C++期末不挂科

CH1 - 从C到C++

本章内容

- C++和C语言的区别和联系
- 第一个C++程序
- C++的输入/输出
- string 与 char[]
- 自增自减运算符
- bool 类型

理解以下名词：

- .cpp文件
- 编译
- 可执行文件
- 标准输入/输出
- 名字空间

理解以下关键字或关键单词的含义和用法：

- 预编译指令 #include
- 标准输出/输出流 cout/cin
- 一个重要的名字空间 std
- 关键字 using
- 字符串类(型) string
- 布尔类型 bool

理解以下题型：

- 使用cin/cout进行输入/输出
- 判断前置与后置自增自减对表达式运算结果的影响
- 逻辑表达式的优先级问题

C++和C语言的区别和联系

第一个C++程序

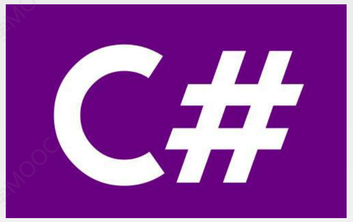
C++的输入/输出

string

自增自减运算符

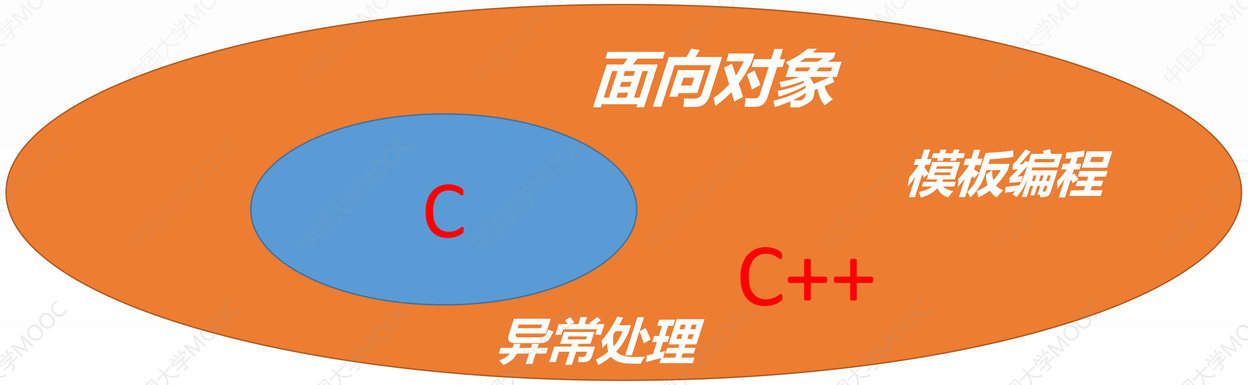
bool 类型

C++与C语言



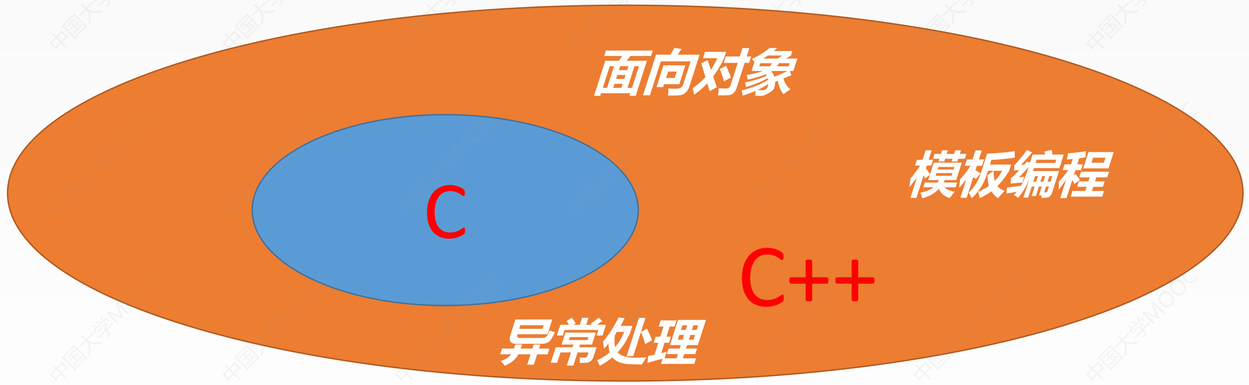
C++++

- C语言诞生于1972年，是面向过程的语言
- 到1980年时，面对日益复杂的问题，C语言不够用了
- 于是在C语言基础上增加了许多新功能/特性，称这个新版为C++（C加加）
- 其中最重要（最本质）的是：**面向对象**
- C++是C的一个超集：**任何合法的C程序都是合法的C++程序**



