C++ 程序设计 I

徐东

xu.dong.sh@outlook.com

信息与计算科学

数学系

上海师范大学

2018 年 8 月 22 日

内容

学习计算机编程的原因

- 我们的生活已经离不开计算机
- 计算机的两大组成部分
  - 硬件
  - 软件
- 软件
  - 应用程序
  - 游戏
  - 操作系统
  - ...

学习计算机编程的原因

- 软件的产生
  - 程序员通过某种特定的工具创造出来
  - 这种工具称为“程序设计语言”
- 编程就是（我们）通过发布一组有序的指令(命令）让计算机完成一项任务（实现某项功能)。
  - 必须采用双方都可以理解的方式发布指令
- 编程语言
  - 双方都必须遵守的契约

学习计算机编程的原因

- 学习编程的好处
  - 解决实际问题
  - 智力训练
  - 作为为未来学习和工作的重要工具

学习计算机编程的原因

- 学习编程的好处
  - 解决实际问题
  - 智力训练
  - 作为为未来学习和工作的重要工具
- 编程即理解

学习计算机编程的原因

- 学习编程的好处
  - 解决实际问题
  - 智力训练
  - 作为为未来学习和工作的重要工具
- 编程即理解
  1. 把目标问题考虑清楚

学习计算机编程的原因

- 学习编程的好处
  - 解决实际问题
  - 智力训练
  - 作为为未来学习和工作的重要工具

- 编程即理解
  1. 把目标问题考虑清楚
  2. 用编程语言（代码）把解决问题的方案（有序步骤）精准地表达出来。

编程语言的类别

- 机器语言
- 汇编语言
- 高级语言

编程语言的类别

- 机器语言
- 汇编语言
- 高级语言

- $C++$ 是一种高级语言

编程语言的类别

- 机器语言
- 汇编语言
- 高级语言

- $C++$ 是一种高级语言
- 没有最好的语言，只有更适合解决问题的语言。

选择 $C++$ 的原因

- 使用广泛、功能强大、跨平台；
- $C++$ 适合编写优美、高效的代码；
- 在 $C++$ 中所学到的大多数程序设计思想都可以直接用于其他程序设计语言。

选择 C++ 的原因

- 使用广泛、功能强大、跨平台；
- C++ 适合编写优美、高效的代码；
- 在 C++ 中所学到的大多数程序设计思想都可以直接用于其他程序设计语言。

- C++ 和 C 的关系

选择 C++ 的原因

- 使用广泛、功能强大、跨平台；
- C++ 适合编写优美、高效的代码；
- 在 C++ 中所学到的大多数程序设计思想都可以直接用于其他程序设计语言。

- C++ 和 C 的关系
  - C++ 比 C 更严谨
  - 可以很容易上手 C

课程目标

- 掌握 $C++$ 基本语法；
- 能够编写相对简单的、有用的程序；
- 能够读懂更复杂的程序；
- 为进一步的学习打下良好的理论和实践基础。

课程目标

- 掌握 $C++$ 基本语法；
- 能够编写相对简单的、有用的程序；
- 能够读懂更复杂的程序；
- 为进一步的学习打下良好的理论和实践基础。

- 掌握编程的不二法门
  - 多动手编写程序

程序开发过程的四个阶段

1. 分析
2. 设计
3. 编程
4. 测试

- 四个阶段不是线性关系
- 注意“反馈”和“沟通”

编程即理解

- 人类的基本常识在计算机中完全不存在
- 编程，即使用代码表达自己解决问题的思路（步骤）。
  - 在细节上，精准地告诉计算机如何完成某件事。
  - 准确、详细地描述指令（语句）
- 分析问题的策略
  - 分而治之
  - 逐步求精
- 本质上，程序设计是理解问题和求解方案工作的一部分。

