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：201700130009 | 姓名： 张愈博 | | 班级： 计科17.3 |
| 实验题目：栈 | | | |
| 实验学时：2 | | 实验日期： 2018.11.03 | |
| 实验目的：  1.掌握栈结构的定义与表现  2.掌握栈结构的使用 | | | |
| 软件环境：  Windows 10  DEV C++ 5.11 | | | |
| 1.实验内容（题目内容，输入要求，输出要求）  (1)创建栈类，采用数组描述；  (2)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。假定表达式 输入格式合法。  (3)以一个 m\*n 的长方阵表示迷宫，0 和 1 分别表示迷宫中的通路和障碍。 设计一个程序，对任意设定的迷宫，求出一条从入口到出口的通路，或得出 没有通路的结论。 迷宫根据一个迷宫数据文件建立。迷宫数据文件由一个包含 0、1 的矩阵 组成。迷宫的通路可以使用通路上各点的坐标序列进行展示(使用图形展示最 佳)。  2.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）  **(1)**首先创建一个基础栈类，包含empty、size、top、pop、push等基本方法。  **(2)**对于计算数学表达式，我们人所习惯的表达式都是符号在两个操作数中间的中缀表达式，而计算机却难以计算这样的算式。所以需要把符合人类阅读习惯的表达式转化为能让计算机计算的表达式，这就是逆波兰表达式（又称后缀表达式），顾名思义它将运算符放于操作数后。下面是把中缀表达式转为后缀表达式的算法：  ①遇到运算数，直接输出。  ②遇到‘（’，压入栈。  ③遇到‘）’，从栈顶不断弹出并输出运算符，直到遇‘（’，把它弹出但不输出。  ④遇到运算符，首先输出‘ ’，再判断若栈为空，直接入栈；若栈非空且该运算符优先级大于栈顶运算符优先级，压入栈；若栈非空且该运算符优先级小于栈顶运算符优先级，不断从栈顶弹出并输出运算符直到遇到优先级比它大的运算符或者栈空为止。  ⑤若读完中缀表达式，把栈中剩余的运算符全部输出，并在它们前加‘ ’。  **(3)**剩下的问题就是如何解读并输出后缀表达式（注意，本表达式中不同操作数、符号间均有空格作为间隔符）。下面给出算法：  ①读到数字（含多位），存入中间数并不断更新直至读到新空格，把最终的中间数存入栈中。  ②读到空格，更新中间数为0。  ③读到运算符号，从栈顶弹出两个运算数做相应的运算，并把它们的运算结果再次存入栈中。  ④读完后缀表达式，将栈中的数弹出作为结果返回。  **(4)**对于带负号的操作数，可以先对中缀表达式进行规范化，如-8变成0-8，那么就是在中缀表达式中负号前无操作数的，要人为的加上0。  **(5)**综上所述，考虑创建一个表达式类。成员变量有用于保存中缀表达式运算符的栈oper和保存表达式的字符数组str，成员函数有中缀转后缀函数、计算后缀表达式函数、判断是否为运算符函数和计算运算符优先级的函数。  **(6)**对于迷宫老鼠问题，首先将迷宫抽象成一个0-1二维数组，其中1代表禁止通行，0代表可以通行。然后根据可以移动的方向设置一个偏移量数组，以目前的位置移动向下一个可行的位置，这里需要将已经走过的位置设上障碍物以免原地转圈，还要把已经走过的路保存在一个栈中，以便在遇到死路时及时返回到上一个位置。不断重复上述过程，直到无路可走或者找到了出口。  3.测试结果（测试输入，测试输出，结果分析）    图1 图2    图3  4.分析与探讨（结果分析，若存在问题，探讨解决问题的途径）  (1)如图1所示，计算器可以计算含多重括号、多位数字的四则运算。但是在加入负号后后缀表达式出现了错误。图2的错误是标准化时没有处理好字符数组下标问题。  （2）在迷宫老鼠从文件中读入迷宫，开始遇到了一些困难，后来经查找资料，用一个文件数组p、getline函数和sscanf函数相结合，来实现对一排元素的读入。值得一提的是sscanf函数，它以固定字符为输入源，并支持集合操作等多种功能。  （3）在输出迷宫时，由于是用字符形式从左上角开始输出，而老鼠的移动路径是保存在栈中的而且无顺序可言，所以难以输出老鼠的移动路径，除非在栈中元素是按行主次序保存的。或者在每次循环的时候都在栈中进行一次搜索，显然这里对栈中元素先进行排序时间复杂度较好。  （4）在迷宫搜索路径算法中，在没有邻近的下一步但仍有位置可以返回时，我开始直接把选择标志option置为了0，这样简单易懂但是会在后续造成冗余的判断。所以更新一种更加高明的写法可以通过计算next和here的相对位置计算出下一步的选择标志：如果栈顶元素表示的next位置与当前位置here同行时，说明前面option的值为0或2，将新的option更新为2+next.col-here.col。同理当他们同列时，前面option的值为1或3，将新的option更新为3+next.row-here.row。可以显著地减小平均时间复杂度。  5.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）  栈.h  #pragma once  #include<iostream>  template<class T>  class arrayStack  {  public:  arrayStack(int initialCapacity=10)  {//构造函数  if(initialCapacity<1)  {  std::cerr<<"初始化非法！"