山东大学软件学院

数据结构、算法与应用课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201800301249 | 姓名： 王帅 | | 班级： 4班 |
| 实验题目：实验四 堆栈的应用 | | | |
| 实验学时：4 | | 实验日期： 2019.10.10 | |
| 实验目的：  掌握堆栈的使用。 | | | |
| 硬件环境：  PC机 | | | |
| 软件环境：  Microsoft Visual C++ | | | |
| 实验步骤与内容：   * 1. 输入一个数学表达式（假定表达式输入格式合法），计算表达式结果并输出。   2. 数学表达式由单个数字和运算符“+”、“-”、“\*”、“/”、“(、) ”构成，例如 2 + 3 \* ( 4 + 5 ) - 6 / 4。   3. 变量、输出采用整数，只舍不入。   实验代码如下：  #include <iostream>  #include <cstdio>  using namespace std;  /\*  1、输入一个数学表达式（假定表达式输入格式合法），计算表达式结果并输出。  2、数学表达式由单个数字和运算符“+”、 “-”、 “\*”、 “/”、 “(、) ”构成， 例如 2+3\*(4+5)-6/4。  3、变量、输出采用整数，只舍不入  \*/  /\*  设两个栈，一个操作数栈，一个运算符栈  当输入为操作数时就将其压入操作数栈。当输入为运算符时，则比较输入运算符和运算符栈的栈顶运算符的优先级的大小。  若输入运算符的优先级高于运算符栈顶运算符的优先级时，则将其压入到运算符栈；  反之，则将栈中的运算符弹出并从操作数栈中弹出两个操作数进行运算，将运算结果作为操作数输出到操作数栈；  然后重新比较输入运算符和更新后的栈顶运算符的优先级的大小。  注意：‘（’进栈之前优先级最高，但进栈之后其优先级最低，只有’）‘可使其出栈。  \*/  template<class T>  class arrayStack  {  public:  arrayStack() //构造方法  {  length = 20;  stack = new T[length];  stackTop = -1;  }    ~arrayStack()  {  delete [] stack;  }    bool empty() //判断是否为空  {  if(stackTop == -1) return 1;  else return 0;  }    int size() //记录栈的长度  {  return stackTop+1;  }    int topindex() //返回栈顶索引  {  return stackTop;  }    T top() //返回栈顶元素  {  if(stackTop!=-1)  {  return stack[stackTop];  }  }    void pop() //删除栈顶元素  {  if(stackTop!=-1)  {  stackTop--;  }  }    void push(T& element) //把一个元素压入栈顶  {  if(stackTop<length-1)  {  stackTop++;  stack[stackTop] = element;  }  }    int stackTop; //栈顶索引  int length; //栈长度  T \*stack; //数组名  };  int level(bool in,char s) //比较优先级的函数  {  int l;  if(in)  {  switch(s)  {  case '+':  case '-':  l=3;  break;  case '\*':  case '/':  l=5;  break;  case '(':  l=1;  break;  case ')':  l=0;  break;  }  }  else  {  switch(s)  {  case '+':  case '-':  l=3;  break;  case '\*':  case '/':  l=5;  break;  case '(':  l=7;  break;  case ')':  l=0;  break;  }  }  return l;  }  int answer(int a,int b,char m) //有两个数和一个符号，求结果  {  int r;  switch (m){  case '+':  r = b + a;  break;  case '-':  r = b - a;  break;  case '\*':  r = b\*a;  break;  case '/':  r = b/a;  break;  }  return r;  }  int main()  {  string s;  cout<<"Input"<<endl;  cin>>s;  cout<<"Output"<<endl;    int l=s.size();  int a,b,c=0;  int result;  char mark;  bool done = 0;  arrayStack<int> numberStack;  arrayStack<char> stringStack;    /\*for(int i=0;i<l;i++) //挨个查看字符串的每一个字符  {  if(isdigit(s.at(i)))  {  c=int(isdigit(s.at(i)))-48;  numberStack.push(c);  c=0;  }    if(!isdigit(s.at(i)))  {  while(!done)  {  if(stringStack.empty()) //若符号栈为空，压入第一个符号  {  stringStack.push(s.at(i));  break;  }  else //栈不为空  {  if(level(0,s.at(i))>level(1,stringStack.top())) //优先级大于栈顶  {  stringStack.push(s.at(i)); //压入栈顶，done  break;  }  else //优先级小于等于栈顶  {  if(level(0,s.at(i))-level(1,stringStack.top())!