目录

[备忘栏： 1](#_Toc504407506)

[数组 2](#_Toc504407507)

[1、数组大小(sizeof) 2](#_Toc504407508)

[2、数组定义 2](#_Toc504407509)

[3、数组函数传参 2](#_Toc504407510)

[字符串 2](#_Toc504407511)

[定义字符数组-字符串： 2](#_Toc504407512)

[打印字符串的3个方法： 3](#_Toc504407513)

[链表 3](#_Toc504407514)

[1、定义结构体： 3](#_Toc504407515)

[2、常用链表操作： 3](#_Toc504407516)

[（1）创建节点 3](#_Toc504407517)

[（2）连接2个节点 3](#_Toc504407518)

[（3）打印链表 4](#_Toc504407519)

[（4）析构链表（删除所有节点） 4](#_Toc504407520)

[（5）在链表末尾添加一个节点（类似push\_back(value)） 4](#_Toc504407521)

[（6）在链表中找到值为value的节点并删除(类似remove(value)) 5](#_Toc504407522)

[函数相关 5](#_Toc504407523)

[1、引用： 5](#_Toc504407524)

[2、函数指针 6](#_Toc504407525)

[STL - Vector 7](#_Toc504407526)

[1、创建（可初始化） 7](#_Toc504407527)

[2、赋值 7](#_Toc504407528)

[3、访问及遍历 7](#_Toc504407529)

[4、常用函数 8](#_Toc504407530)

[5、用已有数组初始化 8](#_Toc504407531)

[STL - Stack 9](#_Toc504407532)

[定义： 9](#_Toc504407533)

[常用操作： 9](#_Toc504407534)

[头文件（调用外部函数） 9](#_Toc504407535)

# 备忘栏：

链表；STL-stack；字符串

# 数组

## 1、数组大小(sizeof)

int a[] = {1,2,3,4,5};

int size1 = sizeof(a); //20 （数组总共占空间为20）

int size2 = sizeof(a[0]); //4 （数组中每个元素占空间为4）

int size3 = size1 / size2; //5 （数组长度为5——可以用此方法求静态数组长度）

printf("%d, %d, %d\n", size1, size2, size3);

int\* b = a;

int size4 = sizeof(b); //4（32位系统上对任意指针求sizeof结果都是4！我电脑也是配置: Debug Win32）

## 2、数组定义

1. 定义静态数组（长度需指定为数值或常量）

int a[] = {1,2,3,4,5};

int b[10];

int c[100] = {0}; //所有元素初始化为0

2. 定义动态数组（长度可以是一个变量）

int\* a = new int[length]; //没初始化

int\* b = new int[length](); //可**初始化每个元素为0**

bool\* c = new bool[length](); //可**初始化每个元素为false**

delete[] b;//最好记得动态释放（否则new多了空间可能会崩溃）

## 3、数组函数传参

数组传入函数后，其形式会变成一个指针，无法用sizeof判断其大小，故需要同时传入数组长度。

void f(int a[], int length){

cout << sizeof(a) << endl; //4 (即指针大小)

}

int a[] = {1, 2, 3}

int length = sizeof(a)/ sizeof(int); //3 (数组长度)

f(a, lenght);

# 字符串

### 定义字符数组-字符串：

char str[] = "i am chinese!";

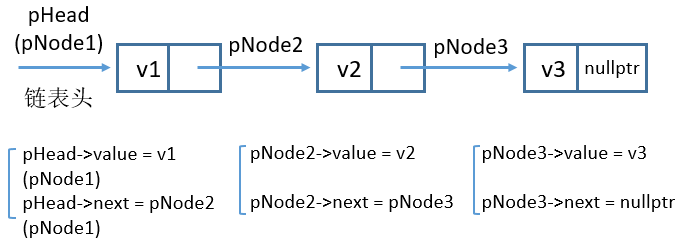
### 打印字符串的3个方法：

cout << str <<endl;

printf("%s\n", str);

puts(str); //自带回车

# 链表



注：空链表的pHead为nullptr；若链表不为空，则pHead = pNode1.

