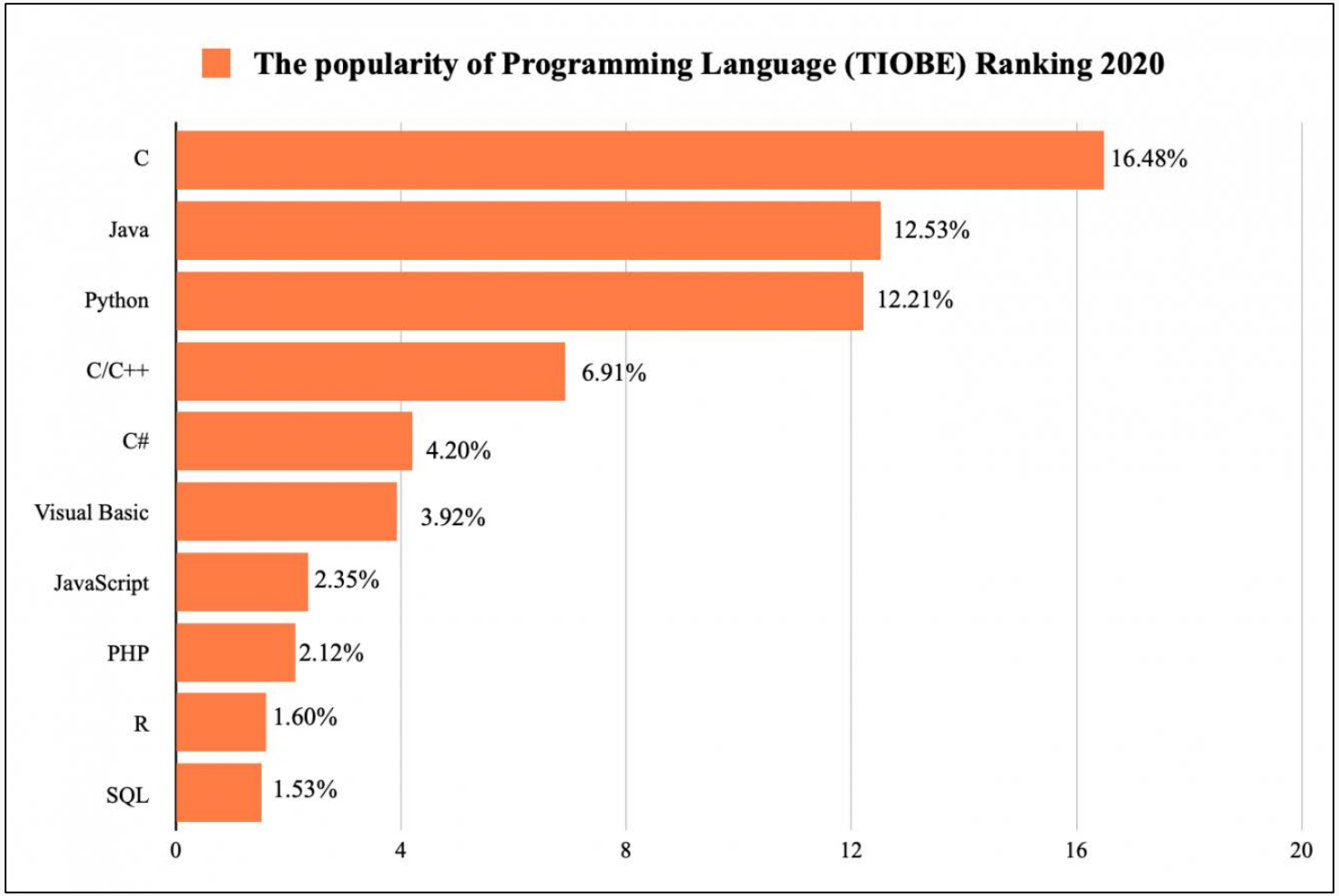
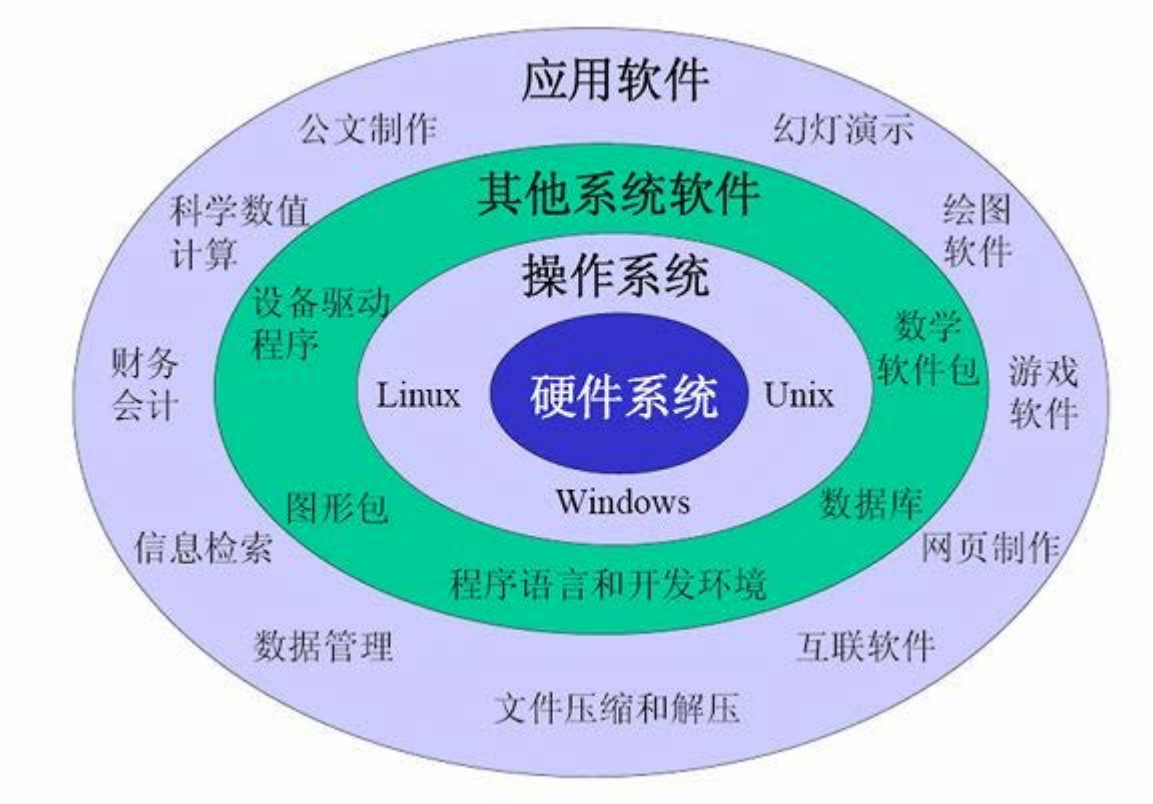
问题

