Федеральное государственное автономное образовательное учреждение

высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

КАФЕДРА СИСТЕМНОГО АНАЛИЗА И ТЕЛЕКОММУНИКАЦИИ

**ПРОВЕРИЛ ВЫПОЛНИЛА**

Старший преподаватель кафедры САиТ Студентка группы КТбо1-2

Лапшин В.С. Бровкина В.Р.

«17» апреля 2023 г. «17» апреля 2023 г.

**ЛАБОРАТОРНАЯ РАБОТА №3**

По дисциплине

«Основы алгоритмизации и программирования»

Вариант 6

На тему

«Динамическое распределение памяти, списки»

**Таганрог 2023**

**Цель**: освоение студентами способов представления и обработки данных в виде списков.

**Задачи**: Написать программы для работы с односвязным и двусвязным списками в соответствии с выданным вариантом задания. Реализация связанного списка в соответствии с выданным вариантом.

**Описание варианта:** написать программы для работы с односвязным и двусвязным списками в соответствии с выданным вариантом задания.

Предусмотреть в программах следующие функции:

1.Включить новый элемент в конец списка.

2.Включить новый элемент на заданное пользователем место списка.

3.Включить новый элемент после элемента с заданной информационной частью.

4.Включить новый элемент перед элементом с заданной информационной частью.

5.Включить новый элемент в середину списка.

6.Исключить элемент из середины списка.

7.Исключить элемент с заданной информационной частью.

8.Исключить элемент из конца списка.

9.Исключить элемент из заданного пользователем места списка.

10.Исключить элемент из головы списка.

Программы должны уметь обрабатывать структуру данных в соответствии с вариантом заданий.

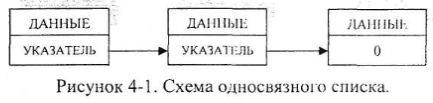
Вариант 6. Домашний бюджет. Список покупок.

**Ход работы**:

**Описание основных понятий:**

Связанные списки. В отличие от стека или очереди связанный список допускает гибкие способы доступа к его элементам, поскольку каждый элемент имеет ссылку на следующий элемент. Кроме того, операция извлечения данных из списка не приводит к удалению элемента из списка. Если элемент необходимо удалить, то для этой цели необходимо дополнительно использовать специальную операцию удаления.

Связанные списки могут быть односвязными и двусвязными. Элемент односвязного списка содержит одну ссылку на следующий элемент списка. В двусвязном списке каждый элемент содержит ссылки на следующий и предыдущий элементы. Выбор типа применяемого списка лежит на программисте и зависит от конкретной решаемой задачи.

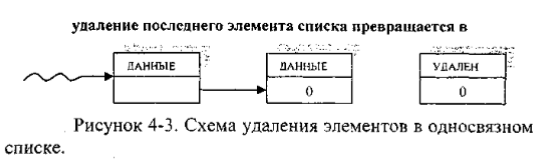
**

При вставке элемента в односвязный список может возникнуть одна из трех ситуаций, когда элемент: становится первым, вставляется между двумя другими, становится последним.



Теперь рассмотрим удаление элемента из односвязного списка. Так же, как и при вставке, возможны три случая: удаление первого элемента, удаление элемента в середине, удаление последнего элемента.



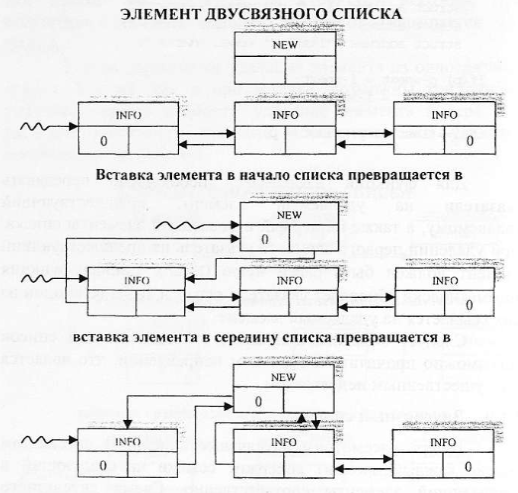


Следует обратить внимание, что односвязный список невозможно прочитать в обратном направлении, что является его существенным недостатком.

