程序设计与算法(一)

C语言程序设计

郭 炜

微信公众号

微博：http://weibo.com/guoweiofpku

学会程序和算法，走遍天下都不怕!

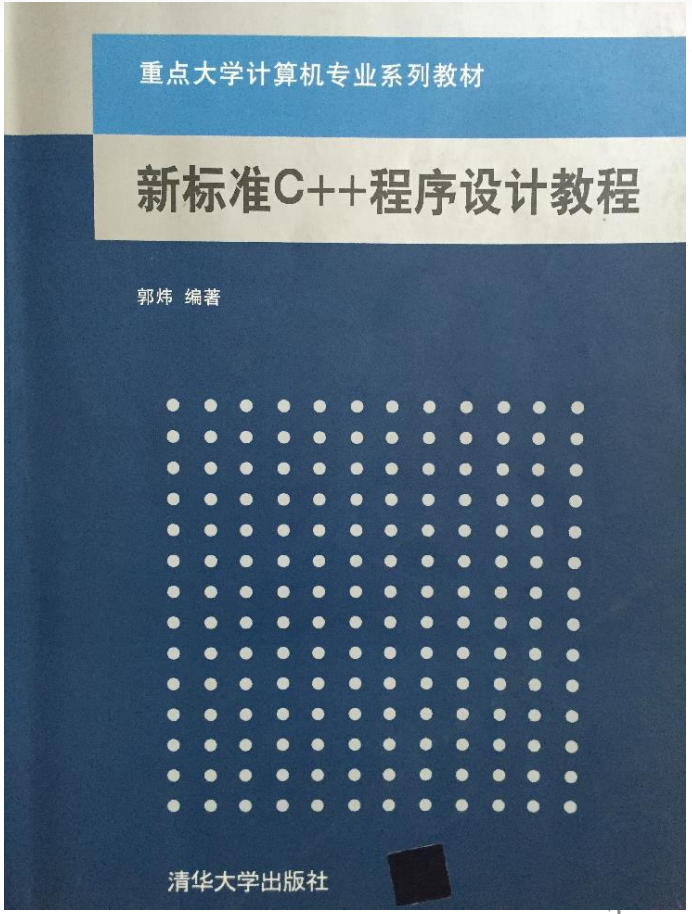
讲义照片均为郭炜拍摄

指定教材：

《新标准C++程序设计教程》

郭炜 编著

清华大学出版社



结构(struct)

结构的概念

法国戛纳海滨

现实需求

●在现实问题中，常常需要用一组不同类型的数据来描述一个事物。比如一个学生的学号、姓名和绩点。一个工人的姓名、性别、年龄、工资、电话……

●如果编程时要用多个不同类型的变量来描述一个事物，就很麻烦。当然希望只用一个变量就能代表一个“学生”这样的事物。

●C++允许程序员自己定义新的数据类型。因此针对“学生”这种事物，可以定义一种新名为Student的数据类型，一个Student类型的变量就能描述一个学生的全部信息。同理，还可以定义数据类型 Worker以表示工人。

结构(struct)

- 用“struct”关键字来定义一个“结构”，也就定义了一个新的数据类型：

```
struct 结构名
{
        类型名  成员变量名；
        类型名  成员变量名；
        类型名  成员变量名；
        ……
};
```

结构(struct)

- 例：

```
struct Student {
        unsigned ID;
        char szName[20];
        float fGPA;
};
```

- **Student 即成为自定义类型的名字，可以用来定义变量**

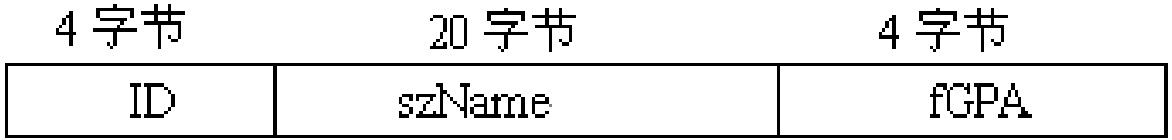
```
  Stuent s1,s2;
```

结构(struct)

