ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки
09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 3

Дисциплина: «Основы современных алгоритмов»

Тема: «Разработка программы односвязного и двусвязного списков»

Выполнил: студент группы 211-723

Сергеев Станислав Олегович

Дата, подпись_____ _____
(Дата) (Подпись)

Проверил:_____ _____
(Фамилия И.О., степень, звание) **(Оценка)**

Дата, подпись_____ _____
(Дата) (Подпись)

Замечания: _____

_____

_____

Москва

2022

Разработка программы односвязного и двусвязного списков

Цель:

Получить практические навыки в создании односвязных и двухсвязных списков.

Постановка задачи:

Используя результаты предшествующей лабораторной работы, в соответствии с заданной структурной единицей хранения информации, разработать алгоритмы:

Создание списка

Добавление элемента в конец списка

Добавление элемента в начало списка

Удаление конечного элемента списка

Удаление начального элемента списка

Поиск элемента по заданному значению поля структуры
Добавление элемента после найденного

Удаление найденного элемента.

Перечисленные алгоритмы разработать для односвязного и двухсвязного списков.

Для этого в заданных структурах данных предусмотреть дополнительные поля, содержащие необходимые указатели.

Односвязный список

```
#include <iostream>
#include <string>
#include <cmath>
using namespace std;

struct Data
{
    char surname[20];
    char name[15];
    char lastname[25];
    int count;
    int day;
    int month;
    int year;
    int duration;
```

```
};

struct Node
{
    Data val;
    Node* next;
    Node(Data _val) : val(_val), next(nullptr) {}
};

class list
{
private:
    Node* first;
    Node* last;
public:
    class ErrorFinding {};
    list() : first(nullptr), last(nullptr) {} // конструктор
    ~list()  //деструктор
    {
        Node* current = first;
        while (current)
        {
            Node* buf = current;
            current = current->next;
            delete buf;
        }
    }
    bool isEmpty() //проверка на пустоту
    {
        return first == nullptr;
    }

    void push_back(Data _val)  //добавление в конец
    {
        Node* p = new Node(_val);
        if (isEmpty()) {
            first = p;
            first->next = nullptr;
            last = p;
            return;
        }
        last->next = p;
        last = p;
        last->next = nullptr;
    }

    void push_front(Data _val) //добавление в начало
    {
        Node* p = new Node(_val);
        if (isEmpty()) {
            first = p;
            first->next = nullptr;
```

```
        last = p;
        return;
    }
    p->next = first;
    first = p;
}

void delete_first() //удалить первый элемент
{
    if (isEmpty()) return;
    Node* p = first;
    first = p->next;
    delete p;
}

void delete_last() //удалить последний элемент
{
    if (isEmpty()) return;
    if (first == last)
    {
        delete_first();
        return;
    }
    Node* p = first;
    while (p->next != last)
        p = p->next;
    p->next = nullptr;
    delete last;
    last = p;
}

void push_insert(Data _val, int _index) //добавление вставкой
{
    Node* current = first;
    while (_index > 1)
    {
        current = current->next;
        _index--;
    }
    Node* buf = current->next;
    current->next = new Node(_val);
    current = current->next;
    current->next = buf;
}

void deleteByIndex(int _index)
{
    Node* current = first;
    if (_index == 1)
        delete_first();
    else
    {
```

```
        while (_index > 2)
        {
            current = current->next;
            _index--;
        }
        Node* buf = current->next;
        current->next = buf->next;
        delete buf;
    }
}

Data findSurname(string _surname) //поиск фамилии
{
    if (isEmpty())
        NULL;
    Node* p = first;
    while (p)
    {
        if (string(p->val.surname).find(_surname) != string::npos)
            return p->val;
        else
            p = p->next;
    }
    throw ErrorFinding();
}

Data findName(string _name) //поиск имени
{
    if (isEmpty())
        NULL;
    Node* p = first;
    while (p)
