# Лабораторная работа №4

Выполнил: Лосев Данил

**Задания:**

1. **Используя алгоритмы из лекции 4, написать программные коды:**

* функции построения очереди
* вставки звена в конец очереди
* удаления звена из начала очереди

1. **Самостоятельно написать коды:**

* отображение на экране очереди,
* поиска звена по значению с выдачей адреса на консоль,
* подсчет количества элементов в очереди.

1. **Разобрать самостоятельно работу кода очистки памяти от очереди, то есть проанализировать письменно каждую команду кода:**

//Очистка памяти от очереди

void OCHISTKA()

{

   r=no;

  if (no!=NULL)

  {

    while (no!=ko)

    {

      no=(\*r).sled;

      delete r;  r=no; }

    delete no;

    no=ko=NULL;

  }

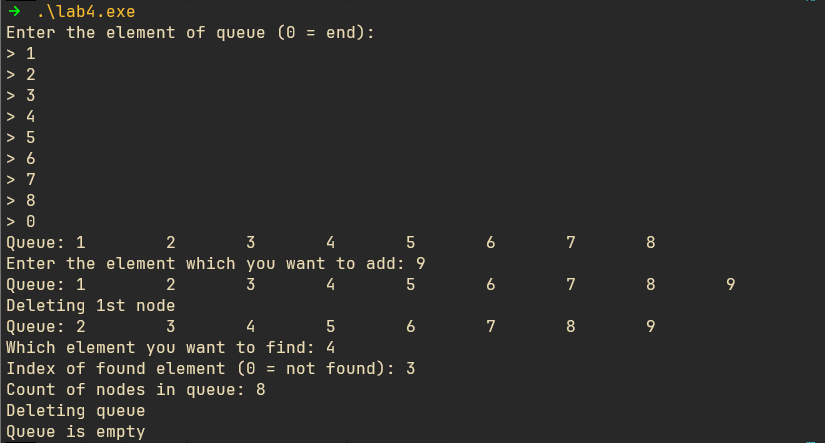
  cout<<"Память очищена: "<<endl;

  system("pause");

}

1. **Объединить все эти функции в один листинг, используя меню, отладить в любой среде программирования на языке С++.**
2. **Предоставить отчет в электронном виде преподавателю для защиты.**

