# 1.一元多项式相加(10分)

成绩: 10 / 折扣: 0.8

### 题目说明：

编写一元多项式加法运算程序。要求用线性链表存储一元多项式（参照课本）。该程序有以下几个功能：

1. 多项式求和

输入：输入三个多项式，建立三个多项式链表Pa、Pb、Pc

（提示：调用CreatePolyn(polynomial &P,int m)。

输出：显示三个输入多项式Pa、Pb、Pc、和多项式Pa+Pb、多项式Pa+Pb+Pc

（提示：调用AddPolyn(polynomial &Pa, polynomial Pb), 调用PrintPolyn(polynomial P))。

0. 退出

### 输入：

根据所选功能的不同，输入格式要求如下所示（第一个数据是功能选择编号，参见测试用例）：

* 1

多项式A包含的项数，以指数递增的顺序输入多项式A各项的系数（整数）、指数（整数）

多项式B包含的项数，以指数递增的顺序输入多项式B各项的系数（整数）、指数（整数）

多项式C包含的项数，以指数递增的顺序输入多项式C各项的系数（整数）、指数（整数）

* 0 －－－操作终止，退出。

### 输出：

对应一组输入，输出一次操作的结果（参见测试用例）。

* 1 多项式输出格式：以指数递增的顺序输出: <系数,指数>,<系数,指数>,<系数,指数>,参见测试用例。零多项式的输出格式为<0,0>
* 0 无输出

#include<iostream>

#include<stdlib.h>

using std::cin;

using std::cout;

using std::endl;

struct date

{

int a;

int b;

struct date\* pnext;

};

typedef struct date DATE;

typedef struct date\* PDATE;

void output(PDATE p)

{

int f=0;

p=p->pnext;

while(p!=NULL)

{

if(p->a!=0)

{

f=1;

cout<<"<"<<p->a<<","<<p->b<<">";

if(p->pnext==NULL)

cout<<endl;

else

cout<<",";

}

p=p->pnext;

}

if(f==0)

cout<<"<0,0>"<<endl;

}

void add(PDATE a,PDATE b,PDATE c)

{

PDATE p1,p2,p3;

p1=a;

p2=b;

p3=c;

if(p1!=NULL) p1=p1->pnext; //skip head

if(p2!=NULL) p2=p2->pnext;

while((p1!=NULL)&&(p2!=NULL))

{

if(p1->b>p2->b)

{

p3->pnext=(PDATE)malloc(sizeof(DATE));

p3=p3->pnext;

p3->a=p2->a;

p3->b=p2->b;

p3->pnext=NULL;

p2=p2->pnext;

}

else if(p1->b<p2->b)

{

p3->pnext=(PDATE)malloc(sizeof(DATE));

p3=p3->pnext;

p3->a=p1->a;

p3->b=p1->b;

p3->pnext=NULL;

p1=p1->pnext;

}

else

{

p3->pnext=(PDATE)malloc(sizeof(DATE));

p3=p3->pnext;

p3->a=p1->a+p2->a;

p3->b=p1->b;

p3->pnext=NULL;

p1=p1->pnext;

p2=p2->pnext;

}

}//end while

if(p1==NULL)

p3->pnext=p2;

if(p2==NULL)

p3->pnext=p1;

}

int main()

{

int flag;

int n;

PDATE P[6]={NULL};

PDATE p=NULL;

for(int i=0;i<6;i++)

{

P[i]=(PDATE)malloc(sizeof(DATE));

P[i]->a=0;

P[i]->b=0;

P[i]->pnext=NULL;

}

cin>>flag;

if(flag==1)

{

for(int i=1;i<4;i++)

{

p=P[i];

cin>>n;

while(n--!=0)

{

p->pnext=(PDATE)malloc(sizeof(DATE));

p=p->pnext;

cin>>p->a>>p->b;

p->pnext=NULL;

}

output(P[i]);

}

}

add(P[1],P[2],P[4]);

output(P[4]);

add(P[4],P[3],P[5]);

output(P[5]);

}

# 0 约瑟夫问题(10分)

成绩: 10 / 折扣: 0.8

0 约瑟夫问题

成绩10分  折扣0.8

      (本题要求用循环链表实现)

      0 ,1, 2, 3题,只能选做三题.

约瑟夫问题是一个经典的问题。已知n个人（不妨分别以编号1，2，3，…，n 代表）围坐在一张圆桌周围，从编号为 k 的人开始，从1开始顺时针报数1, 2, 3, ...，顺时针数到m 的那个人，出列并输出。然后从出列的下一个人开始，从1开始继续顺时针报数，数到m的那个人，出列并输出，…依此重复下去，直到圆桌周围的人全部出列。

输入：n,k,m

输出：按照出列的顺序依次输出出列人的编号，编号中间相隔一个空格,每10个编号为一行。

非法输入的对应输出如下

a)

输入：：n、k、m任一个小于1  
输出：n,m,k must bigger than 0.

b)

输入：k>n

输出：k should not bigger than n.

例

输入

9,3,2

输出

4 6 8 1 3 7 2 9 5

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

struct date

{

int a;

struct date\* next;

};

typedef struct date DATE;

typedef struct date\* PDATE;

PDATE setnew(PDATE p,int a)

{

PDATE pt;

pt=(PDATE) malloc (sizeof(DATE));

pt->a=a;

pt->next=p->next;

p->next=pt;

return pt;

}

int count;

PDATE del(PDATE p0)

{

if(!count)

{

printf("\n");

count=10;

}

printf("%d ",p0->a);

PDATE p=p0->next;

p0->a=p->a;

p0->next=p->next;

free(p);

count--;

return p0;

}

int main()

{

count=10;

int n=0,k=0,m=0;

scanf("%d,%d,%d",&n,&m,&k);

if(!(n>0&&m>0&&k>0))

printf("n,m,k must bigger than 0.\n");

else if(m>n)

printf("k should not bigger than n.\n");

else

{

PDATE p=NULL;

PDATE head=(DATE \*)malloc(sizeof(DATE));

head->next=head;

head->a=1;

p=head;

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

p=setnew(p,i);

while(p->a!=m)

p=p->next;

while(n)

{

// int temp=k;

int temp=k%n+n;

while(--temp)

p=p->next;

del(p);

n--;

}

printf("\n");

}

}

# 2. 综教楼后的那个坑

成绩: 10 / 折扣: 0.8

### 描述

　　在 LIT 综教楼后有一个深坑，关于这个坑的来历，有很多种不同的说法。其中一种说法是，在很多年以前，这个坑就已经在那里了。这种说法也被大多数人认可，这是因为该坑有一种特别的结构，想要人工建造是有相当困难的。

　　从横截面图来看，坑底成阶梯状，由从左至右的 1..N 个的平面构成（其中 1 ≤ N ≤ 100,000），如图：

　　　＊　　　　　　　　　　　　＊ :  
　　　＊　　　　　　　　　　　　＊ :  
　　　＊　　　　　　　　　　　　＊ 8  
　　　＊　　　　＊＊　　　　　　＊ 7  
　　　＊　　　　＊＊　　　　　　＊ 6  
　　　＊　　　　＊＊　　　　　　＊ 5  
　　　＊　　　　＊＊＊＊＊＊＊＊＊ 4 <- 高度  
　　　＊　　　　＊＊＊＊＊＊＊＊＊ 3  
　　　＊＊＊＊＊＊＊＊＊＊＊＊＊＊ 2  
　　　＊＊＊＊＊＊＊＊＊＊＊＊＊＊ 1  
平面　｜　 1　 ｜2｜　　　3　　　 ｜

每个平面 i 可以用两个数字来描述，即它的宽度 Wi 和高度 Hi，其中 1 ≤ Wi ≤ 1,000、1 ≤ Hi ≤ 1,000,000，而这个坑最特别的地方在于坑底每个平面的高度都是不同的。每到夏天，雨水会把坑填满，而在其它的季节，则需要通过人工灌水的方式把坑填满。灌水点设在坑底位置最低的那个平面，每分钟灌水量为一个单位（即高度和宽度均为 1）。随着水位的增长，水自然会向其它平面扩散，当水将某平面覆盖且水高达到一个单位时，就认为该平面被水覆盖了。

　　请你计算每个平面被水覆盖的时间。

　　 灌水              水满后自动扩散  
        |                       |                             
      \* |          \*      \*     |      \*      \*            \*  
      \* V          \*      \*     V      \*      \*            \*  
      \*            \*      \*    ....    \*      \*~~~~~~~~~~~~\*  
      \*    \*\*      \*      \*~~~~\*\* :    \*      \*~~~~\*\*~~~~~~\*  
      \*    \*\*      \*      \*~~~~\*\* :    \*      \*~~~~\*\*~~~~~~\*  
      \*    \*\*      \*      \*~~~~\*\*~~~~~~\*      \*~~~~\*\*~~~~~~\*  
      \*    \*\*\*\*\*\*\*\*\*      \*~~~~\*\*\*\*\*\*\*\*\*      \*~~~~\*\*\*\*\*\*\*\*\*  
      \*~~~~\*\*\*\*\*\*\*\*\*      \*~~~~\*\*\*\*\*\*\*\*\*      \*~~~~\*\*\*\*\*\*\*\*\*  
      \*\*\*\*\*\*\*\*\*\*\*\*\*\*      \*\*\*\*\*\*\*\*\*\*\*\*\*\*      \*\*\*\*\*\*\*\*\*\*\*\*\*\*  
      \*\*\*\*\*\*\*\*\*\*\*\*\*\*      \*\*\*\*\*\*\*\*\*\*\*\*\*\*      \*\*\*\*\*\*\*\*\*\*\*\*\*\*  
　　　4 分钟后        　　　26 分钟后 　　　　　　　50 分钟后  
　　　平面 1 被水覆盖 　　　　平面 3 被水覆盖　　　　平面 2 被水覆盖输入

　　输入的第一行是一个整数 N，表示平面的数量。从第二行开始的 N 行上分别有两个整数，分别表示平面的宽度和高度。

### 输出

　　输出每个平面被水覆盖的时间。

#include<stdlib.h>

#include<stdio.h>

struct date

{

long long \* timedate;

long h;

int w;

struct date\* pl;

struct date\* pr;

};

typedef struct date DATE;

typedef struct date\* PDATE;

