**Белорусский государственный технологический университет**

**Факультет информационных технологий**

**Кафедра программной инженерии**

**Тестирование кода “** **Тема за 13.05.2025”**

По дисциплине «Основы алгоритмизации и программирования»

Выполнил:

Студент 1 курса 10 группы ПИ

Макаревич Кирилл Витальевич

Проверил: Белодед Николай Иванович

Оглавление

[1. Линейный двунаправленный список 2](#_Toc198111953)

[2. Ортогональный список 6](#_Toc198111954)

[3. Кольцевой список 11](#_Toc198111955)

[4. Стек 20](#_Toc198111956)

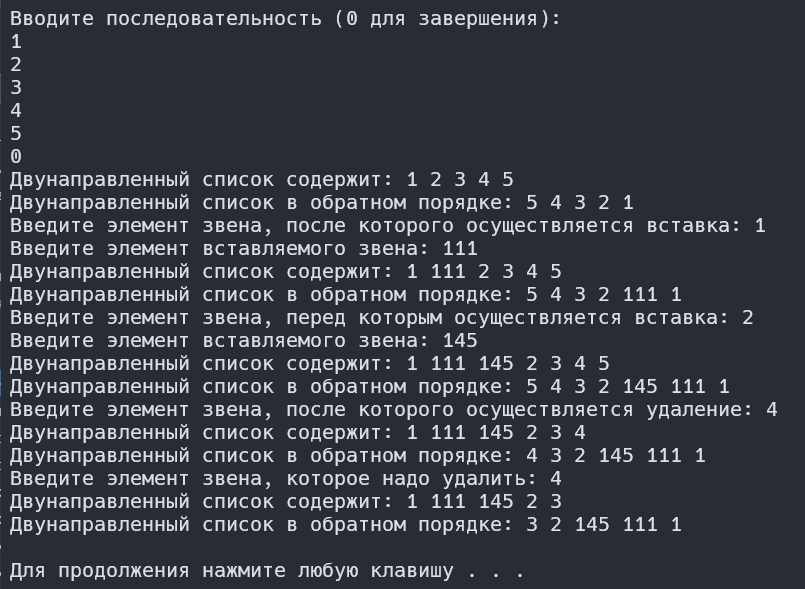
[5. Дек 24](#_Toc198111957)

[6. Очередь 31](#_Toc198111958)

# 1. Линейный двунаправленный список

|  |
| --- |
| #include <iostream>  using namespace std;  // Определение структуры узла двусвязного списка  struct node {  int elem; // Значение, хранимое в узле  node\* sled; // Указатель на следующий узел  node\* pred; // Указатель на предыдущий узел  };  // Класс для работы с двусвязным списком  class Spisok {  private:  node\* nsp; // Указатель на начало списка (заглавный узел)  node\* ksp; // Указатель на конец списка  public:  // Конструктор: инициализирует пустой список  Spisok() { nsp = ksp = NULL; }  // Методы для работы со списком  void Postroenie(); // Создание списка  void VyvodForward(); // Вывод списка от начала к концу  void VyvodBack(); // Вывод списка от конца к началу  void Ochistka(); // Очистка списка (удаление всех узлов)  void InsAfter(int, node\*); // Вставка узла после указанного узла  void InsBefore(int, node\*); // Вставка узла перед указанным узлом  void Delete(node\*); // Удаление указанного узла  void DelAfter(node\*); // Удаление узла, следующего за указанным  node\* PoiskForward(int); // Поиск узла по значению от начала  node\* PoiskBack(int); // Поиск узла по значению от конца  };  // Основная функция программы  void main() {  setlocale(LC\_ALL, "Rus"); // Установка русской локали для вывода текста  Spisok A; // Создание объекта списка  node\* Res; // Указатель для хранения результата поиска  int el, el1; // Переменные для ввода значений  // Создание списка  A.Postroenie();  // Вывод списка в прямом и обратном порядке  A.VyvodForward();  A.VyvodBack();  // Вставка узла после указанного элемента  cout << "Введите элемент звена, после которого ";  cout << "осуществляется вставка: ";  cin >> el;  cout << "Введите элемент вставляемого звена: ";  cin >> el1;  Res = A.PoiskForward(el); // Поиск узла с элементом el  if (Res != NULL) {  A.InsAfter(el1, Res); // Вставка нового узла  A.VyvodForward();  A.VyvodBack();  }  else {  cout << "Звена с заданным элементом в списке нет!\n";  }  // Вставка узла перед указанным элементом  cout << "Введите элемент звена, перед которым ";  cout << "осуществляется вставка: ";  cin >> el;  cout << "Введите элемент вставляемого звена: ";  cin >> el1;  Res = A.PoiskBack(el); // Поиск узла с элементом el  if (Res != NULL) {  A.InsBefore(el1, Res); // Вставка нового узла  A.VyvodForward();  A.VyvodBack();  }  else {  cout << "Звена с заданным элементом в списке нет!\n";  }  // Удаление узла, следующего за указанным  cout << "Введите элемент звена, после которого ";  cout << "осуществляется удаление: ";  cin >> el;  Res = A.PoiskForward(el); // Поиск узла с элементом el  if (Res != NULL) {  A.DelAfter(Res); // Удаление следующего узла  A.VyvodForward();  A.VyvodBack();  }  else {  cout << "Звена с заданным элементом в списке нет!\n";  }  // Удаление указанного узла  cout << "Введите элемент звена, которое ";  cout << "надо удалить: ";  cin >> el;  Res = A.PoiskForward(el); // Поиск узла с элементом el  if (Res != NULL) {  A.Delete(Res); // Удаление узла  A.VyvodForward();  A.VyvodBack();  }  else {  cout << "Звена с заданным элементом в списке нет!\n";  }  // Очистка списка  A.Ochistka();  cout << "\n";  system("PAUSE"); // Пауза перед завершением программы  }  // Метод для создания двусвязного списка  void Spisok::Postroenie() {  node\* rsp; // Вспомогательный указатель для построения  int el; // Переменная для ввода значений  // Создание заглавного узла  nsp = new node;  rsp = nsp;  nsp->pred = NULL;  nsp->sled = NULL;  cout << "Вводите последовательность (0 для завершения):\n";  cin >> el;  // Цикл для добавления узлов, пока не введён 0  while (el != 0) {  rsp->sled = new node; // Создание нового узла  rsp->sled->pred = rsp; // Связь с предыдущим узлом  rsp = rsp->sled; // Переход к новому узлу  rsp->sled = NULL; // Указатель на следующий узел пока пустой  rsp->elem = el; // Запись значения в узел  cin >> el; // Ввод следующего значения  }  ksp = rsp; // Установка указателя на конец списка  }  // Вывод списка от начала к концу  void Spisok::VyvodForward() {  node\* rsp = nsp->sled; // Начало с первого узла после заглавного  cout << "Двунаправленный список содержит: ";  while (rsp != NULL) {  cout << rsp->elem << " "; // Вывод значения узла  rsp = rsp->sled; // Переход к следующему узлу  }  cout << endl;  }  // Вывод списка от конца к началу  void Spisok::VyvodBack() {  node\* rsp = ksp; // Начало с последнего узла  cout << "Двунаправленный список в обратном порядке: ";  while (rsp->pred != NULL) { // Пока не достигнем заглавного узла  cout << rsp->elem << " "; // Вывод значения узла  rsp = rsp->pred; // Переход к предыдущему узлу  }  cout << endl;  }  // Поиск узла по значению от начала списка  node\* Spisok::PoiskForward(int el) {  node\* q = nsp->sled; // Начало с первого узла после заглавного  node\* Res = NULL; // Результат поиска (NULL, если не найден)  while (q != NULL && Res == NULL) {  if (q->elem == el) Res = q; // Если нашли, сохраняем указатель  else q = q->sled; // Иначе переходим к следующему узлу  }  return Res;  }  // Поиск узла по значению от конца списка  node\* Spisok::PoiskBack(int el) {  node\* q = ksp; // Начало с последнего узла  node\* Res = NULL; // Результат поиска (NULL, если не найден)  while (q != NULL && Res == NULL) {  if (q->elem == el) Res = q; // Если нашли, сохраняем указатель  else q = q->pred; // Иначе переходим к предыдущему узлу  }  return Res;  }  // Вставка узла с заданным значением после указанного узла  void Spisok::InsAfter(int el, node\* Res) {  node\* q = new node; // Создание нового узла  q->elem = el; // Запись значения в узел  if (Res->sled != NULL) { // Если есть следующий узел  q->sled = Res->sled; // Связь нового узла с последующим  q->pred = Res->sled->pred; // Связь с предыдущим (Res)  Res->sled->pred = q; // Обновление указателя у следующего узла  Res->sled = q; // Связь Res с новым узлом  }  else { // Если Res — последний узел  q->sled = NULL; // Новый узел становится последним  q->pred = Res; // Связь с Res  ksp = q; // Обновление конца списка  Res->sled = q; // Связь Res с новым узлом  }  }  // Вставка узла с заданным значением перед указанным узлом  void Spisok::InsBefore(int el, node\* Res) {  node\* q = new node; // Создание нового узла  q->elem = el; // Запись значения в узел  q->sled = Res; // Новый узел указывает на Res  q->pred = Res->pred; // Связь с предыдущим узлом  Res->pred->sled = q; // Обновление указателя у предыдущего узла  Res->pred = q; // Обновление указателя у Res  }  // Удаление указанного узла  void Spisok::Delete(node\* Res) {  if (Res->sled != NULL) { // Если Res не последний узел  Res->sled->pred = Res->pred; // Связь следующего узла с предыдущим  Res->pred->sled = Res->sled; // Связь предыдущего с последующим  delete Res; // Освобождение памяти  }  else { // Если Res — последний узел  Res->pred->sled = NULL; // Предыдущий узел становится последним  ksp = ksp->pred; // Обновление конца списка  delete Res; // Освобождение памяти  }  }  // Удаление узла, следующего за указанным  void Spisok::DelAfter(node\* Res) {  if (Res->sled == NULL) { // Если следующего узла нет  cout << "Указано последнее звено!\n";  }  else if (Res->sled->sled != NULL) { // Если есть узел после следующего  node\* q = Res->sled; // Сохраняем указатель на удаляемый узел  Res->sled->sled->pred = Res; // Связь следующего узла с Res  Res->sled = Res->sled->sled; // Обновление указателя у Res  delete q; // Освобождение памяти  }  else { // Если удаляемый узел — последний  node\* q = Res->sled; // Сохраняем указатель на удаляемый узел  Res->sled = NULL; // Res становится последним  ksp = ksp->pred; // Обновление конца списка  delete q; // Освобождение памяти  }  }  // Очистка всего списка  void Spisok::Ochistka() {  node\* q = nsp; // Начало с заглавного узла  node\* q1 = q->sled; // Указатель на следующий узел  while (q1 != NULL) { // Пока есть узлы  q = q1; // Переход к следующему узлу  q1 = q1->sled; // Сохранение следующего узла  delete q; // Освобождение памяти  }  delete nsp; // Удаление заглавного узла  nsp = ksp = NULL; // Сброс указателей  } |

