华科软院复试上机题

笔记本： 华科复试
创建时间： 2019/2/24 21:27 **更新时间：** 2019/3/3 19:26
作者： 302943821@qq.com

2017-1-最长公共子串

```
/*==============================================
求 2 个字符串的最长公共子串（输出时不包含空格）,字符串长度不超过 255。
例如：
输入：
What is your name?
My name is pipihui.
输出：
最长公共子串为”name”。
分析：最长公共子串问题，用动态规划，确定 dp[i][j]
i = 0 | j = 0 时，dp[i][j] = 0
str1[i] == str2[j]时，dp[i][j] = dp[i-1][j-1] + 1
str1[i] != str2[j]时，dp[i][j] = 0
注意：1.空格的问题，前面照常写程序，最后输出的时候简化为只输出不为空格的数
2.最长公共子串可能有多个，用endStr1[N]记录dp[i][j]==maxLen的串的end位置，最后循环输出每个子串
by shine_rise 2019/2/27
==============================================*/
#include <iostream>
#include <cstdio>
#define N 256
using namespace std;

//用于打印的函数
void printSubstring(char str[], int end, int len){
     int start = end - len + 1;
     for(int i=start; i<=end; i++){
          if(str[i]!=' ')
              cout<<str[i];
     }
     cout<<endl;
}

int main(){
     char str1[N],str2[N];
     int dp[N][N] = {0},maxLen = 0;
     gets(str1);
     gets(str2);
     //计算dp[i][j]的值和maxLen
     for(int i=0;str1[i]!='\0';i++){
          for(int j=0;str2[j]!='\0';j++){
              if(str1[i]==str2[j]){
                   if(i>0&&j>0){
                        dp[i][j]=dp[i-1][j-1]+1;
                   }else{
                        dp[i][j]=1;
                   }
                   if(maxLen<dp[i][j]){
                        maxLen = dp[i][j];
                   }
              }
```

```
            }
        }

        int endStr1[N];
        int numMaxLen = 0;

        //记录多个字符串长度为maxLen的end地址
        for(int i=0;str1[i]!='\0';i++){
            for(int j=0;str2[j]!='\0';j++){
                if(dp[i][j] == maxLen){
                    endStr1[numMaxLen++] = i;
                }
            }
        }

        //循环输出每个最长公共子串
        for(int i=0;i<numMaxLen;i++){
            printSubstring(str1, endStr1[i], maxLen);
        }

        getchar();
        return 0;
}
```

拓展1：求N个字符串的最长公共子串，N<=20，字符串长度不超过255，要求程序输入n和n个字符串，输出最长公共子串
如：输入：
2
What is your name ?
My name is pipihui.
输出：
最长公共子串为”name”。
拓展2：最长公共子序列
给出字符串str1和str2，如果一个字符串s同是str1和str2的子序列，则称s为二者的公共子序列，如果s最长，则称s为最长公共子序列，即LCS.
如str1 = "ABCBDCA", str2 = "DABDA", 二者的一个公共子序列为AA，而最长公共子序列为ABDA.
再如，str1 = "BACDABC", str2 = "AXBDC", 二者的LCS不止一个，如BDC, ABC, ADC都是LCS
因此需要明确：LCS有时候不唯一.