- 我需要什么编程语言？



根据领域需求


应用软件
公文制作
幻灯演示
科学数值计算
绘图软件
其他系统软件
操作系统
设备驱动程序
数学软件包
财务会计
游戏软件
Linux
硬件系统
Unix
Windows
图形包
数据库
信息检索
网页制作
程序语言和开发环境
数据管理
互联软件
文件压缩和解压


Five of the best programming languages to learn to be successful in the artificial intelligence industry:
Python
R
Java
Scala
Rust


Web Programming Languages
www.educba.com

问题求解与实践

——C++回顾

主讲教师： 陈雨亭、沈艳艳

What is a program?

A first program – just the guts…

```
// …

int main()                          // main() is where a C++ program starts
{
   cout << "Hello, world!\n";       // output the 13 characters  Hello, world!
                                    // followed by a new line
   return 0;                        // return a value indicating success
}

// quotes delimit a string literal
// NOTE: “smart” quotes “  ” will cause compiler problems.
//                  so make sure your quotes are of the style  " "
// \n is a notation for a new line
```

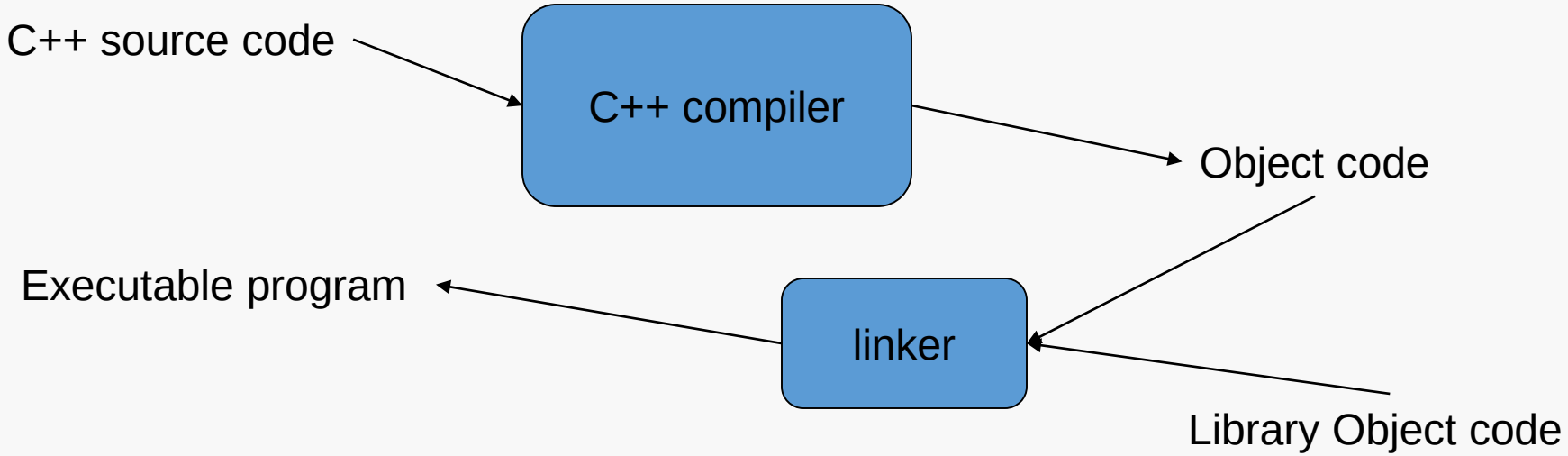
A first program – complete

```
// a first program:

#include "std_lib_facilities.h"        // get the library facilities needed for now

int main()                             // main() is where a C++ program starts
{
   cout << "Hello, world!\n";          // output the 13 characters  Hello, world!
                                       // followed by a new line
   return 0;                           // return a value indicating success
}

   // note the semicolons; they terminate statements
   // braces { … } group statements into a block
   // main( ) is a function that takes no arguments ( )
   //     and returns an int (integer value) to indicate success or failure
```

