1.3.1 为FixedCapacityStackOfStrings添加一个方法isFull()。

|  |
| --- |
| public:  {  private Item[] a;  private int N;  public FixedCapacityStack(int cap)  { a = (Item[]} new Object[cap]; }  public boolean isEmpty() {return N == 0; }  public int size() { return N; }  public void push(Item item)  { a[N++] = item; }  public Item pop()  { return a[--N]; }  } |

1.3.2 给定一下输入

输出为：was best times of the was the it

1.3.3

1. 可能
2. 不可能，最后应该是 1, 0
3. 可能
4. 可能
5. 可能
6. 不可能
7. 不可能
8. 可能

1.3.4 从标准输入读取文本流，并且使用栈判断其中的括号是否配对完整。

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <new>  #include <string>  template<typename Item>  class Stack  {  private:  class Node {  public:  Item item;  Node \*next;  };  Node \*first;  int N;  public:  bool isEmpty() { return first == nullptr; }  int size() { return N; }  void push(Item item){  Node \*oldfirst = first;  first = new Node();  first->item = item;  first->next = oldfirst;  N++;  }  Item pop()  {  Item item = first->item;  first = first->next;  N--;  return item;  }  };  bool Parentheses()  {  using namespace std;  string filename;  cin >> filename;  ifstream file(filename);  if (! file.is\_open())  {  cout << "Error opening file"; exit (1);  }  char ch;  Stack<char> st;  while(file >> ch)  {  if(ch == '(' || ch == '[' ||ch == '{')  st.push(ch);  else {  switch(ch){  case ')': { if(st.pop() != '(') return false; break; }  case ']': { if(st.pop() != '[') return false; break; }  case '}': { if(st.pop() != '{') return false; break; }  }  }  }  if(!st.isEmpty())  return false;  file.close();  return true;  }  int main()  {  std::cout << Parentheses();  return 0;  } |

1.3.5 当N为50时下面这段代码会打印什么？

110010

1.3.6 下面这对代码对队列q进行了什么操作？

先将q从头开始删除，并且将删除的数据存放在Stack中（逆序），而后，将Stack中的数据逆序存储在q中，结果q中元素没有发生变化。

*Answer*: Reverses the items on the queue.

1.3.7 为Stack添加一个方法peek(),返回栈中最近添加的元素（而不弹出他）

|  |
| --- |
| Item peek() {  return first->item;  } |

1.3.9 编写一段程序，补齐括号

|  |
| --- |
| class Solution  {    private:  vector<string> A;  int num = 0;  public:  //constructor  Solution() {}  //destructor  ~Solution() {}  //functions  void push(string s) { A.push\_back(s); }  //string pop() { return A.pop\_back(); }  bool isEmpty() { return num; }  void input() {  string num;  while(cin >> num) {  A.push\_back(num);  }  }  void print\_vec(const std::vector<string>& vec)  {  for (auto x: vec) {  std::cout << ' ' << x;  }  std::cout << '\n';  }  void prac\_1\_3\_9() {  int key1 = 0;  input();  for(std::vector<std::string>::iterator i = A.end()-1;i >= A.begin() ;i--) {  if(\*i == ")")  key1++;  else if(\*i == "+" || \*i == "-" || \*i == "\*" || \*i == "/") {  --i;  if(\*i != ")" && key1 > 0) {    --key1;  i = A.insert(i,"(");  }  }  }  print\_vec(A);  }  }; |

我的代码是存在问题，下面帖一段使用Java语言的正确使用方式

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | public class Ex\_1\_3\_09 | |  | { | |  | public static void main(String[] args) | |  | { | |  | Stack<String> ops = new Stack<String>(); | |  | Stack<String> vals = new Stack<String>(); | |  |  | |  | while (!StdIn.isEmpty()) | |  | { | |  | String s = StdIn.readString(); | |  |  | |  | if (s.equals("(")) ; | |  | else if (s.equals("+") || | |  | s.equals("-") || | |  | s.equals("\*") || | |  | s.equals("/") || | |  | s.equals("sqrt")) ops.push(s); | |  | else if (s.equals(")")) | |  | { | |  | String op = ops.pop(); | |  | String v = vals.pop(); | |  |  | |  | if (op.equals("+") || | |  | op.equals("-") || | |  | op.equals("\*") || | |  | op.equals("/")) | |  | v = String.format("( %s %s %s )", vals.pop(), op, v); | |  | else if (op.equals("sqrt")) | |  | v = String.format("( %s %s )", op, v); | |  |  | |  | vals.push(v); | |  | } | |  | else vals.push(s); | |  | //else vals.push(((Double)Double.parseDouble(s)).toString()); | |  | } | |  |  | |  | StdOut.println(vals.pop()); | |  | } | |  | } | |