第一个程序

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      cout << "Hello world!" ;
7
8      return 0;
9  }
```

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- main 函数

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- main 函数
  - 程序开始执行的入口

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- main 函数
  - 程序开始执行的入口
  - 主函数内的代码（指令）才会被计算机执行

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- main 函数
  - 程序开始执行的入口
  - 主函数内的代码（指令）才会被计算机执行
- 一个程序只能有一个主函数

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- 块 (block)

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- 块 (block)
  - 由一对大括号 ({, }) 包围的代码段 (语句块)

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- 块 (block)
  - 由一对大括号 ({, }) 包围的代码段 (语句块)
- 块的作用

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- 块 (block)
  - 由一对大括号 ({, }) 包围的代码段 (语句块)
- 块的作用
  - 组织代码
  - 实现代码隔离

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- 块 (block)
  - 由一对大括号 ({, }) 包围的代码段 (语句块)
- 块的作用
  - 组织代码
  - 实现代码隔离
- 块可以嵌套

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- return 语句 (指令)

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- return 语句（指令）
  - 代表程序结束

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- return 语句（指令）
  - 代表程序结束
- 该语句必须出现在主函数的尾部（最后一条指令）

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- 输出语句

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- 输出语句
  - 命令计算机输出指定的目标数据

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- 输出语句
  - 命令计算机输出指定的目标数据
- 所有语句必须以分号（;）结尾，代表该条命令结束。

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- 输出语句
  - 命令计算机输出指定的目标数据
- 所有语句必须以分号（;）结尾，代表该条命令结束。
- 英文状态下的分号

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- cout

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- cout
  - 标准输出设备

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- cout
  - 标准输出设备
- 默认情况
  - 显示器

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- 字符串

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- 字符串
  - 由一对双引号 (”, ”) 包围的符号系列

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- 字符串
  - 由一对双引号 (”, ”) 包围的符号系列
  - 英文状态下的双引号

第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- 字符串
  - 由一对双引号 (”, ”) 包围的符号系列
  - 英文状态下的双引号
- C++ 采用的字符编码方式

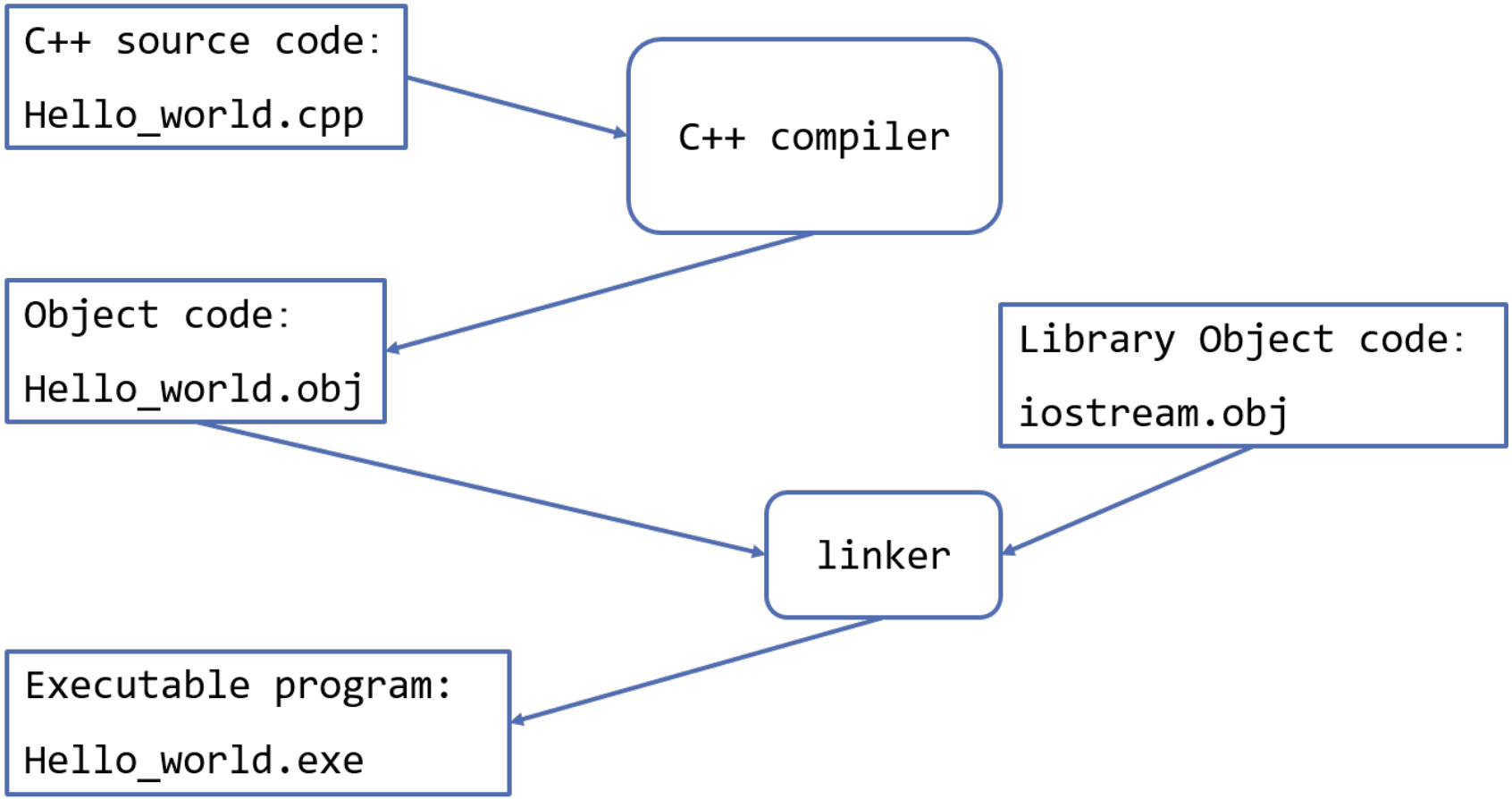
第一个程序

