

**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-21-03* |
|  | | | *(уровень профессионального образования)* | |  | |

|  |
| --- |
| **НИКУЛИНА ДМИТРИЯ ЕВГЕНЬЕВИЧА** |
| *(ФИО)* |

ТЕМА РАБОТЫ

|  |
| --- |
| Последовательность деков предложений |

|  |  |
| --- | --- |
| Направление: | 09.03.01 Информатика и вычислительная техника |
| Профиль подготовки: | «Программное обеспечение средств вычислительной техники и автоматизированных систем» |

|  |  |  |  |
| --- | --- | --- | --- |
| Отчет сдан «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г. | | | |
|  |  |  |  |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | |
|  | | | |
| Преподаватель | Лакунина О.Н., ст. преподаватель |  |  |
|  | *(Ф.И.О., должность, степень, звание.)* |  | *(подпись)* |

МОСКВА 2022

Оглавление

[Задание на курсовую работу 3](#_Toc28115937)

[Описание структуры данных 4](#_Toc28115938)

[Последовательность 4](#_Toc28115939)

[Дек 6](#_Toc28115940)

[Предложение 8](#_Toc28115941)

[Конечная схема реализуемой структуры данных 9](#_Toc28115942)

[Описание структур данных на языке C 10](#_Toc28115943)

[Схема вызова функций 12](#_Toc28115944)

[Список функции и их назначение 13](#_Toc28115945)

[Функции для работы с последоовательностью 13](#_Toc28115946)

[Функции для работы с деком 14](#_Toc28115947)

[Функции для работы с предложением 15](#_Toc28115948)

[Исходный код программы с комментариями 16](#_Toc28115949)

# Задание на курсовую работу

Задание на курсовую работу состояло в написании программы, реализующей структуру данных «Последовательность деков предложений». Программа должна работать в диалоговом режиме, каждая операция должна быть выполнена в отдельной функции.

Последовательность должна быть реализована на базе структуры хранения «Односвязный список текстов».

Дек должен быть реализован на базе структуры хранения «Двусвязный список предложений».

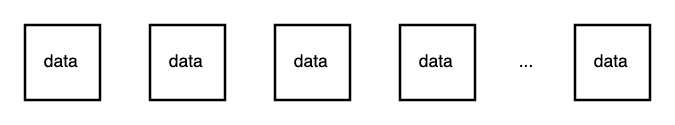
Предложение должно быть реализовано на базе структуры хранения «Односвязный список слов».

Написать отчет по курсовой работе.

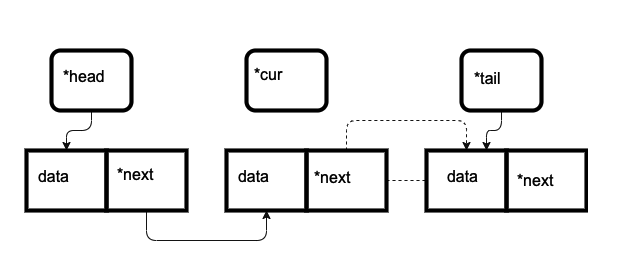
# Описание структуры данных

Последовательность **-** такая логическая структура данных, которая состоит из узлов, которые хранят в качестве данных указатели на деки.

**Схема логической структуры «Последовательность»**



**Схема физической структуры «Последовательность» (на базе односвязного списка)**



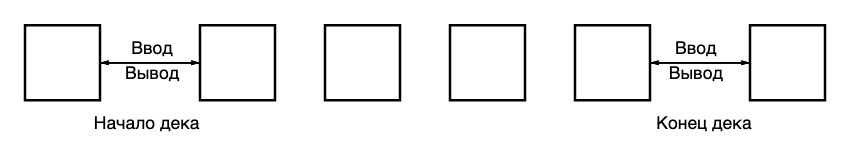
|  |  |
| --- | --- |
| \*head | Указатель на начало последовательности |
| \*tail | Указатель на конец последовательности |
| data | Данные |
| \*next | Указатель на следующий элемент |
| \*cur | Указатель на текущий элемент |

**Список реализуемых функций:**

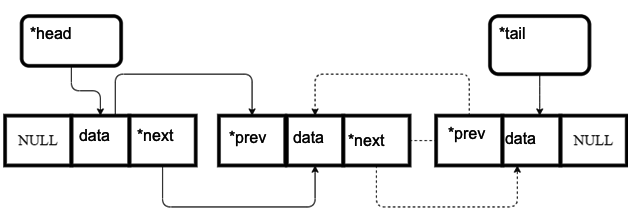
1. Меню
2. Сделать последовательность пустой
3. Проверить последовательность на пустоту
4. Установить указатель на начало
5. Проверить, является ли указатель концом последовательности
6. Переместить указатель вперед
7. Вывести текущий элемент
8. Изменить текущий элемент
9. Извлечь элемент после указателя
10. Добавить элемент после указателя
11. Вывести последовательность
12. Вывести текущий элемент и переместить указатель вперед
13. Узнать размер последовательности

Дек - логическая структура данных, представляющая из себя список элементов, в которой добавление новых элементов и удаление существующих производится с обоих концов.

**Схема логической структуры «Дек»**



**Схема физической структуры «Дек» (на базе двусвязного списка)**



|  |  |
| --- | --- |
| \*head | Указатель на вершину стэка |
| \*tail | Указатель на дно стэка |
| data | Данные |
| \*next | Указатель на следующий элемент стэка |
| \*prev | Указатель на предыдущий элемент стэка |

**Список реализуемых функций:**

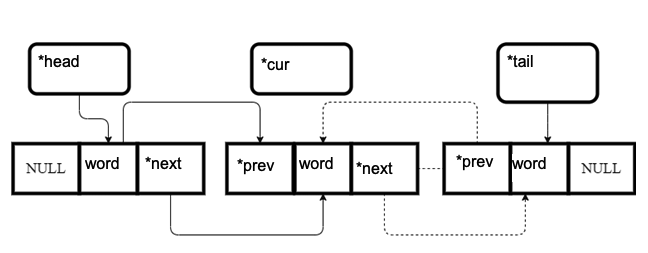
1. Меню
2. Задать глубину дека
3. Сделать дек пустым
4. Проверка на пустоту
5. Вывести первый элемент
6. Вывести последний элемент
7. Удалить первый элемент
8. Удалить последний элемент
9. Изменить первый элемент
10. Изменить последний элемент
11. Добавить в начало
12. Добавить в конец
13. Вывести дек
14. Узнать размер дека

Предложение **-** такая логическая структура данных, которая состоит из узлов, которые хранят в качестве данных слова.

