МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технологический университет «СТАНКИН»
(ФГБОУ ВО «МГТУ «СТАНКИН»)

Институт информационных систем и технологий

Кафедра информационных технологий и вычислительных систем

КУРСОВАЯ РАБОТА

ПО ДИСЦИПЛИНЕ

«СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ»

СТУДЕНТА *2* КУРСА *Бакалавриата* ГРУППЫ *ИДБ-21-02*

ТЕРЕНТЬЕВОЙ ЕКАТЕРИНЫ СЕРГЕЕВНЫ

ТЕМА РАБОТЫ

Последовательность предложений

Направление: 09.03.01 Информатика и вычислительная техника

Профиль подготовки: «Программное обеспечение вычислительной техники и автоматизированных систем»

Отчет сдан «_____» _____20___г.

Оценка _____

Преподаватель Лакунина О.Н., ст. преподаватель _____
(подпись)

МОСКВА 2022

Оглавление

Задание на курсовую работу

Написать программу, реализующую логическую структуру данных – последовательность предложений в односвязном списке. Программа должна работать в диалоговом режиме. Каждая операция должна быть реализована в виде отдельной функции.

Последовательность и предложения должны быть реализованы на базе структуры хранения односвязный список.

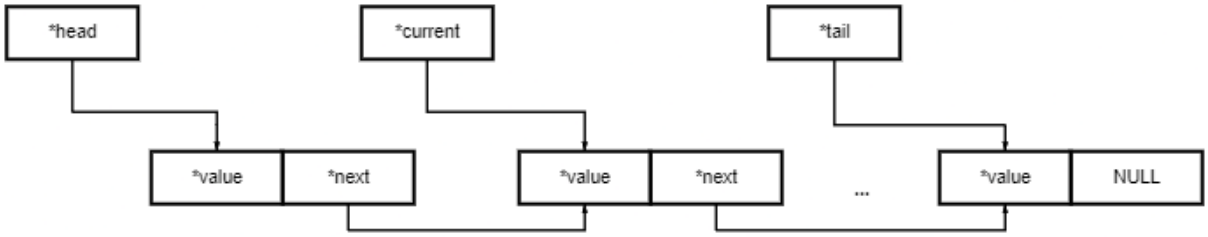
Написать отчёт по курсовой работе.

Описание структуры данных

Последовательность – линейно упорядоченный набор элементов, ее элементы в каждый момент времени можно разделить на 2 части: прочитанную и непрочитанную.

Схема логической структуры «Последовательность»

Схема физической структуры «Последовательность»



*head – указатель на начало списка

value – данные

*next - указатель на следующий элемент

*current - указатель на текущий элемент

*tail – указатель на конец списка

Предложение – это односвязный список слов.

Схема логической структуры «Предложение»

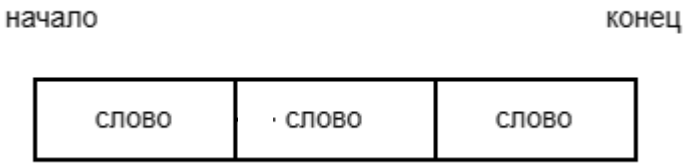
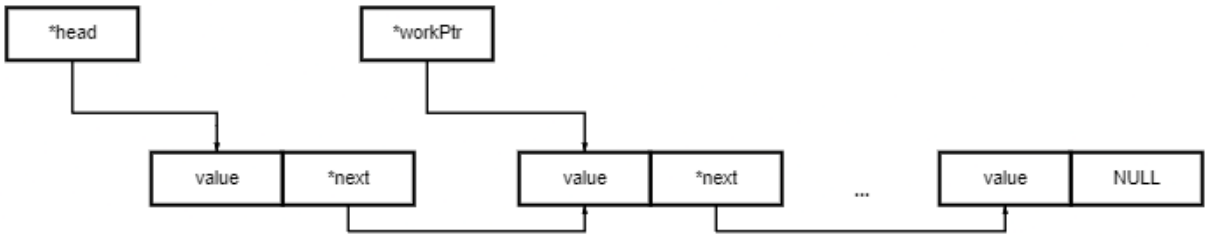


Схема физической структуры «Предложение»



