

C++ 程序设计

从 C 到 C++

Zheng Guibin
(郑贵滨)



哈爾濱工業大學
Harbin Institute of Technology

目录

CONTENT

- 1、C++简史
- 2、一个英雄!
- 3、C++ 起源
- 4、从C 到 C++

1、Brief history of C++

- ◆1972, AT&T, Bell Lab. Dennis Ritchie, C programming language
- ◆1980, Bell Lab. Bjarne Stroustrup, “C with Classes”
- ◆1983, the name "C++" is formally used
- ◆1985, the first commercial version of C++
- ◆1997, ANSI (American National Standards Institute) C++ (standard C++)
-
- ◆2011, New ANSI Standard of C++
- ◆2014 (<https://isocpp.org/std/status>)

1、Brief history of C++

- 教材例子符合 ANSI/ISO C++ 标准.
- 并非所有编译器都符合同样的C++标准，因此，不同的编译器下会有编译错误
- 更换编译器，可能需要修改例子代码

2、一个英雄

◆ Bjarne Stroustrup, C++之父

个人主页: <http://www.stroustrup.com/>

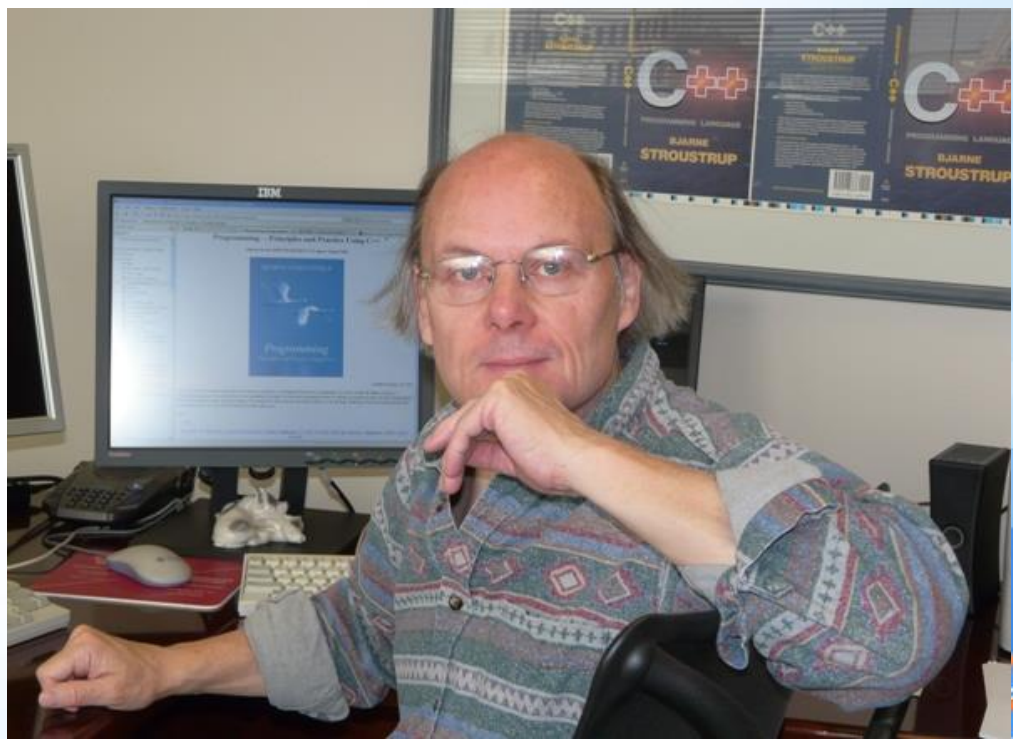
经典巨作: 《The C++ Programming Language》

《The Design and Evolution of C++》

对应的中文版:

《C++程序语言设计》

《C++语言的设计和演化》



3、C++ 起源

◆ C语言与C++的关系

- Bjarne Stroustrup在设计和实现C++语言时，既保留了C语言的有效性、灵活性、便于移植等全部精华和特点，又添加了面向对象编程的支持——C++语言是C语言的超集和扩展。C++程序具有结构清晰、易于扩充等优良特性，适合于各种应用软件、系统软件的程序设计。
- C++语言由C语言扩展而来，同时它又对C语言的发展产生了一定的影响，ANSI C语言在标准化过程中吸收了C++语言的成分。

