# 数据结构
# 实验报告




| | | | |
|---|---|---|---|
| 姓　　名 | 薛旗 | 学　　号 | 20155362 |
| 班　　级 | 软信-1503 | 指 导 教 师 | 马毅 |
| 实 验 名 称 | 数据结构实验 | | |
| 开 设 学 期 | 2016-2017 第 一 学 期 | | |
| 开 设 时 间 | 第 1 周——第 10 周 | | |
| 报 告 日 期 | 2016.11.28 | | |
| 评 定 成 绩 | | 评　　定　　人 | |
| | | 评 定 日 期 | |

东北大学软件学院

# 实验二：栈和队列、串和数组的应用

## 实验目的：

1. 掌握栈、队列、串和数组的抽象数据类型的特征。
2. 掌握栈、队列、串和数组的抽象数据类型在计算机中的实现方法。
3. 学会使用栈、队列来解决一些实际的应用问题。

## 实验内容：

算术表达式求值

### [问题描述]

表达式计算是实现程序设计语言的基本问题之一。也是栈的应用的一个典型例子。设计一个程序，演示用算符优先法或转换成后缀表达式方法对算术表达式进行求值的过程。

### [基本要求]

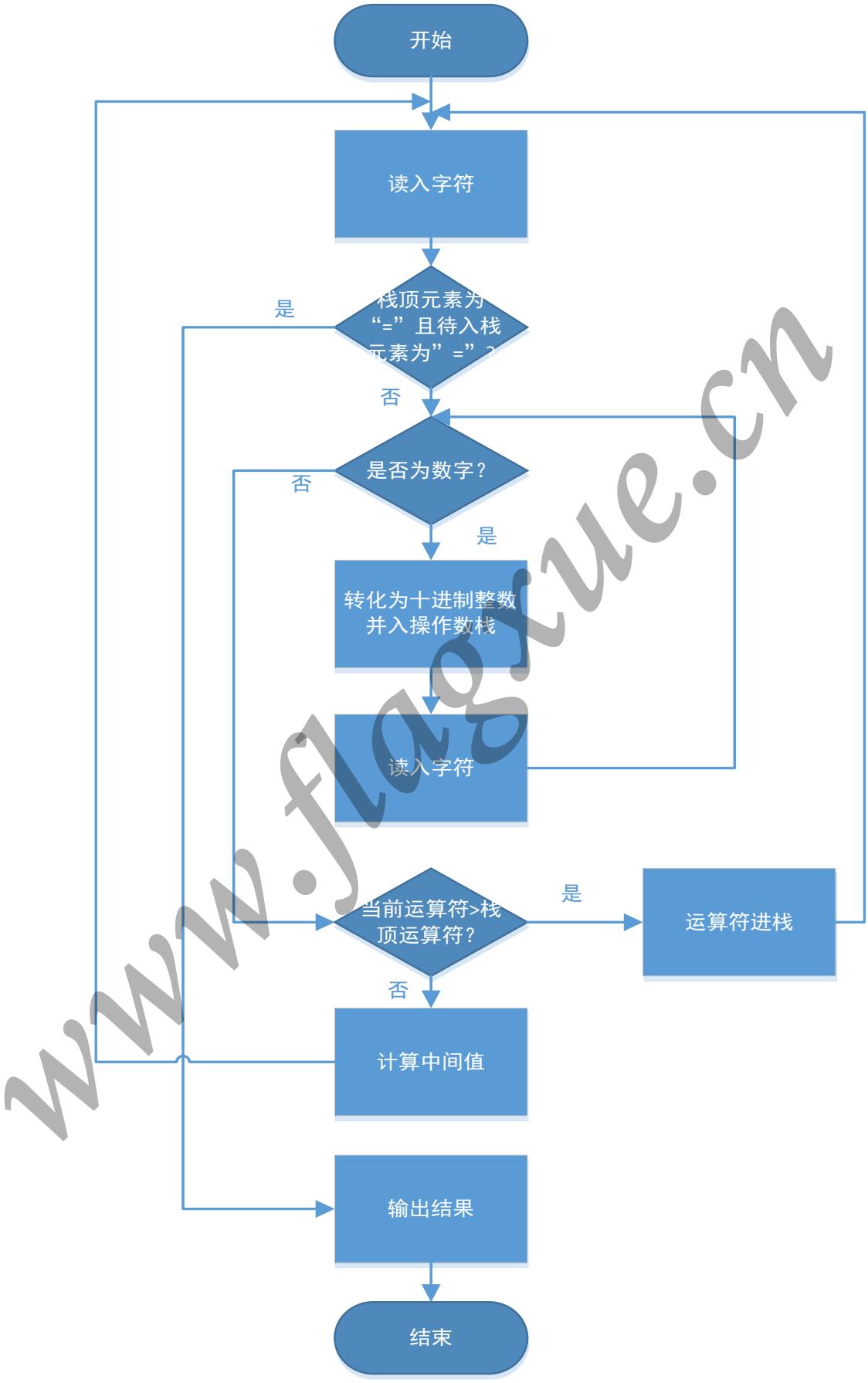
以字符序列的形式从终端输入语法正确的、不含变量的整数表达式。利用教科书表 3.1 给出的算符优先关系，或课件中给出的中缀表达式向后缀表达式转换的方法，实现对算术四则混合运算表达式的求值，并仿照教科书和课件中的例子演示求值中运算符栈、运算数栈、输入字符和主要操作变化的过程。