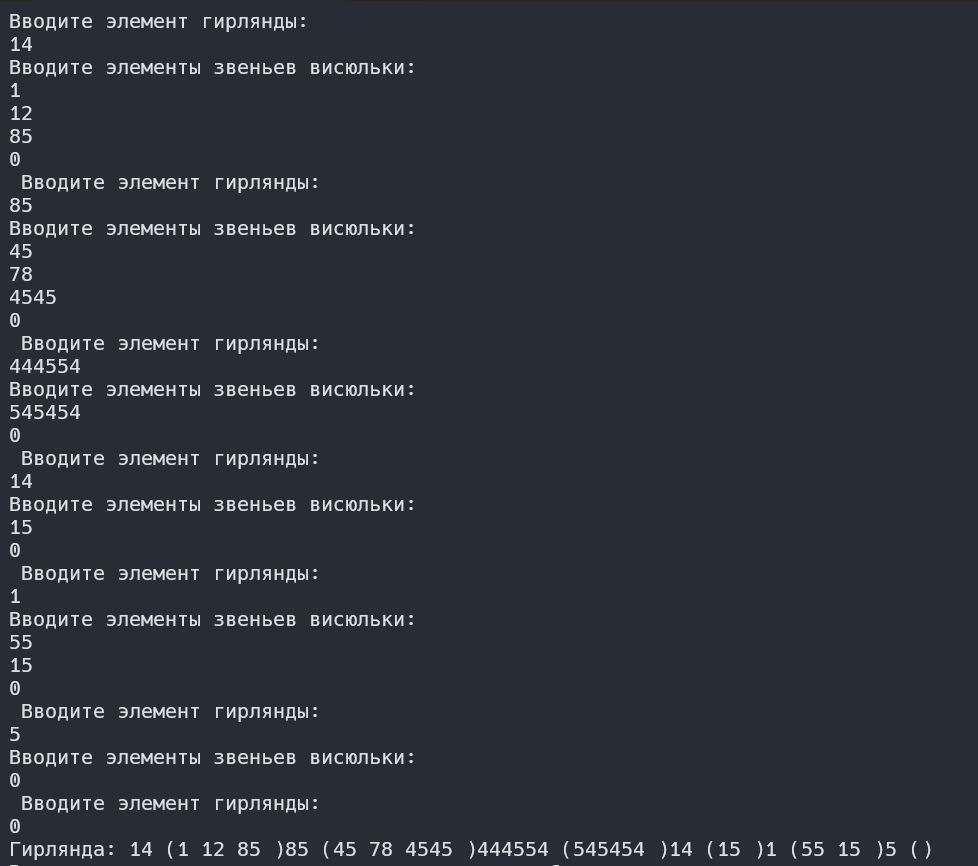
Вывод:



# 2. Ортогональный список

|  |
| --- |
| include <iostream>  using namespace std;  // Описание типа звена висюльки  struct nodeVis {  int elem; // Информационное поле звена висюльки (значение)  nodeVis\* vniz; // Указатель на следующее звено висюльки  };  // Описание типа звена гирлянды  struct nodeGir {  int elem; // Информационное поле звена гирлянды (значение)  nodeVis\* vniz; // Указатель на начало висюльки (список)  nodeGir\* sled; // Указатель на следующее звено гирлянды  };  // Класс для работы с гирляндой и висюльками  class GirVis {  private:  nodeGir\* phead; // Указатель на заглавный узел гирлянды  nodeVis\* pheadVis; // Указатель на заглавный узел текущей висюльки  // Внутренний метод для вывода висюльки  void VisVyvod();  public:  // Конструктор: создаёт заглавный узел гирлянды  GirVis() { phead = new nodeGir; }  // Деструктор: освобождает память, занятую заглавным узлом гирлянды  ~GirVis() { delete phead; }  // Методы для работы с висюлькой  nodeVis\* VisPostr(); // Построение висюльки  nodeVis\* VisPoisk(int); // Поиск элемента в висюльке  void SetpheadVis(nodeVis\* r) { pheadVis = r; } // Установка указателя на голову висюльки  void VisVstav(nodeVis\*, int); // Вставка звена в висюльку после указанного  void Vis1Vstav(nodeVis\*, int); // Вставка звена в висюльку перед указанным  void VisUdale(nodeVis\*); // Удаление звена висюльки после указанного  void Vis1Udale(nodeVis\*); // Удаление указанного звена висюльки  // Методы для работы с гирляндой  void GirPostr(); // Построение гирлянды  void GirVyvod(); // Вывод гирлянды и её висюлек  nodeGir\* GirPoisk(int); // Поиск элемента в гирлянде  void OCHISTKA(); // Очистка всей гирлянды и висюлек  void OCHISTKA1(); // Очистка текущей висюльки  };  // Основная функция программы  void main() {  setlocale(LC\_ALL, "Rus"); // Установка русской локали для вывода текста  GirVis A; // Создание объекта гирлянды  int el, elGir, elVis; // Переменные для ввода значений  nodeGir\* Res; // Указатель на узел гирлянды  nodeVis\* ResVis; // Указатель на узел висюльки  // Построение гирлянды и её висюлек  A.GirPostr();  A.GirVyvod();  // Вставка элемента в висюльку после указанного узла  cout << "\nВведите элемент звена гирлянды, ";  cout << "чьи висюльки будем изменять:\n";  cin >> elGir;  cout << "\nВведите элемент звена висюльки, после которого ";  cout << "осуществляется вставка:\n";  cin >> elVis;  cout << "\nВведите вставляемый элемент:\n";  cin >> el;  Res = A.GirPoisk(elGir); // Поиск узла гирлянды  if (Res != NULL) {  A.SetpheadVis((\*Res).vniz); // Установка указателя на висюльку  ResVis = A.VisPoisk(elVis); // Поиск узла висюльки  if (ResVis != NULL)  A.VisVstav(ResVis, el); // Вставка нового элемента  else  cout << "Элемента в висюльке нет!\n";  }  else {  cout << "Элемента в гирлянде нет\n";  }  A.GirVyvod();  // Вставка элемента в висюльку перед указанным узлом  cout << "\nВведите элемент гирлянды, чью висюльку будем изменять:\n";  cin >> elGir;  cout << "Введите элемент висюльки, перед которым ";  cout << "осуществляется вставка:\n";  cin >> elVis;  cout << "Введите вставляемый элемент:\n";  cin >> el;  Res = A.GirPoisk(elGir); // Поиск узла гирлянды  if (Res != NULL) {  A.SetpheadVis((\*Res).vniz); // Установка указателя на висюльку  ResVis = A.VisPoisk(elVis); // Поиск узла висюльки  if (ResVis != NULL)  A.Vis1Vstav(ResVis, el); // Вставка нового элемента  else  cout << "Элемента в висюльке нет!\n";  }  else {  cout << "Элемента в гирлянде нет!\n";  }  A.GirVyvod();  // Удаление узла висюльки после указанного  cout << "\nВведите элемент гирлянды, чью висюльку будем изменять:\n";  cin >> elGir;  cout << "Введите элемент висюльки, после которого нужно удалить:\n";  cin >> elVis;  Res = A.GirPoisk(elGir); // Поиск узла гирлянды  if (Res != NULL) {  A.SetpheadVis((\*Res).vniz); // Установка указателя на висюльку  ResVis = A.VisPoisk(elVis); // Поиск узла висюльки  if (ResVis != NULL && (\*ResVis).vniz != NULL)  A.VisUdale(ResVis); // Удаление следующего узла  else  cout << "Элемента в висюльке нет!\n";  }  else {  cout << "Элемента в гирлянде нет!\n";  }  A.GirVyvod();  // Удаление указанного узла висюльки  cout << "\nВведите элемент гирлянды, чью висюльку будем изменять:\n";  cin >> elGir;  cout << "Введите элемент висюльки, который удаляется:\n";  cin >> elVis;  Res = A.GirPoisk(elGir); // Поиск узла гирлянды  if (Res != NULL) {  A.SetpheadVis((\*Res).vniz); // Установка указателя на висюльку  ResVis = A.VisPoisk(elVis); // Поиск узла висюльки  if (ResVis != NULL && (\*ResVis).vniz != NULL)  A.Vis1Udale(ResVis); // Удаление узла  else  cout << "Элемента в висюльке нет или он последний!\n";  }  else {  cout << "Элемента в гирлянде нет!\n";  }  A.GirVyvod();  // Очистка всей структуры  A.OCHISTKA();  cout << "\n";  system("PAUSE"); // Пауза перед завершением программы  }  // Очистка всей гирлянды и её висюлек  void GirVis::OCHISTKA() {  nodeGir\* q, \* q1; // Рабочие указатели для обхода гирлянды  q = phead;  q1 = (\*q).sled; // Указатель q1 опережает q  while (q1 != NULL) {  q = q1; // Переход к следующему узлу  q1 = (\*q1).sled; // Сохранение следующего узла  pheadVis = (\*q).vniz; // Установка указателя на висюльку  OCHISTKA1(); // Очистка висюльки  delete q; // Освобождение узла гирлянды  }  }  // Очистка текущей висюльки  void GirVis::OCHISTKA1() {  nodeVis\* q, \* q1; // Рабочие указатели для обхода висюльки  q = pheadVis;  q1 = (\*q).vniz; // Указатель q1 опережает q  while (q1 != NULL) {  q = q1; // Переход к следующему узлу  q1 = (\*q1).vniz; // Сохранение следующего узла  delete q; // Освобождение узла висюльки  }  }  // Построение гирлянды (односвязного списка с заглавным узлом)  void GirVis::GirPostr() {  nodeGir\* t; // Вспомогательный указатель  int el; // Переменная для ввода значений  t = phead; // Начало с заглавного узла  (\*t).sled = NULL; // Инициализация заглавного узла  cout << "Вводите элемент гирлянды: \n";  cin >> el;  while (el != 0) { // Ввод до 0  (\*t).sled = new nodeGir; // Создание нового узла  t = (\*t).sled; // Переход к новому узлу  (\*t).elem = el; // Запись значения  (\*t).sled = NULL; // Указатель на следующий узел пока пустой  (\*t).vniz = VisPostr(); // Построение висюльки для этого узла  cout << " Вводите элемент гирлянды: \n";  cin >> el; // Ввод следующего значения  }  }  // Построение висюльки (односвязного списка с заглавным узлом)  nodeVis\* GirVis::VisPostr() {  nodeVis\* t; // Вспомогательный указатель  int el; // Переменная для ввода значений  pheadVis = new nodeVis; // Создание заглавного узла висюльки  t = pheadVis; // Начало с заглавного узла  (\*t).vniz = NULL; // Инициализация заглавного узла  cout << "Вводите элементы звеньев висюльки: \n";  cin >> el;  while (el != 0) { // Ввод до 0  (\*t).vniz = new nodeVis; // Создание нового узла  t = (\*t).vniz; // Переход к новому узлу  (\*t).elem = el; // Запись значения  (\*t).vniz = NULL; // Указатель на следующий узел пока пустой  cin >> el; // Ввод следующего значения  }  return pheadVis; // Возврат указателя на заглавный узел висюльки  }  // Вывод гирлянды и её висюлек  void GirVis::GirVyvod() {  nodeGir\* t; // Вспомогательный указатель  t = phead; t = (\*t).sled; // Начало с первого узла после заглавного  cout << "Гирлянда: ";  while (t != NULL) {  cout << (\*t).elem << " "; // Вывод значения узла гирлянды  pheadVis = (\*t).vniz; // Установка указателя на висюльку  VisVyvod(); // Вывод висюльки  t = (\*t).sled; // Переход к следующему узлу гирлянды  }  }  // Поиск элемента в гирлянде  nodeGir\* GirVis::GirPoisk(int el) {  nodeGir\* t, \* r; // t - текущий узел, r - результат  r = NULL; // Инициализация результата  t = phead; t = (\*t).sled; // Начало с первого узла после заглавного  while (t != NULL && r == NULL) {  if ((\*t).elem == el) r = t; // Если нашли, сохраняем указатель  else t = (\*t).sled; // Иначе переходим к следующему узлу  }  return r; // Возврат найденного узла или NULL  }  // Вывод висюльки  void GirVis::VisVyvod() {  nodeVis\* t; // Вспомогательный указатель  t = pheadVis; t = (\*t).vniz; // Начало с первого узла после заглавного  cout << "(";  while (t != NULL) {  cout << (\*t).elem << " "; // Вывод значения узла висюльки  t = (\*t).vniz; // Переход к следующему узлу  }  cout << ")";  }  // Поиск элемента в висюльке  nodeVis\* GirVis::VisPoisk(int el) {  nodeVis\* t, \* r; // t - текущий узел, r - результат  r = NULL; // Инициализация результата  t = pheadVis; t = (\*t).vniz; // Начало с первого узла после заглавного  while (t != NULL && r == NULL) {  if ((\*t).elem == el) r = t; // Если нашли, сохраняем указатель  else t = (\*t).vniz; // Иначе переходим к следующему узлу  }  return r; // Возврат найденного узла или NULL  }  // Вставка звена в висюльку после указанного узла  void GirVis::VisVstav(nodeVis\* r, int el) {  nodeVis\* q; // Новый узел  q = new nodeVis; // Создание нового узла  (\*q).elem = el; // Запись значения  (\*q).vniz = (\*r).vniz; // Связь с последующим узлом  (\*r).vniz = q; // Связь указанного узла с новым  }  // Вставка звена в висюльку перед указанным узлом  void GirVis::Vis1Vstav(nodeVis\* r, int el) {  nodeVis\* q; // Новый узел  q = new nodeVis; // Создание нового узла  (\*q).elem = (\*r).elem; // Копирование значения указанного узла  (\*q).vniz = (\*r).vniz; // Копирование указателя на следующий  (\*r).elem = el; // Запись нового значения в указанный узел  (\*r).vniz = q; // Связь с новым узлом  }  // Удаление звена висюльки после указанного  void GirVis::VisUdale(nodeVis\* r) {  nodeVis\* q; // Удаляемый узел  q = (\*r).vniz; // Сохранение указателя на удаляемый узел  if ((\*r).vniz != NULL) {  (\*r).vniz = (\*(\*r).vniz).vniz; // Связь с узлом после удаляемого  delete q; // Освобождение памяти  }  else {  cout << "Звено с заданным элементом - последнее!\n";  }  }  // Удаление указанного звена висюльки  void GirVis::Vis1Udale(nodeVis\* r) {  nodeVis\* g; // Удаляемый узел  if ((\*r).vniz != NULL) {  g = (\*r).vniz; // Сохранение указателя на следующий узел  (\*r).elem = (\*(\*r).vniz).elem; // Копирование значения следующего узла  (\*r).vniz = (\*(\*r).vniz).vniz; // Связь с узлом после следующего  delete g; // Освобождение памяти  }  else {  cout << "Не умею удалять последнее звено!\n";  }  } |

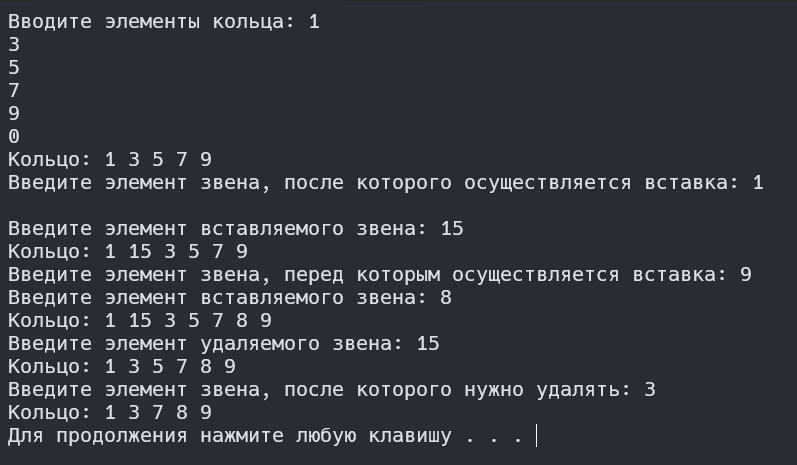
Вывод:



# 3. Кольцевой список

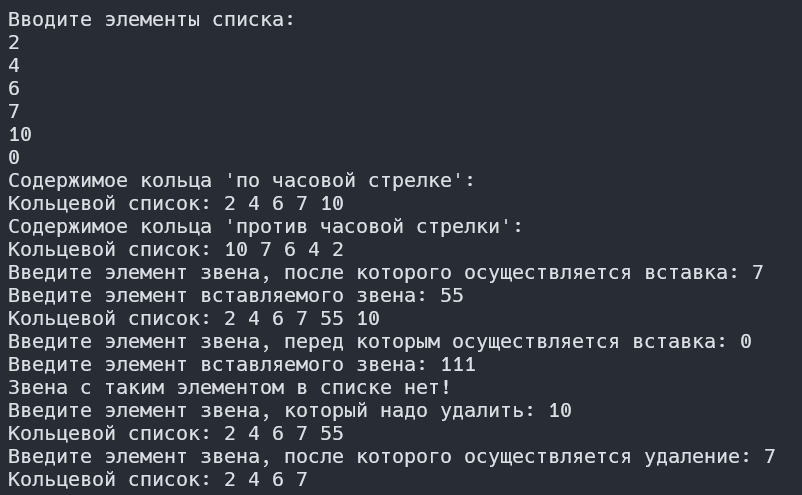
|  |
| --- |
| #include <iostream>  using namespace std;  // Описание структуры узла кольцевого списка  struct node {  int elem; // Информационное поле узла (значение)  node\* sled; // Указатель на следующий узел  };  // Класс для работы с кольцевым односвязным списком  class Spisok {  private:  node\* phead; // Указатель на заглавный узел списка  node\* Res; // Указатель на найденный узел (результат поиска)  public:  // Конструктор: создаёт заглавный узел  Spisok() { phead = new node; Res = NULL; }  // Деструктор: освобождает память, занятую заглавным узлом  ~Spisok() { delete phead; }  // Методы для работы со списком  void POSTROENIE(); // Построение кольцевого списка  void VYVOD(); // Вывод списка  node\* POISK(int); // Поиск узла по значению  void InsAfter(int); // Вставка узла после найденного  void InsBefore(int); // Вставка узла перед найденным  void Delete(); // Удаление найденного узла  void DelAfter(); // Удаление узла после найденного  void OCHISTKA(); // Очистка списка  };  // Основная функция программы  void main() {  setlocale(LC\_ALL, "Rus"); // Установка русской локали для вывода текста  Spisok A; // Создание объекта списка  int el, el1; // Переменные для ввода значений  node\* Res\_Zn; // Указатель для хранения результата поиска  // Построение кольцевого списка  A.POSTROENIE();  A.VYVOD(); // Вывод списка  // Вставка узла после указанного элемента  cout << "\nВведите элемент звена, после которого ";  cout << "осуществляется вставка: ";  cin >> el;  cout << "\nВведите элемент вставляемого звена: ";  cin >> el1;  if (A.POISK(el) != NULL) { // Поиск узла с элементом el  A.InsAfter(el1); // Вставка нового узла  A.VYVOD(); // Вывод списка  }  else {  cout << "Звена с заданным элементом в кольце нет!";  }  // Вставка узла перед указанным элементом  cout << "\nВведите элемент звена, перед которым ";  cout << "осуществляется вставка: ";  cin >> el;  cout << "Введите элемент вставляемого звена: ";  cin >> el1;  if (A.POISK(el) != NULL) { // Поиск узла с элементом el  A.InsBefore(el1); // Вставка нового узла  A.VYVOD(); // Вывод списка  }  else {  cout << "Звена с заданным элементом в кольце нет!";  }  // Удаление указанного узла  cout << "\nВведите элемент удаляемого звена: ";  cin >> el;  if (A.POISK(el) != NULL) { // Поиск узла с элементом el  A.Delete(); // Удаление узла  A.VYVOD(); // Вывод списка  }  else {  cout << "Звена с заданным элементом в кольце нет!";  }  // Удаление узла после указанного  cout << "\nВведите элемент звена, ";  cout << "после которого нужно удалять: ";  cin >> el;  if (A.POISK(el) != NULL) { // Поиск узла с элементом el  A.DelAfter(); // Удаление следующего узла  A.VYVOD(); // Вывод списка  }  else {  cout << " Звена с заданным элементом в кольце нет!";  }  // Очистка списка  A.OCHISTKA();  cout << "\n";  system("PAUSE"); // Пауза перед завершением программы  }  // Построение кольцевого списка  void Spisok::POSTROENIE() {  node\* t; // Вспомогательный указатель для построения  int el; // Переменная для ввода значений  t = phead; // Начало с заглавного узла  (\*t).sled = NULL; // Инициализация заглавного узла  cout << "Вводите элементы кольца: ";  cin >> el;  while (el != 0) { // Ввод значений до 0  (\*t).sled = new node; // Создание нового узла  t = (\*t).sled; // Переход к новому узлу  (\*t).elem = el; // Запись значения  cin >> el; // Ввод следующего значения  }  (\*t).sled = (\*phead).sled; // Замыкание кольца: последний узел указывает на первый  }  // Вывод содержимого кольцевого списка  void Spisok::VYVOD() {  node\* t = (\*phead).sled; // Начало с первого узла после заглавного  cout << "Кольцо: ";  if (t != NULL) { // Если список не пуст  cout << (\*t).elem << " "; // Вывод первого элемента  t = (\*t).sled; // Переход к следующему  while (t != (\*phead).sled) { // Пока не вернёмся к началу  cout << (\*t).elem << " "; // Вывод значения  t = (\*t).sled; // Переход к следующему  }  }  else {  cout << "пусто!\n"; // Если список пуст  }  }  // Поиск узла по значению  node\* Spisok::POISK(int el) {  node\* t = (\*phead).sled; // Начало с первого узла после заглавного  Res = NULL; // Инициализация результата поиска  // Поиск до предпоследнего узла  while ((\*t).sled != (\*phead).sled && Res == NULL) {  if ((\*t).elem == el) Res = t; // Если нашли, сохраняем указатель  else t = (\*t).sled; // Иначе переходим к следующему  }  // Проверка последнего узла  if (Res == NULL && (\*t).elem == el)  Res = t;  return Res; // Возврат найденного узла или NULL  }  // Вставка узла после найденного  void Spisok::InsAfter(int el) {  node\* q = new node; // Создание нового узла  (\*q).elem = el; // Запись значения  (\*q).sled = (\*Res).sled; // Связь с последующим узлом  (\*Res).sled = q; // Связь найденного узла с новым  }  // Вставка узла перед найденным  void Spisok::InsBefore(int el) {  node\* q = new node; // Создание нового узла  (\*q).elem = (\*Res).elem; // Копирование значения найденного узла  (\*q).sled = (\*Res).sled; // Копирование указателя на следующий  (\*Res).elem = el; // Запись нового значения в найденный узел  (\*Res).sled = q; // Связь с новым узлом  }  // Удаление найденного узла  void Spisok::Delete() {  node\* z, \* q; // Вспомогательные указатели  if ((\*Res).sled != (\*phead).sled) { // Если не последний узел  q = (\*Res).sled; // Сохранение следующего узла  (\*Res).elem = (\*q).elem; // Копирование значения следующего  (\*Res).sled = (\*q).sled; // Связь с узлом после следующего  delete q; // Освобождение памяти  }  else if ((\*Res).sled == Res) { // Если единственный узел  q = (\*phead).sled; // Сохранение узла  (\*phead).sled = NULL; // Список становится пустым  delete q; // Освобождение памяти  cout << "Кольцо пусто!";  }  else { // Если последний узел  z = phead; // Начало с заглавного узла  q = (\*phead).sled; // Первый узел  while (q != Res) { // Поиск узла перед Res  z = q;  q = (\*q).sled;  }  (\*z).sled = (\*q).sled; // Связь с узлом после Res  delete q; // Освобождение памяти  }  }  // Удаление узла после найденного  void Spisok::DelAfter() {  node\* q; // Удаляемый узел  if ((\*Res).sled != (\*phead).sled) { // Если не последний узел  q = (\*Res).sled; // Сохранение следующего узла  (\*Res).sled = (\*q).sled; // Связь с узлом после следующего  delete q; // Освобождение памяти  }  else if ((\*Res).sled == Res) { // Если единственный узел  q = (\*phead).sled; // Сохранение узла  (\*phead).sled = NULL; // Список становится пустым  delete q; // Освобождение памяти  cout << "Кольцо пусто!";  }  else { // Если удаляется первый узел  q = (\*phead).sled; // Сохранение первого узла  (\*Res).sled = (\*q).sled; // Связь с узлом после первого  (\*phead).sled = (\*Res).sled; // Обновление начала списка  delete q; // Освобождение памяти  }  }  // Очистка кольцевого списка  void Spisok::OCHISTKA() {  node\* q, \* q1; // Рабочие указатели  q = phead; // Начало с заглавного узла  q1 = (\*q).sled; // Первый узел  if (q1 != NULL) { // Если список не пуст  do {  q = q1; // Переход к следующему узлу  q1 = (\*q1).sled; // Сохранение следующего узла  delete q; // Освобождение памяти  } while (q1 != (\*phead).sled); // Пока не вернёмся к началу  }  (\*phead).sled = NULL; // Сброс указателя заглавного узла  } |