**Схема логической структуры «Предложение»**



**Схема физической структуры «Предложение» (на базе односвязного списка)**

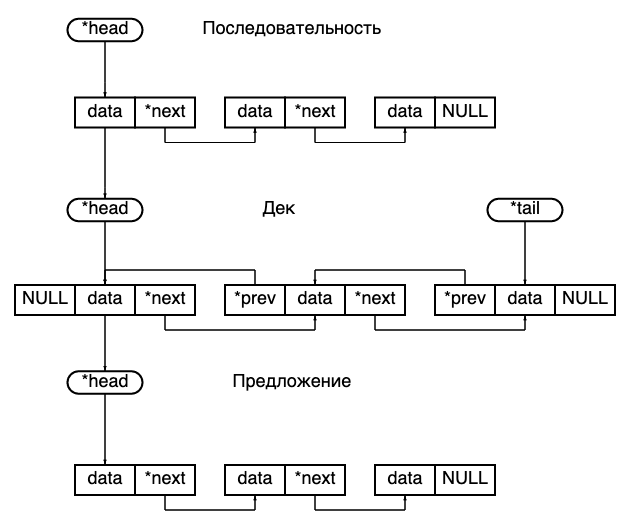
****

|  |  |
| --- | --- |
| \*head | Указатель на первый элемент списка |
| \*cur | Рабочий указатель списка |
| data | Слово, хранимое в узле списка |
| \*next | Указатель на следующий элемент списка |
| \*prev | Указатель на предыдущий элемент списка |

**Список реализуемых функций:**

1. Меню
2. Сделать предложение пустым
3. Проверка на пустоту
4. Установить указатель на начало
5. Проверить, находится ли указатель в конце
6. Переместить указатель вперед
7. Вывести следующий элемент
8. Удалить следующий элемент
9. Извлечь следующий элемент
10. Изменить следующий элемент
11. Добавить элемент после указателя
12. Вывести предложение

# Конечная схема реализуемой структуры данных



# Описание структур данных на языке C

**Последовательность:**

*// Определение структуры элемента последовательности*

*typedef struct sq\_elem {*

*// Данные элемента*

*dq\_set data;*

*// Указатель на следующий элемент последовательности*

*struct sq\_elem\* next;*

*} sq\_elem;*

*// Определение структуры последовательности*

*typedef struct sq\_set {*

*// Указатели на первый, последний и текущий элементы последовательности*

*sq\_elem \*head, \*tail;*

*sq\_elem\* cur;*

*} sq\_set;*

**Дек:**

*// Определение структуры элемента дека*

*typedef struct dq\_elem {*

*// Данные элемента*

*st\_set data;*

*// Указатели на следующий и предыдущий элементы дека*

*struct dq\_elem \*next, \*prev;*

*} dq\_elem;*

*// Определение структуры дека*

*typedef struct dq\_set {*

*// Указатели на первый, последний и текущий элементы дека*

*dq\_elem \*head, \*tail;*

*// Глубина дека*

*size\_t depth;*

*} dq\_set;*

**Предложение (односвязный список слов):**

*// Определение структуры элемента предложения*

*typedef struct st\_elem {*

*// Данные элемента*

*char\* data;*

*// Указатель на следующий элемент предложения*

*struct st\_elem\* next;*

*} st\_elem;*

*// Определение структуры последовательности*

*typedef struct st\_set {*

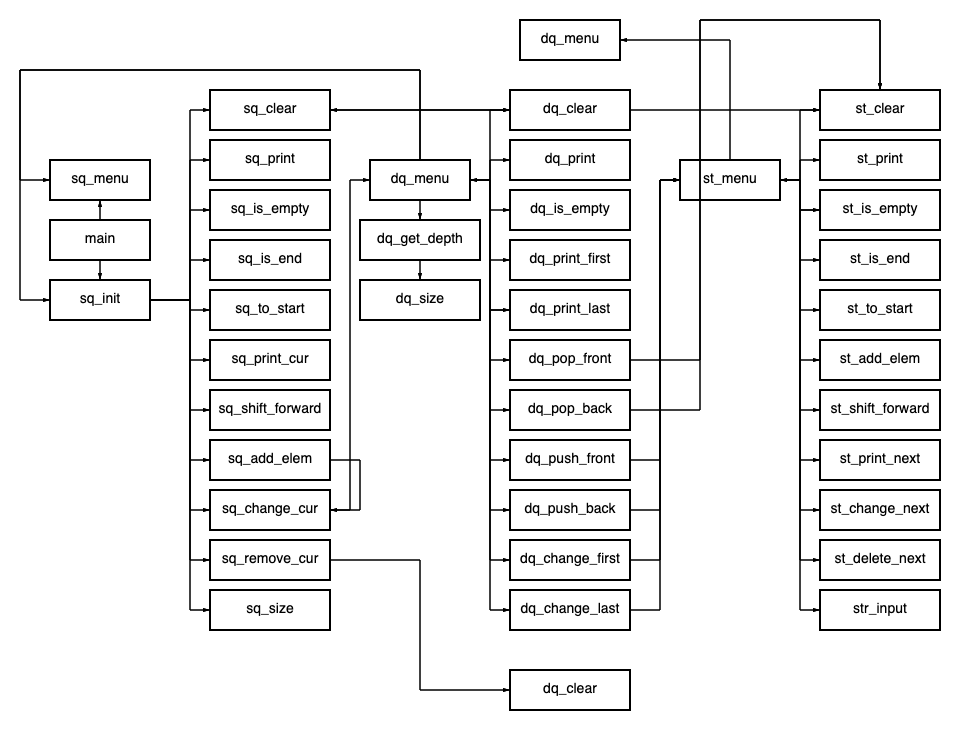
*// Указатели на первый и текущий элементы последовательности*

*st\_elem\* head;*

*st\_elem\* cur;*

*} st\_set;*

# Схема вызова функций



# Список функций и их назначение

Функции для работы с последовательностью

1. **void sq\_menu(sq\_set sequence)** - Функция для отображения меню для работы с последовательностью
2. **void sq\_print(sq\_set\* sequence)** - Функция для вывода элементов последовательности
3. **void sq\_clear(sq\_set\* sequence)** - Функция для очистки последовательности
4. **sq\_set sq\_init()** - Функция для инициализации последовательности
5. **bool sq\_is\_empty(sq\_set\* sequence)** - Функция для проверки, является ли последовательность пустой
6. **bool sq\_is\_end(sq\_set\* sequence)** - Функция для проверки, является ли текущий элемент концом последовательности
7. **void sq\_to\_start(sq\_set\* sequence)** - Функция для установки указателя на начало последовательности
8. **void sq\_add\_elem(sq\_set\* collection)** - Функция для добавления элемента в конец последовательности
9. **void sq\_shift\_forward(sq\_set\* sequence)** - Функция перемещения текущего указателя на следующий элемент в последовательности
10. **void sq\_print\_cur(sq\_set\* sequence)** - Функция вывода текущего элемента последовательности
11. **void sq\_change\_cur(sq\_set\* sequence)** - Функция изменения текущего элемента последовательности
12. **void sq\_remove\_cur(sq\_set\* sequence)** - Функция для удаления текущего элемента из последовательности
13. **size\_t sq\_size(sq\_set\* sequence)** - Функция, возвращающая размер последовательности

Функции для работы с деком

1. **dq\_set dq\_menu(dq\_set\* dq\_ptr)** - Функция для отображения меню для работы с деком
2. **void dq\_get\_depth(dq\_set\* deque)** - Функция, запрашивающая у пользователя глубину дека и валидирующая ввод
3. **size\_t dq\_size(dq\_set\* deque)** - Функция, возвращающая размер дека
4. **void dq\_print(dq\_set\* deque)** - Функция, выводящая содержимое дека
5. **void dq\_clear(dq\_set\* deque)** - Функция, очищающая дек
6. **bool dq\_is\_empty(dq\_set\* deque)** - Функция, проверяющая, пуст ли дек
7. **void dq\_print\_first(dq\_set\* deque)** - Функция, выводящая первый элемент дека
8. **void dq\_print\_last(dq\_set\* deque)** - Функция, выводящая последний элемент дека
9. **void dq\_pop\_front(dq\_set\* deque)** - Функция, удаляющая первый элемент дека
10. **void dq\_pop\_back(dq\_set\* deque)** - Функция для удаления элемента из хвоста дека
11. **void dq\_push\_front(dq\_set\* deque)** - Функция для добавления элемента в начало дека
12. **void dq\_push\_back(dq\_set\* deque)** - Функция для добавления элемента в конец дека
13. **void dq\_change\_first(dq\_set\* deque)** - Функция, изменяющая первый элемент дека
14. **void dq\_change\_last(dq\_set\* deque)** - Функция, изменяющая последний элемент дека

