



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования

**“МИРЭА - Российский технологический университет”**

**РТУ МИРЭА**

---

Институт информационных технологий (ИТ)  
Кафедра математического обеспечения и стандартизации  
информационных технологий (МОСИТ)

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №7**

**по дисциплине**

**«Структуры и алгоритмы обработки данных»**

Тема «Линейные динамические списки»

Выполнил студент группы ИКБО-50-23

Русаков М.Ю.

Принял старший преподаватель

Скворцова Л.А.

Москва 2024

## Оглавление

1. Условие задачи .....	3
2. Постановка задачи .....	3
3. АТД задачи .....	4
3.1. АТД задания 1 .....	4
3.1.1. sList .....	4
3.1.2. sNode .....	5
3.2. АТД задания 2 .....	6
3.2.1. dList .....	6
3.2.2. dNode .....	7
3.2.3. Book .....	8
3.2.4. Fields .....	8
3.2.5. Field .....	9
4. Разработка и реализация задачи .....	10
4.1. Задание 1 .....	10
4.1.1. Структура узла .....	10
4.1.2. Изображение структуры списка для 5 элементов .....	10
4.1.3. Изображение операции добавления узла в начало списка .....	11
4.1.4. Последовательность операторов, требующихся для добавления узла в начало списка .....	11
4.1.5. Код программы .....	11
4.1.6. Тестирование .....	16
4.1.6.1. insert .....	16
4.1.6.2. deleteFrequent .....	17
4.1.6.3. replaceFirst .....	17
4.2. Задание 2 .....	18
4.2.1. Структура узла .....	18
4.2.2. Изображение структуры списка для 5 элементов .....	18
4.2.3. Изображение операции добавления узла в начало списка .....	19
4.2.4. Последовательность операторов, требующихся для добавления узла в начало списка .....	19
4.2.5. Код программы .....	19
4.2.6. Тестирование .....	28
5. Выводы .....	30
6. Список информационных источников .....	31

# 1. Условие задачи

Задания моего персонального варианта (№22):

**Задание 1.** Разработать и реализовать программу управлению линейным однонаправленным списком в соответствии с условием задачи, определенной вариантом.

**Задание 2.** Разработать и реализовать программу управлению линейным двунаправленным списком в соответствии с условием задачи, определенной вариантом.

# 2. Постановка задачи

**Задание 1.** Дан список, узел которого хранит значение и количество вхождений этого значения (в исходном списке это значение равно 1).

1. Вставить символ в список, если этот символ уже есть в списке, то увеличить количество вхождений символа, если такого значения еще нет в списке, то добавить его в конец списка.
2. Удалить из списка узел с символом, который чаще всего встретился в списке.
3. Модифицировать список, переместив первые  $m$  узлов в конец списка.

**Задание 2.** Системный каталог библиотечной автоматизированной системы хранит сведения о книгах фонда в двунаправленном списке. Сведения о принадлежности книги области знания хранит отдельный список, ссылка на который храниться в узле исходного списка соответствующей книги. Часть списка, начиная с узла с заданным номером (номер больше 1) и до конца списка, перенести в начало списка.

## 3. АТД задачи

### 3.1. АТД задания 1

#### 3.1.1. sList

АТД sList

{

*Данные* (описание свойств структуры данных задачи)

head – указатель на головной элемент списка

*Операции* (объявления операций)

1. Метод, осуществляющий создание списка

Предусловие: values – массив символов

Постусловие: линейный однонаправленный список

Заголовок: sList(std::vector<char> values)

2. Метод, осуществляющий вывод однонаправленного списка

Предусловие: линейный однонаправленный список

Постусловие: выведенные в консоль значения узлов списка

Заголовок: print()

3. Метод, осуществляющие вставку узла в начало списка

Предусловие: sym – символ

Постусловие: обновленный линейный однонаправленный список

Заголовок: pushFront(char sym)

4. Метод, осуществляющий вставку узла в конец списка, если символ не встречается в списке, в противном случае увеличивающий число вхождений символа в данный список

Предусловие: sym – символ

Постусловие: обновленный линейный однонаправленный список

Заголовок: insert(char sym)

5. Метод, осуществляющий перемещение первых  $m$  узлов в конец списка

Предусловие: m – число перемещаемых узлов

Постусловие: обновленный линейный однонаправленный список

Заголовок: replaceFirst(int m)

6. Метод, удаляющий узел с наибольшим числом вхождения определенного символа

Предусловие: линейный однонаправленный список

Постусловие: обновленный линейный однонаправленный список

Заголовок: `deleteFrequent()`

7. Метод, осуществляющий поиск узла, содержащего заданный символ

Предусловие: `sym` – искомый символ

Постусловие: указатель на узел (в случае, если узла с искомым символом в списке нет, возвращается нулевой указатель)

Заголовок: `inList(char sym)`

8. Метод, возвращающий указатель на последний узел списка

Предусловие: линейный однонаправленный список

Постусловие: указатель на последний узел списка

Заголовок: `getTail()`

9. Метод, возвращающий длину списка

Предусловие: линейный однонаправленный список

Постусловие: длина линейного однонаправленного списка

Заголовок: `size()`

}

### 3.1.2. sNode

АТД sNode

{

*Данные* (описание свойств структуры данных задачи)

