第3章

栈——实验三

3.1 实验目的及要求

1．理解特殊的线性结构——顺序栈的抽象数据类型的定义，及其在 C 语言环境中的表示方法。

2．理解顺序栈的基本操作的算法，及其在 C 语言环境中一些主要基本操作的实现。

3．在 C 语言环境下实现顺序栈的应用操作：

① 利用栈实现十进制数转换成八进制数。

② 利用栈实现一位数的加减乘除的表达式求解。

3.2 实 验 内 容

经过对实验目的及要求的分析，本实验仍然采用首先描述栈的基本操作集函数，然后分别在两个应用操作中使用基本操作集函数来实现。

由于栈是一种特殊的线性结构，仅在栈顶进行插入和删除操作，即栈具有后进先出的特点，故其操作比一般的线性表更为容易，所以在本实验中有关栈的基本操作集的实现都比较简单，没有做过多的说明，而是在数制转换和表达式求解的应用操作中加入了更多的编程技巧，使读者通过本实验不仅了解栈这种特殊结构的线性表，而且掌握利用栈可实现很多的应用，尤其是在实现表达式求解时用到了两个顺序栈，并且加入了运算符的优先关系的判断等，实现稍有难度。

在程序 Stack.c 中，只包含了数制转换和一位数的表达式求解，多位数的表达式求解思想与一位数表达式求解思想一致，但需要添加多位数的接收处理，请读者自行编写代码。

程序名为：Stack.c。

在 Stack.c 中包含的函数如图 3.1 所示。

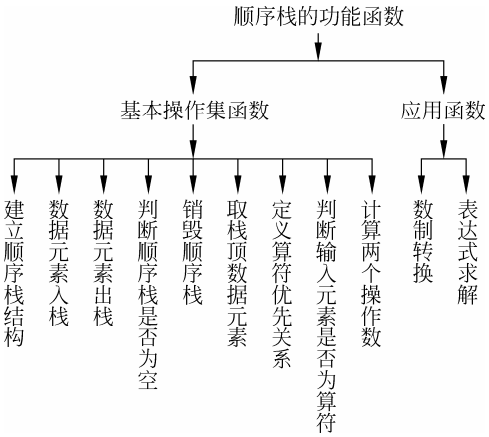

图 3.1　Stack.c 中包含的函数一览表

3.3　功能函数的分析设计及源代码

本部分列出了实现顺序栈的操作的源代码，并在适当的位置上添加了一些文字和流程图的注释，帮助读者理解顺序存储的栈的存储结构及操作算法。

文件名：Stack.c

```
#include "alloc.h"
#include "stdio.h"

#define STACK_INIT_SIZE 10
#define STACKINCREMENT 2

#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define OVERFLOW -2

typedef int SElemType;
typedef int Status;

//定义顺序栈的结构
typedef struct SqStack
{
  SElemType *base;
  SElemType *top;
  int stacksize;
}SqStack;

//初始化一个空栈
Status InitStack(SqStack *S)
{
  (*S).base=(SElemType *)malloc(STACK_INIT_SIZE*sizeof(SElemType));
  if(!(*S).base)
    exit(OVERFLOW);
```

```
    (*S).top=(*S).base;
    (*S).stacksize=STACK_INIT_SIZE;
    return OK;
  }
  //数据元素入栈
  Status Push(SqStack *S,SElemType e)
  {
    if((*S).top-(*S).base>=(*S).stacksize)
    {
     (*S).base=(SElemType*)realloc((*S).base,  ((*S).stacksize+STACKINCREMENT)
*sizeof(SElemType));
     if(!(*S).base)
       exit(OVERFLOW);
     (*S).top=(*S).base+(*S).stacksize;
     (*S).stacksize+=STACKINCREMENT;
    }
    *((*S).top)++=e;
    return OK;
  }
  //数据元素出栈
  Status Pop(SqStack *S,SElemType *e)
  {
   if((*S).top==(*S).base)
     return ERROR;
   *e=*--(*S).top;
   return OK;
  }
  //判断一个栈是否为空
  Status StackEmpty(SqStack S)
  {
    if(S.top==S.base)
      return TRUE;
    else
      return FALSE;
  }
  //销毁一个栈
  Status DestroyStack(SqStack *S)
   {
     free((*S).base);
     (*S).base=NULL;
     (*S).top=NULL;
     (*S).stacksize=0;
     return OK;
   }
  //十进制数转换成八进制
```

本函数实现了无符号十进制数和八进制数间的转换功能。

如输入的是负数，由于系统使用补码表示负数，会自动将其进行补码转换，再通过本

函数对其实现八进制转换。如，当输入－1 时，结果显示 65535。

如果需要将十进制转换成十六进制应该如何修改本函数？方法有两种，第一种，在入栈时，存入十六进制数；第二种，在出栈时，输出十六进制数。请读者自己编写代码。

```
void conversion()
{
  SqStack s;
  unsigned n;
  SElemType e;
  InitStack(&s);
  printf("Please input a decimal number:");
  scanf("%u",&n);
  while(n)
  {
    Push(&s,n%8);
    n=n/8;
  }
  printf("The corresponding octal number is:");
  while(!StackEmpty(s))
  {
    Pop(&s,&e);
    printf("%d",e);
  }
  printf("\n");
  DestroyStack(&s);
}

//取栈顶的数据元素
Status GetTop(SqStack S,SElemType *e)
 {
   if(S.top>S.base)
   {
     *e=*(S.top-1);
     return OK;
   }
   else
     return ERROR;
 }

//按照四则运算法则定义的运算符（包括：＋、－、*、/、()、＃）运算的优先关系，即按照表 3.1
//判断输入的运算符与运算符栈中的运算符的优先关系
```