    {
        if (string(p->val.name).find(_name) != string::npos)
            return p->val;
        else
            p = p->next;
    }
    throw ErrorFinding();
}

Data findLastName(string _lastname) //поиск отчества
{
    if (isEmpty())
        NULL;
    Node* p = first;
    while (p)
    {
        if (string(p->val.lastname).find(_lastname) != string::npos)
            return p->val;
        else
            p = p->next;
```

```
    }
    throw ErrorFinding();
}

Data findCount(int _count) //поиск количества
{
    if (isEmpty())
        NULL;
    Node* p = first;
    while (p)
    {
        if (p->val.count == _count)
            return p->val;
        else
            p = p->next;
    }
    throw ErrorFinding();
}

Data findDay(int _day) //поиск дня
{
    if (isEmpty())
        NULL;
    Node* p = first;
    while (p)
    {
        if (p->val.day == _day)
            return p->val;
        else
            p = p->next;
    }
    throw ErrorFinding();
}

Data findMonth(int _month) //поиск месяца
{
    if (isEmpty())
        NULL;
    Node* p = first;
    while (p)
    {
        if (p->val.month == _month)
            return p->val;
        else
            p = p->next;
    }
    throw ErrorFinding();
}

Data findYear(int _year) //поиск года
{
    if (isEmpty())
```

```
            NULL;
        Node* p = first;
        while (p)
        {
            if (p->val.year == _year)
                return p->val;
            else
                p = p->next;
        }
        throw ErrorFinding();
    }

    Data findDuration(int _duration) //поиск длительности
    {
        if (isEmpty())
            NULL;
        Node* p = first;
        while (p)
        {
            if (p->val.duration == _duration)
                return p->val;
            else
                p = p->next;
        }
        throw ErrorFinding();
    }

    int getCountOfElements()
    {
        Node* fakeFirst = first;
        int i = 0;
        while (fakeFirst)
        {
            i++;
            fakeFirst = fakeFirst->next;
        }
        return i;
    }

    Data currentData(int number)
    {
        if (number == 0) return first->val;
        Node* current = first;
        for (int i = 0; i < number; i++)
            current = current->next;
        return current->val;
    }
};

void PrintData(Data _date)
{
    cout << _date.surname << " " << _date.name << " " << _date.lastname << " " << _date.count << " "
```

```
<< _date.day << "." << _date.month << "." << _date.year << " " << _date.duration << endl;
}

//void PrintLIST(int CountOfElements, list _l) идеальная фунция вывода, но почему-то при втором вызове крашит всю программу, из-за чего цикл внутри этой функции приходится вручную вставлять в код
//{
   //for (int i = 0; i < CountOfElements; i++)
   //{
   //  PrintData(_l.currentData(i));
  //}
//}

void printError() //вывод ошибки на экран
{
   cout << "Совпадений не найдено" << endl;
}

int main()
{
   setlocale(LC_ALL, "Russian");
   int addCount;
   list l;
   Data data;
   string text;
   int index;
   int menu;
   char ch = '.';
   cout << "1. Добавить элемент в конец списка\n2. Добавить элемент в начало списка\n3. Удалить первый элемент\n4. Удалить последний элемент\n5. Поиск элемента по полю\n6. Вставка элемента по номеру\n7. Удаление элемента по номеру" << endl;
   while (1)
   {
      cout << "\nВыберите действие: ";
      cin >> menu;
      switch (menu)
      {
      case 1:
         cout << "Сколько элементов добавим? (элементы добавляются в конец списка) : "; //в конец
         cin >> addCount;
         for (int i = 0; i < addCount; i++)
         {
            cout << "ФИО: ";
            cin >> data.surname >> data.name >> data.lastname;
            cout << "Количество переговоров: ";
            cin >> data.count;
            cout << "Дата переговоров через точки: ";
            cin >> data.day >> ch >> data.month >> ch >> data.year;
            cout << "Длительность переговоров: ";
            cin >> data.duration;
            l.push_back(data);
         }
         cout << "Наш текущий список: " << endl;
```