Функции для работы с предложением

1. **st\_set st\_menu(st\_set\* listPointer)** - // Функция для отображения меню для работы с предложением
2. **void st\_print(st\_set collection)** - Функция, выводящая содержимое предложения
3. **void st\_clear(st\_set\* list)** - Функция, очищающая предложение (удаляющая все элементы)
4. **bool st\_is\_empty(st\_set\* list)** - Функция, проверяющая, является ли предложение пустым
5. **bool st\_is\_end(st\_set\* list)** - Функция, проверяющая, является ли указатель на текущий элемент в конце предложения
6. **void st\_to\_start(st\_set\* list)** - Функция, устанавливающая указатель на первый элемент предложения
7. **void st\_add\_elem(st\_set\* collection, char\* data)** - Функция, добавляющая элемент в предложение после текущего
8. **void st\_shift\_forward(st\_set\* list)** - Функция, сдвигающая указатель на текущий элемент вперед на один элемент
9. **void st\_print\_next(st\_set\* list)** - Функция, выводящая на экран следующий элемент после текущего указателя
10. **void st\_change\_next(st\_set\* list, char\* data)** - Функция, изменяющая следующий элемент после текущего указателя
11. **void st\_delete\_next(st\_set\* list)** - Функция, удаляющая следующий элемент после текущего в предложении
12. **char\* str\_input()** - Функция для ввода слова с клавиатуры

# Исходный код программы с комментариями

**main.h:**

#ifndef MAIN\_H\_

#define MAIN\_H\_

#include "sequence.h"

int main();

#endif // MAIN\_H\_

**main.c:**

#include "main.h"

// главная функция

int main() {

// переменная для хранения выбора пользователя из меню

int choice = 0;

// структура для хранения последовательности

sq\_set sequence;

// очистка экрана

system("clear");

// бесконечный цикл

while (true) {

// вывод заголовка меню в цвете

printf(

"\033[38;5;196mС\033[38;5;202mТ\033[38;5;208mA\033[38;5;"

"214mР\033[38;5;220mТ\033[38;5;226mО\033[38;5;190mВ\033[38;5;154mО\033["

"38;5;118mЕ\033[38;5;82m "

"\033[38;5;47mМ\033[38;5;82mЕ\033[38;5;118mН\033[38;5;154mЮ\033[0m\n");

// вывод приглашения к выбору пункта меню

printf("Выберите пункт меню\n");

// вывод пунктов меню

printf("1. Создать последовательность и запустить программу\n");

printf("2. Выход\n");

// считывание выбора пользователя

scanf("%d", &choice);

// выполнение действий в зависимости от выбора пользователя

switch (choice) {

case 1:

// инициализация последовательности

sequence = sq\_init();

// вызываем функцию sq\_menu с аргументом sequence

sq\_menu(sequence);

// завершаем программу с кодом 0 (успешное завершение)

exit(0);

break;

case 2:

// завершаем программу с кодом 0 (успешное завершение)

exit(0);

break;

default:

// очищаем экран

system("clear");

// выводим сообщение об ошибке

printf("Неправильный ввод! Попробуйте снова\n");

// очищаем буфер ввода

getchar();

break;

}

}

return 0;

}

**sequence.h:**

#ifndef SEQUENCE\_H\_

#define SEQUENCE\_H\_

#include "deque.h"

// Определение структуры элемента последовательности

typedef struct sq\_elem {

// Данные элемента

dq\_set data;

// Указатель на следующий элемент последовательности

struct sq\_elem\* next;

} sq\_elem;

// Определение структуры последовательности

typedef struct sq\_set {

// Указатели на первый, последний и текущий элементы последовательности

sq\_elem \*head, \*tail;

sq\_elem\* cur;

} sq\_set;

void sq\_menu(sq\_set sequence);

void sq\_print(sq\_set\* sequence);

void sq\_clear(sq\_set\* sequence);

sq\_set sq\_init();

bool sq\_is\_empty(sq\_set\* sequence);

bool sq\_is\_end(sq\_set\* sequence);

void sq\_to\_start(sq\_set\* sequence);

void sq\_add\_elem(sq\_set\* collection);

void sq\_shift\_forward(sq\_set\* sequence);

void sq\_print\_cur(sq\_set\* sequence);

void sq\_change\_cur(sq\_set\* sequence);

void sq\_remove\_cur(sq\_set\* sequence);

size\_t sq\_size(sq\_set\* sequence);

#endif // SEQUENCE\_H\_

**sequence.c:**

#include "sequence.h"

// Функция для отображения меню для работы с последовательностью

void sq\_menu(sq\_set sequence) {

int choice = 0, st\_choice = 0;

// Очищаем экран

system("clear");

do {

// Выводим заголовок меню с цветным текстом

printf(

"\033[38;5;196mМ\033[38;5;202mЕ\033[38;5;208mН\033[38;5;"

"214mЮ\033[38;5;220m "

"\033[38;5;226mП\033[38;5;190mО\033[38;5;154mС\033[38;5;"

"118mL\033[38;5;82mЕ\033[38;5;46mД\033[38;5;47mО\033[38;5;"

"48mВ\033[38;5;49mА\033[38;5;50mТ\033[38;5;51mЕ\033[38;5;"

"52mЛ\033[38;5;53mЬ\033[38;5;54mН\033[38;5;55mО\033[38;5;"

"56mС\033[38;5;57mТ\033[38;5;58mИ\033[38;5;59m "

"(\033[38;5;60mL\033[38;5;61mV\033[38;5;62mL\033[38;5;63m "

"\033[38;5;64m1\033[38;5;65m)\033[0m\n");

// Выводим варианты действий, которые может выполнить пользователь

printf("Выберите пункт меню\n");

printf("1. Очистить\n2. Проверить на путоту\n");

printf("3. Установить указатель на начало\n");

printf("4. Проверить, является ли указатель концом последовательности\n");

printf("5. Переместить указатель вперед\n");

printf("6. Вывести текущий элемент\n");

printf("7. Изменить текущий элемент\n");

printf("8. Извлечь элемент после указателя\n");

printf("9. Добавить элемент после указателя\n");

printf("10. Вывести последовательность\n");

printf("11. Вывести текущий элемент и переместить указатель вперед\n");

printf("12. Удалить последовательность и выйти\n");

printf("\nПоследовательность содержит %zu элементов\n", sq\_size(&sequence));

// Если последовательность не пустая, вывести ее элементы

if (sequence.head != NULL) {

printf("Последовательность: \n");

sq\_print(&sequence);

}

// Считываем выбор пользователя

scanf("%d", &choice);

// В зависимости от выбора выполняем соответствующую операцию

switch (choice) {

case 1:

// Очистка последовательности

system("clear");

// Если последовательность не пустая, выполняем очистку

if (sequence.head != NULL) sq\_clear(&sequence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

