山东大学 软件 学院

数据结构 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201400301034 | 姓名： 石兴帮 | | 班级： 软件八班 |
| 实验题目：堆栈和队列的应用 | | | |
| 实验学时：4h | | 实验日期： 2015-12-13 | |
| 实验目的：  掌握堆栈和队列的应用。 | | | |
| 硬件环境：  MacBook Pro  OS X Yosemite 10.10.3 | | | |
| 软件环境：  Xcode 6.4 | | | |
| 实验内容与设计：   1. 实验内容（题目内容，输入要求，输出要求）   1.计算数学表达式的值，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。  2.以一个m＊n的长方形表示迷宫，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个点的坐标序列进行展示（使用图形展示最佳）    3.设计电路布线最短路径  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.计算数学表达式的值：建立数字栈A和符号栈B，遇数字则进站，遇操作符时（加减乘除）时比较优先级并运算，遇左括号不处理，遇右括号停止读操作，从栈中不断取值运算知道取到左括号。  2.迷宫：从起点开始，将周围点打标记并压堆栈，不通则出栈，直到达到目标点。  3.路径：从起点开始，将周围点打标记并入队列，达到目标点时按照标记顺序输出。  3.测试结果（测试输入，测试输出）    **输入表达式：**  1+(2\*3)-4/2  5  输入迷宫大小：  3  输入迷宫：  0 0 1  1 0 1  1 0 0  0  0  1  1  0  1  1  0  0  \* \*    \*    \* \*  输入电路大小：  3  输入电路：  0 1 0  0 0 0  0 0 0  输入起始点：  1 1  输入结束点：  3 3  \*  \*  \* \* \*    4.实现源代码（程序风格清晰易理解，有充分的注释）    //  //  shiyan5.h  //  c++初体验  //  //  Created by SXB on 15/11/20.  //  Copyright (c) 2015年 apple. All rights reserved.  //    #ifndef c\_\_\_\_\_\_shiyan5\_h  #define c\_\_\_\_\_\_shiyan5\_h  using namespace std;  //MARK: 计算器部分    template<class T>  class Stack {  public:      Stack(int MaxStackSize = 10);      bool IsEmpty() const {return top == -1;}      bool IsFull() const {return top == MaxTop;}      T Top() const;      Stack<T>& Add(const T& x );      Stack<T>& Delete(T &x );      void Output(ostream& out) const;      void Output2() const;  private:      int top;      int MaxTop;      T \*stack;  };    template<class T>  Stack<T>::Stack(int MaxStackSize)  {      MaxTop = MaxStackSize -1;      stack = new T[MaxStackSize];      top = -1;  }  Stack<int> A(100);  Stack<char> B(100);  int result;  int youxianji(char c );  int calculate(char c );  void Run();    template<class T>  T Stack<T>::Top() const  {      return stack[top];  }    template<class T>  Stack<T>& Stack<T>::Add(const T&x )  {      stack[++top] = x;      return \*this;  }    template<class T>  Stack<T>& Stack<T>::Delete(T &x )  {      x = stack[top--];      return \*this;  }    template<class T>  void Stack<T>::Output(ostream& out) const  {      for (int i=0; i<top+1; i++) {          out << stack[i] << " ";      }  }    template<class T>  void Stack<T>::Output2() const  {      for (int i = 0; i<top + 1; i++) {          cout << stack[i] << " ";      }  }    template<class T>  ostream& operator<<(ostream& out,const Stack<T>& x)  {      x.Output(out);      return out;  }    void Run(){      char ch,meiyongle;      cout << "输入表达式："<<endl;      while (cin) {          cin.get(ch);          switch (ch ) {              case '(':                  B.Add(ch);                  break;              case ')':                  while (!(B.Top()=='(')) {                      A.Add(calculate(B.Top()));                      B.Delete(meiyongle);                  }                  B.Delete(meiyongle);                  break;              case '+':              case '-':              case '\*':              case '/':                  while (!(B.IsEmpty()) && !(B.Top()=='(')) {                      //cout << A << endl;                      int c = youxianji(B.Top());                      int d = youxianji(ch );                      if (c >= d) {                          int result = calculate(B.Top());                          A.Add(result);                          B.Delete(meiyongle);                          //cout << A <<endl;                      }                      else                          break;                  }                  B.Add(ch );                  break;              case '1':              case '2':              case '3':              case '4':              case '5':              case '6':              case '7':              case '8':              case '9':                  A.Add(ch-48);                  break;                case '\n':              case '\r':                  //cout << A << endl;                  //cout << B << endl;                  //cout << calculate(B.Top());                  while ( !B.IsEmpty()) {                      result = calculate(B.Top());                      A.Add(result);                      char meiyongle2;                      B.Delete(meiyongle2);                  }                  //cout<<A.Top()<<endl;                  cout << result << endl;                    