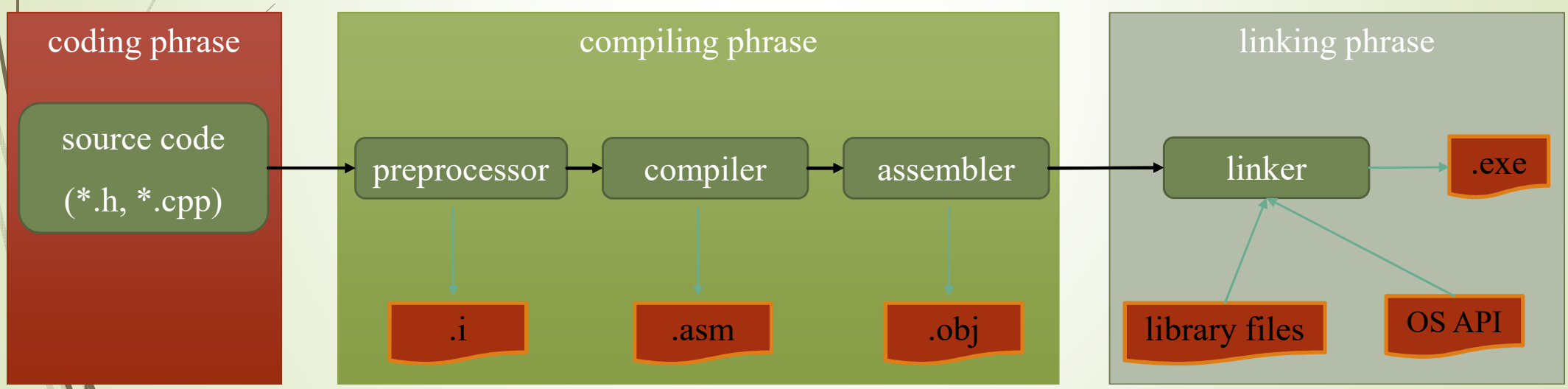
2 The process of language translation

All computer languages are translated from **source code** into **machine instructions**. Traditionally, translators fall into two classes: **interpreters** and **compilers**.

- Interpreters
- Compilers
- Compilation process
- Compilation, linkage, and run

2.1.1 Compilers

- **A compiler translates source code into assembly language or machine instructions (executable codes).**



2.1.2 Interpreters

- An interpreter translates source code into activities and immediately executes those activities, no machine instructions (executable codes).
- Traditional BASIC interpreters translate and execute one line at a time, and then forget that the line has been translated.
- This makes them slow in executing
- Rapid in programming and debugging

2.2 Tools for separate compilation

- Separate compilation is particularly important when building large projects.
- In C++, a program can be created in small, manageable, independently tested pieces.
- A Function is a pieces of code that can be placed in a single file, enabling separate compilation, which may take arguments and a return value.
- A file can contain more than one function.

Including headers

- A header file is a file containing the external declarations for a library. When using the functions and variables in the library, the header file should be included.
  - #include <iostream.h>
  - #include <iostream>
  - using namespace std;
  - #include **"self-defined.h"**

2.3 Your first C++ program

- The program will use the **Standard C++ iostream classes**. These read from and write to console and "standard" input and output.
- In this simple program, a stream object will be used to print a message on the screen.

"Hello, world!"

```
/* Saying Hello with C++ */
#include <iostream>        // Stream declarations
using namespace std;

int  main()
{
   cout << "Hello, World! " << endl;

   int x = 10, y = 20;
   cout << "x = " << x << endl << "y = " << y << endl;
   return 0;
}
```

printf("**Hello, World!\n** ")

printf("**x = %d\ny = %d\n**", x, y)

Input

```
#include <iostream>
using namespace std;
int main()
{
        int x;
        double y;
        char z;
        cin >> x;
        cin >> x >> y >> z;   // Input without any
format
        return 0;
}
```

scanf("%d", &x)

scanf("%d%f%c", &x, &y, &z)

2.4 Introducing strings

- The Standard C++ **string** class is designed to take care of (and hide) all the low-level manipulations of character arrays.
- The C++ header file **<string>** should be included. The **string** class is in the namespace **std** so a **using** directive is necessary.