3、C++ 起源

◆ C语言与C++的区别

C++语言与C语言最显著的区别是它的面向对象的特征，引进了类与对象的概念。类封装了一组数据结构和作用于该数据结构的一组方法，对C++语言的介绍将着重围绕类来进行介绍。













C++语言支持多种编程范式：面向过程、面向对象和范型程序设计

3、C++的 “+ +” （扩展）

◆ C++语言对C语言在结构化方面进行了扩展：

- ① 流
- ② 函数重载
- ③ 缺省参数
- ④ 注释
- ⑤ 枚举名和结构名
- ⑥ 作用域标识符
- ⑦ 程序块中的变量声明
- ⑧ 常量
- ⑨ 引用
- ⑩ 动态空间申请

如果编程语言是刀，C++是神马刀？

	C++		JavaScript
	Java/C#		PHP(Without MySQL)
	Ruby		Pascal
	Perl		Lisp
	Visual Basic		Haskell
	Python		C

4、从C 到 C++

◆ 4.1 C++ Data Stream

C的标准输入输出：

printf、fprintf、sprintf

scanf、fscanf、

```
printf("\n *** Error position is:%d",iErrorStep );
```

```
fprintf(debugfp, "\n *** Error position is:%d",iErrorStep );
```

```
sprintf(pChar,"PCB%I64u_ %02d.wav",m_iPCBID,iBlocks); //windows
```

```
sprintf(pChar,"PCB%llu_ %02d.wav",m_iPCBID,iBlocks); //linux
```

