目录

[1. 字符串 2](#_Toc490583111)

[1.1 substr 2](#_Toc490583112)

[1.2 push\_back 3](#_Toc490583113)

[1.3 length = size 3](#_Toc490583114)

[1.4 clear 3](#_Toc490583115)

[1.5 + 3](#_Toc490583116)

[1.6 strchr 源码 3](#_Toc490583117)

[1.7 sizeof 4](#_Toc490583118)

[1.8 strcpy 源码 4](#_Toc490583119)

[2. Vector 4](#_Toc490583120)

[2.1 size 5](#_Toc490583121)

[2.2 push\_back 5](#_Toc490583122)

[2.3迭代器 5](#_Toc490583123)

[2.3基本简介 5](#_Toc490583124)

[3.数组 6](#_Toc490583125)

[3.1计算数组的长度 7](#_Toc490583126)

[4.链表 7](#_Toc490583127)

[5.栈 10](#_Toc490583128)

[5.1队列 11](#_Toc490583129)

[6.指针 11](#_Toc490583130)

[7.Map 12](#_Toc490583131)

[8.排序 13](#_Toc490583132)

[9.结构体 14](#_Toc490583133)

[10.引用 14](#_Toc490583134)

[11.DP思想-一种求最优解的思路 15](#_Toc490583135)

[12.DFS 15](#_Toc490583136)

[13.BFS 16](#_Toc490583137)

%i，%d，%f 数字，%p 地址，%s 字符串，%c 字符

标准库的内容不用加.h;using namespace std 是规定命名空间；

## 字符串

学习资料：

使用sting需要在头部包含<#include>

C语言printf无法对string进行输出，貌似是C++的一个封装。

声明：**string** \*str = **new** **string**[nRows];

leetcode中第6题非常完整的完成了整个string的各个函数。

字符数组式的输入

String的输入输出

char str[1000];

while(scanf("%s",str) == 1)

{

printf("%s",str);

}

### 1.1 **substr**

实例：

#include <iostream>

#include <stdio.h>

#include<string>

using namespace std;

int main()

{

string s = "123456789";

string sub = s.substr(0,5);

cout<<sub<<endl;

return 0;

}

### 1.2 push\_back

只能输入字符，且只能加入最后一个

str[row].push\_back(s[i]);

### 1.3 length = size

**const** **int** len = (**int**)s.length();

string str1 = “asdf”;

str1.length() 与 str1.size() 是一样的

### 1.4 clear

s.clear();

### 1.5 +

String str1 = “MM”;

String str2 = “LL”;

Cout<<str1+str2<<endl;

输出结果是：MMLL

### 1.6 strchr 源码

#include <stdio.h>

#include <string>

char \*strchr(char \*s,char c)

{

while( \*s != '\0' && \*s != c)

{

s++;

}

return \*s==c? s: NULL;

}//这个函数的源码

int main(){

char str[] = "sdlakjflasjf";

printf("%i",strchr(str,'p'));

return 0;

}

### 1.7 sizeof

### 1.8 strcpy 源码

char\* strcpy(char\* des,char\* source)

{

char\* r = des;

if(des != NULL && source != NULL)

{

while((\*r++ = \*source++) != '\0');

}

return des;

}

范例1：

#include <stdio.h>

#include <string.h>

int strlenls(char const \*string)

{

int length;

for(length=0;\*string++ !='\0';)

length += 1;

return length;

}

int main(){

char a[100];

char b[100];

while(scanf("%s",a) ==1)

{

printf("originString:%s\n",a);

strcpy(b,a); //数组拷贝

printf("copyString:%s\n",b);

printf("strlen:%d\n",strlenls(a));

char \*c = strchr(a,'a'); //返回的是'a'第一次出现的地址！！！

printf("firstchar:%c\n",\*c);

char \*d = strstr(a,"abc"); //查找子串

printf("firstString:%s\n",d);

}

}

## Vector

学习链接：<http://www.cnblogs.com/mr-wid/archive/2013/01/22/2871105.html>

该链接之后还有比较好的C++解释

范例一：

#include <stdio.h>

#include <vector>

int main()

{

std::vector<int> test(10,1); //vector初始化的一种形式

int temp;

int num;

scanf("%d",&num);

for(int j = 0; j<num; j++ )

{

scanf("%d",&temp);

test.push\_back(temp); //可以利用这种方法进行数组的动态增长

}

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

{

printf("%d",test[i]);

}

return 0;

}

### 2.1 size

vector <int> result;

int i,j;

int size =nums.size();

### 2.2 push\_back

result.push\_back(j);

与之对应的是：result.pop\_back();

vector.begin与vector.end 这个两个方法

### 2.3迭代器

#include <stdio.h>

#include <vector>

int main()

{

std::vector<int> test; //vector初始化的一种形式

int temp;

int num;

scanf("%d",&num);

for(int j = 0; j<num; j++ )

{

scanf("%d",&temp);

test.push\_back(temp);

}

std::vector<int>::iterator i;

for(i = test.begin(); i!=test.end(); i++)

{

printf("%d",\*i);

}

return 0;

}

### 2.3基本简介

在c++中，vector是一个十分有用的容器，下面对这个容器做一下总结。

1 基本操作

(1)头文件#include<vector>.

