Chapter 2 Making & Using Objects

Outline

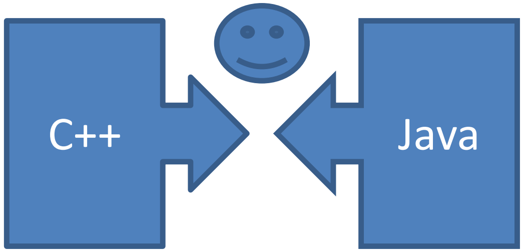
- Introduce enough C++ syntax and program construction concepts to allow you to write and run some simple object-oriented programs.
- C++ preprocessor instructions

The process of language translation

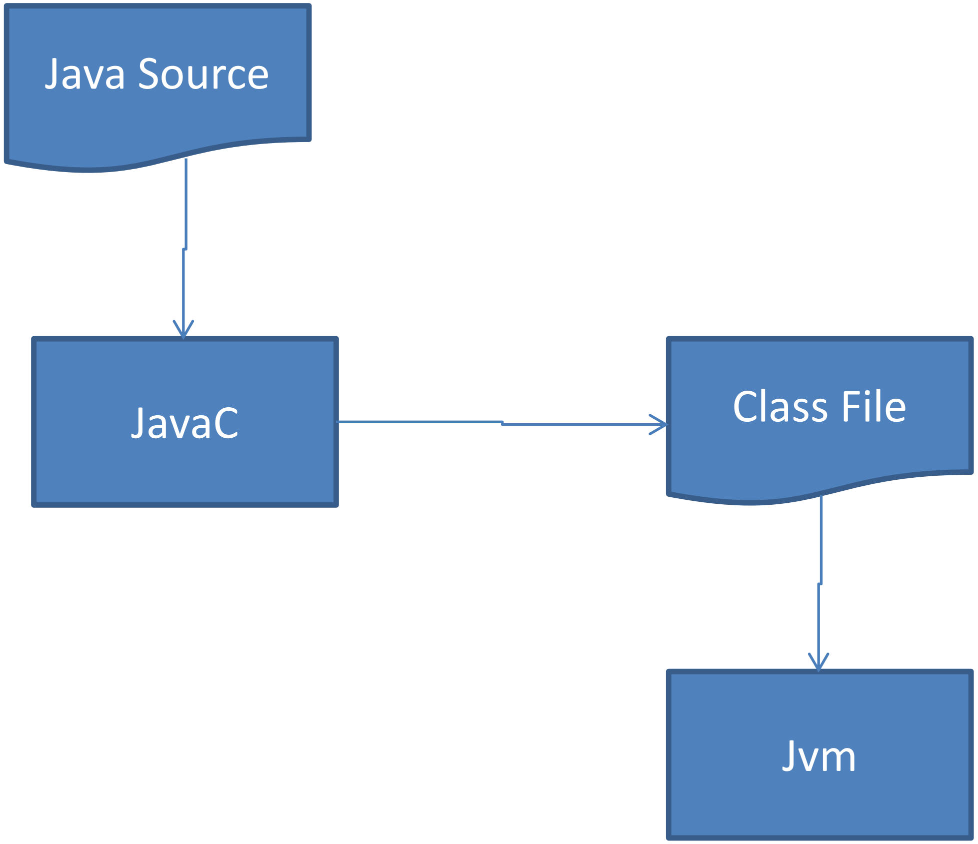
- Interpreters
  - An interpreter translates source code into activities and immediately executes those activities.
  - Interpreter is a virtual machine that can run high-level programming language.
  - Advantages：
    - Cross-platform
    - Ease of interaction and rapid development
  - interpreter may introduce unacceptable speed restrictions