2017-2 国际象棋马走日
假设国际象棋棋盘有 5*5 共 25 个格子。设计一个程序，使棋子从初始位置（棋盘编号为 1 的位）开始跳马，能够把棋盘的格子全部都走一遍，每个格子只允许走一次。
（1）输出一个如图 2 的解，左上角为第一步起点。
（2）总共有多少解。
P.S 国际象棋的棋子是在格子中间的。国际象棋中的“马走日”，如下图所示，第一步为[1,1], 第二步为[2,8]或[2,12]，第三步可以是[3,5]或[3,21]等，以此类推。

```
/*====================================
假设国际象棋棋盘有 5*5 共 25 个格子。设计一个程序，使棋子从初始位置（棋盘编号
```

```
为 1 的位）开始跳马，能够把棋盘的格子全部都走一遍，每个格子只允许走一次。
（1）输出一个如图 2 的解，左上角为第一步起点。
（2）总共有多少解
回溯法
当确定一个位置之后，要考虑8种可以跳马的方向，
设置一个记步数的变量tag，当tag为25时退出递归，
同时把最后确定的一个位置，即写着25的格子的变量重置为0，同时退回到第24步，
去寻找其它第25步的可能位置。
by shine_rise 2019/2/28
=======================================*/
#include <iostream>
#include <string>
using namespace std;
const int ROW = 5;
const int COLUMN = 5;
int sum = 0;//sum表示第几种解法
int chess[ROW][COLUMN] = {0};
int xdir[8] = {-1, -2, -2, -1, 1, 2, 2, 1};
int ydir[8] = {-2, -1, 1, 2, 2, 1, -1, -2};

//用于打印的函数
void print(){
     for(int i=0;i<ROW;i++){
          for(int j=0;j<COLUMN;j++){
              cout<<chess[i][j]<<" ";
          }
          cout<<endl;
     }
}

//从(x,y)点依次访问其各个点，tag表明第几步
void visit(int x, int y, int tag){
     if(x>=0&&x<COLUMN&&y>=0&&y<ROW&&chess[x][y]==0){
          chess[x][y] = tag;
          if(tag == ROW*COLUMN){
              sum++;
              if(sum == 1){
                   cout<<"其中一种解法为："<<endl;
                   print();
              }
          }
          for(int i=0;i<8;i++){
              visit(x+xdir[i], y+ydir[i], tag+1);
          }
          chess[x][y] = 0;
     }
}
int main(){
     int x, y;
     visit(0, 0, 1);
     cout<<"一共有"<<sum<<"种解法"<<endl;
     getchar();
     return 0;
}
```

```
/*=====================================
假设国际象棋棋盘有 5*5 共 25 个格子。设计一个程序，使棋子从初始位置（棋盘编号
为 1 的位）开始跳马，能够把棋盘的格子全部都走一遍，每个格子只允许走一次。
（1）输出一个如图 2 的解，左上角为第一步起点。
（2）总共有多少解
```

```
暴力回溯法
by shine_rise 2019/2/25
=======================================*/
#include <iostream>
#include <string>
using namespace std;
const int ROW = 5;
const int COLUMN = 5;
int sum = 0;//sum表示第几种解法
int chess[ROW][COLUMN] = {0};

//用于打印的函数
void print(){
     for(int i=0;i<ROW;i++){
          for(int j=0;j<COLUMN;j++){
              cout<<chess[i][j]<<" ";
          }
          cout<<endl;
     }
}

//从(x,y)点依次访问其各个点，tag表明第几步
void visit(int x, int y, int tag){
     chess[x][y] = tag;
     if(tag == ROW*COLUMN){
          ++sum;
          if(sum == 1){
              cout<<"其中一种解法为："<<endl;
              print();
          }
     }
     //左上
     if(x-2>=0&&y-1>=0&&chess[x-2][y-1]==0)
          visit(x-2, y-1, tag+1);
     //上左
     if(x-1>=0&&y-2>=0&&chess[x-1][y-2]==0)
          visit(x-1, y-2, tag+1);
     //左下
     if(x-2>=0&&y+1<COLUMN&&chess[x-2][y+1]==0)
          visit(x-2, y+1, tag+1);
     //下左
     if(x-1>=0&&y+2<COLUMN&&chess[x-1][y+2]==0)
          visit(x-1, y+2, tag+1);
     //右上
     if(x+2<ROW&&y-1>=0&&chess[x+2][y-1]==0)
          visit(x+2, y-1, tag+1);
     //上右
     if(x+1<ROW&&y-2>=0&&chess[x+1][y-2]==0)
          visit(x+1, y-2, tag+1);
     //右下
     if(x+2<ROW&&y+1<COLUMN&&chess[x+2][y+1]==0)
          visit(x+2, y+1, tag+1);
     //下右
     if(x+1<ROW&&y+2<COLUMN&&chess[x+1][y+2]==0)
          visit(x+1, y+2, tag+1);
     chess[x][y] = 0;
}
int main(){
     int x, y;
     visit(0, 0, 1);
     cout<<"一共有"<<sum<<"种解法"<<endl;
     getchar();
```

```
    return 0;
}
```