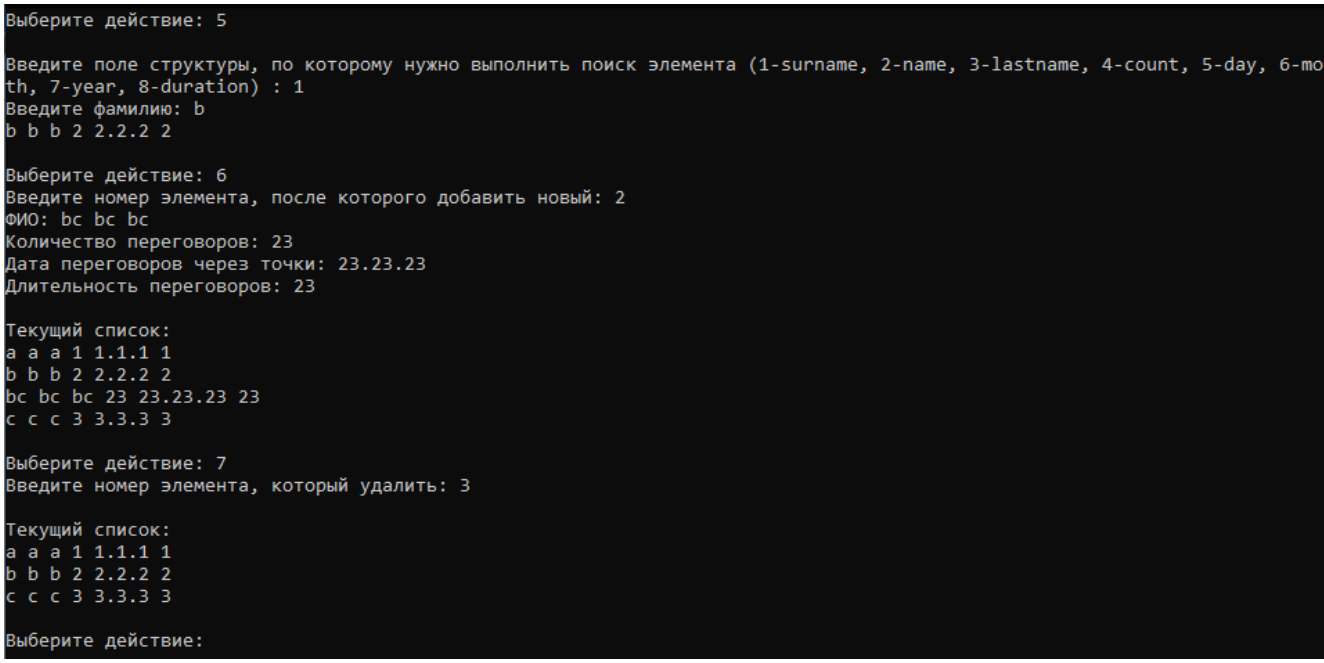
```
            for (int i = 0; i < l.getCountOfElements(); i++)
                PrintData(l.currentData(i));
            break;
        case 2:
            cout << "Сколько элементов добавим? (элементы добавляются в начало списка) : "; //в
начало
            cin >> addCount;
            for (int i = 0; i < addCount; i++)
            {
                cout << "ФИО: ";
                cin >> data.surname >> data.name >> data.lastname;
                cout << "Количество переговоров: ";
                cin >> data.count;
                cout << "Дата переговоров через точки: ";
                cin >> data.day >> ch >> data.month >> ch >> data.year;
                cout << "Длительность переговоров: ";
                cin >> data.duration;
                l.push_front(data);
            }
            cout << "\nНаш текущий список: " << endl;
            for (int i = 0; i < l.getCountOfElements(); i++)
                PrintData(l.currentData(i));
            break;
        case 3:
            cout << "\nУдаление первого элемента. Текущий список: " << endl;
            l.delete_first();
            for (int i = 0; i < l.getCountOfElements(); i++)
                PrintData(l.currentData(i));
            break;
        case 4:
            cout << "\nУдаление последнего элемента. Текущий список: " << endl;
            l.delete_last();
            for (int i = 0; i < l.getCountOfElements(); i++)
                PrintData(l.currentData(i));
            break;
        case 5:
            int choice;
            cout << "\nВведите поле структуры, по которому нужно выполнить поиск элемента (1-
surname, 2-name, 3-lastname, 4-count, 5-day, 6-month, 7-year, 8-duration) : ";
            cin >> choice;
            int IntNumber;
            switch (choice)
            {
            case 1:
                cout << "Введите фамилию: "; cin >> text;
                try
                {
                    PrintData(l.findSurname(text));
                }
                catch (list::ErrorFinding)
                {
                    printError();
```

```
    }
    break;
case 2:
    cout << "Введите имя: "; cin >> text;
    try
    {
        PrintData(l.findName(text));
    }
    catch (list::ErrorFinding)
    {
        printError();
    }
    break;
case 3:
    cout << "Введите отчество: "; cin >> text;
    try
    {
        PrintData(l.findLastName(text));
    }
    catch (list::ErrorFinding)
    {
        printError();
    }
    break;
case 4:
    cout << "Введите количество: "; cin >> IntNumber;
    try
    {
        PrintData(l.findCount(IntNumber));
    }
    catch (list::ErrorFinding)
    {
        printError();
    }
    break;
case 5:
    cout << "Введите день: "; cin >> IntNumber;
    try
    {
        PrintData(l.findDay(IntNumber));
    }
    catch (list::ErrorFinding)
    {
        printError();
    }
    break;
case 6:
    cout << "Введите месяц: "; cin >> IntNumber;
    try
    {
        PrintData(l.findMonth(IntNumber));
    }
```

```
            catch (list::ErrorFinding)
            {