- 例两个同类型的结构变量，可以互相赋值。但是结构变量之间不能用“==”、“!=”、“<”、“>”、“<=”、“>=”进行比较运算。

- 一般来说，一个结构变量所占的内存空间的大小，就是结构中所有成员变量大小之和。结构变量中的各个成员变量在内存中一般是连续存放的，

```
struct Student {
        unsigned ID;
        char szName[20];
        float fGPA;
};
```



结构(struct)

- 一个结构的成员变量可以是任何类型的，包括可以是另一个结构类型:

```
struct Date {
        int year;
        int month;
        int day;
};
struct StudentEx {
        unsigned ID;
        char szName[20];
        float fGPA;
        Date birthday;
};
```

结构(struct)

- 结构的成员变量可以是指向本结构类型的变量的指针

```
struct Employee {
        string name;
        int age;
        int salary;
        Employee * next;
};
```

访问结构变量的成员变量

- 一个结构变量的成员变量，可以完全和一个普通变量一样来使用，也可以取得其地址。使用形式：

结构变量名.成员变量名

```
struct Date {
        int year;
        int month;
        int day;
};
struct StudentEx {
        unsigned ID;
        char szName[20];
        float fGPA;
        Date birthday;
};
```

```
StudentEx stu;
cin >> stu.fGPA;
stu.ID = 12345;
strcpy(stu.szName, "Tom");
cout << stu.fGPA;
stu.birthday.year = 1984;
unsigned int * p = & stu.ID;   //p指向stu中的ID成员变量
```

结构变量的初始化

- 结构变量可以在定义时进行初始化:

```
StudentEx stu = { 1234,"Tom",3.78,{ 1984,12,28 }};
```

```
struct Date {
        int year;
        int month;
        int day;
};
struct StudentEx {
        unsigned ID;
        char szName[20];
        float fGPA;
        Date birthday;
};
```

结构数组和指针

摩纳哥

结构数组

```
StudentEx MyClass [50];

StudentEx MyClass2[50] = {
        { 1234,"Tom",3.78,{ 1984,12,28 }},
        { 1235,"Jack",3.25,{ 1985,12,23 }},
        { 1236,"Mary",4.00,{ 1984,12,21 }},
        { 1237,"Jone",2.78,{ 1985,2,28 }}
};

MyClass[1].ID = 1267;
MyClass[2].birthday.year = 1986;
int n = MyClass[2].birthday.month;
cin >> MyClass[0].szName;
```

指向结构变量的指针

- 定义指向结构变量的指针

结构名 * 指针变量名;

```
StudentEx * pStudent;
StudentEx Stu1;
pStudent = & Stu1;
StudentEx Stu2 = * pStudent;
```

指向结构变量的指针

- 通过指针，访问其指向的结构变量的成员变量：

  指针->成员变量名

  或：

  (* 指针).成员变量名

```
StudentEx Stu;
StudentEx * pStu;
pStu = & Stu;
pStu->ID = 12345;
(*pStu).fGPA = 3.48;
cout << Stu.ID << endl;     //输出 12345
cout << Stu.fGPA << endl;   //输出 3.48
```

C++程序结构

全局变量
局部变量
静态变量

法国圣十字湖

全局变量和局部变量

- 定义在函数内部的变量叫局部变量（函数的形参也是局部变量）
- 定义在所有函数的外面的变量叫全局变量
- 全局变量在所有函数中均可以使用，局部变量只能在定义它的函数内部使用

全局变量和局部变量

```
#include <iostream>
using namespace std;
int n1 = 5, n2 = 10;   //全局变量
void Function1()
{
        int n3 =4;
        n2 = 3;
}
void Function2()
{
        int n4;
        n1 = 4;
        n3 = 5;    //编译出错，n3无定义
}
```