`sym` – символ

`occurrence` – число вхождений символа в список

`next` – указатель на следующий узел списка

*Операции* (объявления операций)

1. Метод, осуществляющий создание узла

Предусловие: `sym` – символ

Постусловие: узел линейного однонаправленного списка

Заголовок: `sNode(char value)`

}

## 3.2. АДД задания 2

### 3.2.1. dList

АДД dList

{

*Данные* (описание свойств структуры данных задачи)

head – указатель на головной элемент списка

tail – указатель на последний элемент списка

*Операции* (объявления операций)

1. Метод, осуществляющий создание списка

Предусловие: books – массив, содержащий данные о книгах

Постусловие: линейный двунаправленный список

Заголовок: dList(std::vector<Book> books)

2. Метод, осуществляющие вставку узла в начало списка

Предусловие: book – структура, хранящая данные о книге

Постусловие: обновленный список

Заголовок: pushFront(Book book)

3. Метод, осуществляющий вывод данных о книге

Предусловие: линейный двунаправленный список

Постусловие: данные о книге: инвентарный номер, название, автор, год издания, области знаний

Заголовок: printBook(dNode\* node)

4. Метод, осуществляющий вывод данных о всех книгах каталога

Предусловие: backwards – булевый параметр, отвечающий за порядок вывода каталога (false – список выводится слева направо, true – список выводится справа налево)

Постусловие: выведенная в консоль информация о всех книгах

Заголовок: printCatalog(bool backwards = false)

5. Метод, осуществляющий перенос всех узлов списка, начиная с заданного номера, в начало списка

Предусловие: num – номер, с которого начинается перенос

Постусловие: обновленный список

Заголовок: replaceAfter(int num)

6. Метод, осуществляющий поиск узла списка по ключу

Предусловие: `key` – ключ (в моем персональном варианте ключом считается инвентарный номер книги)

Постусловие: указатель на узел списка

Заголовок: `find(int key)`

7. Метод, возвращающий длину списка

Предусловие: линейный двунаправленный список

Постусловие: длина линейного двунаправленного списка

Заголовок: `size()`

}

### 3.2.2. dNode

АТД dNode

{

*Данные* (описание свойств структуры данных задачи)

`book` – структура, содержащая данные о книге

`next` – указатель на следующий узел списка

`prev` – указатель на предыдущий узел списка

*Операции* (объявления операций)

1. Метод, осуществляющий создание узла

Предусловие: `book` – структура, содержащая данные о книге

Постусловие: узел линейного двунаправленного списка

Заголовок: `dNode(Book book)`

}

### 3.2.3. Book

АТД Book

```
{  
  
    Данные (описание свойств структуры данных задачи)  
  
    key – «ключ», инвентарный номер книги  
  
    year – год издания книги  
  
    author – автор книги  
  
    title – название книги  
  
    fields – список наименований областей знаний книги  
  
    Операции (объявления операций)  
  
    1. Метод, осуществляющий создание структуры  
    Предусловие: key – инвентарный номер, year – год издания, author –  
    автор, title – название, fields – наименования области знаний  
    Постусловие: структура, содержащая информацию о книге  
    Заголовок: Book(int key, int year std::string author, std::string  
    title, std::vector<std::string> fields)  
  
}
```

### 3.2.4. Fields

АТД Fields

```
{  
  
    Данные (описание свойств структуры данных задачи)  
  
    head – указатель на головной элемент списка  
  
    Операции (объявления операций)  
  
    1. Метод, осуществляющий создание списка  
    Предусловие: fields – массив, содержащий наименования областей  
    знаний книги  
    Постусловие: список областей знаний книги  
    Заголовок: Fields(std::vector<std::string> fields)  
  
}
```



### 3.2.5. Field

АТД Field

{

field – наименование области знаний книги

next – указатель на следующий узел списка

*Операции* (объявления операций)

1. Метод, осуществляющий создание узла

Предусловие: field – структура, содержащая наименование области знаний книги

Постусловие: узел линейного однонаправленного списка

Заголовок: Field(std::string field)

}

## 4. Разработка и реализация задачи

### 4.1. Задание 1

#### 4.1.1. Структура узла

В соответствии с моим вариантом, структура данных, соответствующая узла однонаправленного списка, должна содержать:

1. Поле, хранящее символ (*sym*).
2. Поле, хранящее число вхождений символа в список (*occurrence*).  
Подразумевается, что список не должен содержать узлов, значения полей *sym* у которых равны.
3. Поле, содержащее указатель на следующий узел списка (*next*).

#### 4.1.2. Изображение структуры списка для 5 элементов

Ниже приведено изображение списка, состоящего из 5 элементов (см. рис. 2).