拓展：国际象棋车走直线

国际象棋5*5=25.中的车可以水平的或竖直的移动，一个车要从一个棋盘的左上角(0,0)移到（n,m)这个坐标，有多少种最短路径？

```
#include <iostream>
#include <string>
using namespace std;
/*==============================================
国际象棋5*5=25,中的车可以水平的或竖直的移动，
一个车要从一个棋盘的左上角(0,0)移到（n,m)这个坐标，有多少种最短路径？
dp[i][j] = dp[i-1][j] + dp[i][j-1]
dp[i][0] = 1
dp[0][j] = 1;
by shine_rise 2019/2/25
==============================================*/
const int N = 5;
int main(){
    int map[N][N], n, m;
    //初始化
    for(int i=0;i<N;i++){
        map[i][0] = 1;
        map[0][i] = 1;
    }
    for(int i=1;i<N;i++){
        for(int j=1;j<N;j++){
            map[i][j] = map[i][j-1] + map[i-1][j];
        }
    }
    cin>>n>>m;
    cout<<map[n-1][m-1]<<endl;
    system("pause");
    return 0;
}
```

2016-1 青蛙跳台阶

```
#include <iostream>
#include <string>
using namespace std;
/*==================================
超级青蛙跳台阶。一个台阶总共有 n 级，超级青蛙有能力一次跳到 n 阶台阶，
也可以一次跳 n-1 阶台阶，也可以跳 n-2 阶台阶……也可以跳 1 阶台阶。
问超级青蛙跳到 n 层台阶有多少种跳法？（n<=50）
例：输入台阶数：3
输出种类数：4
解释：4 种跳法分别是（1，1，1），（1，2），（2，1），（3）
分析：f(n) = f(n-1) + f(n-2) + ... + f(2) + f(1) + f(0)
从0级台阶一次跳上n级台阶：f(0) = 1
从1级台阶一次跳上n级台阶：f(1) = f(0)
f(2) = f(1) + f(0)
...
==================================*/

//n阶台阶的方法个数函数
```

```
int getCount(int n){
     int count = 0;//count代表跳台阶的方法个数
     if(n==0){
          return 1;
     }else{
          for(int i=0;i<n;i++){
              count += getCount(i);
          }
          return count;
     }
}

int main(){
     int n;
     cin>>n;
     if(n <= 0){
          cout<<"请输入正确的n级台阶（n为正整数）";
          return -1;
     }else{
          cout<<getCount(n)<<endl;
     }
     return 0;
}
```

拓展：青蛙跳台阶一次只能跳1或2阶

```
#include <iostream>
using namespace std;
/*==============================
一个台阶总共有n级，如果一次可以跳1级或者2级，问一共有多少种跳法
转化成斐波那契数列，n级台阶若第一次跳一级，，后面有f(n-1)次跳法
若第一次跳两级，后面有f(n-2)种跳法
f(n) = f(n-1) + f(n-2)
f(1) = 1
f(2) = 2
by shine_rise 2019/2/25
================================*/

//n级台阶跳法函数
int getCount(int n){
     if(n == 1){
          return 1;
     }else if(n == 2){
          return 2;
     }else{
          return getCount(n-1) + getCount(n-2);
     }
}
int main(){
     int n;
     cin>>n;
     if(n <= 0){
          cout<<"请输入正确的台阶数n（n为正整数）"<<endl;
          return -1;
     }
     cout<<getCount(n)<<endl;
     return 0;
}
```