<<std::endl;  }  stack=new T[initialCapacity];  stackTop=-1; //代表该栈中没有元素  arrayLength=initialCapacity;  }  ~arrayStack()  {//析构函数  delete [] stack;  }  bool empty() const  {//返回true，当且仅当栈为空  if(stackTop==-1) return true;  else return false;  }  int size() const  {//返回栈中元素个数  return (stackTop+1);  }  T& top()  {//返回栈顶元素的引用  if(stackTop==-1)  {//如果为空栈  std::cerr<<"本栈为空"<<std::endl;  exit(1);  }  return stack[stackTop];  }  T pop()  {//删除栈顶元素,提供改进策略：当pop操作将栈中元素减少到不足原来1/4时，将容量减半  if(stackTop==-1)  {//如果为空栈  std::cerr<<"本栈为空"<<std::endl;  exit(1);  }  T Return=stack[stackTop];  stack[stackTop].~T(); //这里有一个问题，如果是int型变量也会被顺利删除么？  stackTop--;    if(stackTop<arrayLength/4)  {//空间过多，容量减半  T \*temp=new T[arrayLength/2];  for(int i=0;i<=stackTop;i++)  { temp[i]=stack[i]; }  delete [] stack;  stack=temp;  arrayLength=arrayLength/2;  }  return Return;  }  void push(const T& ele)  {//将元素ele压入栈顶  if(stackTop==arrayLength-1)  {//空间不足,容量加倍  T \*temp=new T[2\*arrayLength];  for(int i=0;i<=stackTop;i++)  { temp[i]=stack[i]; }  delete [] stack;  stack=temp;  /\* changeLengthlD(stack,arrayLength,2\*arrayLength); \*/  arrayLength\*=2;  }  stackTop++;  stack[stackTop]=ele;  }  private:  int stackTop; //当前栈顶  int arrayLength; //栈容量  T\* stack; //元素数组  };  表达式.h  #pragma once  #include<iostream>  #include <string.h>  #include"栈.h"  using namespace std;  //析构函数要命啊，不能delete？  template<class T>  class expression  {//表达式类:本类可以计算包含加减乘除、负号（尚未实现）、多级括号的数学表达式的值，  public:  expression(int n=70)  {//初始化设置字符串的长度  T \*str=new T[n];  memset(str,'\0',strlen(str)\*sizeof(char));  }  expression(const char a[]) //构造函数  {  int i;  int size=strlen(a);  str=new char[size+1];  for(i=0;i<size;i++) { str[i]=a[i]; }  str[i]='\0';  }  ~expression()  {  delete []str;  }  friend istream& operator>>(istream &in,expression &A)  {  in>>A.str;  return in;  }  friend ostream& operator<<(ostream &out,expression &A)  {  out<<A.str;  return out;  }  void infix\_to\_postfix()  {//本函数将中缀表达式转后缀表达式:(20+3)/1-1 -> 20 3 1 / + 1 -  //2\*(3\*(4+6/3)/2) -> 2 3 4 6 3 / + \* 2 \*  //现在还差最后的输出问题。。动态分配的字符数组怎么为什么不能用cout输出啊？  standardization();  arrayStack<char> oper; //保存运算符的字符栈  char \*ch=new char[2\*strlen(str)];  memset(ch,'\0',2\*strlen(str)\*sizeof(char)); //初始化，用'\0'填充空间  int i=0,k=0; //i是str的游标，k是ch的游标  for(i=0;str[i]!='\0';i++)  {  if(IsOperator(str[i])==false)  {//是数字  ch[k++]=str[i];  }  else  {//是符号  ch[k++]=' ';  if(str[i]=='(') oper.push('<');  else if(str[i]==')')  {  while(oper.top()!='<') { ch[k++]=oper.pop(); }  if(oper.top()=='<') oper.pop(); //弹出但不保存  }  else if(oper.empty()==true)  {//如果栈为空，直接把读到的操作符压入栈  oper.push(str[i]); //cout<<"aaa";  }  else if(Priority(str[i])>Priority(oper.top()))  {//如果读到的运算符优先级大于栈顶运算符优先级  oper.push(str[i]); //cout<<"bbb";  }  else if(Priority(str[i])<=Priority(oper.top()))  {//如果读到的运算符优先级小于等于栈顶运算符优先级  while(oper.empty()==false&&Priority(str[i])<=Priority(oper.top()))  {//在运算符栈非空时  char temp=oper.pop(); //cout<<temp<<" ";  if(temp!