A second program

```
// modified for Windows console mode:

#include "std_lib_facilities.h"      // get the facilities for this course

int main()                           // main() is where a C++ program starts
{
   cout << "Hello, world!\n";        // output the 13 characters  Hello, world!
                                     // followed by a new line
   keep_window_open();               // wait for a keystroke
   return 0;                         // return a value indicating success
}

// without keep_window_open() the output window will be closed immediately
// before you have a chance to read the output (on Visual C++ 20xx)
```

Compilation and linking

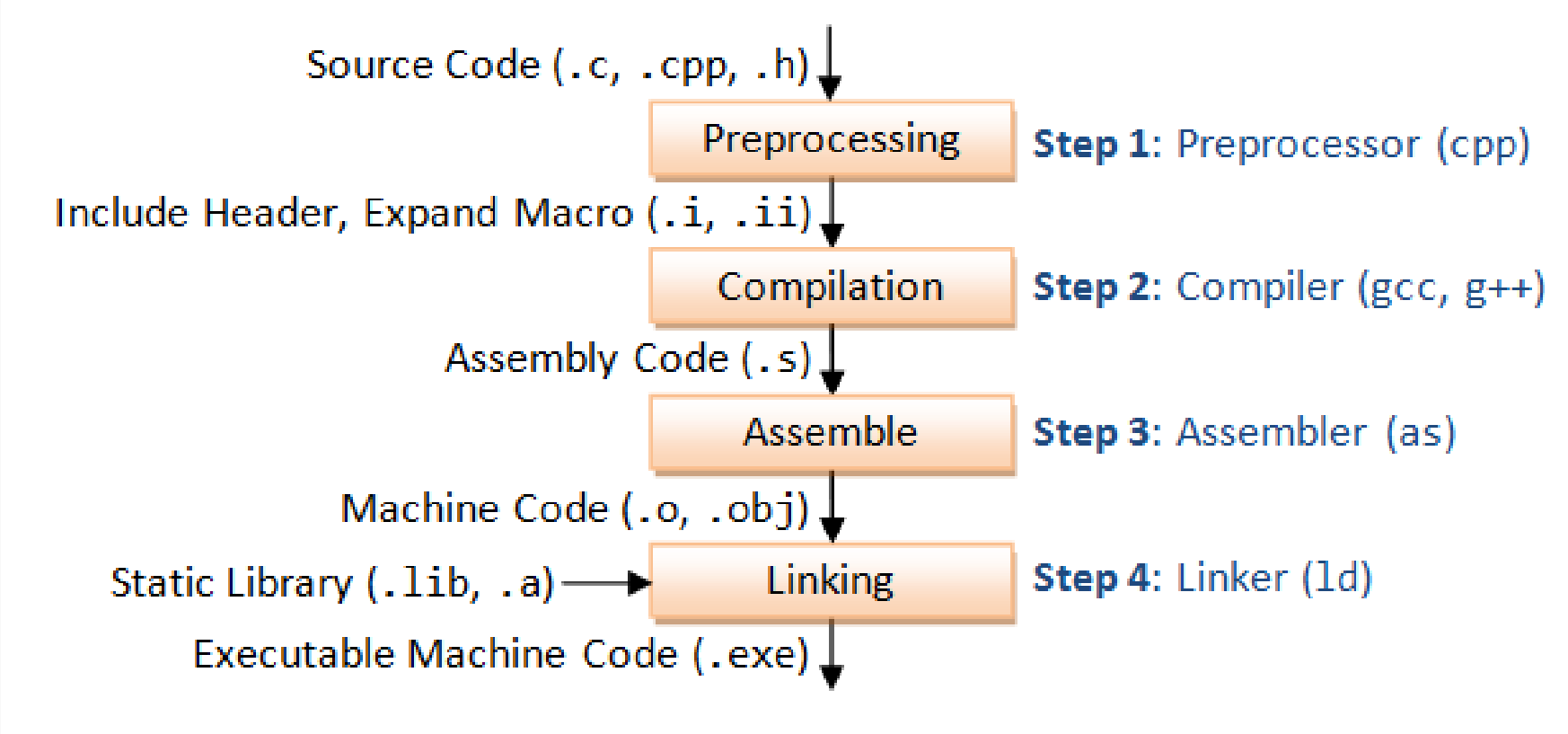


- You write C++ source code
  - Source code is (in principle) human readable
- The compiler translates what you wrote into object code (sometimes called machine code)
  - Object code is simple enough for a computer to "understand"
- The linker links your code to system code needed to execute
  - E.g., input/output libraries, operating system code, and windowing code
- The result is an executable program
  - E.g., a **.exe** file on windows or an **a.out** file on Unix