(2)创建vector对象，vector<int> vec;

(3)尾部插入数字：vec.push\_back(a);

(4)使用下标访问元素，cout<<vec[0]<<endl;记住下标是从0开始的。

(5)使用迭代器访问元素.

vector<int>::iterator it;

for(it=vec.begin();it!=vec.end();it++)

cout<<\*it<<endl;

(6)插入元素： vec.insert(vec.begin()+i,a);在第i+1个元素前面插入a;

(7)删除元素： vec.erase(vec.begin()+2);删除第3个元素

vec.erase(vec.begin()+i,vec.end()+j);删除区间[i,j-1];区间从0开始

(8)向量大小:vec.size();

(9)清空:vec.clear();

示例：

class Solution {

public:

int climbStairs(int n) {

if(n==0)return 0;

vector<int>step(n,0);

step[0] = 1;

step[1] = 2;

for(int i=2; i<n; i++)

step[i] = step[i-2] + step[i-1];

return step[n-1];

}

};

## 3.数组

输入一个数组：

for(i = 0; i < 10 ; i++){

scanf("%d ",&a[i]);

}

### 3.1计算数组的长度

int a[] ={1,2,3,4,5,6,7,8,9,10};

int b;

b = sizeof(a)/sizeof(a[0]);

printf("%i",b);

## 4.链表

参考资料<http://blog.csdn.net/lpp0900320123/article/details/20356143>

范例1 存在错误

#include <stdio.h>

#include <stdlib.h>

struct ListNode

{

int val;

ListNode \*next;

ListNode(int x): val(x),next(NULL){}

};

ListNode\* createList()

{

int len = 10;

int i;

ListNode p(0);

ListNode \*pTail = &p;

for(i=1; i<len; i++)

{

ListNode temp(i);

pTail->next = &temp;

pTail = &temp; //有问题，这个只会搞成所用都是最后一个数

}

return p.next;

}

int main()

{

int i ;

int len = 10;

ListNode \*l2 = createList();

for(i=1 ; i<len; i++)

{

printf("%d ",\*l2);

l2 = l2->next;

}

return 0;

}

范例二：这个完美解决范例一的问题

#include <stdio.h>

#include <stdlib.h>

struct ListNode

{

int val;

ListNode \*next;

ListNode(int x): val(x),next(NULL){}

};

ListNode\* createList()

{

int len = 10;

int i;

ListNode li(0);

ListNode \*pTail = &li;

for(i=1; i<len; i++)

{

ListNode \*temp = (ListNode \*)malloc(sizeof(ListNode));

temp->val = i;

pTail->next = temp;

pTail = pTail->next; //就是这个部分的不同

}

return li.next;

}

int main()

{

int i ;

int len = 10;

ListNode \*l2 = createList();

for(i=1 ; i<len; i++)

{

printf("%d ",\*l2);

l2 = l2->next;

}

return 0;

}

链表的创建范例：（自己第一个手打的数据结构）

#include <stdio.h>

#include <stdlib.h>

typedef struct ListNode

{

int val;

ListNode \*next;

ListNode(int x):val(x),next(NULL){}

}List,\*pNode;

pNode CreateList();

void Traverse(pNode p);

int main()

{

pNode p = CreateList();

Traverse(p);

return 0;

}

pNode CreateList()

{

int len;

ListNode li(0); //ListNode li = new ListNode(0);

ListNode \*pTail = &li;

printf("array length: ");

scanf("%d",&len);

for(int i=0; i<len; i++)

{

int value;

printf("the value ");

scanf("%d",&value);

pNode temp = (pNode)malloc(sizeof(ListNode)); //pNode temp = new ListNode(value);

temp->val = value;

temp->next = NULL;

pTail->next = temp;

pTail = pTail->next;

}

return li.next;

}

void Traverse(pNode p)

{

while(NULL != p)

{

printf("%d ",\*p);

p = p->next;

}

}

## 5.栈

使用范例：

#include <stdio.h>

#include <string>

#include <stack>

#include <map>

using namespace std;

bool isValid(string s) {

stack<char> temp;

map<char, char> m ;

m['}'] = '{';

m[')'] = '(';

m[']'] = '[';

//map<char, char> m = {{']','['},{'}','{'},{')','('}};

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

if (s[i] == '[' || s[i] == '(' || s[i] == '{')

temp.push(s[i]);

else if (s[i] == ']' || s[i] == ')' || s[i] == '}')

