

# 第10章 管理内存和底层数据结构

刘 卉

huiliu@fudan.edu.cn



# 前言

## 如何实现标准库提供的类型？

- 使用语言核心的编程工具和技巧.
- 底层：标准库的基础，与计算机硬件紧密相关.
- 使用起来较难&危险，但更高效.
- 标准库并不能解决所有的问题.

## 本章内容

---

### □ 动态分配内存

- 数组 & 指针, 结合new & delete
- 程序员可直接控制内存分配(标准库的类没有该能力)

### □ 不遵循"提出问题-解决方案"模式.

- 所介绍的工具均在底层工作, 很难使用其中一个解决有意义的问题.

### □ 在此基础上, 第11章将介绍标准库如何利用这些底层工具实现容器.



# 10.1 指针和数组

C和C++最原始的数据结构之一

数组也是一种容器

- 类似于vector，但没有vector强大。

指针是一种随机访问迭代器

- 最重要的功能：访问数组元素。



## 10.1.1 指针

### □ 指针和数组不可分割

- 单纯使用数组不能有效解决问题。
- 数组使指针更有用。

`int *p; int* p;` //两种用法等价

`int* p, q;` //与`int *p, q`等价

`int* p; int q;` //这种写法更好

- 单个对象的指针：指向容器唯一元素的迭代器。



## 10.1.2 指向函数的指针

### □ 函数不是一个变量/对象

- 不能被复制(拷贝)、赋值，或直接作为参数传递.
- 程序所能做的：1) 调用函数；2) 取函数的地址.
- 任何使用函数的地方，如果不是在调用它，就是在取它的地址.

## 指向函数的指针

- 任何使用函数的地方，如果不是在调用它，就是在取它的地址。

```
int next(int n)
```

```
{ return n+1; }
```

```
int (*fp)(int); // fp能指向带一个int形参、返回int值的函数
```

```
fp = next;      // 与fp = &next等价
```

```
i = fp(i);      // 与i = (*fp)(i)等价
```

### □ 返回函数指针的函数

函数指针类型

```
typedef double (*analysis_fp)(const vector<Student_info>&);
```

```
//get_analysis_ptr returns a pointer to an analysis function
```

```
analysis_fp get_analysis_ptr();
```

## 函数指针作参数

```
template<class In, class Pred> bool is_negative(int n)
In find_if(In begin, In end, Pred f) {
{ return n < 0;
    while (begin != end && !f(*begin)) }
        ++begin;
    return begin;
}

vector<int> v;
vector<int>::iterator i = find_if(v.begin(), v.end(), is_negative);
```





## 10.1.3 数组

### □ 数组不是类

- 数组不能像标准库的容器那样，动态地增长/缩小。
- 数组没有size\_type成员，用<cstddef>定义的size\_t类型保存数组的长度。

```
const size_t NDim = 3; // 优于const int NDim = 3
```

```
double coords[NDim]; // 优于double coords[3]
```



## 10.1.4 指针的算术运算

[例] 把数组coords的内容复制到vector对象中

方法1. `vector<double> v;`

```
copy(coords, coords+NDim, back_inserter(v));
```

方法2. `vector<double> v(coords, coords + NDim);`

- `coords+Ndim`并不指向数组元素，而是一个有效的越界迭代器。
- 在包含n个元素的数组a上使用标准库算法：使用a和a+n作迭代器参数，e.g. `sort(a, a+n);`

- 如果a是一个包含n个元素的数组：  
当且仅当 $0 \leq i \leq n$ ，a+i有效；  
当且仅当 $0 \leq i < n$ ，a+i指向a的一个元素。
- 若p和q指向同一数组元素，则p-q是一个整数，其类型为ptrdiff\_t.

e.g. p = coords, q = &coords[2];

ptrdiff\_t dis = p-q; // 优于int dis = p-q



# 数组初始化

```
const int month_lengths[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

⚠ 标准库容器不支持这种初始化方式

```
vector<int> mon = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}; // VS2017支持容器以数组的方式初始化, Dev-C++ 5.11不支持
```



## 10.2 再看字符串常量

### □ const char数组

```
const char hello[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

- 与字符串常量"Hello"意义相同.
- 三种方式构造一个string对象s:

```
string s("Hello"); //字符串常量↔字符指针
```

```
string s(hello); //保存字符串的字符数组
```

```
string s(hello, hello + strlen(hello)); //迭代器区间
```



## 10.3 初始化字符指针数组

### □ 数字表示的成绩→字母表示的成绩

If the grade is at least    97 94 90 87 84 80 77 74 70 60 0  
then the letter grade is    A+ A    A- B+    B B- C+ C    C- D    F

只初始化一次

元素的类型<sup>①</sup>

数组letters不能被改变<sup>②</sup>

```
static const char* const letters[] = { "A+", "A", "A-", "B+",  
"B", "B-", "C+", "C", "C-", "D", "F" };
```

```
letters[0] = "a+";    // 错误. ② 限定
```

```
letters[0][0] = 'a'; // 错误. ① 限定
```



## 10.5 读写文件

### □ 标准错误流cerr/clog

- 输出对程序的注释：提醒用户出现错误/建立日志文件.
- 为了区分注释与普通输出，C++库定义了标准错误流.
- clog
- cerr

- 用在日志上→与cout具有相同的缓冲性质.
- 对程序的运行进行连续注释.

