山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201900161140 | 姓名： 张文浩 | | 班级： 19级人工智能 |
| 实验题目：栈 | | | |
| 实验学时：2 | | 实验日期： 9.28 | |
| 实验目的：   1. 掌握栈结构的定义与实现； 2. 掌握栈结构的使用。 | | | |
| 软件开发工具：  Vscode | | | |
| 1. 实验内容   创建栈类，采用数组描述；计算数学表达式的值。 输入数学表达式，输出表达式的计算结果。数学表达式由单个数字和运算符“+”、“-”、“\*”、“/”、“(”、“) ”构成，例如 2+3\*(4+5)–6/4。  输入输出格式：  输入：  第一行一个整数n(1<=n<=100)，代表表达式的个数。  接下来n行，每行一个表达式，保证表达式内的数字为单个整数，表达式内各运算符和数字间没有空格，且表达式的长度不超过2000。  输出：  每行表达式输出一个浮点数，要求保留两位小数，保证输入表达式合法。   1. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）   定义一个类class，arrayStack。  其中的成员变量包括，stackTop当前栈顶，arrayLength栈容量，以及\*stack栈数组的头指针。  类中封装的方法（函数）包括，构造函数，析构函数，empty（）判断栈是否为空的函数、size（）获得栈的当前大小的函数、top（）获得栈顶的元素、pop（）弹出栈顶的元素、push（x）将x压入栈的操作。  **arrayStack(int initialCapacity)构造函数：**  初始化栈的容量为initialCapacity，为stack数组开出栈大小的空间，并将栈顶stacktop置为-1  **arrayStack<T>::push(const T &x)压栈操作：**  先比较当前栈顶索引位置和栈的容量大小，如果发现stack满了，就为栈开大一点空间。为了避免频繁的发生容量不够，每次为栈多开一倍的空间。  **最关键的是calculate(string a)函数：**  分别建立一个数字栈num，和符号栈op，分别存数字和操作符号。  然后开始读取数学表达式，按照读到的字符的种类进行分类处理  如果读到的是数字，判断是几位数字，并将这个数字push进数字栈  如果读到的是加减，因为加减的优先级最低，因此只要遇到加减号，无论操作符栈中的什么运算符都要运算  如果读到的是左括号，直接压入操作栈  如果读到的是右括号，就一直对符号栈中的操作进行运算，直到遇到左括号  如果读到的是乘除，因为优先级高于加减，因此要判断最上面的是否是乘除，如果是乘除就运算，否则就直接进栈  遍历结束后，当栈中不是空的时候就继续运算，直到栈中空为止  最终的运算结果就在num数字栈顶   1. 测试结果（测试输入，测试输出）     C:\WINDOWS\TEMP\WeChat Files\4cd31641cd2fff2d630ab09ad4348cf.jpgC:\WINDOWS\TEMP\WeChat Files\6c2bba3e3d90f2f897591925e068bfc.jpg   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   在本次实验中，我实现了栈stack的定义及封装操作，充分了解栈的定义方法以及特性。并通过利用栈实现计算数学表达式，可以利用栈实现数学表达式的先后顺序关系问题。  栈是允许在同一端进行插入和删除操作的特殊线性表。允许进行插入和删除操作的一端称为栈顶(top)，另一端为栈底(bottom)；栈底固定，而栈顶浮动；栈中元素个数为零时称为空栈。插入一般称为进栈（PUSH），删除则称为退栈（POP）。栈也称为先进后出表。  栈可以用来在函数调用的时候存储断点，做递归时要用到栈！  创建栈类的难点在于，要考虑到数组可能是变化的。因此数组需要用new来新建，而且当数组容量即将达到上限时要及时倍增其容量，并且要记得在析构函数时将新建数组用delete删去。在处理数字表达式的运算时要考虑运算符的优先顺序。使用两个栈分别存储数字和符号可以使得运算更加易于进行。在第一次编写计算函数operate以及cal的时候，我只建立了一个栈存储运算符号，没有考虑到最终的结果如何存放，因此导致了写出的程序过于复杂。在编写程序时，还应该善于使用调试功能，这样可以节省大量编写的时间。例如这次题目中的数值计算，使用调试功能查看每一次进行计算的两个数字以及符号，可以更容易得出相应的算法。   1. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）   #include <iostream>  using namespace std;  template <class T>  class arrayStack  {  public:      arrayStack(int initialCapacity = 100);        //构造函数      ~arrayStack() { delete[] stack; };            //析构函数      bool empty() const { return stackTop == -1; } //判断stack是否为空      int size() const { return stackTop + 1; }     //获得stack大小      T &top() { return stack[stackTop]; }          //获得栈顶元素      void pop() { stack[stackTop--].~T(); }        //弹出栈顶元素      void push(const T &x);                        //压栈    private:      int stackTop;    //当前栈顶      int arrayLength; //栈容量      T \*stack;        //栈数组  };  template <class T>  arrayStack<T>::arrayStack(int initialCapacity)  {      arrayLength = initialCapacity;      stack = new T[arrayLength];      stackTop = -1;  }  template <class T>  void arrayStack<T>::push(const T &x)  {      if (stackTop == arrayLength - 1) //stack满了，需要开一个更大的空间      {          T \*big = new T[2 \* arrayLength];          copy(stack, stack + arrayLength, big);          delete[] stack;          stack = big;          arrayLength \*= 2;      }      stack[++stackTop] = x;  }  double calculate(string a)  {      arrayStack<double> num; //数字栈      arrayStack<char> op;    //操作栈      double number;      for (int i = 0; i < a.size(); i++)      {          //如果是数字的话直接进栈          if (a[i] >= '0' && a[i] <= '9')          {              number = a[i] - '0';              while (a[i + 1] >= '0' && a[i + 1] <= '9') //判断是几位数字，并转换成整型存储              {                  number = 10 \* number + (a[i + 1] - '0');                  i++;              }              num.push(number); //数字进栈          }          else if (a[i] == '+' || a[i] == '-') //如果是加减的话，因为加减的优先级最低，因此只要遇到加减号，无论操作符栈中的什么运算符都要运算          {              while (!op.empty() && (op.top() == '+' || op.top() == '-' || op.top() == '\*' || op.top() == '/'))              {                  if (op.top() == '+')                  {                      number = num.top();                      num.pop();                      number += num.top();                      num.pop();                  }                  else if (op.top() == '-')                  {                      number = -num.top();                      num.pop();                      number += num.top();                      num.pop();                  }                  else if (op.top() == '\*')                  {                      number = num.top();                      num.pop();                      number \*= num.top();                      num.pop();                  }                  else if (op.top() == '/')                  {                      double temp = num.top();                      num.pop();                      number = num.top() / temp;                      num.pop();                  }                  num.push(number);                  op.pop();              }              //将当前操作符进栈              op.push(a[i]);          }          //当前运算符是'('时，直接压入操作符栈          else if (a[i] == '(')          {              op.push(a[i]);          }          //当前操作符是')'时，一直清除栈中的运算符号直至左括号          else if (a[i] == ')')          {              while (op.top() != '(')              {                  if (op.top() == '+')                  {                      number = num.top();                      num.pop();                      number += num.top();                      num.pop();                  }                  else if (op.top() == '-')                  {                      number = -num.top();                      num.pop();                      number += num.top();                      num.pop();                  }                  else if (op.top() == '\*')                  {                      number = num.top();                      num.pop();                      number \*= num.top();                      num.pop();                  }                  else if (op.top() == '/')                  {                      double temp = num.top();                      num.pop();                      number = num.top() / temp;                      num.pop();                  }                  num.push(number);                  op.pop();              }              //将遇到的左括号弹出去              op.pop();          }          //当前运算符是乘除时，因为优先级高于加减，因此要判断最上面的是否是乘除，如果是乘除就运算，否则就直接进栈          else if (a[i] == '\*' || a[i] == '/')          {              while (!op.empty() && (op.top() == '\*' || op.top() == '/'))              {                  if (op.top() == '\*')                  {                      number = num.top();                      num.pop();                      number \*= num.top();                      num.pop();                  }                  else if (op.top() == '/')                  {                      double temp = num.top();                      num.pop();                      number = num.top() / temp;                      num.pop();                  }                  //将当前的结果入栈，并将运算符栈顶弹出                  num.push(number);                  op.pop();              }              //将当前的运算符压入栈              op.push(a[i]);          }      }      //遍历结束后，当栈中不是空的时候就继续运算，直到栈中空为止      while (!op.empty())      {          if (op.top() == '+')          {              number = num.top();              num.pop();              number += num.top();              num.pop();          }          else if (op.top() == '-')          {              number = -num.top();              num.pop();              number += num.top();              num.pop();          }          else if (op.top() == '\*')          {              number = num.top();              num.pop();              number \*= num.top();              num.pop();          }          else if (op.top() == '/')          {              double temp = num.top();              num.pop();              number = num.top() / temp;              num.pop();          }          num.push(number);          op.pop();      }      //返回最终结果      number = num.top();      return number;  }  int main()  {      int n;      cin >> n;      while (n--)      {          string s;          cin >> s;          // for (int i = 1; i < s.size() - 1; i++)          // {          //     if (s[i] == '(' && ((s[i - 1] >= '0' && s[i - 1] <= '9') || s[i - 1] == ')'))          //     {          //         s.insert(i, "\*");          //     }          //     if (s[i] == ')' && ((s[i + 1] >= '0' && s[i + 1] <= '9') || s[i + 1] == '('))          //     {          //         s.insert(i + 1, "\*");          //     }          // }          // cout<<s<<endl;          printf("%.2f\n",calculate(s));      }      system("pause");      return 0;  }  // 1+6/17+214+9/1+209+9+7/(95)-16-08-7-92+6-(0-5-28-7-95\*(6-5526-2-7-5+67+69-100+30+2/1-6/6+5))  // 0-4-1/6\*(1-(6/7)-4+6+2+61)-17+2-82+0-(4+6-61+(3-86/4-6-5)\*6/4/8+7-14/95)-0/6+1-0-2+7-2+64-36+2/8+6+162 | | | |