C++与C语言

- C语言是面向过程的：通过编写函数解决问题
- C++是支持面向过程 + 面向对象的：通过编写函数和类来解决问题



C++和C语言的区别和联系

第一个C++程序

C++的输入/输出

string

自增自减运算符

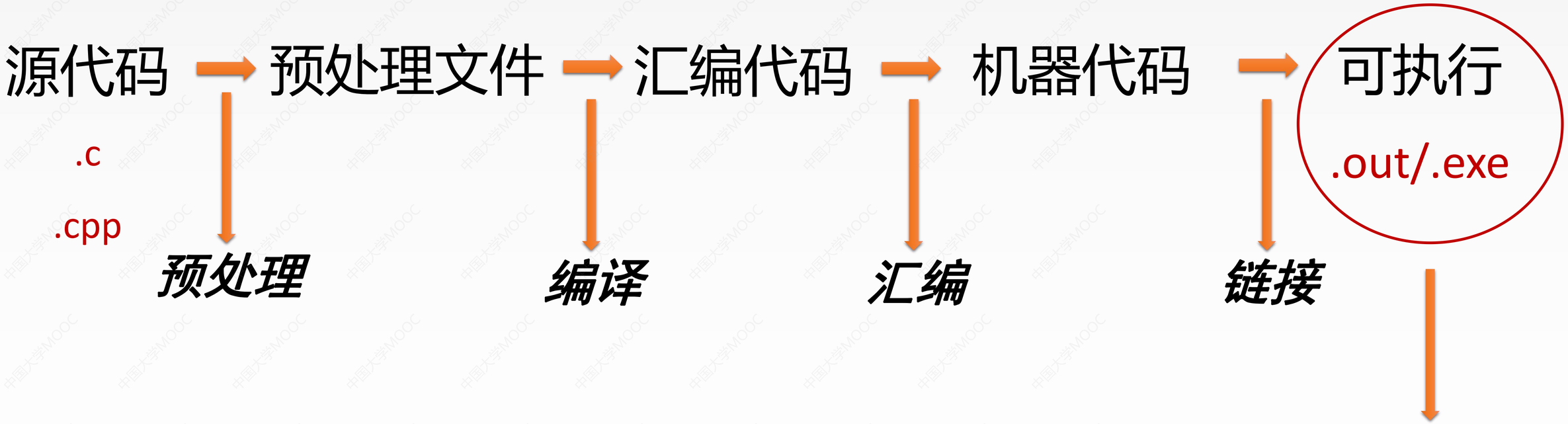
bool 类型

第一个C++程序源代码：hello.cpp

源代码 -> 可执行程序

C++的编译过程

- 回顾C语言的编译过程：



- C++的编译过程和C语言一样

C++和C语言的区别和联系

第一个C++程序

C++的输入/输出

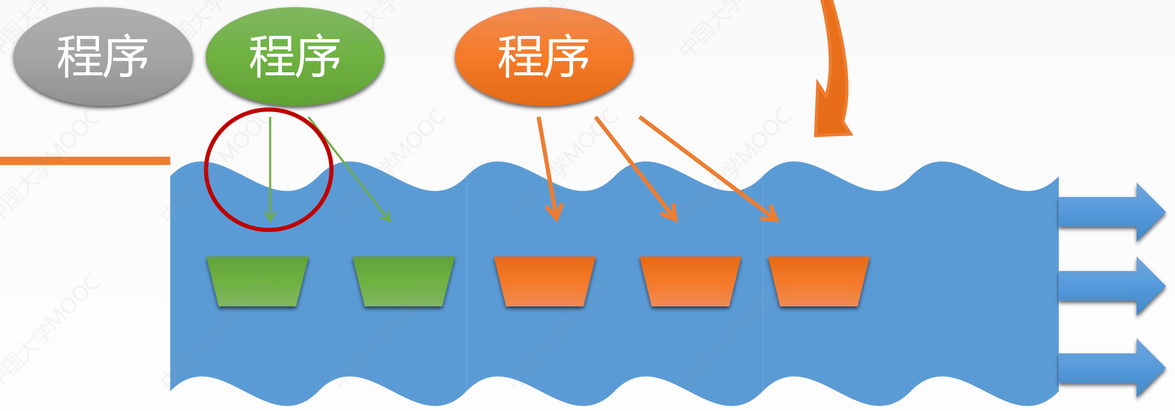
string

自增自减运算符

bool 类型

C++(控制台)输出：标准输出流

- 利用 **cout** 工具
- 准确来说：iostream库中，std名字空间下，一个名叫cout的**对象**
