**Лабораторная работа № 8. Полустатические структуры данных: очереди**

|  |  |  |  |
| --- | --- | --- | --- |
| 1. В программе, приведенной справа, демонстрируется реализация очереди на основе *односвязного списка.*  Внести изменения в программу с тем, чтобы выводились не только буквы, но и слова. | #include <iostream>  #include <string>  using namespace std;  struct Queue {  string symbol;  Queue\* next;  };  void intoFIFO(Queue\*\* ph, const string& v);  void scan(Queue\* ph);  string fromFIFO(Queue\*\* ph);  int main() {  setlocale(0, "rus");  Queue\* head = nullptr;  Queue\* tail = nullptr;  intoFIFO(&head, "word");  intoFIFO(&head, "b");  intoFIFO(&head, "lala");  scan(head);  intoFIFO(&head, "word2");  intoFIFO(&head, "word3");  scan(head);  string vv = fromFIFO(&head);  cout << "Извлеченное слово: " << vv << endl;  scan(head);  return 0;  }  void intoFIFO(Queue\*\* ph, const string& v) {  Queue\* p = new Queue;  p->symbol = v;  p->next = nullptr;  if (\*ph == nullptr) {  \*ph = p;  }  else {  Queue\* tail = \*ph;  while (tail->next != nullptr) {  tail = tail->next;  }  tail->next = p;  }  }  void scan(Queue\* ph)  {  for (Queue\* p = ph; p != nullptr; p = p->next)  cout << p->symbol << ' ';  cout << endl;  }  string fromFIFO(Queue\*\* ph)  {  if (\*ph == nullptr) return "";  Queue\* q = \*ph;  \*ph = q->next;  string v = q->symbol;  delete q;  return v;  } |  | |
| 2. Изучить способы манипуляции с элементами очереди, реализованной на основе *односвязного списка* выполнив программу, приведенную в правой части. | #include<iostream>  using namespace std;  struct Number  {  int info;  Number\* next;  };  void create(Number\*\* begin, Number\*\* end, int p);  void view(Number\* begin);  Number\* minElem(Number\* begin);  void DeltoMin(Number\*\* begin, Number\*\* p);  int main()  {  Number\* begin = NULL, \* end, \* t;  t = new Number;  int p, size;  cout << "\nEnter size queue="; cin >> size;  cout << "Enter number= "; cin >> p;  t->info = p;  t->next = NULL;  begin = end = t;  for (int i = 1; i < size; i++)  {  cout << "Enter number= "; cin >> p;  create(&begin, &end, p);  }  cout << "\nelements of queue: \n";  if (begin == NULL)  cout << "No elements" << endl;  else  view(begin);  t = minElem(begin);  cout << "minimum=" << t->info << endl;  DeltoMin(&begin, &t);  cout << "\nnew Queue:\n";  view(begin);  return 0;  }  void create(Number\*\* begin, Number\*\* end, int p) {  Number\* t = new Number;  t->next = NULL;  if (\*begin == NULL)  \*begin = \*end = t;  else  {  t->info = p;  (\*end)->next = t;  \*end = t;  }  }  void view(Number\* begin)  {  Number\* t = begin;  if (t == NULL)  {  cout << "Number is empty\n";  return;  }  else  while (t != NULL)  {  cout << t->info << endl;  t = t->next;  }  }  Number\* minElem(Number\* begin)  {  Number\* t = begin, \* mn = nullptr;  int min;  if (t == NULL)  {  cout << "Number is empty\n"; return 0;  }  else  {  min = t->info;  while (t != NULL)  {  if (t->info <= min)  {  min = t->info;  mn = t;  }  t = t->next;  }  }  return mn;  }  void DeltoMin(Number\*\* begin, Number\*\* p) {  Number\* t;  t = new Number;  while (\*begin != \*p)  {  t = \*begin;  \*begin = (\*begin)->next;  delete t;  }  } |  | |