PDATE setnew(PDATE p0,int w,long h,long long \* num)//p0为左邻

{

PDATE p=(PDATE) malloc(sizeof(DATE));

p->timedate=num;

p->pl=p0;

p->pr=NULL;

p0->pr=p;

p->h=h;

p->w=w;

return p;

}

void output(long long\* p,long n)

{

while(n--)

printf("%lld\n",\*(++p));

}

int main()

{

long long myclock;

long n;

int w;

long h;

PDATE p=NULL,pt=NULL;

//set leftp

PDATE left=(PDATE) malloc(sizeof(DATE));

left->timedate=NULL;

left->pl=NULL;

left->pr=NULL;

left->h=1000000;

left->w=0;

p=left;

pt=left;

scanf("%d",&n);

long long\* timedate=new long long[n+1];

for(long i=0;i<n;i++)

{

//cin>>w>>h;

scanf("%d%d",&w,&h);

p=setnew(p,w,h,timedate+i+1);

if(pt->h>h)

pt=p;

}

PDATE right=setnew(p,0,1000000,NULL);

p=pt;

myclock=0;

while(p->pl->h!=p->pr->h)

{

\*(p->timedate)=myclock+p->w;

//计算时间并删除合并

if(p->pl->h>p->pr->h)

{

myclock+=(p->pr->h-p->h)\*p->w;

p->pr->w+=p->w;

p->pl->pr=p->pr;

p->pr->pl=p->pl;

pt=p;

p=p->pr;

delete pt;

}

else if(p->pl->h<p->pr->h)

{

myclock+=(p->pl->h-p->h)\*p->w;

p->pl->w+=p->w;

p->pl->pr=p->pr;

p->pr->pl=p->pl;

pt=p;

p=p->pl;

delete pt;

}

//移至下一进水点

if(p->pl->h>p->h&&p->pr->h>p->h)

continue;

else if(p->pl->h<p->pr->h)//左移

{

while(p->h>p->pl->h)

p=p->pl;

}

else //右移

{

while(p->h>p->pr->h)

p=p->pr;

}

}

myclock+=p->w;

\*(p->timedate)=myclock;

output(timedate,n);

}

# 3. 单词压缩存储(10分)

成绩: 10 / 折扣: 0.8

如果采用单链表保存单词，可采用如下办法压缩存储空间。如果两个单词的后缀相同，则可以用同一个存储空间保存相同的后缀。例如，原来分别采用单链表保存的单词Str1“abcdef”和单词Str2“dbdef”，经过压缩后的存储形式如下。

请设计一个高效的算法完成两个单链表的压缩存储，并估计你所设计算法的时间复杂度。

**要求：**阅读预设代码，编写函数SNODE \* ziplist( SNODE \* head1, SNODE \* head2 )  
ziplist的功能是：在两个串链表中，查找公共后缀，若有公共后缀，则压缩 并返回指向公共后缀的指针；否则返回NULL

## 预设代码

### 前置代码

[view plaincopy to clipboardprint?](http://cms.bit.edu.cn/moodle/mod/programming/view.php?id=19770)

1. /\* PRESET CODE BEGIN - NEVER TOUCH CODE BELOW \*/
3. #include <stdio.h>
4. #include <stdlib.h>
6. typedef struct sdata
7. {  char data;
8. struct sdata \*next;
9. } SNODE;
11. void setlink( SNODE \*, char \* ), outlink( SNODE \* );
12. int listlen( SNODE \* );
13. SNODE \* ziplist( SNODE \*, SNODE \* );
14. SNODE \* findlist( SNODE \*, SNODE \* );
16. int main( )
17. {
18. SNODE \* head1, \* head2, \*head;
19. char str1[100], str2[100];
21. gets( str1 );
22. gets( str2 );
24. head1 = (SNODE \*) malloc( sizeof(SNODE) );
25. head2 = (SNODE \*) malloc( sizeof(SNODE) );
26. head = (SNODE \*) malloc( sizeof(SNODE) );
27. head->next = head1->next = head2->next = NULL;
29. setlink( head1, str1 );
30. setlink( head2, str2);
32. head->next = ziplist( head1, head2 );
34. head->next = findlist( head1, head2 );
35. outlink( head );
36. return 0;
37. }
39. void setlink( SNODE \*head, char \*str )
40. {
41. SNODE \*p ;
43. while ( \*str != '\0' )
44. {   p  = ( SNODE \* ) malloc( sizeof( SNODE ) );
45. p->data = \*str;
46. p->next = NULL;
47. str++;
48. head->next = p;
49. head = p;
50. }
51. return;
52. }
54. void outlink( SNODE \* head )
55. {
56. while ( head->next != NULL )
57. {
58. printf( "%c", head -> next -> data );
59. head = head -> next;
60. }
61. printf("\n");
62. return;
63. }
65. int listlen( SNODE \* head )
66. {
67. int len=0;
68. while ( head->next != NULL )
69. {
70. len ++;
71. head = head->next;
72. }
73. return len;
74. }
76. SNODE \* findlist( SNODE \* head1, SNODE \* head2 )
77. {
78. int m, n;
79. SNODE \*p1=head1, \*p2=head2;
81. m = listlen( head1 );
82. n = listlen( head2 );
84. while  ( m > n )
85. {   p1 = p1->next;
86. m--;
87. }
88. while  ( m < n )
89. {   p2 = p2->next;
90. n--;
91. }
93. while( p1->next != NULL && p1->next != p2->next )
94. {
95. p1 = p1->next;
96. p2 = p2->next;
97. }
98. return p1->next;
99. }
101. /\* Here is waiting for you!       \*/
102. /\*
103. SNODE \* ziplist( SNODE \* head1, SNODE \* head2 )
104. {
105. }
106. \*/
108. /\* PRESET CODE END - NEVER TOUCH CODE ABOVE \*/

SNODE \* ziplist( SNODE \* head1, SNODE \* head2 )

{

int m, n;

SNODE \*p1=head1, \*p2=head2,\*p11=NULL,\*p22=NULL;

m = listlen( head1 );

n = listlen( head2 );

while ( m > n )

{ p1 = p1->next;

m--;

}

while ( m < n )

{ p2 = p2->next;

n--;

}

p11=p1;

p22=p2;

while(p1->next->next!=NULL)

{

if(p1->next->data!=p2->next->data)

{

p11=p1->next;

p22=p2->next;

}

p1=p1->next;

p2=p2->next;

}

if(p1->next->data!=p2->next->data)

return NULL;

else

{

p22->next=p11->next;

return p11->next;

}

}

# 4. 括号匹配（10分）

成绩: 10 / 折扣: 0.8

4 括号匹配 （10分）

成绩: 10 / 折扣: 0.8

 假设一个算术表达式中包含圆括号、方括号两种类型的括号，试编写一个判断表达式中括号是否匹配的程序，匹配返回Match succeed!，否则返回Match false!。

例

[1+2\*(3+4\*(5+6))]     括号匹配

(1+2)\*(1+2\*[(1+2)+3)  括号不匹配

## 输入

包含圆括号、方括号两种类型括号的算术表达式

## 输出

匹配输出    Match succeed!

不匹配输出  Match false!

例

输入  [1+2\* (3+4\*(5+6))]

输出 Match succeed!

#include<stdio.h>

int main()

{

int flag=0;

char a[1000]={0};

char\* p;

p=&a[0];

char temp;

temp=getchar();

\*p=temp;

while(temp!='\n')

{

switch (temp)

{

case '(':

{

p++;

\*p=temp;

break;

}

case ')':

{

if(\*p!='(')

{

printf("Match false!\n");

return 0;

}

\*p=0;

p--;

break;

}

case '[':

{

p++;

\*p=temp;

break;

}

case']':

{

if(\*p!='[')

{

printf("Match false!\n");

return 0;

}

\*p=0;

p--;

break;

}

}//endswiych

temp=getchar();

}//end whilw

printf("Match succeed!\n");

return 0;

}

# 5. 迷宫问题（15分）

成绩: 15 / 折扣: 0.8

5  迷宫问题（15分）

成绩: 15 / 折扣: 0.8

迷宫有一个入口，一个出口。一个人从入口走进迷宫，目标是找到出口。阴影部分和迷宫的外框为墙，每一步走一格，每格有四个可走的方向，探索顺序为：南、东、北、西。

## 迷宫示意图

## 输入：输入迷宫数组

## 输出：若有解，输出从入口到出口的一条路径，否则输出 there is no solution!

## 例（上图所示的迷宫数组）

输入

4 4

0 0 1 0

0 1 0 1

0 0 0 0

0 1 0 0

输出

<1,1> <2,1> <3,1> <3,2> <3,3> <4,3> <4,4>

#include<iostream>

using std::cin;

using std::cout;

using std::endl;

int main()

{

int a,b;

cin>>a>>b;

bool date[102][102];

for(int i=0;i<102;i++)

for (int j=0;j<102;j++)

date[i][j]=1;

int stack[500][4]={0};

int p=1;

stack[0][2]=5;

stack[1][0]=1;

stack[1][1]=1;

stack[1][3]=5;

for(int x=1;x<=a;x++)//input start

{

for(int y=1;y<=b;y++)

{

bool temp;

cin>>temp;

date[x][y]=temp;

}

}//input finish

int p1,p2;

while(!(stack[p][0]==a&&stack[p][1]==b))//find start

{

switch (stack[p][2])

{

case 0://down

{

if(stack[p][2]==stack[p][3])

{

stack[p][2]++;

break;

}

p1=stack[p][0]+1;

p2=stack[p][1];

if(date[p1][p2])//wall

{

stack[p][2]++;

goto x1;

}

else//road

{

stack[p][2]++;

p++;

stack[p][0]=p1;

stack[p][1]=p2;

stack[p][3]=2;

break;

}

}

case 1://right

{

x1: if(stack[p][2]==stack[p][3])

{

stack[p][2]++;

break;

}

p1=stack[p][0];

p2=stack[p][1]+1;

if(date[p1][p2])//wall

{

stack[p][2]++;

goto x2;

}

else//road

{

stack[p][2]++;

p++;

stack[p][0]=p1;

stack[p][1]=p2;

stack[p][3]=3;

break;

}

}

case 2://up

{

x2: if(stack[p][2]==stack[p][3])

{

stack[p][2]++;

break;

}

p1=stack[p][0]-1;

p2=stack[p][1];

if(date[p1][p2])//wall

{

stack[p][2]++;

goto x3;