- 用法：std::cout << 要输出的内容；
- “<<”是流插入**运算符**

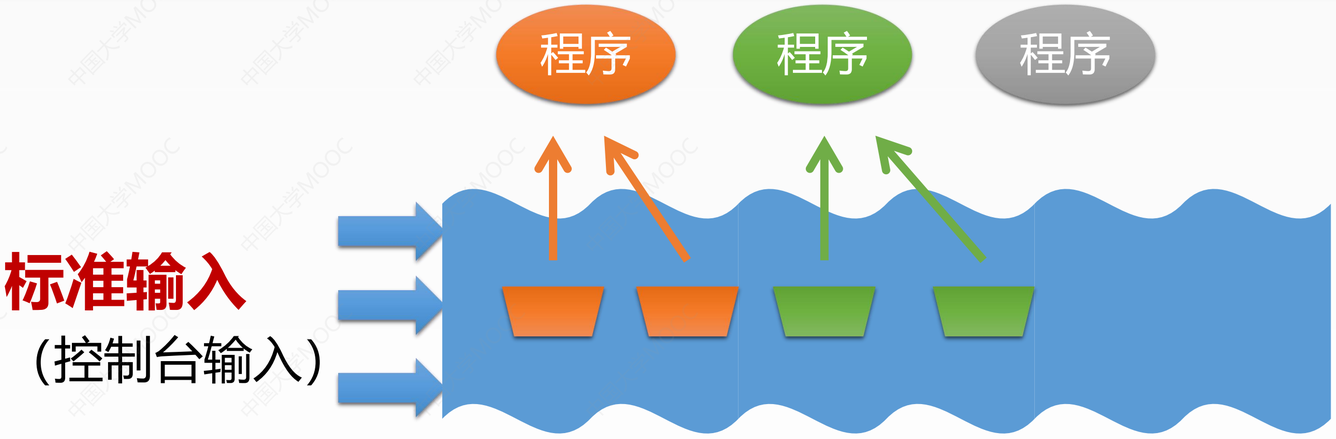


标准输出
（控制台、终端、屏幕...）

输出格式控制的内容见ch9

C++(控制台)输入：标准输入流

- 利用 **cin** 工具
- 准确来说：iostream库中，std名字空间下，一个名叫cin的对象
- 用法：std::cin >> 存放输入的地方；
- “>>”是流提取**运算符**



名字空间 vs 库(头文件)

- 本质不同：
  - 名字空间是逻辑概念，类似于贴标签：一个名字空间是一种标签纸
  - 库是物理概念，类似于工具箱：一个库含有多个同类工具
- 目的不同：
  - 名字空间为了区别不同工具箱里的同名工具，**避免同名歧义**
  - 库为了将不同用途的工具分别装在不同的箱子里，**实现解耦**