                printError();
            }
            break;
        case 7:
            cout << "Введите год: "; cin >> IntNumber;
            try
            {
                PrintData(l.findYear(IntNumber));
            }
            catch (list::ErrorFinding)
            {
                printError();
            }
            break;
        case 8:
            cout << "Введите длительность: "; cin >> IntNumber;
            try
            {
                PrintData(l.findDuration(IntNumber));
            }
            catch (list::ErrorFinding)
            {
                printError();
            }
            break;
        }
        break;
    case 6:
        cout << "Введите номер элемента, после которого добавить новый: ";
        cin >> index;
        cout << "ФИО: ";
        cin >> data.surname >> data.name >> data.lastname;
        cout << "Количество переговоров: ";
        cin >> data.count;
        cout << "Дата переговоров через точки: ";
        cin >> data.day >> ch >> data.month >> ch >> data.year;
        cout << "Длительность переговоров: ";
        cin >> data.duration;
        l.push_insert(data, index);
        cout << "\nТекущий список: " << endl;
        for (int i = 0; i < l.getCountOfElements(); i++)
            PrintData(l.currentData(i));
        break;
    case 7:
        cout << "Введите номер элемента, который удалить: ";
        cin >> index;
        l.deleteByIndex(index);
        cout << "\nТекущий список: " << endl;
        for (int i = 0; i < l.getCountOfElements(); i++)
            PrintData(l.currentData(i));
```

```
            break;
        }
    }

    return 0;
}
```

```
Выбрать D:\МПУ\VisualStudio\9односвязный список\Debug\9.exe
1. Добавить элемент в конец списка
2. Добавить элемент в начало списка
3. Удалить первый элемент
4. Удалить последний элемент
5. Поиск элемента по полю
6. Вставка элемента по номеру
7. Удаление элемента по номеру

Выберите действие: 1
Сколько элементов добавим? (элементы добавляются в конец списка) : 2
ФИО: a a a
Количество переговоров: 1
Дата переговоров через точки: 1.1.1
Длительность переговоров: 1
ФИО: b b b
Количество переговоров: 2
Дата переговоров через точки: 2.2.2
Длительность переговоров: 2
Наш текущий список:
a a a 1 1.1.1 1
b b b 2 2.2.2 2

Выберите действие: 2
Сколько элементов добавим? (элементы добавляются в начало списка) : 1
ФИО: o o o
Количество переговоров: 0
Дата переговоров через точки: 0.0.0
Длительность переговоров: 0

Наш текущий список:
o o o 0 0.0.0 0
a a a 1 1.1.1 1
b b b 2 2.2.2 2

Выберите действие: 3

Удаление первого элемента. Текущий список:
a a a 1 1.1.1 1
b b b 2 2.2.2 2

Выберите действие: 4

Удаление последнего элемента. Текущий список:
a a a 1 1.1.1 1

Выберите действие: 1
Сколько элементов добавим? (элементы добавляются в конец списка) : 2
ФИО: b b b
Количество переговоров: 2
Дата переговоров через точки: 2.2.2
Длительность переговоров: 2
ФИО: c c c
Количество переговоров: 3
Дата переговоров через точки: 3.3.3
Длительность переговоров: 3
Наш текущий список:
a a a 1 1.1.1 1
b b b 2 2.2.2 2
c c c 3 3.3.3 3

Выберите действие: 5
```