### [测试数据]

3*(7-2)　1+2+3+4　88-1*5　1024/4*8　(6+2*(3+6*(6+6)))

## 实验步骤：

### 1. 流程图



## 2. 函数说明

main():从键盘依次读入字符，若读到的字符是数字，则将字符转化为整型并将处理后的数字进操作数栈；若读到的字符为运算符，则比较栈顶运算符与当前运算符的优先级。如果栈顶运算符优先级较高，则先计算中间值；如果当前运算符优先级较高，则将运算符进栈，继续读取下一个字符。继续执行循环体，当且仅当栈顶元素为“=”且待进栈元素为“=”号时，表明其间的表达式计算完毕，退出循环。将操作数栈元素出栈，此时该元素即为计算结果。

InitStack(SqStack &S):顺序栈初始化

Push(SqStack &S, ElemType ch):入栈

Pop(SqStack &S, ElemType &ch):出栈

GetTop(SqStack S):返回 S 的栈顶元素，不修改栈顶指针

in(char c, char *ch):检验字符 c 是否位于字符串 ch 中

precede(char a, char b):比较运算符 a 和 b 的优先级

compute(int a, char oper, int b):计算中间值

## 3. 源代码

```
#include<stdio.h>
#include<malloc.h>
#include<assert.h>
#include<stdlib.h>

#define STACK_INIT_SIZE 20
#define STACKINCREMENT 5

typedef union
{
  char character;
  int number;
}ElemType;

typedef struct
{
  ElemType *base;
```

```
  ElemType *top;
  int stacksize;
}SqStack;

void InitStack(SqStack &S)        //顺序栈初始化
{
  S.base = (ElemType *)malloc(sizeof(ElemType)*STACK_INIT_SIZE);
  if (!S.base)
  {
      printf("内存分配失败！");
      exit(-1);
  }
  S.top = S.base;
  S.stacksize = STACK_INIT_SIZE;
}

bool Push(SqStack &S, ElemType ch)
{
  if (S.top - S.base >= S.stacksize)
  {
      S.base = (ElemType *)realloc(S.base, sizeof(ElemType)*(S.stacksize +
STACKINCREMENT));
      S.stacksize = S.stacksize + STACKINCREMENT;
  }
  *S.top++ = ch;
  return true;
}

bool Pop(SqStack &S, ElemType &ch)
{
  if (S.top == S.base)
      return false;
  ch = *--S.top;
  return true;
}

ElemType GetTop(SqStack S)//返回S的栈顶元素，不修改栈顶指针
{
  if (S.top != S.base)
      return *(S.top - 1);
}

bool in(char c, char *ch)//检验字符c是否位于字符串ch中
{
```

```
    while (*ch&&*ch != c)
        ch++;
    if (*ch)
        return true;
    return false;
}

char precede(char a, char b)//比较运算符a和b的优先级
{
    switch (a)
    {
    case '+':
    case '-':
        if (in(b, "+-)="))
            return '>';
        else if (in(b, "*/("))
            return '<';
        break;
    case '*':
    case '/':
        if (in(b, "+-*/)="))
            return '>';
        else if (in(b, "("))
            return '<';
        break;
    case '(':
        if (in(b, "+-*/("))
            return '<';
        else if (in(b, ")"))
            return '=';
        break;
    case ')':
        if (in(b, "+-*/)="))
            return '>';
        break;
    case '=':
        if (in(b, "+-*/("))
            return '<';
        else if (in(b, "="))
            return '=';
        break;
    }
}
```

```
int compute(int a, char oper, int b)//计算中间值
{
  switch (oper)
  {
  case '+':
       return a + b;
  case'-':
       return a - b;
  case'*':
       return a * b;
  case '/':
       assert(b != 0);
       return a / b;

  }
}

void main()
{
  printf("------------------说明-----------------\n");
  printf("* 开发环境: Microsoft Visual C++ 6.0\n");
  printf("* 表达式必须正确，且仅限整数运算\n");
  printf("* 表达式结尾以\"=\"结束,按回车输出结果\n");
  printf("---------------------------------------\n");

  ElemType ch, ch_prior, temp;//temp是用于进栈和出栈的临时变量
  ElemType left, right;
  ElemType result;
  SqStack OPTR, OPND;//运算符栈和操作数栈
  char *num = "0123456789";
  char *oper = "+-*/()=";

  InitStack(OPTR);
  temp.character = '=';
  Push(OPTR, temp);
  InitStack(OPND);
  ch.character = getchar();

  //循环停止条件：栈顶元素为'=',并且当前待进栈元素为'=',表明其间的表达式计算完毕
  while (!(GetTop(OPTR).character == '=' && ch.character == '='))
  {
       temp.number = 0;//置初值
       while (in(ch.character, num))//若当前输入的字符为数字
       {
```

```
            temp.number = temp.number * 10 + (ch.character - '0');
            ch.character = getchar();//继续读取下一个字符
            if (!in(ch.character, num))//若当前字符不是数字，则将temp入操作数栈
            {
                Push(OPND, temp);
                temp.number = 0;
            }
        }
        //如果当前字符为运算符，比较栈顶运算符与当前运算符的优先级
        switch (precede(GetTop(OPTR).character, ch.character))
        {
        case '>'://若栈顶运算符优先级较高，则先计算中间值
            Pop(OPND, right);
            Pop(OPND, left);
            Pop(OPTR, ch_prior);
            temp.number = compute(left.number, ch_prior.character, right.number);
            Push(OPND, temp);
            break;
        case '<'://当前运算符优先级较高，将运算符进栈，继续读取下一个字符
            Push(OPTR, ch);
            ch.character = getchar();
            break;
        case '=': //两个运算符优先级相同表明，栈顶运算符为'('，当前运算符为')'
            Pop(OPTR, ch_prior);
            ch.character = getchar();
            break;
        }
    }
    Pop(OPND, result);//操作数栈元素即为计算结果
    printf("%d\n", result.number);
    system("pause");
}
```

