## 实验二 栈的基本操作

## **1.表达式计算**

## **1.1 需求分析**

输入输出形式：

表达式输入

输出结果

功能：利用栈结构实现个位数表达式+、—、\*、/、^、%运算求值

数据范围：0~9

测试数据：2\*(1+1)-3^2+5%2-4/2# -6

2^3-7+2\*(1+7%3-4/2)# 1

**1.2 概要设计**



**1.3 详细设计**

创建数据栈、运算符栈

输入表达式，#结束

逐个读入

数字入栈数据栈

运算符判断与栈内优先级

{

If（读入运算符优先级高）

入运算符栈

else if（相等）

弹出运算符栈顶元素

else

计算

}

读入#

输出数据栈顶数字即运算结果

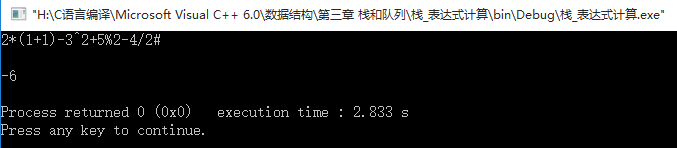
**1.4 调试分析**

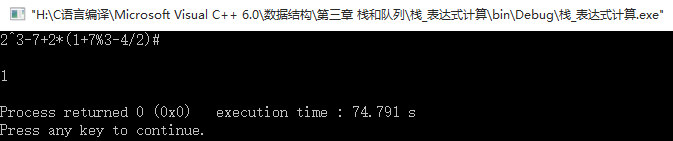
时间复杂度为n，空间复杂度由栈内数组大小确定，对于优先级的确认有两种方法，可以用二维数组，也可以用switch返回具体数字确定，一定要画流程图！

**1.5 用户使用说明**

用户可以输入包含+、—、\*、%、^的个位数表达式以#结束，回车即可看到运算结果

**1.6 测试结果**

****

****

**1.7 附录**  
//创建数组栈结构体

#define MAXSIZE 100  
typedef struct SqstackOpnd{  
    int s[MAXSIZE];  
    int top;  
}SqstackOpnd;  
  
typedef struct SqstackOptr{  
    char s[MAXSIZE];  
    int top;  
}SqstackOptr;  
  
void opndpush(SqstackOpnd opnd, int num) //数据入栈  
{  
    opnd.s[opnd.top] = num;  
    opnd.top = opnd.top + 1;  
}  
  
int opndpop(SqstackOpnd opnd)     //弹出数据栈的栈顶数据  
{  
    int num;  
    opnd.top = opnd.top - 1;  
    num = opnd.s[opnd.top];  
    return num;  
}  
  
int opndtop(SqstackOpnd opnd)     //返回数据栈的栈顶元素  
{  
    int num;  
    num = opnd.s[opnd.top-1];  
    return num;  
}  
  
void optrpush(SqstackOptr optr, char ch)   //将运算符入栈  
{  
    optr.s[optr.top] = ch;  
    optr.top = optr.top + 1;  
}  
  
char optrpop(SqstackOptr optr)   //弹出运算符栈的栈顶元素  
{  
    char ch;  
    optr.top = optr.top - 1;  
    ch = optr.s[optr.top];  
    return ch;  
}  
  
char optrtop(SqstackOptr optr)  //返回运算符栈的栈顶元素  
{  
    char ch;  
    ch = optr.s[optr.top-1];  
    return ch;  
}  
  
int Isp(char ch)     //栈内运算符的优先级  
{  
 switch(ch)  
 {  
 case '#'  :  return 0;  
 case '('  :  return 1;  
 case '+'  :  return 3;  
 case '-'  :  return 3;  
 case '\*'  :  return 5;  
 case '/'  :  return 5;  
 case '%'  :  return 5;  
 case '^'  :  return 7;  
 case ')'  :  return 8;  
 }  
}  
  
int Osp(char ch)     //栈外运算符的优先级  
{  
 switch(ch)  
 {  
 case '#'  :  return 0;  
 case '('  :  return 8;  
 case '+'  :  return 2;

case '-'  :  return 2;  
case '\*'  :  return 4;  
case '/'  :  return 4;  
case '%'  :  return 4;  
case '^'  :  return 6;  
case ')'  :  return 1;

}  
}

//获取栈内外运算符优先级  
char getPriority(char inoptr, char outoptr)  
{  
    char ch;  
    if(Isp(inoptr)<Osp(outoptr))  
        ch = '<';  
    else if(Isp(inoptr)==Osp(outoptr))  
        ch = '=';  
    else  
        ch = '>';  
       return ch;  
}  
  
int calculate(int a, char op, int b)  //计算a op b  
{  
    switch(op)  
    {  
        case '+' : return a+b;  
        case '-' : return a-b;  
        case '\*' : return a\*b;  
        case '/' : return a/b;  
        case '%' : return a%b;  
        case '^' : return pow(a, b);  
        default : break;  
    }  
}  
  
void initOpnd(SqstackOpnd opnd)  //初始化数据栈  
{  
    opnd.top = 0;  
}  
  
void initOptr(SqstackOptr optr)   //初始化运算符栈  
{  
    optr.top = 1;  
    optr.s[0] = '#';  
}  
  
//计算表达式，得到结果  
int getAnswer(SqstackOpnd opnd, SqstackOptr optr)  
{  
  char c = getchar();  
  while (c != '#' || optrtop(optr) != '#') //终止条件  
  {  
    if (isdigit(c))   //如果c在'0'~'9'之间  
    {  
      opndpush(opnd, c);     //将c对应的数值入栈opval  
      getchar();  
    }  
    else  
    {  
      switch(getPriority(optrtop(optr), c));

//获取运算符栈optr栈顶元素与c之间的优先级  
    {  
      case '<' :    // <则将c入栈optr  
      {  
         optrpush(optr, c);  
          getchar();  
          break;  
       }  
       case '=' : //=则将optr栈顶元素弹出，用于括号的处理  
       {  
         optrpop(optr);  
         getchar();  
         break;  
       }  
      case '>' :    // >则计算  
      {  
        char op = optrpop(optr);  
        int b = opndpop(opnd);  
        int a = opndpop(opnd);  
        opndpush(opnd, calculate(a, op, b));  
      }  
     }  
   }  
  }  
    return opndtop(opnd);    //返回opnd的栈顶元素的值  
}  
  
int main()  
{  
    SqstackOpnd opnd;  
    SqstackOptr optr;  
  
    initOpnd(opnd); //初始化栈  
    initOptr(optr);

getchar(); //输入表达式  
    int answer;  
    answer = getAnswer(opnd, optr); //运算结果

cout<<endl<<answer<<endl;

    return 0;  
}