}

else//road

{

stack[p][2]++;

p++;

stack[p][0]=p1;

stack[p][1]=p2;

stack[p][3]=0;

break;

}

}

case 3://left

{

x3: if(stack[p][2]==stack[p][3])

{

stack[p][2]++;

break;

}

p1=stack[p][0];

p2=stack[p][1]-1;

if(date[p1][p2])//wall

{

stack[p][2]++;

goto x4;

}

else//road

{

stack[p][2]++;

p++;

stack[p][0]=p1;

stack[p][1]=p2;

stack[p][3]=1;

break;

}

}

case 4://back

{

x4: stack[p][2]=0;

p--;

break;

}

case 5:

{

cout<<"there is no solution!\n";

return 0;

}

}

}//find finish

p=1;

while(stack[p][2])

{

cout<<"<"<<stack[p][0]<<","<<stack[p][1]<<"> ";

p++;

}

cout<<"<"<<stack[p][0]<<","<<stack[p][1]<<"> "<<endl;

return 0;

}

# 6. 飞机场调度(15分)

成绩: 15 / 折扣: 0.8

在本实验中，需要同学们利用队列实现一个飞机场调度模拟，根据不同的输入参数得到不同的模拟结果。程序运行开始，首先需要输入以下参数：

*机场跑道数，飞机降落占用跑道时间（整数）， 飞机起飞占用跑道时间（整数）*

整个模拟的时间以分钟为单位，从 0 开始，每分钟的开始需要输入：

*该分钟要求降落飞机数， 该分钟要求起飞飞机数*

机场调度原则是降落优先起飞，在此原则下按来的顺序排队；每驾飞机都有一个编号，要起飞飞机从 1 开始，要降落飞机从 5001 开始；每驾飞机需要等待的时间是从其提要求开始到分配跑道为止；每个跑道都有一个编号（从 1 开始），都可以用来降落和起飞，但同一时间只能被一架飞机占用，占用时间为该飞机降落（起飞）占用跑道时间。

当输入的要求降落飞机数和要求起飞飞机数都小于 0 时，表示机场关闭，不再接受新的请求，但余下没有降落（起飞）的飞机需照常进行。

模拟过程中需要随时输出以下数据：

1. 当前时间 (%4d)

2. 所有从占用变为空闲的跑道编号 （在输入降落、起飞飞机数前输出）

3. 可以降落（起飞）飞机编号（％ 04d ）、跑道编号（％ 02d ） （在输入降落、起飞飞机数后输出）

模拟结束后，程序需输出以下统计结果：

1. 模拟时间（％ 4d ）

2. 降落平均等待时间（％ 4.1f ）

3. 起飞平均等待时间（％ 4.1f ）

4. 每条跑道被占用时间（％ 4d ）

5. 跑道平均被占用的百分比（％ 4.1f ， 平均占用时间× 100/ 模拟时间）

**例：** （下面的黑斜体为输入）

**4 3 5**

Current Time: 0

**1 4**

airplane 5001 is ready to land on runway 01

airplane 0001 is ready to takeoff on runway 02

airplane 0002 is ready to takeoff on runway 03

airplane 0003 is ready to takeoff on runway 04

Current Time: 1

**0 0**

Current Time: 2

**0 2**

Current Time: 3

runway 01 is free

**3 0**

airplane 5002 is ready to land on runway 01

Current Time: 4

**0 0**

Current Time: 5

runway 02 is free

runway 03 is free

runway 04 is free

**0 0**

airplane 5003 is ready to land on runway 02

airplane 5004 is ready to land on runway 03

airplane 0004 is ready to takeoff on runway 04

Current Time: 6

runway 01 is free

**2 4**

airplane 5005 is ready to land on runway 01

Current Time: 7

**-1 -1**

Current Time: 8  
runway 02 is free  
runway 03 is free  
airplane 5006 is ready to land on runway 02  
airplane 0005 is ready to takeoff on runway 03  
Current Time: 9  
runway 01 is free  
airplane 0006 is ready to takeoff on runway 01  
Current Time: 10  
runway 04 is free  
airplane 0007 is ready to takeoff on runway 04  
Current Time: 11  
runway 02 is free  
airplane 0008 is ready to takeoff on runway 02  
Current Time: 12  
Current Time: 13  
runway 03 is free  
airplane 0009 is ready to takeoff on runway 03  
Current Time: 14  
runway 01 is free  
airplane 0010 is ready to takeoff on runway 01  
Current Time: 15  
runway 04 is free  
Current Time: 16  
runway 02 is free  
Current Time: 17  
Current Time: 18  
runway 03 is free  
Current Time: 19  
runway 01 is free  
simulation **finished**  
simulation time: 19  
average waiting time of landing: 1.0  
average waiting time of takeoff: 4.2  
runway 01 busy time: 19  
runway 02 busy time: 16  
runway 03 busy time: 18  
runway 04 busy time: 15  
runway average busy time percentage: 89.5%

* /

#include<stdio.h>

#include<stdlib.h>

int time,wayn,downtime,uptime,up,down,upwaiting,downwaiting,upcount,downcount,use;

int\* way;

float\* waytime;

int manage()

{

for(int i=1;i<=wayn;i++)

{

if(\*(way+i)==0)

{

use--;

if(downwaiting>0)

{

\*(way+i)=downtime;

(\*(waytime+i))+=downtime;

printf("airplane %04d is ready to land on runway %02d\n",downcount++,i);

downwaiting--;

use++;

}

else if(upwaiting>0)

{

\*(way+i)=uptime;

(\*(waytime+i))+=uptime;

printf("airplane %04d is ready to takeoff on runway %02d\n",upcount++,i);

upwaiting--;

use++;

}

else

\*(way+i)=-1;

}

else if(\*(way+i)<0)

{

if(downwaiting>0)

{

\*(way+i)=downtime;

(\*(waytime+i))+=downtime;

printf("airplane %04d is ready to land on runway %02d\n",downcount++,i);

downwaiting--;

use++;

}

else if(upwaiting>0)

{

\*(way+i)=uptime;

(\*(waytime+i))+=uptime;

printf("airplane %04d is ready to takeoff on runway %02d\n",upcount++,i);

upwaiting--;

use++;

}

}

}//end of for

return 0;

}

int main()

{

float avewaitingup=0;

float avewaitingdown=0;

scanf("%d%d%d",&wayn,&downtime,&uptime);

way=(int\*) malloc (sizeof(int)\*(wayn+1));

waytime=(float\*) malloc (sizeof(float)\*(wayn+1));

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

{

\*(way+i)=-1;

\*(waytime+i)=0;

}

downcount=5001;

upcount=1;

use=0;

printf("Current Time: 0\n");

scanf("%d%d",&down,&up);

for(time=1;up>=0&&down>=0;)

{

upwaiting+=up;

downwaiting+=down;

manage();

avewaitingup+=upwaiting;

avewaitingdown+=downwaiting;

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

(\*(way+i))--;

printf("Current Time: %4d\n",time++);

for(int i=1;i<=wayn;i++)

{

if(\*(way+i)==0)

{

printf("runway %02d is free\n",i);

}

}

scanf("%d%d",&down,&up);

}

manage();

avewaitingup+=upwaiting;

avewaitingdown+=downwaiting;

while(use)

{

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

(\*(way+i))--;

printf("Current Time: %4d\n",time++);

for(int i=1;i<=wayn;i++)

{

if(\*(way+i)==0)

{

printf("runway %02d is free\n",i);

}

}

manage();

avewaitingup+=upwaiting;

avewaitingdown+=downwaiting;

}

time--;

//finish

printf("simulation finished\nsimulation time: %4d\n",time);

float aa=avewaitingdown/(downcount-5000-1),bb=avewaitingup/(upcount-1),cc=0;

if(avewaitingdown==0||downcount==5001)

aa=0;

if(avewaitingup==0||upcount==1)

bb=0;

printf("average waiting time of landing: %4.1f\naverage waiting time of takeoff: %4.1f\n",aa,bb);

float all=0;

int i=1;

for(;i<=wayn;i++)

{

printf("runway %02d busy time: %4.0f\n",i,\*(waytime+i));

all+=\*(waytime+i);

}

if(all==0||time==0||wayn==0)

cc=0;

else

cc=all/wayn/time\*100;

printf("runway average busy time percentage: %4.1f%%\n",cc);

return 0;

}

# 7. 股票撮合系统(15分)

成绩: 15 / 折扣: 0.8

在股票交易中，股民可以通过各种手段将委托送到股票交易所。每个委托主要说明了股民身份、买卖的股票、价格和数量。交易的规则是价格优先、时间优先，即出的价格最高的人先买，出的价格最低的人先卖。两个委托只有价格合适时才能成交，未成交的委托按价格顺序放在撮合队列中。每个股票有两个撮合队列：买队列和卖队列。只有当买委托的价格高于等于卖委托的价格，两个委托才可以成交，成交价取两个委托价格的平均值，成交量取两个委托数量的最小值。委托可以是完全成交或部分成交，部分成交的委托保留在撮合队列中继续交易。试利用单链表作为存放委托的数据结构（撮合队列），编写一模拟股票交易的程序，该程序有以下几个功能：

1. 委托申请：

输入：每个委托包括四个数据项，股票编码（ 4 位数字）、价格（浮点数）、数量（整数）、买 / 卖（ B/S ）

输出： a. 程序为每个委托产生一个唯一的序号（ %04d ），该序号从 1 开始； b. 每笔成交包括：成交价格（ %6.1f ）、成交量（ %4d ）、新的委托序号（ %04d ）、匹配的委托序号（ %04d ）。

2. 查询未成交的委托：

输入：股票编码

输出：按撮合队列中委托的顺序，分别输出该股票未成交的委托，每个输出的委托包括：委托序号（ %04d ）、 股票编码 （ %04d ） 、 价格（ %6.1f ）、数量（ %4d ）、 B/S （买 / 卖 )