// Ждем нажатия Enter для возврата в меню

if (getchar() == '\n') getchar();

break;

case 2:

system("clear");

sq\_is\_empty(&sequence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

if (getchar() == '\n') getchar();

break;

case 3:

system("clear");

sq\_to\_start(&sequence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

if (getchar() == '\n') getchar();

break;

case 4:

system("clear");

if (sequence.head != NULL) {

sq\_is\_end(&sequence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, пустая последовательность\n");

}

if (getchar() == '\n') getchar();

break;

case 5:

system("clear");

if (sequence.head != NULL) {

if (sequence.cur->next != NULL) {

sq\_shift\_forward(&sequence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, конец последовательности");

}

} else {

printf("Ошибка, пустая последовательность");

}

if (getchar() == '\n') getchar();

break;

case 6:

system("clear");

if (sequence.head != NULL) {

sq\_print\_cur(&sequence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, пустая последовательность\n");

}

if (getchar() == '\n') getchar();

break;

case 7:

system("clear");

if (sequence.head != NULL) {

sq\_change\_cur(&sequence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, пустая последовательность\n");

}

if (getchar() == '\n') getchar();

break;

case 8:

system("clear");

if (sequence.head != NULL) {

sq\_print\_cur(&sequence);

sq\_remove\_cur(&sequence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, пустая последовательность\n");

}

if (getchar() == '\n') getchar();

break;

case 9:

system("clear");

do {

printf(

"\033[38;5;196mС\033[38;5;202mТ\033[38;5;208mA\033[38;5;"

"214mР\033[38;5;220mТ\033[38;5;226mО\033[38;5;190mВ\033[38;5;"

"154mО\033["

"38;5;118mЕ\033[38;5;82m "

"\033[38;5;47mМ\033[38;5;82mЕ\033[38;5;118mН\033[38;5;154mЮ\033["

"0m\n");

printf("Выберите пункт меню\n");

printf("1. Создать дек\n");

printf("2. Вернуться в меню последовательности\n");

scanf("%d", &st\_choice);

switch (st\_choice) {

case 1:

sq\_add\_elem(&sequence);

break;

case 2:

break;

default:

system("clear");

printf("Неправильный ввод! Попробуйте снова\n");

if (getchar() == '\n') getchar();

break;

}

} while (st\_choice != 2);

system("clear");

break;

case 10:

system("clear");

if (sequence.head != NULL) {

sq\_print(&sequence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, пустая последовательность\n");

}

if (getchar() == '\n') getchar();

break;

case 11:

system("clear");

if (sequence.head != NULL) {

if (sequence.cur->next != NULL) {

dq\_print(&sequence.cur->data);

sq\_shift\_forward(&sequence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, конец последовательности");

}

} else {

printf("Ошибка, пустая последовательность\n");

}

if (getchar() == '\n') getchar();

break;

case 12:

if (sequence.head != NULL) sq\_clear(&sequence);

break;

default:

system("clear");

printf("Неправильный ввод! Попробуйте снова\n");

break;

}

} while (choice != 12);

system("clear");

}

// Функция для вывода элементов последовательности

void sq\_print(sq\_set\* sequence) {

// Указатель на текущий элемент последовательности

sq\_elem\* cur = sequence->head;

// Счетчик элементов

size\_t count = 0;

// Пока указатель не равен NULL, выводим текущий элемент

while (cur != NULL) {

// Выводим номер элемента и указатель на текущий элемент, если нужно

printf("[\033[0;32m%zu\033[0m]:", count);

if (cur == sequence->cur) printf("\033[0;32m<--\033[0m");

printf("\n");

// Выводим содержимое элемента

dq\_print(&cur->data);

// Увеличиваем счетчик

count++;

// Переходим к следующему элементу

cur = cur->next;

}

}

// Функция для удаления текущего элемента из последовательности

void sq\_remove\_cur(sq\_set\* sequence) {

// Если текущего элемента нет, то ничего удалять не нужно

if (sequence->cur == NULL) return;

// Указатель на текущий элемент

sq\_elem\* node = sequence->head;

// Указатель на предыдущий элемент

sq\_elem\* prev = NULL;

// Пока указатель не равен NULL и текущий элемент не равен текущему элементу

// последовательности, ищем текущий элемент

while (node != NULL && node != sequence->cur) {

prev = node;

node = node->next;

}

// Текущий элемент не найден в последовательности

if (node == NULL) return;

// Отсоединяем текущий элемент от последовательности

// Текущий элемент является головой последовательности

if (prev == NULL)

sequence->head = node->next;

else

prev->next = node->next;

// Текущий элемент является хвостом последовательности

if (node->next == NULL) sequence->tail = prev;

// Очищаем данные в текущем элементе и освобождаем память

dq\_clear(&node->data);

free(node);

// Обновляем указатель на текущий элемент

sequence->cur = sequence->head;

}

// Функция для очистки последовательности

void sq\_clear(sq\_set\* sequence) {

// Указатель на голову последовательности

sq\_elem\* node = sequence->head;

// Указатель на элемент для удаления

sq\_elem\* to\_delete = node;

// Пока у элемента есть следующий, удаляем текущий элемент

while (node->next != NULL) {

to\_delete = node;

node = node->next;

// Очищаем данные в элементе и освобождаем память

dq\_clear(&to\_delete->data);

free(to\_delete);

}

// Удаляем последний элемент

dq\_clear(&node->data);

free(node);

// Обнуляем указатели в последовательности

sequence->head = NULL;

sequence->cur = NULL;

sequence->tail = NULL;

}

// Функция для инициализации последовательности

sq\_set sq\_init() {

// Создаем последовательность

sq\_set sequence;

// Обнуляем указатели

sequence.cur = NULL;

sequence.head = NULL;

sequence.tail = NULL;

// Возвращаем последовательность

return sequence;

}

// Функция для проверки, является ли последовательность пустой

bool sq\_is\_empty(sq\_set\* sequence) {

// Если голова последовательности равна NULL, последовательность пуста

if (sequence->head == NULL) {

printf("Последовательность пуста\n");

return true;

} else {

// Иначе последовательность не пустая

printf("Последовательность не пустая\n");

return false;

}

}

// Функция для проверки, является ли текущий элемент концом последовательности

bool sq\_is\_end(sq\_set\* sequence) {

// Если у текущего элемента нет следующего, он является концом

// последовательности

if (sequence->cur->next == NULL) {

printf("Указатель находится в конце последовательности\n");

return true;

} else {

// Иначе текущий элемент не является концом последовательности

printf("Указатель не находится в конце последовательности\n");

}

// Возвращаем false

return false;

}

// Функция для установки указателя на начало последовательности

void sq\_to\_start(sq\_set\* sequence) {

// Устанавливаем указатель на голову последовательности

sequence->cur = sequence->head;

}

// Функция для добавления элемента в конец последовательности

void sq\_add\_elem(sq\_set\* sequence) {

// Выделяем память под новый элемент

sq\_elem\* cur = (struct sq\_elem\*)malloc(sizeof(struct sq\_elem));

// Если память не была выделена, выводим сообщение об ошибке и завершаем

// программу

if (cur == NULL) {

printf("Ошибка, NULL\n");

exit(1);

}

// Инициализируем данные в элементе

cur->data = dq\_menu(NULL);

// Устанавливаем указатель на следующий элемент в NULL

cur->next = NULL;

