Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа 4

По дисциплине «Основы алгоритмизации программирования»

На тему «Полустатические структуры данных: очереди»

Выполнил:

Студент 1 курса 6 группы

Кучерук Николай Петрович

Преподаватель: асс. Андронова М.В.

2023, Минск

**Задание**

| **№ варианта** | **Условие задачи** |
| --- | --- |
| **9** | Создать очередь для целых чисел и функции для ввода, вывода, удаления и определения размера очереди. Разработать функцию, которая по одной очереди строит две новых: **Queue1** из четных элементов и **Queue2** − из остальных элементов очереди. |

|  |
| --- |
| **Код программы** |
| #include <iostream>  #include <fstream>  using namespace std;  struct node//элемент очереди на основе односвязного списка  {  int data;//информация, хранящаяся в элементе  node\* next;//указатель на следующий элемент  node(int data) {//конструктор элемента  this->data = data;  this->next = NULL;  }  };  struct queue//структура очереди на основе односвязного списка  {  node\* head, \* tail;//указатели на первый и последний элемент списка  queue() {//конструктор очерди  this->head = this->tail = NULL;  }  ~queue() {//т.к очередь является динамичной структурой, то для предотвращения утечки памяти создаем деструктор  while (head != NULL) pop\_back();  }  void pop\_back() {//метод для удаления элемента в конце очерди  if (head == NULL) return;  if (head == tail) {  delete tail;  head = tail = NULL;  return;  }  node\* Node = head;  while (Node != NULL) {  if (Node->next == tail)  {  tail = Node;  delete Node->next;  Node->next = NULL;  return;  }  Node = Node->next;  }  }  void push\_front(int data) {//метод для добавления элемента в начало очереди  node\* Node = new node(data);  Node->next = head;  head = Node;  if (tail == NULL)tail = Node;  }  void output() {//метод для вывода очереди  for (node\* Node = head; Node != NULL; Node = Node->next) {  cout << Node->data << " ";  }  cout << endl;  }  short size() {//метод для определения размера очереди  short counter = 0;  for (node\* Node = head; Node != NULL; Node = Node->next) {  counter++;  }  return counter;  }    };  void menu(void); //функция, показывающая меню  void solution(queue base,queue ,queue );  int main()  {  setlocale(LC\_CTYPE, "Russian");  queue first,queue1,queue2;//создаем очереди  int choice;  int value;  menu(); // вывести меню  cout << " ? ";  cin >> choice;  while (choice != 4)  {  switch (choice)  {  case 1: cout << "Введите число " << endl; // добавить число в начало очереди  cin >> value;  first.push\_front(value);  cout << "Очередь: ";  first.output();  break;  case 2:  {  first.pop\_back();  cout << "Очередь: ";  first.output();  }  break;  case 3: solution(first,queue1,queue2); // вычисление суммы  break;  default: cout << "Неправильный выбор" << endl;  menu();  break;  }  cout << "? ";  cin >> choice;  }  cout << "Конец" << endl;  return 0;  }  void menu(void) //Вывод меню  {  cout << "Сделайте выбор:" << endl;  cout << " 1 - Ввод числа" << endl;  cout << " 2 - Удаление числа" << endl;  cout << " 3 - Разделение четных и нечетных элементов" << endl;  cout << " 4 - Выход" << endl;  }  void solution(queue base, queue first, queue second) {//функция,для перераспределения элементов в новые очереди  for (node\* Node = base.head; Node != NULL; Node = Node->next) {  if (Node->data % 2 == 0) { first.push\_front(Node->data); }  else{ second.push\_front(Node->data); }  }  cout << endl << "Первый полученный список: " << endl;  first.output();  cout << endl << "Второй полученный список: " << endl;  second.output();  } |
| **Результат программы** |
|  |

Дополнительные задания

1. В приложении 1 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.