|  |
| --- |
| #include <cstddef>  #include <iostream>  // Это структура узла, здесь я храню данные и указатель на следующий узел.  struct Node  {  int elem; // Здесь храню сам элемент (значение).  Node \*next; // Указатель на следующий узел в очереди.  };  // Структура очереди, здесь у меня два указателя — на начало и конец очереди.  struct Queue  {  Node \*start; // Указатель на первый узел очереди.  Node \*end; // Указатель на последний узел очереди.  };  // Объявляю функции для работы с очередью  void buildingQueue(Node \*&fStart, Node \*&fEnd); // Эта функция строит очередь  void addNode(Node \*&fEnd, int fElement); // Добавляю узел в конец очереди  int deleteNode(Node \*&fStart); // Удаляю узел из начала очереди  void printQueue(Node \*fStart); // Печатаю всю очередь  int findNode(Node \*fStart, int fElement); // Ищу узел с нужным элементом  int countOfNodes(Node \*fStart); // Считаю количество узлов в очереди  void deletingQueue(Node \*&fStart, Node \*&fEnd); // Удаляю всю очередь  int main()  {  Queue queue; // Создаю очередь  int element;  buildingQueue(queue.start, queue.end); // Строю очередь через пользовательский ввод  printQueue(queue.start); // Печатаю очередь, чтобы убедиться, что она правильно построена  std::cout << "Enter the element which you want to add: ";  std::cin >> element;  addNode(queue.end, element); // Добавляю новый элемент в конец  printQueue(queue.start); // Печатаю очередь после добавления  std::cout << "Deleting 1st node " << std::endl;  element = deleteNode(queue.start); // Удаляю первый узел  printQueue(queue.start); // Печатаю очередь после удаления  std::cout << "Which element you want to find: ";  std::cin >> element;  int index = findNode(queue.start, element); // Ищу элемент, который ввел пользователь  std::cout << "Index of found element (0 = not found): " << index << std::endl;  std::cout << "Count of nodes in queue: " << countOfNodes(queue.start)  << std::endl; // Показываю количество узлов в очереди  std::cout << "Deleting queue" << std::endl;  deletingQueue(queue.start, queue.end); // Полностью очищаю очередь  printQueue(queue.start); // Печатаю пустую очередь для подтверждения  return 0;  }  // Строю очередь через пользовательский ввод  void buildingQueue(Node \*&fStart, Node \*&fEnd)  {  Node \*fQueue = new (Node); // Создаю новый узел для начала очереди  int element = 0;  std::cout << "Enter the element of queue (0 = end):" << '\n' << "> ";  std::cin >> element;  (\*fQueue).elem = element; // Сохраняю первый элемент  (\*fQueue).next = NULL; // Пока других узлов нет, следующий — NULL  fStart = fQueue; // Начало очереди — это мой первый узел  fEnd = fQueue; // Конец очереди пока тоже этот узел  std::cout << "> ";  std::cin >> element;  while (element != 0) // Продолжаю строить очередь, пока не введут 0  {  addNode(fEnd, element); // Добавляю новый элемент в конец  std::cout << "> ";  std::cin >> element;  }  }  // Добавляю новый узел в конец очереди  void addNode(Node \*&fEnd, int fElement)  {  Node \*fQueue = new (Node); // Создаю новый узел  (\*fQueue).elem = fElement; // Присваиваю ему значение  (\*fQueue).next = NULL; // Этот узел последний, поэтому следующий — NULL  (\*fEnd).next = fQueue; // Связываю текущий конец очереди с новым узлом  fEnd = fQueue; // Обновляю указатель на конец очереди  }  // Удаляю первый узел очереди  int deleteNode(Node \*&fStart)  {  Node \*fQueue; // Временный указатель для удаляемого узла  int klad = 0;  klad = (\*fStart).elem; // Сохраняю значение удаляемого элемента  fQueue = fStart; // Запоминаю первый узел  fStart = (\*fStart).next; // Передвигаю указатель на следующий узел  delete fQueue; // Удаляю первый узел  return klad; // Возвращаю удаленное значение  }  // Печатаю очередь  void printQueue(Node \*fStart)  {  if (fStart == NULL) // Если очередь пуста, показываю сообщение  {  std::cout << "Queue is empty" << std::endl;  }  else  {  Node \*curNode = fStart; // Начинаю с первого узла  std::cout << "Queue: ";  while (curNode != NULL) // Прохожу по всей очереди  {  std::cout << (\*curNode).elem << '\t'; // Вывожу значение текущего узла  curNode = (\*curNode).next; // Перехожу к следующему узлу  }  std::cout << std::endl;  }  }  // Ищу элемент в очереди  int findNode(Node \*fStart, int fElement)  {  int index = 0; // Счетчик для нахождения индекса элемента  Node \*fQueue = fStart; // Временный указатель для прохода по очереди  while (fQueue != NULL) // Пока не дойду до конца очереди  {  if ((\*fQueue).elem != fElement) // Если элемент не тот, продолжаю искать  {  index++;  fQueue = (\*fQueue).next; // Переход к следующему узлу  }  else if ((\*fQueue).elem == fElement) // Если элемент нашелся, возвращаю индекс  {  index++;  return index;  }  }  return 0; // Если элемент не найден, возвращаю 0  }  // Подсчитываю количество узлов в очереди  int countOfNodes(Node \*fStart)  {  int count = 0; // Счетчик узлов  Node \*fQueue = fStart; // Временный указатель для прохода по очереди  while (fQueue != NULL) // Прохожу по всем узлам  {  count++; // Увеличиваю счетчик на каждый узел  fQueue = (\*fQueue).next; // Перехожу к следующему узлу  }  return count; // Возвращаю общее количество узлов  }  // Удаляю всю очередь  void deletingQueue(Node \*&fStart, Node \*&fEnd)  {  Node \*fQueue;  fQueue = fStart; // Начинаю с первого узла  if (fStart != NULL) // Проверяю, есть ли вообще элементы в очереди  {  while (fStart != fEnd) // Удаляю все узлы, кроме последнего  {  fStart = (\*fQueue).next; // Перемещаю указатель на следующий узел  delete fQueue; // Удаляю текущий узел  fQueue = fStart; // Переход к следующему узлу  }  delete fStart; // Удаляю последний узел  fStart = fEnd = NULL; // Обнуляю указатели на начало и конец очереди  }  } |