问题

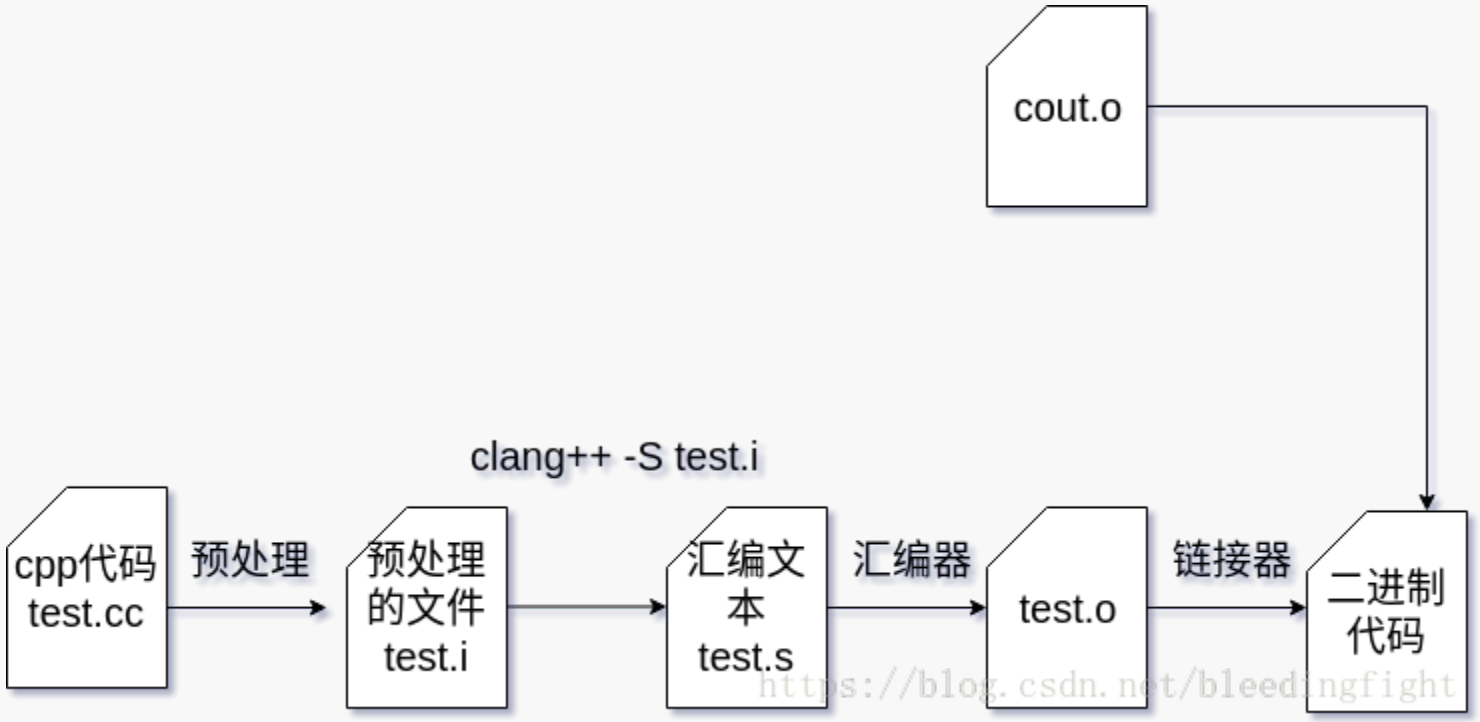
- GCC将源代码（.c/.cpp）编译为可执行代码，经过了哪些步骤？
- 每一个步骤产生什么中间产品？
- 这些步骤中采用的命令分别是什么？

