Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

Лабораторная работа 8

По дисциплине «Основы алгоритмизации и программирования»

На тему «Полустатические структуры данных: очереди»

Выполнила:

Студентка 1 курса 2 группы

Глухова Д.В.

Преподаватель: асс. Андронова М.В.

2023, Минск

**Вариант 2.**

4. Создать проект из нескольких файлов, демонстрирующий работу с очередью. В соответствии со своим вариантом выполнить задание из таблицы, представленной ниже.

Разработать меню и реализовать все операции с очередью через функции. Максимальный размер очереди ввести с клавиатуры.

Создать очередь для целых чисел и функции для ввода, вывода и удаления элементов очереди. Найти количество элементов между максимальным и минимальным элементами очереди

|  |
| --- |
| Программный код |
| Программный модуль с главной функцией |
| // Создать очередь для целых чисел и функции для ввода, вывода и удаления элементов очереди. Найти количество элементов между максимальным и минимальным элементами очереди.  #include<iostream>  #include <windows.h>  #include "head.h"  void main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  setlocale(LC\_ALL, "ru");  Number\* begin = NULL, \* end, \* t;  t = new Number;  short size, choose = NULL, p, sizeQueue;  while (choose != 9)  {  cout << "Выберите, что сделать:\n"  << "1 — ввод очереди\n"  << "2 — вывод очереди\n"  << "3 — очистка очереди\n"  << "4 — определение размера очереди\n"  << "5 — найти кол-во между max и min элементами\n"  << "9 — выход из программы.\n";  cin >> choose;  cout << '\n';  switch (choose)  {  case 1:  cout << "Введите размер очереди:\n";  cin >> size;  cout << "Введите число: ";  cin >> p;  create(&begin, &end, p);  t->info = p;  t->next = NULL;  begin = end = t;  for (int i = 1; i < size; i++) //создание очереди  {  cout << "Введите число: ";  cin >> p;  create(&begin, &end, p);  }  cout << '\n';  break;  case 2:  view(begin);  break;  case 3:  del(begin);  break;  case 4:  sizeQueue = sizeQ(begin);  break;  case 5:  task(begin);  break;  case 9:  exit(0);  }  }  } |
| Модуль с функциями |
| #include "head.h"  // создание очереди  void create(Number\*\* begin, Number\*\* end, int p) {  Number\* t = new Number; // выделение памяти  t->next = NULL;  if (\*begin == NULL)  \*begin = \*end = t;  else  {  t->info = p; // заполнение поля  (\*end)->next = t; // дальше  \*end = t;  }  }  // для вывода очереди  void view(Number\* begin)  {  Number\* t = begin; // хранение дреса элемента  if (t == NULL)  {  cout << "пусто\n";  }  else {  cout << "очередь:\n";  while (t != NULL) // пока очередь не пуста  {  cout << t->info << endl; // выводим элемент  t = t->next; // и на следующий  }  cout << '\n';  }  }  // для удаления очереди  void del(Number\* begin)  {  Number\* t = begin; // хранение адреса элемента  if (t == NULL)  {  cout << "пусто\n";  }  else  {  while (t != NULL)  {  t->info = NULL; // устанавливаем в ноль  t = t->next; // и на следующий  }  }  cout << "элементы удалены\n\n";  }  // узнать размер очереди  short sizeQ(Number\* begin)  {  Number\* t = begin; // хранение адреса элемента  short counter = 0, data = t->info;  while (t != NULL)  {  if (data != NULL)  {  counter++; // счетчик увеличивается  t = t->next; // перемещаемся дальше  }  else t = t->next;  }  cout << "размер очереди = " << counter << "\n\n";  return counter;  }  // задание из варианта  Number\* task(Number\* begin)  {  Number\* t = begin;  int max = SHRT\_MIN, min = SHRT\_MAX;  short couter = 1, data = t->info, pos1 = 0, pos2 = 0;  while (t != NULL)  {  if (data != NULL)  {  if (t->info >= max)  {  max = t->info;  pos1 = couter;  }  if (t->info <= min)  {  min = t->info;  pos2 = couter;  }  couter++; // счетчик увеличивается  t = t->next; // перемещаемся дальше  }  else t = t->next;  }  cout << "max элемент: " << max << "\n";  cout << "min элемент: " << min << "\n";  cout << "Позиция max: " << pos1 << "\n";  cout << "Позиция min: " << pos2 << "\n";  cout << "Количество элементов между max и min: " << abs(pos1 - pos2) - 1 << "\n\n";  return (t);  } |
| заголовочный файл **head.h**, который содержит описание структуры стека и прототипы программ обработки его элементов.  #pragma once  #include<iostream>  using namespace std;  struct Number  {  int info;  Number\* next;  };  void create(Number\*\* begin, Number\*\* end, int p);  void view(Number\* begin);  void del(Number\* begin);  short sizeQ(Number\* begin);  Number\* task(Number\* begin); |
| Консоль отладки |
|  |

Дополнительные задания.

1. В [приложении 1](#prilogenie1) приведен проект, в котором реализована очередь на основе односвязного *списка* с *приоритетным* включением. На основе данного проекта разработать функции, которые предлагается создать в данном приложении.

|  |
| --- |
| Программный код |
| // 1. В приложении 1 приведен проект, в котором реализована очередь на основе односвязного списка с приоритет-ным включением.  // На основе данного проекта разработать функции, которые предлагается создать в данном приложе-нии.  #include <iostream>  using namespace std;  struct Item  {  int data;  Item\* next;  };  Item\* head, \* tail;  bool isNull(void) //Проверка на пустоту  {  return (head == nullptr);  }  void deletFirst() //Извлечение элемента из начала  {  if (isNull())  cout << "Очередь пуста" << endl;  else  {  Item\* p = head; // адрес элемента  head = head->next; // перемещаемся дальше  delete p; // удаляем  }  }  void deletlast() //Извлечение элемента из начала  {  if (isNull())  cout << "Очередь пуста" << endl;  else  {  Item\* p;  tail->data = NULL;  }  }  void getFromHead() // элемент из начала  {  if (isNull())  cout << "Очередь пуста" << endl;  else  cout << "Начало = " << head->data << endl; // выводим элемент в начале  }  void getFromTail() // элемент из конца  {  if (isNull())  cout << "Очередь пуста" << endl;  else  cout << "Конец = " << tail->data << endl; // выводим элемент в конце  }  void insertToQueue(int x) //Добавление элемента в очередь  {  Item\* p = new Item; //новый указатель  p->data = x; // заполняем поле переменной x  p->next = nullptr;  Item\* v = new Item; //указатель для нового числа  Item\* p1 = new Item;  Item\* p2 = new Item;  int i = 0; //флажок  if (isNull())  head = tail = p;  else  {  p2 = head; p1 = head; // !!!Чтобы реализовать ввод LIFO необходимо чтобы указатель p2 был сзади, а FIFO наоборот  while (p1 != nullptr) //пока очередь не закончится  {  if (i == 1) //ТАКЖЕ НЕОБХОДИМЫ ДЛЯ РЕАЛИЗАЦИИ LIFO  {  if (i == 1)  {  if (x > p1->data) //число больше, чем в очереди  {  v->data = x;  v->next = p1;  p2->next = v;  return;  }  if (x == p1->data)  {  v->data = x;  v->next = p1->next;  p1->next = v;  return;  }  p2 = p2->next;  }  if (x > p1->data)  {  v->data = x;  v->next = p1;  head = v;  return;  }  if (p1->next == nullptr)  {  v->data = x;  v->next = nullptr;  tail->next = v;  tail = v;  return;  }  i = 1;  p1 = p1->next; // следующее число  }  else // else необходимое для реализации вставки LiFo  {  if (x >= p1->data)  {  v->data = x;  v->next = p2;  head = v;  return;  }  }  p1 = p1->next; //ТАКЖЕ НЕОБХОДИМЫ ДЛЯ РЕАЛИЗАЦИИ LiFo  i = 1;  }  if (p1 == NULL) // Выражение ДЛЯ РЕАЛИЗЦИИ LiFo  {  v->data = x;  v->next = tail;  p2->next = v;  }  }  }  void printQueue() //Вывод очереди  {  int g;  Item\* p = new Item;  if (isNull())  cout << "Очередь пуста" << endl;  else  {  cout << "Очередь = ";  p = head;  while (!isNull())  {  if (p != nullptr)  {  g = p->data;  if (g == 0) {  }  else { cout << p->data << " "; cout << "-> "; }  p = p->next;  }  else  {  cout << "NULL" << endl;  return;  }  }  }  }  void clrQueue() //Очистка очереди  {  while (!isNull()) deletFirst();  }  int main()  {  setlocale(LC\_CTYPE, "Russian");  int i = 1, choice = 1, z; head = nullptr; tail = nullptr;  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 << "6 - получить элемент с конца" << endl;  cout << "0 - выход" << endl;  cout << "Выберите действие:\n"; 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;  case 6: getFromTail(); break;  case 7: deletlast(); break;  }  }  return 0;  } |
| Консоль отладки |
|  |

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.

|  |
| --- |
| Программный код |
| #include <iostream>  #include <windows.h>  #include <fstream>  #include <string>  using namespace std;  struct Stack  {  int data; // целочисленное информационное поле  Stack\* head;  Stack\* next;  };  void dop2(Stack\* MyStk);  int main() {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  setlocale(LC\_ALL, ".1251");  Stack\* MyStk = new Stack; //выделение памяти для стека  MyStk->head = NULL;  dop2(MyStk);  }  void dop2(Stack\* MyStk)  {  ifstream file("text.txt");  string str;  short counter = 1, seqNum = 1, min = SHRT\_MAX, minData;  int\* Array = new int[10];  while (getline(file, str))  {  Stack\* e = new Stack; // выделение памяти  e->data = str.length(); // заполняем целочисленное поле  if (e->data <= min) // поиск мин строки  {  min = e->data;  minData = min; // для мин длины  seqNum = counter; // для номера строки  }  e->next = MyStk->head;  MyStk->head = e;  counter++;  }  cout << "Длины строк: \n"; // с конца файла, тк LIFO  while (MyStk->head)  {  cout << MyStk->head->data << "\t";  MyStk->head = MyStk->head->next;  }  cout << "\nCамая короткая строка номер " << seqNum << " длина " << minData << ".\n";  } |
| Консоль отладки |
|  |

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.

|  |
| --- |
| Программный код |
| #include <iostream>  #include <fstream>  using namespace std;  struct Queue  {  char data[255]; //массив данных  int head; //указатель на начало  };  void creatQueue(Queue\* Q) //Создание очереди Q  {  Q->head = NULL;  }  bool isFull(Queue\* Q) //Проверка очереди на пустоту  {  if (!Q->head)  return true;  else  return false;  }  void showQueue(Queue\* Q) {  ofstream fout("g.txt", ios\_base::app);  for (int i = 0; i < Q->head; i++)  {  fout << Q->data[i];  }  Q->head = 0;  fout.close();  }  void addToQueue(Queue\* Q, char value) //Добавление элемента  {  Q->data[Q->head++] = value;  }  ifstream fin("f.txt");  int main()  {  ofstream fout("g.txt", ios\_base::trunc);  fout.close();  setlocale(LC\_ALL, "Rus");  Queue Q1, Q2;  creatQueue(&Q1);  creatQueue(&Q2);  while (!fin.eof()) {  char s[255] = "";  fin.getline(s, 253);  for (int i = 0; i < strlen(s); i++)  {  if ((s[i] != '0') && (s[i] != '1') && (s[i] != '2') && (s[i] != '3') && (s[i] != '4') && (s[i] != '5') && (s[i] != '6') && (s[i] != '7') && (s[i] != '8') && (s[i] != '9'))  {  addToQueue(&Q1, s[i]);  }  else  addToQueue(&Q2, s[i]);  }  showQueue(&Q1);  showQueue(&Q2);  ofstream fout("g.txt", ios\_base::app);  fout << endl;  fout.close();  }  cout << "Информация записана в файл.\n";  system("pause");  return 0;  } |
| Консоль отладки |
|  |
| Исходный файл |
|  |
| Конечный файл |
|  |