| 3. В правой части приведен *проект*, в котором реализация очереди осуществлена на основе *динамического* *массива элементов, представленных в виде структур*.  Изменить главную функцию, включив операторы работы с функциями добавления, извлечения и вывода различных элементов. | **Программный модуль с главной функцией:**  #include <iostream>  #include "MyQueue.h"  using namespace std;  struct myQue {  int a;  char b;  };  void printQueue(Queue& s) {  Queue tempQueue = createQueue(s.Size);  while (!(s.isEmpty())) {  myQue\* item = (myQue\*)peekQueue(s);  cout << item->a << " " << item->b << endl;  enQueue(tempQueue, item);  delQueue(s);  }  while (!(tempQueue.isEmpty())) {  myQue\* item = (myQue\*)delQueue(tempQueue);  enQueue(s, item);  }  releaseQueue(tempQueue);  }  void addQueueElement(Queue& q, int a, char b) {  myQue\* newElement = new myQue{ a, b };  if (!enQueue(q, newElement)) {  cout << "Очередь заполнена" << endl;  delete newElement;  }  }  void extractQueueElement(Queue& q) {  myQue\* extractedElement = (myQue\*)delQueue(q);  if (extractedElement) {  cout << "Извлеченный элемент: " << extractedElement->a << " " << extractedElement->b << endl;  delete extractedElement;  }  else {  cout << "Очередь пуста" << endl;  }  }  int main() {  setlocale(0, "rus");  Queue q1 = createQueue(4);  addQueueElement(q1, 1, 'q');  addQueueElement(q1, 2, 'w');  addQueueElement(q1, 3, 'e');  addQueueElement(q1, 4, 'r');  cout << "Элементы очереди после добавления:" << endl;  printQueue(q1);  int choice;  do {  cout << "\n1. Извлечь элемент из очереди" << endl;  cout << "2. Показать элементы очереди" << endl;  cout << "0. Выход" << endl;  cout << "Выберите действие: ";  cin >> choice;  switch (choice) {  case 1:  extractQueueElement(q1);  break;  case 2:  cout << "Элементы очереди:" << endl;  printQueue(q1);  break;  case 0:  cout << "Выход..." << endl;  break;  default:  cout << "Неверный выбор" << endl;  }  } while (choice != 0);  releaseQueue(q1);  return 0;  }  **Программный модуль MyQueue.cpp:**  #include "MyQueue.h"  Queue createQueue(int n)  {  return \*(new Queue(n));  };  Queue createQueue(const Queue& pq)  {  Queue\* rc = new Queue(pq.Size - 1);  rc->Head = pq.Head;  rc->Tail = pq.Tail;  for (int i = 0; i < pq.Size; i++)  rc->Data[i] = pq.Data[i];  return \*rc;  }  bool Queue::isFull() const  {  return (Head % Size == (Tail + 1) % Size);  }  bool Queue::isEmpty()const  {  return (Head % Size == Tail % Size);  }  bool enQueue(Queue& q, void\* x)  {  bool rc = true;  if (rc = !q.isFull())  {  q.Data[q.Tail] = x;  q.Tail = (q.Tail + 1) % q.Size;  }  else  rc = false;  return rc;  };  void\* delQueue(Queue& q)  {  void\* rc = (void\*)MYQUEUE1\_EQE;  if (!q.isEmpty())  {  rc = q.Data[q.Head];  q.Head = (q.Head + 1) % q.Size;  }  else  rc = nullptr;  return rc;  }  void\* peekQueue(const Queue& q) {  void\* rc = (void\*)MYQUEUE1\_EQE;  if (!q.isEmpty())  rc = q.Data[q.Head];  else  rc = nullptr;  return rc;  }  int clearQueue(Queue& q)  {  int rc = (q.Tail - q.Head) >= 0 ? q.Tail - q.Head : (q.Size - q.Head + q.Tail + 1);  q.Tail = q.Head = 0;  return rc;  }  void releaseQueue(Queue& q)  {  delete[] q.Data;  q.Size = 1;  q.Head = q.Tail = 0;  }  **Заголовочная функция MyQueue.h:**  #pragma once  #define MYQUEUE1\_EQE 0x0000  struct Queue  {  int Head;  int Tail;  int Size;  void\*\* Data;  Queue(int size)  {  Head = Tail = 0;  Data = new void\* [Size = size + 1];  }  bool isFull() const;  bool isEmpty()const;  };  Queue createQueue(int n);  Queue createQueue(const Queue& pq);  bool enQueue(Queue& q, void\* x);  void\* delQueue(Queue& q);  void\* peekQueue(const Queue& q);  int clearQueue(Queue& q);  void releaseQueue(Queue& q); |  | |