Двусвязный список

```
#include <iostream>
#include <string>
#include <cmath>
using namespace std;

struct Data
{
   char surname[20];
   char name[15];
   char lastname[25];
   int count;
   int day;
   int month;
   int year;
   int duration;
};

struct Node //узел для двусвязного списка
{
   Data val;
   Node* next;
   Node* prev;
   Node(Data _val) : val(_val), next(nullptr), prev(nullptr) {}
};

class list //двусвязный список
{
private:
   Node* first;
   Node* last;
public:
   class ErrorFinding {};
```

```
list() : first(nullptr), last(nullptr) {} // конструктор
~list()  //деструктор
{
   Node* current = first;
   while (current != nullptr)
   {
      Node* buf = current;
      current = current->next;
      delete buf;
   }
}

bool isEmpty() //проверка на пустоту
{
   return first == nullptr;
}

void push_back(Data _val)  //добавление в конец
{
   Node* p = new Node(_val);
   if (isEmpty())
   {
      first = p;
      first->prev = nullptr;
      first->next = nullptr;
      last = p;
      last->prev = nullptr;
      last->next = nullptr;
      return;
   }
   last->next = p;
   p->prev = last;
   last = p;
   last->next = nullptr;
}

void push_front(Data _val) //добавление в начало
{
   Node* p = new Node(_val);
   if (isEmpty()) {
      first = p;
      last = p;
      return;
   }
   first->prev = p;
   p->next = first;
   first = p;
}

void delete_first() //удалить первый элемент
{
   if (isEmpty()) return;
```

```
    Node* p = first;
    if (p->next != nullptr)
    {
        first = p->next;
        delete p;
        first->prev = nullptr;
    }
    else
    {
        delete p;
        first = nullptr;
    }

}

void delete_last() //удалить последний элемент
{
    if (isEmpty()) return;
    if (first == last)
    {
        delete_first();
        return;
    }
    Node* p = last;
    last = p->prev;
    last->next = nullptr;
    delete p;
}

void push_insert(Data _val, int _index) //добавление вставкой
{
    Node* current = first;
    while (_index > 1)
    {
        current = current->next;
        _index--;
    }
    if (current == last) push_back(_val);
    else
    {
        Node* buf1 = current->next;
        current = current->next;
        Node* buf2 = current->prev;
        current = current->prev;
        current->next = new Node(_val);
        current = current->next;
        current->prev = buf2;
        current->next = buf1;
        Node* buf3 = current;
        current = current->next;
        current->prev = buf3;
    }
```

```
}

void deleteByIndex(int _index)
{
   Node* current = first;
   if (_index == 1)
      delete_first();
   else
   {
      while (_index > 2)
      {
         current = current->next;
         _index--;
      }
      if (current->next == last) delete_last();
      else
      {
         Node* buf = current->next;
         current->next = buf->next;
         current = current->next;

         buf = current->prev;
         current = current->prev;
         buf = buf->prev;
         delete current;
      }
   }


}

Data findSurname(string _surname) //поиск фамилии
{
   if (isEmpty())
      NULL;
   Node* p = first;
   while (p)
   {
      if (string(p->val.surname).find(_surname) != string::npos)
         return p->val;
      else
         p = p->next;
   }
   throw ErrorFinding();
}

Data findName(string _name) //поиск имени
{
   if (isEmpty())
      NULL;
   Node* p = first;
```

```
    while (p)
    {
        if (string(p->val.name).find(_name) != string::npos)
            return p->val;
        else
            p = p->next;
    }
    throw ErrorFinding();
}

Data findLastName(string _lastname) //поиск отчества