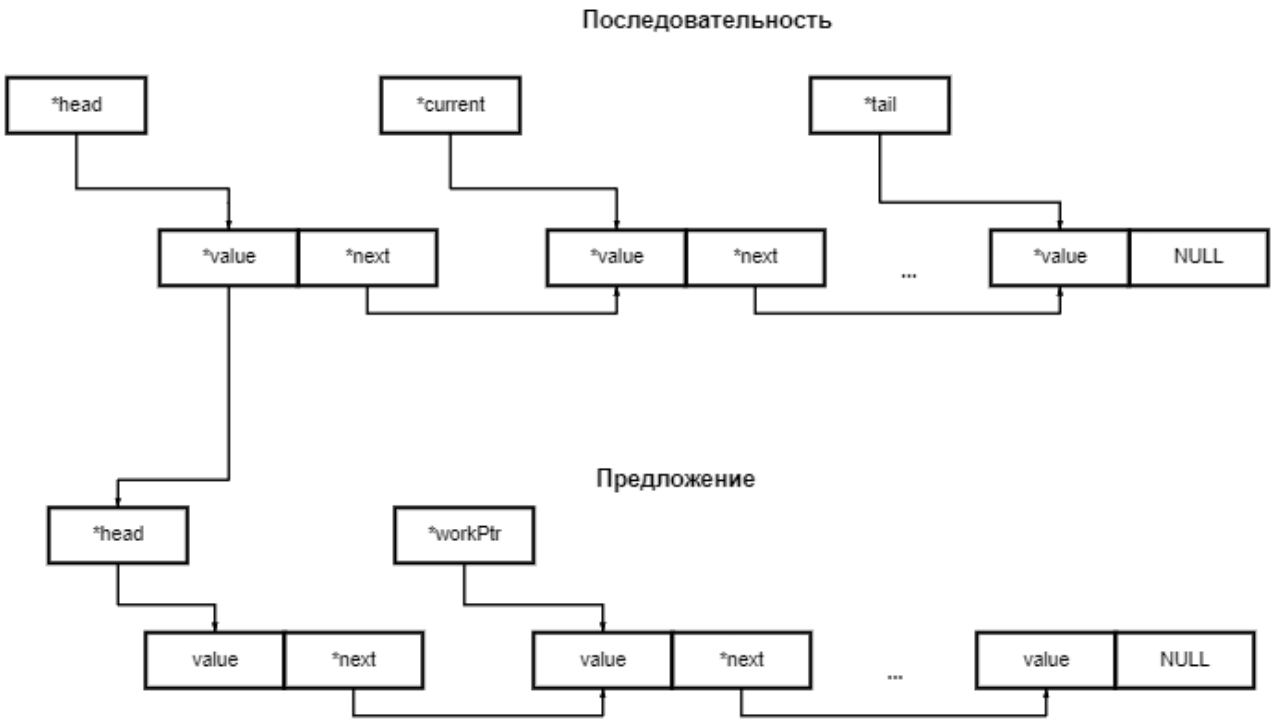
*head – указатель на начало списка

value - данные

*next - указатель на следующий элемент

*workPtr – рабочий указатель

Конечная схема реализуемой структуры данных



Список реализуемых функций для внешней структуры (последовательность)

1. Начать работу с последовательностью предложений
2. Сделать последовательность пустой
3. Проверить последовательность пуста/не пуста
4. Показать значение очередного элемента
5. Пропустить очередной элемент
6. Прочитать очередной элемент последовательности
7. Изменить значение очередного элемента
8. Удалить значение очередного элемента
9. Добавить элемент в конец последовательности
10. Установить указатель очередного элемента в начало последовательности
11. Проверить есть ли непрочитанные элементы?
12. Распечатать последовательность предложений
13. Конец работы с последовательностью предложений

Список реализуемых функций для внутренней структуры (предложения)

1. Начать работу с предложением
2. Сделать предложение пустым
3. Проверить предложение пустое или не пустое
4. Установить рабочий указатель на первое слово предложения
5. Проверить, рабочий указатель в конце предложения?
6. Передвинуть рабочий указатель вперед на одно слово
7. Показать значение слова за указателем
8. Удалить слово за указателем
9. Взять слово за указателем
10. Изменить слово за указателем
11. Добавить слово за указателем
12. Распечатать предложение
13. Внести сохраненное слово за указателем
14. Ввести новое предложение
15. Закончить работу с предложением