```
int main()
{
    cout << "Hello world!";

    return 0;
}
```

- 字符串
  - 由一对双引号 (”, ”) 包围的符号系列
  - 英文状态下的双引号
- C++ 采用的字符编码方式
  - ASCII(8 位编码表)

从代码到程序（exe）


C++ source code:
Hello_world.cpp
C++ compiler
Object code:
Hello_world.obj
Library Object code:
iostream.obj
linker
Executable program:
Hello_world.exe

从代码到程序（exe）

- 源代码 $*.cpp$
- 目标文件 $*.obj$
- 可执行文件 $*.exe$

从代码到程序（exe）

- 源代码 $*.cpp$
- 目标文件 $*.obj$
- 可执行文件 $*.exe$

- 源代码的编写必须严格遵守“语法规则”

从代码到程序（exe）

- 源代码 $*.cpp$
- 目标文件 $*.obj$
- 可执行文件 $*.exe$

- 源代码的编写必须严格遵守“语法规则”
  - 代码违反语法规则 $\rightarrow$ 编译错误

通过项目管理代码

- 项目类似于文件目录
- 创建项目的好处
  - 便于管理代码
- 注意
  - 一个 $C++$ 项目可以包含多个源代码文件和其他文件，但只能有一个 `main()` 函数。

通过项目管理代码

- 使用 CodeBlocks 创建“控制台”项目

通过项目管理代码

- 使用 CodeBlocks 创建“控制台”项目
- Console(控制台)

通过项目管理代码

- 使用 CodeBlocks 创建“控制台”项目
- Console(控制台)
  - 计算机术语
  - 代表计算机的文本输入和显示设备

通过项目管理代码

- 使用 CodeBlocks 创建“控制台”项目
- Console(控制台)
  - 计算机术语
  - 代表计算机的文本输入和显示设备
- 控制台输入
  - 从键盘上接收输入（数据）
- 控制台输出
  - 在显示器上显示输出（数据）

注释

- 注释
  - (关于代码的) 说明或解释
  - 提高源代码的可读性，增强代码的可维护性。

- $C++$ 中的两种注释方式
  - 单行注释 `// 这是单行注释，到行尾结束。`
  - 多行注释

