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：201805130155 | 姓名： 赵雨晗 | | 班级： 18计科3班 |
| 实验题目：栈 | | | |
| 实验学时：2 | | 实验日期： 2019.10.16 | |
| 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：  Windows : Vs Code + MingW | | | |
| 1. 实验内容   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)。   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）   (1) 数学表达式计算：将表达式转换为后缀表达式，对后缀表达式进行操作时，遇到操作符则提取栈顶两个数并将运算结果压入栈顶直至栈空。中缀转后缀的方法为：遍历表达式中的元素，用符号栈暂存结果，如果为数字则存入结果线性表中，如果为运算符’)’则不断弹出符号栈中的符号存入结果线性表中直至遇到’(‘，如果为其他运算符，则将符号栈中所有比当前运算符优先级高的运算符弹出存入结果线性表中，再讲当前运算符压入符号栈。  (2) 采用深度优先搜索。对于当前点，尝试他的四周，如果有点是合法的，则进入这个点继续搜索。如果这个点是终点则遍历系统栈中的节点，倒序输出即为路径。3. 测试结果（测试输入，测试输出）   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   一开始一直WA，后来发现在使用了ios::sync\_with\_stdio的同时使用scanf造成了错误  并无问题   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<cstdio>  #include<iostream>  #include<cstring>  #include<string>  #include<algorithm>  #include<stdexcept>  using namespace std;  template <class T>  class ArrayList{  protected:  T\* \_Ele;  int Arr\_len;  int list\_size;  public:  class iterator;  iterator begin(){  return iterator(\_Ele);  }  iterator end() {  return iterator(\_Ele + Arr\_len);  }  class iterator{  protected:  T\* position;  public:  typedef bidirectional\_iterator\_tag iterator\_category;  typedef T value\_type;  typedef ptrdiff\_t difference\_type;  typedef T\* pointer;  typedef T& reference;  iterator(T\* thePosition = NULL){position = thePosition;}  T& operator\*() const{return \*position;}  T\* operator->() const{return &\*position;}  iterator &operator++(){  ++position;return \*this;  }  iterator operator++(int){  iterator old = \*this;  ++position;  return old;  }  iterator& operator--(){  --position; return \*this;  }  iterator operator--(int){  iterator old = \*this;  --position;  return old;  }  };  ArrayList(int init\_L = 10);  ArrayList(const ArrayList<T>& );  void merge(ArrayList<T> a, ArrayList<T> b);  void push\_back(const T &x);  void ch\_sort();  void output();  void reverse();  int size(){  return Arr\_len;  }  };  template <class T>  ArrayList<T> :: ArrayList(int init\_L){  list\_size = init\_L;  \_Ele = new T[init\_L];  Arr\_len = 0;  };  template <class T>  ArrayList<T> ::ArrayList(const ArrayList<T> &t) {  list\_size = t.list\_size;  \_Ele = new T[list\_size];  Arr\_len = t.Arr\_len;  copy(t.\_Ele, t.\_Ele + t.Arr\_len, \_Ele);  }  template <class T>  void ArrayList<T> ::push\_back(const T &x) {  if (list\_size == Arr\_len) {  ArrayList<T>A(\*this);  list\_size = list\_size \* 2;  delete[] \_Ele;  \_Ele = new T[list\_size];  copy(A.\_Ele + 1, A.\_Ele + Arr\_len, \_Ele);  }  \_Ele[Arr\_len++] = x;  }  template <class T>  void ArrayList<T> ::ch\_sort() {  bool sorted = false;  for (int size = Arr\_len; !sorted && (size > 1); size--){  int ind = 0;  sorted = true;  for (int i = 1; i < size; ++i)  if (\_Ele[ind] <= \_Ele[i]) ind = i;  else sorted = false;  swap(\_Ele[ind], \_Ele[size - 1]);  }  }  template <class T>  void ArrayList<T> ::output() {  for (int i = 0; i < Arr\_len - 1; ++i)cout<<\_Ele[i]<<' ';  cout<<\_Ele[Arr\_len - 1];  // cout<<endl;  }  template<typename T>  struct chainNode  {  T element;  chainNode<T>\* \_next;  chainNode(const T& \_element, chainNode<T>\* \_next = NULL) : element(\_element), \_next(\_next) {}  chainNode(const