3. 委托撤消：

输入：要撤消的委托号。

输出：若成功，显示该委托信息，其中委托包括数据项：委托序号、股票编码、价格、数量、 B/S （买 / 卖 ) ；否则显示“ not found ”失败信息。

**委托输入格式** ： 1 股票编码 价格 数量 买卖

**查询输入格式** ： 2 股票编码

**委托撤销** ： .3 委托号

**退出：** 0

**例：** （下面的黑斜体为输入,    输入输出格式参见测试用例）

***1 0038 20 1000 b***

*orderid: 0001*

***1 0278 18 2000 s***

*orderid: 0002*

***1 0003 8 5000 b***

*orderid: 0003*

***1 0003 12 1000 b***

*orderid: 0004*

***1 0003 10 500 b***

*orderid: 0005*

***1 0003 11 9000 b***

*orderid: 0006*

***1 0003 18 1000 s***

*orderid: 0007*

***2 0003***

*buy orders:*

*orderid: 0004, stockid:0003, price: 12.0, quantity: 1000, b/s: b*

*orderid: 0006, stockid:0003, price: 11.0, quantity: 9000, b/s: b*

*orderid: 0005, stockid:0003, price: 10.0, quantity: 500, b/s: b*

*orderid: 0003, stockid:0003, price: 8.0, quantity: 5000, b/s: b*

*sell orders:*

*orderid: 0007, stockid:0003, price: 18.0, quantity: 1000, b/s: s*

***3 0006***

*deleted order:orderid: 0006, stockid:0003, price: 11.0, quantity: 9000, b/s: b*

***3 0197***

*not found*

***2 0003***

*buy orders:*

*orderid: 0004, stockid:0003, price: 12.0, quantity: 1000, b/s: b*

*orderid: 0005, stockid:0003, price: 10.0, quantity: 500, b/s: b*

*orderid: 0003, stockid:0003, price: 8.0, quantity: 5000, b/s: b*

*sell orders:*

*orderid: 0007, stockid:0003, price: 18.0, quantity: 1000, b/s: s*

***1 0003 9 1200 s***

*orderid: 0008*

*deal--price: 10.5 quantity:1000 sellorder:0008 buyorder:0004*

*deal--price: 9.5 quantity: 200 sellorder:0008 buyorder:0005*

***2 0003***

*buy orders:*

*orderid: 0005, stockid:0003, price: 10.0, quantity: 300, b/s: b*

*orderid: 0003, stockid:0003, price: 8.0, quantity: 5000, b/s: b*

*sell orders:*

*orderid: 0007, stockid:0003, price: 18.0, quantity: 1000, b/s: s*

（注意：当查询未成交委托时，如果没有委托，仍要输出“buy orders:”或“sell orders: ”）

#include<stdio.h>

#include<stdlib.h>

int count;

struct date

{

int orderid;

int stockid;

float price;

int quantity;

int bs;

struct date\* next;

};

typedef struct date DATE;

typedef struct date\* PDATE;

struct stock

{

int stockid;

struct stock\* next;

struct date\* b;

struct date\* s;

};

typedef struct stock STOCK;

typedef struct stock\* PSTOCK;

PDATE adress[10000];

PSTOCK ad[10000];

PSTOCK home;

PDATE setnew(PDATE p0,int stockid,float price,int quantity,int bs)

{

PDATE p;

p=(PDATE) malloc (sizeof(DATE));

p->bs=bs;

if(p0!=NULL)

{

p->next=p0->next;

p0->next=p;

}

else

p->next=NULL;

p->orderid=count++;

adress[p->orderid]=p;

p->price=price;

p->quantity=quantity;

p->stockid=stockid;

return p;

}

PSTOCK setnewstock(int stockid)

{

PSTOCK p=NULL;

p=(PSTOCK)malloc(sizeof(STOCK));

ad[stockid]=p;

p->b=NULL;

p->s=NULL;

p->stockid=stockid;

return p;

}

PSTOCK insert(PSTOCK home,int stockid,float price,int bs,int quantity)

{

PDATE p2;

if(ad[stockid]==NULL)

setnewstock(stockid);

PDATE p1=NULL;

if(bs==1)//大to小

{

p1=ad[stockid]->b;

if(p1==NULL)

{

ad[stockid]->b=setnew(NULL,stockid,price,quantity,bs);

return ad[stockid];

}

else

{

while(1)

{

if(p1->price<price)

{

p2=setnew(p1,p1->stockid,p1->price,p1->quantity,p1->bs);

p1->bs=bs;

int tempid;

tempid=p2->orderid;

p2->orderid=p1->orderid;

p1->orderid=tempid;

p1->price=price;

p1->quantity=quantity;

adress[p2->orderid]=p2;

adress[p1->orderid]=p1;

return ad[stockid];

}

else if(p1->next==NULL)

{

setnew(p1,stockid,price,quantity,bs);

return ad[stockid];

}

p1=p1->next;

}

}

}

else//小to大

{

p1=ad[stockid]->s;

if(p1==NULL)

{

ad[stockid]->s=setnew(NULL,stockid,price,quantity,bs);

return ad[stockid];

}

else

{

while(1)

{

if(p1->price>price)

{

p2=setnew(p1,p1->stockid,p1->price,p1->quantity,p1->bs);

p1->bs=bs;

int tempid;

tempid=p2->orderid;

p2->orderid=p1->orderid;

p1->orderid=tempid;

p1->price=price;

p1->quantity=quantity;

adress[p2->orderid]=p2;

adress[p1->orderid]=p1;

return ad[stockid];

}

else if(p1->next==NULL)

{

setnew(p1,stockid,price,quantity,bs);

return ad[stockid];

}

p1=p1->next;

}

}

}

}

PDATE del(PDATE p)

{

adress[p->orderid]=NULL;

if(p->next==NULL)

{

PSTOCK p0=home;

p0=ad[p->stockid];

PDATE p1=NULL;

if(p->bs==1)

{

p1=p0->b;

if(p1==p)

{

p0->b=NULL;

free(p);

}

else

{

while(p1->next!=p)

p1=p1->next;

p1->next=NULL;

free(p);

}

}

else

{

p1=p0->s;

if(p1==p)

{

p0->s=NULL;

free(p);

}

else

{

while(p1->next!=p)

p1=p1->next;

p1->next=NULL;

free(p);

}

}

return NULL;

}

PDATE pn=p->next;

p->bs=pn->bs;

p->next=pn->next;

p->orderid=pn->orderid;

p->price=pn->price;

p->quantity=pn->quantity;

p->stockid=pn->stockid;

free(pn);

adress[p->orderid]=p;

return p;

}

int fun1(PSTOCK p,int bs,int quantity)

{

printf("orderid: %04d\n",count-1);

PDATE ps,pb;

ps=p->s;

pb=p->b;

if(!(ps!=NULL&&pb!=NULL))

goto z;

while(ps->price<=pb->price)//deal--price: 10.5 quantity:1000 sellorder:0008 buyorder:0004

{

if(ps->quantity>pb->quantity)

{

if(bs==1)

printf("deal--price:%6.1f quantity:%4d buyorder:%04d sellorder:%04d\n",(ps->price+pb->price)/2.0,pb->quantity,pb->orderid,ps->orderid);

else

printf("deal--price:%6.1f quantity:%4d sellorder:%04d buyorder:%04d\n",(ps->price+pb->price)/2.0,pb->quantity,ps->orderid,pb->orderid);

ps->quantity-=pb->quantity;

pb=del(pb);

}

else if(ps->quantity<pb->quantity)

{

if(bs==1)

printf("deal--price:%6.1f quantity:%4d buyorder:%04d sellorder:%04d\n",(ps->price+pb->price)/2.0,ps->quantity,pb->orderid,ps->orderid);

else

printf("deal--price:%6.1f quantity:%4d sellorder:%04d buyorder:%04d\n",(ps->price+pb->price)/2.0,ps->quantity,ps->orderid,pb->orderid);

pb->quantity-=ps->quantity;

ps=del(ps);

}

else

{

if(bs==1)

printf("deal--price:%6.1f quantity:%4d buyorder:%04d sellorder:%04d\n",(ps->price+pb->price)/2.0,ps->quantity,pb->orderid,ps->orderid);

else

printf("deal--price:%6.1f quantity:%4d sellorder:%04d buyorder:%04d\n",(ps->price+pb->price)/2.0,ps->quantity,ps->orderid,pb->orderid);

pb=del(pb);

ps=del(ps);

}

if(ps==NULL||pb==NULL)

break;

}

z: if(quantity==0)

{

PDATE pdel=((bs==1)?p->b:p->s);

while(pdel!=NULL)

if(pdel->quantity==0)

break;

else

pdel=pdel->next;

if(pdel!=NULL)

del(pdel);

}

return 0;

}

void fun2(PSTOCK p)//orderid: 0005, stockid:0003, price: 10.0, quantity: 300, b/s: b

{

PDATE pt;

pt=p->b;

printf("buy orders:\n");

while(pt!=NULL)

{

printf("orderid: %04d, stockid:%04d, price: %6.1f, quantity: %4d, b/s: %c\n",pt->orderid,pt->stockid,pt->price,pt->quantity,pt->bs==1?'b':'s');

pt=pt->next;

}

pt=p->s;

printf("sell orders:\n");

while(pt!=NULL)

{

printf("orderid: %04d, stockid:%04d, price: %6.1f, quantity: %4d, b/s: %c\n",pt->orderid,pt->stockid,pt->price,pt->quantity,pt->bs==1?'b':'s');

pt=pt->next;

}

}

void fun3(PDATE p)

{

if(p==NULL)

{

printf("not found\n");

}

else

{

printf("deleted order:orderid: %04d, stockid:%04d, price: %6.1f, quantity: %4d, b/s: %c\n",p->orderid,p->stockid,p->price,p->quantity,(p->bs==1)?'b':'s');

del(p);

}

}

int main()

{

char choice='\n';

char ctemp;

count=1;

for(int i=0;i<1000;i++)

{

adress[i]=NULL;

ad[i]=NULL;

}

PSTOCK p0t;

home=setnewstock(0);

DATE temp;

while(choice)

{

switch (choice)

{

case '1':

{

scanf("%d%f%d %c",&temp.stockid,&temp.price,&temp.quantity,&ctemp);

temp.bs=(ctemp=='b'?1:2);

p0t=insert(home,temp.stockid,temp.price,temp.bs,temp.quantity);

fun1(p0t,temp.bs,temp.quantity);

break;

}

case '2':

{

scanf("%d",&temp.stockid);

p0t=ad[temp.stockid];

if(p0t==NULL)

{

printf("buy orders:\n");

printf("sell orders:\n");

goto x;

}

fun2(p0t);

break;

}

case '3':

{

int tempid;

scanf("%d",&tempid);

fun3(adress[tempid]);

break;