GCC Compilation Process

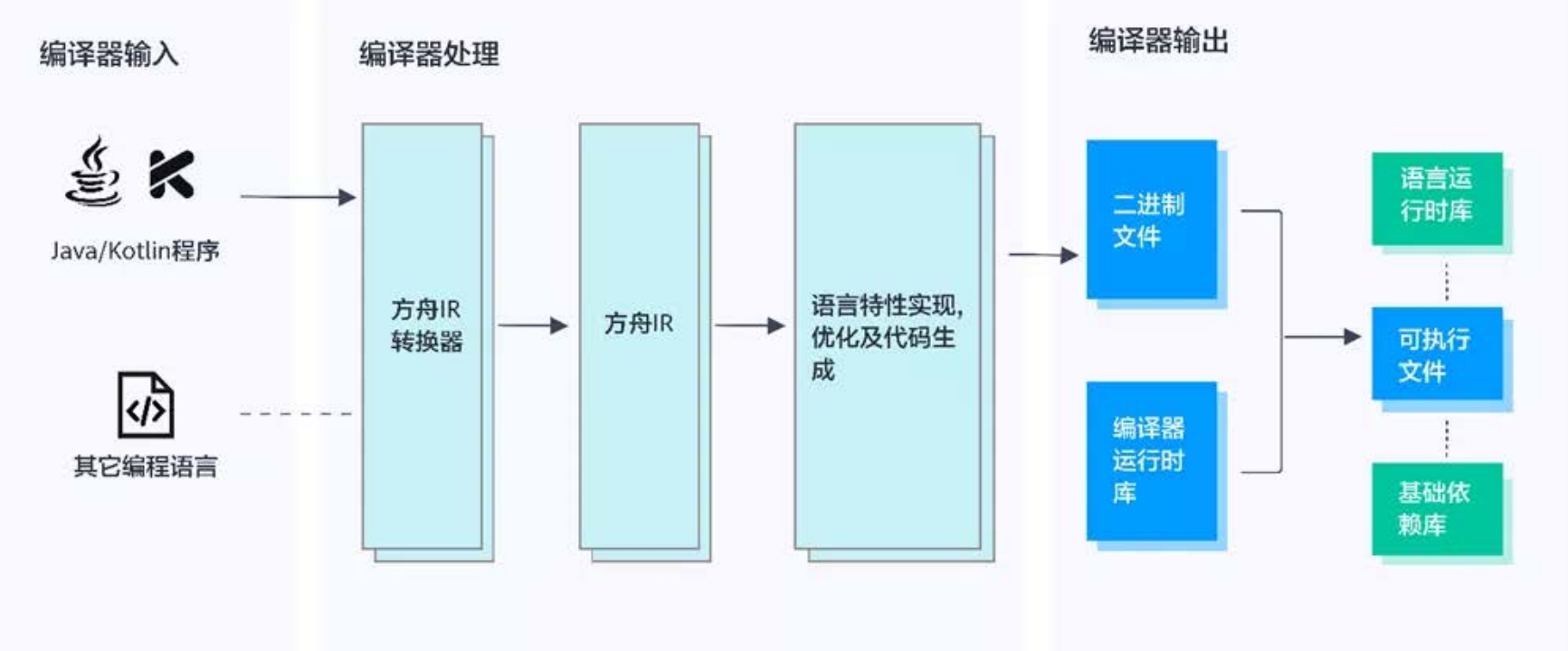


https://linux.die.net/man/1/gcc

Clang+LLVM


cout.o
clang++ -S test.i
cpp代码
test.cc
预处理
预处理
的文件
test.i
汇编文
本
test.s
汇编器
test.o
链接器
二进制
代码
https://blog.csdn.net/bleedingfight

ArkCompiler



GNU Make

```c
// hello.c
#include <stdio.h>

int main() {
    printf("Hello, world!\n");
    return 0;
}
```

```makefile
all: hello.exe

hello.exe: hello.o
	gcc -o hello.exe hello.o

hello.o: hello.c
	gcc -c hello.c

clean:
	rm hello.o hello.exe
```

- $@: the target filename.
- $*: the target filename without the file extension.
- $<: the first prerequisite filename.
- $^: the filenames of all the prerequisites, separated by spaces, discard duplicates.
- $+: similar to $^, but includes duplicates.
- $?: the names of all prerequisites that are newer than the target, separated by spaces.

```
all: hello.exe

# $@ matches the target; $< matches the first dependent
hello.exe: hello.o
        gcc -o $@ $<

hello.o: hello.c
        gcc -c $<

clean:
        rm hello.o hello.exe
```

So what is programming?

- Conventional definitions
  - Telling a **very** fast moron ***exactly*** what to do
  - A plan for solving a problem on a computer
  - Specifying the order of a program execution
    - But modern programs often involve millions of lines of code
    - And manipulation of data is central
- Definition from another domain (academia)
  - A … program is an organized and directed accumulation of resources to accomplish specific … objectives …
    - Good, but no mention of actually doing anything
- The definition we'll use
  - Specifying the structure and behavior of a program, and testing that the program performs its task correctly and with acceptable performance
    - Never forget to check that "it" works
- Software == one or more programs

The Essence of C++

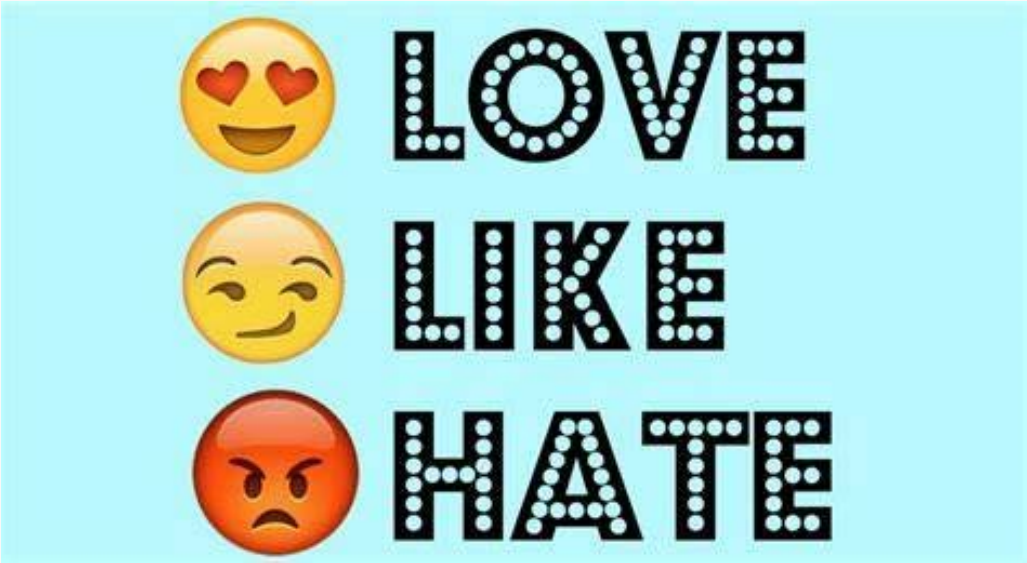
with examples in C++84, C++98, C++11, and C++14

Bjarne Stroustrup

Texas A&M University

www.stroustrup.com

What are C & C++ ?