chainNode<T>\*& c) : element(c->element), \_next(c->\_next) {}  };  template<typename T>  class chain{  public:  chain(int = 10);  chain(const chain<T>&);  ~chain();  bool empty() const;  int size() const;  int find(const T&) const;  void erase(int);  void insert(int, const T&);  void clear();  void push\_back(const T&);  void reverse();  void merge(chain<T> &a, chain<T> &b);  chain<T>& operator=(const chain<T>&);  void print();  T& operator[](int);  const T& operator[](int) const;  class iterator;  class const\_iterator;  iterator begin() {return iterator(pHead -> \_next);}  iterator end() {return iterator(NULL);}  void revprint(iterator a);  const\_iterator begin() const {return const\_iterator(pHead -> \_next);}  const iterator end() const {return const\_iterator(NULL);}  class iterator{  public:  typedef forward\_iterator\_tag iterator\_category;  typedef T value\_type;  typedef ptrdiff\_t difference\_type;  typedef T\* pointer;  typedef T& reference;  iterator(chainNode<T>\* theNode = NULL) :node(theNode) {}  T& operator\*() {return node -> element;}  T\* operator->() {return &node->element; }  iterator& operator++(){  node = node -> \_next;  return \*this;  }  iterator operator++(int){  iterator old = \*this;  node = node->\_next;  return old;  }    bool operator==(const iterator right) const {return node == right.node;}  bool operator!=(const iterator right) const {return node != right.node;}  protected:  chainNode<T>\* node;  };  class const\_iterator{  public:  typedef forward\_iterator\_tag iterator\_category;  typedef T value\_type;  typedef ptrdiff\_t difference\_type;  typedef T\* pointer;  typedef T& reference;  const\_iterator(chainNode<T>\* theNode) :node(theNode) {};  const T& operator\*() { return node->element; }  const T\* operator->() { return &node->element; }  const\_iterator& operator++()  {  node = node->\_next;  return \*this;  }  const\_iterator operator++(int)  {  const\_iterator old = \*this;  node = node->\_next;  return old;  }  bool operator==(const const\_iterator right) const { return node == right.node; }  bool operator!=(const const\_iterator right) const { return node != right.node; }  protected:  chainNode<T>\* node;  };  protected:  chainNode<T>\* pHead;  chainNode<T>\* pTail;  int listSize;  void checkIndex(int) const;  };  template<typename T>  chain<T>::chain(int initialCapacity){  if (initialCapacity < 1) throw out\_of\_range("the initial Capacity of arrayList must > 0");  listSize = 0;  pHead = new chainNode<T>(T());  pTail = pHead;  }  template<typename T>  chain<T>::chain(const chain<T>& c){  pHead = new chainNode<T>(c.pHead->element);  pTail = pHead;  chainNode<T>\* sourceNode = c.pHead ->\_next;  chainNode<T>\* currentNode = pHead;  while(sourceNode != NULL){  pTail = currentNode->\_next;  sourceNode = sourceNode->\_next;  }  listSize = c.listSize;  }  template<typename T>  chain<T>::~chain(){  chainNode<T>\* currentNode = pHead ->\_next;  chainNode<T>\* deleteNode;  while(currentNode != NULL){  deleteNode = currentNode;  currentNode = currentNode ->\_next;  delete deleteNode;  }  delete pHead;  }  template<typename T>  bool chain<T>::empty() const {return listSize == 0;}  template<typename T>  int chain<T>::size() const {return listSize;}  template<typename T>  int chain<T>::find(const T& theElement) const{  int index = 0;  chainNode<T>\* currentNode = pHead ->\_next;  while(currentNode != NULL){  if (currentNode->element == theElement) return index;  currentNode = currentNode->\_next;  ++index;  }  return -1;  }  template<typename T>  void chain<T>::erase(int theIndex){  checkIndex(theIndex);  chainNode<T>\* deleteNode;  chainNode<T>\* pre = pHead;  for (int i = 0; i < theIndex; ++i) pre = pre->\_next;  if (theIndex == listSize - 1) pTail = pre;  deleteNode = pre->\_next;  pre->\_next = pre->\_next->\_next;  --listSize;  delete deleteNode;  }  template<typename T>  void chain<T>::insert(int theIndex, const T& theElement){  if (theIndex < 0 || theIndex > listSize) throw out\_of\_range("illegalIndex");  chainNode<T>\* pre = pHead;  for (int i = 0; i < theIndex; ++i) pre = pre->\_next;  pre->\_next = new chainNode<T>(theElement, pre->\_next);  if (theIndex == listSize) pTail = pre->\_next;  ++listSize;  }  template<typename T>  void chain<T>::clear(){  chainNode<T>\* currentNode = pHead->\_next;  chainNode<T>\* deleteNode;  while(currentNode != NULL){  deleteNode = currentNode;  currentNode = currentNode->\_next;  delete deleteNode;  }  listSize = 0;  pHead ->\_next = NULL;  pTail = pHead;  }  template<typename T>  void chain<T>::push\_back(const T& theElement){  pTail->\_next = new chainNode<T>(theElement, pTail->\_next);  pTail = pTail->\_next;  listSize++;  }  template<typename T>  chain<T>& chain<T>::operator=(const chain<T>& c){  if (this == &c) return \*this;  clear();  chainNode<T>\* currentNode = pHead;  chainNode<T>\* sourceNode = c.pHead->\_next;  while (sourceNode != NULL)  {  pTail = currentNode->\_next = new chainNode<T>(sourceNode->element);  currentNode = currentNode->\_next;  sourceNode = sourceNode->\_next;  }  listSize = c.listSize;  return \*this;  }  template<typename T>  T& chain<T>::operator[](int index){  checkIndex(index);  chainNode<T>\* currentNode = pHead->\_next;  for (int i = 0; i < index; ++i) currentNode = currentNode->\_next;  return currentNode->element;  }  template<typename T>  const T& chain<T>::operator[](int index) const{  checkIndex(index);  chainNode<T> \*currentNode = pHead->\_next;  for (int i = 0; i < index ; ++i) currentNode = currentNode->\_next;  return currentNode->element;  }  template<typename T>  void chain<T>::checkIndex(int theIndex) const{  if (theIndex < 0 || theIndex >= listSize)  throw out\_of\_range("the index is out of range");  }  template<typename T>  void chain<T>::print(){  typename chain<T>::iterator it = begin();  for (int i = 0; i < listSize; ++i){  printf("%d ",\*it);  it++;  }  printf("\n");  }  template<typename T>  void chain<T>::revprint(iterator a){  if (a != end()){  T v = \*a;  revprint(++a);  printf("%d ", v);  }  }  int main()  {  chain<int> a;  int T, op, v;  cin>>T;  while(T--){  scanf("%d", &op);  if (op == 1) scanf("%d", &v), a.insert(0, v);  else if (op == 2) scanf("%d", &v), a.erase(a.find(v));  else if (op == 3) scanf("%d", &v), printf("%d\n", a.find(v));  else if (op == 4) a.print();  else if (op == 5) a.revprint(a.begin()), printf("\n");  else {  int n;  chain<int> b, c;  scanf("%d", &n);  for (int i = 1; i <= n; ++i) scanf("%d", &v), b.push\_back(v);  scanf("%d", &n);  for (int i = 1; i <= n; ++i) scanf("%d", &v), c.push\_back(v);  merge(b,c).print();  }  }  system("pause");  }  } | | | |