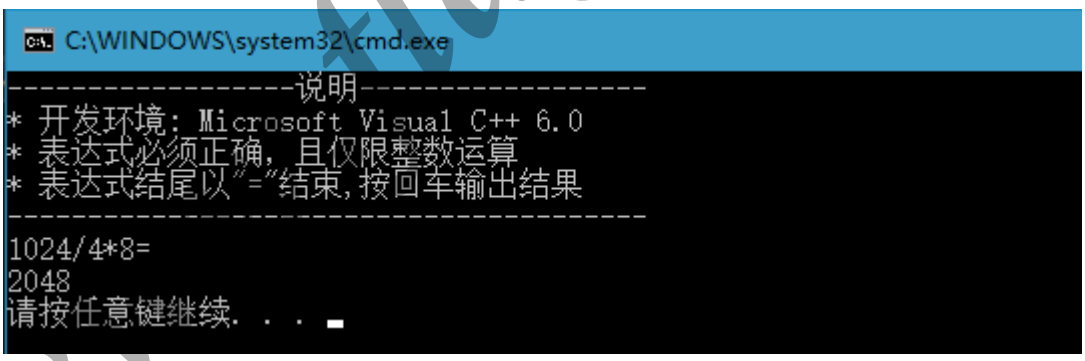
# 实验结果：

## 测试一：

测试二：


测试三：


测试四：




测试五：

## 实验总结：

表达式求值是栈应用的一个典型例子。“算符优先法”，是一种简单直观、广为使用的表达式求值算法。。

要把一个表达式翻译成正确求值的一个机器指令序列，或者直接对表达式求值，首先要能够正确解释表达式。算符优先法就是根据算术四则运算规则确定的运算有限关系，实现对表达式的编译或解释执行的。在表达式计算中先出现的运算符不一定先运算，具体运算时机的确定通过栈来完成。将扫描到的不能进行运算的运算数和运算符先分别压入运算数栈和运算符栈中，在条件满足时再分别从栈中弹出进行运算。

## 教师评语或评价表格：（任课教师可根据实际情况，做适当调整）

评语及评价表格的字体颜色为红色

评价表格：

| 评价内容 | 具 体 要 求 | 分值 | 得分 |
|---|---|---|---|
| 平时表现 | 课程设计过程中，无缺勤现象，态度积极，具有严谨的学习态度和认真、踏实、一丝不苟的科学作风。 | 10 | |
| 提交材料 | 能够按照规范提交课程设计要求的所有材料（要求在以“学号-姓名”命名的文件夹中，包含实验报告电子版和实验源代码等），材料完备，格式内容等符合要求。 | 10 | |
| 报告质量 | 实验报告格式规范，符合要求；报告内容充实、正确，实验目的归纳合理到位。 | 30 | |
| 实验内容 | 能够按实验要求合理设计并开发出程序，功能完整性强，原理及实验结果分析准确，归纳总结充分。 | 50 | |
| 总 分 | | | |