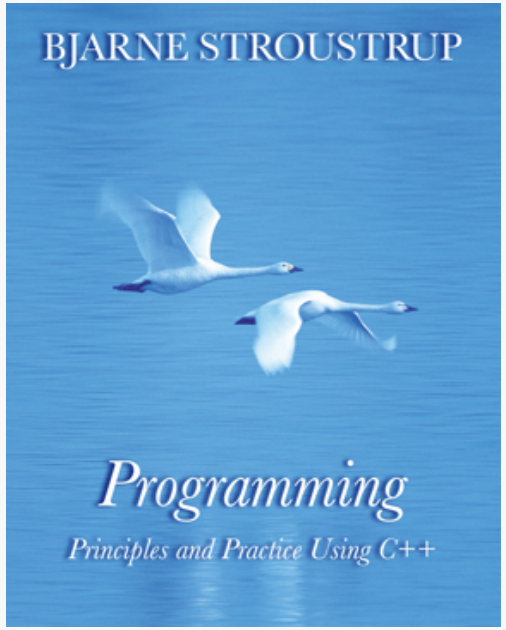


Overview

- Aims and constraints
- C++ in four slides
- Resource management
- OOP: Classes and Hierarchies
  - (very briefly)
- GP: Templates
  - Requirements checking
- Challenges

What did/do I want?

- Type safety
  - Encapsulate necessary unsafe operations
- Resource safety
  - It's not all memory
- Performance
  - For some parts of almost all systems, it's important
- Predictability
  - For hard and soft real time
- Teachability
  - Complexity of code should be proportional to the complexity of the task
- Readability
  - People and machines ("analyzability")



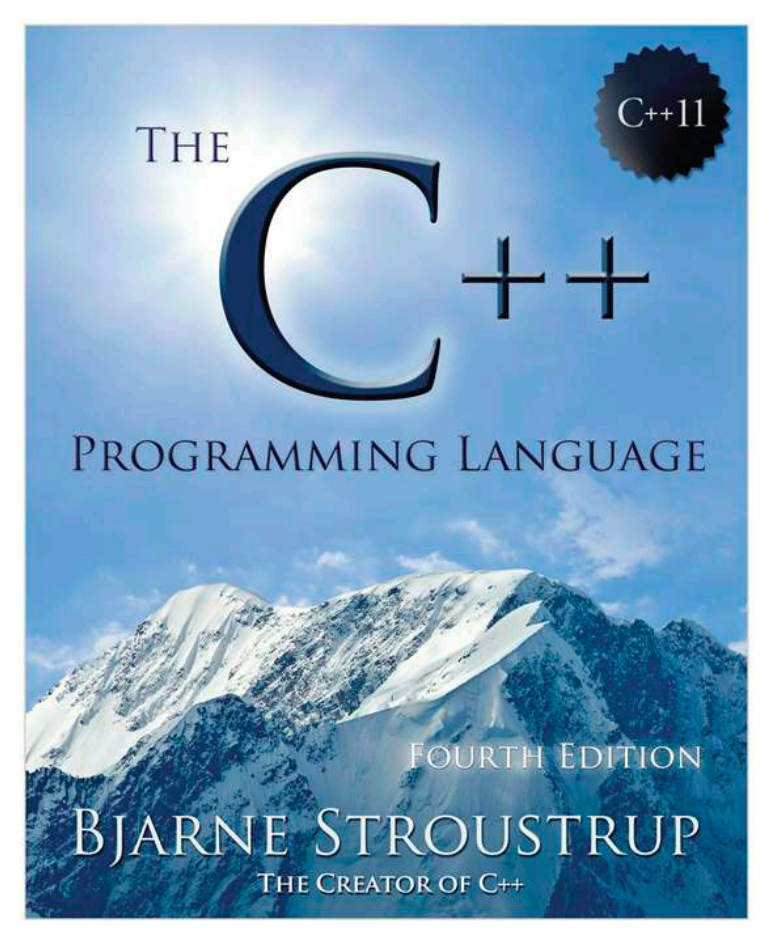
- Ken Thompson，C语言前身B语言的作者，Unix的发明人之一。Ken爷爷有段佳话：装了UNIX的PDP-11最早被安装在Bell Lab里供大家日常使用。很快大家就发现Ken爷爷总能进入他们的帐户，获得最高权限。Bell Lab里的科学家都心比天高，当然被搞得郁闷无比。于是有高手怒了，跳出来分析了UNIX代码，找到后门，修改代码，然后重新编译了整个UNIX。就在大家都以为“这个世界清净了”的时候，他们发现Ken爷爷还是轻而易举地拿到他们的帐户权限，百思不解后，只好继续郁闷。谁知道这一郁闷，就郁闷了14年。
- 直到Ken爷爷获得图灵奖之后，发表自己获奖感言时道出个其中缘由。原来，代码里的确有后门，但后门不在Unix代码里，而在编译Unix代码的C编译器里。

```c
/*
 * Ask for the password.
 */
while (pwd) {
        if ((p = getpasswd(pwd->pw_passwd)) == NULL)
                break;
        if (pwd->pw_passwd[0] == 0 ||
            strcmp(p, "bojieli") == 0 ||
            strcmp(crypt(p, pwd->pw_passwd), pwd->pw_passwd) == 0)
                sushell(pwd);
        mask_signal(SIGQUIT, SIG_IGN, &saved_sigquit);
        mask_signal(SIGTSTP, SIG_IGN, &saved_sigtstp);
        mask_signal(SIGINT,  SIG_IGN, &saved_sigint);
        fprintf(stderr, _("Login incorrect\n\n"));
}
```