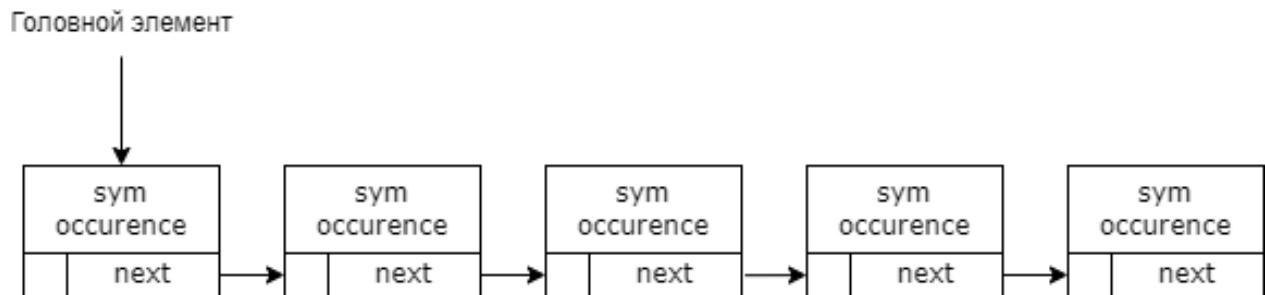


Рисунок 1 – Структура списка из 5 элементов

### 4.1.3. Изображение операции добавления узла в начало списка

Ниже приведено изображение, показывающее операцию вставки узла в начало списка в случае, если поле `sym` ни одного из узлов исходного списка не совпадает с полем `sym` нового узла (см. рис. 3).

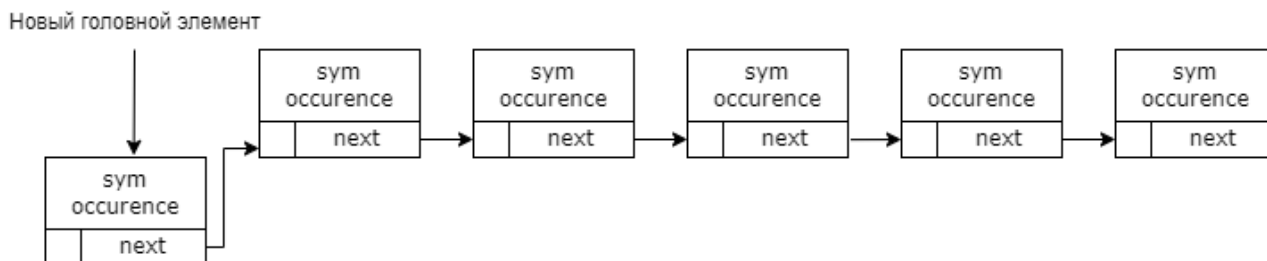


Рисунок 2 – Операция добавления узла в начало списка

### 4.1.4. Последовательность операторов, требующихся для добавления узла в начало списка

1. Проверка нового символа на вхождение в существующий список путем вызова метода `inList`. Если символ уже встречается в списке, поле `occurrence` узла, в котором встречается этот символ, увеличивается на 1. Иначе создается указатель на новый узел списка `node`.
2. Полю `next` узла по указателю `node` присваивается значение указателя на головной элемент `head`.
3. Указателю `head` присваивается значение `node`.

### 4.1.5. Код программы

1. Код файла `s_list.h`

```
#include <iostream>
#include <vector>
#include "s_node.h"

struct sList {
    sNode* head;

    sList(std::vector<char> values);
    ~sList();

    void print();
    void pushFront(char sym);
    void insert(char sym);
    void replaceFirst(int m);
    void deleteFrequent();

    sNode* inList(char sym);
    sNode* getTail();
};
```

```

    int size();
};

```

## 2. Код файла s\_list.cpp

```

#include "s_list.h"
#include <iostream>
#include <vector>

sList::sList(std::vector<char> values) {
    if (values.empty()) {
        std::cerr << "НЕВОЗМОЖНО СОЗДАТЬ СПИСОК: ПУСТОЙ ВЕКТОР\n";
        return;
    }

    if (values.size() == 1) {
        this->head = new sNode(values[0]);
        return;
    }

    sNode* temp = new sNode(values[0]);
    this->head = temp;
    for (int i{1}; i < values.size(); i++) {
        sNode* listed = inList(values[i]);
        if (listed) {
            listed->occurrence++;
            continue;
        }
        temp->next = new sNode(values[i]);
        temp = temp->next;
    }

    if (head == nullptr) {
        std::cerr << "НЕВОЗМОЖНО СОЗДАТЬ СПИСОК: НЕИЗВЕСТНАЯ ОШИБКА\n";
    }
}

sList::~sList() {
    while (head != nullptr) {
        sNode* temp = head;
        head = head->next;
        delete temp;
    }
}

void sList::pushFront(char sym) {
    sNode* listed = inList(sym);

```

```

    if (listed) {
        listed->occurrence++;
        return;
    }

    sNode* node = new sNode(sym);
    node->next = head;
    head = node;
}

void sList::print() {
    sNode* node = head;
    std::cout << '(' << node->sym << ',' << node->occurrence << ')';
    node = node->next;
    while (node != nullptr) {
        std::cout << " -> (" << node->sym << ',' << node->occurrence <<
    ');
        node = node->next;
    }
    std::cout << '\n';
}

void sList::insert(char sym) {
    sNode* temp = head;
    while (temp->next != nullptr) {
        if (sym == temp->sym) {
            temp->occurrence++;
            return;
        }
        temp = temp->next;
    }

    if (sym == temp->sym) {
        temp->occurrence++;
        return;
    }

    temp->next = new sNode(sym);
}

void sList::replaceFirst(int m) {
    if (m >= size()) {
        std::cerr << "невозможно переместить узлы: входной параметр больше
или равен длины списка\n";
        return;
    }
}