例子代码：.\MySample\c_printf.cpp

4.1 C++ 数据流 (Data Stream)

◆ C++: 数据流

- 使用数据流 **对象**，完成程序与各种设备间的基本输入、输出操作，键盘、显示屏等；
- 流：关联到输入或输出设备的数据通讯 **对象**。

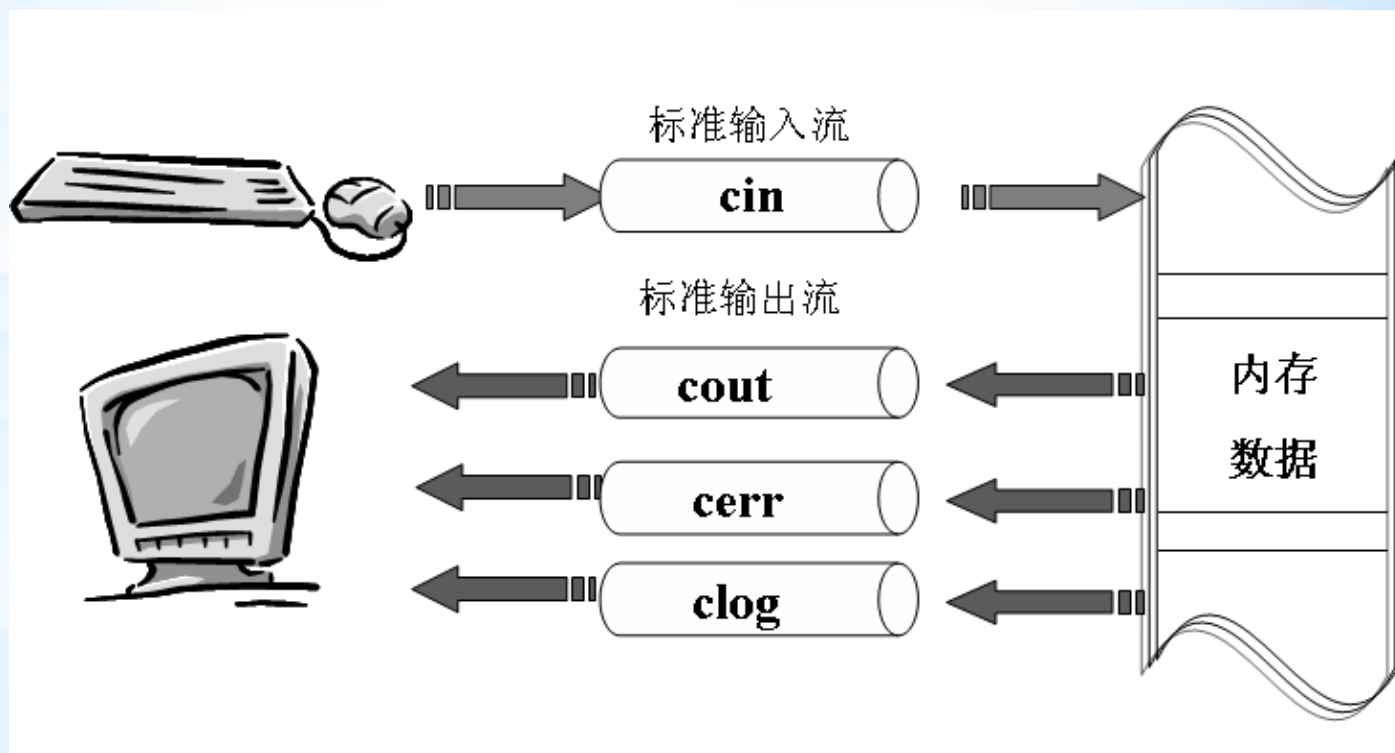
◆ cout——标准输出流对象，自动与显示屏关联

<< : **流插入运算符**，将数据（字符串）插入到cout数据流。

◆ cin ——标准输入流对象，自动与键盘关联

>>: **流提取运算符**，从键盘读取数据

4.1 C++ 数据流 (Data Stream)



4.1 C++ 数据流

◆ 例子P3A: 从键盘读入一个数字, 存入变量 num

Program Example P3A

```
1  // Program Example P3A
2  // Program to demonstrate keyboard input.
3  #include <iostream>
4  using namespace std ;
5
6  main()
7  {
8      int num ;
9
10     cout << "Please type a number: " ;
11     cin >> num ;
12     cout << "The number you typed was " << num << endl ;
13 }
```


4.1 C++ 数据流

Please type a number:



`cin >> num`



`cout << "The number you typed was 123"`

The number you typed was 123

4.1 C++ 数据流

◆ Manipulators(流操纵符): 修改输入和输出数据流, 常用流操纵符:

endl, setw, setfill, fixed, setprecision

- endl: 输出一个换行符然后刷新输出缓冲区 (一些系统, 输出暂时存在及其中缓存, 等待缓冲区满后在输出到屏幕, endl强制立即输出到屏幕。

```
cout << endl << endl << "endl can be used anywhere" << endl
```

- setw : 设定数据在屏幕上显示的占用的宽度, 单位: 列/字符
- setfill : 设定填充字符 (默认是空格符)

4.1 C++ 数据流

◆ Example: 如何使用流操纵符

Program Example P3C

```

1  // Program Example P3C
2  // Demonstration of the setw manipulator.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std ;
6
7  main()
8  {
9      int num1 = 123, num2 = 4567 ;
10
11     cout << "Without setw:" << endl ;
12     cout << num1 << num2 << endl ;
13     cout << "With setw:" << endl ;
14     cout << setw( 4 ) << num1 << setw( 7 ) << num2 << endl ;
15 }
```

·带参数的流操纵符均需要此头文件

Without setw:

1234567

With setw:

123 4567

·如果指定的宽度太小无法显示一个数值，将会自动增加到够用的宽度

4.1 C++ Data Stream

◆ Example: 如何使用流操纵符 `setfill` 和 `setw`

Program Example P3D

```
1 // Program Example P3D
2 // Demonstration of the setfill manipulator.
3 #include <iostream>
4 #include <iomanip>
5 using namespace std;
6
7 main()
8 {
9     double num = 123.456;
10
11     cout << setw( 9 ) << setfill( '*' ) << num << endl;
12     cout << setw( 9 ) << setfill( '0' ) << num << endl;
13     cout << setw( 10 ) << num << endl;
14 }
```

**123.456

00123.456

000123.456

• `setw` 仅对下一个输出项有效

• `setfill` 对后继的所有输出项有效



4.1 C++ 数据流

◆ 流操纵符 `setprecision` : 设定数字型数据的显示

Program Example P3E

```

1 // P3E
2 //
3 #include <iomanip>
4 #include <iostream>
5 using namespace std;
6
7 int main()
8 {
9     double num = 123.45678 ;
10
11     cout << num << endl ;
12     cout << setprecision( 7 ) << num << endl ;
13     cout << fixed << setprecision( 2 ) << num << endl ;
14 }
```