静态变量

●全局变量都是静态变量。局部变量定义时如果前面加了“static”关键字，则该变量也成为静态变量

●静态变量的存放地址，在整个程序运行期间，都是固定不变的

●非静态变量(一定是局部变量)地址每次函数调用时都可能不同,在函数的一次执行期间不变

●如果未明确初始化，则静态变量会被自动初始化成全0(每个bit都是0），局部非静态变量的值则随机

静态变量示例

```
#include <iostream>
using namespace std;
void Func()
{
        static int n = 4;   //静态变量只初始化一次
        cout << n << endl;
        ++ n;
}
int main()
{
        Func(); Func(); Func();
}
```

输出结果：

4
5
6

静态变量应用:strtok的实现

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
        char str[] ="- This, a sample string, OK.";
        //下面要从str逐个抽取出被" ,.-"这几个字符分隔的字串
        char * p = strtok (str," ,.-");
        while ( p != NULL) //只要p不为NULL，就说明找到了一个子串
        {
                cout << p << endl;
                p = strtok(NULL, " ,.-");
                //后续调用，第一个参数必须是NULL
        }
        return 0;
}
```

```
This
a
sample
string
OK
```

```
char * p = strtok (str," ,.-");
while ( p != NULL) //只要p不为NULL，就说明找到了一个子串
{
        cout << p << endl;
        p = strtok(NULL, " ,.-");
        //后续调用，第一个参数必须是NULL
}
```

```
This
a
sample
string
OK
```

-		T	h	i	s	,		a		s	a	m	p	l	e		s	t	r	i	n	g	,		O	K	\0

静态变量应用:strtok的实现

```
char * Strtok(char * p,char * sep)
{
        static char * start ; //本次查找子串的起点
        if(p)
                start = p;
        for(; *start && strchr(sep,*start); ++ start); //跳过分隔符号
        if( * start == 0)
                return NULL;
        char * q = start;
        for(; *start && !strchr(sep,*start); ++ start); //跳过非分隔符号
        if( * start) {
                * start = 0;
                ++ start;
        }
        return q;
}
```

标识符作用域
变量生存期

法国普罗旺斯

标识符的作用域

●变量名、函数名、类型名统称为“标识符”。一个标识符能够起作用的范围，叫做该标识符的作用域

●在一个标识符的作用域之外使用该标识符，会导致“标识符没有定义”的编译错误。使用标识符的语句，必须出现在它们的声明或定义之后

●在单文件的程序中，结构、函数和全局变量的作用域是其定义所在的整个文件

标识符的作用域

- 函数形参的作用域是整个函数
- 局部变量的作用域，是从定义它的语句开始，到包含它的最内层的那一对大括号“{}”的右大括号 “}”为止
- for循环里定义的循环控制变量，其作用域就是整个for循环
- 同名标示符的作用域，可能一个被另一个包含。则在小的作用域里，作用域大的那个标识符被屏蔽，不起作用。

标识符的作用域

```
void Func(int m)
{
        for( int i = 0; i < 4;++i ) {
                if( m <= 0 ) {
                        int k = 3;
                        m = m *( k ++ );
                }
                else {
                        k = 0; //编译出错，k无定义
                        int m = 4;
                        cout << m;
                }
        }
        i= 2; //编译出错，i无定义
}
```

变量的生存期

●所谓变量的“生存期”，指的是在此期间，变量占有内存空间，其占有的内存空间只能归它使用，不会被用来存放别的东西。

●而变量的生存期终止，就意味着该变量不再占有内存空间，它原来占有的内存空间，随时可能被派做他用。

●全局变量的生存期，从程序被装入内存开始，到整个程序结束。

变量的生存期

●静态局部变量的生存期，从定义它语句第一次被执行开始，到整个程序结束为止。

●函数形参的生存期从函数执行开始，到函数返回时结束。非静态局部变量的生存期，从执行到定义它的语句开始，一旦程序执行到了它的作用域之外，其生存期即告终止。

变量的生存期