break;                default:                  //                return;                  break;            }      }    }    int youxianji(char c )  {      switch (c ) {          case '+':          case '-':              return 1;          case '\*':          case '/':              return 2;            default:              exit(1);      }  }    int calculate(char c )  {      int a,b,meiyongle;      b=A.Top();  A.Delete(meiyongle);      a=A.Top();  A.Delete(meiyongle);      switch (c ) {          case '+':              return a+b;          case '-':              return a-b;          case '\*':              return a\*b;          case '/':              return a/b;          default:              exit(1);      }  }    // MARK:   迷宫  struct Position  {      int row;      int col;  };  int \*\*maze,m;  char \*\*lujin;  Stack<Position> \*path;    bool FindPath()  {      //    寻找从位置(1,1) 到出口(m,m)的路径      //    如果成功则返回true， 否则返回false      //    如果内存不足则引发异常NoMem          path = new Stack<Position>(m\*m -1);      //    对偏移量进行初始化          Position offset[4];          offset[0].row = 0;     offset[0].col = 1; // 向右          offset[1].row = 1;     offset[1].col = 0; // 向下          offset[2].row = 0;     offset[2].col = -1; // 向左          offset[3].row = -1;    offset[3].col = 0; // 向上      //    在迷宫周围增加一圈障碍物          for (int i=0; i< m+2; i++) {              maze[0][i] = maze[m+1][i] = 1; // 底和顶              maze[i][0] = maze[i][m+1] = 1; // 左和右          }          Position here;          here.row = 1;          here.col = 1;          maze[1][1] = 1; // 阻止返回入口          int option = 0;          int LastOption = 3;      //    寻找一条路径          while (here.row != m || here.col != m) //  不是出口          {      //        寻找并移动到一个相邻位置              int r = 0,c = 0;              while (option <= LastOption) {                  r = here.row + offset[option].row;                  c = here.col + offset[option].col;                  if (maze[r][c] == 0) {                      break;                  }                  option++; //  下一个选择              }      //        找到一个相邻位置了吗？              if (option <= LastOption) {      //            移动到maze[r][c]                  path->Add(here);                  here.row = r;                  here.col = c;      //            设置障碍物以阻止再次访问                  maze[r][c] = 1;                  option = 0;              }              else              {      //            没有可用的相邻位置，回溯                  if (path->IsEmpty()) {                      return false;                  }                  Position next;                  path->Delete(next);                  if (next.row == here.row) {                      option = 2 + next.col - here.col;                  }else                      option = 3 + next.row - here.row;                  here = next;              }          }      Position last;      last.row = m;      last.col = m;      path->Add(last);      while (!path->IsEmpty()) {          Position tmp;          path->Delete(tmp);          lujin[tmp.row-1][tmp.col-1] = '\*';      }          return true; //  到达迷宫出口  }    //  MARK:  最短电路  template<class T>  class Queue {  public:      Queue(int MaxQueueSize = 10);      bool IsEmpty() const {          return front == rear;      }      bool IsFull() const {          return (((rear+1) % MaxSize == front) ? 1: 0);      }      T First() const;      T Last() const;      Queue<T>& Add(const T& x);      Queue<T>& Delete( T& x);  private:      int front;      int rear;      int MaxSize;      T \*queue;  };    template<class T>  Queue<T>::Queue(int MaxQueueSize)  {      MaxSize = MaxQueueSize + 1;      queue = new T[MaxSize];      front = rear = 0;  }    template<class T>  T Queue<T>::First() const  {      return queue[(front +1) % MaxSize];  }    template<class T>  T Queue<T>::Last() const  {      return queue[rear];  }    template<class T>  Queue<T>& Queue<T>::Add(const T& x)  {      rear = (rear+1)% MaxSize;      queue[rear] = x;      return \*this;  }    template<class T>  Queue<T>& Queue<T>::Delete(T& x)  {      front = (front+1) % MaxSize;      x = queue[front];      return \*this;  }    int \*\*grid;  Position \*queuepath;  char \*\*queuelujin;  bool FindPath(Position start, Position finish, int& PathLen, Position \* &path)  {  //    寻找从start到 finish的路径  //    如果成功，则返回true，否则返回false  //    如果空间不足，则引发异常NoMem        if ((start.row == finish.row) && (start.col == finish.col)) {          