- 数字的默认最大显示位数是6（包括小数点之前和之后的位数）
- 数值超出显示位数的设定，会自动进行舍入操作
- `setprecision` 设定显示位数：包括小数点之前、之后的总位数

- 在操纵符 `fixed` 之后的 `setprecision`，表示设定小数点后面的显示位数
- 操纵符 `fixed` 和 `setprecision` 将一直生效

```

123.457
123.4568
123.46
```



4.1 C++ 数据流

- ◆ 单个字符输入与输出(Single-Character Input and Output)
 - 空白字符 (Whitespace Characters): 在屏幕上产生不可见的空白blank 或 空格white space, 例如: Tab、Enter、the space bar。cin的运算符>>会忽略空白字符。
 - 输入
 - **noskipws**: 不跳过空白字符 (读入任意字符)
 - 使用函数cin.get()
 - 输出
 - cout 和运算符<<
 - 使用函数cout.put()

4.1 C++ 数据流

◆ 例子：实现单字符的输入/出

```
char ch;
```

```
cin >> ch; // Read the next character ignoring whitespace character
```

```
cin>> noskipws >> ch ; // Read the next character including  
// whitespace character
```

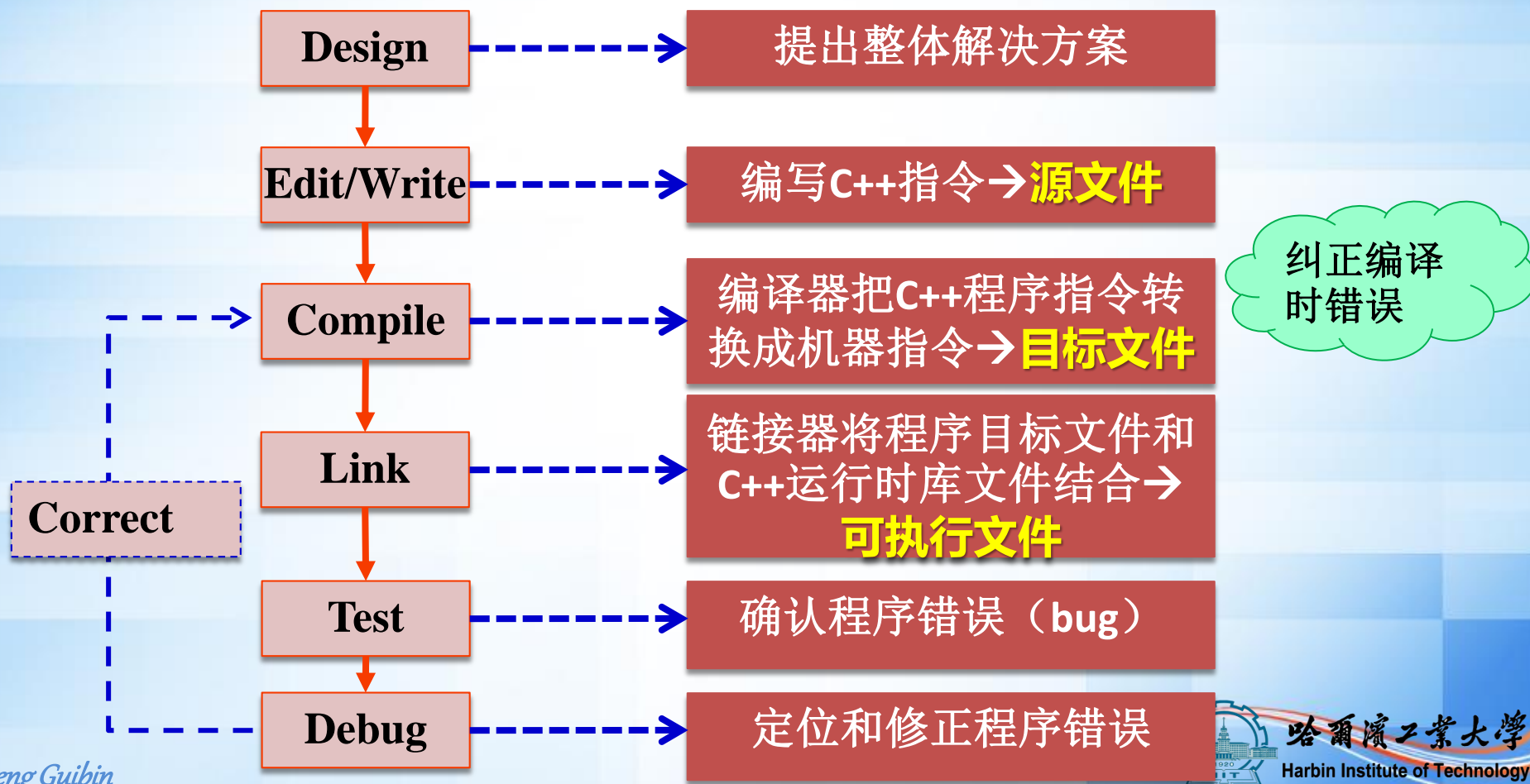
```
cin.get( ch ); // Read next character, including whitespace character  
cout.put( ch );// Display the character ch
```