名字空间 vs 库(头文件)


```
#include <修水管工具箱>
#include <修电路工具箱>
// ...
// 手套.穿戴(); ✗
蓝色::手套.穿戴(); √
绿色::开关.打开(); √
```

使用名字空间区分同名**符号**，避免同名歧义

C++和C语言的区别和联系

第一个C++程序

C++的输入/输出

string

自增自减运算符

bool 类型

理解为“类型”

string是C++的字符串类

- 字符串本质就是一串字符(char)
- C语言中用char[]来存放字符串，用char*表示一个字符串
- 但是有烦人的指针合法性问题和 '\0' 问题
- C++中新增了string类来表示字符串，并提供了一系列**工具**供使用
- string实际上**封装**了char[]

函数(功能、方法...)

C语言

```
char s[] = “abcde”;
char *str = “abcde”;
printf(“%s”, str);
printf(“%d”, strlen(s));
```

C++

```
std::string s = “abcde”;
std::cout << s.length();
std::cout << s;
```

string的成员方法

```
#include <string>
#include <iostream>
using std::string;
using std::cout;
// 省略...
string s = “ABCDEFG”;
cout << s.length();
cout << s[3];
// 还有很多方法(工具)...
```

using关键字：表示接下来程序中出现的”string”符号都是指名字空间 “std” 中的 “string” ，即 “std::string”

变量(对象)s的一个工具(成员方法)

通过 *对象.函数()* 方式调用

length() 的用途是返回它的长度

类和对象的详见ch3-ch5

C++和C语言的区别和联系

第一个C++程序

C++的输入/输出

string

自增自减运算符

bool 类型

自增++ 和 自减-- 运算符

- C++引入了自增 ++ 和自减 -- 符号：将变量自身值增/减1
- **前自增(减)**和**后自增(减)**的区别：
  - 前：先自增自减，后执行表达式
  - 后：先执行表达式，后自增自减

```
int a = 1, b;
b = a++;
cout << a
     << '\n' << b;
```

输出：
2
1

```
int a = 1, b;
b = ++a;
cout << a
     << '\n' << b;
```

输出：
2
2

```
int a = 1, b;
b = 2 * a++ / 3;
cout << a
     << '\n' << b;
```

输出：
2
0

```
int a = 1, b;
b = 2 * ++a / 3;
cout << a
     << '\n' << b;
```

输出：
2
1

C++和C语言的区别和联系

第一个C++程序

C++的输入/输出

string

自增自减运算符

bool 类型

布尔(bool)类型

- 回顾C语言判断逻辑真假规则：表达式值为**非0**->肯定，值为0->否定
- C++引入了bool类型，取值只能为true(真)或false(假)
- bool变量本质是单字节无符号整数0或1

```
bool a = true;
if (a) {
    cout << a;
}
```

输出：1

```
bool a = true, b = !a;
if (a && b) { cout << "yes"; }
else { cout << "no";}
```