## 1、定义结构体：

struct ListNode {

int val;

ListNode \*next;

};

## 2、常用链表操作：

### （1）创建节点

ListNode\* CreateListNode(int value) {

ListNode\* pNode = new ListNode();

pNode->val = value;

pNode->next = nullptr; //默认next为空

return pNode;

}

### （2）连接2个节点

void ConnectListNodes(ListNode\* pCurrent, ListNode\* pNext) {

if (pCurrent == nullptr) { //如果当前节点指针为空，则报错退出

printf("Error to connect 2 nodes.\n");

}

else {

pCurrent->next = pNext;

}

}

### （3）打印链表

void PrintList(ListNode\* pHead) {

//printf("PrintList starts.\n");

ListNode\* pNode = pHead;

while (pNode != nullptr) {

cout << pNode->val << " ";

pNode = pNode->next;

}

cout << endl;

//printf("PrintList ends.\n");

}

### （4）析构链表（删除所有节点）

void DestroyList(ListNode\* pHead) {

ListNode\* pNode = pHead;

ListNode\* pNext;

while (pNode != nullptr) {

pNext = pNode->next;

delete pNode;

pNode = pNext;

}

}

### （5）在链表末尾添加一个节点（类似push\_back(value)）

void PushBackNode(ListNode\*\* ppHead, int value) {

//头指针可能改变（如空链表时）！所以传入指针的指针！

//新建节点

ListNode\* pNew = new ListNode();

pNew->val = value;

pNew->next = nullptr;

//判断是否为空链表(表头为空)

if (\*ppHead == nullptr) {

\*ppHead = pNew; //若为空，表头指向新节点

}

else {

ListNode\* pNode = \*ppHead;

while (**pNode->next** != nullptr) { **//注：两个红色部分要统一！**

pNode = pNode->next;

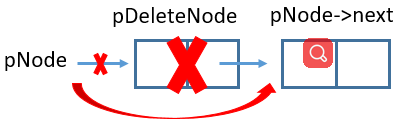
}

**pNode->next** = pNew; //让末尾节点指向新节点

}

}

### （6）在链表中找到值为value的节点并删除(类似remove(value))



void RemoveNode(ListNode\*\* ppHead, int value) {

//空链表（表头为空）

if (ppHead == nullptr || \*ppHead == nullptr) {

cout << "The List is empty.\n";

return;

}

//表头指向的第一个节点就是要删除的点（需改变表头）

ListNode\* pDeleteNode = nullptr;

if ((\*ppHead)->val == value) {

pDeleteNode = \*ppHead;

\*ppHead = (\*ppHead)->next;

}

else {

ListNode\* pNode = \*ppHead;

while (pNode->val != value && pNode != nullptr) {

pNode = pNode->next;

}

if (pNode->val == value) {

pDeleteNode = pNode;

pNode = pNode->next;

}

}

//删除节点

if (pDeleteNode != nullptr) {

delete pDeleteNode;

pDeleteNode = nullptr; //记得删完令为空指针！

}

}

# 函数相关

## 1、引用：

函数三种传参形式：值传递，指针（地址）传递，引用传递。

1. 定义：

引用就是给变量起一个别名，对引用的操作与对变量直接操作完全一样。

（和指针有点像，但是指针本身也是一个对象，而引用不是。）

int a = 1;

**int& b = a; //定义引用（初始化）（之后不可更改）**

//a=1, b=1

b = 3; // a=3, b=3

a = 5; // a=5, b=5 （引用后，a,b的值始终保持一致！）

int& c = b; //一个变量可以有多个引用

c = 7; // a=7, b=7, c=7

1. 函数传参：

（1）传常量引用——用于传递不需要修改的大对象时；

void f(**const** vector<double>& v){…}

（2）传引用——用于修改容器或需改变多个对象时。

void swap(int& a, int& b) { //引用传参

int c = a;

a = b;

b = c;

}

void test2() {

int a = 1;

int b = 2;

printf("Original: a = %d, b = %d\n", a, b); //a=1, b=2

swap(a, b);

printf("After swap: a = %d, b = %d\n", a, b); //a=2, b=1(用了引用后，swap中变化，这里也会变化！)