| Вариант 10  Разработать функцию, которая будет находить минимальный элемент очереди, и помещать его в новую очередь **Queue1**, затем будет находить максимальный элемент и помещать его в очередь **Queue2**, затем снова минимальный и так до тех пор, пока из одной очереди не образуется две | Main.cpp  #include "Queue.h"  int main() {  setlocale(LC\_ALL, "rus");  Queue\* myQueue=new Queue;  cout << "Введите количество элементов для ввода в очередь:";  int size;  cin >> size;  for (int i = 0; i < size; i++) {  int buf;  cin >> buf;  push(buf, myQueue);  }  Queue\* q1 = new Queue;  Queue\* q2 = new Queue;  minmax(myQueue, q1, q2);  print(myQueue);  print(q1);  print(q2);  }  Queue.cpp  #include "Queue.h"  using namespace std;  void clear(Queue\* q) {  q->first = q->mas;  q->last = q->mas;  }  void fixQueue(Queue\* q) {  A\* newfirst = q->mas;  A\* newlast = q->mas;  while (q->first != q->last) {  \*newlast++ = \*q->first++;  }  q->first = newfirst;  q->last = newlast;  }  bool isEmpty(Queue\* q) {  return q->first == q->last;  }  bool isFull(Queue\* q) {  return q->first == q->mas && q->last == (q->mas + MAXSIZE - 1);  }  A top(Queue\* q) {  if (!isEmpty(q)) {  return \*q->first;  }  else {  cout << "Очередь пуста\n";  return -1;  }  }  A pop(Queue\* q) {  if (!isEmpty(q)) {  return \*q->first++;  }  else {  cout << "Очередь пуста\n";  return -1;  }  }  bool push(A element, Queue\* q) {  if (isFull(q)) {  cout << "Очередь заполнена\n";  return false;  }  if (q->last == (q->mas + MAXSIZE)) {  fixQueue(q);  }  \*q->last++ = element;  return true;  }  void print(Queue\* q) {  Queue\* temp = new Queue;  while (!isEmpty(q)) {  A buf = pop(q);  cout << buf << ' ';  push(buf, temp);  }  cout << '\n';  while (!isEmpty(temp)) {  push(pop(temp), q);  }  delete temp;  return;  }  void printToFile(Queue\* q) {    }  void copyQueue(Queue\* q1, Queue\* q2) {  Queue\* temp = new Queue;  while (!isEmpty(q1)) {  A buf = pop(q1);  push(buf, temp);  push(buf, q2);  }  while (!isEmpty(temp)) {  push(pop(temp), q1);  }  }  int curSize(Queue\* q) {  return q->last - q->first;  }  void minmax(Queue\* q, Queue\* q1, Queue\* q2) {  Queue\* temp = new Queue;  copyQueue(q, temp);  Queue\* buf = new Queue;  int i = 0;  while (!isEmpty(temp)) {  A minmax = pop(temp);  push(minmax, buf);  while (!isEmpty(temp)) {  A el = pop(temp);  push(el, buf);  if ((i % 2 == 0 ? minmax<el : minmax>el)) {  minmax = el;  }  }  while (!isEmpty(buf)) {  A el = pop(buf);  if (el == minmax) {  push(el, (i % 2 == 0 ? q1 : q2));  }  else {  push(el, temp);  }  }  i++;  }  delete buf;  delete temp;  }  void moveMax(Queue\* q) {  Queue\* temp = new Queue;  A max = pop(q);  push(max, temp);  while (!isEmpty(q)) {  A buf = pop(q);  if (max < buf) {  max = buf;  }  push(buf, temp);  }  while (!isEmpty(temp)) {  push(pop(temp), q);  }  A buf = top(q);  while (buf != max) {  buf = pop(q);  push(buf, q);  buf = top(q);  }  return;  }  Queue.h  #pragma once  #include <iostream>  #include <fstream>  #define MAXSIZE 1000  #define A float  using namespace std;  struct Queue {  A mas[MAXSIZE];  A\* first = mas;  A\* last = mas;  };  int curSize(Queue\* q);  void clear(Queue\* q);  bool isEmpty(Queue\* q);  bool isFull(Queue\* q);  A top(Queue\* q);  A pop(Queue\* q);  bool push(A element, Queue\* q);  void print(Queue\* q);  void minmax(Queue\* q, Queue\* q1, Queue\* q2);  void moveMax(Queue\* q); | |  |
