**河南开封科技传媒学院实验报告**

••••••••••••••••••••••••••••••••• 密 ••••••••••••••••••••••••••••••••• 封 ••••••••••••••••••••••••••••••••• 线 •••••••••••••••••••••••••••••••••

**20 -20 学年第 学期**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名： | 刘东 | | 学号： | 2136101045 | 专业： | 软件工程 | |
| 所在学院： | | 理工学院 | | | 指导老师： | | 袁帅 |
| 实验时间： | | 2023/3/22 | | | 实验地点： | | 6503 |
| 课程名称： | | 数据结构 | | | | | |
| 实验题目： | | 第五次实验 | | | | | |
| 实验目的： | | | | | | | |
| 实验内容：  .1.尝试用代码实现普通队列，并完成如下操作  （1）输入1-10的数字组成普通线性队列；  （2）判断此时队列的队首元素；  （3）打印出队列操作后的队首元素；  （4）由于普通线性队列的限制，此时尝试使用循环队列来解决上述问题。  思考题：尝试用代码实现以下问题  1.能否使用两个堆栈实现一个队列？ 要求：只能使用栈的pop(),top()和push()，以及测试栈是否为空empty()四个操作. 来实现队列的empty(), push(),pop(),back(),front()等操作。  2.能否使用两个队列去实现一个堆栈？  请实现他的两个函数Push 和 Pop，分别完成入栈和出栈功能。 | | | | | | | |
| 实验代码：  1，  #include<iostream>  #include<stdio.h>  #include<stdlib.h>  using namespace std;  #define OK 1  #define ERROR 0  #define TRUE 1  #define FALSE 0  typedef int Status;  typedef struct QNode {  int data;  struct QNode \*next;  }QNode, \*QueuePtr;  typedef struct {  QueuePtr front, rear;  } LinkQueue;  int visit(int c) {  cout << c << " ";  return 1;  }  /\* 构造一个空队列Q \*/  Status InitQueue(LinkQueue \*Q) {  Q->front=Q->rear=(QueuePtr)malloc(sizeof(QNode));  if(!Q->front)  ERROR;  Q->front->next=NULL;  return OK;  }  /\* 若Q为空队列,则返回TRUE,否则返回FALSE \*/  int QueueEmpty(LinkQueue Q) {  if(Q.front==Q.rear)  return 1;  else  return 0;  }  /\* 若队列不空,则用e返回Q的队头元素,并返回OK,否则返回ERROR \*/  Status GetHead(LinkQueue Q, int \*e) {  QueuePtr p;  if(Q.front==Q.rear)  return ERROR;  p=Q.front->next;  \*e=p->data;  return OK;  }  /\* 插入元素e为Q的新的队尾元素 \*/  Status EnQueue(LinkQueue \*Q,int e) {  QueuePtr s=(QueuePtr)malloc(sizeof(QNode));  if(!s) /\* 存储分配失败 \*/  return ERROR;  s->data=e;  s->next=NULL;  Q->rear->next=s;  Q->rear=s;  return OK;  }  /\* 若队列不空,删除Q的队头元素,用e返回其值,并返回OK,否则返回ERROR \*/  Status DeQueue(LinkQueue \*Q,int \*e) {  QueuePtr p;  if(Q->front==Q->rear)  return ERROR;  p=Q->front->next;  \*e=p->data;  Q->front->next=p->next;  if(Q->rear==p)  Q->rear=Q->front;  free(p);  return OK;  }  /\* 从队头到队尾依次对队列Q中每个元素输出 \*/  Status QueueTraverse(LinkQueue Q) {  QueuePtr p;  p=Q.front->next;  while(p)  {  visit(p->data);  p=p->next;  }  cout << endl;  return OK;  }  int main() {    int e;  LinkQueue q;  InitQueue(&q);  for (int i = 1; i <= 10; i++) {  EnQueue(&q, i);  }  cout << "（1）输入1-10的数字组成普通线性队列；" << endl;  QueueTraverse(q);      cout << "（2）判断此时队列的队首元素；" << endl;  GetHead(q, &e);  cout << e <<endl;    cout << "（3）打印出队列操作后的队首元素；" << endl;  DeQueue(&q, &e);  GetHead(q, &e);  cout << e <<endl;  return 0;  }  循环队列实现：  #include<stdio.h>  #include<iostream>  