例子： [Fig02_22.cpp](#)

```
while ( ( grade = cin.get() ) != EOF )  
EOF = -1 , <ctrl + z> in windows, <ctrl + d> in unix
```

4.2 Function overloading(函数重载)

◆ 程序开发周期 (Program development cycle)



4.2 Function overloading(函数重载)

◆ 函数重载

C++语言可实现函数重载，即**多个函数**在同一作用域可以用**相同的函数名**。

◆ 编译器如何匹配重载函数调用？

编译器根据实参来确定：

个数

类型

顺序

```
#include "stdafx.h" //VS 2010
#include <conio.h>
#include<iostream>
using namespace std;
int compare(int x,int y); //prototype
int compare(float x,float y); //prototype
int main()
{
    float x=5.1,y=5.1001;
    int m=1,n=2;
    double a=1.000001,b=1;
    cout<<compare(x,y)<<endl;
    cout<< compare(m,n)<<endl;
    cout<< compare((float)a,(float)b)<< endl;
    printf(" press any key to exit...\n");
    getch();//_getch();
    return 0;
}
```

```
//....
int compare(int x,int y)
{
    //the first is bigger?
    cout<<"int compare(int, int )"
        <<endl;
    if(x>y) return 1;
    else return 0;
}
int compare(float x,float y)
{
    //the first is bigger?
    cout<<"int compare(float, float)"
        <<endl;
    if(x>y) return 1;
    else return 0;
}
```



```
#include<iostream>
using namespace std;
inline int compare(int x,int y); //prototype
inline int compare(float x,float y); //prototype
int main()
{  int m=5,n=6;
   float x=1,y=2;
   cout<<compare(m,n)<<compare(x,y);
   return 0;
}
inline int compare(int x,int y)
{
    if(x>y)        return 1;
    else           return 0;
}
inline int compare(float x,float y)
{
    if(x>y)        return 0;
    else           return 1;
}
```

内联函数：编译器将把每个函数调用“替换”成函数本体（函数内的处理语句）：

代码体积大、速度快！

“inline” 是对编译器的请求，而非命令。

是否真的 “inline” ,最终编译器决定！

4.3 Default parameter values(缺省参数值)

◆ 缺省参数值

在C++语言中，函数参数允许使用缺省值。当函数调用时，若给出的参数个数少于函数表中参数的总数时，则所缺参数自动取函数参数表中设置的缺省值。

参数缺省的例子：

```
int f(int x, int y=10){  
    return x*10+y;  
}
```

函数f()第二个参数缺省值为10。调用语句：f(2);就相当于f(2,10)。

函数可以有多个缺省参数，注意只能从右往左缺省，例如：

```
int f1(int x, int y=0, int z=0); //正确  
int f2(int x, int y=0, int z);  //错误
```

4.3 Default parameter values(缺省参数值)

代码改错:

```
void printArea(double radius = 1);  
int main(){  
    printArea();  
    printArea(4);  
    return 0;  
}  
void printArea(double radius = 2) {  
    // Do something  
}
```

1. 将 “=1” 改为 “=2”?