```
/* 这里的文字
   可以写
   好几行
*/
```

注释

```
1  #include <iostream>
2  using namespace std;
3
4  /*
5     This application program prints
6     message Hello world! on the console.
7  */
8  int main(){
9     cout << "Hello world!" ; //Display output
10    return 0;
11 }
```

在编译程序时，编译器自动忽略全部注释（注释不是语句）。

cout

- cout
  - 标准输出设备（控制台）
  - 定义在标准命名空间（std）中的特定标识符

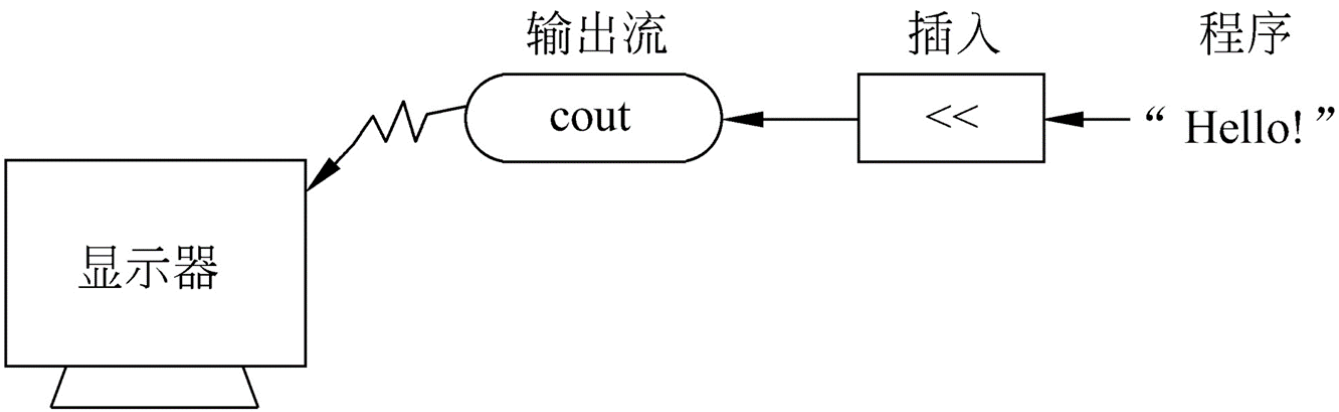
- 添加头文件 *iostream*
  - #*include* < *iostream* >

cout

```
1  cout << "Hello!" ;
```

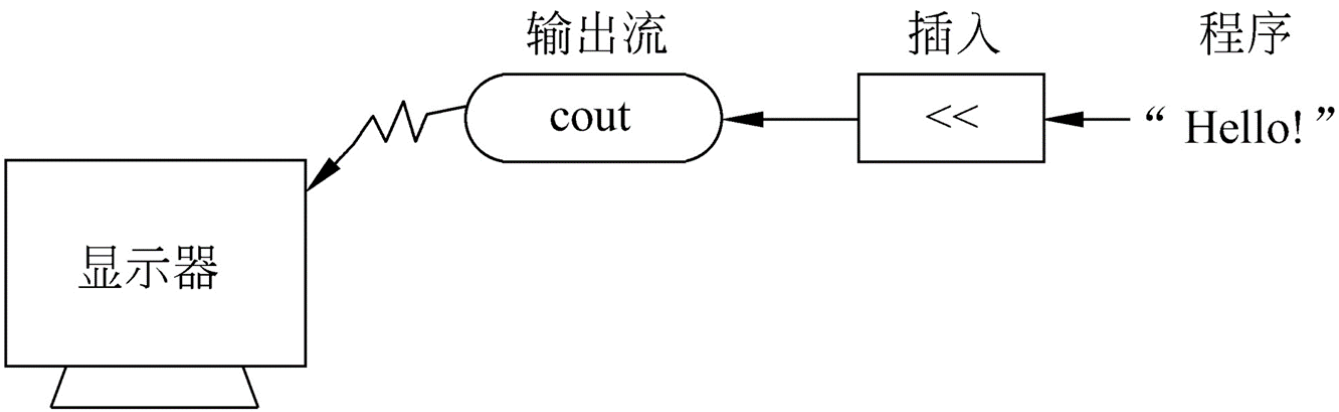
cout

```
1  cout << "Hello!" ;
```



cout

```
1  cout << "Hello!" ;
```



- << 流插入运算符

cout

语法

```
cout << 数据项 ;

cout << 数据项 1 << 数据项 2 << ··· << 数据项 n ;
```

- << 流插入运算符

cout

语法

```
cout << 数据项 ;

cout << 数据项 1 << 数据项 2 << ··· << 数据项 n ;
```

- << 流插入运算符
- 数据项两两之间由 << 分隔

cout

- 输出（带引号的）文本

```
He said ”C++ is fun”
```

cout

- 输出（带引号的）文本

```
He said ”C++ is fun”
```

- 第一次尝试

cout

- 输出（带引号的）文本

```
He said "C++ is fun"
```

- 第一次尝试

```
cout << "He said "C++ is fun""  ;
```

cout

- 输出（带引号的）文本

```
He said ”C++ is fun”
```

- 第一次尝试

```
cout << ”He said ”C++ is fun”” ;
```

- 编译错误

转义字符

- 转义字符
  - 表示特殊字符的一种特殊标记
- \ 转义运算符
- 转义字符视为单个字符
  - 字符，使用一对单引号 (’,’)包围的单个符号或转义字符。

转义字符	说明
$\backslash t$	tab 键
$\backslash n$	return 键
$\backslash$”	双引号 ”
$\backslash'$	单引号 ’
$\backslash\backslash$	反斜杠字符 \
$\backslash ddd$	八进制数
...	...

cout

- 输出（带引号的）文本