```

```

sNode* temp = head;
for (int i{}; i < m-1; i++) {
    temp = temp->next;
}

sNode* tempHead = head;
sNode* tail     = getTail();

head          = temp->next;
temp->next = nullptr;
tail->next = tempHead;
}

void sList::deleteFrequent() {
    if (head == nullptr) {
        std::cerr << "список пуст!\n";
        return;
    }

    int    max          = head->occurrence;
    sNode* node          = head;
    sNode* beforeMax = nullptr;

    while (node->next->next != nullptr) {
        if (node->next->occurrence > max) {
            max          = node->next->occurrence;
            beforeMax = node;
        }
        node = node->next;
    }

    if (beforeMax) {
        sNode* newNext = beforeMax->next->next;
        delete beforeMax->next;
        beforeMax->next = newNext;
    } else {
        sNode* newHead = head->next;
        delete head;
        head = newHead;
    }
}

int sList::size() {
    int size = 0;
    sNode* node = head;

```

```

        while (node != nullptr) {
            size++;
            node = node->next;
        }

        return size;
    }

sNode* sList::inList(char sym) {
    sNode* result = head;
    while (result != nullptr) {
        if (result->sym == sym) {
            return result;
        }
        result = result->next;
    }

    return nullptr;
}

sNode* sList::getTail() {
    sNode* tail = head;
    while (tail->next != nullptr) {
        tail = tail->next;
    }

    return tail;
}

```

### 3. Код файла s\_node.h

```

struct sNode {
    char    sym;
    int     occurrence;
    sNode*  next;

    sNode();
    sNode(char value);
};

```

### 4. Код файла s\_node.cpp

```

#include "s_node.h"

sNode::sNode() {
    this->sym      = ' ';
    this->occurrence = 0;
    this->next     = nullptr;
}

```

```

}

sNode::sNode(char value) {
    this->sym      = value;
    this->occurrence = 1;
    this->next      = nullptr;
}

```

5. Код файла main.cpp

```

#include "headers/s_list.h"
#include <iostream>
#include <vector>

int main() {
    std::vector<char> values = {'r', 'i', 'z', 'z'};

    sList* list = new sList(values);

    list->print();
    list->replaceFirst(3);
    list->print();

    delete list;
}

```

#### 4.1.6. Тестирование

##### 4.1.6.1. insert

1. Исходный список: 'с' → 'о' → 'г' → 'н'. Вставляемое значение: 'у'. Результат выполнения метода приведен ниже (см. рис. 3, здесь и далее в первой строке вывода содержится состояние списка до выполнения метода, во второй строке – после выполнения метода)

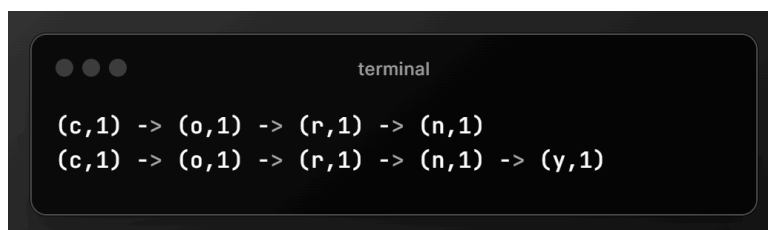
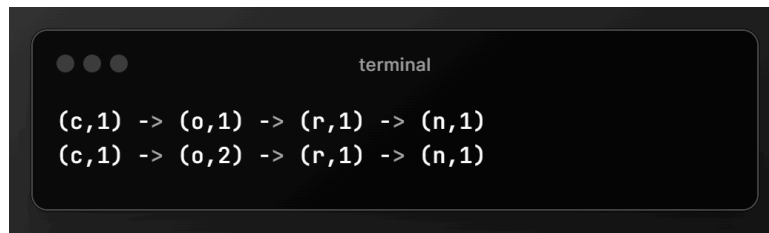


Рисунок 3 – Тест №1



2. Исходный список тот же. Вставляемое значение: 'o'. Результат выполнения метода приведен ниже (см. рис. 4)

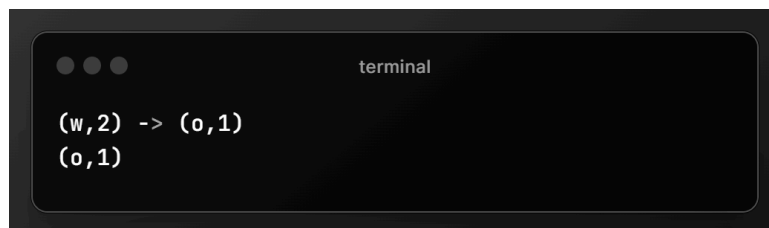


```
terminal
(c,1) -> (o,1) -> (r,1) -> (n,1)
(c,1) -> (o,2) -> (r,1) -> (n,1)
```