{
    if (isEmpty())
        NULL;
    Node* p = first;
    while (p)
    {
        if (string(p->val.lastname).find(_lastname) != string::npos)
            return p->val;
        else
            p = p->next;
    }
    throw ErrorFinding();
}

Data findCount(int _count) //поиск количества
{
    if (isEmpty())
        NULL;
    Node* p = first;
    while (p)
    {
        if (p->val.count == _count)
            return p->val;
        else
            p = p->next;
    }
    throw ErrorFinding();
}

Data findDay(int _day) //поиск дня
{
    if (isEmpty())
        NULL;
    Node* p = first;
    while (p)
    {
        if (p->val.day == _day)
            return p->val;
        else
            p = p->next;
    }
```

```
    throw ErrorFinding();
}

Data findMonth(int _month) //поиск месяца
{
    if (isEmpty())
        NULL;
    Node* p = first;
    while (p)
    {
        if (p->val.month == _month)
            return p->val;
        else
            p = p->next;
    }
    throw ErrorFinding();
}

Data findYear(int _year) //поиск года
{
    if (isEmpty())
        NULL;
    Node* p = first;
    while (p)
    {
        if (p->val.year == _year)
            return p->val;
        else
            p = p->next;
    }
    throw ErrorFinding();
}

Data findDuration(int _duration) //поиск длительности
{
    if (isEmpty())
        NULL;
    Node* p = first;
    while (p)
    {
        if (p->val.duration == _duration)
            return p->val;
        else
            p = p->next;
    }
    throw ErrorFinding();
}

int getCountOfElements()
{
    Node* fakeFirst = first;
    int i = 0;
```

```
        while (fakeFirst)
        {
            i++;
            fakeFirst = fakeFirst->next;
        }
        return i;
    }

    Data currentData(int number)
    {
        if (number == 0) return first->val;
        Node* current = first;
        for (int i = 0; i < number; i++)
            current = current->next;
        return current->val;
    }

};

void PrintData(Data _date)
{
    cout << _date.surname << " " << _date.name << " " << _date.lastname << " " << _date.count << " "
<< _date.day << "." << _date.month << "." << _date.year << " " << _date.duration << endl;
}

//void PrintLIST(int CountOfElements, list _l) идеальная фунция вывода, но почему-то при втором
вызове крашит всю программу, из-за чего цикл внутри этой функции приходится вручную
вставлять в код
//{
    //for (int i = 0; i < CountOfElements; i++)
    //{
    //  PrintData(_l.currentData(i));
  //}
//}



void printError() //вывод ошибки на экран
{
    cout << "Совпадений не найдено" << endl;
}

int main()
{
    setlocale(LC_ALL, "Russian");
    int addCount;
    list l;
    Data data;
    string text;
    int index;
    int menu;
    char ch = '.';
    cout << "1. Добавить элемент в конец списка\n2. Добавить элемент в начало списка\n3. Удалить
первый элемент\n4. Удалить последний элемент\n5. Поиск элемента по полю\n6. Вставка
```

```
элемента по номеру\n7. Удаление элемента по номеру" << endl;
   while (1)
   {
      cout << "\nВыберите действие: ";
      cin >> menu;
      switch (menu)
      {
      case 1:
         cout << "Сколько элементов добавим? (элементы добавляются в конец списка) : "; //в конец
         cin >> addCount;
         for (int i = 0; i < addCount; i++)
         {
            cout << "ФИО: ";
            cin >> data.surname >> data.name >> data.lastname;
            cout << "Количество переговоров: ";
            cin >> data.count;
            cout << "Дата переговоров через точки: ";
            cin >> data.day >> ch >> data.month >> ch >> data.year;