Теперь рассмотрим двусвязные списки. В двусвязном списке каждый элемент содержит ссылки на следующий и предыдущий элементы соответственно

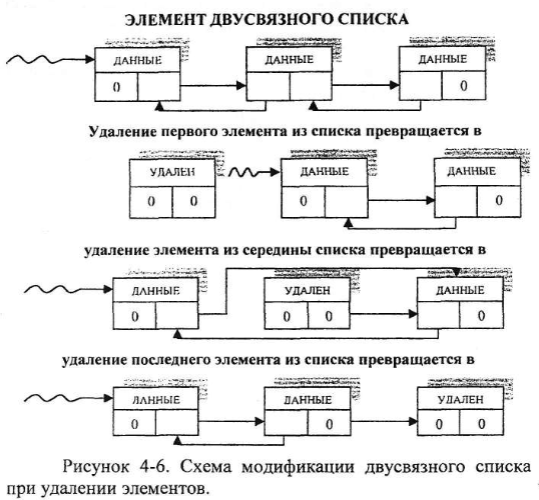


Наличие двух ссылок у элемента позволяет реализовать перемещение по списку в обоих направлениях, что является важным преимуществом. Кроме того, при сбоях двусвязный список имеет больше шансов на правильное восстановление. Ниже на рисунке 4-5 показаны схематично три случая вставки элементов в двусвязный список.

****

****

Подобно односвязным спискам, двусвязные списки можно создавать с помощью функции, которая помещает элементы в определенные позиции, а не только в конец списка.

