

- (3) 多项式 $A(x)$ 和 $B(x)$ 相加得到多项式 $C(x)$ ，输出 $C(x)$ ；
 (4) 多项式 $A(x)$ 和 $B(x)$ 相减得到多项式 $D(x)$ ，输出 $D(x)$ 。

【测试数据】

- (1) $A(x)=2x+5x^8-3.1x^{11}$ $B(x)=7-5x^8+11x^9$ $C(x)=-3.1x^{11}+11x^9+2x+7$
 (2) $A(x)=1+x+x^2+x^3+x^4+x^5$ $B(x)=-x^3-x^4$ $C(x)=1+x+x^2+x^5$
 (3) $A(x)=x+x^3$ $B(x)=-x-x^3$ $C(x)=0$
 (4) $A(x)=x+x^{100}$ $B(x)=x^{100}+x^{200}$ $C(x)=x+2x^{100}+x^{200}$
 (5) $A(x)=x+x^2+x^3$ $B(x)=0$ $C(x)=x+x^2+x^3$
 (6) $A(x)=6x^{-3}-x+4.4x^2-1.2x^9$ $B(x)=-6x^{-3}+5.4x^2-x^2+7.8x^{15}$
 $D(x)=-7.8x^{15}-1.2x^9+12x^{-3}-x$

【实现提示】

用带头结点的单链表存储多项式，多项式的项数可放入头结点中。

实验二 栈和队列的实现与应用

题目一 数制转换

【问题描述】

十进制数 N 和其它 d 进制数的转换是计算机实现计算的基本问题，其解决方法很多，其中一个简单的算法是基于下列原理：

$N = (N \text{ div } d) * d + N \text{ mod } d$ 。其中： div 为整除运算， mod 为求余运算。

例如， $43_{10} = 101011_2$

被除数	除数	商	余数
43	2	21	1
21	2	10	1
10	2	5	0
5	2	2	1
2	2	1	0
1	2	0	1

从上述运算过程可见：①从低位到高位顺序产生二进制数的各个位数；②与打印输出由高位到低位的顺序相反。

因此，将计算过程中得到的二八进制数的各位顺序进栈，再按出栈序列打印输出，即可得到与输入对应的二进制数。

【基本要求】

写一个程序，利用栈将一个十进制数转换为二进制数的形式输出，其中，栈的基本运算应自行实

现，不能使用开发工具中提供的栈类型。

题目二 括号匹配问题

【问题描述】

设一个表达式中可以包含三种括号：“(”和“)”，“[”和“]”，“{”和“}”，并且这三种括号可以按照任意的次序嵌套使用，考查表达式中的括号是否匹配。

【基本要求】

写一个程序，判断给定表达式中的括号是否匹配。

【测试数据】

有多个表达式，每个表达式（不超过 100 个字符）占一行。例如，

[(d+f)*{}2]

[(2+3))

()}

[4(6)7)9

【实现提示】

自左至右扫描表达式，若遇左括号，则将左括号入栈，若遇右括号，则将其与栈顶的左括号进行匹配，若配对，则栈顶的左括号出栈，否则出现括号不匹配错误。

程序运行时，对输入每个表达式，判断其括号匹配情况。若其中的括号是匹配的，则输出“yes”，否则输出“no”。

【算法】

```
Status CheckBrackets(char *expr)
//表达式作为字符串保存在 expr 中，括号完全匹配时返回 OK，否则返回 ERROR
InitStack(S);           // 构造一个空栈，用于暂存左括号;
for(i = 0; expr[i]!='\0'; i++) {    // 逐个考查表达式中的字符，忽略非括号字符
    if (expr[i]=='(' || expr[i]=='[' || expr[i]=='{')
        push(S, expr[i]);          // 遇到左括号时，令左括号入栈
    else
        if (expr[i]==')' || expr[i]==']' || expr[i]=='}') { // 遇到右括号时，判断匹配情况
            if (StackEmpty(S))    // 遇到右括号时栈为空，即不存在与之匹配的左括号
                return ERROR;
            else {                // 栈中有左括号
                Gettop(S,c);      // 读取栈顶的左括号（不出栈）
                switch (expr[i]) {
                    case ')': if (c!='(') // 不匹配：当前括号是“）”，栈顶不是“（”
                        return ERROR;
                        break;        // 当前的“）”与栈顶的“（”相匹配
                    case ']': if (c!='[') // 不匹配：当前括号是“】”，栈顶不是“[”
                        return ERROR;
                        break;        // 当前的“】”与栈顶的“[”相匹配
                    case '}': if (c!='{') // 不匹配：当前括号是“}”，栈顶不是“{”
                        return ERROR;
                        break;
                }
            }
        }
    }
```

```

        return ERROR;
    break;    // 当前的“}”与栈顶的“{”相匹配
} // switch
pop(S);      // 栈顶的左括号出栈
} // if-else
} // if
} // for
if (StackEmpty(S)) return OK; // 已到表达式结尾且栈空，说明所有括号都匹配
return ERROR; // 表达式中的左括号多（留在栈中），右括号少，括号不完全匹配
}