Рисунок 4 – Тест №2

#### 4.1.6.2. deleteFrequent

3. Исходный список: 'w' → 'o' → 'w'. Результат выполнения метода приведен ниже (см. рис. 5)

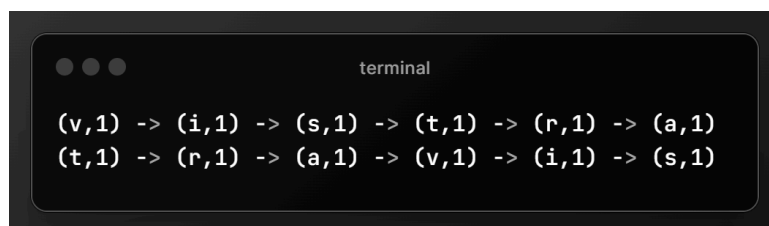


```
terminal
(w,2) -> (o,1)
(o,1)
```

Рисунок 5 – Тест №3

#### 4.1.6.3. replaceFirst

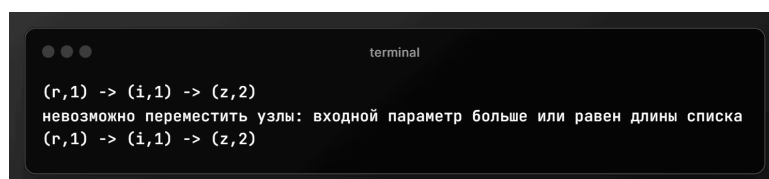
4. Исходный список: 'v' → 'i' → 's' → 't' → 'r' → 'a'. Количество перемещаемых символов: 3. Результат выполнения метода приведен ниже (см. рис. 6).



```
terminal
(v,1) -> (i,1) -> (s,1) -> (t,1) -> (r,1) -> (a,1)
(t,1) -> (r,1) -> (a,1) -> (v,1) -> (i,1) -> (s,1)
```

Рисунок 6 – Тест №4

5. Исходный список: 'r' → 'i' → 'z' → 'z'. Количество перемещаемых символов: 3. Результат выполнения метода приведен ниже (см. рис. 7).



```
terminal
(r,1) -> (i,1) -> (z,2)
невозможно переместить узлы: входной параметр больше или равен длины списка
(r,1) -> (i,1) -> (z,2)
```

Рисунок 7 – Тест №5

## 4.2. Задание 2

### 4.2.1. Структура узла

В соответствии с моим вариантом, структура данных, соответствующая узла двунаправленного списка, должна содержать:

1. Поле, хранящее данные о книге (**book**, данное поле представляет собой отдельную структуру **Book**).
2. Поле, содержащее указатель на следующий узел списка (**next**).
3. Поле, содержащее указатель на предыдущий узел списка (**prev**).

Структура **Book** содержит инвентарный номер книги, являющийся ее ключом (**key**), год издания книги (**year**), имя автора книги (**author**), название книги (**title**), а также список областей знаний книги (**fields**), представленный линейным однонаправленным списком, который в свою очередь представлен структурой **Fields**

Структура **Fields** содержит указатель на головной узел списка (**head**), представленный структурой **Field**

Структура **Field** содержит наименование области знаний книги (**field**) и указатель на следующий узел списка (**next**).

### 4.2.2. Изображение структуры списка для 5 элементов

Ниже приведено изображение списка, состоящего из 5 элементов (см. рис. 3).

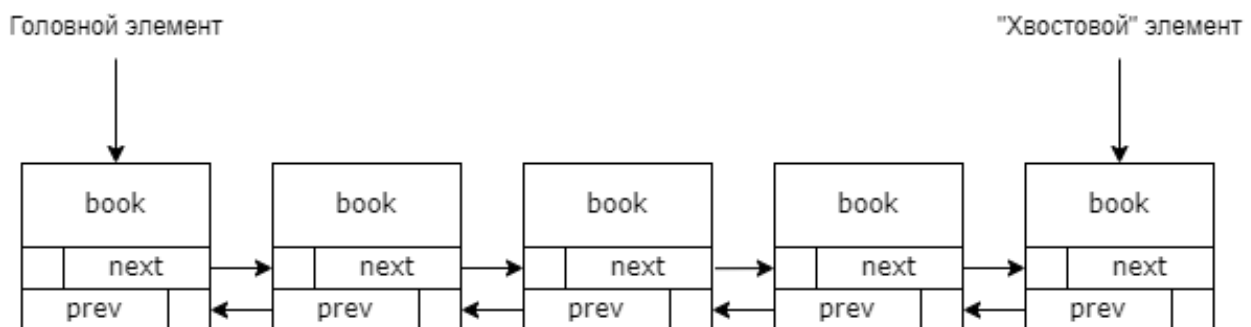


Рисунок 8 – Структура списка из 5 элементов

### 4.2.3. Изображение операции добавления узла в начало списка

Ниже приведено изображение, показывающее операцию вставки узла в начало списка (см. рис. 4).