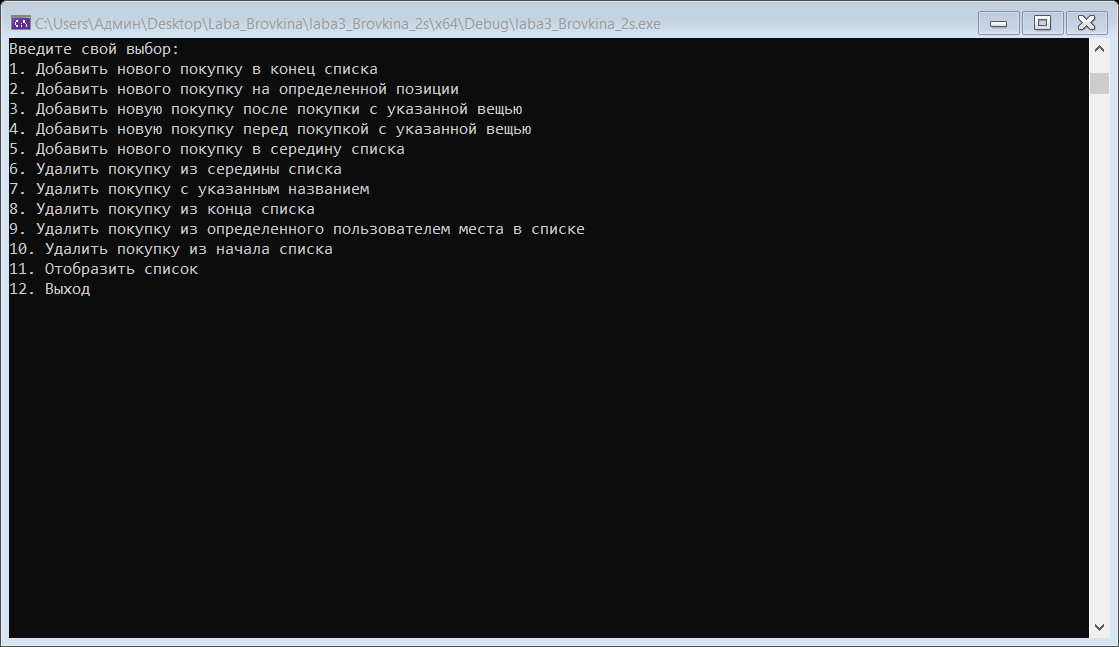
****

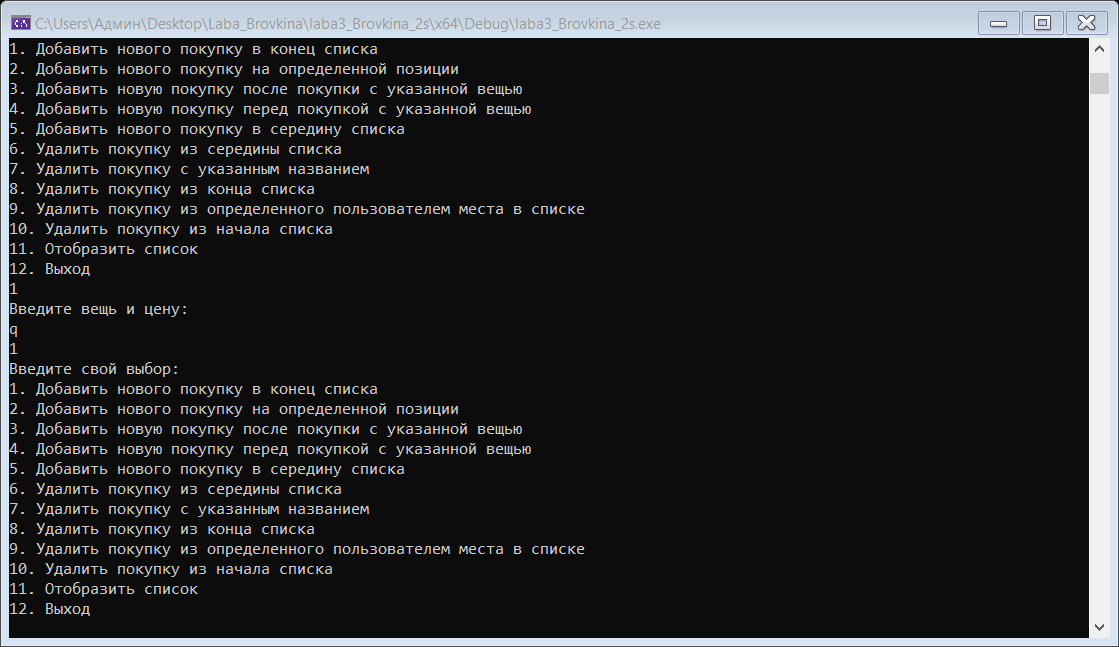
**Реализация .** *(Код программы на языке с++)*