```

题目三 停车场管理

【问题描述】

设停车场是一个可停放 n 辆汽车的狭长通道，且只有一个大门可供汽车进出。汽车在停车场内按车辆到达时间的先后顺序，依次由北向南排列（大门在最南端，最先到达的第一辆车停放在车场的最北端），若车场内已停满 n 辆汽车，则后来的汽车只能在门外的便道上等候，一旦有车开走，则排在便道上的第一辆车即可开入。当停车场内某辆车要离开时，在它之后进入的车辆必须先退出车场为它让路，待该辆车开出大门外，其他车辆再按原次序进入车场，每辆停放在车场的车在它离开停车场时必须按它停留的时间长短交纳费用。试为停车场编制按上述要求进行管理的模拟程序。

【基本要求】

以栈模拟停车场，以队列模拟车场外的便道，按照从终端读入的输入数据序列进行模拟管理。每一组输入数据包括三个数据项：汽车“到达”或“离去”信息、汽车牌照号码以及到达或离去的时刻。对每一组输入数据进行操作后的输出信息为：若是车辆到达，则输出汽车在停车场内或便道上的停车位置，若是车辆离去，则输出汽车在停车场内停留的时间和应交纳的费用（在便道上停留的时间不收费）。栈以顺序结构实现，队列以链表结构实现。

【测试数据】

设 $n=2$ ，输入数据为：('A',1,5), ('A',2,10), ('D',1,15), ('A',3,20), ('A',4,25), ('A',5,30), ('D',2,35), ('D',4,40), ('E',0,0)。其中：'A'表示到达（Arrival）；'D'表示离去（Departure）；'E'表示输入结束（End）。

【实现提示】

需另设一个栈，临时停放为给要离去的汽车让路而从停车场退出来的汽车，也用顺序存储结构实现。输入数据按到达或离去的时刻有序。栈中的每个元素表示一辆汽车，包含两个数据项：汽车的牌照号码和进入停车场的时刻。

题目四 迷宫问题

【问题描述】

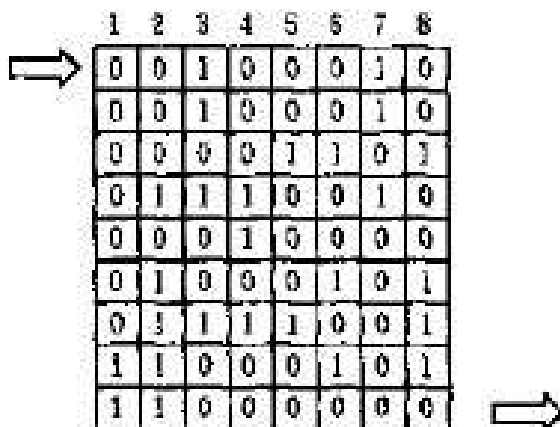
以一个 $m \times n$ 的长方阵表示迷宫，0 和 1 分别表示迷宫中的通路和障碍。设计一个程序，对任意设定的迷宫，求出一条从入口到出口的通路，或得出没有通路的结论。

【基本要求】

首先实现一个以链表作存储结构的栈类型，然后编写一个求解迷宫的非递归程序。求得的通路以三元组 (i, j, d) 的形式输出，其中 (i, j) 指示迷宫中的一个位置（行号和列号）， d 表示走到下一位置的方向（对于迷宫中任一位置，均有下、右、上、左四个方向来走出下一个位置，这四个方向可分别编号为 1, 2, 3, 4）。例如，对于下面测试数据给出的迷宫，输出的一条通路为： $(1,1,1)$, $(2,1,1)$, $(3,1,1)$, $(4,1,1)$, $(5,1,2)$, $(5,2,2)$, $(5,3,1)$, ...。

【测试数据】

迷宫如下图所示：左上角 $(1,1)$ 为入口，右下角 $(8,9)$ 为出口。



	1	2	3	4	5	6	7	8
1	0	0	1	0	0	0	1	0
2	0	0	1	0	0	0	1	0
3	0	0	0	0	1	1	0	1
4	0	1	1	1	0	0	1	0
5	0	0	0	1	0	0	0	0
6	0	1	0	0	0	1	0	1
7	0	1	1	1	1	0	0	1
8	1	1	0	0	0	1	0	1
9	1	1	0	0	0	0	0	0

【实现提示】

计算机解迷宫通常用的是“穷举求解”方法，即从入口出发，顺着某一个方向进行探索，若能走通，则继续往前进；否则沿着原路退回，换一个方向继续探索，直至出口位置，求得一条通路。假如所有可能的通路都探索到而未能到达出口，则所设定的迷宫没有通路。

可以用二维数组存储迷宫数据，用数组元素的下标 $[rowNo][colNo]$ 表示坐标。

通常设定入口点的下标为 $(1,1)$ ，出口点的下标为 (n,n) 。为处理方便起见，可在迷宫的四周加一圈障碍（由行号为 0 和 $n+1$ 、列号为 0 和 $n+1$ 的所有数组元素构成）。

例如，上图所示的迷宫可如下表示：

```
#define N 8
char maze[N+2][N+2]; //此处 char 作为小整数类型使用
int rowNo,colNo;
for(rowNo = 0; rowNo < N+2; rowNo++) //迷宫位置全部初始化为有障碍
    for(colNo = 0; colNo < N+2; colNo++)
        maze[rowNo][colNo] = 1;
```

然后通过键盘输入或读取提前保存在文件中的迷宫数据设置迷宫的状态，以及出口和入口位置。