Описание структур на языке С

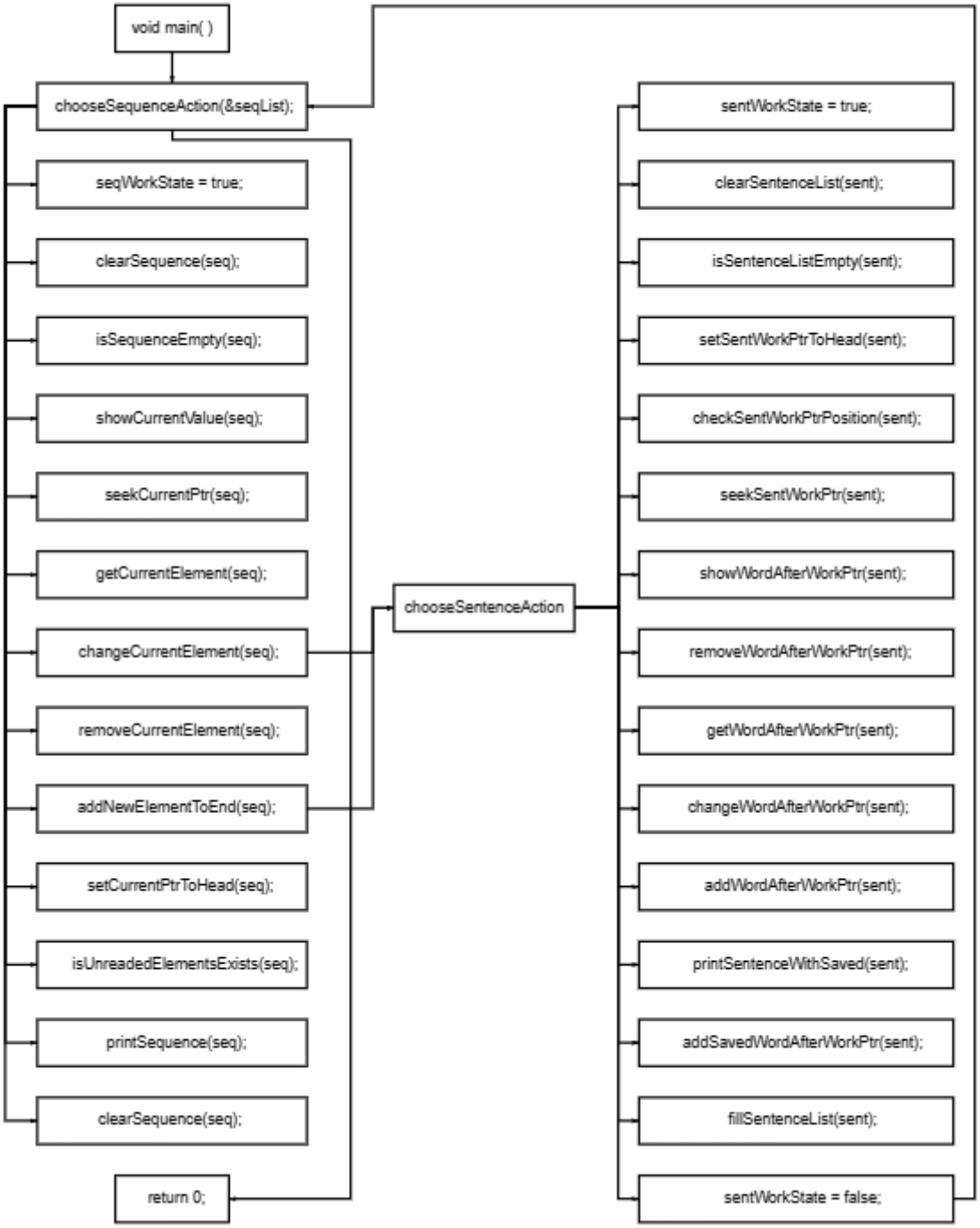
```
typedef struct WordN { // структура для хранения элемента предложения
    char* value; // указатель на массив типа char
    struct WordN* next; // переменная для хранения указателя на следующий
элемент предложения
} WordNode; // имя структурного типа элемента предложения


typedef struct SentL { // структура для хранения предложения
    WordNode* head; // указатель на начало предложения
    WordNode* workPtr; // указатель на рабочий указатель в предложении
} SentList; // имя структурного типа предложения


typedef struct SeqN { // структура для хранения элемента последовательности
    SentList* value; // указатель на последовательность предложений
    struct SeqN* next; // переменная для хранения указателя на следующий
элемент последовательности
} SeqNode; // имя структурного типа элемента последовательности


typedef struct SeqL { // структура для хранения последовательности
    SeqNode* head; // указатель на начало последовательности
    SeqNode* tail; // указатель на конец последовательности
    SeqNode* current; // указатель на очередной элемент последовательности
} SeqList; // имя структурного типа последовательности
```

Схема вызовов функций



Список функций и их назначения

Для внешней структуры (последовательность):

- void clearSequence(SeqList* seq) - очистить последовательность, принимает указатель на структуру, хранящую указатель на последовательность, ничего не возвращает
- void isSequenceEmpty(SeqList* seq) - проверить на пустоту последовательность, принимает указатель на структуру, хранящую указатель на последовательность, ничего не возвращает
- void showCurrentValue(SeqList* seq) - показать значение очередного элемента последовательности, принимает указатель на структуру, хранящую указатель на последовательность, ничего не возвращает
- void seekCurrentPtr(SeqList* seq) - пропустить очередной элемент последовательности, принимает указатель на структуру, хранящую указатель на последовательность, ничего не возвращает
- void getCurrentElement(SeqList* seq) - прочитать очередной элемент последовательности, принимает указатель на структуру, хранящую указатель на последовательность, ничего не возвращает
- void changeCurrentElement(SeqList* seq) - изменить значение очередного элемента последовательности, принимает указатель на структуру, хранящую указатель на последовательность, ничего не возвращает
- void removeCurrentElement(SeqList* seq) - удалить значение очередного элемента последовательности, принимает указатель на структуру, хранящую указатель на последовательность, ничего не возвращает
- void addNewElementToEnd(SeqList* seq) - добавить элемент в конец последовательности, принимает указатель на структуру, хранящую указатель на последовательность, ничего не возвращает
- void setCurrentPtrToHead(SeqList* seq) - установить указатель очередного элемента в начало последовательности, принимает указатель на структуру, хранящую указатель на последовательность, ничего не возвращает
- void isUnreadedElementsExists(SeqList* seq) - проверить, есть ли непрочитанные элементы, принимает указатель на структуру, хранящую указатель на последовательность, ничего не возвращает
- void printSequence(SeqList* seq) - распечатать последовательность предложений, принимает указатель на структуру, хранящую указатель на последовательность, ничего не возвращает

Для внутренней структуры (предложения):