=-1) //不是 “（”和 “）”  {  a=numberStack.top();  numberStack.pop();  b=numberStack.top();  numberStack.pop();  mark=stringStack.top();  stringStack.pop();  c=answer(a,b,mark);  numberStack.push(c); //弹出了两个数和一个符号，继续和下一个栈顶符号比优先级  a=0; b=0; c=0; //a,b,c清零  }  else  {  stringStack.pop(); // 是 “（”和 “）” ，直接把 “（ ” 弹出  break;  }  }  }  }    done = 0;  }  }\*/      for(int i=0;i<l;i++)  {    switch(s.at(i))  {  case '+':  {  while(!done)  {  if(stringStack.topindex()==-1||stringStack.top()=='(') {stringStack.push(s.at(i)); done=1;}  else  {  a=numberStack.top();  numberStack.pop();  b=numberStack.top();  numberStack.pop();  mark=stringStack.top();  stringStack.pop();  c=answer(a,b,mark);  numberStack.push(c);  a=0; b=0; c=0;  }  }  break;  };  case '-':  {  while(!done)  {  if(stringStack.topindex()==-1||stringStack.top()=='(') {stringStack.push(s.at(i)); done=1;}  else  {  a=numberStack.top();  numberStack.pop();  b=numberStack.top();  numberStack.pop();  mark=stringStack.top();  stringStack.pop();  c=answer(a,b,mark);  numberStack.push(c);  a=0; b=0; c=0;  }  }  break;  };  case '\*':  {  while(!done)  {  if(stringStack.topindex()==-1||stringStack.top()=='('||stringStack.top()=='+'||stringStack.top()=='-')  {stringStack.push(s.at(i)); done=1;}  else  {  a=numberStack.top();  numberStack.pop();  b=numberStack.top();  numberStack.pop();  mark=stringStack.top();  stringStack.pop();  c=answer(a,b,mark);  numberStack.push(c);  a=0; b=0; c=0;  }  }  break;  };  case '/':  {  while(!done)  {  if(stringStack.topindex()==-1||stringStack.top()=='('||stringStack.top()=='+'||stringStack.top()=='-')  {stringStack.push(s.at(i)); done=1;}  else  {  a=numberStack.top();  numberStack.pop();  b=numberStack.top();  numberStack.pop();  mark=stringStack.top();  stringStack.pop();  c=answer(a,b,mark);  numberStack.push(c);  a=0; b=0; c=0;  }  }  break;  };  case '(':  {  stringStack.push(s.at(i));  break;  }  case ')':  {  while(!done)  {  if(stringStack.top()=='(')  {stringStack.pop(); done=1;}  else  {  a=numberStack.top();  numberStack.pop();  b=numberStack.top();  numberStack.pop();  mark=stringStack.top();  stringStack.pop();  c=answer(a,b,mark);  numberStack.push(c);  a=0; b=0; c=0;  }  }  break;}  default:  {  c=int(s.at(i))-48;  numberStack.push(c);  c=0;  break;  }    }  done=0;    }      //字符串查看完毕  while(numberStack.stackTop>0) //栈顶元素索引大于0，说明至少还有两个数  {  a=numberStack.top();  numberStack.pop();  b=numberStack.top();  numberStack.pop();  mark=stringStack.top();  stringStack.pop();  c=answer(a,b,mark);  numberStack.push(c);  a=0; b=0; c=0;  }    if(numberStack.stackTop==0) //只剩最后一个数，即为结果  {  result = numberStack.top();  }    cout<<result<<endl;  cout<<"End";    return 0;  } | | | |
| 结论分析与体会：  堆栈也是一种比较基础的数据结构，之前接触的不多，需要多加学习。 | | | |