}

注：常量引用可以传一个**值**进去，但引用必须传**变量**进去！

## 2、函数指针

函数指针——指向函数的指针。定义时需要声明的内容：函数类型，参数个数及类型。

用途：有几个类似的函数（函数类型、参数个数及类型）需要在不同情况下调用不同的函数，且可能多次调用。

bool add(int a, int b, int\* c) {

…

}

bool sub(int a, int b, int\* c) {

…

}

void test1(bool(\*func)(int a, int b, int\* c)) {

int a = 10;

int b = 6;

int c = 0;

func(a, b, &c);

}

void test2(bool(\*func)(int a, int b, int\* c)) {

int a = 22;

int b = 2;

int c = 0;

func(a, b, &c);

}

bool(\*func)(int, int, int\* ); //定义函数指针func，它所指向的函数带有3个参数并且返回bool

if (type == 'add') func = add;

if (type == 'sub') func = sub;

test1(func);

test2(func);

# STL - Vector

头文件：

#include<vector>

## 1、创建（可初始化）

vector<int> v;

vector<double> v(10); //定义大小为10（并初始化为0，若此时用push\_back则会放在v[10]）

vector<double> v(10, 2.5); //初始化每个数为2.5

vector<int> v2(v1); //拷贝创建

## 2、赋值

v.push\_back(3); //在末尾放入3

v.insert(v.end(), 10); //在末尾插入10（需指定迭代器）

v[i] = 5; //把第i位赋值为5

## 3、访问及遍历

**（1）数组方式访问**

可以用v[i]直接访问vector的第i个元素（和数组类似）。

故可以直接用循环遍历：

for (int i = 0; i < v.size(); i++) {

cout << v[i] << endl;

}

**（2）迭代器访问**

vector<int>::iterator i;

for (i = v.begin(); i != v.end(); i++) {

cout << \*i << endl;

}

首尾迭代器：

v.begin() //第一个元素的“地址”（类型为迭代器）

v.end() //最后一个元素的“地址”的后一个(其实为结束符)！（类型为迭代器）

## 4、常用函数

v.size() //返回vector大小（元素个数）

v.empyt() //判断vector是否为空，若为空返回1(true)，不空返回0(false)

v.clear() //清空整个vector

v.front() //返回vector的第一个元素值(要求vector不为空)

v.back() //返回vector的最后一个元素值(要求vector不为空)

v.push\_back(a); //在末尾添加一个新元素a

if (!v.empty()) v.pop\_back(); //删除末尾元素 注：vector为空时不能用pop！

v.erase(v.begin()+1) //删除第2个的元素（需指定迭代器）

## 5、用已有数组初始化

已知数组及其长度：

int a[5] = { 3, 2, 5, 7, 8 };

int length = sizeof(a)/sizeof(int);

（1）方法1：逐一加入

vector<int> v1; //若直接再这里定义(length)，则会都初始化为0，push\_back会在已有的0后继续往后加！(0 0 0 0 0 3 2 5 7 8)

v1.reserve(length); //不写这句也行,但写了可以减少内存分配次数

for (int i = 0; i < length; i++) {

v1.push\_back(a[i]);

//v[i] = a[i]; //两种写法选一个即可

}

copy(v1.begin(), v1.end(), ostream\_iterator<int>(cout, " ")); //打印vector！！

cout << endl;

（2）方法2【最方便】：定义时直接带入已知数组的头、尾指针

vector<int> v2(&a[0], &a[length]);

copy(v2.begin(), v2.end(), ostream\_iterator<int>(cout, " "));

cout << endl;

（3）方法3：定义后再insert已知数组的头、尾指针

vector<int> v3; //若直接再这里定义(length)，则会都初始化为0，insert后会有一串0(3 2 5 7 8 0 0 0 0 0)

v3.reserve(length); //不写这句也行,但写了可以减少内存分配次数

v3.insert(v3.begin(), &a[0], &a[length]);

copy(v3.begin(), v3.end(), ostream\_iterator<int>(cout, " "));

cout << endl;

