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*

НИКУЛИНА ДМИТРИЯ ЕВГЕНЬЕВИЧА

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 _____
(подпись)

МОСКВА 2022

Оглавление

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»

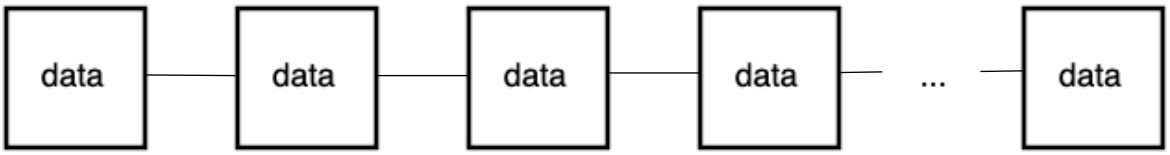
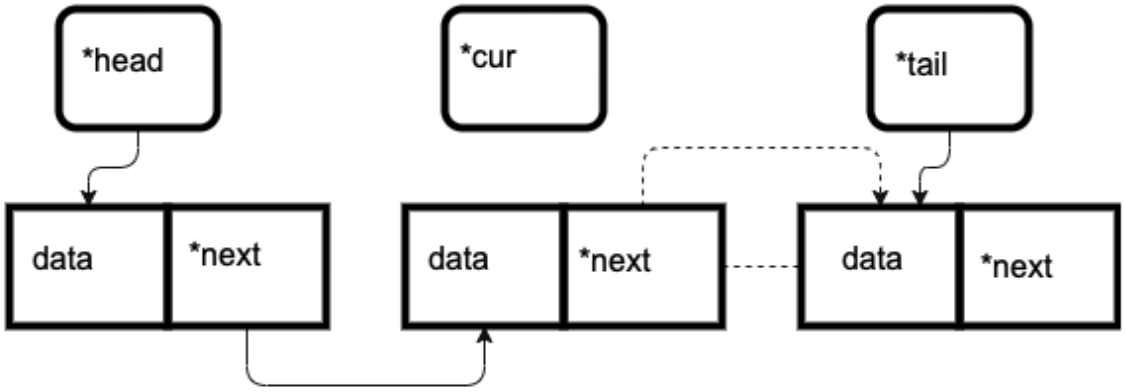


Схема физической структуры «Последовательность» (на базе односвязного списка)



*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»

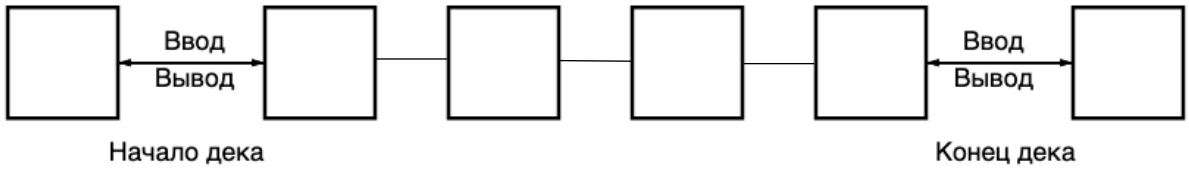
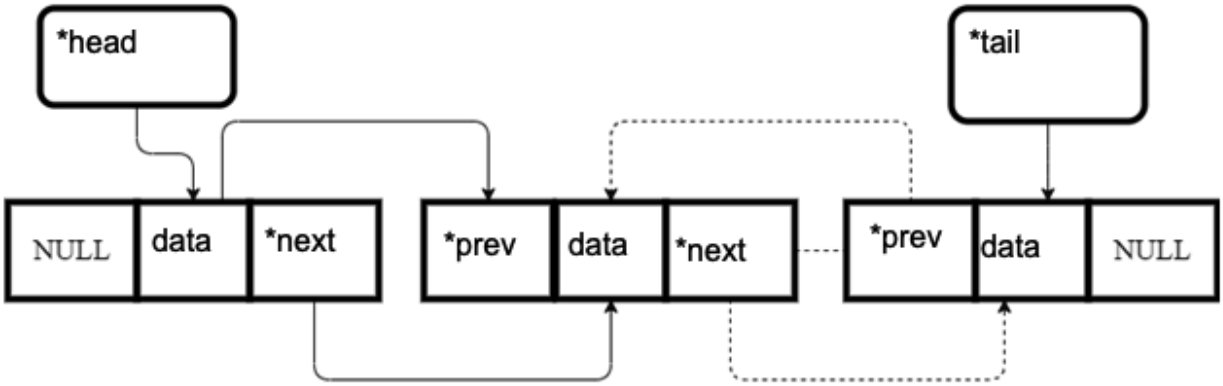