表 3.1 的运算符间的优先关系是按照四则运算法则定义的，也就是本函数的运行结果。如果读者希望在运算中再加入其他运算符（如乘方等），则必须先定义它们的关系，丰富表 3.1 的内容，同时在函数中添加所加入的运算符的比较内容。其中 θ_1 为栈顶元素，θ_2 为输入元素。定义“#”运算符为输入表达式时的结束符及存放运算符的栈的栈底元素。

本函数不包含任何数据结构的内容，仅是在输入表达式的过程中，按照表 3.1 判断运算符栈的栈顶元素与输入的运算符的关系（＜、＞、＝）。因一个表达式包含多个运算符，故将此部分作为一个独立的函数进行编写。

表 3.1　运算符间的优先关系

θ_1 \ θ_2	+	−	*	/	(	)	#
+	>	>	<	<	<	>	>
−	>	>	<	<	<	>	>
*	>	>	>	>	<	>	>
/	>	>	>	>	<	>	>
(	<	<	<	<	<	=	
)	>	>	>	>		>	>
#	<	<	<	<	<		=

```
SElemType Precede(SElemType t1,SElemType t2)
 {
   SElemType f;
   switch(t2)
   {
     case '+':
     case '-':if(t1=='('||t1=='#')
                f='<';
              else
                f='>';
              break;
     case '*':
     case '/':if(t1=='*'||t1=='/'||t1==')')
                f='>';
              else
                f='<';
              break;
     case '(':if(t1==')')
              {
                printf("Input Error\n");
                exit(ERROR);
              }
              else
                f='<';
              break;
     case ')':switch(t1)
              {
                case '(':f='=';
                         break;
                case '#':printf("Input ERROR\n");
                         exit(ERROR);
                default: f='>';
              }
              break;
     case '#':switch(t1)
              {
                case '#':f='=';
                         break;
```

```
                case '(':printf("Input Error\n");
                        exit(ERROR);
                default: f='>';
              }
    }
    return f;
  }

 //判断 c 是否为运算符（运算符包括：＋、－、*、/、()、＃）
  Status In(SElemType c)
  {
    switch(c)
    {
      case'+':
      case'-':
      case'*':
      case'/':
      case'(':
      case')':
      case'#':return TRUE;
      default:return FALSE;
    }
  }

 //计算 a 和 b，返回运算结果
 SElemType Operate(SElemType a,SElemType theta,SElemType b)
  {
    SElemType c;
    switch(theta)
    {
      case'+':c=a+b;
              break;
      case'-':c=a-b;
              break;
      case'*':c=a*b;
              break;
      case'/':c=a/b;
    }
    return c;
  }

 //表达式求解
```