}

case '0':

{

return 0;

}

}

x: choice=getchar();

}

}

# 8. 二叉树的建立与基本操作(10分)

成绩: 10 / 折扣: 0.8

程序输入是一个（扩展）二叉树的先序序列。例如：扩展二叉树先序序列为ab#d##ce###。其中#代表空指针。

编写程序实现二叉树的如下操作：  
1） 建立二叉链表  
2）二叉树的先序、中序、后序遍历  
3）求解二叉树的叶子结点个数  
4）将二叉树中所有结点的左、右子树相互交换

**程序的输出**

　　二叉树的凹入表示，二叉树的先序序列、中序序列、后序序列，二叉树的叶子结点个数，左、右子树相互交换后的二叉树的凹入表示, 左、右子树相互交换后的二叉树的先序序列、中序序列、后序序列。

　　在输出凹入表示的二叉树时，先输出根结点，然后依次输出左右子树，上下层结点之间相隔3 个空格，如：针对上面给出的输入，输出的内容应为：

# BiTree a b d c e pre\_sequence: abdce in\_sequence: bdaec post\_sequence : dbeca Number of leaf: 2 BiTree swapped a c e b d pre\_sequence: acebd in\_sequence: ceadb post\_sequence : ecdba

#include<stdio.h>

#include<stdlib.h>

typedef struct bt

{

char d;

struct bt\* r;

struct bt\* l;

}BT;

int leaf;

BT\* creataBT(char d)

{

BT\* p=NULL;

p=(BT\*)malloc(sizeof(BT));

p->d=d;

p->l=NULL;

p->r=NULL;

return p;

}

int creatBT(BT\* root)

{

char temp;

//left

scanf("%c",&temp);

if(temp=='#')

root->l=NULL;

else

{

root->l=creataBT(temp);

creatBT(root->l);

}

//right

scanf("%c",&temp);

if(temp=='#')

root->r=NULL;

else

{

root->r=creataBT(temp);

creatBT(root->r);

}

return 0;

}

int printtree(BT\* root,int rank)

{

if(root==NULL)

return 0;

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

printf(" ");

printf("%c\n",root->d);

printtree(root->l,rank+1);

printtree(root->r,rank+1);

return 0;

}

int printper(BT\* head)

{

if(head==NULL)

return 0;

printf("%c",head->d);

printper(head->l);

printper(head->r);

return 0;

}

int printin(BT\* head)

{

if(head==NULL)

return 0;

printin(head->l);

printf("%c",head->d);

printin(head->r);

return 0;

}

int printpost(BT\* head)

{

if(head==NULL)

return 0;

printpost(head->l);

printpost(head->r);

printf("%c",head->d);

return 0;

}

int count(BT\* head)

{

if(head->l==NULL&&head->r==NULL)

leaf++;

if((head->l))

count(head->l);

if((head->r))

count(head->r);

return 0;

}

BT\* swapped(BT\* head)

{

BT\* p=NULL;

if(head==NULL)

return NULL;

p=head->l;

head->l=head->r;

head->r=p;

swapped(head->r);

swapped(head->l);

return NULL;

}

int main()

{

BT\* head=NULL;

//creat

char temp;

scanf("%c",&temp);

head=creataBT(temp);

creatBT(head);

//manage

printf("BiTree\n");

printtree(head,0);

printf("pre\_sequence : ");

printper(head);

printf("\n");

printf("in\_sequence : ");

printin(head);

printf("\n");

printf("post\_sequence : ");

printpost(head);

printf("\n");

leaf=0;

count(head);

printf("Number of leaf: %d\n",leaf);

BT\* heads=NULL;

heads=swapped(head);

heads=head;

printf("BiTree swapped\n");

printtree(heads,0);

printf("pre\_sequence : ");

printper(heads );

printf("\n");

printf("in\_sequence : ");

printin(heads);

printf("\n");

printf("post\_sequence : ");

printpost(heads);

printf("\n");

return 0;

}

# 9. 树的建立与基本操作(15分)

成绩: 15 / 折扣: 0.8

在本实验中，程序的输入是一个表示树结构的广义表。假设树的根为 root ，其子树森林 F ＝ （ T1 ， T2 ， … ， Tn ），设与该树对应的广义表为 L ，则 L ＝（原子，子表 1 ，子表 2 ， … ，子表 n ），其中原子对应 root ，子表 i （ 1<i<=n ）对应 Ti 。例如：广义表 (a,(b,(c),(d)),(f,(g),(h ),(i))) 表示的树如图所示：

程序的输出为树的层次结构、树的度以及各种度的结点个数。

在输出树的层次结构时，先输出根结点，然后依次输出各个子树，每个子树向里缩进 4 个空格，如：针对上图表示的树，输出的内容应为：

a

    b

        c

        d

    f

        g

        h

        i

Degree of tree: 3

Number of nodes of degree 0: 5

Number of nodes of degree 1: 0

Number of nodes of degree 2: 2

Number of nodes of degree 3: 1

**例：** （下面的黑体为输入）

**(a,(b),(c,(d),(e,(g),(h )),(f)))**

a

    b

    c

        d

        e

            g

            h

        f

Degree of tree: 3

Number of nodes of degree 0: 5

Number of nodes of degree 1: 0

Number of nodes of degree 2: 2

Number of nodes of degree 3: 1

#include<stdio.h>

int main()

{

char words;

int rank=0;

int ranktemp=0;

int form[101]={0};

int stack[20]={0};

int pstack=0;

int num=0;

do

{

scanf("%c",&words);

switch(words)

{

case '(':

{

rank++;

break;

}

case ')':

{

rank--;

break;

}

case ',':

break;

default:

{

num++;

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

printf(" ");

printf("%c\n",words);

if(rank==ranktemp)

form[num]=stack[pstack];

else if(rank>ranktemp)

{

pstack++;

stack[pstack]=num-1;

form[num]=stack[pstack];

ranktemp=rank;

}

else if(rank<ranktemp)

{

pstack--;

form[num]=stack[pstack];

ranktemp=rank;

}

}

}

}

while(rank);

//统计

int formnet[101]={0};

for(int i=1;i<=num;i++)

{

formnet[form[i]]++;

}

int net[101]={0};

int max=0;

for(int i=1;i<=num;i++)

{

if(formnet[i]>max)

max=formnet[i];

net[formnet[i]]++;

}

printf("Degree of tree: %d\n",max);

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

{

printf("Number of nodes of degree %d: %d\n",i,net[i]);

}

return 0;

}

# 10. 博弈树 (15分)

成绩: 15 / 折扣: 0.8

下棋属于一种博弈游戏，博弈过程可以用树（博弈树）来表示。假设游戏由两个人（ A 和 B ）玩，开始由某个人从根结点开始走，两个人轮流走棋，每次只能走一步， 下一步棋只能选择当前结点的孩子结点，谁先走到叶子结点为胜。例如，对于下图所示的博弈树，若 A 先走，可以选 f ， B 若选 h ，则 A 选 j 胜。

编写一程序，让计算机和人下棋。当计算机走下一步时，可以根据以下情况决定下一步：

（1） 若存在可以确保取胜的一个孩子结点，则选择该结点作为下一步；

（2） 若存在多个可以确保取胜的孩子结点，则选择其中高度最小的结点作为下一步（若有多个选择，则选最左边的结点）；

（3） 若不存在可以确保取胜的一个孩子结点，则选择高度最大的孩子结点作为下一步（若有多个选择，则选最左边的结点）；

**例：** （下面的黑体为输入）

**(a,(b,(x)),(c,(d),(e,(g),heart),(f)))**

a

b

x

c

d

e

g

h

f

Who play first(0: computer; 1: player )?

**1**

player:

**c**

computer: d

Sorry, you lost.

Continue(y/n)?

**y**

Who play first(0: computer; 1: player )?

**1**

player:

**x**

illegal move.

player:

**b**

computer: x

Sorry, you lost.

Continue(y/n)?

**y**

Who play first(0: computer; 1: player )?

**0**

computer: c

player:

**f**

Congratulate, you win.

Continue(y/n)?

**n**

#include<stdio.h>

#include<stdlib.h>

typedef struct node

{

char date;

char mytype;//1 mustlose 2 mustwin 3 donot know

char comtype;//1mustlose 2mustwin 3donotknow

int high;

struct node\* bro;

struct node\* chi;

struct node\* par;

}NODE;

typedef struct node \* PNODE;

PNODE setnew(char date,PNODE bro,PNODE par)

{

PNODE p;

p=(PNODE) malloc (sizeof(NODE));

p->chi=NULL;

p->date=date;

p->bro=NULL;

p->par=par;

p->high=0;

p->mytype=0;

p->comtype=0;

if(par->chi==NULL)

par->chi=p;

else

bro->bro=p;

return p;

}

PNODE inputtree()

{

PNODE home;//set head

home=(PNODE)malloc(sizeof(NODE));

home->chi=NULL;

home->date=0;

home->bro=NULL;

home->par=NULL;

home->high=-1;

home->mytype=-1;

home->comtype=-1;

home->bro=NULL;

PNODE p=home;

PNODE stack[100];

stack[1]=home;

stack[0]=home;

PNODE pbro=home;

char temp=',';

int count=0;

do

{

scanf("%c",&temp);

switch (temp)

{

case ',':

break;

case '(':

{

count++;

stack[count]=pbro;

pbro=stack[count]->chi;

if(pbro)

while(pbro->bro!=NULL)

pbro=pbro->bro;

scanf("%c",&temp);

pbro=setnew(temp,pbro,stack[count]);

break;

}

case ')':

{

count--;

stack[count+1]=NULL;

pbro=stack[count]->chi;

while(pbro->bro!=NULL)

pbro=pbro->bro;

break;

}

default:

{

break;

}

}

}

while(count);

return home;

}

int rank;

int print(PNODE home)

{

PNODE p=home;

while(p!=NULL)

{

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

printf(" ");

printf("%c\n",p->date);

rank++;

print(p->chi);

rank--;

p=p->bro;

}

return 0;

}

int back(PNODE p0)

{

PNODE p=p0,pt=NULL;

for(int i=1;p->par;p=p->par)

{

if(p->high<i)

p->high=i++;

if(p->chi==NULL)

{

p->mytype=2;

p->comtype=2;

}

else

{

int flag[4]={0};

pt=p->chi;

while(pt)

{

flag[pt->mytype]++;

pt=pt->bro;