2016-2 青蛙杯足球比赛
青蛙（frog）杯第一届棒球比赛开赛啦。
你现在是一名记分员，输入一个字符串数组（比赛记录情况），按如下规则计分：
1. 如果该字符串是数字：代表当轮比赛的得分情况。
2. 如果该字符串是“+”： 代表当轮比赛得分情况为上两轮之和。
3. 如果该字符串是“C”： 代表上一轮得分无效。
4. 如果该字符串是“D”：代表当轮比赛得分为上一轮得分的两倍。
你需要得出最后总的得分情况并返回结果。
例：
输入：52CD+
输出：30
解释：
第 1 轮得分 5 分，当前总共得分 5 分。
第 2 轮得分 2 分，当前总共得分 5+2=7 分。
第 3 轮取消上轮得分，当前总共得分 5 分。
第 4 轮获得上一轮双倍得分，当前总共得分 5+10=15 分。
第 5 轮获得上两轮得分之和，当前总共得分 15+5+10=30 分

```
#include <iostream>
#include <string>
using namespace std;
/*==================================
青蛙（frog）杯第一届棒球比赛开赛啦。
你现在是一名记分员，输入一个字符串数组（比赛记录情况），按如下规则计分：
1. 如果该字符串是数字：代表当轮比赛的得分情况。
2. 如果该字符串是“+”： 代表当轮比赛得分情况为上两轮之和。
3. 如果该字符串是“C”： 代表上一轮得分无效。
4. 如果该字符串是“D”：代表当轮比赛得分为上一轮得分的两倍。
你需要得出最后总的得分情况并返回结果。
例：
输入：52CD+
输出：30
解释：
第 1 轮得分 5 分，当前总共得分 5 分。
第 2 轮得分 2 分，当前总共得分 5+2=7 分。
第 3 轮取消上轮得分，当前总共得分 5 分。
第 4 轮获得上一轮双倍得分，当前总共得分 5+10=15 分。
第 5 轮获得上两轮得分之和，当前总共得分 15+5+10=30 分。
分析：栈的用法
by shine_rise 2019/2/26
================================*/
int main(){
     string s;
     cin>>s;
     int len = s.length();
     //new[]和delete[]对应使用
     int *stack = new int[len];
     int top = -1;
     //先将数据存储到栈中
     for(int i=0;i<len;i++){
          if(s[i]>='0'&&s[i]<='9'){
              stack[++top] = s[i] - '0';
          }else if(s[i] == '+'){
```

```
            if(top < 1){
                cout<<"非法"<<endl;
                return -1;
            }
            int a = stack[top-1] + stack[top]; //c为该轮得分
            stack[++top] = a;
        }else if(s[i] == 'C'){
            if(top < 0){
                cout<<"非法"<<endl;
                return -1;
            }
            top--;
        }else if(s[i] == 'D'){
            int a = stack[top] * 2;
            stack[++top] = a;
        }else{
            cout<<"非法"<<endl;
            return -1;
        }
    }

    //计算总得分
    int sum = 0;
    for(int i=0;i<=top;i++){
        sum += stack[i];
    }
    cout<<"总得分为"<<sum<<endl;
    return 0;
}
```

2015-1-最长数字串

```
#include <iostream>
#include <string>
using namespace std;
/*==============================
1.设计一个程序，输入一个字符串以#结尾，输出此字符串中连续出现最长的数字串以及其开始的下标，
例：
输入：
ab125ff1234567#
输出：
1234567 开始位置为 8
by shine_rise 2019/2/26
==============================*/
int main(){
    string s;
    cin>>s;
    char *scpy = new char[s.length()];//scpy用于存储连续数字串的长度
    //将连续数字串长度存储到scpy中
    for(int i=0;s[i]!='#';i++){
        if(s[i]>='0'&&s[i]<='9'){
            if(i != 0){
                scpy[i] = scpy[i-1] + 1;
            }else{
                scpy[i] = 0;
            }
        }else{
            scpy[i] = 0;
```

```
            }
        }

        int maxLen = 0, end;//end表示最长数字串结束的下标
        //求最长数字串
        for(int i=0;s[i] != '#';i++){
            if(maxLen < scpy[i]){
                maxLen = scpy[i];
                end = i;
            }
        }

        //输出最长数字串以及其开始的下标
        cout<<"最长数字串为";
        for(int i=end-maxLen + 1;i<=end;i++){
            cout<<s[i];
        }
        cout<<endl;
        cout<<"开始的下标为"<<end-maxLen+2<<endl;
        return 0;
}
```

2014-1-完全平方数

```
#include <iostream>
using namespace std;
/*==================================
在三位整数（100 至 999）中寻找符合下列条件的整数并依次从小到大输出。
（1）完全平方数。
（2）含有两位数字相同。
例：144，676…
by shine_rise 2019/2/26
==================================*/
int main(){
    for(int i=10;i<32;i++){
        int j = i * i;//j为100-999之间的完全平方数
        int a, b, c;//a,b,c分别为j的个位、十位、百位
        a = j % 10;
        b = j / 10 % 10;
        c = j / 100;
        if((a==b&&a!=c&&b!=c)||(a!=b&&b==c&&a!=c)||(a!=b&&a==c&&b!=c)){
            cout<<j<<endl;
        }
    }
    system("pause");
    return 0;
}
```