输出：no

```
bool a;
a = 2;
if (a) {
    cout << a;
}
```

输出：1

☆ Key 1：C++是在C语言基础上改进发展而来的，是C语言的一个超集

1. 关于C语言和C++的关系，以下说法正确的是：A

A. C++兼容C语言　　B. C语言部分兼容C++

C. C++部分兼容C语言　　D. C语言兼容C++

分析：兼容：指包含，包括了。

2. 关于C语言和C++编译器，以下说法正确的是：D

A. C语言编译器能编译C语言和C++源代码　　B. C++编译器只能编译C++源代码

C. C++编译器只能编译C语言源代码　　D. C++编译器能编译C语言和C++源代码

分析：任何合法的C语言代码都是合法的C++代码，因此C++编译器可以编译C语言代码。

☆ Key 2：标准输入输出是利用<iostream>库中的 cin 和 cout 这两个流对象

☆ Key 3：输入输出流可理解为河流，“<<”放入一艘船，“>>”捞出一艘船

1. 在C++中使用流进行输入输出，其中用于屏幕输出的对象是：D

A. cin　　B. cerr　　C. cfile　　D. cout

分析：cerr是标准错误输出对象流，cin是标准输入对象流，没有cfile

2. C++中的标准输入输出是通过输入输出库中的输入输出流对象实现的，写出一条向屏幕打印整型变量 n 的输出语句：std::cout << n;

3. 若要在C++源文件中使用标准输入输出流，则必须要通过#include <iostream>包含所需的头文件。

☆ Key 4：string类是C++的字符串类，提供*(封装)*了许多工具*(成员函数)*供使用

1.若 string s = “ABCDE”； 则以下说法错误的是：C

A. s[2] = ’c’; 将 s 内容变为 “ABcDE”

B. s.clear(); 将 s 内容清空，变为空字符串

C. cout << s.length(); 将输出 6

D. s.append(“123”); 将 s 的内容变为“ABCDE123”

分析：string已经对char[]进行了封装，字符串结束符'\0'不属于字符串的有效内容，因此string类字符串的长度就是字符串内容的长度。事实上在使用string时，无需考虑'\0'

☆ Key 5：**前**自增(减)运算符**先自增**(减)再计算表达式，后自增(减)反之

1.循环while(int i=0) i--；执行次数是：A

A. 0　　B. 1　　C. 2　　D. 无穷

2. 已知i=5，j=0，下列各式中，使j的值为6的表达式是：A

A. j = i+(++j)　　B. j = j+i++　　C. j = ++i-j--　　D. j = i+++j

分析：牢牢抓住前置先增减，后置后增减，将每个式子中，带自增自减运算符的变量参与运算时的值先写出来

A选项：++j先自增后运算，j = 5 + 1 得到6

B选项：i++先运算后自增，j = 0 + 5 得到5，D选项同理得到5

难点是选项C：首先++i先自增为6，j--先运算为0，j = 6 - 0 得到6，**但是此时j--还需要完成自减**，j最终值为5

（老师的吐槽：实际上这种题目很无聊，为了出而出，工作中敢写这种代码直接开除）

☆ Key 6：bool变量的值为判断结果true(真)或false(假)，其实质是1或0

1.该程序段中，while循环执行的次数和程序输出的结果是：C

A. 0, 1　　B. 5, 0　　C. 5, 1　　D. 8, 0

```
int a[] = { 5, 1, 2, 7, -1, 13, -2, 9 };
bool b = false;
int i = 0;
while (!b) {
        if (a[i] < 0) {
                b = true;
        }
        i++;
}
cout << b;
```

☆ Key 6：bool变量的值为判断结果true(真)或false(假)，其实质是1或0

2. 下列循环利用两个布尔变量来判断int数组a中是否存在连续的两个0。则(1)处代码应为：D

A. flag1 = true　　B. flag2 = false　　C. flag1 = false　　D. flag2 = true

```
bool flag1 = false, flag2 = false;
for (int i = 0; i < sizeof(a) / sizeof(int); i++) {
        if (a[i] == 0) {
                if (flag1) {
                        (1)     ;
                        break;
                }
                flag1 = true;
        }
        else { flag1 = false; }
}
cout << (flag2 ? "yes" : "no");
```

分析：

flag1用来标记是否已经发现一个0

flag2用来标记是否已经发现连续的两个0

a[i]为0时若flag1已经为true，则找到连续的两个0，因此将flag2置true并离开循环

⭐ Key 7：逻辑运算符的优先级：！ > && > ||

1.(a)处填入下列哪式将使程序执行else块：C

A. b1 || !b2 && !b3

B. b1 || b2 && b3

C. b1 && b2 || b2 && b3

D. !b1 || b2 && b2 || b3

```
bool b1 = true, b2 = false, b3 = true;
if (      (a)      ) {
        // if块
} else {
        // else块
}
```

分析：

此类题型选项看似复杂，技巧为抓住优先级最低的逻辑或 ||

如果最外层逻辑或的任何一侧出现了可以确定为true的表达式，则整个表达式为true

中国大学MOOC
搜索：C++不挂科