using namespace std;  #define MAXSIZE 100  typedef int QElemType;  typedef struct {  QElemType data[MAXSIZE];  int front; //头指针  int rear; //尾指针 若队列不空 指向队尾元素的下一个位置  } SqQueue;    //初始化队列  int InitQueue(SqQueue \*Q) {  Q->front = 0;  Q->rear = 0;  return 1;  }  //返回Q的元素的个数， 也就是队列的当前长度  int QueueLength(SqQueue Q) {  return (Q.rear - Q.front+MAXSIZE) % MAXSIZE;  }  /\* 若队列不空,则用e返回Q的队头元素,并返回OK,否则返回ERROR \*/  int GetHead(SqQueue Q,int \*e)  {  if(Q.front==Q.rear) /\* 队列空 \*/  return 0;  \*e=Q.data[Q.front];  return 1;  }  //循环入队 若队未满 则插入元素e为Q新的队尾元素  int EnQueue(SqQueue \*Q, QElemType e) {  if ((Q->rear + 1) % MAXSIZE == Q->front) //队满的判断  return 0;  Q->data[Q->rear] = e;  Q->rear = (Q->rear + 1) % MAXSIZE; //rear指针向后移动一位 若到最后转到数组的头部  return 1;  }  //若队列不为空 则删除Q中队头元素 用e返回其值  int DeQueue(SqQueue \*Q, QElemType \*e) {  if (Q->front == Q->rear) {  return 0;  }  \*e = Q->data[Q->front];  Q->front = (Q->front + 1) % MAXSIZE;  return 1;  }  /\* 从队头到队尾依次对队列Q中每个元素输出 \*/  int QueueTraverse(SqQueue Q) {  if (Q.front==Q.rear) return 0;  int i;  i=Q.front;  while((i+ MAXSIZE) % MAXSIZE!=Q.rear)  {  printf("%d ",Q.data[i]);  i++;  }  printf("\n");  return 1;  }  int main() {  int e;  SqQueue q;  InitQueue(&q);  for (int i = 1; i <= 10; i++) {  EnQueue(&q, i);  }  cout << "（4）由于普通线性队列的限制，此时尝试使用循环队列来解决上述问题。" << endl;  cout << "（1）输入1-10的数字组成普通线性队列；" << endl;  QueueTraverse(q);    cout << "（2）判断此时队列的队首元素；" << endl;  GetHead(q, &e);  printf("%d \n", e);    cout << "（3）打印出队列操作后的队首元素；" << endl;  DeQueue(&q, &e);  GetHead(q, &e);  printf("%d \n", e);    return 0;  }  思考1和2：  #include<iostream>  using namespace std;  #include<stack>  #include<queue>  #include<algorithm>  //思考1 两栈转队  void test01() {  stack<int> s1;  stack<int> s2;  cout << "两栈转队" <<endl;  cout << "输入 1 2 3 4 5" << endl;  for (int i = 1; i <= 5; i++) {  s1.push(i);  }  for (int i = 1; i <= 5; i++) {  s2.push(s1.top());  s1.pop();  //可以用while和empty()检测是否为空，然后结束s1的数据传到s2  }    cout << "打印数据：" <<endl;  for (int i = 1; i <= 5; i++) {  if (s2.empty()) break;  cout << s2.top() << endl;  s2.pop();  }  }  void test02() {  queue<int> q1;  queue<int> q2;    cout << "输入 1 2 3 4 5" << endl;  for (int i = 1; i <= 5; i++) {  q1.push(i);  }      while(q1.size()) {  if (q1.size() == 1) {  cout << q1.front() << endl;  q1.pop();  while(q2.size()) {  if (q2.size() == 1) {  cout << q2.front() << endl;  q2.pop();  break;  }  q1.push(q2.front());  q2.pop();  }  }  q2.push(q1.front());  q1.pop();  }  //不加这句话，会导致q2中的数据有残余，导致少了一个数据  if (!q2.empty()) {  cout<< q2.front() << endl;  }  }  int main() {  test01();  test02();  } | | | | | | | |
| 实验结果截图及实验心得： | | | | | | | |