// Если в последовательности нет элементов

if (sequence->head == NULL) {

// То новый элемент становится первым, текущим и последним в

// последовательности

sequence->head = cur;

sequence->cur = cur;

sequence->tail = cur;

} else {

// В противном случае, новый элемент становится последним в

// последовательности

sequence->tail->next = cur;

sequence->tail = cur;

}

}

// Функция перемещения текущего указателя на следующий элемент в

// последовательности

void sq\_shift\_forward(sq\_set\* sequence) {

// Перемещение текущего указателя на следующий элемент

sequence->cur = sequence->cur->next;

}

// Функция вывода текущего элемента последовательности

void sq\_print\_cur(sq\_set\* sequence) {

// Вывод текущего элемента

dq\_print(&sequence->cur->data);

}

// Функция изменения текущего элемента последовательности

void sq\_change\_cur(sq\_set\* sequence) {

// Изменение данных текущего элемента

sequence->cur->data = dq\_menu(&sequence->cur->data);

}

// Функция, возвращающая размер последовательности

size\_t sq\_size(sq\_set\* sequence) {

// Указатель на текущий элемент последовательности

sq\_elem\* cur = sequence->head;

// Счетчик элементов в последовательности

size\_t count = 0;

// Пока текущий элемент существует

while (cur != NULL) {

// Переход к следующему элементу

cur = cur->next;

// Увеличение счетчика

count++;

}

// Возвращение размера последовательности

return count;

}

**deque.h:**

#ifndef DEQUE\_H\_

#define DEQUE\_H\_

#include "sentence.h"

// Определение структуры элемента дека

typedef struct dq\_elem {

// Данные элемента

st\_set data;

// Указатели на следующий и предыдущий элементы дека

struct dq\_elem \*next, \*prev;

} dq\_elem;

// Определение структуры дека

typedef struct dq\_set {

// Указатели на первый, последний и текущий элементы дека

dq\_elem \*head, \*tail;

// Глубина дека

size\_t depth;

} dq\_set;

dq\_set dq\_menu(dq\_set\* dq\_ptr);

void dq\_get\_depth(dq\_set\* deque);

size\_t dq\_size(dq\_set\* deque);

void dq\_print(dq\_set\* deque);

void dq\_clear(dq\_set\* deque);

bool dq\_is\_empty(dq\_set\* deque);

void dq\_print\_first(dq\_set\* deque);

void dq\_print\_last(dq\_set\* deque);

void dq\_pop\_front(dq\_set\* deque);

void dq\_pop\_back(dq\_set\* deque);

void dq\_push\_front(dq\_set\* deque);

void dq\_push\_back(dq\_set\* deque);

void dq\_change\_first(dq\_set\* deque);

void dq\_change\_last(dq\_set\* deque);

#endif // DEQUE\_H\_

**deque.c:**

#include "deque.h"

// Функция для отображения меню для работы с деком

dq\_set dq\_menu(dq\_set\* dq\_ptr) {

int choice = 0, st\_choice = 0;

dq\_set deque;

if (dq\_ptr != NULL) {

deque = \*dq\_ptr;

} else {

deque.head = NULL;

deque.tail = NULL;

}

dq\_get\_depth(&deque);

system("clear");

do {

// Выводим заголовок меню с цветным текстом

printf(

"\033[38;5;196mМ\033[38;5;202mЕ\033[38;5;208mН\033[38;5;214mЮ\033[38;5;"

"220m "

"\033[38;5;226mД\033[38;5;190mЕ\033[38;5;154mК\033[38;5;118mA\033[38;5;"

"82m (\033[38;5;46mL\033[38;5;47mV\033[38;5;48mL\033[38;5;49m "

"\033[38;5;50m2\033[38;5;51m)\033[0m\n");

// Выводим варианты действий, которые может выполнить пользователь

printf("Выберите пункт меню\n");

printf("1. Очистить\n2. Проверить на пустоту\n");

printf("3. Вывести первый элемент\n");

printf("4. Вывести последний элемент\n");

printf("5. Удалить первый элемент\n");

printf("6. Удалить последний элемент\n");

printf("7. Извлечь первый элемент\n");

printf("8. Извлечь последний элемент\n");

printf("9. Изменить первый элемент\n");

printf("10. Изменить последний элемент\n");

printf("11. Добавить в начало\n");

printf("12. Добавить в конец\n");

printf("13. Вывести дек\n");

printf("14. Вернуться в стартовое меню\n");

// Если дек не пустой, вывести его элементы

printf("\nДек содержит %zu элементов (глубина = %zu)\n", dq\_size(&deque),

deque.depth);

if (dq\_size(&deque) != 0) {

printf("Дек:\n");

dq\_print(&deque);

}

// Считываем выбор пользователя

scanf("%d", &choice);

// В зависимости от выбора выполняем соответствующую операцию

switch (choice) {

case 1:

// Очистка дека

system("clear");

// Если дек не пустой, выполняем очистку

if (deque.head != NULL) dq\_clear(&deque);

// Ждем нажатия Enter для возврата в меню

printf("Готово! Нажмите Enter, чтобы вернуться\n");

if (getchar() == '\n') getchar();

break;

case 2:

system("clear");

dq\_is\_empty(&deque);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

if (getchar() == '\n') getchar();

break;

case 3:

system("clear");

if (deque.head != NULL) {

dq\_print\_first(&deque);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, пустой дек\n");

}

if (getchar() == '\n') getchar();

break;

case 4:

system("clear");

if (deque.head != NULL) {

dq\_print\_last(&deque);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, пустой дек\n");

}

if (getchar() == '\n') getchar();

break;

case 5:

system("clear");

if (deque.tail != NULL) {

dq\_pop\_front(&deque);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, пустой дек\n");

}

if (getchar() == '\n') getchar();

break;

case 6:

system("clear");

if (deque.tail != NULL) {

dq\_pop\_back(&deque);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, пустой дек\n");

}

if (getchar() == '\n') getchar();

break;

case 7:

system("clear");

if (deque.head != NULL) {

dq\_print\_first(&deque);

dq\_pop\_front(&deque);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, пустой дек\n");

}

if (getchar() == '\n') getchar();

break;

case 8:

system("clear");

if (deque.head != NULL) {

dq\_print\_last(&deque);

dq\_pop\_back(&deque);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, пустой дек\n");

}

if (getchar() == '\n') getchar();

break;

case 9:

system("clear");

if (deque.tail != NULL) {

dq\_change\_first(&deque);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, пустой дек\n");

}

if (getchar() == '\n') getchar();

break;

case 10:

system("clear");

if (deque.tail != NULL) {

dq\_change\_last(&deque);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, пустой дек\n");

}

if (getchar() == '\n') getchar();

break;

break;

case 11:

system("clear");

do {

printf(

"\033[38;5;196mС\033[38;5;202mТ\033[38;5;208mA\033[38;5;"

"214mР\033[38;5;220mТ\033[38;5;226mО\033[38;5;190mВ\033[38;5;"

"154mО\033["

"38;5;118mЕ\033[38;5;82m "

"\033[38;5;47mМ\033[38;5;82mЕ\033[38;5;118mН\033[38;5;154mЮ\033["

"0m\n");

printf("Выберите пункт меню\n");

printf("1. Создать предложение\n");

printf("2. Вернуться в меню дека\n");

scanf("%d", &st\_choice);

switch (st\_choice) {

case 1:

if (dq\_size(&deque) < deque.depth) {

dq\_push\_front(&deque);