Схема физической структуры «Дек» (на базе двусвязного списка)



*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»

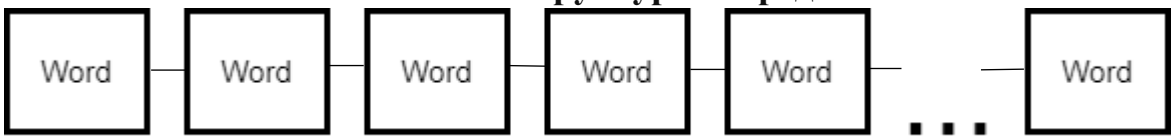
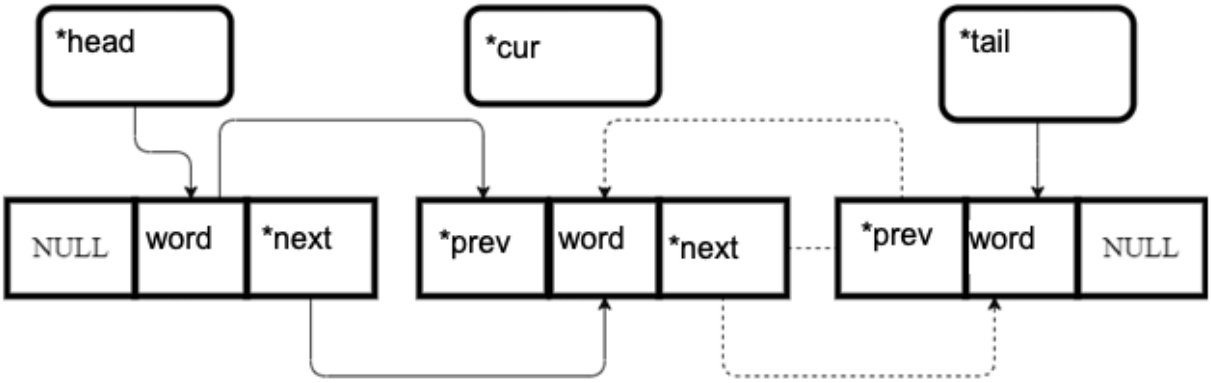


Схема физической структуры «Предложение» (на базе односвязного списка)