            cout << "Длительность переговоров: ";
            cin >> data.duration;
            l.push_back(data);
         }
         cout << "Наш текущий список: " << endl;
         for (int i = 0; i < l.getCountOfElements(); i++)
            PrintData(l.currentData(i));
         break;
      case 2:
         cout << "Сколько элементов добавим? (элементы добавляются в начало списка) : "; //в
начало
         cin >> addCount;
         for (int i = 0; i < addCount; i++)
         {
            cout << "ФИО: ";
            cin >> data.surname >> data.name >> data.lastname;
            cout << "Количество переговоров: ";
            cin >> data.count;
            cout << "Дата переговоров через точки: ";
            cin >> data.day >> ch >> data.month >> ch >> data.year;
            cout << "Длительность переговоров: ";
            cin >> data.duration;
            l.push_front(data);
         }
         cout << "\nНаш текущий список: " << endl;
         for (int i = 0; i < l.getCountOfElements(); i++)
            PrintData(l.currentData(i));
         break;
      case 3:
         cout << "\nУдаление первого элемента. Текущий список: " << endl;
         l.delete_first();
         for (int i = 0; i < l.getCountOfElements(); i++)
            PrintData(l.currentData(i));
         break;
```

```
        case 4:
            cout << "\nУдаление последнего элемента. Текущий список: " << endl;
            l.delete_last();
            for (int i = 0; i < l.getCountOfElements(); i++)
                PrintData(l.currentData(i));
            break;
        case 5:
            int choice;
            cout << "\nВведите поле структуры, по которому нужно выполнить поиск элемента (1-surname, 2-name, 3-lastname, 4-count, 5-day, 6-month, 7-year, 8-duration) : ";
            cin >> choice;
            int IntNumber;
            switch (choice)
            {
            case 1:
                cout << "Введите фамилию: "; cin >> text;
                try
                {
                    PrintData(l.findSurname(text));
                }
                catch (list::ErrorFinding)
                {
                    printError();
                }
                break;
            case 2:
                cout << "Введите имя: "; cin >> text;
                try
                {
                    PrintData(l.findName(text));
                }
                catch (list::ErrorFinding)
                {
                    printError();
                }
                break;
            case 3:
                cout << "Введите отчество: "; cin >> text;
                try
                {
                    PrintData(l.findLastName(text));
                }
                catch (list::ErrorFinding)
                {
                    printError();
                }
                break;
            case 4:
                cout << "Введите количество: "; cin >> IntNumber;
                try
                {
                    PrintData(l.findCount(IntNumber));
```

```
    }
    catch (list::ErrorFinding)
    {
        printError();
    }
    break;
case 5:
    cout << "Введите день: "; cin >> IntNumber;
    try
    {
        PrintData(l.findDay(IntNumber));
    }
    catch (list::ErrorFinding)
    {
        printError();
    }
    break;
case 6:
    cout << "Введите месяц: "; cin >> IntNumber;
    try
    {
         PrintData(l.findMonth(IntNumber));
    }
    catch (list::ErrorFinding)
    {
        printError();
    }
    break;
case 7:
    cout << "Введите год: "; cin >> IntNumber;
    try
    {
        PrintData(l.findYear(IntNumber));
    }