}

if(flag[0])

goto z;

if(!(flag[2]+flag[3]))//all 1

{

p->comtype=2;

p->mytype=2;

}

else if(!(flag[1]+flag[3]))//all 2

{

p->comtype=1;

p->mytype=1;

}

else if(flag[2])//have 2

{

p->comtype=1;

p->mytype=3;

}

else

{

p->comtype=3;

p->mytype=3;

}

}//else

z: ;

}//for

return 0;

}

int addjudge(PNODE p0)

{

int t=1;

PNODE p=p0->chi;

while(p)

{

if(p->comtype!=1)

t=0;

p=p->bro;

}

return t;

}

void manage(PNODE home)

{

PNODE p=home;

while(p)

{

if(!p->chi)

back(p);

else

manage(p->chi);

p=p->bro;

}

}

int start(PNODE head,int choice)

{

char temp,t;

PNODE p;

x: p=head;

if(choice)//1 playerfirst

{

printf("player:\n");

scanf("%c",&temp);

t=getchar();

p=p->chi;

while(p->date!=temp)

{

p=p->bro;

if(!p)

{

printf("illegal move.\n");

goto x;

}

}

if(p->chi==NULL)

{

printf("Congratulate, you win.\n");

return 0;

}

}

p=p->chi;

//1finish

//computer

while(1)

{

int a=3,hmax=0,hmin=60000;

PNODE pt=p;

while(p)

{

switch (p->mytype)

{

case 1:case 3:

{

if(addjudge(p))

goto w;

if(a==2)

break;

if(hmax<p->high)

{

pt=p;

hmax=p->high;

}

break;

}

case 2:

{

w: a=2;

if(hmin>p->high)

{

pt=p;

hmin=p->high;

}

break;

}

}

p=p->bro;

}//while

p=pt;

printf("computer: %c\n",p->date);

if(p->chi==NULL)

{

printf("Sorry, you lost.\n");

return 0;

}

//player

y: printf("player:\n");

scanf("%c",&temp);

t=getchar();

p=p->chi;

while(p->date!=temp)

{

p=p->bro;

if(!p)

{

printf("illegal move.\n");

p=pt;

goto y;

}

}

if(p->chi==NULL)

{

printf("Congratulate, you win.\n");

return 0;

}

p=p->chi;

}

}

int main()

{

int choice;

char yn,temp;

NODE\* home;

home=inputtree();

rank=0;

manage(home->chi);

print(home->chi);

while(1)

{

printf("Who play first(0: computer; 1: player )?\n");

scanf("%d",&choice);

yn=getchar();

start(home->chi,choice);

printf("Continue(y/n)?\n");

scanf("%c",&yn);

temp=getchar();

if(yn=='n')

break;

}

}

# 1.一元多项式相乘

成绩: 10 / 折扣: 0.8

### 题目说明

　　编程求两个一元多项式的乘积。

### 输入

　　输入数据为两行，分别表示两个一元多项式。每个一元多项式以指数递增的顺序输入多项式各项的系数（整数）、指数（整数）。

　　例如：1+2x+x2表示为：<1,0>,<2,1>,<1,2>,

### 输出

　　以指数递增的顺序输出乘积： <系数,指数>,<系数,指数>,<系数,指数>,

　　零多项式的输出格式为：<0,0>,

#include<stdio.h>

#include<stdlib.h>

int flag;

struct date

{

int a;

int b;

struct date\* next;

};

typedef struct date DATE;

typedef struct date\* PDATE;

PDATE setnew(PDATE p0,int a,int b)

{

PDATE p=NULL;

p=(PDATE) malloc (sizeof(DATE));

p0->next=p;

p->a=a;

p->b=b;

p->next=NULL;

return p;

}

PDATE find(PDATE p,int b)//向右查找同次，不存在则创建

{

if(!b)

return p;

if(p->next==NULL)

return setnew(p,0,b);

PDATE pt=p,temp=NULL;

p=p->next;

while(p->b<b&&(p->next!=NULL))

{

p=p->next;

}

if(p->b==b)

return p;

if(p->b>b)

{

temp=p->next;

pt=setnew(p,p->a,p->b);

pt->next=temp;

p->a=0;

p->b=b;

return p;

}

if(p->next==NULL)

return setnew(p,0,b);

}

void out(PDATE p)

{

if(p->a!=0)

{

printf("<%d,%d>,",p->a,p->b);

flag=0;

}

}

int main()

{

flag=1;

char temp;

int a=0,b=0;

//initial heads

PDATE p1=(PDATE) malloc (sizeof(DATE));

p1->a=0;

p1->b=0;

p1->next=NULL;

PDATE head1=p1;

PDATE p2=(PDATE) malloc (sizeof(DATE));

p2->a=0;

p2->b=0;

p2->next=NULL;

PDATE head2=p2;

PDATE p3=(PDATE) malloc (sizeof(DATE));

p3->a=0;

p3->b=0;

p3->next=NULL;

PDATE head3=p3;

//read

temp=getchar();

while(temp!='\n')

{

scanf("%d,%d>,",&a,&b);

p1=setnew(p1,a,b);

temp=getchar();

}

temp=getchar();

while(temp!='\n')

{

scanf("%d,%d>,",&a,&b);

p2=setnew(p2,a,b);

temp=getchar();

}

for(p1=head1->next;p1!=NULL;p1=p1->next)

{

p3=head3;

for(p2=head2->next;p2!=NULL;p2=p2->next)

{

p3=find(p3,p1->b+p2->b);

p3->a+=p1->a\*p2->a;

}

}

p3=head3;

while(p3!=NULL)

{

out(p3);

p3=p3->next;

}

if(flag)

printf("<0,0>,\n");

else

printf("\n");

}

# 3. 双向约瑟夫问题-【选做】

成绩: 10 / 折扣: 0.8

　　约瑟夫问题是一个经典的问题，我们不妨将这个经典问题进行扩展，变成一个双向的约瑟夫问题。

　　已知n个人（不妨分别以编号1，2，3，…，n 代表 ）围坐在一张圆桌周围，首先从编号为 k 的人从1开始顺时针报数，1, 2, 3, ...，记下顺时针数到 m 的那个人，同时从编号为 k 的人开始逆时针报数，1, 2, 3, ...，数到 m 后，两个人同时出列。然后从出列的下一个人又从 1 开始继续进行双向报数，数到m的那两个人同时出列，…；。依此重复下去，直到圆桌周围的人全部出列。直到圆桌周围只剩一个人为止。

 　　如果双向报数报到 m 时落在同一个人身上，那本次出列的只有一个人。

　　例如：5,1,2。则总共5个人，从1开始顺时针报数，数到2，定位编号2；同时从1开始报数数到2，定位编号5；2和5同时出列。然后继续开始报数，顺时针报数3,4，定位到4；逆时针报数4,3，定位3；4和3同时出列。最后剩余的为编号1。输出为：2-5,4-3,1,。

　　如果输入：6,2,3。则输出：4-6,2,1-3,5,。其中第2次只输出一个2，表示第二次双向报数时，恰好都落在编号2上，所以只有一个编号出列。

### 输入

n,k,m

### 输出

按照出列的顺序依次输出编号。同时出列编号中间用减号“-”连接。

### 非法输入的对应输出如下

a)

输入：n、k、m任一个为0  
输出：n,m,k must bigger than 0.

b)

输入：k>n  
输出：k should not bigger than n.

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

struct date

{

int a;

struct date\* l;

struct date\* r;

};

typedef struct date DATE;

typedef struct date\* PDATE;

PDATE setnew(PDATE p,int a)

{

PDATE pt;

pt=(PDATE) malloc (sizeof(DATE));

pt->a=a;

pt->r=p->r;

p->r=pt;

pt->l=p;

pt->r->l=pt;

return pt;

}

PDATE count(PDATE p,int n,int op)

{

if(n==0)

return p;

PDATE pt=p;

n--;

int tempn=n;

switch (op)

{

case 1:

{

while(n--)

pt=pt->r;

break;

}

case 2:

{

while(n--)

pt=pt->l;

break;

}

}

return pt;

}

PDATE del(PDATE p,int op)

{

PDATE pt;

if(op==1)

pt=p->r;

else

pt=p->l;

p->l->r=p->r;

p->r->l=p->l;

return pt;

}

int main()

{

int n=0,k=0,m=0;

scanf("%d,%d,%d",&n,&k,&m);

if(!(n\*k\*m))

printf("n,m,k must bigger than 0.\n");

else if(k>n)

printf("k should not bigger than n.\n");

else

{

PDATE p=NULL,p1=NULL,p2=NULL;

PDATE head=(DATE \*)malloc(sizeof(DATE));

head->l=head;

head->r=head;

head->a=1;

p=head;

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

p=setnew(p,i);

for(p=head;p->a!=k;p=p->l)

{;}

p1=p;

p2=p;

while(n!=0&&n!=1)

{

//printf("real=%d\n",real);

p1=count(p1,m,1);

p2=count(p2,m,2);

if(p1->a==p2->a)

{

n--;

printf("%d,",p1->a);

}

else

{

n-=2;

printf("%d-%d,",p1->a,p2->a);

}

if(p1->r==p2)

{

p1=del(p1,1);

p2=del(p2,2);

p1=p1->r;

p2=p2->l;

}

else

{

p1=del(p1,1);

p2=del(p2,2);

}

}

if(n)

printf("%d,",p1->a);

printf("\n");

}//end of else

}

# 4. 挤牛奶【选做】

成绩: 10 / 折扣: 0.8

三个农民每天清晨5点起床，然后去牛棚给3头牛挤奶。第一个农民在300时刻(从5点开始计时，秒为单位)给他的牛挤奶，一直到1000时刻。第二个农民在700时刻开始，在 1200时刻结束。第三个农民在1500时刻开始2100时刻结束。期间最长的至少有一个农民在挤奶的连续时间为900秒(从300时刻到1200时刻)，而最长的无人挤奶的连续时间(从挤奶开始一直到挤奶结束)为300秒(从1200时刻到1500时刻)。

你的任务是编一个程序，读入一个有N个农民(1 <= N <= 5000)挤N头牛的工作时间列表，计算以下两点(均以秒为单位):

* 最长至少有一人在挤奶的时间段。
* 最长的无人挤奶的时间段。

## 输入

|  |  |
| --- | --- |
| 第1行: | 一个整数N |
| 第22..N+1行: | 每行两个小于1000000的非负整数，表示一个农民的开始时刻与结束时刻。 |

## 输出

对应输入，输出上述两个需要求的值。

#include<stdio.h>

#include<stdlib.h>

struct date

{

long a;

long b;

struct date\* l;

struct date\* r;

};

typedef struct date DATE;

typedef struct date\* PDATE;

PDATE setnew(PDATE l,PDATE r,long a,long b)

{

PDATE p=NULL;

p=(PDATE)malloc(sizeof(DATE));

p->l=l;

p->r=r;

p->a=a;

p->b=b;

if(l!=NULL)

l->r=p;

if(r!=NULL)

r->l=p;

return p;

}

int del(PDATE p)//del p and p...