- void clearSentenceList(SentList* sent) - сделать предложение пустым, принимает указатель на структуру, хранящую указатель на предложение, ничего не возвращает
- void isSentenceListEmpty(SentList* sent) - проверить на пустоту предложение, принимает указатель на структуру, хранящую указатель на предложение, ничего не возвращает
- void setSentWorkPtrToHead(SentList* sent) - установить рабочий указатель на первое слово предложения, принимает указатель на структуру, хранящую указатель на предложение, ничего не возвращает
- void checkSentWorkPtrPosition(SentList* sent) - проверить, находится ли указатель в конце предложения, принимает указатель на структуру, хранящую указатель на предложение, ничего не возвращает
- void seekSentWorkPtr(SentList* sent) - передвинуть рабочий указатель вперед на одно слово в предложении, принимает указатель на структуру, хранящую указатель на предложение, ничего не возвращает
- void showWordAfterWorkPtr(SentList* sent) - показать значение слова за указателем в предложении, принимает указатель на структуру, хранящую указатель на предложение, ничего не возвращает
- void removeWordAfterWorkPtr(SentList* sent) - удалить слово за указателем в предложении, принимает указатель на структуру, хранящую указатель на предложение, ничего не возвращает
- void getWordAfterWorkPtr(SentList* sent) - взять слово за указателем в предложении, принимает указатель на структуру, хранящую указатель на предложение, ничего не возвращает
- void changeWordAfterWorkPtr(SentList* sent) - изменить слово за указателем в предложении, принимает указатель на структуру, хранящую указатель на предложение, ничего не возвращает
- void addWordAfterWorkPtr(SentList* sent) - добавить слово за указателем в предложении, принимает указатель на структуру, хранящую указатель на предложение, ничего не возвращает
- void printSentenceWithSaved(SentList* sent) - распечатать предложение, принимает указатель на структуру, хранящую указатель на предложение, ничего не возвращает
- void addSavedWordAfterWorkPtr(SentList* sent) - внести сохраненное слово за указателем в предложении, принимает указатель на структуру, хранящую указатель на предложение, ничего не возвращает
- void fillSentenceList(SentList* sent) - ввести новое предложение, принимает указатель на структуру, хранящую указатель на предложение, ничего не возвращает

Исходный код программы с комментариями

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>
#include <windows.h>

HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE); // для изменения цвета в сообщениях об ошибке

//* === РЕАЛИЗАЦИЯ ПРЕДЛОЖЕНИЯ === */
typedef struct WordN {
    char* value;
    struct WordN* next;
} WordNode;

// Предложение (односвязный список слов)
typedef struct SentL {
    WordNode* head;
    WordNode* workPtr;
} SentList;

WordNode* savedWord; // для взятия элемента (9 действие)

bool sentWorkState = false; // значение для действий начала и завершения работы

void printWithColor(const char* str, WORD num) {
    SetConsoleTextAttribute(hConsole, num); // Устанавливаем аттрибут цвета
    printf(str);
    SetConsoleTextAttribute(hConsole, 15);
} // функция для изменения цвета

void printMessage(const char* str) {
    printWithColor(str, 1); // печать с синим цветом
} // функция для печати сообщения для пользователя

// Распечатать предложение
void printSentence(SentList* sent) {
    printWithColor("{", 9);
    if (sent->head == NULL) {
        printWithColor("= пусто =", 14);
    }
```

```
    else {
        WordNode* ptr = sent->head;
        while (ptr != NULL) {
            if (ptr == sent->workPtr) {
                printWithColor("*", 14); // ставим значок указателя
            }
            printWithColor("\"", 14);
            if (ptr->value != NULL) {
                printf("%s", ptr->value); // выводим введенную строку символов
            }
            printWithColor("\"", 14);
            ptr = ptr->next;
            if (ptr != NULL) {
                printf(" ");
            }
        }
    }
    printWithColor("}", 9);
}

void printSentenceWithSaved(SentList* sent) {
    printf("\n");
    printSentence(sent);
    printf("\n----------------------------\n");
    if (savedWord != NULL) {
        printf("[%s]\n", savedWord->value);
    }
    else {
        printf("Нет сохраненного элемента\n");
    }
}

char* readWordFromConsole() { // функция чтения строки любой длины через
буфер
#define BUFSIZE 200
    char* input = NULL;
    char tmpbuf[BUFSIZE] = "";
    void* alloc; // указатель на массив символов
    size_t inputlen = 0, tmplen = 0;
    do {
        if (fgets(tmpbuf, BUFSIZE, stdin) == NULL) {
            break;
        }
        tmplen = strlen(tmpbuf);
```

```
        alloc = realloc(input, inputlen + tmplen + 1); // меняем размер массива, что бы поместились новые символы
        if (alloc == NULL) { // ошибка при выделении памяти
            break;
        }
        input = (char*)alloc; // преобразуем тип указателя
        strcpy_s(input + inputlen, tmplen + 1, tmpbuf); копируем строку из буфера в выделенную область
        inputlen += tmplen;
    } while (tmplen == BUFSIZE - 1 && tmpbuf[BUFSIZE - 2] != '\n');
    if (input != NULL) {
        input[inputlen - 1] = '\0'; // принудительно завершаем строку нулем, что бы убрать символ новой строки
    }
    return input;
}

// 2
void clearSentenceList(SentList* sent) { // отчищаем предложение
    sent->workPtr = NULL; // обнуляем рабочий указатель
    WordNode* ptr = NULL; // новый указатель для прохода по структуре
    while (sent->head != NULL) {
        ptr = sent->head;
        sent->head = sent->head->next;
        free(ptr->value); // отчищаем массив символов
        free(ptr); // отчищаем структуру
    }
    printSentenceWithSaved(sent);
}

