

1. & 2. Introducing, Making & Using Objects

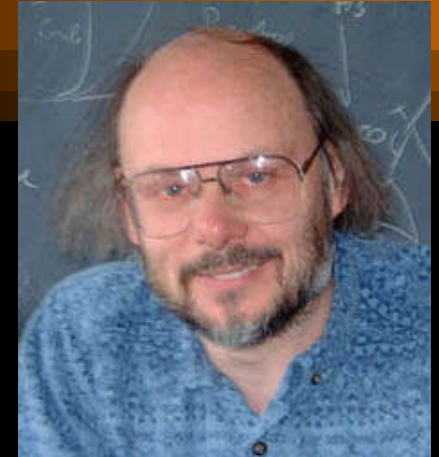
Hu Sikang
skhu@163.com

School of Computer
Beijing Institute of Technology

The history of C++

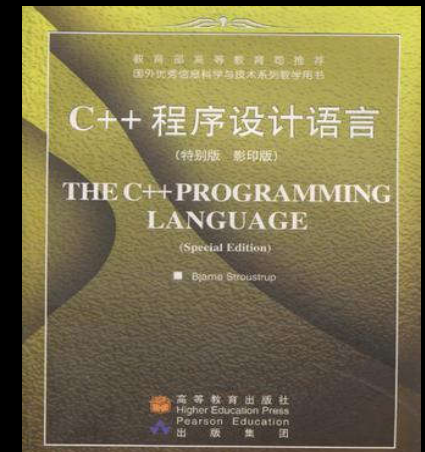
- C++:
the father of C++ --- Bjarne Stroustrup,
bell lab.

<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.

Programming -- Principles and Practice Using C++

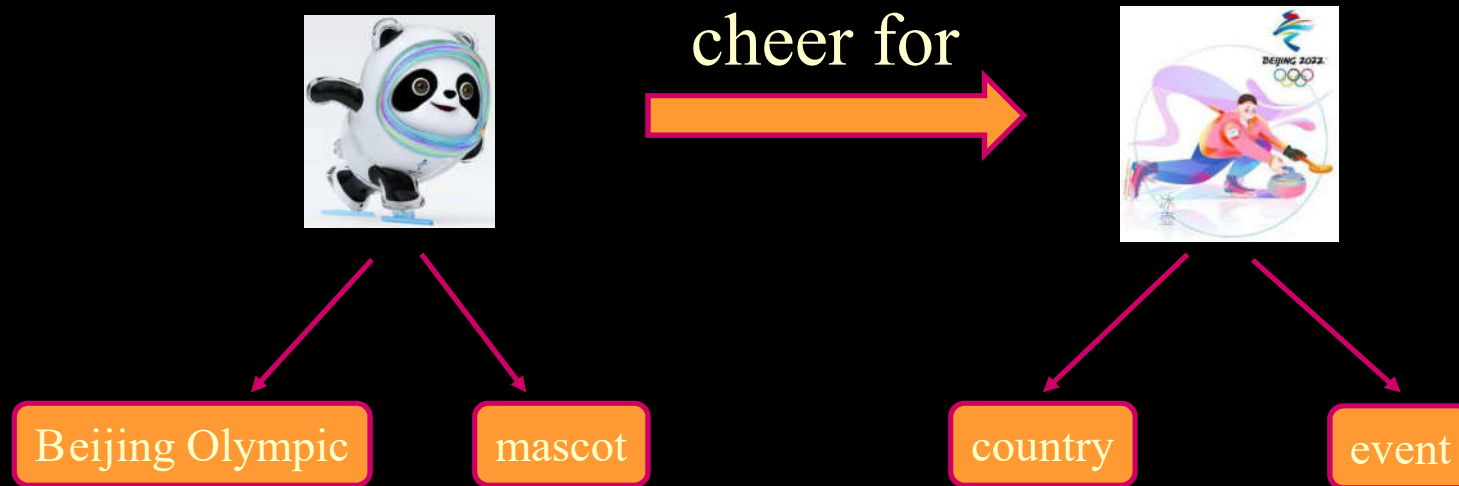


Introducing, Making & Using Objects

- **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.