2014-2-下标为奇数的小写字母转化为大写字母

```
#include <iostream>
#include <string>
using namespace std;
/*==================================
输入一串字符串，把下标为奇数的小写字母(从 0 开始编号)转换为大写字母。
输出转换后的字符串。
by shine_rise 2019/2/26
```

```
====================================*/
int main(){
     string s;
     cin>>s;
     for(int i=0;i<s.length();i++){
          if(i%2==1){
              if(s[i]>='a'&&s[i]<='z'){
                   s[i] = 'A'-'a'+s[i];
              }
          cout<<s[i];
          }
     }
     return 0;
}
```

2013-2-八皇后

```
/*============================================
编程解决“八皇后问题”：即在一个 8*8 的矩形格子中排放 8 个皇后，要满足的条件包
括：任意两个皇后不能在同一行，同一列，也不能在同一条对角线上。
要求编程给出解的个数
by shine_rise 2019/2/28
============================================*/
#include <iostream>
using namespace std;
const int N = 9;
int a[50], b[50], c[50];//a表示行，b表示"\"， c表示"/",0为可放入，1为不可放入
int chess[N][N] = {0};//chess表示皇后放入的位置，1为皇后，0为空
int sum = 0;//sum表示解个数

//用于打印皇后位置
void print(){
     sum++;
     cout<<"第"<<sum<<"种解法为："<<endl;
     for(int i=1;i<N;i++){
          for(int j=1;j<N;j++){
              cout<<chess[i][j]<<" ";
          }
          cout<<endl;
     }
}

//回溯法,column为按列搜索能填入的位置
void search(int column){
     //搜索第column列的第row行
     for(int row=1;row<N;row++){
          //皇后满足的条件
          if((a[row]==0)&&(b[row+column]==0)&&(c[column+7-row]==0)){
              //可以放入皇后，放入后将每一行，每一对角线置为1
              chess[row][column] = 1;
              a[row] = 1;
              b[row+column] = 1;
              c[column+7-row] = 1;
              if(column == 8){
                   print();
              }else
                   search(column+1);
              //回溯前清理
              chess[row][column] = 0;
```

```
                a[row] = 0;
                b[row+column] = 0;
                c[column+7-row] = 0;
            }
        }
}

int main(){
     search(1);
     cout<<"共有"<<sum<<"种解法"<<endl;
     system("pause");
     return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>

#define max 8

int queen[max], sum=0; /* max为棋盘最大坐标 */

void show() /* 输出所有皇后的坐标 */
{
    int i;
    for(i = 0; i < max; i++)
    {
         printf("(%d,%d) ", i, queen[i]);
    }
    printf("\n");
    sum++;
}

int check(int n) /* 检查当前列能否放置皇后 */
{
    int i;
    for(i = 0; i < n; i++) /* 检查横排和对角线上是否可以放置皇后 */
    {
        if(queen[i] == queen[n] || abs(queen[i] - queen[n]) == (n - i))
        {
            return 1;
        }
    }
    return 0;
}

void put(int n) /* 回溯尝试皇后位置,n为横坐标 */
{
    int i;
    for(i = 0; i < max; i++)
    {
        queen[n] = i; /* 将皇后摆到当前循环到的位置 */
        if(!check(n))
        {
            if(n == max - 1)
            {
                show(); /* 如果全部摆好，则输出所有皇后的坐标 */
            }
            else
            {
                put(n + 1); /* 否则继续摆放下一个皇后 */
```

```
            }
        }
    }
}

int main()
{
    put(0); /* 从横坐标为0开始依次尝试 */
    printf("%d", sum);
    return 0;
}
```

2012-2-约瑟夫环

一群人（排列的编号从 1 到 N，N 可以设定）围成一圈，按一定规则出列。剩余的人仍然围成一圈，出列规则是顺着 1 到 N 的方向对圈内的人从 1 到 C 记数（C 可以设定）。圈内记数为 C 的人出列。剩余的人重新计数。按上述规则，让圈内所有的人出列。请编程输出出列编号的序列。