**Контрольные вопросы**

1. Что такое очередь? Принцип работы очереди?

Очередь — это абстрактная структура данных, работающая по принципу **FIFO** (First In, First Out), что означает "первым вошел — первым вышел". Принцип работы очереди заключается в том, что элементы добавляются в конец очереди (операция **enqueue**) и извлекаются из начала (операция **dequeue**).

1. Отличия очереди от стека?

Основное отличие очереди от стека заключается в порядке извлечения элементов:

* В **стеке** действует принцип **LIFO** (Last In, First Out), то есть последний добавленный элемент извлекается первым.
* В **очереди** действует принцип **FIFO** (First In, First Out), первый добавленный элемент извлекается первым.

1. Действия над очередями?

 **enqueue (вставка)**: добавление элемента в конец очереди;

 **dequeue (удаление)**: удаление элемента из начала очереди;

1. Сколько всего указателей используют при построении очереди?

 **head (или front)** — указывает на начало очереди, откуда элементы извлекаются;

 **tail (или rear)** — указывает на конец очереди, куда элементы добавляются.

1. Какой структурой является очередь?

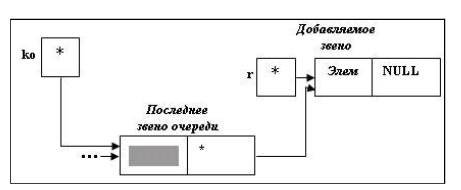
Очередь является **линейной динамической структурой данных**, которая организована последовательно. Это может быть реализовано как с помощью массивов, так и с помощью связных списков.

1. Назовите действия  при построении очереди, которое выполняют следующие операторы:

(\*r).elem=el;   - заполняем информационное поле **elem** числовым значением **el**

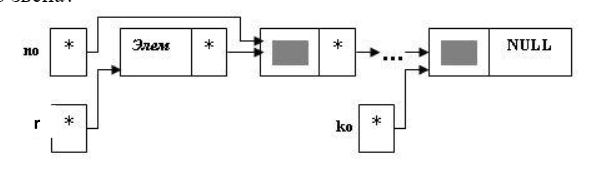
(\*r).sled=NULL;  - адресному полю sled присваиваем NULL

7. Каким оператором можно описать присоединение звена **Элем** к последнему звену очереди?



(\*ko).sled=r;

8. Каким оператором перенастраиваем указатель **no** на звено, следующее после удаляемого звена?

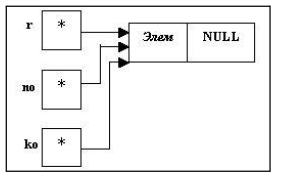


no=(\*no).sled;

9. Можно ли добавить звенья в середину очереди или удалить их из середины очереди?

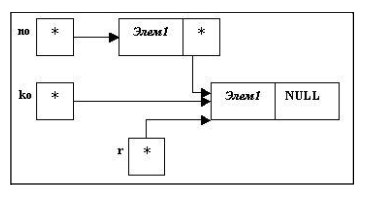
Нет, в классической очереди нельзя добавлять или удалять элементы из середины. Это связано с основным принципом работы очереди — FIFO (первым вошел — первым вышел).

10. Какое действие при построении очереди показано на рисунке?



Настройка указателей начала и конца очереди на первое звено

11. Какое действие при построении очереди показано на рисунке?