2. 将 “=1” 删掉?

3. 将这一行整个删掉?

4. 将 “=2” 改为 “=1”?

5. 将 “=2” 删掉?

6. 结论?

4.3 Default parameter values(缺省参数值)

```
#include <iostream>
using namespace std;
int add (int x, int y=10) {
    return x + y;
}
int add (int x) {
    return x + 100;
}
int main() {
    add(1); //How to solve this?
    cin.get();
    return 0;
}
```

error C2668: “add”: 对重载函数的调用不明确
可能是 “int add(int)”
或 “int add(int,int)”

Dev-C++: [Error] call of overloaded 'add(int)' is ambiguous



4.5 Enumeration and structure name

◆ 枚举名与结构名

C++中枚举可以命名，一个枚举名就是一个类型名字，因此在枚举类型名前不必加标识符**enum**；

定义的结构就是一个用户定义的数据类型，在结构名前也不必加标识符**struct**。例如下面定义一个结构类型：

```
struct student{  
    char name[10];  
    int number;  
    int page;  
};
```

则在定义该结构类型变量时可以使用如下方式：

student s1, s2; //不用像C那样在studeng前面写**struct**

4.6 Scope resolution operator(作用域运算符)

◆ 作用域标识符

在C++语言中增加了**作用域运算符**（或称为**名字解析运算符**）`::`，用以解决局部变量名与全局变量的同名重复问题。

在局部变量的作用域内可用作用域标识符`::`对被其隐藏的同名全局变量进行访问。

```
int x=0;  
void test(int x)  
{  
    x=5; //引用局部变量x  
    ::x=9; //引用全局变量x  
}
```

4.7 Varibale definition in program block

◆ 程序块中的变量定义

- 程序块：大括号{ }内的语句构成了一个程序块。
- C89，必须在函数体前面定义。
- C++语言中**可以在任何位置声明变量**。

```
float sum = 0; //C & C++
```

```
float sum2(0); //C++, OO style
```

```
for(int i=0; i<10; i++) //for(int i(0); i<10; i++)  
{  
    sum += static_cast<float>( i )/(i+1);  
    int j;  
    for(j=0;j<i; j++);  
}
```

4.8 Constant (常量)

- ◆ C++增加了常量类型,用标识符**const**声明,其值在作用域内保持不变,ANSI C标准也引进了**const**修饰符。

```
const int maxSize=128;
```

```
const int intArray[]={1,2,3,4,5,6};
```

常量与指针:

```
const int * x
```

```
int * const y
```

const在谁前面谁就不允许改变。

4.8 Constant (常量)

◆ 常量指针/常指针

(A variable pointer to a constant value, Pointer to Constant)

指针所指向的内容不可以通过指针的间接引用(*p)来改变。

```
const int* p1;
```

```
const int x = 1;
```

```
p1 = &x; //指针 p1的类型是 (const int*)
```

```
*p1 = 10; // Error!
```

```
const char *pStr2="Hello, world";
```

指针pStr2所指对象不能被改变, pStr2本身可以被改变。

```
pStr2="Hi, there!"; // correct
```

4.8 Constant (常量)

◆ 指针常量 (Pointer Constant)

指针本身的内容是个常量，不可以改变。

```
int x = 1, y = 1;
```

```
int* const p2 = &x; //常量 p2的类型是 (int*)
```

```
char * const pStr="Hello, world! ";
```

```
*p2 = 10; // Okay!    x=10
```

```
p2 = &y; // Error! p2 is a constant
```

```
pStr = "Hi, there! "; // Error! pStr is a constant
```

数组名: 数组的首地址的别名。本质: 数组名是个
指针常量。

4.9 Reference(引用)

- ◆ 引用的目的是**为了消除指针**，往往用在给函数传递大参数时。
 - 引用顾名思义就是引用这个变量。
 - 引用必须初始化。引用总是指向在初始化时被指定的对象，以后不能改变。
 - 不存在指向空值的引用。所以在使用引用之前不需要测试它的合法性。

```
int a;
```

```
int &b=a;
```