```
void Func(int m)
{
        for( int i = 0; i < 4;++i ) {
                if( m <= 0 ) {
                        int k = 3;
                        m = m *( k ++ );
                }
                else {
                        k = 0; //编译出错，k无定义
                        int m = 4;
                        cout << m;
                }
        }
        i= 2; //编译出错，i无定义
}
```

简单排序

排序问题

- 例题: 编程接收键盘输入的若干个整数，排序后从小到大输出。先输入一个整数n，表明有n个整数需要排序，接下来再输入待排序的n个整数。

解题思路：先将n个整数输入到一个数组中，然后对该数组进行排序，最后遍历整个数组，逐个输出其元素。

对数组排序有很多种简单方法，如“冒泡排序”、“选择排序”、“插入排序”等

选择排序

法国尼斯英国人大道

选择排序

如果有N个元素需要排序，那么首先从N个元素中找到最小的那个(称为第0小的)放在第0个位子上(和原来的第0个位子上的元素交换位置)，然后再从剩下的N-1个元素中找到最小的放在第1个位子上，然后再从剩下的N-2个元素中找到最小的放在第2个位子上……直到所有的元素都就位。

选择排序

```
void SelectionSort(int a[] ,int size)
{
        for( int i = 0; i < size - 1; ++i ){//每次循环后将第i小的元素放好

                int tmpMin = i;
                //用来记录从第i个到第size-1个元素中，最小的那个元素的下标
                for( int j = i+1; j < size ; ++j) {
                        if( a[j] < a[tmpMin] )
                                tmpMin = j;
                }
//下面将第i小的元素放在第i个位子上，并将原来占着第i个位子的元素挪到后面
                int tmp = a[i];
                a[i] = a[tmpMin];
                a[tmpMin] = tmp;
        }
}
```

插入排序

俯瞰巴黎

插入排序

- 将整个数组a分为有序的部分和无序的两个部分。前者在左边，后者在右边。
- 开始有序的部分只有a[0]，其余都属于无序的部分
- 每次取出无序部分的第一个（最左边）元素，把它加入到有序部分。假设插入到合适位置p, 则原p位置及其后面的有序部分元素，都向右移动一个位子。有序的部分即增加了一个元素。
- 直到无序的部分没有元素

插入排序

```
void InsertionSort(int a[] ,int size)
{
        for(int i = 1;i < size; ++i )  {
                //a[i]是最左的无序元素，每次循环将a[i]放到合适位置
                for(int j = 0; j < i; ++j)
                        if( a[j]>a[i]) {
                //要把a[i]放到位置j，原下标j到 i-1的元素都往后移一个位子
                                int tmp = a[i];
                                for(int k = i; k > j; --k)
                                        a[k] = a[k-1];
                                a[j] = tmp;
                                break;
                        }
        }
}
```

冒泡排序

埃菲尔铁塔

冒泡排序

- 将整个数组a分为有序的部分和无序的两个部分。前者在右，后者在左边。

- 开始，整个数组都是无序的。有序的部分没有元素。

- 每次要使得无序部分最大的元素移动到有序部分第一个元素的左边。移动的方法是：依次比较相邻的两个元素，如果前面的比后面的大，就交换他们的位置。这样，大的元素就像水里气泡一样不断往上浮。移动结束有序部分增加了一个元素。

- 直到无序的部分没有元素

冒泡排序

```
void BubbleSort(int a[] ,int size)
{
        for(int i = size-1;i > 0; --i )   {
        //每次要将未排序部分的最大值移动到下标i的位置
                for(int j = 0; j < i; ++j) //依次比较相邻的两个元素
                        if( a[j] > a[j+1]) {
                                int tmp = a[j];
                                a[j] = a[j+1];
                                a[j+1] = tmp;
                        }
        }
}
```

简单排序的效率

- 上面3种简单排序算法，都要做 n^2量级次数的比较(n是元素个数）！

- 好的排序算法，如快速排序，归并排序等,只需要做$n*\log_2 n$量级次数的比较!