Настроили указатель на конец очереди

12. Нужно ли сохранять элемент очереди, перед его удалением? да

|  |  |
| --- | --- |
| void buildingQueue(Node \*&fStart, Node \*&fEnd)  {  Node \*fQueue = new (Node); // Создаю новый узел для начала очереди  int element = 0;  std::cout << "Enter the element of queue (0 = end):" << '\n' << "> ";  std::cin >> element;  (\*fQueue).elem = element; // Сохраняю первый элемент  (\*fQueue).next = NULL; // Пока других узлов нет, следующий — NULL  fStart = fQueue; // Начало очереди — это мой первый узел  fEnd = fQueue; // Конец очереди пока тоже этот узел  std::cout << "> ";  std::cin >> element;  while (element != 0) // Продолжаю строить очередь, пока не введут 0  {  addNode(fEnd, element); // Добавляю новый элемент в конец  std::cout << "> ";  std::cin >> element;  }  } |  |
| void addNode(Node \*&fEnd, int fElement)  {  Node \*fQueue = new (Node); // Создаю новый узел  (\*fQueue).elem = fElement; // Присваиваю ему значение  (\*fQueue).next = NULL; // Этот узел последний, поэтому следующий — NULL  (\*fEnd).next = fQueue; // Связываю текущий конец очереди с новым узлом  fEnd = fQueue; // Обновляю указатель на конец очереди  } |  |
| int deleteNode(Node \*&fStart)  {  Node \*fQueue; // Временный указатель для удаляемого узла  int klad = 0;  klad = (\*fStart).elem; // Сохраняю значение удаляемого элемента  fQueue = fStart; // Запоминаю первый узел  fStart = (\*fStart).next; // Передвигаю указатель на следующий узел  delete fQueue; // Удаляю первый узел  return klad; // Возвращаю удаленное значение  } |  |
| void printQueue(Node \*fStart)  {  if (fStart == NULL) // Если очередь пуста, показываю сообщение  {  std::cout << "Queue is empty" << std::endl;  }  else  {  Node \*curNode = fStart; // Начинаю с первого узла  std::cout << "Queue: ";  while (curNode != NULL) // Прохожу по всей очереди  {  std::cout << (\*curNode).elem << '\t'; // Вывожу значение текущего узла  curNode = (\*curNode).next; // Перехожу к следующему узлу  }  std::cout << std::endl;  }  } |  |
| int findNode(Node \*fStart, int fElement)  {  int index = 0; // Счетчик для нахождения индекса элемента  Node \*fQueue = fStart; // Временный указатель для прохода по очереди  while (fQueue != NULL) // Пока не дойду до конца очереди  {  if ((\*fQueue).elem != fElement) // Если элемент не тот, продолжаю искать  {  index++;  fQueue = (\*fQueue).next; // Переход к следующему узлу  }  else if ((\*fQueue).elem == fElement) // Если элемент нашелся, возвращаю индекс  {  index++;  return index;  }  }  return 0; // Если элемент не найден, возвращаю 0  } |  |
| int countOfNodes(Node \*fStart)  {  int count = 0; // Счетчик узлов  Node \*fQueue = fStart; // Временный указатель для прохода по очереди  while (fQueue != NULL) // Прохожу по всем узлам  {  count++; // Увеличиваю счетчик на каждый узел  fQueue = (\*fQueue).next; // Перехожу к следующему узлу  }  return count; // Возвращаю общее количество узлов  } |  |
| void deletingQueue(Node \*&fStart, Node \*&fEnd)  {  Node \*fQueue;  fQueue = fStart; // Начинаю с первого узла  if (fStart != NULL) // Проверяю, есть ли вообще элементы в очереди  {  while (fStart != fEnd) // Удаляю все узлы, кроме последнего  {  fStart = (\*fQueue).next; // Перемещаю указатель на следующий узел  delete fQueue; // Удаляю текущий узел  fQueue = fStart; // Переход к следующему узлу  }  delete fStart; // Удаляю последний узел  fStart = fEnd = NULL; // Обнуляю указатели на начало и конец очереди  }  } |  |