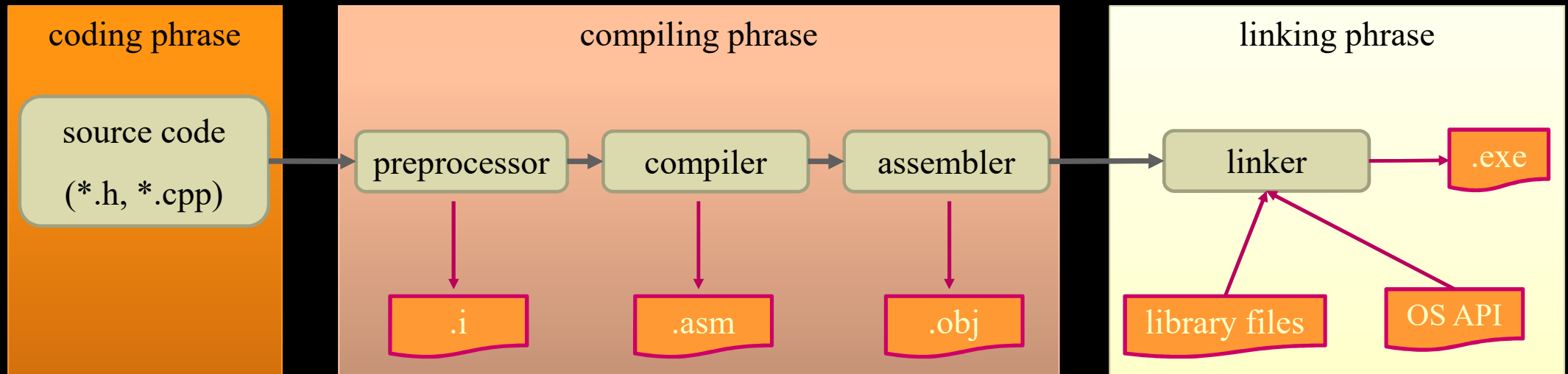
2 The process of language translation

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.
- To create a program with multiple files, functions in one file must access functions and data in other files. When compiling a file, the C++ compiler must know about the functions and data in the other files, in particular their names and proper usage.

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 "local.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;

void main()

{

cout << "Hello, World! " << endl;

int x = 10, y = 20;

cout << "x = " << x << endl << "y = " << y << endl;

}

printf("Hello, World!\n ")

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

Input

```
#include <iostream>
```

```
using namespace std;
```

```
void main()
```

```
{
```

```
    int x;
```

```
    double y;
```

```
    char z;
```

```
    cin >> x;
```

```
    cin >> x >> y >> z; // Input without any format
```

```
}
```

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;
```

```
void main()
```

```
{
```

```
    string s1;
```

// Empty strings

```
    string s2 = " World";
```

// Initialized

```
    s1 = "Hello";
```

```
    cout << s1 + s2 + "!" << endl;
```

```
    s1 += s2 + "!";
```

// Appending to a string

```
    cout << s1 << endl;
```

```
}
```

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;

void 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
}
```


2.5 Reading and writing files

// Copy File

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

void main(int argc, char *argv[]) {
    if (CopyFile(argv[1], argv[2]))
        cout << "OK" << endl;
    else
        cout << "ERROR" << endl;
}
```

```
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 **strings** 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;
void 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;
}
```

file1.cpp

a

bb

ccc

dddd

eeee

0: a

1: bb

2: ccc

3: dddd

4: eeee

// Creating a vector that holds integers

```
#include <iostream>
```

```
#include <vector>
```

```
using namespace std;
```

```
void 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;
```

```
} ///:~
```

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

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**