{

if (temp.empty()||temp.top() != m[s[i]]) return false;

else temp.pop();

}

}

return temp.empty();

}

int main(){

string str = "{}{}[][]{}";

bool flag = isValid(str);

printf("%d",flag);

return 0;

}

手写MinStack一个范例：

class MinStack {

private:

stack<int> s1;

stack<int> s2;

public:

/\*\* initialize your data structure here. \*/

MinStack() {

}

void push(int x) {

s1.push(x);

if(s2.empty() || x<=getMin()) s2.push(x);

}

void pop() {

if(s1.top() == s2.top()) s2.pop();

s1.pop();

}

int top() {

return s1.top();

}

int getMin() {

return s2.top();

}

};

## 5.1队列

queue<int>q;

q.front(); //取队头的元素

q.pop(); //删除队头的元素

q.push(); //像队尾添加元素

## 6.指针

指针调用结构体的内容用->,变量用”.”

赋值的时候最好这样：

int \*a;

int x = 10;

a = &x;

范例1：利用指针来打印数组

void printArray(int \*ptr)

{

for( ;\*ptr!=10; )

printf("%d", \*ptr++);

}

int main()

{

int array[6] = {2,4,6,8,10};

printArray(array);

return 0;

}

## 7.Map

初始化：map<char, char> m = { {']','['},{')','('},{'}','{'} };

C++ 11 以前的内容：map<char, char> m ;

m['}'] = '{';

m[')'] = '(';

m[']'] = '[';

Map是c++的一个标准容器，她提供了很好一对一的关系，在一些程序中建立一个map可以起到事半功倍的效果，总结了一些map基本简单实用的操作！

1. map最基本的构造函数；

map<string , int >mapstring; map<int ,string >mapint;

map<sring, char>mapstring; map< char ,string>mapchar;

map<char ,int>mapchar; map<int ,char >mapint；

2. map添加数据；

map<int ,string> maplive;

1.maplive.insert(pair<int,string>(102,"aclive"));

2.maplive.insert(map<int,string>::value\_type(321,"hai"));

3, maplive[112]="April";//map中最简单最常用的插入添加！

3，map中元素的查找：

find()函数返回一个迭代器指向键值为key的元素，如果没找到就返回指向map尾部的迭代器。

map<int ,string >::iterator l\_it;;

l\_it=maplive.find(112);

if(l\_it==maplive.end())

cout<<"we do not find 112"<<endl;

else cout<<"wo find 112"<<endl;

4,map中元素的删除：

如果删除112；

map<int ,string >::iterator l\_it;;

l\_it=maplive.find(112);

if(l\_it==maplive.end())

cout<<"we do not find 112"<<endl;

else maplive.erase(l\_it); //delete 112;

## 8.排序

排序的效率主要集中在交换移动上面

1.快速排序

2.交换排序

int main(){

int a[10] = {0,1,2,3,4,5,6,7,8,9};

int i,j;

for(i = 0; i < 10; i++){

for(j = i+1; j < 10; j++)

if(a[i]> a[j])

{

int t = a[i];

a[i] = a[j];

a[j] = t;

}

}

for(i = 0; i < 10; i++)

{

printf("%d",a[i]);

}

return 0;

}

例子：

4 2 3 1

第一次：2 4 3 1

2 4 3 1

1 4 3 2

第二次：1 3 4 2

1 2 4 3

第三次：1 2 3 4

会打乱已有的顺序

3.冒泡排序

上面的交换排序，效率特别低，因为排完之后仍然是乱的，进行“临近两两交换”，减少了上面频繁的移动

for(I = 0; i<n-1; i++)

for(j=0; j<n-1-i; j++)

if(a[j]>a[j+1])swap(a[j],a[j+1]);

此处的n-1是“比较的次数”

例子：

4 2 3 1

第一次：2 4 3 1

2 3 4 1

2 3 1 4

第二次：2 3 1 4

2 1 3 4

第三次：1 2 3 4

范例：

#include <stdio.h>

void swap(int \*a,int \*b){

int c;

c = \*a;

\*a = \*b;

\*b = c;

}

int main(){

int a[10] = {0,1,2,3,4,5,6,7,8,9};

int i,j;

for(i = 0; i < 9; i++)

for(j = 0; j < 9-i; j++)

if(a[j] < a[j+1])

swap(&a[j],&a[j+1]);

for(i = 0; i < 10; i++)

{

printf("%d",a[i]);

}

return 0;

}

## 9.结构体

定义结构体变量，系统会为这个变量分配实际的内存空间。定义一个结构体指针，那它就只是个指针，大小也就是系统的地址长度，它里面的值应该指向一个具体的结构体内存空间，才算正常操作。定义了指针以后，要么就用malloc分配空间，要么指向一个已经存在的变量的地址，才能做实际操作