Вывод:



|  |
| --- |
| #include <iostream>  using namespace std;  // Описание структуры узла двусвязного кольцевого списка  struct node {  int elem; // Информационное поле узла (значение)  node\* sled; // Указатель на следующий узел  node\* pred; // Указатель на предыдущий узел  };  // Класс для работы с двусвязным кольцевым списком  class Spisok {  private:  node\* nsp; // Указатель на заглавный узел списка  public:  // Конструктор: инициализирует пустой список  Spisok() { nsp = NULL; }  // Методы для работы со списком  void BuiltRing(); // Построение кольцевого списка  void VyvodLeftRight(); // Вывод списка "по часовой стрелке"  void VyvodRightLeft(); // Вывод списка "против часовой стрелки"  void InsAfter(node\*, int); // Вставка узла после указанного  void InsBefore(node\*, int); // Вставка узла перед указанным  void Delete(node\*); // Удаление указанного узла  void DelAfter(node\*); // Удаление узла после указанного  node\* SearchRing(int); // Поиск узла по значению  void Ochistka(); // Очистка списка  };  // Основная функция программы  void main() {  setlocale(LC\_ALL, "Rus"); // Установка русской локали для вывода текста  Spisok A; // Создание объекта списка  node\* Res; // Указатель для хранения результата поиска  int el, el1; // Переменные для ввода значений  // Построение кольцевого списка  A.BuiltRing();  cout << "Содержимое кольца 'по часовой стрелке': \n";  A.VyvodLeftRight(); // Вывод списка от первого узла вправо  cout << "Содержимое кольца 'против часовой стрелки': \n";  A.VyvodRightLeft(); // Вывод списка от последнего узла влево  // Вставка узла после указанного элемента  cout << "Введите элемент звена, после которого ";  cout << "осуществляется вставка: ";  cin >> el;  cout << "Введите элемент вставляемого звена: ";  cin >> el1;  Res = A.SearchRing(el); // Поиск узла с элементом el  if (Res != NULL) { // Если узел найден  A.InsAfter(Res, el1); // Вставка нового узла  A.VyvodLeftRight(); // Вывод списка  }  else {  cout << "Звена с таким элементом в списке нет!\n";  }  // Вставка узла перед указанным элементом  cout << "Введите элемент звена, перед которым ";  cout << "осуществляется вставка: ";  cin >> el;  cout << "Введите элемент вставляемого звена: ";  cin >> el1;  Res = A.SearchRing(el); // Поиск узла с элементом el  if (Res != NULL) { // Если узел найден  A.InsBefore(Res, el1); // Вставка нового узла  A.VyvodLeftRight(); // Вывод списка  }  else {  cout << "Звена с таким элементом в списке нет!\n";  }  // Удаление указанного узла  cout << "Введите элемент звена, который ";  cout << "надо удалить: ";  cin >> el;  Res = A.SearchRing(el); // Поиск узла с элементом el  if (Res != NULL) { // Если узел найден  A.Delete(Res); // Удаление узла  A.VyvodLeftRight(); // Вывод списка  }  else {  cout << "Звена с таким элементом в списке нет!\n";  }  // Удаление узла после указанного  cout << "Введите элемент звена, после которого ";  cout << "осуществляется удаление: ";  cin >> el;  Res = A.SearchRing(el); // Поиск узла с элементом el  if (Res != NULL) { // Если узел найден  A.DelAfter(Res); // Удаление следующего узла  A.VyvodLeftRight(); // Вывод списка  }  else {  cout << "Звена с таким элементом в списке нет!\n";  }  // Очистка списка  A.Ochistka();  cout << "\n";  system("PAUSE"); // Пауза перед завершением программы  }  // Построение двусвязного кольцевого списка  void Spisok::BuiltRing() {  node\* r; // Вспомогательный указатель для построения  int el; // Переменная для ввода значений  // Создание заглавного узла  nsp = new node;  r = nsp;  (\*nsp).pred = NULL;  (\*nsp).sled = NULL;  cout << "Вводите элементы списка: \n";  cin >> el;  while (el != 0) { // Ввод значений до 0  (\*r).sled = new node; // Создание нового узла  (\*((\*r).sled)).pred = r; // Связь нового узла с предыдущим  r = (\*r).sled; // Переход к новому узлу  (\*r).sled = NULL; // Указатель на следующий узел пуст  (\*r).elem = el; // Запись значения  cin >> el; // Ввод следующего значения  }  // Замыкание кольца  if ((\*nsp).sled != NULL) { // Если список не пуст  (\*((\*nsp).sled)).pred = r; // Связь первого узла с последним  (\*r).sled = (\*nsp).sled; // Связь последнего узла с первым  }  else { // Если список пуст  cout << "Кольцевой список пуст!\n";  }  }  // Вывод списка "по часовой стрелке" (от первого узла вправо)  void Spisok::VyvodLeftRight() {  node\* r; // Вспомогательный указатель  cout << "Кольцевой список: ";  if ((\*nsp).sled != NULL) { // Если список не пуст  cout << (\*((\*nsp).sled)).elem << " "; // Вывод первого элемента  r = (\*((\*nsp).sled)).sled; // Переход к следующему узлу  while (r != (\*nsp).sled) { // Пока не вернёмся к первому  cout << (\*r).elem << " "; // Вывод значения  r = (\*r).sled; // Переход к следующему  }  cout << endl;  }  else {  cout << "пуст!";  }  }  // Вывод списка "против часовой стрелки" (от последнего узла влево)  void Spisok::VyvodRightLeft() {  node\* r; // Вспомогательный указатель  cout << "Кольцевой список: ";  if ((\*nsp).sled != NULL) { // Если список не пуст  cout << (\*((\*((\*nsp).sled)).pred)).elem << " "; // Вывод последнего элемента  r = (\*((\*((\*nsp).sled)).pred)).pred; // Переход к предыдущему узлу  while (r != (\*((\*nsp).sled)).pred) { // Пока не вернёмся к последнему  cout << (\*r).elem << " "; // Вывод значения  r = (\*r).pred; // Переход к предыдущему  }  cout << endl;  }  else {  cout << "пуст!";  }  }  // Поиск узла по значению  node\* Spisok::SearchRing(int el) {  node\* q; // Вспомогательный указатель для обхода  node\* p; // Указатель на заглавный узел  node\* Res; // Указатель для результата поиска  Res = NULL; // Инициализация результата  p = nsp; // Начало с заглавного узла  if ((\*((\*p).sled)).elem == el) { // Проверка первого узла  Res = (\*p).sled;  }  else { // Поиск по остальным узлам  q = (\*((\*p).sled)).sled; // Начало со второго узла  while (q != (\*p).sled && Res == NULL) { // Пока не вернёмся к первому  if ((\*q).elem == el) Res = q; // Если нашли, сохраняем указатель  else q = (\*q).sled; // Иначе переходим к следующему  }  }  return Res; // Возврат найденного узла или NULL  }  // Вставка узла после указанного  void Spisok::InsAfter(node\* Res, int el) {  node\* q = new node; // Создание нового узла  (\*q).elem = el; // Запись значения  (\*q).sled = (\*Res).sled; // Связь с последующим узлом  (\*q).pred = (\*(\*Res).sled).pred; // Связь с предыдущим (Res)  (\*(\*Res).sled).pred = q; // Обновление указателя у следующего узла  (\*Res).sled = q; // Связь Res с новым узлом  }  // Вставка узла перед указанным  void Spisok::InsBefore(node\* Res, int el) {  node\* q = new node; // Создание нового узла  (\*q).elem = el; // Запись значения  (\*q).sled = (\*(\*Res).pred).sled; // Связь с Res  (\*q).pred = (\*Res).pred; // Связь с предыдущим узлом  (\*(\*Res).pred).sled = q; // Обновление указателя у предыдущего  (\*Res).pred = q; // Обновление указателя у Res  if (Res == (\*nsp).sled) { // Если вставка перед первым узлом  (\*nsp).sled = q; // Обновление указателя на первый узел  }  }  // Удаление указанного узла  void Spisok::Delete(node\* Res) {  if ((\*Res).sled == Res) { // Если в кольце один узел  (\*nsp).sled = NULL; // Список становится пустым  delete Res; // Освобождение памяти  }  else { // Если в кольце несколько узлов  (\*(\*Res).sled).pred = (\*Res).pred; // Связь следующего с предыдущим  (\*(\*Res).pred).sled = (\*Res).sled; // Связь предыдущего с последующим  if ((\*nsp).sled == Res) { // Если удаляется первый узел  (\*nsp).sled = (\*Res).sled; // Обновление указателя на первый  }  delete Res; // Освобождение памяти  }  }  // Удаление узла после указанного  void Spisok::DelAfter(node\* Res) {  node\* q; // Указатель для удаляемого узла  if ((\*Res).sled == Res) { // Если в кольце один узел  (\*nsp).sled = NULL; // Список становится пустым  delete Res; // Освобождение памяти  }  else { // Если в кольце несколько узлов  q = (\*Res).sled; // Сохранение указателя на удаляемый узел  (\*(\*(\*Res).sled).sled).pred = (\*(\*Res).sled).pred; // Связь следующего с Res  (\*Res).sled = (\*(\*Res).sled).sled; // Связь Res с узлом после удаляемого  if ((\*(\*nsp).sled).pred == Res) { // Если удаляется последний узел  (\*nsp).sled = (\*Res).sled; // Обновление указателя на первый  }  delete q; // Освобождение памяти  }  }  // Очистка кольцевого списка  void Spisok::Ochistka() {  node\* q, \* q1; // Рабочие указатели  if ((\*nsp).sled != NULL) { // Если список не пуст  q = (\*((\*nsp).sled)).sled; // Начало со второго узла  q1 = (\*q).sled; // Сохранение следующего узла  while (q1 != (\*((\*nsp).sled)).sled) { // Пока не вернёмся к первому  delete q; // Освобождение текущего узла  q = q1; // Переход к следующему  q1 = (\*q1).sled; // Сохранение следующего узла  }  delete q; // Освобождение последнего узла  }  delete nsp; // Освобождение заглавного узла  nsp = NULL; // Сброс указателя  } |