PathLen = 0;          return true;      }   //  start = finish    //    初始化包围网格的“围墙”      for (int i=0; i < m+2; i++) {          grid[0][i] = grid[m+1][i] = 1; //  底和顶          grid[i][0] = grid[i][m+1] = 1; //  左和右      }    //    初始化offset      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        int NumOfNbrs = 4; //  一个网格位置的相邻位置数      Position here, nbr;      here.row = start.row;      here.col = start.col;      grid[start.row][start.col] = 2; //  封锁    //    标记可到达的网络位置      Queue<Position> Q;      do {  //        标记相邻位置          for (int i=0; i < NumOfNbrs; i++) {              nbr.row = here.row + offset[i].row;              nbr.col = here.col + offset[i].col;              if (grid[nbr.row][nbr.col] == 0) {  //                unlabeled nrb, label it                  grid[nbr.row][nbr.col] = grid[here.row][here.col] + 1;                  if ((nbr.row == finish.row) && (nbr.col == finish.col)) {                      break; //  完成                  }                  Q.Add(nbr);              }          }  //        已到达finish吗？          if ((nbr.row == finish.row) && (nbr.col == finish.col)) {              break;          }  //        未达到finish， 可移动到nbr吗？          if (Q.IsEmpty()) {              return false;   //   没有路径          }          Q.Delete(here);      }   while(true);    //    构造路径      PathLen = grid[finish.row][finish.col] - 2;      queuepath = new Position[PathLen];    //    回溯至finish      here = finish;      for (int j = PathLen-1; j>= 0; j--) {          path[j] = here;    //        加入路径表          queuelujin[here.row-1][here.col-1] = '\*';  //        寻找前一个位置          for (int i=0; i < NumOfNbrs; i++) {              nbr.row = here.row + offset[i].row;              nbr.col = here.col + offset[i].col;              if (grid[nbr.row][nbr.col] == j+2) {                  break;              }          }          here = nbr; //  移动到前一个位置      }        return true;  }            #endif  //  MARK： main方法        //#include <iostream>  //#include "shiyan5.h"  //using namespace std ;  //  MARK:  计算器部分main方法    //int main(){  //    Run();  //}    //  MARK:  迷宫main方法  //int main()  //{  //    cout << "输入迷宫大小：" << endl;  //    int siz;  //    cin >> siz;  //    m=siz;  //    maze = new int\*[siz+2];  //    for (int k=0; k<siz+2; k++) {  //        maze[k] = new int[siz+2];  //    }  //    lujin = new char\*[siz];  //    for (int k=0; k<siz; k++) {  //        lujin[k] = new char[siz];  //    }  //    for (int i=0; i<siz; i++) {  //        for (int j=0; j<siz; j++) {  //            lujin[i][j] = ' ';  //        }  //    }  //    cout << "输入迷宫：" << endl;  //    for (int i=0; i<siz; i++) {  //        for (int j=0; j<siz; j++) {  //            int tmp;  //            cin >> tmp;  //            maze[i+1][j+1] = tmp;  //        }  //    }  //  //    for (int i=0; i<siz; i++) {  //        for (int j=0; j<siz; j++) {  //            cout << maze[i+1][j+1] << "  ";  //        }  //        cout << endl;  //    }  //    if (FindPath()) {  //        for (int i=0; i<siz; i++) {  //            for (int j=0; j<siz; j++) {  //                cout << lujin[i][j] << " ";  //            }  //            cout << endl;  //        }  //    }  //}      //  MARK:  电路main方法  //int main(){  //    cout << "输入电路大小： " << endl;  //    int siz;  //    cin >> siz;  //    m = siz;  //    grid = new int\*[siz+2];  //    for (int k=0; k<siz+2; k++) {  //        grid[k] = new int[siz+2];  //    }  //    queuelujin = new char\*[siz];  //    for (int k=0; k<siz; k++) {  //        queuelujin[k] = new char[siz];  //    }  //    for (int i=0; i<siz; i++) {  //        for (int j=0; j<siz; j++) {  //            queuelujin[i][j] = ' ';  //        }  //    }  //    cout << "输入电路： " << endl;  //    for (int i=0; i<siz; i++) {  //        for (int j=0; j<siz; j++) {  //            int tmp;  //            cin >> tmp;  //            grid[i+1][j+1] = tmp;  //        }  //    }  //    Position start,finish;  //    cout << "输入起始点：" << endl;  //    cin >> start.row >> start.col;  //    cout << "输入结束点：" << endl;  //    cin >> finish.row >> finish.col;  //    int pathlen;  //    FindPath(start, finish, pathlen, queuepath);  //    queuelujin[start.row-1][start.col-1] = '\*';  //    for (int i=0; i<siz; i++) {  //        for (int j=0; j<siz; j++) {  //            cout << queuelujin[i][j] << " ";  //        }  //        cout << endl;  //    }  //} | | | |
| 结论分析与体会：  掌握了堆和队列的应用。 | | | |