“string” class

```
// The basics of the Standard C++ string class
#include <string>
#include <iostream>
using namespace std;

int  main()  {
  string s1;                    // Empty strings
  string s2 = " World";         // Initialized
  s1 = "Hello";
  cout << s1 + s2 + "!" << endl;
  s1 += s2 + "!";               // Appending to a string
  cout << s1 << endl;
  return 0;
}
```

2.5 Reading and writing files

- To open files for reading and writing, you must include **<fstream>.**
- To open a file for reading, you create an **ifstream** object, which then behaves like **cin**.
- To open a file for writing, you create an **ofstream** object, which then behaves like **cout**.
- The function **getline( )** allows you to read one line into a **string** object. The first argument is the **ifstream** object you're reading from and the second argument is the **string** object.

2.5 Reading and writing files

```
#include <string>
#include <iostream>
#include <fstream>
using namespace std;
int main()  {
     ifstream in("file1.txt");        // in.open("file1.txt", ios::in)
     string line;
     if (in) {                        // OR: in.is_open()
       while (getline(in, line))
              cout << line << endl; // display a line
     }
    in.close();                      // So in can be used to handle other file
     return 0;
}
```

2.5 Reading and writing files

```
// Copy File
#include <string>
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc, char *argv[]) {
    if (CopyFile(argv[1], argv[2]))
       cout << "OK" << endl;
    else
       cout << "ERROR" << endl;
    return 0;
}
```

```
bool CopyFile(char* from, char* to) {
    ifstream fin(from);
    ofstream  fout(to);
    string line;

    if (fin) {
        while (getline(fin, line))
           fout << line << endl; // copy a line to fout
        fin.close();
        fout.close();
        return true;
    }
    return false; // Fail
}
```

2.6 Vector

- With reading lines from a file into individual **string** objects, you do not know how many **string**s you are going to need – you only know after you have read the entire file.
- **Container classes** can help us to solve the problem.
- **Vector** is the most basic of standard containers.
- The **vector** class is a *template*, which means that it can be efficiently applied to different types.

```
// Copy an entire file into a vector of string
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;
int main()
{
    vector<string> v;
    ifstream in("file1.cpp");
    string line;
    while(getline(in, line))
      v.push_back(line);                    // Add the line to the end
    for(int i = 0; i < v.size(); i++)       // Add line numbers:
      cout << i << ": " << v[i] << endl;
    return 0;
}
```

file1.cpp

```
a
bb
ccc
dddd
eeeee
```

```
0: a
1: bb
2: ccc
3: dddd
4: eeeee
```

```
// Creating a vector that holds integers
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> v;
    for(int i = 0; i < 10; i++)
          v.push_back(i);
    for( i = 0; i < v.size(); i++)
     v[i] = v[i] * 10;                    // Assignment
    for(i = 0; i < v.size(); i++)
     cout << v[i] << ", ";
    cout << endl;
    return 0;
} ///:~
```

```
0,10,20,30,40,50,60,70,80,90,
```

2.7 Stream conventions

- Stream naming conventions

	Input	Output	Header
Generic	istream	ostream	<iostream>
File	ifsteam	ofstream	<fstream>
C string(legacy)	istrstream	ostrstream	<strstream>
C string	istringsteam	ostringstream	<sstream>

2.7 Stream conventions

- Predefined steams

Object of ostream	Purpose
cin	standard input
cout	standard output
cerr	unbuffered error(debugging) output
clog	buffered information(log text) output

2.7 Stream conventions

- Manipulators

Manipulator	Effect	Type
dec, hex, oct	Set numeric conversion	I, O
endl	insert newline and flush	I, O
flush	flush stream	I, O
setw(int)	set field width	I, O
setfill(ch)	change fill charactre	I, O
setbase(int)	set number base	I, O
ws	skip whitespace	I, O
setprecision(int)	set floating point precision	I, O
setiosflags(long)	turn on specified flags	I, O
resetiosflags(long)	turn off specified flags	I, O

2.7 Stream conventions

- Stream flags control fomatting

Flag	Efflect
ios::skipws	skip leading white space
ios::left, ios::right	justification
ios::internal	pad between sign and value
ios::dec, ios::oct, ios::hex	format for numbers
ios::showbase	show base of number
ios::showpoint	always show decimal point
ios::uppercase	put base in uppercase
ios::showpos	display '+' on positive numbers
ios::scientific, ios::fixed	floating point format
ios::unitbuf	Flush on every write

Summary

- The *object-oriented programming(OOP)* can be easy, if someone else has gone to the work of defining the classes for you. You include a header file, create the objects, and send messages to them.
- Some basic types: **streams**, **string**, **vector**