} else {

system("clear");

printf("Ошибка, достигнуто максимальное значение глубины\n");

if (getchar() == '\n') getchar();

}

break;

case 2:

break;

default:

system("clear");

printf("Неправильный ввод! Попробуйте снова\n");

if (getchar() == '\n') getchar();

break;

}

} while (st\_choice != 2);

break;

case 12:

system("clear");

do {

printf(

"\033[38;5;196mС\033[38;5;202mТ\033[38;5;208mA\033[38;5;"

"214mР\033[38;5;220mТ\033[38;5;226mО\033[38;5;190mВ\033[38;5;"

"154mО\033["

"38;5;118mЕ\033[38;5;82m "

"\033[38;5;47mМ\033[38;5;82mЕ\033[38;5;118mН\033[38;5;154mЮ\033["

"0m\n");

printf("Выберите пункт меню\n");

printf("1. Создать предложение\n");

printf("2. Вернуться в меню дека\n");

scanf("%d", &st\_choice);

switch (st\_choice) {

case 1:

if (dq\_size(&deque) < deque.depth) {

dq\_push\_back(&deque);

} else {

system("clear");

printf("Ошибка, достигнуто максимальное значение глубины\n");

if (getchar() == '\n') getchar();

}

break;

case 2:

break;

default:

system("clear");

printf("Неправильный ввод! Попробуйте снова\n");

if (getchar() == '\n') getchar();

break;

}

} while (st\_choice != 2);

break;

case 13:

system("clear");

if (deque.tail != NULL) {

dq\_print(&deque);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, пустой дек\n");

}

if (getchar() == '\n') getchar();

break;

case 14:

break;

default:

system("clear");

printf("Неправильный ввод! Попробуйте снова\n");

if (getchar() == '\n') getchar();

break;

}

} while (choice != 14);

system("clear");

return deque;

}

// Функция, запрашивающая у пользователя глубину дека и валидирующая ввод

void dq\_get\_depth(dq\_set\* deque) {

// Глубина дека

size\_t depth;

// Буфер для ввода строки

char input[10];

// Очистка экрана

system("clear");

// Бесконечный цикл

while (true) {

// Запрос глубины дека у пользователя

printf("Введите глубину дека (целое положительное число):\n");

// Очистка буфера stdin

getchar();

// Чтение строки с консоли

fgets(input, sizeof(input), stdin);

// Проверка, что введено целое положительное число

if (sscanf(input, "%zu", &depth) == 1 && depth > 0) {

// Если введено корректное число, то устанавливаем глубину дека и выходим

// из цикла

deque->depth = depth;

break;

} else {

// Иначе, очищаем экран и сообщаем об ошибке

system("clear");

printf("Неправильный ввод! Попробуйте снова\n");

// Очистка буфера stdin

getchar();

}

}

}

// Функция, возвращающая размер дека

size\_t dq\_size(dq\_set\* deque) {

// Указатель на текущий элемент дека

dq\_elem\* cur = deque->head;

// Счетчик элементов в деке

size\_t count = 0;

// Пока текущий элемент существует

while (cur != NULL) {

// Переход к следующему элементу

cur = cur->next;

// Увеличение счетчика

count++;

}

// Возвращение размера дека

return count;

}

// Функция, очищающая дек

void dq\_clear(dq\_set\* deque) {

// Указатель на текущий элемент дека

dq\_elem\* node = deque->head;

// Указатель на элемент для удаления

dq\_elem\* to\_delete = node;

// Пока текущий элемент не является последним в деке

while (node->next != NULL) {

// Сохраняем указатель на текущий элемент для удаления

to\_delete = node;

// Переход к следующему элементу

node = node->next;

// Освобождение памяти, занятой текущим элементом

free(to\_delete);

}

// Обнуление указателей на первый и последний элементы дека

deque->head = NULL;

deque->tail = NULL;

}

// Функция, проверяющая, пуст ли дек

bool dq\_is\_empty(dq\_set\* deque) {

// Если оба указателя на начало и конец дека равны NULL, то дек пуст

if (deque->head == NULL && deque->tail == NULL) {

// Сообщение об отсутствии элементов в деке

printf("Дек пуст\n");

// Возвращение результата

return true;

} else if (deque->head != NULL && deque->tail != NULL) {

// Сообщение о наличии элементов в деке

printf("Дек не пустой\n");

}

// Возвращение результата

return false;

}

// Функция, выводящая первый элемент дека

void dq\_print\_first(dq\_set\* deque) {

// Вывод названия элемента

printf("Первый элемент: ");

// Вывод данных элемента

st\_print(deque->tail->data);

}

// Функция, выводящая последний элемент дека

void dq\_print\_last(dq\_set\* deque) {

// Вывод названия элемента

printf("Последний элемент: ");

// Вывод данных элемента

st\_print(deque->head->data);

}

// Функция, удаляющая первый элемент дека

void dq\_pop\_front(dq\_set\* deque) {

// Указатель на элемент для удаления

dq\_elem\* to\_delete = deque->head;

// Если в деке больше одного элемента

if (deque->head != deque->tail) {

// Переустанавливаем указатель на первый элемент дека

deque->head = deque->head->next;

// Обнуляем указатель на предыдущий элемент у нового первого элемента

deque->head->prev = NULL;

} else {

// Иначе голова и хвост дека становятся равными NULL

deque->head = NULL;

deque->tail = NULL;

}

// Очищаем данные в элементе

st\_clear(&to\_delete->data);

// Освобождаем память

free(to\_delete);

}

// Функция для удаления элемента из хвоста дека

void dq\_pop\_back(dq\_set\* deque) {

// Сохраняем указатель на удаляемый элемент

dq\_elem\* to\_delete = deque->tail;

// Если у удаляемого элемента есть предыдущий, устанавливаем хвост дека на

// этот элемент

if (deque->head != deque->tail) {

deque->tail = deque->tail->prev;

deque->tail->next = NULL;

} else {

// Иначе голова и хвост дека становятся равными NULL

deque->tail = NULL;

deque->head = NULL;

}

// Очищаем данные в удаляемом элементе и освобождаем память

st\_clear(&to\_delete->data);

free(to\_delete);

}

// Функция для добавления элемента в начало дека

void dq\_push\_front(dq\_set\* deque) {

// Выделяем память для нового элемента

dq\_elem\* new\_elem = (struct dq\_elem\*)malloc(sizeof(dq\_elem));

// Если память не удалось выделить, выводим сообщение об ошибке и завершаем

// программу

if (new\_elem == NULL) {

printf("Ошибка, NULL\n");

exit(1);

}

// Инициализируем данные в новом элементе

new\_elem->data = st\_menu(NULL);

new\_elem->prev = NULL;

// Если хвост дека равен NULL, значит дек пуст. Устанавливаем голову и хвост

// дека на новый элемент

if (deque->tail == NULL) {

new\_elem->next = NULL;

deque->tail = new\_elem;

deque->head = deque->tail;

} else {

// Иначе устанавливаем указатель на предыдущий элемент

new\_elem->prev = deque->tail;

deque->tail->next = new\_elem;

deque->tail = new\_elem;

}

}

// Функция для добавления элемента в конец дека

void dq\_push\_back(dq\_set\* deque) {

// Выделяем память для нового элемента

dq\_elem\* new\_elem = (struct dq\_elem\*)malloc(sizeof(dq\_elem));

// Если память не удалось выделить, выводим сообщение об ошибке и завершаем

// программу

if (new\_elem == NULL) {

printf("Ошибка, NULL\n");

exit(1);