='('||temp!=')')  { ch[k++]=temp; ch[k++]=' '; }  }  oper.push(str[i]); //cout<<"ccc";  }  //cout<<"ddd"<<endl;  }  }  while(oper.empty()==false)  {  ch[k++]=' ';  ch[k++]=oper.pop(); //读入剩余的运算符  }  //cout<<ch;  delete [] str; //加上就错.... 为什么？  str=ch;  //cout<<str;  }  float calculate()  {//for exmple:20 3 1 / + 1 -  float pre,post,temp; //前运算数、后运算数、中间结果  arrayStack<float> num; //用来保存运算数的int栈  //下面不断更新pre和post  //这里的str已经是后缀表达式了  int k=0;  for(int i=0;str[i]!='\0';i++)  {  if(IsOperator(str[i])==false)  {//读到数字  while(IsOperator(str[i])==false)  {//读多位数字,跳出循环说明此时str[i]为空格  k=k\*10+str[i]-'0';  i++;  }  i--; //这里需要回一位，否则出错，看了好久..  num.push(k);  }  else  {  if(str[i]==' ')  {//读到了空格,将新运算数置0,然后等下一个字符  k=0;  }  else{//读到了运算符号  post=num.pop(); pre=num.pop(); //把两个运算数从栈中掏出  if(str[i]=='+') temp=pre+post; //计算中间值  if(str[i]=='-') temp=pre-post;  if(str[i]=='\*') temp=pre\*post;  if(str[i]=='/') temp=pre/post;  num.push(temp); //把中间结果压入栈  }  }  }  return num.pop();  }  bool IsOperator(char ch)  {//判断是否为运算符  switch (ch)  {  case '+':return true; break;  case '-':return true; break;  case '\*':return true; break;  case '/':return true; break;  case ' ':return true; break;  case '(':return true; break;  case ')':return true; break;  default: return false;  }  }  int Priority(char ch)  {  if(ch=='+'||ch=='-') return 1;  if(ch=='\*'||ch=='/') return 2;  if(ch=='('||ch==')') return 10;  if(ch=='<'||ch=='>') return 0; //用来表示压入栈后的'('和')'  }  void standardization()  {//把-8转成0-8  // cout<<strlen(str)<<endl;  for (int i = 0; i < strlen(str); i++)  {  if (str[i] == '-')  {  if (i == 0)  insert(str,0,'0');  else if (str[i - 1] == '(')  insert(str, i, '0');  }  }  //cout<<str<<endl;  }  void insert(char \*str,int i,char ch)  {//在字符串索引为i的位置插入字符ch  int size=strlen(str);  if(i<0||i>=size)  {  cerr<<"非法插入字符"<<endl;  exit(1);  }  int temp=strlen(str);  for(int j=strlen(str)-1;j>=i;j--)  {//我这里是一个new出来的字符串,实际长度大于strlen  str[j+1]=str[j];  }  str[i]=ch;  str[strlen(str)]='\0'; //扫尾工作  //cout<<str<<endl;  }  private:  T \*str=NULL; //字符数组保存表达式字符串(中缀）  arrayStack<T> oper; //保存运算符用堆栈  };  Main.cpp  #include<iostream>  #include <string.h>  #include"栈.h"  #include"表达式.h"  using namespace std;  int main()  {  expression<char> A("((-1)+(7)\*((5\*6))-(0-3)/(9/3))-(5+(9+4\*3))+(-2+3\*8/6\*3-(2+3\*4/6-(-2)\*4))");  cout<<A<<endl;  A.standardization();  A.infix\_to\_postfix(); cout<<endl;  