// 3
void isSentenceListEmpty(SentList* sent) {
    if (sent->head != NULL) {
        printMessage("предложение не пустое!\n");
    }
    else {
        printMessage("предложение пустое!\n");
    }
    printSentenceWithSaved(sent);
}

// 4
void setSentWorkPtrToHead(SentList* sent) {
    if (sent->head == NULL) {
```

```
        printMessage("Предложение пустое!\n");
        return;
    }
    sent->workPtr = sent->head;
    printSentenceWithSaved(sent);
}

// 5
void checkSentWorkPtrPosition(SentList* sent) {
    if (sent->head == NULL) {
        printMessage("Предложение пустое!\n");
        return;
    }
    if (sent->workPtr == NULL) {
        printMessage("Рабочий указатель не установлен!\n");
    }
    else {
        if (sent->workPtr->next == NULL) {
            printMessage("Рабочий указатель в конце предложения.\n");
        }
        else {
            printMessage("Рабочий указатель не в конце предложения.\n");
        }
    }
    printSentenceWithSaved(sent);
}

// 6
void seekSentWorkPtr(SentList* sent) {
    if (sent->head == NULL) {
        printMessage("Предложение пустое!\n");
        return;
    }
    if (sent->workPtr == NULL) {
        printMessage("Рабочий указатель не установлен!\n");
    }
    else {
        if (sent->workPtr->next != NULL) { // проверяем если ли следующий
элемент
            sent->workPtr = sent->workPtr->next;
        }
        else {
            printMessage("Достигнут конец предложения\n");
        }
```

```
    }
    printSentenceWithSaved(sent);
}

// 7
void showWordAfterWorkPtr(SentList* sent) {
    if (sent->head == NULL) {
        printMessage("Предложение пустое!\n");
        return;
    }
    if (sent->workPtr == NULL) {
        printMessage("Рабочий указатель не установлен!\n");
    }
    else {
        if (sent->workPtr->next != NULL) { // проверяем если ли следующий
элемент
            printf("[%s]\n", sent->workPtr->next->value);
        }
        else {
            printMessage("После рабочего указателя нет значений\n");
        }
    }
    printSentenceWithSaved(sent);
}

// 8
void removeWordAfterWorkPtr(SentList* sent) {
    if (sent->head == NULL) {
        printMessage("Предложение пустое!\n");
        return;
    }
    if (sent->workPtr == NULL) {
        printMessage("Рабочий указатель не установлен!\n");
    }
    else {
        WordNode* ptr = NULL; // указатель на удаляемый элемент
        if (sent->workPtr->next != NULL) {
            ptr = sent->workPtr->next; // присваиваем элемент
            sent->workPtr->next = ptr->next; // сцепляем элементы без удаляемого
            free(ptr->value); // удаляем массив символов
            free(ptr); // удаляем структуру
        }
        else {
            printMessage("За рабочим указателем нет элементов\n");
```

```
        }
    }
    printSentenceWithSaved(sent);
}

// 9
void getWordAfterWorkPtr(SentList* sent) {
    if (sent->head == NULL) {
        printMessage("Предложение пустое!\n");
        return;
    }
    if (sent->workPtr == NULL) {
        printMessage("Рабочий указатель не установлен!\n");
    }
    else {
        if (sent->workPtr->next != NULL) {
            savedWord = sent->workPtr->next; // сохраняем указатель на элемент
            sent->workPtr->next = savedWord->next; // сцепляем элементы без
сохраненного
            savedWord->next = NULL; // обнуляем указатель на следующий элемен
        }
        else {
            printMessage("За рабочим указателем нет элементов\n");
        }
    }
    printSentenceWithSaved(sent);
}

// 10
void changeWordAfterWorkPtr(SentList* sent) {
    if (sent->head == NULL) {
        printMessage("Предложение пустое!\n");
        return;
    }
    if (sent->workPtr == NULL) {
        printMessage("Рабочий указатель не установлен!\n");
    }
    else {
        printf("Введите слово: \n");
        char* word = readWordFromConsole();
        if (sent->workPtr->next != NULL) {
            sent->workPtr->next->value = word;
        }
        else {
```

```
            printMessage("За рабочим указателем нет элементов\n");
        }
    }
    printSentenceWithSaved(sent);
}

// 11
void addWordAfterWorkPtr(SentList* sent) {
    printf("Введите слово: \n");
    char* word = readWordFromConsole();
    WordNode* tmp = (WordNode*)malloc(sizeof(WordNode));
    if (!tmp) {
        printMessage("Ошибка при попытке создать элемент предложения\n");
        return;
    }
    tmp->value = word;
    tmp->next = NULL;

    if (sent->head == NULL) {
        sent->head = tmp;     // when linked list is empty
        sent->workPtr = sent->head;
    }
    else {
        if (sent->workPtr == NULL) {
            free(tmp->value);
            free(tmp);
            printMessage("Рабочий указатель не установлен!\n");
            return;
        }
        WordNode* ptr = sent->head;
        while (ptr != sent->workPtr && ptr != NULL) {
            ptr = ptr->next;
        }
        if (ptr == NULL) {
            free(tmp);
            printMessage("Ошибка при попытке добавить элемент в список\n");
            return;
        }
        tmp->next = sent->workPtr->next;
        sent->workPtr->next = tmp;
    }
    printSentenceWithSaved(sent);
}
```