{

PDATE temp;

while(p!=NULL)

{

temp=p;

p=p->r;

free(temp);

}

return 0;

}

int make(long a,long b,PDATE head)

{

PDATE pa=head;

PDATE pb=head;

while(pa->r!=NULL)

{

if(pa->a>a)

{

pa=pa->l;

break;

}

else

pa=pa->r;

}//end while pa

if(pa->a>a)

pa=pa->l;

if(b<=pa->b)//ab在pa内

return 0;

if(pa->r==NULL)//pa指向最后一个

{

if(a<=pa->b)//a在pa的ab间

{

pa->b=b;

return 0;

}

else//a>pa的b

{

setnew(pa,NULL,a,b);

return 0;

}

}

pb=pa;

while(pb->r!=NULL)

{

if(pb->b>b)

break;

else

pb=pb->r;

}//end while pa

if(pb->r==NULL&&pb->b<b)//b 超过最后一个

{

if(pa->b>a)//a in

{

pa->b=b;

//del(pa->r);

return 0;

}

else//a out

{

setnew(pa,NULL,a,b);

//del(pa->r->r);

return 0;

}

}

if(a<pa->b)

{

if(b>=pb->a)//ab in

{

pa->b=pb->b;

//PDATE TEMP=pa->r;

pa->r=pb->r;

if(pb->r!=NULL)

pb->r->l=pa;

//pb->r=NULL;

//del(TEMP);

return 0;

}

else//ain bout

{

pa->b=b;

//pb->l->r=NULL;

//PDATE TEMP=pa->r;

//del(TEMP);

pa->r=pb;

pb->l=pa;

return 0;

}

}

else

{

if(b>pb->a)//aout bin

{

pb->a=a;

//pb->l->r=NULL;

//del(pa->r);

pa->r=pb;

pb->l=pa;

return 0;

}

else//ab out

{

//pb->l->r=NULL;

//del(pa->r);

setnew(pa,pb,a,b);

return 0;

}

}

}

void print(PDATE head)

{

PDATE p=head->r->r;

long x1=0,x2=0;

if(x1<(p->b-p->a))

{

x1=(p->b-p->a);

}

p=p->r;

while(p!=NULL)

{

if(x1<(p->b-p->a))

{

x1=(p->b-p->a);

}

if(x2<(p->a-p->l->b))

{

x2=p->a-p->l->b;

}

p=p->r;

}

printf("%ld %ld\n",x1,x2);

}

int main()

{

int n;

PDATE head=setnew(NULL,NULL,0,0);

PDATE p=setnew(head,NULL,0,0);

scanf("%d",&n);

long a,b;

while(n--)

{

scanf("%ld%ld",&a,&b);

make(a,b,head);

}

print(head);

}

# 5.从中缀向后缀转换表达式

成绩: 10 / 折扣: 0.8

**问题描述**

　　中缀表达式就是我们通常所书写的数学表达式，后缀表达式也称为逆波兰表达式，在编译程序对我们书写的程序中的表达式进行语法检查时，往往就可以通过逆波兰表达式进行。我们所要设计并实现的程序就是将中缀表示的算术表达式转换成后缀表示，例如，将中缀表达式

(A 一 (B\*C 十 D)\*E) ／ (F 十 G )

转换为后缀表示为：

ABC\*D十E\*—FG十／

**注意：**为了简化编程实现，假定变量名均为单个字母，运算符只有＋，－，＊，／ 和＾（指数运算），可以处理圆括号（），并假定输入的算术表达式正确。

**要求：**使用栈数据结构实现 ，输入的中缀表达式以＃号结束

## ****输入****

整数N。表示下面有N个中缀表达式   
N个由单个字母和运算符构成的表达式

## ****输出****

N个后缀表达式。

#include<stdio.h>

#include<stdlib.h>

int main()

{

int n;

char c;

char date[100];

int p;

scanf("%d",&n);

while(n--)

{

p=0;

scanf("\n%c",&c);

while(c!='#')

{

if((c>='A'&&c<='Z')||(c>='a'&&c<='z'))

{

printf("%c",c);

scanf("%c",&c);

continue;

}

if(c=='(')

{

date[p++]=c;

scanf("%c",&c);

continue;

}

x: if(p==0)

{

date[p]=c;

p++;

}

else if(date[p-1]=='('&&c!=')')

{

date[p]=c;

p++;

}

else if(c=='+'||c=='-'||c=='\*'||c=='/'||c=='^')

{

if(date[p-1]=='^')

{

if(c=='^')

{

date[p]=c;

p++;

}

else

{

printf("%c",date[p-1]);

p--;

goto x;

}

}

else if(date[p-1]=='\*'||date[p-1]=='/')

{

if(c=='^')

{

date[p]=c;

p++;

}

else

{

printf("%c",date[p-1]);

p--;

goto x;

}

}

else

{

if(c=='+'||c=='-')

{

p--;

printf("%c",date[p]);

goto x;

}

else

{

date[p]=c;

p++;

}

}

}

else if(c==')')

{

p--;

while(date[p]!='(')

{

printf("%c",date[p]);

p--;

}

}

scanf("%c",&c);

}

while(--p!=-1)

{

printf("%c",date[p]);

}

printf("\n");

}

}

# 6. 孤独的运货员

成绩: 10 / 折扣: 0.8

## 背景

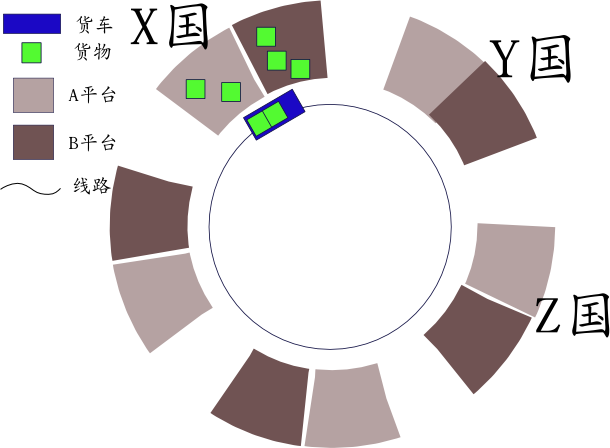
航空公司的货运飞机往返于各个国家和货物分理中心之间，分理中心的货物运送到各个国家，再将各个国家发出的货物送到分理中心。

分理中心非常庞大，每个国家都有一个自己的货运站，每个货运站有两个平台用于堆放货物。其中 A 平台用于堆放运送到该国的货物，而 B 平台堆放着其它国家的货物。各个国家的货运站排成一个环形。

平时，有货运汽车沿着环形的线路在各个国家的货运站之间运行。货运汽车的仓库是非常狭窄的，先放进去的货物只能等后放进去的货物移走之后才能搬出来。

当货运汽车到达一个站点 X 时，它首先卸货。卸货的过程是：首先查看最外面的箱子是否标记着 X 标签，如果是，代表这是运送到 X 国的货物，则放到 A 平台上，如果不是，则把该货箱放到B平台货物队列的末尾，然后处理下一箱货。当 B 平台放满或者是车被搬空，则开始装货。装货的过程是：从 B 平台货物队列的开始处装车，把车装满或把 B 平台搬空则装货结束。车子开往下一个货运站。

卸下一箱货物和装载一箱货物均需要一分钟的时间，且假设当货车从一个货运站开往另一个货运站的时间为两分钟。



每天，货车和货运飞机都这样周而复始的运行着，直到有一天因为报酬的原因，分理中心的工人开始罢工。货运飞机依然运行，但却无法运走任何货物。

## 任务

你是老板的心腹，平时深受老板关照。因此，你决定停止罢工，独自担当起分理的任务。现在，分理中心已经堆积了大量的货物，你决定在开始前先写程序计算一下要完成搬运需要多长的时间。

## 输入

对于输入部分包含几组数据，第一行是一个整数SET，表明下面有几组数据，随后便是这几组输入数据。

对于每一组输入数据，其第一行为三个整数 N、S 和 Q。其中 N(2<=N<=100) 表示环中站的个数。S(1<=S<=100) 表示货车的最大容量，即货车同时可以装载多少箱货物。而 Q(1<=Q<=100) 表示 B 平台所能放置的货物的总数，假定该系统中所有货运站 B 平台的最大容量都相同。

从第二行往后一共有 N 行，每行一的第一个数字代表该站 B 平台上货物的总数，之后的每个数字代表每箱货物运送到哪个站。

## 输出

对于每组输入，你需要在独立的一行中输出完成该任务所需要的时间。

#include<stdio.h>

#include<stdlib.h>

struct goods

{

int destination;

struct goods \* next;

struct goods \* last;

};

struct station

{

int No;

int quantity;

struct goods \* b;

struct goods \* end;

};

typedef struct goods GOODS;

typedef struct goods\* PGOODS;

typedef struct station STATION;

typedef struct station\* PSTATION;

long time,count;

int N,S,Q;

PSTATION STA[101];

PGOODS setnewG(int des,PGOODS l,PGOODS n)

{

PGOODS p=NULL;

p=(PGOODS) malloc (sizeof(GOODS));

p->destination=des;

p->last=NULL;

p->next=NULL;

if(l!=NULL)

{

p->last=l;

l->next=p;

}

if(n!=NULL)

{

p->next=n;

n->last=p;

}

return p;

}

PSTATION setnewSTA(int No,int quantity)

{

PSTATION p=NULL;

p=(PSTATION) malloc (sizeof(STATION));

p->quantity=quantity;

p->b=setnewG(0,NULL,NULL);

p->No=No;

p->end=p->b;

return p;