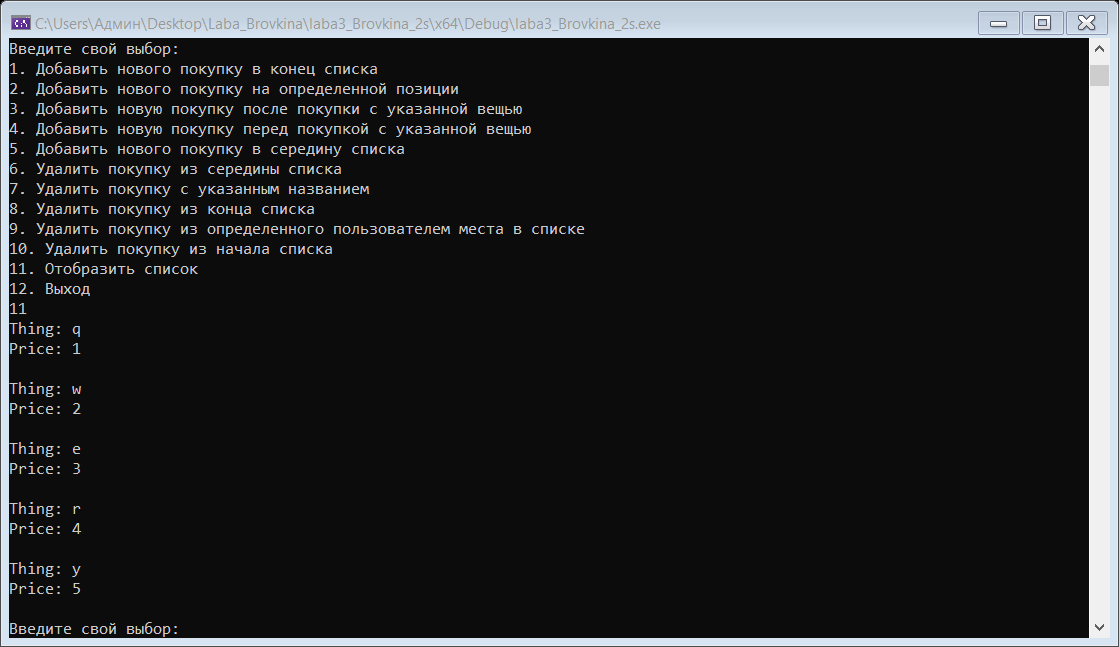
*Вводные слова перед реализацией*

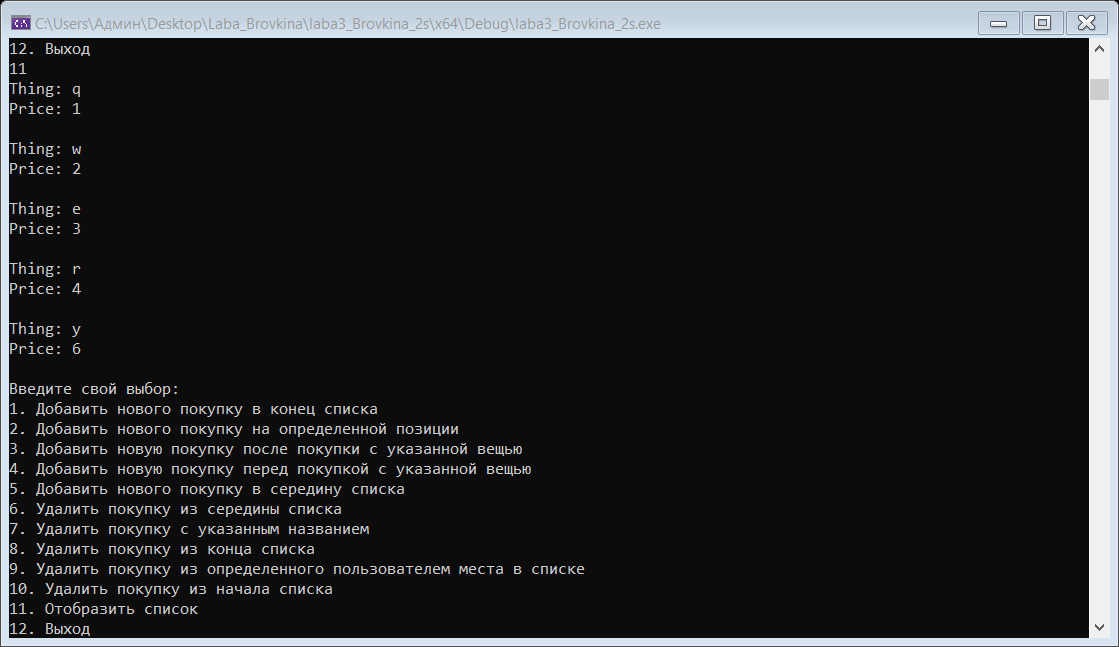
|  |
| --- |
| #include <iostream>  #include <fstream>  #include <string>  using namespace std;  struct Buy {  string thing;  string price;  Buy \* next;  Buy \* prev;  };  Buy \* head = nullptr;  Buy \* tail = nullptr;  void display(Buy \* p) {  cout << "Thing: " << p->thing << endl;  cout << "Price: " << p->price << endl;  }  void displayAll() {  if (head == nullptr) {  cout << "Список покупок пуст" << endl;  return;  }  Buy \* curr = head;  while (curr != nullptr) {  display(curr);  cout << endl;  curr = curr->next;  }  }  void insertAtEnd(string thing, string price) {  Buy \* p = new Buy;  p->thing = thing;  p->price = price;  p->next = nullptr;  p->prev = tail;  if (tail != nullptr) {  tail->next = p;  }  tail = p;  if (head == nullptr) {  head = p;  }  }  void insertAtPos(int pos, string thing, string price) {  Buy \* p = new Buy;  p->thing = thing;  p->price = price;  p->next = nullptr;  p->prev = nullptr;  if (pos == 1) {  p->next = head;  head->prev = p;  head = p;  return;  }  Buy \* curr = head;  for (int i = 1; i < pos - 1 && curr != nullptr; i++) {  curr = curr->next;  }  if (curr == nullptr) {  cout << "Неверная позиция" << endl;  return;  }  p->prev = curr;  p->next = curr->next;  if (curr->next != nullptr) {  curr->next->prev = p;  }  curr->next = p;  }  void insertAfter(string thing, string newThing, string newPrice) {  Buy \* curr = head;  while (curr != nullptr && curr->thing != thing) {  curr = curr->next;  }  if (curr == nullptr) {  cout << "Предмет не найден" << endl;  return;  }  Buy \* p = new Buy;  p->thing = newThing;  p->price = newPrice;  p->next = curr->next;  p->prev = curr;  if (curr->next != nullptr) {  curr->next->prev = p;  }  curr->next = p;  if (tail == curr) {  tail = p;  }  }  void insertBefore(string thing, string newThing, string newPrice) {  Buy \* curr = head;  while (curr != nullptr && curr->thing != thing) {  curr = curr->next;  }  if (curr == nullptr) {  cout << "Предмет не найден" << endl;  return;  }  Buy \* p = new Buy;  p->thing = newThing;  p->price = newPrice;  p->prev = curr->prev;  p->next = curr;  if (curr->prev != nullptr) {  curr->prev->next = p;  }  curr->prev = p;  if (head == curr) {  head = p;  }  }  void insertInMiddle(string thing, string price) {  Buy \* slow = head;  Buy \* fast = head;  while (fast != nullptr && fast->next != nullptr) {  slow = slow->next;  fast = fast->next->next;  }  Buy \* p = new Buy;  p->thing = thing;  p->price = price;  p->prev = slow;  p->next = slow->next;  if (slow->next != nullptr) {  slow->next->prev = p;  }  slow->next = p;  if (tail == slow) {  tail = p;  }  }  void deleteFromMiddle() {  if (head == nullptr) {  cout << "Список покупок пуст" << endl;  return;  }  Buy \* slow = head;  Buy \* fast = head;  while (fast != nullptr && fast->next != nullptr) {  slow = slow->next;  fast = fast->next->next;  }  if (slow == head) {  cout << slow << endl;  cout << head << endl;  head = head->next;  if (head != nullptr) {  head->prev = nullptr;  }  else {  tail = nullptr;  }  delete slow;  }  else if (slow == tail) {  cout << slow << endl;  cout << tail << endl;  tail = tail->prev;  if (tail != nullptr) {  tail->next = nullptr;  }  else {  head = nullptr;  }  delete slow;  }  else {  slow->prev->next = slow->next;  slow->next->prev = slow->prev;  delete slow;  }  }  void deleteByThing(string thing) {  Buy \* curr = head;  while (curr != nullptr && curr->thing != thing) {  curr = curr->next;  }    if (curr == nullptr) {  cout << "Предмет не найден" << endl;  return;  }  if (curr == head) {  head = head->next;  if (head != nullptr) {  head->prev = nullptr;  }  else {  tail = nullptr;  }  delete curr;  }  else if (curr == tail) {  tail = tail->prev;  if (tail != nullptr) {  tail->next = nullptr;  }  else {  head = nullptr;  }  delete curr;  }  else {  curr->prev->next = curr->next;  curr->next->prev = curr->prev;  delete curr;  }  }  void deleteFromEnd() {  if (head == nullptr) {  cout << "Список покупок пуст" << endl;  return;  }  if (tail == head) {  delete head;  head = nullptr;  tail = nullptr;  return;  }  Buy \* curr = tail;  tail = tail->prev;  tail->next = nullptr;  delete curr;  }  void deleteFromPos(int pos) {  if (head == nullptr) {  cout << "Список покупок пуст" << endl;  return;  }  if (pos == 1) {  return;  }  Person\* curr = head;  int i = 1;  while (i < pos && curr != nullptr) {  curr = curr->next;  i++;  }  if (curr == nullptr) {  cout << "Позиция не найдена" << endl;  return;  }  if (curr == tail) {  tail = tail->prev;  if (tail != nullptr) {  tail->next = nullptr;  }  else {  head = nullptr;  }  delete curr;  }  else {  curr->prev->next = curr->next;  curr->next->prev = curr->prev;  delete curr;  }  }  void deleteFromBeginning() {  if (head == nullptr) {  cout << "Список покупок пуст" << endl;  return;  }  if (head == tail) {  delete head;  head = nullptr;  tail = nullptr;  return;  }  Buy \* curr = head;  head = head->next;  head->prev = nullptr;  delete curr;  }  void display() {  Buy \* curr = head;  while (curr != nullptr) {  cout << "Thing: " << curr->thing << endl;  cout << "Price: " << curr->price << endl;  cout << endl;  curr = curr->next;  }  }  void saveToFile() {  ofstream outFile("file.txt");  Buy \* curr = head;  while (curr != nullptr) {  outFile << curr->thing << endl;  outFile << curr->price << endl;  curr = curr->next;  }  outFile.close();  }  void loadFromFile() {  ifstream inFile("file.txt");  if (!inFile) {  cout << "Невозможно открыть файл" << endl;  return;  }  string thing, price;  while (getline(inFile, thing)) {  getline(inFile, price);  insertAtEnd(thing, price);  }  inFile.close();  }  int main() {  setlocale(LC\_ALL, "RUS");  loadFromFile();  string thing, price;  string afterThing, beforeThing;  while (true) {  int choice;  cout << "Введите свой выбор:" << endl;  cout << "1. Добавить нового покупку в конец списка" << endl;  cout << "2. Добавить нового покупку на определенной позиции" << endl;  cout << "3. Добавить новую покупку после покупки с указанной вещью" << endl;  cout << "4. Добавить новую покупку перед покупкой с указанной вещью" << endl;  cout << "5. Добавить нового покупку в середину списка" << endl;  cout << "6. Удалить покупку из середины списка" << endl;  cout << "7. Удалить покупку с указанным названием" << endl;  cout << "8. Удалить покупку из конца списка" << endl;  cout << "9. Удалить покупку из определенного пользователем места в списке" << endl;  cout << "10. Удалить покупку из начала списка" << endl;  cout << "11. Отобразить список" << endl;  cout << "12. Выход" << endl;  cin >> choice;  switch (choice) {  case 1:  cout << "Введите вещь и цену:\n";  cin >> thing;  cin >> price;  insertAtEnd(thing, price);  saveToFile();  break;  case 2:  int pos2;  cout << "Введите позицию:\n";  cin >> pos2;  cout << "Введите новую цену вещи: ";  cin >> thing;  cin >> price;  insertAtPos(pos2, thing, price);  saveToFile();  break;  case 3:  cout << "Введите вещь:\n";  cin >> thing;  cout << "Введите вещь, после которой нужно вставить: ";  cin >> afterThing;  cin >> price;  insertAfter(thing, afterThing, price);  saveToFile();  break;  case 4:  cout << "Введите вещь:\n";  cin >> thing;  cout << "Введите вещь, до которой нужно вставить: ";  cin >> beforeThing;  cin >> price;  insertBefore(thing, beforeThing, price);  saveToFile();  break;  case 5:  cout << "Введите вещь и цену:\n";  cin >> thing;  cin >> price;  insertInMiddle(thing, price);  saveToFile();  break;  case 6:  cout << "Введите позицию для удаления покупки: ";  deleteFromMiddle();  saveToFile();  break;  case 7:  cout << "Введите название для удаления покупки: ";  cin >> thing;  deleteByThing(thing);  saveToFile();  break;  case 8:  deleteFromEnd();  saveToFile();  break;  case 9:  cout << "Введите позицию для удаления покупки: ";  int pos1;  cin >> pos1;  deleteFromPos(pos1);  saveToFile();  break;  case 10:  deleteFromBeginning();  saveToFile();  break;  case 11:  display();  break;  case 12:  return 0;  default:  cout << "Неверный выбор" << endl;  }  }  return 0;  } |