```
// 13
void addSavedWordAfterWorkPtr(SentList* sent) {
    if (savedWord == NULL) {
        printMessage("Нет сохраненных слов!\n");
        return;
    }
    if (sent->head == NULL) {
        sent->head = savedWord;
        sent->workPtr = sent->head;
        savedWord = NULL;
    }
    else {
        if (sent->workPtr == NULL) {
            printMessage("Рабочий указатель не установлен!\n");
        }
        else {
            WordNode* ptr = sent->head;
            while (ptr != sent->workPtr && ptr != NULL) {
                ptr = ptr->next;
            }
            if (ptr == NULL) {
                printMessage("Ошибка при попытке добавить элемент в список\n");
                return;
            }
            savedWord->next = sent->workPtr->next;
            sent->workPtr->next = savedWord;
            savedWord = NULL;
        }
    }
    printSentenceWithSaved(sent);
}

void addWordToEnd(SentList* sent) {
    printf("Введите слово: \n");
    char* word = readWordFromConsole();
    if (!word) {
        printMessage("Ошибка при чтении слова\n");
        return;
    }
    WordNode* tmp = (WordNode*)malloc(sizeof(WordNode));
    if (!tmp) {
        printMessage("Ошибка при попытке создать элемент предложения\n");
        return;
    }
```

```
    tmp->value = word;
    tmp->next = NULL;

    if (sent->head == NULL) {
        sent->head = tmp;     // when linked list is empty
        sent->workPtr = sent->head;
    }
    else {
        WordNode* ptr = sent->head;
        while (ptr->next != NULL) {
            ptr = ptr->next;
        }
        ptr->next = tmp;
    }
    printSentenceWithSaved(sent);
}

// 14
void fillSentenceList(SentList* sent) { // Функция для быстрого заполения
предложения
    printf("Ввод предложения: \n");
    bool loop = true;
    char s[3], c;
    while (loop) {
        addWordToEnd(sent);
        printf("Ввести еще одно? (y/n)\n");
        scanf_s("%s", &s, 3);
        while ((c = getchar()) != '\n') {} //зачищаем все до \n, чтобы очистить
буфер
        if (s[0] != 'y' && s[0] != 'Y') {
            loop = false;
        }
    }
    printSentenceWithSaved(sent);
}

void printSentenceMenu() {
    printf("\n");
    printf("Выберите действие:\n");
    printf(" 1. Начать работу с предложением\n");
    printf(" 2. Сделать предложение пустым\n");
    printf(" 3. Проверить предложение пустое или не пустое\n");
    printf(" 4. Установить рабочий указатель на первое слово предложения\n");
    printf(" 5. Проверить, рабочий указатель в конце предложения?\n");
```

```
    printf(" 6. Передвинуть рабочий указатель вперед на одно слово\n");
    printf(" 7. Показать значение слова за указателем\n");
    printf(" 8. Удалить слово за указателем\n");
    printf(" 9. Взять слово за указателем\n");
    printf("10. Изменить слово за указателем\n");
    printf("11. Добавить слово за указателем\n");
    printf("12. Распечатать предложение\n");
    printf("13. Внести сохраненное слово за указателем\n"); // связано с 9
    printf("14. Ввести новое предложение\n");
    printf("15. Закончить работу с предложением\n");
    printf("\n");
}

SentList* chooseSentenceAction(SentList* sent) {
    if (sent == NULL) {
        sent = (SentList*)malloc(sizeof(SentList));
        if (sent == NULL) {
            printMessage("\nОшибка при попытке создать предложение!\n");
            return sent;
        }
        sent->head = sent->workPtr = NULL;
    }
    int loop = 1;
    while (loop) {
        char c;
        int a = 0;
        printSentenceMenu();
        scanf_s("%d", &a);
        printf("\n");
        while ((c = getchar()) != '\n') {} //зачищает все \n, чтобы очистить буфер
        if (a < 1 || a > 15) {
            printMessage("Выберете другое действие!\n");
            continue;
        }
        if (sentWorkState == false && a > 1 && a < 15) {
            printMessage("Начните работу с выполнения операции \"1. Начать
работу с предложением\"\n");
            continue;
        }
        switch (a) {
        case 1:
            sentWorkState = true;
            break;
        case 2:
```

```
    clearSentenceList(sent);
    break;
case 3:
    isSentenceListEmpty(sent);
    break;
case 4:
    setSentWorkPtrToHead(sent);
    break;
case 5:
    checkSentWorkPtrPosition(sent);
    break;
case 6:
    seekSentWorkPtr(sent);
    break;
case 7:
    showWordAfterWorkPtr(sent);
    break;
case 8:
    removeWordAfterWorkPtr(sent);
    break;
case 9:
    getWordAfterWorkPtr(sent);
    break;
case 10:
    changeWordAfterWorkPtr(sent);
    break;
case 11:
    addWordAfterWorkPtr(sent);
    break;
case 12:
    printSentenceWithSaved(sent);
    break;
case 13:
    addSavedWordAfterWorkPtr(sent);
    break;
case 14:
    fillSentenceList(sent);
    break;
case 15:
    sentWorkState = false;
    loop = 0;
    break;
default:
    break;
```