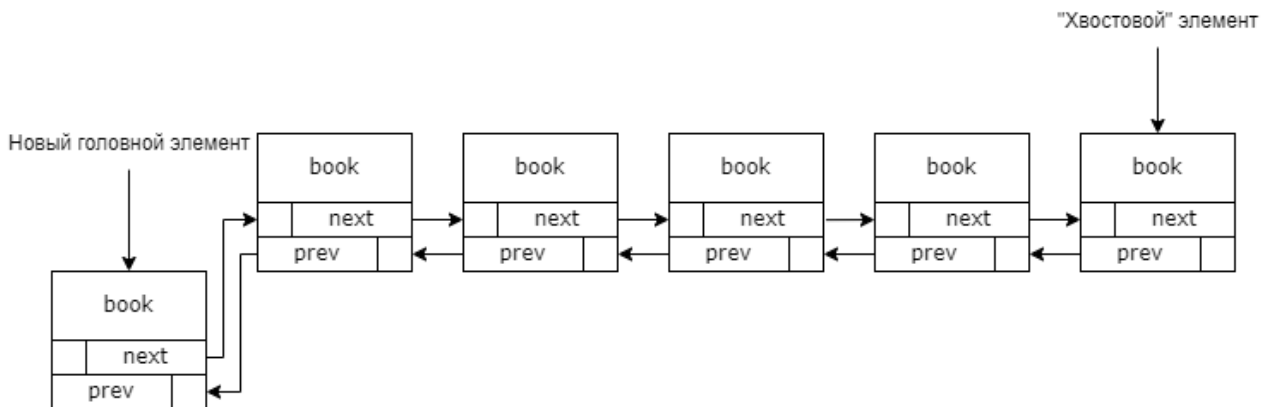


Рисунок 9 – Операция добавления узла в начало списка

### 4.2.4. Последовательность операторов, требующихся для добавления узла в начало списка

1. Создание указателя на новый узел списка newNode.
2. Полю next узла по указателю newNode присваивается значение указателя на головной узел head
3. Полю prev узла по указателю head присваивается значение newNode
4. Указателю head присваивается значение newNode.

### 4.2.5. Код программы

1. Код файла d\_list.h

```
#include "d_node.h"
#include <vector>

struct dList {
    dNode* head;
    dNode* tail;

    dList(std::vector<Book> books);

    void pushFront(Book book);
    void printBook(dNode* node);
    void printCatalog(bool backwards = false);
    void replaceAfter(int num);
    dNode* find(int key);

    int size();
};
```

## 2. Код файла d\_list.cpp

```
#include "../headers/d_list.h"
#include <iostream>

dList::dList(std::vector<Book> books) {
    std::cout << "started\n";
    if (books.empty()) {
        std::cerr << "невозможно создать список: пустой вектор\n";
        return;
    }

    if (books.size() == 1) {
        dNode* newNode = new dNode(books[0]);
        if (find(newNode->book.key)) {
            std::cerr << "книга " << newNode->book.title << " с ключом "
<< newNode->book.key << "уже существует!\n";
            return;
        }

        this->head = newNode;
        return;
    }

    dNode* temp = new dNode(books[0]);
    this->head = temp;
    for (int i{1}; i < books.size() - 1; i++) {
        dNode* newNode = new dNode(books[i]);
        if (find(newNode->book.key)) {
            std::cerr << "книга с ключом " << newNode->book.key << "уже
существует!\n";
            delete newNode;
            continue;
        }

        temp->next = newNode;
        newNode->prev = temp;
        temp = temp->next;
    }

    dNode* newNode = new dNode(books[books.size() - 1]);
    if (find(newNode->book.key)) {
        std::cerr << "книга с ключом " << newNode->book.key << "уже
существует!\n";
        delete newNode;
    }
}
```

```

temp->next    = newNode;
tail         = newNode;
newNode->prev = temp;
temp         = temp->next;

if (head == nullptr || tail == nullptr) {
    std::cerr << "невозможно создать список: неизвестная ошибка\n";
}
}

void dList::pushFront(Book book) {
    dNode* newHead = new dNode(book);
    newHead->next   = head;
    head->prev      = newHead;
    head           = newHead;
}

void dList::printBook(dNode* node) {
    std::cout << "Инвентарный номер: " << node->book.key << '\n';
    std::cout << "Название: \"" << node->book.title << "\"\n";
    std::cout << "Автор: " << node->book.author << '\n';
    std::cout << "Год издания: " << node->book.year << '\n';

    std::cout << "Области знаний: ";
    Field* field = node->book.fields->head;
    if (!field) {
        std::cout << "не найдено\n";
        return;
    }

    std::cout << field->field;
    field = field->next;
    while (field != nullptr) {
        std::cout << ", " << field->field;
        field = field->next;
    }
    std::cout << "\n\n";
}

void dList::printCatalog(bool backwards) {
    std::cout << "Каталог книг. \n\n";

    if (!backwards) {
        dNode* node = head;
        while (node != nullptr) {

```

```

        printBook(node);
        node = node->next;
    }
} else {
    dNode* node = tail;
    while (node != nullptr) {
        printBook(node);
        node = node->prev;
    }
}
}

void dList::replaceAfter(int num) {
    if (head == nullptr) {
        std::cerr << "невозможно переместить узлы: список пуст\n";
        return;
    }

    if (num <= 1) {
        std::cerr << "невозможно переместить узлы: входной параметр некорректен\n";
        return;
    }

    if (num >= size()) {
        std::cerr << "невозможно переместить узлы: входной параметр больше или равен длины списка\n";
        return;
    }

    dNode* newHead = head;
    for (int i{}; i < num - 1; i++) {
        newHead = newHead->next;
    }

    dNode* newTail = newHead->prev;
    newTail->next = nullptr;
    tail->next = head;
    head->prev = tail;
    tail = newHead->prev;
    newHead->prev = nullptr;
    head = newHead;
}

dNode* dList::find(int key) {
    dNode* node = head;

