#include<string.h>

#include<ctype.h>

#include<malloc.h> /\* malloc()等 \*/

#include<limits.h> /\* INT\_MAX等 \*/

#include<stdio.h> /\* EOF(=^Z或F6),NULL \*/

#include<stdlib.h> /\* atoi() \*/

#include<io.h> /\* eof() \*/

#include<math.h> /\* floor(),ceil(),abs() \*/

#include<process.h> /\* exit() \*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

typedef int Status;

#define LIST\_INIT\_SIZE 100

#define LISTINCREMRNT 10

typedef char ElemType;

FILE \*fp;

typedef struct

{

ElemType \*base; //储存空间基地址

ElemType \*top; // 记录当前链表长度

int stacksize; //链表规模

} stack;

Status InitStack(stack \*S)

{

(\*S).base = (ElemType\*)malloc(LIST\_INIT\_SIZE\*sizeof(ElemType));

if(!(\*S).base)

{

exit(OVERFLOW);

}

(\*S).top = (\*S).base;

(\*S).stacksize = LIST\_INIT\_SIZE;

return OK;

}

Status DestroyStack(stack \*S)

{

free((\*S).base);

(\*S).base=NULL;

(\*S).top=NULL;

(\*S).stacksize=0;

return OK;

}

Status ClearStack(stack \*S)

{ /\* 把S置为空栈 \*/

(\*S).top=(\*S).base;

return OK;

}

Status StackEmpty(stack S)

{ /\* 若栈S为空栈，则返回TRUE，否则返回FALSE \*/

if(S.top==S.base)

return TRUE;

else

return FALSE;

}

int StackLength(stack S)

{

return S.top-S.base;

}

Status GetTop(stack S,ElemType \*e)

{ /\* 若栈不空，则用e返回S的栈顶元素，并返回OK；否则返回ERROR \*/

if(S.top>S.base)

{

\*e=\*(S.top-1);

return OK;

}

else

return ERROR;

}

Status Push(stack \*S, ElemType e)

{

if((\*S).top - (\*S).base >= (\*S).stacksize)

{

(\*S).base = (ElemType\*)realloc((\*S).base, ((\*S).stacksize + LISTINCREMRNT) \* sizeof(ElemType));

if(!(\*S).base)

exit(OVERFLOW);

(\*S).top = (\*S).base + (\*S).stacksize;

(\*S).stacksize += LISTINCREMRNT;

}

\*(S->top) = e;

(S->top)++;

return OK;

}

Status Pop(stack \*S,ElemType \*e)

{

if((\*S).top==(\*S).base)

return ERROR;

(\*S).top--;

\*e = \*((\*S).top);

return OK;

}

/\*Status ListTraverse(stack S)

{

ElemType \*e = S.base;

while (e < S.top)

{

printf("%d \n", \*e++);

}

return OK;

}\*/

Status StackTraverse(stack S,Status(\*visit)(ElemType))

{ /\* 从栈底到栈顶依次对栈中每个元素调用函数visit()。 \*/

/\* 一旦visit()失败，则操作失败 \*/

while(S.top>S.base)

visit(\*S.base++);

printf("\n");

return OK;

}

void check()

{ /\* 对于输入的任意一个字符串，检验括号是否配对 \*/

stack s;

ElemType ch[80],\*p,e;

if(InitStack(&s)) /\* 初始化栈成功 \*/

{

printf("请输入表达式\n");

gets(ch);

p = ch;

while (\*p)

{

switch(\*p)

{

case '(':

case '[':

Push(&s,\*p++);

break;

case ')':

case ']':

if(!StackEmpty(s))

{

Pop(&s, &e);

if(\*p==')'&&e!='('||\*p==']'&&e!='[')

{

printf("左右括号不配对\n");

exit(ERROR);

}

else

{

p++;

break; /\* 跳出switch语句 \*/

}

}

else

{

printf("缺乏左括号\n");

exit(ERROR);

}

default: p++; /\* 其它字符不处理，指针向后移 \*/

}

}

if(StackEmpty(s)) /\* 字符串结束时栈空 \*/

printf("括号匹配\n");

else

printf("缺乏右括号\n");

}

}

Status copy(ElemType c)

{ /\* 将字符c送至fp所指的文件中 \*/

fputc(c,fp);

return OK;

}

void LineEdit()

{

stack s;

char ch,c;

InitStack(&s);

printf("请输入,^Z结束输入:\n");

ch=getchar();

while(ch!=EOF)

{

while(ch!=EOF&&ch!='\n')

{

switch(ch)

{

case '#':

Pop(&s,&c);

break;

case '@':

ClearStack(&s);

break;

default:

Push(&s, ch);

}

ch = getchar();

}

StackTraverse(s,copy);

ClearStack(&s);

fputc('\n',fp);

if(ch!=EOF)

ch=getchar();

}

DestroyStack(&s);

}

void main()

{

check();

fp=fopen("mystack.txt","w"); /\* 在当前目录下建立ED.DAT文件，用于写数据， \*/

if(fp) /\* 如已有同名文件则先删除原文件 \*/

{

LineEdit();

fclose(fp); /\* 关闭fp所指的文件 \*/

}

else

printf("建立文件失败!\n");

}