例：

若 N=3,C=1,则出列编号的序列为 1，2，3

若 N=3,C=2,则出列编号的序列为 2，1，3

```
#include <iostream>
#include <string>
using namespace std;
/*=======================================
约瑟夫环
一群人（排列的编号从 1 到 N，N 可以设定）围成一圈，按一定规则出列。剩余的人仍
然围成一圈，出列规则是顺着 1 到 N 的方向对圈内的人从 1 到 C 记数（C 可以设定）。圈内
记数为 C 的人出列。剩余的人重新计数。按上述规则，让圈内所有的人出列。请编程输出
出列编号的序列。
例：
若 N=3,C=1,则出列编号的序列为 1，2，3
若 N=3,C=2,则出列编号的序列为 2，1，3
分析：
方法一：链表，麻烦，省略
方法二：循环数组，
循环条件为：index = (index + 1) % count
初始alive = count,每出圈一个，alive --,
循环结束条件为：alive > 0
每一次循环，就是过一个人，该人在圈内还是圈外，圈内number++,圈外number不变，
故定义一数组circle[count]，记录该编号是已出圈还是未出圈，圈内为0，圈外为1
number = number + 1 - circle[index]

malloc()只分配内存，calloc()除了分配内存外，还将分配的内存清空为0。
这两段代码等价
int* buffer = (int*)malloc(512 * sizeof(int));
memset(buffer, 0, 512 * sizeof(int));
int* buffer = (int*)calloc(512, sizeof(int));
by shine_rise 2019/2/27
=======================================*/
void sequence(int count, int boom){
     int number = 0;//number为计数
     int alive = count;//alive表示是否全出圈
     int index = 0;//index为下标
     int *circle = (int*)calloc(count, sizeof(int));//创建数组，用于表示该元素
是在圈内还是圈外，初始为0
     while(alive > 0){
```

```
            number = number + 1 - circle[index];
            if(number == boom){
                //出圈的操作
                cout<<index + 1<<" ";
                circle[index] = 1;//circle置为1表示在圈外
                number = 0;//重新计数
                alive --;
            }
            index = (index + 1)%count;
        }
        free(circle);
}
int main(){
        int n, c;//n为总人数，第c个出圈
        //cin空字符(包括回车,TAB,空格)都会当成一个输入的结束
        cin>>n>>c;
        sequence(n, c);
        return 0;
}
```

拓展：约瑟夫环，给定n,c，选定一个位置使得你最后一个被杀死

2010-2-马走日，只能向右不能向左

```
/*=========================================
在半个中国象棋棋盘(5*5)上,马在左上角(1，1)处,马走日字。
只能往右走，不能向左，可上可下。求从起点到(m, n)处有几种不同的走法
分析：
=========================================*/
#include <iostream>
#include <string>
using namespace std;
const int ROW = 5;
const int COLUMN = 5;
int sum = 0;
int chess[ROW][COLUMN] = {0};
int xdir[4] = {-2, -1, 1, 2};
int ydir[4] = {1, 2, 2, 1};

//访问函数，x,y为此时坐标，m,n为要到达的位置
void visit(int x, int y, int m, int n){
        if(x>=0&&x<COLUMN&&y>=0&&y<ROW&&chess[x][y]==0){
                if(x==m&&y==n){
                        sum++;
                        return;
                }
                chess[x][y] = 1;
                for(int i=0;i<4;i++)
                        visit(x+xdir[i], y+ydir[i], m, n);
                chess[x][y] = 0;
        }
}

int main(){
        int m, n;
        cin>>m>>n;
        //为方便，从下标为0开始
        visit(0, 0, m-1, n-1);
        cout<<"有"<<sum<<"种走法"<<endl;
```

```
     return 0;
}
```

2008-1-斐波那契数列

```
#include <iostream>
#include <string>
using namespace std;
/*=======================================
求数列 s(n)=s(n-1)+s(n-2)的第 n 项的值。其中 s(1)=s(2)=1。
输入：n,
输出：s(n)
by shine_rise 2019/2/27
=======================================*/
int getNum(int n){
     if( n==1 || n==2 ){
          return 1;
     }else{
          return getNum(n-1)+getNum(n-2);
     }
}
int main(){
     int n;
     cin>>n;
     cout<<getNum(n);
     return 0;
}
```