The process of language translation

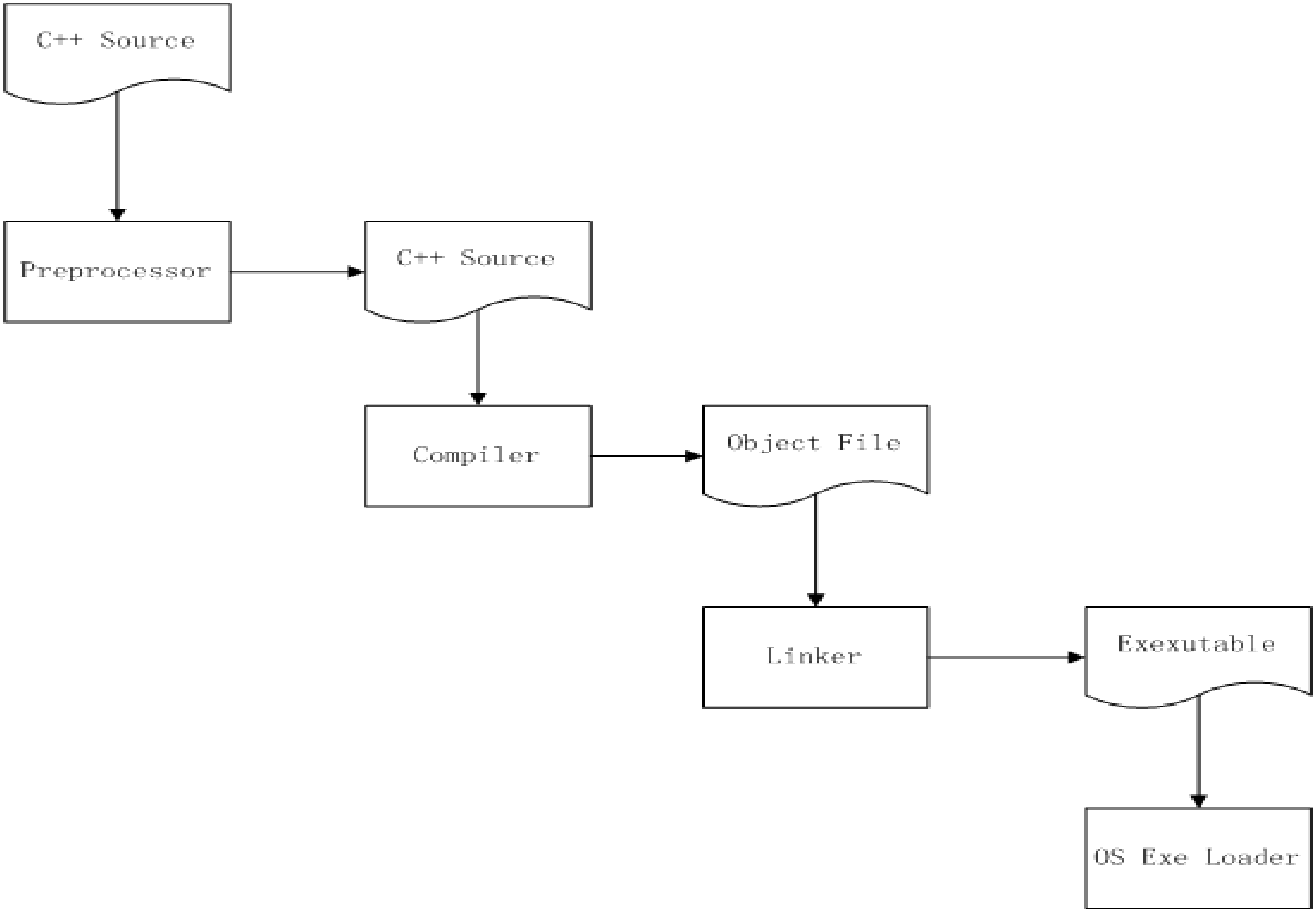
- Compilers
  - A compiler translates source code directly into assembly language or machine instructions.
  - Advantages
    - Performance
    - Separate compilation



Java Compilation Process



C++ compilation process



Hello.cpp

```
#include <iostream>
using namespace std;
int main(void) {
        cout << "hello c++ world“<<endl;
        return 0;
}
```

cout是一个ostream类的对象，有一个成员运算符函数operator<<，调用的时候就会向输出设备（屏幕）。

Hello.java

```
public class HelloJava {

        public static void main(String[] args) {
                System.out.println("hello java world");
        }

}
```

out是一个java.io.PrintStream类的对象，println()就是java.io.PrintStream类里的一个方法，它的作用就是用来向控制台输出信息的。

Demo

- Compile a java source and a c++ source

目标文件与连接

- 连接器把由编译产生的目标模块（一般是带”.o”或”.obj”扩展名的文件）连接成为操作系统可以加载和执行的程序。

Declarations vs. definitions

- declaration(声明)：
  - A declaration introduces a name – an identifier – to the compiler.
- definition(定义)：
  - A definition, on the other hand, says: “Make this variable here” or “Make this function here.”

Declarations vs. definitions

- Function declaration
  - int func1(int,int);
  - int func1(int length, int width);
- Function definition
  - int func1(int length, int width){……}

Declarations vs. definitions

- Variable declaration
  - extern int i;
- Variable definition
  - int i;

Static type checking

- The compiler performs type checking during the first pass
- Type checking tests for the proper use of arguments in functions and prevents many kinds of programming errors.
- dynamic type checking
  - Some object-oriented languages perform some type checking at runtime .