（4）二维vector定义及初始化

//二维vector

//1 2 8 9

//2 4 9 12

//4 7 10 13

//6 8 11 15

int n = 4;

int m = 4;

int a4[4][4] = { { 1, 2, 8, 9 },{ 2, 4, 9, 12 },{ 4, 7, 10, 13 },{ 6, 8, 11, 15 } };

vector<vector<int>> v4(n, vector<int>(m)); //定义二维vector！

for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) {

v4[i][j] = a4[i][j];

}

copy(v4[i].begin(), v4[i].end(), ostream\_iterator<int>(cout, " "));

cout << endl;

}

# STL - Stack

头文件：

#include<stack>

### 定义：

stack<int> s;

### 常用操作：

s.empty() //判断是否为空（空为true，非空为false）

s.top() //返回栈中顶端元素

s.push(3); //把3压进栈（放在顶端）

s.pop(); //弹出栈中顶端元素

# STL - Set

set的各成员函数列表如下:

1. begin()--返回指向第一个元素的迭代器

2. clear()--清除所有元素

3. count()--返回某个值元素的个数

4. empty()--如果集合为空，返回true

5. end()--返回指向最后一个元素的迭代器

6. equal\_range()--返回集合中与给定值相等的上下限的两个迭代器

7. erase()--删除集合中的元素

8. find()--返回一个指向被查找到元素的迭代器（**若set中没有，则返回s.end()**）

9. get\_allocator()--返回集合的分配器

10. insert()--在集合中插入元素

11. lower\_bound()--返回指向大于（或等于）某值的第一个元素的迭代器

12. key\_comp()--返回一个用于元素间值比较的函数

13. max\_size()--返回集合能容纳的元素的最大限值

14. rbegin()--返回指向集合中最后一个元素的反向迭代器

15. rend()--返回指向集合中第一个元素的反向迭代器

16. size()--集合中元素的数目

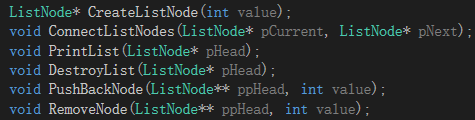
17. swap()--交换两个集合变量

18. upper\_bound()--返回大于某个值元素的迭代器

19. value\_comp()--返回一个用于比较元素间的值的函数

# 头文件（调用外部函数）

假设test.cpp想调用list.cpp中的函数，则可定义头文件list.h，并在list.h中声明list.cpp中的函数（如下），并在两个cpp文件中都#include”list.h”。



其他：

bool duplicate1(int numbers[], int length) {

//\*\*\*test

cout << sizeof(numbers) / sizeof(int) << endl; //1 注！数组带进来后，它自己不知道自己有多长（变成指针类型）！长度要另外带参数！！！

/\*\*\*/

//\*\*\*错误方法1:

int record[length] = { 0 }; //静态数组定义方式不能含有变量！直接报错！

/\*\*\*/

//\*\*\*错误方法2:

int\* const record = (int\*)malloc(sizeof(int)\*length);

memset(record,0,sizeof(record)); //此时record相当于一个指针类型，求sizeof为4！相当于只清零了第一个元素！

/\*\*\*/

//\*\*\*正确方法1(malloc和calloc主要是C的写法):

int\* const record = (int\*)malloc(sizeof(int)\*length);

memset(record,0,sizeof(int)\*length); //把初始化的空间大小改成sizeof(int)\*length

/\*\*\*/

//\*\*\*正确方法2:

int\* const record = (int\*)calloc(length, sizeof(int)); //calloc可以分配完动态空间后自动初始化为0！（注，参数和malloc略有不同）

//不能直接用int record[length]这种静态定义形式！因为这种形式length必须是常亮（在执行前就分配空间）！所以这里只能用动态内存分配！(melloc不初始化，calloc会初始化为0)

/\*\*\*/

//\*\*\*正确方法3（C++主流用new）:

int\* record = new int[length]; //别忘记delete[] record

memset(record, 0, sizeof(int)\*length);

/\*\*\*/

//正确方法4:

int\* record = new int[length](); //【如果改为(0)，这种初始化方法线上编译器clang c++出错（虽然本地可以运行）！】

}