| 2. Создать очередь с вещественными числами, и заполнить ее с клавиатуры. Выполнить циклический сдвиг элементов в очереди так, чтобы в ее начале был расположен наибольший элемент. | #include "Queue.h"  using namespace std;  int main() {  Queue\* myQueue = new Queue;  cout << "Введите количество элементов для ввода в очередь:";  int size;  cin >> size;  for (int i = 0; i < size; i++) {  A buf;  cin >> buf;  push(buf, myQueue);  }  moveMax(myQueue);  print(myQueue);  return 0;  } | | |
| 3. Содержимое текстового файла **f**, разделенное на строки, переписать в текстовый файл **g**, перенося при этом в конец каждой строки все входящие в нее цифры (с сохранением исходного взаимного порядка, как среди цифр, так и среди остальных литер строки). Использовать очереди. | #include "/Уник/2sem/OAP\_2sem/Lab 8/Lab 8/Queue.h"  using namespace std;  int main() {  ifstream fin("f.txt");  ofstream fout("g.txt");  char buf[256];  while (fin.getline(buf, 256)) {  Queue\* q=new Queue;  int start = 0, end;  int i = 0;  while (buf[i-1] != '\0') {  if (buf[i] == ' '||buf[i]=='\0') {  char c[256];  end = i;  bool flag = true;  for (int j = start; j < end; j++) {  c[j - start] = buf[j];  if (!isdigit(buf[j])) {  flag = false;  }  }  c[end-start] = '\0';  if (flag) {  push(atoi(c), q);  }  else {  fout << c<<' ';  }  start = i + 1;  }  i++;  }  Queue\* temp = new Queue;  while (!isEmpty(q)) {  A buf = pop(q);  fout << buf << ' ';  push(buf, temp);  }  fout << '\n';  while (!isEmpty(temp)) {  push(pop(temp), q);  }  delete temp;  }  return 0;  } | | |

|  |  |
| --- | --- |
| **Создание очереди с приоритетным включением**  1. Разработать функции работы с приоритетной очередью. Постановка запросов в очередь выполняется по приоритету, снятие − подряд из младших адресов (начало очереди). Приоритет: минимальное значение числового параметра, при совпадении параметров − **LIFO**. | #include<iostream>  using namespace std;  struct Item  {  int data;  Item\* next;  };  Item\* head, \* tail;  bool isNull(void)  {  return (head == NULL);  }  void deletFirst()  {  if (isNull())  cout << "Очередь пуста" << endl;  else  {  Item\* p = head;  head = head->next;  delete p;  }  }  void getFromHead()  {  if (isNull())  cout << "Очередь пуста" << endl;  else  cout << "Начало = " << head->data << endl;  }  void insertToQueue(int x) {  Item\* p = new Item;  p->data = x;  p->next = nullptr;  if (isNull()) {  head = tail = p;  }  else {  Item\* prev = nullptr;  Item\* current = head;  while (current != nullptr && x > current->data) {  prev = current;  current = current->next;  }  if (prev == nullptr) {  // более высокмй приоритет  p->next = head;  head = p;  }  else {  // менее высокоий приоритет  if (current == nullptr) { //  tail->next = p;  tail = p;  }  else { // если в середине  p->next = current;  prev->next = p;  }  }  }  }  void printQueue()  {  Item\* p = new Item;  if (isNull())  cout << "Очередь пуста" << endl;  else  {  cout << "Очередь = ";  p = head;  while (!isNull())  {  if (p != NULL)  {  cout << p->data << " "; cout << "->";  p = p->next;  }  else  {  cout << "NULL" << endl;  return;  }  }  }  }  void clrQueue()  {  while (!isNull()) deletFirst();  }  int main()  {  setlocale(LC\_CTYPE, "Russian");  int i = 1, choice = 1, z; head = NULL; tail = NULL;  while (choice != 0)  {  cout << "1 - добавить элемент" << endl;  cout << "2 - получить элемент с начала" << endl;  cout << "3 - извлечь элемент с начала" << endl;  cout << "4 - вывести элементы" << endl;  cout << "5 - очистить очередь" << endl;  cout << "0 - выход" << endl;  cout << "Выберите действие "; cin >> choice;  switch (choice)  {  case 1: cout << "Введите элемент: "; cin >> z;  insertToQueue(z); printQueue(); break;  case 2: getFromHead(); break;  case 3: deletFirst(); break;  case 4: printQueue(); break;  case 5: clrQueue(); break;  }  }  return 0;  } |
| 2. Разработать функции работы с приоритетной очередью. Постановка запросов в очередь выполняется по приоритету, снятие − подряд из начала очереди. Приоритет: иаксимальное значение числового параметра, при совпадении параметров − **FIFO**. | void insertToQueue(int x) {  Item\* p = new Item;  p->data = x;  p->next = nullptr;  if (isNull()) {  head = tail = p;  }  else {  Item\* prev = nullptr;  Item\* current = head;  while (current != nullptr && x <= current->data) {  prev = current;  current = current->next;  }  if (prev == nullptr) {    p->next = head;  head = p;  }  else {    if (current == nullptr) {  tail->next = p;  tail = p;  }  else {  p->next = current;  prev->next = p;  }  }  }  } |
| 3. Разработать функции работы с приоритетной очередью. Постановка запросов в очередь выполняется по приоритету, снятие − подряд из старших адресов (конец очереди). Приоритет: минимальное значение числового параметра, при совпадении параметров − **LIFO**.  4. Разработать функции работы с приоритетной очередью. Постановка запросов в очередь выполняется по приоритету, снятие − подряд из старших адресов (конец очереди). Приоритет: минимальное значение числового параметра, при совпадении параметров − **LIFO**. | void deletFirst()  {  if (isNull())  cout << "Очередь пуста" << endl;  else  {  Item\* p = nullptr;  Item\* current = head;  while (!isNull())  {  if (current->next == tail) {  current->next = nullptr;  p = tail;  tail = current;  break;  }  current = current->next;  }  delete p;  }  }  void getFromHead()  {  if (isNull())  cout << "Очередь пуста" << endl;  else  cout << "Конец = " << tail->data << endl;  }  void insertToQueue(int x) {  Item\* p = new Item;  p->data = x;  p->next = nullptr;  if (isNull()) {  head = tail = p;  }  else {  Item\* prev = nullptr;  Item\* current = head;  while (current != nullptr && x > current->data) {  prev = current;  current = current->next;  }  if (prev == nullptr) {    p->next = head;  head = p;  }  else {    if (current == nullptr) { tail->next = p;  tail = p;  }  else {  p->next = current;  prev->next = p;  }  }  }  } |
| 5. Разработать функции работы с приоритетной очередью. Постановка запросов в очередь выполняется по приоритету, снятие − подряд из старших адресов (конец очереди). Приоритет: максимальное значение числового параметра, при совпадении параметров − **FIFO**. | void deletFirst()  {  if (isNull())  cout << "Очередь пуста" << endl;  else  {  Item\* p = nullptr;  Item\* current = head;  while (!isNull())  {    if (current->next == tail) {  current->next = nullptr;  p = tail;  tail = current;  break;  }  current = current->next;  }  delete p;  }  }  void getFromHead()  {  if (isNull())  cout << "Очередь пуста" << endl;  else  cout << "Конец = " << tail->data << endl;  }  void insertToQueue(int x) {  Item\* p = new Item;  p->data = x;  p->next = nullptr;  if (isNull()) {  head = tail = p;  }  else {  Item\* prev = nullptr;  Item\* current = head;  while (current != nullptr && x <= current->data) {  prev = current;  current = current->next;  }  if (prev == nullptr) {    p->next = head;  head = p;  }  else {    if (current == nullptr) { tail->next = p;  tail = p;  }  else {  p->next = current;  prev->next = p;  }  }  }  } |