在栈的基本操作集中，主要就是入栈和出栈操作，而且栈的数据元素的特点是后进先出，表达式求解的应用正是利用栈的这个特性。

在对这个应用进行分析时，设计了中间函数，如算符优先级别判断，计算每次运算的结果等。

在这个函数中定义了两个栈，一个栈用来存放操作数，另一个栈用来存放运算符，由此又设计了对输入字符的判断函数。这样就使得表达式求解的函数更直观、简单、易理解。

该函数处理过程有一定难度，其流程图如图 3.2 所示。

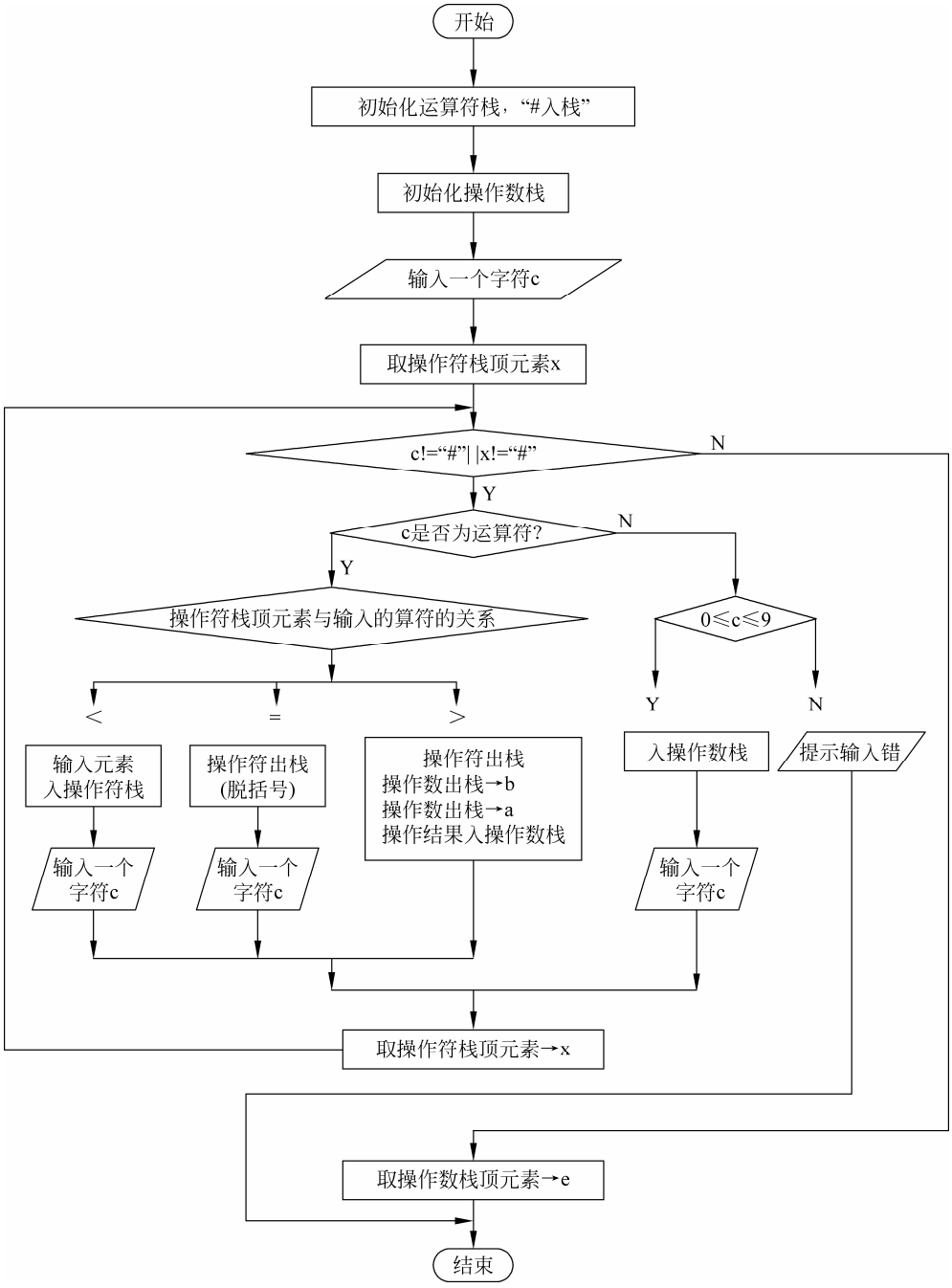

图 3.2　表达式求解的流程图

```
SElemType EvaluateExpression(SElemType *e)
 {
   SqStack OPTR,OPND;
   SElemType a,b,c,x,theta;

   InitStack(&OPTR); Push(&OPTR,'#');
   InitStack(&OPND); c=getchar();
```

```
  GetTop(OPTR,&x);
  while(c!='#'||x!='#')
  {
    if(In(c))
      switch(Precede(x,c))
      {
        case'<':Push(&OPTR,c);
                c=getchar();
                break;
        case'=':Pop(&OPTR,&x);
                c=getchar();
                break;
        case'>':Pop(&OPTR,&theta);
                Pop(&OPND,&b);
                Pop(&OPND,&a);
                Push(&OPND,Operate(a,theta,b));
                break;
      }
    else if(c>='0'&&c<='9')
    {
      Push(&OPND,c-48);
      c=getchar();
    }
    else
    {
      printf("Input Error\n");
      return ERROR;
    }
    GetTop(OPTR,&x);
  }
  GetTop(OPND,&x);
  *e=x;
  return OK;
 }
//主函数
```

本实验实现两个功能，一是十进制转换成八进制，二是实现 10 以内个位数字的加、减、乘、除运算。

```
main()
{
  char chs[256],ch;
  SElemType e=0;

  clrscr();
  printf("\n1:Decimal-octal Conversion");
  printf("\n2:Numeric Expression Evaluator");
  printf("\n0:Exit\n");

  printf("\ninput function select:");
  gets(chs);
```

```
 while (1)
 {
  if (strlen(chs)==1)
  {
    ch=chs[0];
    switch (ch)
    {
      case '1':conversion();
          break;
      case '2':printf("Input expression ended by '#'.\n");
          printf("(Notice: Operand should be 0-9):");
          if(EvaluateExpression(&e))
            printf("%d\n",e);
          break;
      case '0': return OK;
      default: printf("\nBad input...\n");
    }
  }
  else
  {
     printf("\nBad input...\n");
  }
  printf("\ncontinue input function select:");
  fflush(stdin);
  gets(chs);
 }
}
```

3.4　习　　题

1．验证栈顺序存储时的基本操作集函数和利用基本操作集实现十进制和八进制转换、一位数的加减乘除的表达式求解功能。

2．在主函数中编写代码实现十进制和八进制转换功能。

3．在主函数中编写代码实现一位数表达式求解功能。