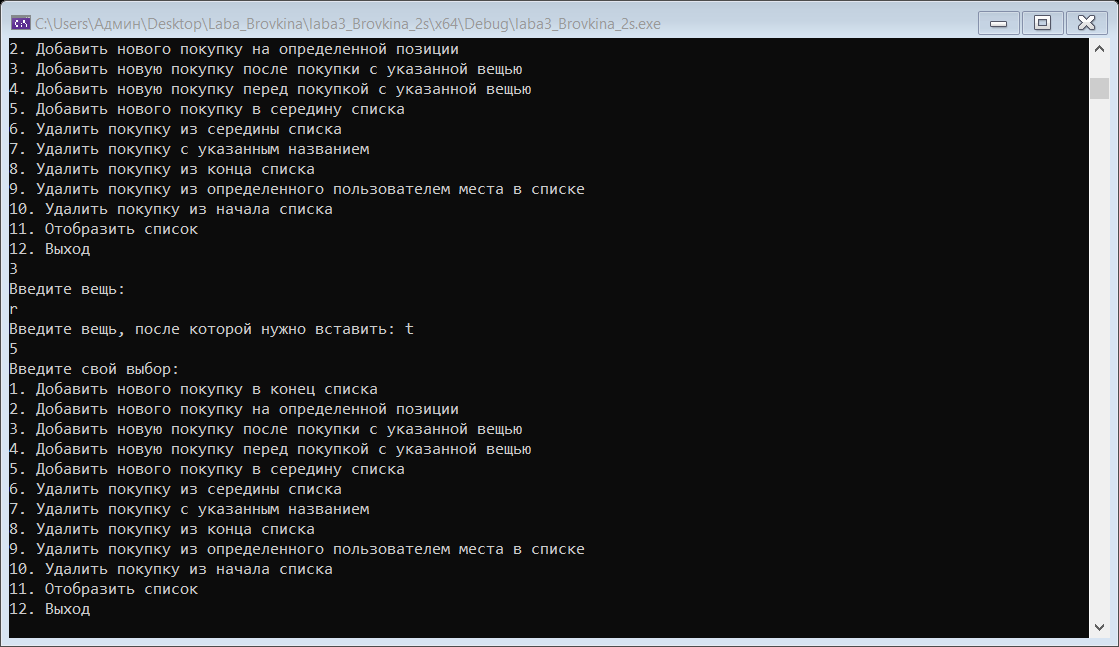
**Результат работы программы.** *(Скриншот)*