cout<<"The answer is:"<<A.calculate();  return 0;  }  迷宫老鼠.cpp  #include<iostream>  #include"栈.h"  #include<fstream>  #include<string.h>  using namespace std;  static int m,n;  class position  {  public:  int row;  int col;  };  void inputMaze(int \*\*&A)  {//从文件中读入一个m\*n的二维数组迷宫  m=n=10;  A=new int\* [m+2]; //为增加边框预留空间  for(int i=0;i<m+2;i++)  {  A[i]=new int [n+2];  }  cout<<"请输入要打开的文件名：";  char ch[50];  cin>>ch;  char \*p=new char[40];  fstream fin;  fin.open("maze.txt");  if(!fin)  {  cerr<<"未找到此文件！"<<endl;  exit(1);  }  int j=1,r=1;  while(!fin.eof())  {  fin.getline(p,40); //从文件流fin中读40个字节？  //cout<<endl<<p<<endl;  sscanf(p,"%d %d %d %d %d %d %d %d %d %d",&A[r][1],&A[r][2],&A[r][3],&A[r][4],&A[r][5],&A[r][6],&A[r][7],&A[r][8],&A[r][9],&A[r][10]);  r++;  }  fin.close();  }  static arrayStack<position> path;  bool findPath(int \*\*A)  {//寻找一条从入口（1,1）到出口（m,n)的路径  int \*\*maze=new int\* [m+2];  for(int i=0;i<m+2;i++)  {  maze[i]=new int [n+2];  }  for(int i=0;i<m+2;i++)  for(int j=0;j<n+2;j++)  maze[i][j]=A[i][j];    //初始化寻路偏移量  position offset[4];  offset[0].row=0; offset[0].col=1; //右  offset[1].row=1; offset[1].col=0; //下  offset[2].row=0; offset[2].col=-1; //左  offset[3].row=-1; offset[3].col=0; //上  //初始化迷宫四周的围墙  for(int i=0;i<m+2;i++)  A[i][0]=A[i][n+1]=maze[i][0]=maze[i][n+1]=1;  for(int i=0;i<n+2;i++)  A[0][i]=A[m+1][i]=maze[0][i]=maze[m+1][i]=1;  //初始化老鼠在迷宫当前位置  position here,next;  here.row=1; here.col=1;  path.push(here);  maze[1][1]=1; //防止回到入口  int option=0;  while(here.row!=m||here.col!=n) //只有当row==m&&col==n时才脱出循环  {//这里是一步一个循环  next.row=here.row+offset[option].row;  next.col=here.col+offset[option].col;  option++;    if(maze[next.row][next.col]==0)  {//如果下一步存在  path.push(next); //下一步存入栈  here=next; //走到下一步  maze[here.row][here.col]=1; //加上障碍物  option=0; //重置选择数  }  else  {//下一步不存在：1.当前位置仍有可走路径 2.当前位置无可走路径，返回上一步继续看有无可走路径  if(option<=3) { ; }  else  {  if(path.empty()==true)  return false;  //需要回到上一步  next=path.top();  path.pop();  here=next;  option=0; //重置选择数，但是重置为0的会进行无效判断，有无更高的写法？  }  }  }  //跳出循环说明找到了出口  path.push(here);  return true;  }  void printPath(int \*\*A)  {  //输出地图  cout<<"Your maze is as followed:"<<endl;  for(int i=0;i<12;i++)  {  for(int j=0;j<12;j++)  {  if(A[i][j]==1) cout<<"█";  else cout<<" ";  }  cout<<endl;  }  //输出路径  cout<<endl<<"The path is:"<<endl;  for(int i=0;i<12;i++)  {  for(int j=0;j<12;j++)  {  if(A[i][j]==1) cout<<"█";  else if(path.maze\_pos\_search(i,j)) cout<<"·";  else cout<<" ";  }  cout<<endl;  }    }  int main()  {  int \*\*A;  inputMaze(A);  findPath(A);  printPath(A);  return 0;  } | | | |