}

// Инициализируем данные в новом элементе

new\_elem->data = st\_menu(NULL);

new\_elem->next = NULL;

// Если голова дека равна NULL, значит дек пуст. Устанавливаем голову и хвост

// дека на новый элемент

if (deque->head == NULL) {

new\_elem->prev = NULL;

deque->head = new\_elem;

deque->tail = deque->head;

} else {

// Иначе устанавливаем указатель на следующий элемент текущего хвоста на

// новый элемент

new\_elem->next = deque->head;

deque->head->prev = new\_elem;

deque->head = new\_elem;

}

}

// Функция, выводящая содержимое дека

void dq\_print(dq\_set\* deque) {

// Указатель на текущий элемент дека

dq\_elem\* cur = deque->head;

// Вывод названия конца дека

printf("\033[38;5;15m\033[48;5;1mКОНЕЦ\033[0m\n");

// Цикл перебора элементов дека

while (cur != NULL) {

// Отступ элемента от края

printf(" ");

// Вывод данных текущего элемента

st\_print(cur->data);

// Переход к следующему элементу

cur = cur->next;

}

// Вывод названия начала дека

printf("\033[38;5;15m\033[48;5;1mНАЧАЛО\033[0m\n");

}

// Функция, изменяющая первый элемент дека

void dq\_change\_first(dq\_set\* deque) {

// Изменение данных первого элемента

deque->tail->data = st\_menu(&deque->tail->data);

}

// Функция, изменяющая последний элемент дека

void dq\_change\_last(dq\_set\* deque) {

// Изменение данных последнего элемента

deque->head->data = st\_menu(&deque->head->data);

}

**sentence.h:**

#ifndef SENTENCE\_H\_

#define SENTENCE\_H\_

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

// Определение структуры элемента предложения

typedef struct st\_elem {

// Данные элемента

char\* data;

// Указатель на следующий элемент предложения

struct st\_elem\* next;

} st\_elem;

// Определение структуры последовательности

typedef struct st\_set {

// Указатели на первый и текущий элементы последовательности

st\_elem\* head;

st\_elem\* cur;

} st\_set;

st\_set st\_menu(st\_set\* listPointer);

void st\_print(st\_set collection);

void st\_clear(st\_set\* list);

bool st\_is\_empty(st\_set\* list);

bool st\_is\_end(st\_set\* list);

void st\_to\_start(st\_set\* list);

void st\_add\_elem(st\_set\* collection, char\* data);

void st\_shift\_forward(st\_set\* list);

void st\_print\_next(st\_set\* list);

void st\_change\_next(st\_set\* list, char\* data);

void st\_delete\_next(st\_set\* list);

char\* str\_input();

#endif // SENTENCE\_H\_

**sentence.c:**

#include "sentence.h"

// Функция для отображения меню для работы с предложением

st\_set st\_menu(st\_set\* sentence\_ptr) {

int choice = 0;

struct st\_set sentence;

if (sentence\_ptr != NULL) {

sentence = \*sentence\_ptr;

} else {

sentence.cur = NULL;

sentence.head = NULL;

}

system("clear");

do {

// Выводим заголовок меню с цветным текстом

printf(

"\033[38;5;196mМ\033[38;5;202mЕ\033[38;5;208mН\033[38;5;214mЮ\033[38;5;"

"220m "

"\033[38;5;226mП\033[38;5;190mР\033[38;5;154mЕ\033[38;5;118mД\033[38;5;"

"82mЛ\033[38;5;46mО\033[38;5;47mЖ\033[38;5;48mЕ\033[38;5;49mН\033[38;5;"

"50mИ\033[38;5;51mЯ\033[38;5;52m "

"(\033[38;5;53mL\033[38;5;54mV\033[38;5;55mL\033[38;5;56m "

"\033[38;5;57m3\033[38;5;58m)\033[0m\n");

// Выводим варианты действий, которые может выполнить пользователь

printf("Выберите пункт меню\n");

printf("1. Очистить\n");

printf("2. Проверить на пустоту\n");

printf("3. Установить указатель на начало\n");

printf("4. Проверить, находится ли указатель в конце\n");

printf("5. Переместить указатель вперед\n");

printf("6. Вывести следующий элемент\n");

printf("7. Удалить следующий элемент\n");

printf("8. Извлечь следующий элемент\n");

printf("9. Изменить следующий элемент\n");

printf("10. Добавить элемент после указателя\n");

printf("11. Вывести предложение\n");

printf("12. Вернуться в стартовое меню\n");

// Если предложение не пустое, вывести его

printf("\nПредложение: \n");

if (sentence.head != NULL)

st\_print(sentence);

else

// Иначе сообщить о том, что предложение пусто

printf("Предложение пусто\n");

// Считываем выбор пользователя

scanf("%d", &choice);

// В зависимости от выбора выполняем соответствующую операцию

switch (choice) {

case 1:

// Очистка предложения

system("clear");

// Если предложение не пустое, выполняем очистку

if (sentence.head != NULL) st\_clear(&sentence);

// Ждем нажатия Enter для возврата в меню

printf("Готово! Нажмите Enter, чтобы вернуться\n");

if (getchar() == '\n') getchar();

break;

case 2:

system("clear");

st\_is\_empty(&sentence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

if (getchar() == '\n') getchar();

break;

case 3:

system("clear");

st\_to\_start(&sentence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

if (getchar() == '\n') getchar();

break;

case 4:

system("clear");

if (sentence.head != NULL) {

st\_is\_end(&sentence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, пустое предложение\n");

}

if (getchar() == '\n') getchar();

break;

case 5:

system("clear");

if (sentence.head != NULL) {

if (sentence.cur->next != NULL) {

st\_shift\_forward(&sentence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else

printf("Ошибка, конец предложения");

} else {

printf("Ошибка, пустое предложение\n");

}

if (getchar() == '\n') getchar();

break;

case 6:

system("clear");

if (sentence.head != NULL) {

if (sentence.cur->next != NULL) {

st\_print\_next(&sentence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else

printf("Ошибка, конец предложения");

} else {

printf("Ошибка, пустое предложение\n");

}

if (getchar() == '\n') getchar();

break;

case 7:

system("clear");

if (sentence.head != NULL) {

if (sentence.cur->next != NULL) {

st\_delete\_next(&sentence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else

printf("Ошибка, конец предложения");

} else {

printf("Ошибка, пустое предложение\n");

}

if (getchar() == '\n') getchar();

break;

case 8:

system("clear");

if (sentence.head != NULL) {

if (sentence.cur->next != NULL) {

st\_print\_next(&sentence);

st\_delete\_next(&sentence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else

printf("Ошибка, конец предложения");

} else {

printf("Ошибка, пустое предложение\n");

}

if (getchar() == '\n') getchar();

break;

case 9:

system("clear");

if (sentence.head != NULL) {

if (sentence.cur->next != NULL) {

char\* inputChangeElement;

inputChangeElement = str\_input();

st\_change\_next(&sentence, inputChangeElement);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else

printf("Ошибка, конец предложения\n");

} else {

printf("Ошибка, пустое предложение\n");

}

if (getchar() == '\n') getchar();

break;

case 10:

system("clear");

getchar();

char\* inputElement = str\_input();

st\_add\_elem(&sentence, inputElement);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

getchar();

break;

case 11:

system("clear");

if (sentence.head != NULL) {

st\_print(sentence);

printf("Готово! Нажмите Enter, чтобы вернуться\n");