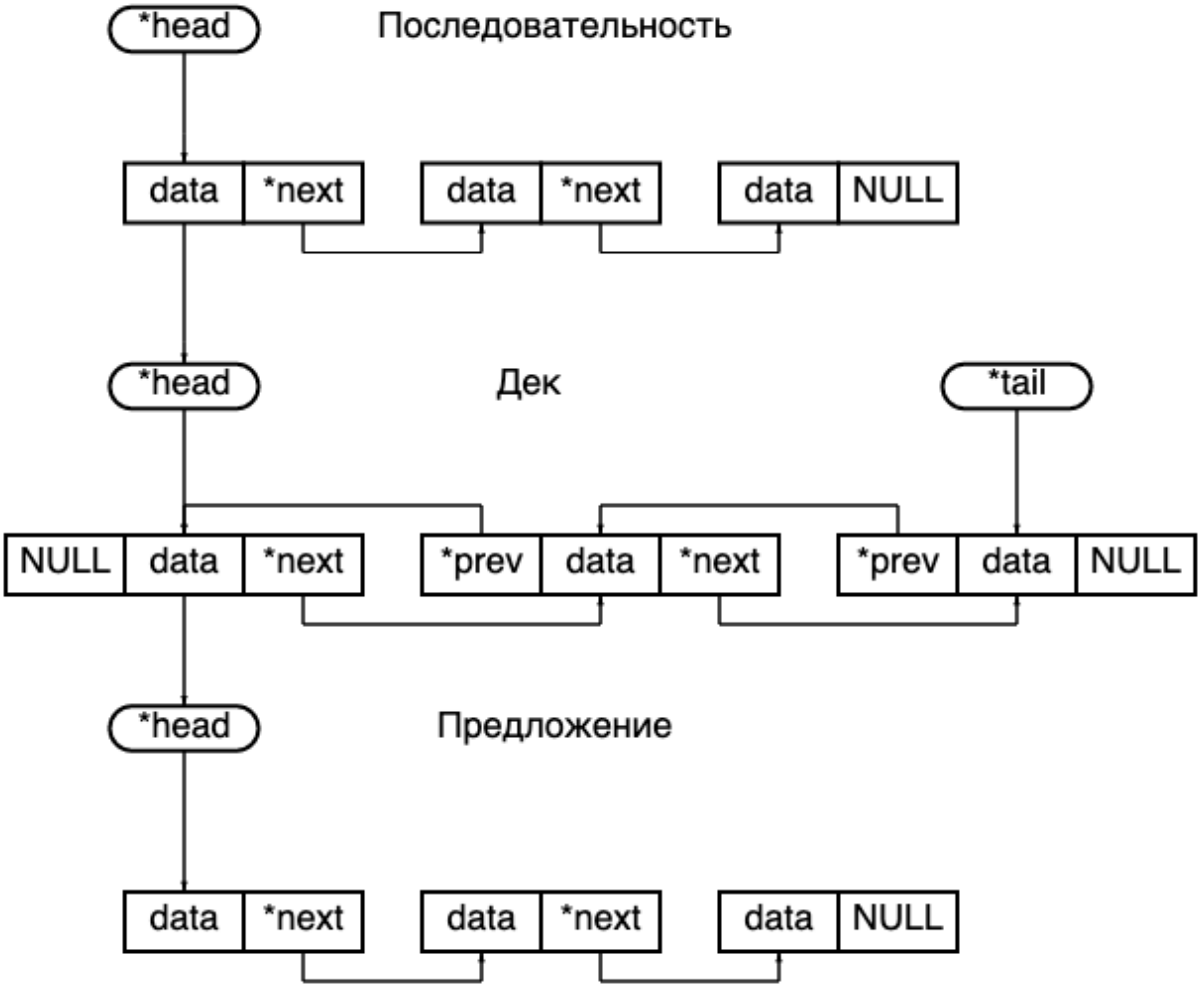


*head	Указатель на первый элемент списка
*cur	Рабочий указатель списка
data	Слово, хранимое в узле списка
*next	Указатель на следующий элемент списка
*prev	Указатель на предыдущий элемент списка