2008-2-水仙花数

```
/*=====================================
打印出所有的“水仙花数”，所谓“水仙花数”是指一个 3 位数，其各位数字立方和等于
该数本身。
例：153=1*1*1+3*3*3+5*5*5
by shine_rise 2019/2/27
=====================================*/
#include <iostream>
#include <string>
using namespace std;
int main(){
     for(int i=100;i<1000;i++){
          int a, b, c;//a,b,c分别为i的个位，十位，百位
          a = i % 10;
          b = i / 10 % 10;
          c = i / 100;
          if(i==(a*a*a+b*b*b+c*c*c)){
              cout<<i<<endl;
          }
     }
     return 0;
}
```

2008-3-已知前序中序序列求后续

```
/*=================================
```

已知一颗二叉树 S 的前序遍历和中序遍历序列，请编程输出二叉树 S 的后续遍历序列。
例：
输入：
pred[]/先序：A、B、D、E、C、F、G
inod[]/中序：D、B、E、A、C、G、F
输出：
后序遍历序列是：D、E、B、G、F、C、A
by shine_rise 2019/2/27
==========*/

```
#include <iostream>
#include <string>
using namespace std;
const int N = 50;
typedef char ElemType;
typedef struct BiTree{
     ElemType data;
     struct BiTree *lchild;
     struct BiTree *rchild;
}BiTree;

//获取元素ch在数组array中的下标，数组array长度为len
int getIndex(ElemType ch, ElemType *array, int len){
     for(int i=0;i<len;i++){
          if(array[i] == ch){
              return i;
          }
     }
}

//构造二叉树根节点,pre为先序数组，in为中序数组，len为数组长度
BiTree *createBiTree(ElemType *pre, ElemType *in, int len){
     if(len <= 0){
          return NULL;
     }
     BiTree *root = new BiTree;
     root->data = *pre;
     int index = getIndex(*pre, in, len);
     root->lchild = createBiTree(pre+1, in, index);
     root->rchild = createBiTree(pre+1+index, in+1+index, len-index-1);
     return root;
}

//后序遍历序列
void postOrder(BiTree *root){
     if(root != NULL){
          postOrder(root->lchild);
          postOrder(root->rchild);
          cout<<root->data;
     }
}

int main(){
     ElemType *pre = new ElemType[N];
     ElemType *in = new ElemType[N];
     cin>>pre;
     cin>>in;
     //strlen为数组长度
     BiTree *root = createBiTree(pre, in, strlen(in));
     postOrder(root);
     cout<<endl;
     return 0;
```

```
}
```

2008-3-拓展，已知中序后序序列求前序

```
/*==================================
已知一颗二叉树 S 的后序遍历和中序遍历序列，请编程输出二叉树 S 的先序遍历序列。
by shine_rise 2019/2/27
====================================*/
#include <iostream>
#include <string>
using namespace std;
typedef struct BiTree{
     char data;
     struct BiTree *lchild;
     struct BiTree *rchild;
}BiTree;
const int N = 50;

//获取元素在中序数组中的下标，ch为元素，array为数组，len为数组长度
int getIndex(char ch, char *array, int len){
     for(int i=0;i<len;i++){
          if(array[i] == ch){
              return i;
          }
     }
}

//构造二叉树,post为后序数组，in为中序数组，len为数组长度
BiTree *createBiTree(char *in, char *post, int len){
     if(len <= 0){
          return NULL;
     }
     BiTree *root = new BiTree;
     char ch = post[len-1];
     root->data = ch;
     int index = getIndex(ch, in, len);
     root->lchild = createBiTree(in, post, index);
     root->rchild = createBiTree(in+index+1, post+index, len-index-1);
     return root;
}

//输出先序遍历序列
void preOrder(BiTree *root){
     if(root != NULL){
          cout<<root->data;
          preOrder(root->lchild);
          preOrder(root->rchild);
     }
}

int main(){
     char *in = new char[N];
     char *post = new char[N];
     cin>>in;
     cin>>post;
     BiTree *root = createBiTree(in, post, strlen(in));
     preOrder(root);
     cout<<endl;
     return 0;
```

```
}
```