} else {

printf("Ошибка, пустое предложение\n");

}

if (getchar() == '\n') getchar();

break;

case 12:

break;

default:

system("clear");

printf("Неправильный ввод! Попробуйте снова\n");

break;

}

} while (choice != 12);

system("clear");

return sentence;

}

// Функция, выводящая содержимое предложения

void st\_print(st\_set sentence) {

// Указатель на текущий элемент предложения

st\_elem\* node = sentence.head;

// Если элементы в предложении есть

if (node != NULL) {

// Цикл перебора элементов

while (true) {

// Указатель на текущий символ

char\* ptr;

// Поиск длины строки

for (ptr = node->data; \*ptr; ++ptr)

;

// Цикл вывода символов строки

for (int i = 0; i < ptr - node->data; i++) {

// Если текущий элемент является текущим в предложении

if (node == sentence.cur)

// Вывод символа с подсветкой

printf("\033[0;34m%c\033[0m", node->data[i]);

else

// Вывод символа

printf("%c", node->data[i]);

}

// Вывод пробела

printf(" ");

// Переход к следующему элементу

node = node->next;

// Если следующего элемента нет, выходим из цикла

if (node == NULL) {

// Переход на новую строку

printf("\n");

break;

}

}

} else

// Переход на новую строку

printf("\n");

}

// Функция, удаляющая следующий элемент после текущего в предложении

void st\_delete\_next(st\_set\* sentence) {

// Если у следующего элемента есть следующий

if (sentence->cur->next->next != 0) {

// Указатель на элемент для удаления

st\_elem\* to\_delete = sentence->cur->next;

// Установка указателя на следующий элемент у текущего элемента

sentence->cur->next = sentence->cur->next->next;

// Освобождение памяти, выделенной под элемент

free(to\_delete);

} else {

// Установка указателя на следующий элемент у текущего элемента в NULL

sentence->cur->next = NULL;

}

}

// Функция, очищающая предложение (удаляющая все элементы)

void st\_clear(st\_set\* sentence) {

// Указатель на первый элемент предложения

st\_elem\* node = sentence->head;

// Указатель на элемент для удаления

st\_elem\* to\_delete = node;

// Если элементы в предложении есть

if (node != NULL) {

// Цикл удаления элементов

while (node->next != NULL) {

// Запоминаем элемент для удаления

to\_delete = node;

// Переходим к следующему элементу

node = node->next;

// Освобождение памяти, выделенной под элемент

free(to\_delete);

}

}

// Установка указателя на первый элемент в NULL

sentence->head = NULL;

// Установка указателя на текущий элемент в NULL

sentence->cur = NULL;

}

// Функция, проверяющая, является ли предложение пустым

bool st\_is\_empty(st\_set\* sentence) {

// Если первый элемент предложения равен NULL

if (sentence->head == NULL) {

// Вывод сообщения о том, что предложение пусто

printf("Предложение пусто\n");

// Возврат значения true

return true;

} else {

// Вывод сообщения о том, что предложение не пустое

printf("Предложение не пустое\n");

// Возврат значения false

return false;

}

}

// Функция, проверяющая, является ли указатель на текущий элемент в конце

// предложения

bool st\_is\_end(st\_set\* sentence) {

// Если у текущего элемента нет следующего

if (sentence->cur->next == NULL) {

// Вывод сообщения о том, что указатель в конце

printf("Указатель в конце\n");

// Возврат значения true

return true;

} else {

// Вывод сообщения о том, что указатель не в конце

printf("Указатель не в конце\n");

// Возврат значения false

return false;

}

}

// Функция, устанавливающая указатель на первый элемент предложения

void st\_to\_start(st\_set\* sentence) {

// Установка указателя на первый элемент предложения

sentence->cur = sentence->head;

}

// Функция, добавляющая элемент в предложение после текущего

void st\_add\_elem(st\_set\* sentence, char\* data) {

// Создание указателя на элемент типа st\_elem

struct st\_elem\* cur;

// Выделение памяти под элемент

cur = (struct st\_elem\*)malloc(sizeof(struct st\_elem));

// Проверка, успешно ли выделена память

if (cur == NULL) {

// Вывод сообщения об ошибке и завершение программы

printf("Ошибка, NULL\n");

exit(1);

}

// Запись данных в элемент

cur->data = data;

// Установка указателя на следующий элемент равным NULL

cur->next = NULL;

// Если предложение пустое

if (sentence->head == NULL) {

// Устанавливаем указатель на первый элемент равным текущему элементу

sentence->head = cur;

// Устанавливаем указатель на текущий элемент равным текущему элементу

sentence->cur = cur;

}

// Если текущий элемент не является хвостом последовательности, то

// устанавливаем указатель на следующий элемент текущего элемента на новый

// элемент

if (sentence->cur->next == NULL) {

sentence->cur->next = cur;

} else {

// Иначе устанавливаем указатель на следующий элемент нового элемента

cur->next = sentence->cur->next;

sentence->cur->next = cur;

}

}

// Функция, сдвигающая указатель на текущий элемент вперед на один элемент

void st\_shift\_forward(st\_set\* sentence) {

// Проверка, что указатель на текущий элемент не равен NULL

if (sentence->cur != NULL) {

// Сдвиг указателя на текущий элемент вперед на один элемент

sentence->cur = sentence->cur->next;

}

}

// Функция, выводящая на экран следующий элемент после текущего указателя

void st\_print\_next(st\_set\* sentence) {

// Вывод сообщения о следующем элементе

printf("Следующий элемент: ");

// Указатель на текущую позицию в строке

char\* ptr;

// Цикл для нахождения длины строки

for (ptr = sentence->cur->next->data; \*ptr; ++ptr)

;

// Цикл для вывода символов строки

for (int i = 0; i < ptr - sentence->cur->next->data; i++) {

// Вывод символа

printf("%c", sentence->cur->next->data[i]);

}

// Переход на новую строку после вывода строки

printf("\n");

}

// Функция, изменяющая следующий элемент после текущего указателя

void st\_change\_next(st\_set\* sentence, char\* data) {

// Изменение данных следующего элемента

sentence->cur->next->data = data;

}

// Функция для ввода слова с клавиатуры

char\* str\_input() {

printf("Введите слово\n");

// Счетчик для хранения размера строки

size\_t count = 0;

// Буфер для хранения символа, введенного с клавиатуры

char buffer = getchar();

char\* str = NULL; // Указатель на строку, которую будем возвращать

// Выделение памяти под первый символ строки

str = (char\*)malloc(sizeof(char));

// Проверка на ошибку выделения памяти

if (str == NULL) {

printf("Ошибка, NULL\n");

exit(1);

}

// Цикл для считывания символов с клавиатуры, пока не будет нажат Enter

while (buffer != '\n' && buffer != '\0' && buffer != ' ') {

// Запись введенного символа в строку

str[count] = buffer;

// Инкремент счетчика размера строки

count++;

// Считываем символы, пока они не равны переводу строки или концу файл

buffer = getchar();

// Выделяем память под ещё один символ

str = (char\*)realloc(str, sizeof(char) \* count + 1);

// Проверка на ошибку выделения памяти

if (str == NULL) {

printf("Ошибка, NULL\n");

exit(1);

}

}

// Добавляем нулевой символ в конец строки

str[count] = '\0';

// Возвращаем считанное слово

return str;

}