## 10.引用

#include <stdio.h>

#include <stdlib.h>

void swap(int &a,int &b)

{

int c;

c = a;

a = b;

b = c;

}

int main(){

int a = 1, b =10;

int &c = a;

int &d = b;

swap(c,d);

printf("%d %d",a,b);

return 0;

}

范例2：指针交换

#include <stdio.h>

void swap(int \*a,int \*b)

{

int c;

c = \*a;

\*a = \*b;

\*b = c;

}

int main(){

int array[10];

for(int i=0;i<10;i++)

scanf("%d",&array[i]);

for(int i=0;i<10-1;i++)

for(int j=0;j<10-i-1;j++)

if(array[j]>array[j+1])

swap(&array[j],&array[j+1]);

for(int i=0; i<10; i++)

printf("%d",array[i]);

}

范例3 引用

#include <stdio.h>

void swap(int &a,int &b)

{

int c;

c = a;

a = b;

b = c;

}

int main(){

int array[5];

for(int i=0;i<5;i++)

scanf("%d ",&array[i]);

for(int i=0;i<5-1;i++)

for(int j=0;j<5-i-1;j++)

if(array[j]>array[j+1])

swap(array[j],array[j+1]);

for(int i=0; i<5; i++)

printf("%d",array[i]);

}

## 11.DP思想-一种求最优解的思路

学习资料：<http://www.cnblogs.com/wust-owen/archive/2016/04/11/5377126.html>

例题爬楼梯问题：70leetcode

## 12.DFS

学习链接：<https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2/5224976?fr=aladdin&fromid=5055&fromtitle=DFS>

#include <stdio.h>

#include <stdlib.h>

#include <stack>

struct node{

int self; //数据

node \*left; //左节点

node \*right; //右节点

};

const int TREE\_SIZE = 9;

int main(){

std::stack<node\*> visited, unvisited;

node nodes[TREE\_SIZE];

node\* current;

for( int i=0; i<TREE\_SIZE; i++) //初始化树

{

nodes[i].self = i;

int child = i\*2+1;

if( child<TREE\_SIZE ) //Left child

nodes[i].left = &nodes[child];

else nodes[i].left = NULL;

child++;

if( child<TREE\_SIZE ) //Right child

nodes[i].right = &nodes[child];

else nodes[i].right = NULL;

}

unvisited.push(&nodes[0]); //先把0放入UNVISITED stack

while(!unvisited.empty()) //只有UNVISITED不空

{

current=(unvisited.top()); //当前应该访问的

unvisited.pop();

if(current->right!=NULL)

unvisited.push(current->right); // 把右边压入 因为右边的访问次序是在左边之后

if(current->left!=NULL)

unvisited.push(current->left);

visited.push(current);

printf("%d ",current->self);

}

return 0;

}

13.BFS

一般用于求解最短路径，或者走迷宫一类的问题

学习链接：

<https://baike.baidu.com/item/%E5%AE%BD%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2/5224802?fr=aladdin&fromid=542084&fromtitle=BFS>

#include <stdio.h>

#include <queue>

int mazeArr[6][5] = {{0,0,1,0,0},{0,1,0,0,0},{0,1,0,1,1},{0,1,0,0,0},{0,0,0,1,0},{0,0,0,0,0}}; //表示的是01矩阵 这是一个迷宫的初始化

int stepArr[4][2] = {{-1,0},{1,0},{0,-1},{0,1}}; //表示上下左右4个方向

int visit[6][5]; //表示该点是否被访问过，防止回溯，回溯很耗时。

struct Node

{

int x;

int y;

int step;

Node(int x1,int y1,int step1):x(x1),y(y1),step(step1){}

};

int BFS(int m,int n)

{

Node node(0,0,0);

std::queue<Node>q;

while(!q.empty())q.pop();

q.push(node);

while(!q.empty())

{

node=q.front();

q.pop();

if(node.x==m-1 && node.y==n-1)

{

return node.step;

}

visit[node.x][node.y]=1;

for(int i=0;i<4;i++)

{

int x=node.x+stepArr[i][0];

int y=node.y+stepArr[i][1];

if(x>=0 && y>=0 && x<m && y<n && visit[x][y]==0 && mazeArr[x][y]==0)

{

visit[x][y]=1;

Node next(x,y,node.step+1);

q.push(next);

}

}

}

return-1;

}

int main(){

printf("%d",BFS(6,5));

return 0;

}

## 14.二叉树

1.二叉树层遍历 107 leetcode

2. 二叉查找树（Binary Search Tree），（又：二叉搜索树，二叉排序树）它或者是一棵空树，或者是具有下列性质的二叉树： 若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值； 若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值； 它的左、右子树也分别为二叉排序树。

将有序数组变成二叉搜索树 108leetcode

3.