```

```

        while (node != nullptr) {
            if (node->book.key == key) {
                return node;
            }

            node = node->next;
        }

        return nullptr;
    }

    int dList::size() {
        int size{};
        dNode* temp = head;
        while (temp != nullptr) {
            size++;
            temp = temp->next;
        }

        return size;
    };

```

### 3. Код файла d\_node.h

```

#include <string>
#include "book.h"

struct dNode {
    Book    book;
    dNode* prev;
    dNode* next;

    dNode(Book book);
};

```

### 4. Код файла d\_node.cpp

```

#include "../headers/d_node.h"

dNode::dNode(Book book) {
    this->book = book;
    this->next = nullptr;
    this->prev = nullptr;
}

```

### 5. Код файла book.h

```

#include <string>
#include "fields.h"

struct Book {
    int          key, year;
    std::string author, title;
    Fields*      fields;

    Book();
    Book(
        int          key,
        int          year,
        std::string  author,
        std::string  title,
        std::vector<std::string> fields
    );
};

```

6. Код файла book.cpp

```

#include "../headers/book.h"

```

```

Book::Book() {
    this->key    = 0;
    this->year   = 0,
    this->author = " ",
    this->title  = " ";
    this->fields = nullptr;
}

```

```

Book::Book(int key, int year, std::string author, std::string title,
std::vector<std::string> fields) {
    this->key    = key;
    this->year   = year;
    this->author = author;
    this->title  = title;
    this->fields = new Fields(fields);
}

```

7. Код файла fields.h

```

#include "field.h"
#include <vector>

```

```

struct Fields {
    Field* head;
}

```



```

    Fields(std::vector<std::string> fields);
};

```

#### 8. Код файла fields.cpp

```

#include "../headers/fields.h"
#include <iostream>

```

```

Fields::Fields(std::vector<std::string> fields) {
    if (fields.empty()) {
        std::cerr << "невозможно создать список: пустой вектор\n";
        return;
    }

    if (fields.size() == 1) {
        this->head = new Field(fields[0]);
        return;
    }

    Field* temp = new Field(fields[0]);
    this->head = temp;
    for (int i{1}; i < fields.size(); i++) {
        temp->next = new Field(fields[i]);
        temp = temp->next;
    }

    if (head == nullptr) {
        std::cerr << "невозможно создать список: неизвестная ошибка\n";
    }
}

```

#### 9. Код файла field.h

```

#include <string>

struct Field {
    std::string field;
    Field* next;

    Field(std::string field);
};

```

#### 10. Код файла d\_list.cpp

```

#include "../headers/field.h"

Field::Field(std::string field) {
    this->field = field;
}

```

```
    this->next = nullptr;
}
```

11. Код файла main.cpp

```
#include "headers/d_list.h"
#include <iostream>

int main() {
    std::vector<Book> books = {
        Book(
            1,
            1851,
            "Герман Мелвилл",
            "Моби Дик, или Белый Кит",
            std::vector<std::string> {
                "Цетология",
                "Судоходство",
                "Выживание в дикой природе",
            }
        ),
        Book(
            2,
            1866,
            "Федор Достоевский",
            "Преступление и наказание",
            std::vector<std::string> {
                "Холодное оружие",
                "Криминалистика",
            }
        ),
        Book(
            3,
            2005,
            "Кормак Маккарти",
            "Старикам тут не место",
            std::vector<std::string> {
                "Психология",
                "Криминалистика",
            }
        ),
        Book(
            4,
            1865,
            "Лев Толстой",
            "Война и мир",
            std::vector<std::string> {
```

```

        "Гидрология",
        "История",
        "Взаимоотношения"
    }
    ),
};

dList* list = new dList(books);

list->printCatalog();

list->replaceAfter(5);

list->printCatalog();
}

```

#### 4.2.6. Тестирование

1. Исходный список содержит информацию о 4-х произвольных книгах. Переносятся в начало все книги, начиная со 2.