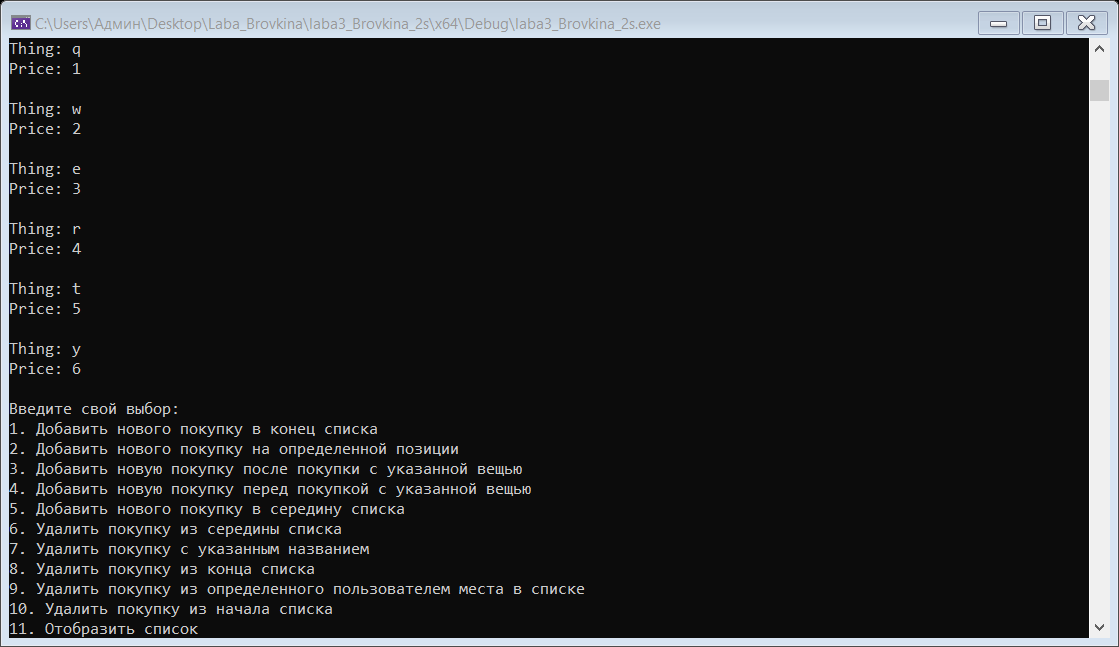


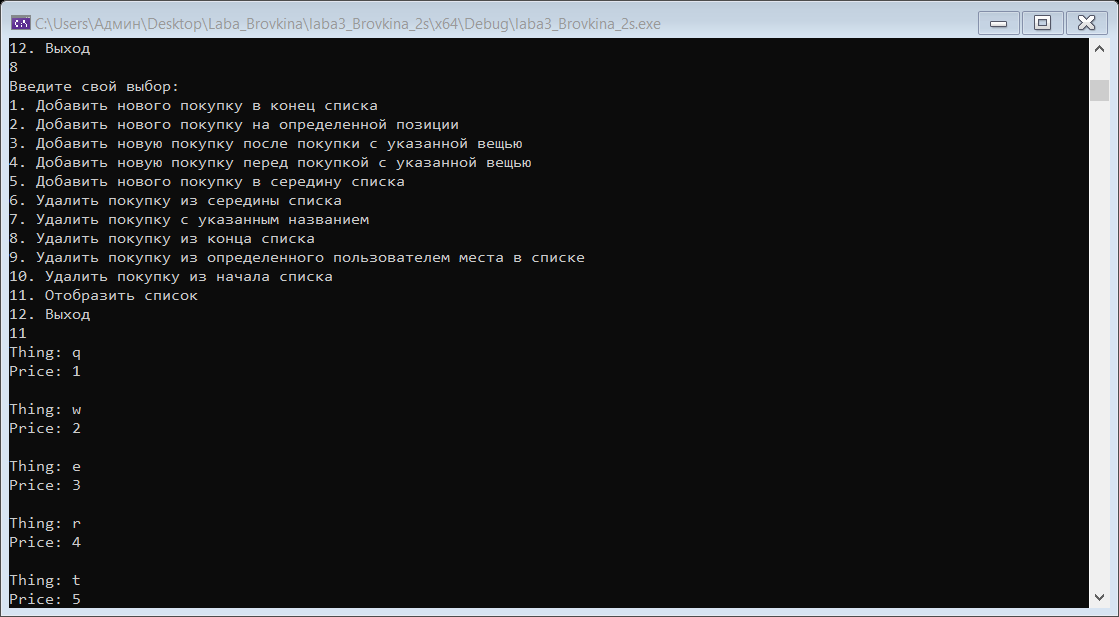


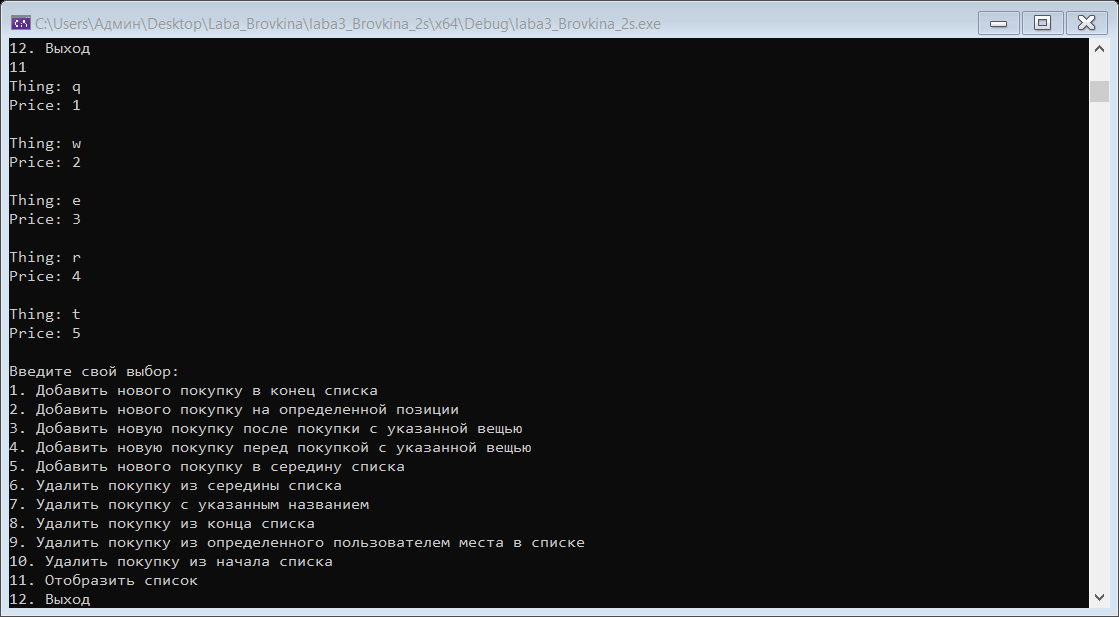












**Вывод:** в этой лабораторной работе была реализована работа структуры данных в виде одно- и двусвязных списков на C++, а также реализованы различные функции для добавления, удаления и изменения узлов в связанных списках.