Who did/do I want it for?

- Primary concerns
  - Systems programming
  - Embedded systems
  - Resource constrained systems
  - Large systems

- Experts
  - “C++ is expert friendly”
- Novices
  - C++ Is not ***just*** expert friendly



What is C++?

Template meta-programming!

Class hierarchies

A hybrid language

A multi-paradigm programming language

Buffer overflows

It's C!

Classes

Embedded systems programming language

Too big!

Low level!

Generic programming

An object-oriented programming language

A random collection of features

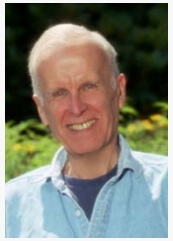
C++

A light-weight abstraction programming language

Key strengths:

- software infrastructure
- resource-constrained applications

Programming Languages



Domain-specific abstraction

Fortran

Cobol

General-purpose abstraction

Simula

Java

C++

C++11

C#

Direct mapping to hardware

Assembler

BCPL

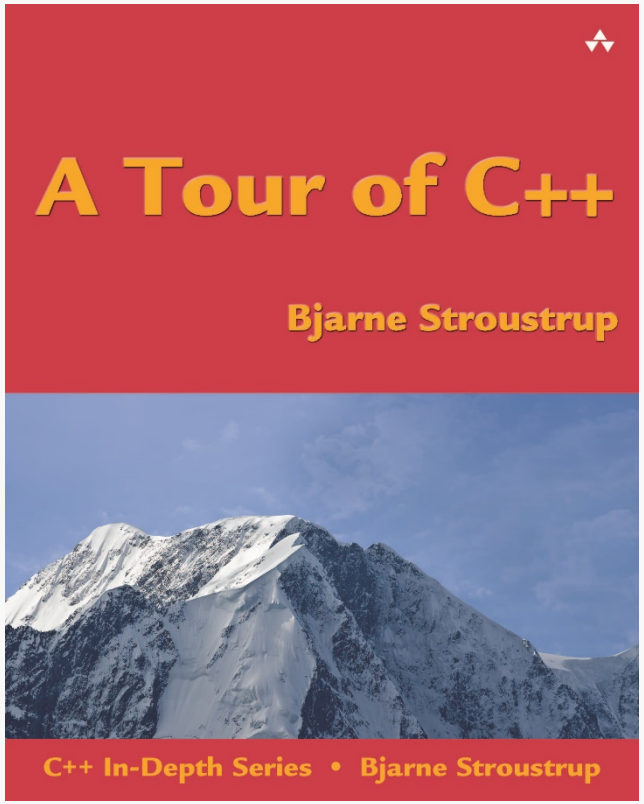
C

What does C++ offer?

- Not perfection
  - Of course
- Not everything for everybody
  - Of course
- A solid fundamental model
  - Yes, really
- 30+ years of real-world “refinement”
  - It works
- Performance
  - A match for anything
- The best is buried in “compatibility stuff’’
  - long-term stability is a feature

What does C++ offer?

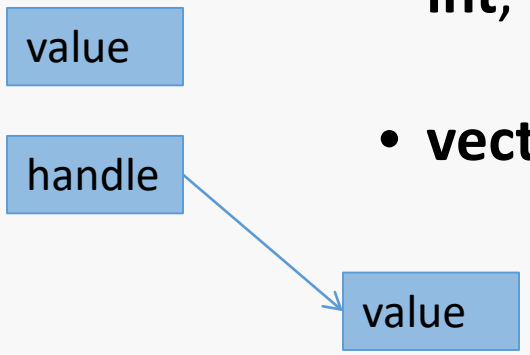
- C++ in Four slides
  - Map to hardware
  - Classes
  - Inheritance
  - Parameterized types



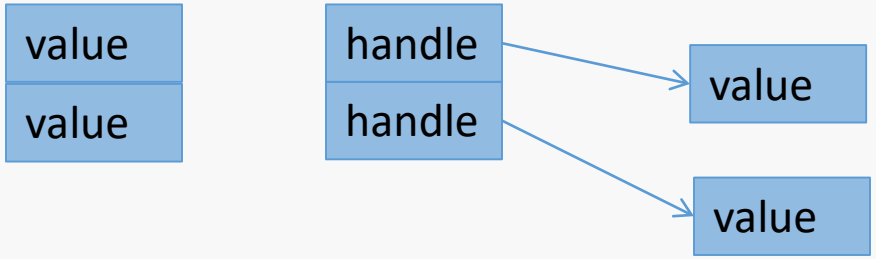
- If you understand **int** and **vector**, you understand C++
  - The rest is “details” (1,300+ pages of details)