- 总是立即输出→可能会带来一些开销.
- 输出一个紧急错误.

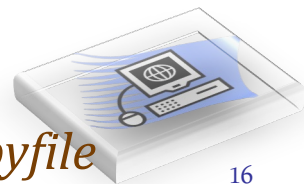


# 读写磁盘文件

## □ ifstream, ofstream——<fstream>

//copies a file named *in* to a file named *out*

```
int main() {  
    //定义用于输入的文件对象infile, 并将其与磁盘文件in(已存在)绑定  
    ifstream infile("in");  
    //定义用于输出的文件对象outfile, 并将其与磁盘文件out绑定  
    ofstream outfile("out");  
    string s;  
    while (getline(infile, s)) //why not infile >> s?  
        outfile << s << endl;  
    return 0;  
}
```





## □ 文件名——必须是字符串指针

```
string file("in"); // file: string对象
```

```
ifstream infile(file.c_str()); //将file转换为字符串
```



## 10.6 三种内存管理

### 1. 自动内存管理

- 与局部变量相关，系统自动分配与释放。
- 避免使用无效指针。

```
// this function deliberately yields an invalid pointer.  
// it is intended as a negative example—don't do this!
```

```
int* invalid_pointer()
```

```
{
```

```
    int x;
```

```
    return &x; // instant disaster!
```

```
}
```

C++并不诊断该错误，仍会得到返回值，但值不确定。



## 2. 静态分配内存

```
// This function is completely legitimate.
```

```
int* pointer_to_static()
```

```
{
```

```
    static int x;
```

```
    return &x;
```

```
}
```

- 函数被调用之前，系统已分配x的内存，且只分配一次。
- 只要程序运行，就不释放该静态变量的内存。
- 潜在缺陷：每次调用都返回同一个对象的指针。



# 3. 动态分配内存

---

- I. 为单个对象分配和释放内存
- II. 为数组分配并释放内存



## 10.6.1 为单个对象分配和释放内存

[例1] `int* p = new int(42);` // 申请一块空间(由p指向), 存放42

`++*p;` // \*p is now 43

`delete p;` // p所指的空间被释放

[例2] `int* pointer_to_dynamic()`

`{ return new int(0); }` // 申请一块空间, 初始化为0, 返回指针



调用该函数的程序, 应在适当的时候释放该对象.



## 10.6.2 为数组分配并释放内存

- `new T[n]`: 数组的每个元素都被默认初始化
  - `T`是内置类型, 且在局部生存空间分配→不会被初始化;
  - `T`是一个类→每个元素通过类的默认构造函数初始化.
    - 如果该类不允许默认初始化, 编译器将终止该程序.
  - 每个元素被初始化会带来一定开销→标准库提供了一种更灵活的机制来动态分配数组.

`new T[0]`

- 允许为一个不包含任何元素的数组分配空间，返回一个越界指针。
- 这个特殊行为，使得如下代码在n为0时也能运行：

```
T* p = new T[n];  
vector<T> v(p, p+n);  
delete[] p;
```

- p和p+n都是指针，可以比较。
- []是必须的，告诉系统释放整个数组。

## [例] 复制字符串

```
char* duplicate_chars(const char* p)
{
    // allocate enough space; remember to add one for the null
    size_t length = strlen(p) + 1;
    char* result = new char[length];
    // copy into our newly allocated space and return pointer to
    // first element
    copy(p, p+length, result);
    return result;
}
```





# 小结

- 指针是随机访问迭代器
- 数组是大小固定的内置容器
  - 它的迭代器是指针.
  - 数组索引: 按照指针的操作定义.
- 函数指针

```
vector<string> (*sp)(const string&) = split;
```

- 定义sp是一个函数指针, 指向split函数.

## □ 输入-输出

- cerr: 标准错误流, 输出不进入缓冲.
- clog: 用于日志的标准错误, 输出进入缓冲.
- ifstream(cp): 绑定在char\* cp所命名文件上的输入流, 支持istream操作.
- ofstream(cp)
- <fstream>

## □ 内存管理

- `new T, new T(args), delete p`
- `new T[n], delete[] p`