拓展1：正数负数零，负数在前，零在中间，正数在后

```
/*=====================================
一个长度为 n 数组由负数、0、正数组成。编写函数，将其重新排序为前段都是负数，中间
均为 0 ，后段均为正数的结构。
第一行输入n，第二行输入这个长度为n的数组
by shine_rise 2019/2/27
=====================================*/
#include <iostream>
#include <string>
using namespace std;
const int N = 255;
void temp(int *a, int *b){
     int temp = *a;
     *a = *b;
     *b = temp;
}
int main(){
     int s[N];
     int n;
     cin>>n;
     for(int i=0;i<n;i++){
          cin>>s[i];
     }
     int left = 0, right = n - 1, count = 0, i = 0;
     while(count < n){
          if(s[i] > 0){
              temp(&s[i], &s[right]);
              right-- ;
              count++;
          }
          if(s[i] == 0){
              i++;
              count++;
          }
          if(s[i]<0){
              temp(&s[i], &s[left]);
              left++;
              count++;
              i++;
          }
     }
     for(int i=0;i<n;i++){
          cout<<s[i]<<" ";
     }
     return 0;
}
```

拓展2：正数负数，负数在前，正数在后

```
/*=====================================
一个长度为 n 数组由负数、正数组成。编写函数，将其重新排序为前段都是负数，后段均为正数的
结构。
第一行输入n，第二行输入这个长度为n的数组
```

```
by shine_rise 2019/2/27
=======================================*/
#include <iostream>
#include <string>
using namespace std;
const int N = 255;

//用于交换的函数
void temp(int *a, int *b){
     int temp = *a;
     *a = *b;
     *b = temp;
}

int main(){
     int n, str[N];
     cin>>n;
     for(int i=0;i<n;i++){
          cin>>str[i];
     }
     int left = 0;
     int right = n - 1;
     int i = 0;
     int count = 0;
     while(count < n){
          if(str[i] < 0){
              left++;
              count++;
              i++;
          }
          if(str[i] > 0){
              temp(&str[i], &str[right]);
              right--;
              count++;
          }
     }
     for(int i=0;i<n;i++){
          cout<<str[i]<<" ";
     }
     cout<<endl;
     return 0;
}
```

拓展3-子串在主串中匹配出现次数

```
//子串在主串中匹配，求子串出现的次数，不重复
#include <iostream>
using namespace std;
const int N = 255;
int main(){
     char *str = new char[N];
     char *substr = new char[N];
     cin>>str;
     cin>>substr;
     int a = 0, b = 0;
     int num = 0;//num记录次数
     while(a<strlen(str)&&b<strlen(substr)){
          if(str[a] == substr[b]){
              if(b == strlen(substr) - 1){
```

```
                    num++;
                    a++;
                    b = 0;
               }else{
                    a++;
                    b++;
               }
          }else{
               a++;
               b = 0;
          }
     }
     cout<<num<<endl;
     return 0;
}
```

拓展4-数字串中奇数移到偶数的前面

```
//将数组中奇数移到偶数前面
#include <iostream>
#include <string>
using namespace std;
int main(){
     int n;
     cin>>n;
     int *str = new int[255];
     for(int i=0;i<n;i++){
          cin>>str[i];
     }
     int left = 0;
     int right = n - 1;
     int i = 0;
     int count = 0;
     while(count < n){
          if(str[i]%2==1){
               left++;
               count++;
               i++;
          }else{
               int temp = str[i];
               str[i] = str[right];
               str[right] = temp;
               count++;
               right--;
          }
     }
     for(int i=0;i<n;i++){
          cout<<str[i]<<" ";
     }
     cout<<endl;
     return 0;
}
```

拓展5-已知满二叉树前序序列，求后序序列

```
//已知满二叉树先序遍历序列，求后序遍历序列
#include <iostream>
```

```
using namespace std;

void search(char *str, int len){
     if(len<=0){
          return;
     }else{
          int sublen = (len - 1)/2;
          search(str+1, sublen);
          search(str+1+sublen, sublen);
          cout<<*str;
     }
}
int main(){
     char *str = new char[50];
     cin>>str;
     search(str, strlen(str));
     return 0;
}
```