}

PGOODS move(PGOODS p1,int a,PGOODS p2,int b)//p2 connect to p1

{

p2->last->next=p2->next;

if(p2->next!=NULL)

p2->next->last=p2->last;

p2->next=p1->next;

if(p1->next!=NULL)

p1->next->last=p2;

p2->last=p1;

p1->next=p2;

time+=1;

STA[a]->quantity++;

STA[b]->quantity--;

return p2;

}

PGOODS del(PGOODS p)

{

p->last->next=p->next;

if(p->next!=NULL)

p->next->last=p->last;

PGOODS pn=p->next;

free(p);

time+=1;

count--;

STA[0]->quantity--;

return pn;

}

int unload(int i)

{

PGOODS p1=STA[i]->end;

PGOODS p0=STA[0]->b->next;

if(p0!=NULL)

while(p0->destination==i)

{

p0=del(p0);

if(!p0)

break;

}

while(p0!=NULL&&STA[i]->quantity<Q)

{

if(p0->destination!=i)

p1=move(p1,i,p0,0);

else

del(p0);

p0=STA[0]->b->next;

}

if(p0!=NULL)

while(p0->destination==i)

{

p0=del(p0);

if(!p0)

break;

}

STA[i]->end=p1;

return 0;

}

int load(int i)

{

PGOODS p1=STA[i]->b->next;

PGOODS p0=STA[0]->b;

while(p1!=NULL&&STA[0]->quantity<S)

{

move(p0,0,p1,i);

p1=STA[i]->b->next;

}

if(STA[i]->quantity==0)

STA[i]->end=STA[i]->b;

return 0;

}

int main()

{

int SET,tempint,i;

scanf("%d",&SET);

while(SET--)

{

time=0;

count=0;

//START IMPUT

scanf("%d%d%d",&N,&S,&Q);//S:track maxvolume Q:station maxvolume

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

{

scanf("%d",&tempint);

STA[i+1]=setnewSTA(i+1,tempint);

count+=tempint;

PGOODS p=NULL;

p=STA[i+1]->b;

for(int j=0;j<STA[i+1]->quantity;j++)

{

scanf("%d",&tempint);

p=setnewG(tempint,p,NULL);

}

STA[i+1]->end=p;

}//finish input

STA[0]=setnewSTA(0,0);//initial track

//start management

i=0;

while(1)

{

if(i!=N)

i++;

else

i=1;

unload(i);

load(i);

if(count==0)

{

printf("%ld\n",time);

break;

}

time+=2;

}

}//end while

return 0;

}

# 8. 迷宫问题【选做】

成绩: 10 / 折扣: 0.8

**题目背景：**

迷宫有一个入口，一个出口。一个人从入口走进迷宫，目标是找到出口。阴影部分和迷宫的外框为墙，每一步走一格，每格有四个可走的方向，探索顺序为：南、东、北、西。

## 迷宫示意图

## 输入：输入迷宫数组

## 输出：若有解，输出从入口到出口的一条路径，否则输出 there is no solution!

## 例（上图所示的迷宫数组）

输入

4 4

0 0 1 0

0 1 0 1

0 0 0 0

0 1 0 0

输出

<1,1> <2,1> <3,1> <3,2> <3,3> <4,3> <4,4>

#include<stdio.h>

int main()

{

int a,b;

scanf("%d%d",&a,&b);

int date[102][102];

for(int i=0;i<102;i++)

for (int j=0;j<102;j++)

date[i][j]=1;

int stack[500][4]={0};

int p=1;

stack[0][2]=5;

stack[1][0]=1;

stack[1][1]=1;

stack[1][3]=5;

for(int x=1;x<=a;x++)//input start

{

for(int y=1;y<=b;y++)

{

int temp;

scanf("%d",&temp);

date[x][y]=temp;

}

}//input finish

int p1,p2;

while(!(stack[p][0]==a&&stack[p][1]==b))//find start

{

switch (stack[p][2])

{

case 0://down

{

if(stack[p][2]==stack[p][3])

{

stack[p][2]++;

break;

}

p1=stack[p][0]+1;

p2=stack[p][1];

if(date[p1][p2])//wall

{

stack[p][2]++;

goto x1;

}

else//road

{

stack[p][2]++;

p++;

stack[p][0]=p1;

stack[p][1]=p2;

stack[p][3]=2;

break;

}

}

case 1://right

{

x1: if(stack[p][2]==stack[p][3])

{

stack[p][2]++;

break;

}

p1=stack[p][0];

p2=stack[p][1]+1;

if(date[p1][p2])//wall

{

stack[p][2]++;

goto x2;

}

else//road

{

stack[p][2]++;

p++;

stack[p][0]=p1;

stack[p][1]=p2;

stack[p][3]=3;

break;

}

}

case 2://up

{

x2: if(stack[p][2]==stack[p][3])

{

stack[p][2]++;

break;

}

p1=stack[p][0]-1;

p2=stack[p][1];

if(date[p1][p2])//wall

{

stack[p][2]++;

goto x3;

}

else//road

{

stack[p][2]++;

p++;

stack[p][0]=p1;

stack[p][1]=p2;

stack[p][3]=0;

break;

}

}

case 3://left

{

x3: if(stack[p][2]==stack[p][3])

{

stack[p][2]++;

break;

}

p1=stack[p][0];

p2=stack[p][1]-1;

if(date[p1][p2])//wall

{

stack[p][2]++;

goto x4;

}

else//road

{

stack[p][2]++;

p++;

stack[p][0]=p1;

stack[p][1]=p2;

stack[p][3]=1;

break;

}

}

case 4://back

{

x4: stack[p][2]=0;

p--;

break;

}

case 5:

{

printf("there is no solution!\n");

return 0;

}

}

}//find finish

p=1;

while(stack[p][2])

{

printf("<%d,%d> ",stack[p][0],stack[p][1]);

p++;

}

printf("<%d,%d> \n",stack[p][0],stack[p][1]);

return 0;

}

# 15.Binary Tree Traversals【选做】

成绩: 10 / 折扣: 0.8

## Problem Description

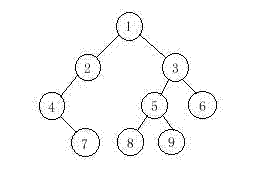
A binary tree is a finite set of vertices that is either empty or consists of a root r and two disjoint binary trees called the left and right subtrees. There are three most important ways in which the vertices of a binary tree can be systematically traversed or ordered. They are preorder, inorder and postorder. Let T be a binary tree with root r and subtrees T1,T2.

In a preorder traversal of the vertices of T, we visit the root r followed by visiting the vertices of T1 in preorder, then the vertices of T2 in preorder.

In an inorder traversal of the vertices of T, we visit the vertices of T1 in inorder, then the root r, followed by the vertices of T2 in inorder.

In a postorder traversal of the vertices of T, we visit the vertices of T1 in postorder, then the vertices of T2 in postorder and finally we visit r.

Now you are given the preorder sequence and inorder sequence of a certain binary tree. Try to find out its postorder sequence.



## Input

The input contains several test cases. The first line of each test case contains a single integer n (1

## Output

For each test case print a single line specifying the corresponding postorder sequence.

### Sample Input

9  
1 2 4 7 3 5 8 9 6  
4 7 2 1 8 5 9 3 6

### Sample Output

7 4 2 8 9 5 6 3 1

#include<stdio.h>

#include<stdlib.h>

struct bt

{

int n;

struct bt\* l;

struct bt\* r;

};

typedef struct bt BT;

typedef struct bt\* PBT;

int c;

PBT setnew(int n)

{

PBT p;

p=(PBT) malloc (sizeof(BT));

p->n=n;

p->l=NULL;

p->r=NULL;

return p;

}

PBT setline(PBT l,PBT r,int n)

{

PBT p;

p=setnew(n);

if(l)

{

l->r=p;

p->l=l;

}

if(r)

{

r->l=p;

p->r=r;

}

return p;

}

PBT del(PBT p)

{

if(p->l)

p->l->r=NULL;

if(p->r)

p->r->l=NULL;

free(p);

return NULL;

}

PBT find(int n,PBT start,PBT end)

{

PBT p=start;

while(p->n!=n)

p=p->r;

return p;

}

int before(PBT p1,int n)//p1 before p2 return 1,else return 0

{

PBT p=p1;

while(p)

{

if(p->n==n)

return 1;

p=p->l;

}

return 0;

}

PBT make(PBT start1,PBT start2,PBT p0)

{

int flag=0;

if(start1->r==NULL)

return NULL;

PBT s11=start1->r,s12=start1,s21=start2,s22=start2,p=start2;

while(start1->n!=p->n)

{

p=p->r;

s12=s12->r;

}

s12=s12->r;

if(before(p,s11->n))//p前有s11->n)

{

p0->l=setnew(s11->n);

flag=1;

}

if(s12)

p0->r=setnew(s12->n);

s22=p->r;

if(s22&&s12)

{

if(s11->l)

{

s11->l->r=NULL;

s11->l=NULL;

}

if(s12->l)

{

s12->l->r=NULL;

s12->l=NULL;

}

if(s21)

s21->l=NULL;

if(s22->l)

{

s22->l->r=NULL;

s22->l=NULL;

}

del(start1);

del(p);

if(flag)

make(s11,s21,p0->l);

make(s12,s22,p0->r);

}

else

{

if(s11->l)

{

s11->l->r=NULL;

s11->l=NULL;

}

if(s21)

s21->l=NULL;

free(start1);

del(p);

make(s11,s21,p0->l);

}

return NULL;

}

int printpost(BT\* head)

{

if(head==NULL)

return 0;

printpost(head->l);

printpost(head->r);

printf("%d",head->n);

free(head);

if(--c)

printf(" ");

else

printf("\n");

return 0;

}

int main()

{

int n,temp,ntemp;

x: if(scanf("%d",&n)==0)

return 0;

c=n;

if(n==ntemp)

return 0;

ntemp=n;

//setline

PBT head1=setnew(-1),head2=setnew(-2);

PBT p=head1;

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

{

scanf("%d",&temp);

p=setline(p,NULL,temp);

}

p=head2;

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

{

scanf("%d",&temp);

p=setline(p,NULL,temp);

}

//set 1

PBT head;

head1=head1->r;

head2=head2->r;

del(head1->l);

del(head2->l);

head=setnew(head1->n);

make(head1,head2,head);

printpost(head);

goto x;

return 0;

}