4.9 引用

- ◆ 引用类型的函数参数:函数内部可以修改实参的数值

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

void func(int *pInt)
{
    if( pInt == NULL )
        return;
    *pInt = 2;
}

int main()
{
    int a;
    func(&a);
    cout<<"a= "<<a<<endl;
    return 0;
}
```

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

void func(int &pInt)
{
    pInt = 2;
}

int main()
{
    int a;
    func(a);
    cout<<"a= "<<a<<endl;
    return 0;
}
```

4.9 Reference(引用)

◆ 引用类型的函数参数

函数内部可以修改实参的数值

P7H: `void swap_vals(float& val1, float& val2) ;`

◆ 引用类型的函数参数

减少参数传递的代价（拷贝）

- 一维数组例子P7I:

`int sum_array(const int array[], int no_of_elements) ;`

- 二维数组例子P7K:

`int sum_array(const int array[][2], int no_of_rows) ;`

- 结构体变量例子P7L:

`void get_student_data(struct student_rec& student_ref) ;`

`void display_student_data(struct student_rec student_data) ;`

`void display_student_data(const struct student_rec &student_data) ;`

4.10 动态空间申请——new、delete

◆ 运算符new申请动态内存

- new <类型名> (初值); //申请一个变量的空间
- new <类型名> [常量表达式]; //申请数组

如果申请成功，返回指定类型内存的地址；

如果申请失败，返回空指针(整数0)。

◆ 运算符delete释放。

- delete <指针名>; //删除一个变量/对象
- delete []<指针名>; //删除数组空间

4.10 动态空间申请——new、delete

◆ C的动态空间申请

```
//void *malloc(size_t size);  
//void *realloc( void *memblock, size_t new_size );  
//void *calloc(size_t num, size_t size );Allocates an array  
//                               in memory with elements initialized to 0.
```

```
int *p1 = (int *)malloc(sizeof(int) * length);  
free(p1);
```

◆ C++增加new和delete运算符

```
int *p2 = new int(2); 初始化  
delete p2; // 释放单个用法
```

```
int *p3 = new int[100]; 数组元素个数  
delete []p3; // 释放数组用法
```


4.10 动态空间申请——new、delete

	C	C++
	malloc(); free();	new delete
Example01	char *s = (char*)malloc(1); free(s);	char* s = new char(97); delete s;
Example02	int *p = (int*)malloc(4*10); free(p);	int **p = new int[10][3]; delete []p;
Example03	int **q = (int **) malloc(4*10*3); free(q);	int (*q)[3] = new int[10][3]; delete []p;

4.10 动态空间申请——new、delete

◆ new和delete运算符....

```
int dim = 15;
```

```
char (*pchar)[10] = new char[dim ][10];
```

```
delete [] pchar;
```

```
int (*q) () = NULL; //指向函数的指针
```

```
int (**p) () = NULL; // ?
```

```
p = new (int (*[7]) ());
```

```
q = new ( int() );// error, why?
```

```
delete []p;
```

```
// int (*q) () = NULL;
typedef int(*NoParaFuncPtr)() ;
NoParaFuncPtr *pp;
//p = new (int (*[7]) ());
pp = new NoParaFuncPtr[7];
```

4.10 动态空间申请——new、delete

◆ new 和 malloc 分配内存的区别

- new 是c++中的操作符，malloc是c 中的一个函数
- new 不止是分配内存，而且会调用类的构造函数，同理delete会调用类的析构函数；而malloc则只分配内存，不会进行初始化类成员的工作，同样free也不会调用析构函数
- 内存泄漏对于malloc或者new都可以检查出来的（例：Insure++）

Simplified Memory Model (C++的内存模型)

注意:

这个简化模型仅用于初学者示意
与实际模型并不完全一致

1. Stack (栈)

编译器自动分配释放

2. Heap (堆)

一般由程序员分配释放，若程序员不释放，程序结束时可能由OS回收

3. Global/Static (全局区/静态区)

全局变量和静态变量的存储是放在一块的。

可以简单认为：

程序启动全局/静态变量就在此处

程序结束释放。

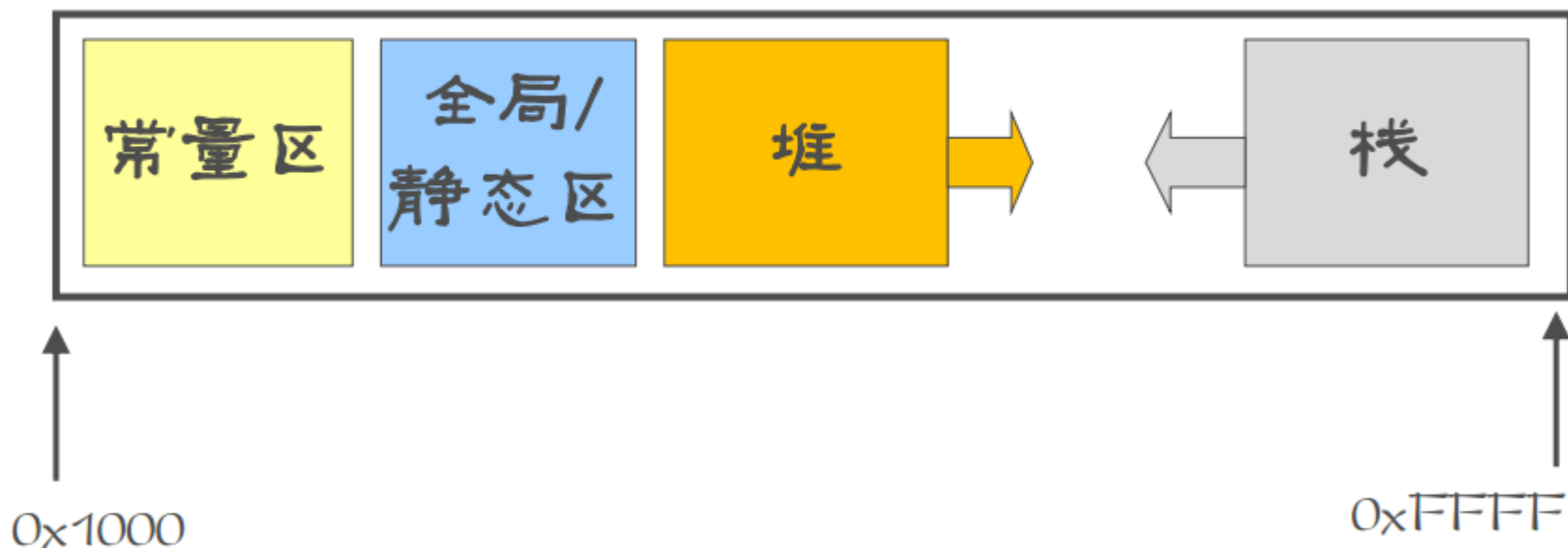
4. Constant (常量区)

可以简单理解为所有常量都放在一起
该区域内容不可修改

Example of memory allocation (C++程序的内存示例)

❖ 堆向高地址方向生长

❖ 栈向低地址方向生长



注意:

这个示例模型仅用于初学者示意
与实际并不完全一致

Location of a variable (变量存放位置)

```
int arr[3];  
int myFunc()  
{
```

```
    int a;  
    char *p;  
    char* str="hello world"  
}
```



Location of a variable (变量存放位置) – 续

```
int myFunc1(int* pi)
{
    char *pc;
    pc= static_cast<char*> new char[8];
}
```



4.11 编程技巧

- ◆ 1. 不要混淆流插入运算符 (insertion operator) << 和流提取(读取)运算符(extraction operator) >>.
- ◆ 2. 有些流操纵符(manipulator)仅对随后的数据项有效, 例如setw;有些流操纵符将对其后的所有输出项一直有效。
- ◆ 3. 如果使用了带参数的流操纵符, 需要包含头文件:

`#include <iomanip>`

- ◆ 4. **避免使用下划线和双下划线开头的标识符**, C++编译器内部使用这类名称,从而, 防止与编译器选择的名称相同。
- ◆ 不用float或double表示货币, 浮点数不是精确表示, 可能导致错误, 产生不准确的货币值。
- ◆ 条件表达式 `x == 7` 不如 `7 == x` (why?)
- ◆ inline 只用于经常使用的小函数
- ◆ **将字符作为字符串参数的值参, 可能导致严重错误**

Q&A...



Thank You!

Thank You!