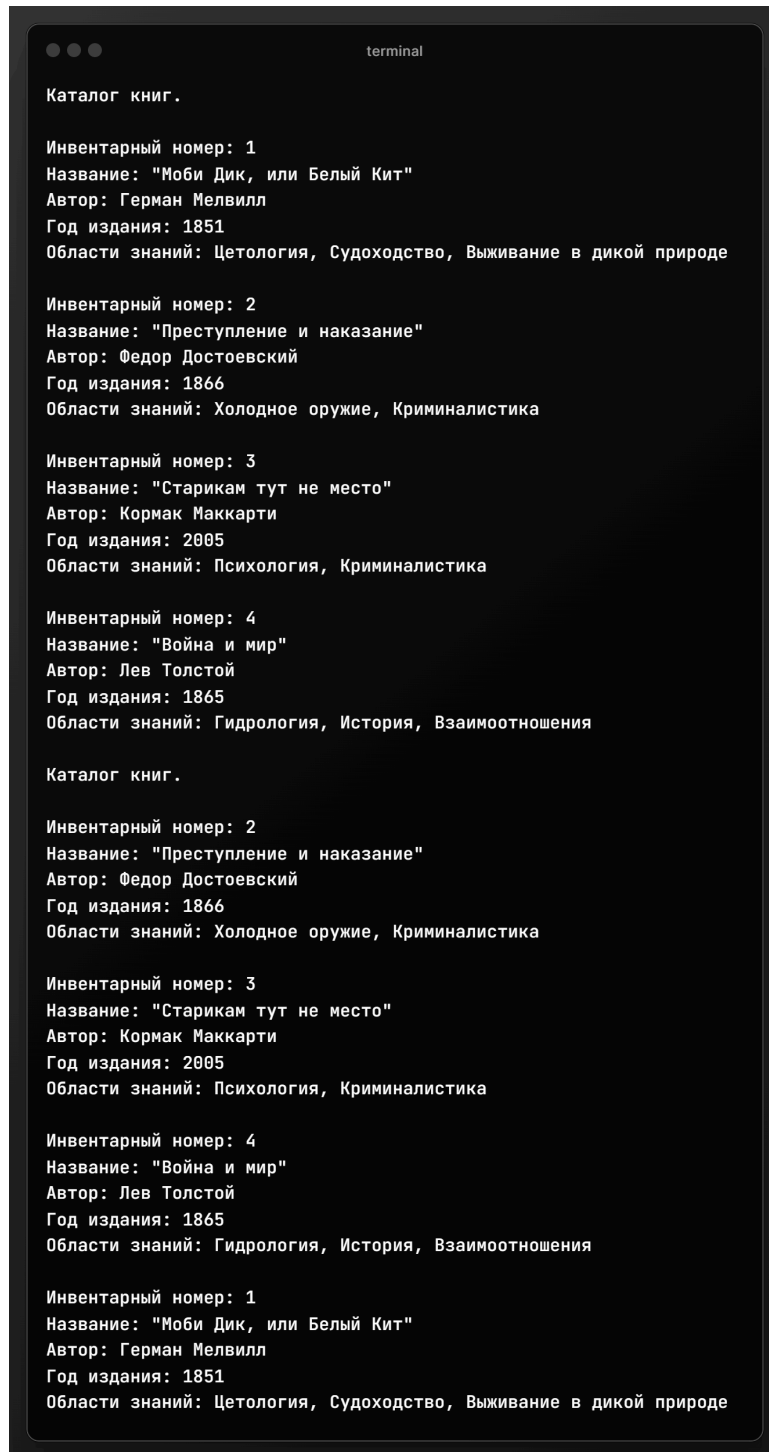


Рисунок 10 – Тест №1

2. Исходный список содержит информацию о тех же самых книгах. Производится попытка перенести в начало книги, начиная с 5.

```
terminal

Каталог книг.

Инвентарный номер: 1
Название: "Моби Дик, или Белый Кит"
Автор: Герман Мелвилл
Год издания: 1851
Области знаний: Цетология, Судоходство, Выживание в дикой природе

Инвентарный номер: 2
Название: "Преступление и наказание"
Автор: Федор Достоевский
Год издания: 1866
Области знаний: Холодное оружие, Криминалистика

Инвентарный номер: 3
Название: "Старикам тут не место"
Автор: Кормак Маккарти
Год издания: 2005
Области знаний: Психология, Криминалистика

Инвентарный номер: 4
Название: "Война и мир"
Автор: Лев Толстой
Год издания: 1865
Области знаний: Гидрология, История, Взаимоотношения

невозможно переместить узлы: входной параметр больше или равен длины списка
Каталог книг.

Инвентарный номер: 1
Название: "Моби Дик, или Белый Кит"
Автор: Герман Мелвилл
Год издания: 1851
Области знаний: Цетология, Судоходство, Выживание в дикой природе

Инвентарный номер: 2
Название: "Преступление и наказание"
Автор: Федор Достоевский
Год издания: 1866
Области знаний: Холодное оружие, Криминалистика

Инвентарный номер: 3
Название: "Старикам тут не место"
Автор: Кормак Маккарти
Год издания: 2005
Области знаний: Психология, Криминалистика

Инвентарный номер: 4
Название: "Война и мир"
Автор: Лев Толстой
Год издания: 1865
Области знаний: Гидрология, История, Взаимоотношения
```

Рисунок 11 – Тест №1

## 5. Выводы

В результате выполнения самостоятельной работы были получены умения и навыки по созданию динамически формируемых структур данных средствами языка C++, а также применению линейных списков в реализации алгоритмов.

## **6. Список информационных источников**

1. Структуры данных: связный список / Хабр // habr.com URL: <https://habr.com/ru/articles/717572/> (дата обращения: 06.05.24).
2. Linked List Data Structure – GeeksforGeeks // geeksforgeeks.-com URL: <https://www.geeksforgeeks.org/data-structures/linked-list/> (дата обращения: 06.05.24)