    catch (list::ErrorFinding)
    {
        printError();
    }
    break;
case 8:
    cout << "Введите длительность: "; cin >> IntNumber;
    try
    {
        PrintData(l.findDuration(IntNumber));
    }
    catch (list::ErrorFinding)
    {
        printError();
    }
    break;
}
break;
```

```
        case 6:
            cout << "Введите номер элемента, после которого добавить новый: ";
            cin >> index;
            cout << "ФИО: ";
            cin >> data.surname >> data.name >> data.lastname;
            cout << "Количество переговоров: ";
            cin >> data.count;
            cout << "Дата переговоров через точки: ";
            cin >> data.day >> ch >> data.month >> ch >> data.year;
            cout << "Длительность переговоров: ";
            cin >> data.duration;
            l.push_insert(data, index);
            cout << "\nТекущий список: " << endl;
            for (int i = 0; i < l.getCountOfElements(); i++)
                PrintData(l.currentData(i));
            break;
        case 7:
            cout << "Введите номер элемента, который удалить: ";
            cin >> index;
            l.deleteByIndex(index);
            cout << "\nТекущий список: " << endl;
            for (int i = 0; i < l.getCountOfElements(); i++)
                PrintData(l.currentData(i));
            break;
        }
    }

    return 0;
}
```

```
D:\МПУ\VisualStudio\10двусвязный список\Debug\10.exe
1. Добавить элемент в конец списка
2. Добавить элемент в начало списка
3. Удалить первый элемент
4. Удалить последний элемент
5. Поиск элемента по полю
6. Вставка элемента по номеру
7. Удаление элемента по номеру

Выберите действие: 1
Сколько элементов добавим? (элементы добавляются в конец списка) : 5
ФИО: a a a
Количество переговоров: 1
Дата переговоров через точки: 1.1.1
Длительность переговоров: 1
ФИО: b b b
Количество переговоров: 2
Дата переговоров через точки: 2.2.2
Длительность переговоров: 2
ФИО: c c c
Количество переговоров: 3
Дата переговоров через точки: 3.3.3
Длительность переговоров: 3
ФИО: d d d
Количество переговоров: 4
Дата переговоров через точки: 4.4.4
Длительность переговоров: 4
ФИО: e e e
Количество переговоров: 5
Дата переговоров через точки: 5.5.5
Длительность переговоров: 5
Наш текущий список:
a a a 1 1.1.1 1
b b b 2 2.2.2 2
c c c 3 3.3.3 3
d d d 4 4.4.4 4
e e e 5 5.5.5 5

Выберите действие: 2
Сколько элементов добавим? (элементы добавляются в начало списка) : 1
ФИО: o o o
Количество переговоров: 0
Дата переговоров через точки: 0.0.0
Длительность переговоров: 0

Наш текущий список:
o o o 0 0.0.0 0
a a a 1 1.1.1 1
b b b 2 2.2.2 2
c c c 3 3.3.3 3
d d d 4 4.4.4 4
e e e 5 5.5.5 5

Выберите действие: 3

Удаление первого элемента. Текущий список:
a a a 1 1.1.1 1
b b b 2 2.2.2 2
c c c 3 3.3.3 3
d d d 4 4.4.4 4
e e e 5 5.5.5 5
```

```
Выберите действие: 4

Удаление последнего элемента. Текущий список:
a a a 1 1.1.1 1
b b b 2 2.2.2 2
c c c 3 3.3.3 3
d d d 4 4.4.4 4

Выберите действие: 5

Введите поле структуры, по которому нужно выполнить поиск элемента (1-surname, 2-name, 3-lastname, 4-count, 5-day, 6-mon
th, 7-year, 8-duration) : 1
Введите фамилию: c
c c c 3 3.3.3 3

Выберите действие: 6
Введите номер элемента, после которого добавить новый: 3
ФИО: cd cd cd
Количество переговоров: 34
Дата переговоров через точки: 34.34.34
Длительность переговоров: 34

Текущий список:
a a a 1 1.1.1 1
b b b 2 2.2.2 2
c c c 3 3.3.3 3
cd cd cd 34 34.34.34 34
d d d 4 4.4.4 4

Выберите действие: 7
Введите номер элемента, который удалить: 2

Текущий список:
a a a 1 1.1.1 1
c c c 3 3.3.3 3
cd cd cd 34 34.34.34 34
d d d 4 4.4.4 4

Выберите действие:
```