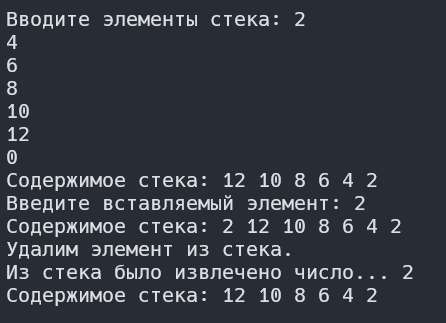
Вывод:



# 4. Стек

|  |
| --- |
| #include <iostream>  using namespace std;  // Описание структуры узла стека  struct node {  int elem; // Информационное поле узла (значение)  node\* sled; // Указатель на следующий узел  };  // Класс для работы со стеком  class Spisok {  private:  node\* stk; // Указатель на вершину стека  int klad; // Переменная для хранения значения удалённого элемента  public:  // Конструктор: инициализирует пустой стек  Spisok() { stk = NULL; }  // Метод для получения значения последнего удалённого элемента  int Set\_Stack() { return klad; }  // Методы для работы со стеком  void POSTROENIE(); // Построение стека  void VYVOD(); // Вывод содержимого стека  void W\_S(int); // Добавление элемента в стек  void YDALENIE(); // Удаление элемента из стека  void OCHISTKA(); // Очистка стека  };  // Основная функция программы  void main() {  setlocale(LC\_ALL, "Rus"); // Установка русской локали для вывода текста  Spisok A; // Создание объекта стека  int el; // Переменная для ввода значений  int t; // Переменная для хранения удалённого элемента  // Построение стека  A.POSTROENIE();  A.VYVOD(); // Вывод стека  // Добавление элемента в стек  cout << "Введите вставляемый элемент: ";  cin >> el;  A.W\_S(el); // Помещение элемента в стек  A.VYVOD(); // Вывод стека  // Удаление элемента из стека  cout << "Удалим элемент из стека.\n";  A.YDALENIE(); // Удаление верхнего элемента  t = A.Set\_Stack(); // Получение удалённого значения  cout << "Из стека было извлечено число... " << t << endl;  A.VYVOD(); // Вывод стека  // Очистка стека  A.OCHISTKA();  cout << "\n";  system("PAUSE"); // Пауза перед завершением программы  }  // Построение стека  void Spisok::POSTROENIE() {  node\* t; // Вспомогательный указатель для построения  int el; // Переменная для ввода значений  cout << "Вводите элементы стека: ";  cin >> el;  while (el != 0) { // Ввод значений до 0  t = new node; // Создание нового узла  (\*t).elem = el; // Запись значения  (\*t).sled = stk; // Связь с предыдущей вершиной стека  stk = t; // Новый узел становится вершиной  cin >> el; // Ввод следующего значения  }  }  // Вывод содержимого стека  void Spisok::VYVOD() {  node\* t = stk; // Начало с вершины стека  cout << "Содержимое стека: ";  while (t != NULL) { // Пока есть узлы  cout << (\*t).elem << " "; // Вывод значения  t = (\*t).sled; // Переход к следующему узлу  }  cout << endl;  }  // Помещение элемента в стек  void Spisok::W\_S(int el) {  node\* q = new node; // Создание нового узла  (\*q).elem = el; // Запись значения  (\*q).sled = stk; // Связь с текущей вершиной стека  stk = q; // Новый узел становится вершиной  }  // Удаление элемента из стека  void Spisok::YDALENIE() {  node\* q; // Указатель для удаляемого узла  if (stk == NULL) { // Проверка на пустой стек  cout << "Стек пуст!\n";  }  else {  klad = (\*stk).elem; // Сохранение значения верхнего элемента  q = stk; // Сохранение указателя на вершину  stk = (\*stk).sled; // Переход к следующему узлу  delete q; // Освобождение памяти  }  }  // Очистка стека  void Spisok::OCHISTKA() {  node\* t, \* q; // Рабочие указатели  t = stk; // Начало с вершины стека  if (t != NULL) { // Если стек не пуст  q = (\*t).sled; // Сохранение следующего узла  while (q != NULL) { // Пока есть узлы  delete t; // Освобождение текущего узла  t = q; // Переход к следующему  q = (\*q).sled; // Сохранение следующего узла  }  delete t; // Освобождение последнего узла  }  stk = NULL; // Сброс указателя на вершину  } |

Вывод:



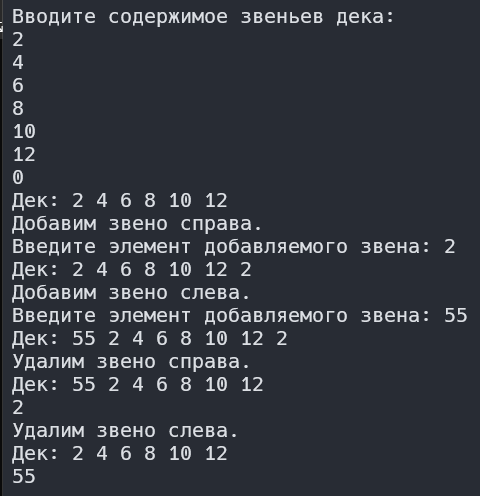
|  |
| --- |
| #include <iostream>  #include <string.h>  using namespace std;  // Максимальный размер стека  const int MAXSIZE = 100;  // Структура для стека символов  struct Stack {  char data[MAXSIZE]; // Массив для хранения символов (скобок)  int size; // Текущий размер стека (количество элементов)  };  // Функция для добавления элемента в стек  void Push(Stack& S, char x) {  if (S.size == MAXSIZE) { // Проверка на переполнение стека  printf("Стек переполнен");  return;  }  S.data[S.size] = x; // Добавление символа в конец массива  S.size++; // Увеличение размера стека  }  // Функция для извлечения элемента из стека  char Pop(Stack& S) {  if (S.size == 0) { // Проверка на пустой стек  printf("Стек пуст");  return char(255); // Возврат специального значения при ошибке  }  S.size--; // Уменьшение размера стека  return S.data[S.size]; // Возврат верхнего элемента  }  // Основная функция программы  void main() {  setlocale(LC\_ALL, "Rus"); // Установка русской локали для вывода текста  char br1[3] = { '(', '[', '{' }; // Массив открывающих скобок  char br2[3] = { ')', ']', '}' }; // Массив закрывающих скобок  char s[80], upper; // s - строка для ввода, upper - извлечённый символ  int i, k, OK; // i, k - счётчики циклов, OK - флаг корректности  Stack S; // Стек для хранения скобок  printf("Введите выражение со скобками> ");  gets\_s(s); // Ввод строки (устаревшая функция, лучше использовать std::cin.getline)  S.size = 0; // Инициализация пустого стека  OK = 1; // Изначально предполагаем, что выражение правильное  // Проверка каждого символа в строке  for (i = 0; OK && (s[i] != '\0'); i++) {  for (k = 0; k < 3; k++) { // Проверка трёх типов скобок  if (s[i] == br1[k]) { // Если найдена открывающая скобка  Push(S, s[i]); // Поместить её в стек  break; // Прервать цикл по типам скобок  }  if (s[i] == br2[k]) { // Если найдена закрывающая скобка  upper = Pop(S); // Извлечь верхнюю скобку из стека  if (upper != br1[k]) { // Проверка соответствия типов скобок  OK = 0; // Если не соответствует, выражение неверное  }  break; // Прервать цикл по типам скобок  }  }  }  // Проверка результата  if (OK && (S.size == 0)) { // Если все скобки закрыты и стек пуст  printf("\nВыpажение пpавильное\n");  }  else { // Если есть незакрытые скобки или ошибка  printf("\nВыpажение непpавильное \n");  }  system("PAUSE"); // Пауза перед завершением программы  } |