|  |
| --- |
| **Код программы** |
| #include<iostream>  using namespace std;  // Структура элемента очереди  struct Item {  int data;  Item\* next;  };  // Глобальные указатели на начало и конец очереди  Item\* head = nullptr;  Item\* tail = nullptr;  // Функция проверки на пустоту очереди  bool isNull() {  return (head == nullptr);  }  // Функция удаления первого элемента из очереди  void deletFirst() {  if (isNull())  cout << "Очередь пуста" << endl;  else {  Item\* p = head;  head = head->next;  delete p;  }  }  // Функция получения первого элемента из очереди  void getFromHead() {  if (isNull())  cout << "Очередь пуста" << endl;  else  cout << "Начало = " << head->data << endl;  }  // Функция добавления элемента в приоритетную очередь с минимальным приоритетом и LIFO  void insertMinPriorityLIFO(int x) {  Item\* p = new Item;  p->data = x;  p->next = nullptr;  if (isNull()) {  head = tail = p;  }  else {  Item\* prev = nullptr;  Item\* curr = head;  while (curr != nullptr && curr->data <= x) {  prev = curr;  curr = curr->next;  }  if (prev == nullptr) {  p->next = head;  head = p;  }  else {  prev->next = p;  p->next = curr;  }  }  }  // Функция добавления элемента в приоритетную очередь с максимальным приоритетом и FIFO  void insertMaxPriorityFIFO(int x) {  Item\* p = new Item;  p->data = x;  p->next = nullptr;  if (isNull()) {  head = tail = p;  }  else {  Item\* prev = nullptr;  Item\* curr = head;  while (curr != nullptr && curr->data >= x) {  prev = curr;  curr = curr->next;  }  if (prev == nullptr) {  p->next = head;  head = p;  }  else {  prev->next = p;  p->next = curr;  }  }  }  // Функция удаления последнего элемента из приоритетной очереди с минимальным приоритетом и LIFO  void deletLastMinPriorityLIFO() {  if (isNull()) {  cout << "Очередь пуста" << endl;  }  else {  Item\* prev = nullptr;  Item\* curr = head;  while (curr->next != nullptr) {  prev = curr;  curr = curr->next;  }  if (prev == nullptr) {  // В очереди только один элемент  delete curr;  head = tail = nullptr;  }  else {  delete tail;  tail = prev;  tail->next = nullptr;  }  }  }  // Функция удаления последнего элемента из приоритетной очереди с минимальным приоритетом и FIFO  void deletLastMinPriorityFIFO() {  deletLastMinPriorityLIFO();  }  // Функция добавления элемента в приоритетную очередь с максимальным приоритетом и FIFO  void insertMaxPriorityFIFOTiebreaker(int x) {  insertMaxPriorityFIFO(x);  }  // Функция вывода элементов очереди  void printQueue() {  if (isNull())  cout << "Очередь пуста" << endl;  else {  cout << "Очередь = ";  Item\* p = head;  while (p != nullptr) {  cout << p->data << " -> ";  p = p->next;  }  cout << "NULL" << endl;  }  }  // Функция очистки очереди  void clrQueue() {  while (!isNull())  deletFirst();  }  // Основная функция  int main() {  setlocale(LC\_CTYPE, "Russian");  int choice = 1, z;  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;  insertMinPriorityLIFO(z);  printQueue();  break;  case 2:  getFromHead();  break;  case 3:  deletFirst();  break;  case 4:  printQueue();  break;  case 5:  clrQueue();  break;  }  }  return 0;  } |
| **Результат программы** |
|  |

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.

|  |
| --- |
| **Код программы** |
| #include <iostream>  using namespace std;  const int MAX\_SIZE = 100;  // Функция для выполнения циклического сдвига массива так, чтобы в его начале был наибольший элемент  void cyclicShift(double arr[], int size) {  double maxElement = arr[0];  int maxIndex = 0;  // Находим индекс наибольшего элемента  for (int i = 1; i < size; ++i) {  if (arr[i] > maxElement) {  maxElement = arr[i];  maxIndex = i;  }  }  // Выполняем циклический сдвиг  for (int i = 0; i < maxIndex; ++i) {  double temp = arr[0];  for (int j = 1; j < size; ++j) {  arr[j - 1] = arr[j];  }  arr[size - 1] = temp;  }  }  int main() {  setlocale(LC\_ALL, "RUS");  double arr[MAX\_SIZE];  int n;  cout << "Введите количество элементов в очереди: ";  cin >> n;  if (n <= 0 || n > MAX\_SIZE) {  cout << "Недопустимое количество элементов!" << endl;  return 1;  }  // Заполнение массива с клавиатуры  cout << "Введите " << n << " вещественных чисел: ";  for (int i = 0; i < n; ++i) {  cin >> arr[i];  }  cyclicShift(arr, n);  // Выводим результат циклического сдвига  cout << "Результат циклического сдвига: ";  for (int i = 0; i < n; ++i) {  cout << arr[i] << " ";  }  cout << endl;  return 0;  } |
| **Результат программы** |
|  |

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.

|  |  |
| --- | --- |
| **Код программы** |  |
| #include <iostream>  #include <fstream>  #include <string>  using namespace std;  // Функция для переноса цифр в конец строки  string moveDigitsToEnd(const string& line) {  string result;  string digits;  for (char ch : line) {  if (isdigit(ch)) {  digits += ch;  }  else {  result += ch;  }  }  result += digits;  return result;  }  int main() {  setlocale(LC\_ALL, "RUS");  ifstream inFile("f.txt");  ofstream outFile("g.txt");  if (!inFile) {  cout << "Ошибка открытия файла f.txt" << endl;  return 1;  }  if (!outFile) {  cout << "Ошибка открытия/создания файла g.txt" << endl;  return 1;  }  string line;  while (getline(inFile, line)) {  string modifiedLine = moveDigitsToEnd(line);  outFile << modifiedLine << endl;  }  cout << "Процесс завершен. Результат записан в файл g.txt" << endl;  inFile.close();  outFile.close();  return 0;  } |  |
| **Результат программы** |  |
|  |  |