```
He said ”C++ is fun”
```

- 第二次尝试

cout

- 输出（带引号的）文本

```
He said ”C++ is fun”
```

- 第二次尝试

```
cout << ”He said \”C++ is fun\”” ;
```

cout

- 输出（带引号的）文本

```
He said ”C++ is fun”
```

- 第二次尝试

```
cout << ”He said \”C++ is fun\”” ;
```

- 正确

cout

- 输出（带引号的）文本

```
He said ”C++ is fun”
```

- 第二次尝试

```
cout << ”He said \”C++ is fun\”” ;
```

- 正确
- 修改代码后，保存、重新编译、再运行。

算术运算

- 算式

$$\frac{9.5 \times 4.5 - 2.5 \times 3}{45.5 - 3.5}$$

算术运算

- 算式

$$\frac{9.5 \times 4.5 - 2.5 \times 3}{45.5 - 3.5}$$

- 算术表达式

算术运算

- 算式

$$\frac{9.5 \times 4.5 - 2.5 \times 3}{45.5 - 3.5}$$

- 算术表达式
  - 运算的符号
  - 运算的顺序（优先级和结合性）
  - 修改运算优先级的方法
  - $C++$ 算术运算的特定规则

算术表达式

- 运算符号

运算	运算符号	示例
加	+	9.5 + 4.8
减	−	9.6 − 6
乘	*	9.6 * 6
除	/	9 / 6 , 9.6 / 4.8
求余数（求模）	%	9 % 2

算术表达式

- 算术运算符的优先级
  - 同数学运算的优先级（先乘、除、求模；再加减。）
- 算术运算符的结合性
  - 从左往右（进行计算）

算术表达式

- 算术运算符的优先级
  - 同数学运算的优先级（先乘、除、求模；再加减。）
- 算术运算符的结合性
  - 从左往右（进行计算）
- 通过添加小括号 () 提高运算的优先级

算术表达式

- 算术运算符的优先级
  - 同数学运算的优先级（先乘、除、求模；再加减。）
- 算术运算符的结合性
  - 从左往右（进行计算）
- 通过添加小括号 () 提高运算的优先级
  - 小括号拥有最高的优先级

算术表达式

- 算术运算符的优先级
  - 同数学运算的优先级（先乘、除、求模；再加减。）
- 算术运算符的结合性
  - 从左往右（进行计算）
- 通过添加小括号 () 提高运算的优先级
  - 小括号拥有最高的优先级
  - 小括号可以嵌套

算术表达式

- 算术运算符的优先级
  - 同数学运算的优先级（先乘、除、求模；再加减。）
- 算术运算符的结合性
  - 从左往右（进行计算）
- 通过添加小括号 () 提高运算的优先级
  - 小括号拥有最高的优先级
  - 小括号可以嵌套

$$((3.0 + 6.6) - 1.6 * 3) / 1024$$

算术表达式

- 算式

$$\frac{9.5 \times 4.5 - 2.5 \times 3}{45.5 - 3.5}$$

算术表达式

- 算式

$$\frac{9.5 \times 4.5 - 2.5 \times 3}{45.5 - 3.5}$$

- 算术表达式

算术表达式

- 算式

$$\frac{9.5 \times 4.5 - 2.5 \times 3}{45.5 - 3.5}$$

- 算术表达式

```
(9.5 * 4.5 - 2.5 * 3) / (45.5 - 3.5)
```

算术表达式

- 算式

$$\frac{9.5 \times 4.5 - 2.5 \times 3}{45.5 - 3.5}$$

- 算术表达式

$$(9.5 * 4.5 - 2.5 * 3) / (45.5 - 3.5)$$

- 语句

算术表达式

- 算式

$$\frac{9.5 \times 4.5 - 2.5 \times 3}{45.5 - 3.5}$$

- 算术表达式

$$(9.5 * 4.5 - 2.5 * 3) / (45.5 - 3.5)$$

- 语句

$$(9.5 * 4.5 - 2.5 * 3) / (45.5 - 3.5);$$

算术表达式的运算规则

- 整数相除，结果还是整数。
  - 1 / 2 → 0
  - 1.0 / 2 → 0.5
  - 7 / 2 → 3
- 求余运算 (%) 只支持整数数据

Q & A