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;
```

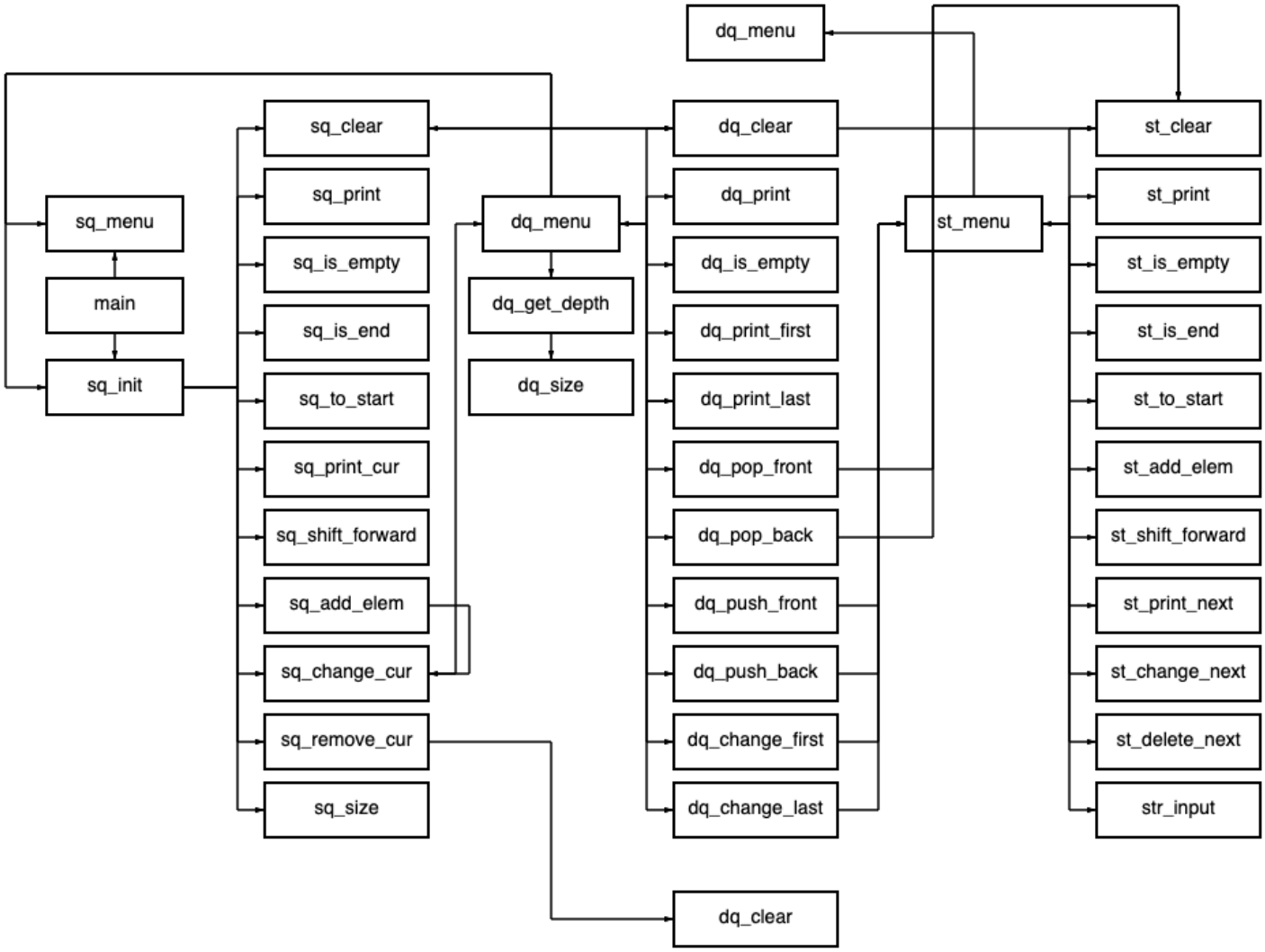
Дек:

```
// 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;196mC\033[38;5;202mT\033[38;5;208mA\033[38;5;"
        "214mP\033[38;5;220mT\033[38;5;226mO\033[38;5;190mB\033[38;5;154mO\033["
        "38;5;118mE\033[38;5;82m "
        "\033[38;5;47mM\033[38;5;82mE\033[38;5;118mH\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;196mM\033[38;5;202mE\033[38;5;208mH\033[38;5;"
        "214mЮ\033[38;5;220m "
        "\033[38;5;226mП\033[38;5;190mO\033[38;5;154mC\033[38;5;"
        "118mL\033[38;5;82mE\033[38;5;46mД\033[38;5;47mO\033[38;5;"
        "48mB\033[38;5;49mA\033[38;5;50mT\033[38;5;51mE\033[38;5;"
        "52mЛ\033[38;5;53mЬ\033[38;5;54mH\033[38;5;55mO\033[38;5;"
        "56mC\033[38;5;57mT\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;196mC\033[38;5;202mT\033[38;5;208mA\033[38;5;"
        "214mP\033[38;5;220mT\033[38;5;226mO\033[38;5;190mB\033[38;5;"
        "154mO\033["
        "38;5;118mE\033[38;5;82m "
        "\033[38;5;47mM\033[38;5;82mE\033[38;5;118mH\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;196mM\033[38;5;202mE\033[38;5;208mH\033[38;5;214mЮ\033[38;5;"
        "220m "
        "\033[38;5;226mД\033[38;5;190mE\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;196mC\033[38;5;202mT\033[38;5;208mA\033[38;5;"
        "214mP\033[38;5;220mT\033[38;5;226mO\033[38;5;190mB\033[38;5;"
        "154mO\033["
        "38;5;118mE\033[38;5;82m "
        "\033[38;5;47mM\033[38;5;82mE\033[38;5;118mH\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;196mC\033[38;5;202mT\033[38;5;208mA\033[38;5;"
          "214mP\033[38;5;220mT\033[38;5;226mO\033[38;5;190mB\033[38;5;"
          "154mO\033["
          "38;5;118mE\033[38;5;82m "
          "\033[38;5;47mM\033[38;5;82mE\033[38;5;118mH\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;
}
```