```
        }
    }
    return sent;
}
/* --- КОНЕЦ РЕАЛИЗАЦИИ ПРЕДЛОЖЕНИЯ --- */

//* === РЕАЛИЗАЦИЯ ПОСЛЕДОВАТЕЛЬНОСТИ === */

typedef struct SeqN {
    SentList* value;
    struct SeqN* next;
} SeqNode;

// Последовательность
typedef struct SeqL {
    SeqNode* head;
    SeqNode* tail;
    SeqNode* current;
} SeqList;

SeqNode* savedSeq; // для взятия элемента (6 действие)

bool seqWorkState = false;

// 11 Распечатать последовательность
void printSequence(SeqList* seq) {
    if (seq->head == NULL) {
        printMessage("= последовательность пуста =\n");
    }
    else {
        SeqNode* ptr = seq->head;
        printWithColor("(", 12); // красный
        while (ptr != NULL) {
            if (ptr == seq->current) {
                printWithColor("$", 9); // синий
            }
            printSentence(ptr->value);
            ptr = ptr->next;
            if (ptr != NULL) {
                printf(" ");
            }
        }
        printWithColor(")\n", 12);
    }
```

```
}

// 2
void clearSequence(SeqList* seq) {
    seq->current = NULL;
    seq->tail = NULL;
    SeqNode* ptr = NULL;
    while (seq->head != NULL) {
        ptr = seq->head;
        seq->head = seq->head->next;
        clearSentenceList(ptr->value);
        free(ptr->value);
        free(ptr);
    }
    printSequence(seq);
}

// 3
void isSequenceEmpty(SeqList* seq) {
    if (seq->head == NULL) {
        printMessage("Последовательность пуста!\n");
        return;
    }
    printMessage("Последовательность не пуста!\n\n");

    printSequence(seq);
}

// 4
void showCurrentValue(SeqList* seq) {
    if (seq->head == NULL) {
        printMessage("Последовательность пустая!\n");
        return;
    }
    if (seq->current == NULL) {
        printMessage("Указатель очередного элемента находится в конце!\n");
    }
    else {
        printf("Значение:\n");
        printSentence(seq->current->value);
    }
    printf("\n\n");
    printSequence(seq);
}
```

```
// 5
void seekCurrentPtr(SeqList* seq) {
   if (seq->head == NULL) {
      printMessage("Последовательность пустая!\n");
      return;
   }
   if (seq->current == NULL) {
      printMessage("Указатель очередного элемента находится в конце!\n");
   }
   else {
      seq->current = seq->current->next;
   }
   printSequence(seq);
}

// 6 //
void getCurrentElement(SeqList* seq) {
   if (seq->head == NULL) {
      printMessage("Последовательность пустая!\n");
      return;
   }
   if (seq->current == NULL) {
      printMessage("Указатель очередного элемента находится в конце!\n");
   }
   else {
      savedSeq = seq->current;
      seq->current = seq->current->next;
      if (savedSeq == NULL) {
         printMessage("Указатель очередного элемента находится в конце!\n");
      }
      else {
         printf("Прочитанное значение:\n");
         printSentence(savedSeq->value);
         printf("\n\n");
      }
   }
   printSequence(seq);
}

// 7
void changeCurrentElement(SeqList* seq) {
   if (seq->head == NULL) {
      printMessage("Последовательность пустая!\n");
```

```
        return;
    }
    if (seq->current == NULL) {
        printMessage("Указатель очередного элемента находится в конце!\n");
    }
    else {
        printf("Переход к работе с предложением ->\n");
        SentList* tmp = chooseSentenceAction(seq->current->value);
        if (tmp == NULL) {
            printMessage("Ошибка при попытке изменить элемент!\n");
            return;
        }
        seq->current->value = tmp;
    }
    printSequence(seq);
}

// 8 //
void removeCurrentElement(SeqList* seq) {
    if (seq->head == NULL) {
        printMessage("Последовательность пустая!\n");
        return;
    }
    if (seq->current == NULL) {
        printMessage("Указатель очередного элемента находится в конце!\n");
    }
    else { // при изменениях переподключаем head, current, tail
        SeqNode* tmp = seq->current;
        if (seq->current == seq->head) {
            seq->head = seq->head->next;
            if (seq->current == seq->tail) {
                seq->tail = seq->tail->next;
            }
            seq->current = seq->current->next;
        }
        else {
            SeqNode* ptr = seq->head;
            while (ptr->next != seq->current) {
                ptr = ptr->next;
            }
            ptr->next = seq->current->next;
            if (seq->current == seq->tail) {
                seq->current = ptr;
                seq->tail = ptr;
```

```
            }
            else {
                seq->current = seq->current->next;
            }
        }
        tmp->next = NULL;
        clearSentenceList(tmp->value);
        free(tmp->value);
        free(tmp);
    }
    printSequence(seq);
}