思路与之前的双栈算术表达式求值算法类似，不过在这段程序里面，将两个括号以及器内容视为一个元素，这样保证了所有的右括号都能够成功匹配

1.3.10 将中序表达式转换为后序表达式

我的答案：

|  |
| --- |
| class InfixToPostfix  {  private:  vector<string> result;  vector<string> ops;  string exp;  public:  InfixToPostfix() {  while(cin >> exp) {  result.push\_back(exp);  }  }  void solution() {  for(auto i = result.begin(); i < result.end(); i++) {  if(\*i == "(")  ops.push\_back(\*i);  else if(\*i == "+" || \*i == "-" || \*i == "\*" || \*i == "/") {  while(1) {  if(ops.empty()) { ops.push\_back(\*i); break;}  else if(ops.back() == "(") { ops.push\_back(\*i); break;}  else if((\*i == "\*" || \*i == "/") && (ops.back() == "+" || ops.back() == "-")) {  ops.push\_back(\*i); break;  }  else {  cout << ops.back() << " ";  ops.pop\_back();  }    }  }  else if(\*i == ")") {  while(ops.back() != "("){  cout << ops.back() << " ";  ops.pop\_back();  }  ops.pop\_back();  }  else  cout << \*i << " ";  }  while(!ops.empty())  {  cout << ops.back() << " ";  ops.pop\_back();  }  }  }; |

Java版答案：（感觉这个答案并不是特别严谨，存在没有括号的情况可能会失败）

|  |
| --- |
| public class InfixToPostfix {  public static void main(String[] args) {  Stack<String> stack = new Stack<String>();  while (!StdIn.isEmpty()) {  String s = StdIn.readString();  if (s.equals("+")) stack.push(s);  else if (s.equals("\*")) stack.push(s);  else if (s.equals(")")) StdOut.print(stack.pop() + " ");  else if (s.equals("(")) StdOut.print("");  else StdOut.print(s + " ");  }  StdOut.println();  }  } |

1.3.11 得到一个后序表达式，求其结果。

我的答案：

|  |
| --- |
| class EvalutatePostfix{  private:  vector<int> val;  public:  EvalutatePostfix() {}  void Solution() {  string s;  while(cin >> s) {  if(s != "+" && s != "-" && s != "\*" && s != "/") {  //const char\* c = s;  val.push\_back(stoi(s));  }  else {  int a,b;  a = val.back();  val.pop\_back();  b = val.back();  val.pop\_back();  if(s == "+")  val.push\_back(a + b);  else if(s == "-")  val.push\_back(b - a);  else if(s == "\*")  val.push\_back(a \* b);  else if(s == "/")  val.push\_back(b / a);  }  }  cout << endl << val.back() << endl;;  }  }; |

1.3.12编写一个可以迭代的Stack用例，他含有一个静态的copy()方法，接受一个自负串的栈作为参数并且返回该栈的一个副本。

我的答案：

|  |
| --- |
| shared\_ptr<Stack<Item>> copy() {  shared\_ptr<Stack<Item>> Sc = make\_shared<Stack<Item>>();  Node \* s\_node = this->first;  for(int i = 0; i < this->size(); i++) {  Sc->push(s\_node->item);  s\_node = s\_node->next;  }  return Sc;  } |

1.3.13 下面哪种序列是不可能产生的?

A.Yes

B.No

C.No

D.No

1.3.14 编写一个类 ResizingArrayQueueOfStrings ,使用定长数组实现队列的抽象,然后扩展实现,使用调整数组的方法突破大小的限制。

我的答案：

|  |
| --- |
| template<typename Item>  class Solution{  private:  int n = 0;  int max = 3;  Item\* ar = new Item[max];  public:  ~Solution() {  delete [] ar;  }  void enqueue(Item data) {  if(n >= max) {  max += 10;  Item\* ar2 = new Item[max];  for(int i = 0; i < max - 10; i++) {  ar2[i] = ar[i];  }  delete [] ar;  ar = ar2;  }  ar[n] = data;  n++;//increase the number of member  }  Item dequeue() {  if(!this->isEmpty())  return ar[--n];  return 0;  }  bool isEmpty(){  if(n == 0)  return true;  return false;  }  int size() {  return max;  }  }; |

1.3.15-1.3.17 略

链表练习：

1.3.18 删除链表的尾节点

Bool delete() {

If(first == nullptr) return 0;

Else if(first->next == nullptr) first = nullptr;