示例：Stream

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

int main() {
  // Specifying formats with manipulators:
  cout << "a number in decimal: "
         << dec << 15 << endl;
  cout << "in octal: " << oct << 15 << endl;
  cout << "in hex: " << hex << 15 << endl;
  cout << "a floating-point number: "
         << 3.14159 << endl;
  cout << "non-printing char (escape): "
         << char(27) << endl;
} ///:~
```

```
a number in decimal: 15
in octal: 17
in hex: f
a floating-point number: 3.14159
non-printing char (escape):
```

示例：字符串

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

int main() {
  string s1, s2;                    // Empty strings
  string s3 = "Hello, World.";     // Initialized
  string s4("I am");               // Also initialized
  s2 = "Today";                    // Assigning to a string
  s1 = s3 + " " + s4;              // Combining strings
  s1 += " 8 ";                     // Appending to a string
  cout << s1 + s2 + "!" << endl;
} ///:~
```

```
Hello, World. I am 8 Today!
```

示例：C文件复制

```
FILE *in_file, *out_file;
char data[BUF_SIZE];
size_t bytes_in, bytes_out;
in_file = fopen(argv[1], "rb");
out_file =  fopen(argv[2], "wb");
while ( (bytes_in = fread(data, 1, BUF_SIZE, in_file)) > 0 )
    {
        bytes_out = fwrite(data, 1, bytes_in, out_file);
        if ( bytes_in != bytes_out )
        {
            perror("Fatal write error.\n");
            return 1;
        }
    }
fclose(in_file);
fclose(out_file);
```

示例：C++文件复制

```
int main() {
   ifstream in("Scopy.cpp"); // Open for reading
   ofstream out("Scopy2.cpp"); // Open for writing
   string s;
   while(getline(in, s)) // Discards newline char
         out << s << "\n"; // ... must add it back
} ///:~
```

示例：Vector

```
int main() {
    vector<string> v;
    ifstream in("Fillvector.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++)
        cout << i << ":" << v[i] << endl;
} ///:~
```

容器

- 定义
  - 在数据存储上，有一种对象类型，它可以持有其它对象或指向其它对象的指针，这种对象类型就叫做容器。
- 特点：
  - 容器类是一种对特定代码重用问题的良好的解决方案
  - 容器可以自行扩展，自动增长容量能力

头文件

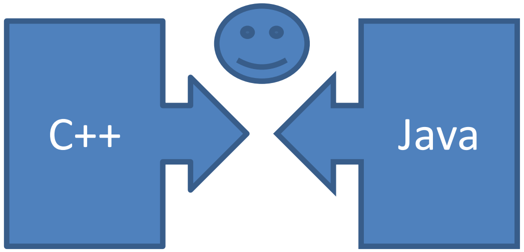
- 头文件用于声明目标文件或库文件中定义的变量或者函数。
- 包含头文件的语法：
  - #include <stdio.h>
  - #include “local.h”
  - #include <iostream>
- 三种包含头文件方式的区别

预处理

- 编译器编译C++程序时，它做的第一件事是进入预处理阶段。
- 预处理指令可以认为是独立于C++的一种程序设计语言，使用预处理指令编写的程序有时又被称为元程序设计(metaprogramming)

例：调试信息的输出

```
#ifdef DEBUG
cout << "debug message here";
#endif
```



```
logger.debug("debug message here");
```

常用的预处理指令

- #ifdef
- #ifndef
- #define
- #undef
- #include

预处理指令：#define

- 符号常量
  - #define PI 3.1415926 ;

  - 与const double PI = 3.1415926的比较

  X=sin(y*(PI/2))

预处理指令：#define

- 宏指令
  - 宏指令看上去类似于函数调用，但他们是通过文字替换而非真正的函数调用实现的。

```
#define max(A,B) ((A)>(B) ? (A) : (B))
```

```
max(i,j) ➔((i)>(j)?(i)?(j))
```

预处理指令：#define

- 控制编译

```
#ifdef DEBUG
#define log(A) cout << A
#endif

#ifndef DEBUG
#define log(A)
#endif

log("debug message here");
```

其它预处理指令

- #if　　　　　　　　　　　表达式非零就对代码进行编译
  - #if expression
    your code
    #endif
- #elif
- #else
- #end
- #error　　　　　　　　　输出一个错误信息
- #line
- #undef
- #pragma
  - #pragma message（“消息文本”）
  - #pragma once　　　　　　保证头文件被编译一次