// 9 //
void addNewElementToEnd(SeqList* seq) {
    SentList* tmp = (SentList*)malloc(sizeof(SentList));
    if (tmp == NULL) {
        printMessage("Ошибка при попытке создать предложение!\n");
        return;
    }
    tmp->head = NULL;
    tmp->workPtr = NULL;
    printf("Переход к работе с предложением ->\n");
    SentList* tmpSentList = chooseSentenceAction(tmp);
    if (tmpSentList == NULL) {
        printMessage("Ошибка при попытке создать предложение!\n");
        clearSentenceList(tmp);
        free(tmp);
        return;
    }
    SeqNode* tmpSeqNode = (SeqNode*)malloc(sizeof(SeqNode));
    if (tmpSeqNode == NULL) {
        printMessage("Ошибка при попытке создать новый элемент
последовательности!\n");
        clearSentenceList(tmp);
        free(tmp);
        clearSentenceList(tmpSentList);
        free(tmpSentList);
        return;
    }
    tmpSeqNode->value = tmpSentList;
    tmpSeqNode->next = NULL;

    if (seq->head == NULL) {
```

```
        seq->head = tmpSeqNode;
        seq->tail = tmpSeqNode;
        seq->current = tmpSeqNode;
    }
    else {
        seq->tail->next = tmpSeqNode;
        seq->tail = tmpSeqNode;
    }
    printSequence(seq);
}

// 10
void setCurrentPtrToHead(SeqList* seq) {
    if (seq->head == NULL) {
        printMessage("Последовательность пустая!\n");
        return;
    }
    seq->current = seq->head;

    printSequence(seq);
}

// 11
void isUnreadedElementsExists(SeqList* seq) {
    if (seq->head == NULL) {
        printMessage("Последовательность пустая!\n");
        return;
    }

    if (seq->current == NULL) {
        printMessage("Непрочитанных элементов нет\n");
    }
    else {
        printMessage("Есть непрочитанные элементы\n");
    }
    printSequence(seq);
}

void printSequenceMenu() {
    printf("\n");
    printf("Выберите действие:\n");
    printf(" 1. Начать работу с последовательностью предложений\n");
    printf(" 2. Сделать последовательность пустой\n");
    printf(" 3. Проверить последовательность пуста/не пуста\n");
```

```
    printf(" 4. Показать значение очередного элемента\n");
    printf(" 5. Пропустить очередной элемент (передвинуть указатель вперед на
одну позицию)\n");
    printf(" 6. Прочитать очередной элемент последовательности\n"); // взять
элемент
    printf(" 7. Изменить значение очередного элемента\n");
    printf(" 8. Удалить значение очередного элемента\n"); // добавила т.к. не
было в списке операций, но в лекции 6 в списке числится ("Над всеми
логическими структурами данных могут выполняться следующие
операции:")
    printf(" 9. Добавить элемент в конец последовательности\n");
    printf("10. Установить указатель очередного элемента в начало
последовательности\n");
    printf("11. Проверить есть ли непрочитанные элементы?\n");
    printf("12. Распечатать последовательность предложений\n");
    printf("13. Конец работы с последовательностью предложений\n");
    printf(" 0. Завершить программу\n");
    printf("\n");
}

SeqList* chooseSequenceAction(SeqList* seq) {
    if (seq == NULL) {
        seq = (SeqList*)malloc(sizeof(SeqList));
        if (seq == NULL) {
            printMessage("\nОшибка при попытке создать
последовательность!\n");
            return seq;
        }
        seq->head = seq->tail = seq->current = NULL;
    }
    int loop = 1;
    while (loop) {
        char c;
        int a = 0;
        printSequenceMenu();
        scanf_s("%d", &a);
        printf("\n");
        while ((c = getchar()) != '\n') {} //зачищает все \n, чтобы очистить буфер
        if (a < 0 || a > 13) {
            printMessage("Выберете другое действие!\n");
            continue;
        }
        if (seqWorkState == false && a > 1 && a < 14) {
```

```
            printMessage("Начните работу с выполнения операции \"1. Начать
работу с последовательностью предложений\"\n");
            continue;
        }
        if (seqWorkState == true && a == 0) {
            printMessage("\nПеред завершеним программы выберите \"13. Конец
работы с последовательностью предложений\"\n");
            continue;
        }
        switch (a) {
        case 1:
            seqWorkState = true;
            break;
        case 2:
            clearSequence(seq);
            break;
        case 3:
            isSequenceEmpty(seq);
            break;
        case 4:
            showCurrentValue(seq);
            break;
        case 5:
            seekCurrentPtr(seq);
            break;
        case 6:
            getCurrentElement(seq);
            break;
        case 7:
            changeCurrentElement(seq);
            break;
        case 8:
            removeCurrentElement(seq);
            break;
        case 9:
            addNewElementToEnd(seq);
            break;
        case 10:
            setCurrentPtrToHead(seq);
            break;
        case 11:
            isUnreadedElementsExists(seq);
            break;
        case 12:
```

```
            printSequence(seq);
            break;
        case 13:
            clearSequence(seq);
            seqWorkState = false;
            break;
        case 0:
            loop = 0;
            break;
        default:
            break;
        }
    }
    return seq;
}
/* --- КОНЕЦ РЕАЛИЗАЦИИ ПОСЛЕДОВАТЕЛЬНОСТИ --- */

int main() {
    setlocale(LC_ALL, "Russian"); // подключаем русский язык

    SeqList seqList = { NULL, NULL, NULL }; //создаем структуру и
инициализируем все указатели значение NULL (head, tail, current)
    chooseSequenceAction(&seqList);
    return 0;
}
```