Else {

auto k = first;

While(k->next->next !=nullptr) k = k->next;

1. >next = nullptr;

}

Return 1;

}

1.3.24 使用环形链表实现Queue

我的程序：

|  |
| --- |
| template<typename Item>  class prac\_1\_3\_29{  private:  class Node{  public:  Item item;  Node \*next = nullptr;  Node(Item a):item(a){}  Node(Item a,Node \*b):item(a),next(b) {}  };  int N = 0;  public:  Node \*first = nullptr;  Node \*end = nullptr;  void enqueue(Item item){  if(first == nullptr) { //one element  first = new Node(item);  end = first;  first->next = end;  N = 1;  }  else if(first->next == first) {  first->next = new Node(item);  end = first->next;  end->next = first;  N = 2;  }  else {  first = new Node(item,first);  end->next = first;  N++;  }  }  Item dequeue() {  Item re;  if(end == nullptr){ // list is empty  cout << "cannot delete NULL list\n";  exit(1);  }  else if(end->next == end) { //list has only one element  re = end->item;  delete end;  first = end = nullptr;  N = 0;  }  else {  re = end->item;  auto k = first;  while(k->next != end)  k = k->next;  k->next = first;  delete end;  end = k;  N--;  }  return re;  }  }; |

更好、更简洁的答案（Java没有delete）：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | /\* | |  | \* Add the item to the queue. | |  | \*/ | |  | public void enqueue(Item item) { | |  | Node x = new Node(); | |  | x.item = item; | |  | if (isEmpty()) | |  | x.next = x; | |  | else | |  | { | |  | x.next = last.next; | |  | last.next = x; | |  | } | |  | last = x; | |  | N++; | |  | } | |  | /\* | |  | \* Remove and return the item on the queue least recently added. | |  | \* Throw an exception if the queue is empty. | |  | \*/ | |  | public Item dequeue() { | |  | if (isEmpty()) throw new RuntimeException("Queue underflow"); | |  | Item item = last.next.item; | |  | if (last.next == last) | |  | last = null; | |  | else | |  | last.next = last.next.next; | |  | N--; | |  | return item; | |  | } | |

1.3.31

我的答案：略

与参考答案不同的地方:

|  |
| --- |
| private class DoubleNode |
|  | { |
|  | private DoublyLinkedList<Item> parent = list(); |
|  | private Item item; |
|  | private DoubleNode prev; |
|  | private DoubleNode next; |
|  | } |

在这里面加入了一个parent，是个好思路

1.3.37

我的答案：

|  |
| --- |
| int main(int argc,char \*argv[]){  Queue<int> order,buff;  int N = atoi(argv[1]);  int M = atoi(argv[2]);  for(int i = 0; i < N; i++)  order.enqueue(i);  while(!order.Earn()) {  for(int i = 1; i < M ; i++) {  buff.enqueue(order.dequeue());  }  std::cout << order.dequeue() << " ";  while(!buff.isEmpty())  order.enqueue(buff.dequeue());  }  std::cout << order.dequeue() << " ";  std::cout << "\nover\n";  return 0;  } |

网络答案（更简洁）

|  |
| --- |
| while (!queue.isEmpty()) {  for (int i = 0; i < m-1; i++)  queue.enqueue(queue.dequeue());  StdOut.print(queue.dequeue() + " ");  } |

1.3.45

网页版解答：

|  |
| --- |
| 如果某个排列可以产生，那么他产生的方式一定是唯一的：如果输出排列中的下一个整数在栈顶，则弹出，否则将他压入栈之中。  public static String[] problem2(String[] seq){ // 返回一系列操作顺序， seq是出栈顺序  Stack<String> s = new Stack<String>();  String[] ans = new String[2\*seq.length];  int N=0,p=0;  s.push(String.valueOf(N));  ans[p++] = String.valueOf(N);  N++;  for(int i=0;i<seq.length;i++){  while(N<seq.length && s.peek().compareTo(seq[i])!=0){  s.push(String.valueOf(N));  ans[p++]=String.valueOf(N);  N++;  }  if(s.peek().compareTo(seq[i])!=0) return null;  else {  s.pop();  ans[p++]="-";  }  }  return ans;  }  ---------------------  作者：BRCOCOLI  来源：CSDN  原文：https://blog.csdn.net/qq\_34446253/article/details/77852571  版权声明：本文为博主原创文章，转载请附上博文链接！  //  思路：主要是针对栈内剩余的元素以及要输出的元素惊醒对比 |

1.3.6 ：

证明否的问题：先假设是，然后判别矛盾。