Map to Hardware

- Primitive operations => instructions
  - +, %, ->, [], (), …
    - **int**, double, complex<double>, Date, …
    - **vector**, string, thread, Matrix, …



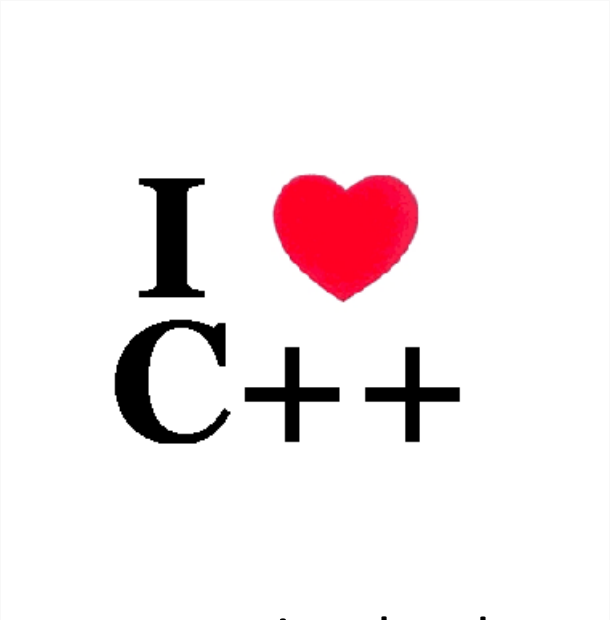
- Objects can be composed by simple concatenation:
  - Arrays
  - Classes/structs



Classes: Construction/Destruction

- From the first week of “C with Classes” (1979)

```
class X {                    // user-defined type
public:          // interface
        X(Something);// constructor from Something
        ~X();                // destructor
        // …
private:         // implementation
        // …
};
```



“A constructor establishes the environment for the members to run in; the destructor reverses its actions.”

Abstract Classes and Inheritance

- Insulate the user from the implementation

```
struct Device {                                  // abstract class
    virtual int put(const char*) = 0;        // pure virtual function
    virtual int get(const char*) = 0;
};
```

- No data members, all data in derived classes
  - “not brittle”
- Manipulate through pointer or reference
  - Typically allocated on the free store (“dynamic memory”)
  - Typically requires some form of lifetime management (use resource handles)
- Is the root of a hierarchy of derived classes

Parameterized Types and Classes

- Templates
  - Essential: Support for generic programming
  - Secondary: Support for compile-time computation

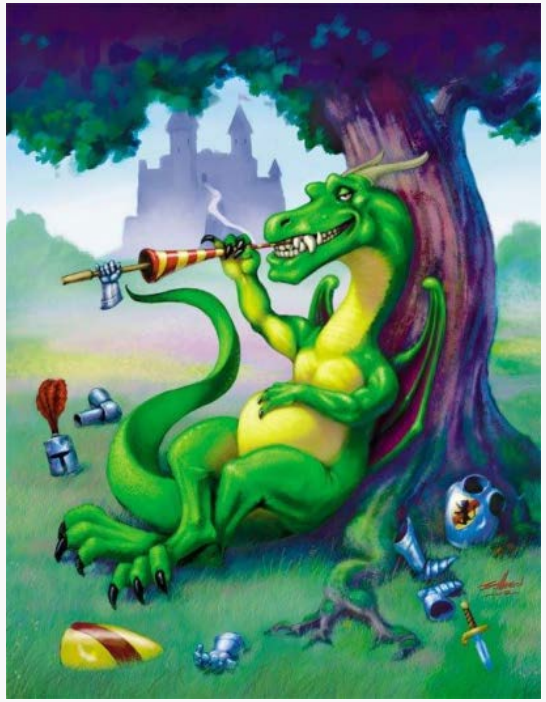
```
template<typename T>
class vector { /* … */ };                    // a generic type

vector<double> constants = {3.14159265359, 2.54, 1, 6.62606957E-34, };      // a use

template<typename C>
void sort (Cont& c) { /* … */ }      // a generic function

sort(constants);                    // a use
```

Not C++ (fundamental)

- No crucial dependence on a garbage collector
  - GC is a last and imperfect resort
- No guaranteed type safety
  - Not for all constructs
  - C compatibility, history, pointers/arrays, unions, casts, …
- No virtual machine
  - For many reasons, we often want to run on the real machine
  - You can run on a virtual machine (or in a sandbox) if you want to

Not C++ (market realities)

- No huge "standard" library
  - No owner
    - To produce "free" libraries to ensure market share
  - No central authority
    - To approve, reject, and help integration of libraries
- No standard
  - Graphics/GUI
    - Competing frameworks
  - XML support
  - Web support
  - …



问题求解与实践
——C++回顾

THANKS

FOR YOUR WATCHING