|  |  |
| --- | --- |
|  |  |

# 5. Дек

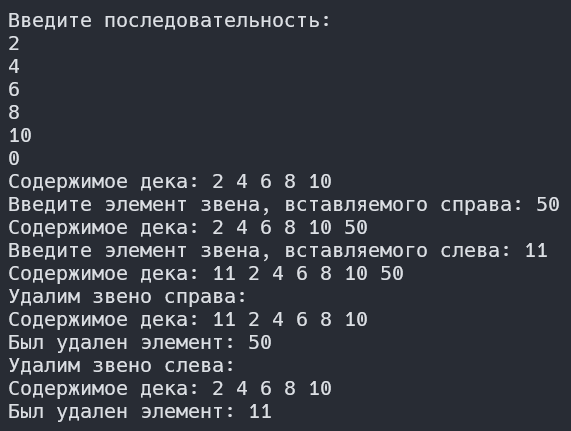
|  |
| --- |
| #include <iostream>  using namespace std;  // Описание структуры узла дека  struct node {  int elem; // Информационное поле узла (значение)  node\* sled; // Указатель на следующий узел  };  // Класс для работы с деком  class Spisok {  private:  node\* ld; // Указатель на левый (первый) конец дека  node\* rd; // Указатель на правый (последний) конец дека  int el\_left; // Переменная для хранения значения удалённого слева элемента  int el\_right; // Переменная для хранения значения удалённого справа элемента  public:  // Методы для работы с деком  void POSTROENIE(); // Построение дека  void VYVOD(); // Вывод содержимого дека  void VSTAV1(int); // Добавление элемента справа  void VSTAV2(int); // Добавление элемента слева  int SetElLeft() { return el\_left; } // Получение значения удалённого слева  int SetElRight() { return el\_right; } // Получение значения удалённого справа  void YDALE1(); // Удаление элемента справа  void YDALE2(); // Удаление элемента слева  void OCHISTKA(); // Очистка дека  };  // Основная функция программы  void main() {  setlocale(LC\_ALL, "Rus"); // Установка русской локали для вывода текста  Spisok A; // Создание объекта дека  int el; // Переменная для ввода значений  // Построение дека  A.POSTROENIE();  A.VYVOD(); // Вывод дека  // Добавление элемента справа  cout << "Добавим звено справа.\n";  cout << "Введите элемент добавляемого звена: ";  cin >> el;  A.VSTAV1(el); // Добавление элемента  A.VYVOD(); // Вывод дека  // Добавление элемента слева  cout << "Добавим звено слева.\n";  cout << "Введите элемент добавляемого звена: ";  cin >> el;  A.VSTAV2(el); // Добавление элемента  A.VYVOD(); // Вывод дека  // Удаление элемента справа  cout << "Удалим звено справа.\n";  A.YDALE1(); // Удаление элемента  A.VYVOD(); // Вывод дека  cout << A.SetElRight() << endl; // Вывод удалённого значения  // Удаление элемента слева  cout << "Удалим звено слева.\n";  A.YDALE2(); // Удаление элемента  A.VYVOD(); // Вывод дека  cout << A.SetElLeft() << endl; // Вывод удалённого значения  // Очистка дека  A.OCHISTKA();  cout << "\n";  system("PAUSE"); // Пауза перед завершением программы  }  // Построение дека  void Spisok::POSTROENIE() {  node\* k; // Вспомогательный указатель для построения  int el; // Переменная для ввода значений  cout << "Вводите содержимое звеньев дека: \n";  cin >> el;  if (el != 0) { // Если введён ненулевой элемент  k = new node; // Создание первого узла  (\*k).elem = el; // Запись значения  (\*k).sled = NULL; // Указатель на следующий узел пуст  ld = k; // Установка левого конца  rd = k; // Установка правого конца  cin >> el;  while (el != 0) { // Добавление остальных элементов справа  VSTAV1(el); // Использование метода добавления справа  cin >> el; // Ввод следующего значения  }  }  else { // Если введён 0, дек пуст  rd = NULL;  ld = NULL;  }  }  // Вывод содержимого дека  void Spisok::VYVOD() {  node\* k = ld; // Начало с левого конца  cout << "Дек: ";  while (k != NULL) { // Пока есть узлы  cout << (\*k).elem << " "; // Вывод значения  k = (\*k).sled; // Переход к следующему узлу  }  cout << endl;  }  // Добавление элемента справа  void Spisok::VSTAV1(int el) {  node\* k = new node; // Создание нового узла  (\*k).elem = el; // Запись значения  (\*k).sled = NULL; // Указатель на следующий узел пуст  if (rd != NULL) { // Если дек не пуст  (\*rd).sled = k; // Связь текущего правого конца с новым узлом  rd = k; // Обновление правого конца  }  else { // Если дек пуст  rd = k; // Новый узел становится правым концом  ld = k; // Новый узел становится левым концом  }  }  // Добавление элемента слева  void Spisok::VSTAV2(int el) {  node\* k = new node; // Создание нового узла  (\*k).elem = el; // Запись значения  (\*k).sled = ld; // Связь с текущим левым концом  if (ld != NULL) { // Если дек не пуст  ld = k; // Новый узел становится левым концом  }  else { // Если дек пуст  ld = k; // Новый узел становится левым концом  rd = k; // Новый узел становится правым концом  }  }  // Удаление элемента справа  void Spisok::YDALE1() {  node\* z, \* k; // Вспомогательные указатели  if (rd == ld) { // Если в деке один узел  el\_right = (\*rd).elem; // Сохранение значения  delete rd; // Освобождение памяти  ld = rd = NULL; // Дек становится пустым  cout << "Дек пуст!\n";  }  else { // Если в деке несколько узлов  z = ld; // Начало с левого конца  k = (\*ld).sled; // Второй узел  while (k != rd) { // Поиск предпоследнего узла  z = k;  k = (\*k).sled;  }  el\_right = (\*rd).elem; // Сохранение значения правого конца  (\*z).sled = NULL; // Обнуление связи предпоследнего узла  delete rd; // Освобождение памяти  rd = z; // Обновление правого конца  }  }  // Удаление элемента слева  void Spisok::YDALE2() {  node\* q; // Указатель для удаляемого узла  if (ld != NULL) { // Если дек не пуст  el\_left = (\*ld).elem; // Сохранение значения левого конца  q = ld; // Сохранение указателя на левый конец  ld = (\*ld).sled; // Переход к следующему узлу  delete q; // Освобождение памяти  }  else { // Если дек пуст  cout << "Дек пуст!\n";  }  }  // Очистка дека  void Spisok::OCHISTKA() {  node\* k, \* q; // Рабочие указатели  k = ld; // Начало с левого конца  if (k != NULL) { // Если дек не пуст  q = (\*k).sled; // Сохранение следующего узла  while (q != NULL) { // Пока есть узлы  delete k; // Освобождение текущего узла  k = q; // Переход к следующему  q = (\*k).sled; // Сохранение следующего узла  }  delete k; // Освобождение последнего узла  }  ld = rd = NULL; // Сброс указателей  } |

Вывод:



|  |
| --- |
| #include <iostream>  using namespace std;  // Описание структуры узла двусвязного дека  struct node {  int elem; // Информационное поле узла (значение)  node\* sled; // Указатель на следующий узел  node\* pred; // Указатель на предыдущий узел  };  // Класс для работы с двусвязным деком  class Spisok {  private:  node\* nd; // Указатель на начало дека (левый конец)  node\* kd; // Указатель на конец дека (правый конец)  int klad; // Переменная для хранения значения удалённого элемента  public:  // Методы для работы с деком  void BuiltDeck(); // Построение дека  void VyvodDeck(); // Вывод содержимого дека  void InsLeft(int); // Добавление элемента слева  void InsRight(int); // Добавление элемента справа  void DelLeft(); // Удаление элемента слева  void DelRight(); // Удаление элемента справа  int Get\_Klad() { return klad; } // Получение значения удалённого элемента  void Ochistka(); // Очистка дека  };  // Основная функция программы  void main() {  setlocale(LC\_ALL, "Rus"); // Установка русской локали для вывода текста  Spisok A; // Создание объекта дека  int el; // Переменная для ввода значений  // Построение дека  A.BuiltDeck();  A.VyvodDeck(); // Вывод дека  // Добавление элемента справа  cout << "Введите элемент звена, вставляемого справа: ";  cin >> el;  A.InsRight(el); // Добавление элемента  A.VyvodDeck(); // Вывод дека  // Добавление элемента слева  cout << "Введите элемент звена, вставляемого слева: ";  cin >> el;  A.InsLeft(el); // Добавление элемента  A.VyvodDeck(); // Вывод дека  // Удаление элемента справа  cout << "Удалим звено справа: \n";  A.DelRight(); // Удаление элемента  A.VyvodDeck(); // Вывод дека  cout << "Был удален элемент: " << A.Get\_Klad() << endl; // Вывод удалённого значения  // Удаление элемента слева  cout << "Удалим звено слева: \n";  A.DelLeft(); // Удаление элемента  A.VyvodDeck(); // Вывод дека  cout << "Был удален элемент: " << A.Get\_Klad() << endl; // Вывод удалённого значения  // Очистка дека  A.Ochistka();  cout << "\n";  system("PAUSE"); // Пауза перед завершением программы  }  // Построение двусвязного дека  void Spisok::BuiltDeck() {  node\* q; // Вспомогательный указатель для удаления заглавного узла  node\* z; // Вспомогательный указатель для построения  int el; // Переменная для ввода значений  nd = new node; // Создание заглавного узла  z = nd; // Начало с заглавного узла  (\*nd).pred = (\*nd).sled = NULL; // Инициализация указателей  cout << "Введите последовательность: \n";  cin >> el;  while (el != 0) { // Ввод значений до 0  (\*z).sled = new node; // Создание нового узла  (\*((\*z).sled)).pred = z; // Связь нового узла с предыдущим  z = (\*z).sled; // Переход к новому узлу  (\*z).sled = NULL; // Указатель на следующий узел пуст  (\*z).elem = el; // Запись значения  cin >> el; // Ввод следующего значения  }  if ((\*nd).sled != NULL) { // Если дек не пуст  q = nd; // Сохранение заглавного узла  nd = (\*nd).sled; // Начало дека с первого элемента  (\*nd).pred = NULL; // Обнуление указателя на предыдущий  kd = z; // Установка конца дека  delete q; // Удаление заглавного узла  }  else { // Если дек пуст  delete nd; // Удаление заглавного узла  nd = kd = NULL; // Сброс указателей  }  }  // Вывод содержимого дека  void Spisok::VyvodDeck() {  node\* z = nd; // Начало с левого конца  cout << "Содержимое дека: ";  if (z != NULL) { // Если дек не пуст  while (z != NULL) { // Пока есть узлы  cout << (\*z).elem << " "; // Вывод значения  z = (\*z).sled; // Переход к следующему узлу  }  }  else { // Если дек пуст  cout << "он пуст!\n";  }  cout << endl;  }  // Добавление элемента слева  void Spisok::InsLeft(int el) {  node\* q = new node; // Создание нового узла  (\*q).elem = el; // Запись значения  if (nd == NULL) { // Если дек пуст  nd = q; // Новый узел становится началом  (\*q).sled = (\*q).pred = NULL; // Указатели пусты  kd = q; // Новый узел становится концом  }  else { // Если дек не пуст  (\*q).sled = nd; // Связь с текущим началом  (\*q).pred = NULL; // Указатель на предыдущий пуст  (\*nd).pred = q; // Связь текущего начала с новым узлом  nd = q; // Обновление начала дека  }  }  // Добавление элемента справа  void Spisok::InsRight(int el) {  node\* q = new node; // Создание нового узла  (\*q).elem = el; // Запись значения  if (kd == NULL) { // Если дек пуст  nd = q; // Новый узел становится началом  (\*q).sled = (\*q).pred = NULL; // Указатели пусты  kd = q; // Новый узел становится концом  }  else { // Если дек не пуст  (\*q).sled = NULL; // Указатель на следующий пуст  (\*q).pred = kd; // Связь с текущим концом  (\*kd).sled = q; // Связь текущего конца с новым узлом  kd = q; // Обновление конца дека  }  }  // Удаление элемента слева  void Spisok::DelLeft() {  node\* q; // Указатель для удаляемого узла  if ((\*nd).sled != NULL) { // Если в деке больше одного элемента  q = nd; // Сохранение указателя на начало  klad = (\*q).elem; // Сохранение значения  nd = (\*nd).sled; // Переход к следующему узлу  (\*nd).pred = NULL; // Обнуление указателя на предыдущий  delete q; // Освобождение памяти  }  else { // Если в деке один элемент  q = nd; // Сохранение указателя на начало  klad = (\*q).elem; // Сохранение значения  nd = kd = NULL; // Дек становится пустым  delete q; // Освобождение памяти  cout << "Дек пуст!\n";  }  }  // Удаление элемента справа  void Spisok::DelRight() {  node\* q; // Указатель для удаляемого узла  if ((\*kd).pred != NULL) { // Если в деке больше одного элемента  q = kd; // Сохранение указателя на конец  klad = (\*q).elem; // Сохранение значения  kd = (\*kd).pred; // Переход к предыдущему узлу  (\*kd).sled = NULL; // Обнуление указателя на следующий  delete q; // Освобождение памяти  }  else { // Если в деке один элемент  q = kd; // Сохранение указателя на конец  klad = (\*q).elem; // Сохранение значения  nd = kd = NULL; // Дек становится пустым  delete q; // Освобождение памяти  cout << "Дек пуст!\n";  }  }  // Очистка дека  void Spisok::Ochistka() {  node\* q, \* q1; // Рабочие указатели  q = nd; // Начало с левого конца  if (q != NULL) { // Если дек не пуст  q1 = (\*q).sled; // Сохранение следующего узла  while (q1 != NULL) { // Пока есть узлы  delete q; // Освобождение текущего узла  q = q1; // Переход к следующему  q1 = (\*q).sled; // Сохранение следующего узла  }  delete q; // Освобождение последнего узла  }  nd = kd = NULL; // Сброс указателей  } |

Вывод:



# 6. Очередь

|  |
| --- |
| #include <iostream>  using namespace std;  // Описание структуры узла очереди  struct node {  int elem; // Информационное поле узла (значение)  node\* sled; // Указатель на следующий узел  };  // Класс для работы с очередью  class Spisok {  private:  node\* no; // Указатель на начало очереди (первый элемент)  node\* ko; // Указатель на конец очереди (последний элемент)  int klad; // Переменная для хранения значения удалённого элемента  public:  // Конструктор: инициализирует пустую очередь  Spisok() { no = ko = NULL; }  // Методы для работы с очередью  void POSTROENIE(); // Построение очереди  void VYVOD(); // Вывод содержимого очереди  void DOBAVLENIE(int); // Добавление элемента в конец очереди  int Set\_Udal() { return klad; } // Получение значения удалённого элемента  void YDALENIE(); // Удаление элемента из начала очереди  void OCHISTKA(); // Очистка очереди  };  // Основная функция программы  void main() {  setlocale(LC\_ALL, "Rus"); // Установка русской локали для вывода текста  Spisok A; // Создание объекта очереди  int el; // Переменная для ввода значений  // Построение очереди  A.POSTROENIE();  A.VYVOD(); // Вывод очереди  // Добавление элемента в очередь  cout << "Введите добавляемый элемент: ";  cin >> el;  A.DOBAVLENIE(el); // Добавление элемента в конец  A.VYVOD(); // Вывод очереди  // Удаление элемента из очереди  cout << "Удалим элемент из очереди.\n";  A.YDALENIE(); // Удаление элемента из начала  A.VYVOD(); // Вывод очереди  el = A.Set\_Udal(); // Получение удалённого значения  cout << "Информационное поле удаленного звена: " << el << endl;  // Очистка очереди  A.OCHISTKA();  cout << "\n";  system("PAUSE"); // Пауза перед завершением программы  }  // Построение очереди  void Spisok::POSTROENIE() {  node\* r; // Вспомогательный указатель для построения  int el; // Переменная для ввода значений  cout << "Вводите элементы очереди:\n";  cin >> el;  if (el != 0) { // Если введён ненулевой элемент  r = new node; // Создание первого узла  (\*r).elem = el; // Запись значения  (\*r).sled = NULL; // Указатель на следующий узел пуст  no = r; // Установка начала очереди  ko = r; // Установка конца очереди  cin >> el;  while (el != 0) { // Добавление остальных элементов  r = new node; // Создание нового узла  (\*r).elem = el; // Запись значения  (\*r).sled = NULL; // Указатель на следующий узел пуст  (\*ko).sled = r; // Связь текущего конца с новым узлом  ko = r; // Обновление конца очереди  cin >> el; // Ввод следующего значения  }  }  else { // Если введён 0, очередь пуста  r = NULL;  no = r;  ko = r;  }  }  // Вывод содержимого очереди  void Spisok::VYVOD() {  node\* r = no; // Начало с первого элемента  cout << "Очередь: ";  while (r != NULL) { // Пока есть узлы  cout << (\*r).elem << " "; // Вывод значения  r = (\*r).sled; // Переход к следующему узлу  }  cout << endl;  }  // Добавление элемента в конец очереди  void Spisok::DOBAVLENIE(int el) {  node\* r = new node; // Создание нового узла  (\*r).elem = el; // Запись значения  (\*r).sled = NULL; // Указатель на следующий узел пуст  if (no != NULL) { // Если очередь не пуста  (\*ko).sled = r; // Связь текущего конца с новым узлом  ko = r; // Обновление конца очереди  }  else { // Если очередь пуста  no = r; // Новый узел становится началом  ko = r; // Новый узел становится концом  }  }  // Удаление элемента из начала очереди  void Spisok::YDALENIE() {  node\* q; // Указатель для удаляемого узла  if (no == NULL) { // Проверка на пустую очередь  cout << "Удалить нельзя, так как очередь пуста!\n";  }  else {  klad = (\*no).elem; // Сохранение значения первого элемента  q = no; // Сохранение указателя на начало  no = (\*no).sled; // Переход к следующему узлу  delete q; // Освобождение памяти  }  }  // Очистка очереди  void Spisok::OCHISTKA() {  node\* q; // Вспомогательный указатель  q = no; // Начало с первого элемента  if (no != NULL) { // Если очередь не пуста  while (no != ko) { // Пока не достигнут конец  no = (\*q).sled; // Переход к следующему узлу  delete q; // Освобождение текущего узла  q = no; // Обновление указателя  }  delete no; // Освобождение последнего узла  no = ko = NULL; // Сброс указателей  }  } |

Вывод:

