МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Институт математики и информационных систем
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Дата сдачи на проверку:

«__»_____ 2025 г.

Проверено:

«__»_____ 2025 г.

РЕАЛИЗАЦИЯ ЭЛЕМЕНТАРНЫХ СТРУКУР ДАННЫХ НА ОСНОВЕ ДИНАМИЧЕСКОЙ ПАМЯТИ

Отчёт по лабораторной работе №6

по дисциплине

«Программирование»

Разработал студент гр. ИВТб-1301-05-00 _____/Черкасов А. А./

(подпись)

Заведующая кафедры ЭВМ _____/Должженкова М. Л./

(подпись)

Работа защищена «__»_____ 2025 г.

Киров

2025

Цель

Цель работы: Изучение структуры и принципов организации программных модулей, закрепление навыков работы с динамической памятью. Получение базовых навыков организации работы в режиме командной строки.

Задание

- Написать программу для работы со структурой данных "Кольцевой двусвязный список".
- Структура данных должна быть реализована на основе динамической памяти.
- Структура данных (поля и методы) должна быть описана в отдельном модуле.
- Работа со структурой должна осуществляться в режиме командной строки (с реализацией автодополнения и истории команд).
- Предусмотреть наглядную визуализацию содержимого структуры.

Решение

Схемы алгоритмов решения задач представлена на рисунках 1 и 2. Исходный код решений представлен в Приложениях А1 и А2.

Рисунок 1 - Схема алгоритма Задания 1.

Рисунок 2 - Схема алгоритма Задания 2.

Вывод

В результате работы были реализована консольная программа для работы со структурой "Кольцевой двусвязный список"с использованием программных модулей.

Приложение А1. Исходный код

```
#ifndef CDLL_H
#define CDLL_H

typedef struct Node {
  char *data;
  struct Node *next;
  struct Node *prev;
} Node;

typedef struct {
  Node *head;
} CDLLists;

Node *createNode(const char *data);

void append(CDLLists *list, const char *data);

void display(const CDLLists *list);

void deleteNode(CDLLists *list, const char *data);

void freeList(CDLLists *list);

#endif // CDLL_H
```

Приложение А2. Исходный код

```
#include "cdll.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
Node *createNode(const char *data) {
  Node *newNode = (Node *)malloc(sizeof(Node));
  if (!newNode) {
    fprintf(stderr, "Memory allocation failed! :( \n");
    exit(EXIT_FAILURE);
  }
  newNode->data = (char *)malloc(strlen(data) + 1);
  if (!newNode->data) {
    fprintf(stderr, "Memory allocation for data failed! :( \n");
    free(newNode);
    exit(EXIT_FAILURE);
  }
  strcpy(newNode->data, data);
  newNode->next = newNode;
  newNode->prev = newNode;
  return newNode;
}

void append(CDLLists *list, const char *data) {
  if (!list->head) {
    list->head = createNode(data);
    return;
  }

  Node *current = list->head;
  do {
    if (strcmp(current->data, data) == 0) {
      printf("Элемент '%s' уже существует в списке.\n", data);
      return;
    }
    current = current->next;
  } while (current != list->head);

  Node *newNode = createNode(data);
```

```
  Node *tail = list->head->prev;
  tail->next = newNode;
  newNode->prev = tail;
  newNode->next = list->head;
  list->head->prev = newNode;
}

void display(const CDLLists *list) {
  if (!list->head) {
    printf("Список пустой :(.\n");
    return;
  }

  Node *current = list->head;
  do {
    printf("%s <-> ", current->data);
    current = current->next;
  } while (current != list->head);
  printf("(head)\n");
}

void deleteNode(CDLLists *list, const char *data) {
  if (!list->head) {
    printf("Список пуст.\n");
    return;
  }

  Node *current = list->head;

  do {
    if (strcmp(current->data, data) == 0) {
      if (current->next == current) {
        free(current->data);
        free(current);
```

```
        list->head = NULL;
      } else {
        current->prev->next = current->next;
        current->next->prev = current->prev;
        if (current == list->head) {
          list->head = current->next;
        }
        free(current->data);
        free(current);
      }
      printf("Элемент '%s' удалён.\n", data);
      return;
    }
    current = current->next;
  } while (current != list->head);

  printf("Элемент '%s' не найден.\n", data);
}

void freeList(CDLLists *list) {
  if (!list->head)
    return;

  Node *current = list->head;
  Node *temp;
  do {
    temp = current;
    current = current->next;
    free(temp->data);
    free(temp);
  } while (current != list->head);

  list->head = NULL;
}
```

Приложение А3. Исходный код

```c
#include "cdll.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <readline/history.h>
#include <readline/readline.h>

void print_help() {
  printf("Доступные команды:\n");
  printf("  append <data>  - Добавить элемент в список\n");
  printf("  display        - Показать содержимое списка\n");
  printf("  delete <data>  - Удалить элемент из списка\n");
  printf("  clear          - Очистить экран\n");
  printf("  help           - Показать это сообщение\n");
  printf("  exit           - Выйти из программы\n");
}

char *commands[] = {"append", "display", "delete", "clear",
                    "help",   "exit",    NULL};

char *command_generator(const char *text, int state) {
  static int index, len;
  if (!state) {
    index = 0;
    len = strlen(text);
  }

  char *name;
  while ((name = commands[index++])) {
    if (strncmp(name, text, len) == 0) {
      if (strcmp(name, "append") == 0 || strcmp(name, "delete") == 0) {
        char *name_with_space = malloc(strlen(name) + 2);
```

```
        sprintf(name_with_space, "%s ", name);
        return name_with_space;
      }
      return strdup(name);
    }
  }
  return NULL;
}

char **cli_completion(const char *text, int start, int end) {
  rl_attempted_completion_over = 1;
  rl_completion_append_character = '\0';
  return rl_completion_matches(text, command_generator);
}

int parse_command(const char *command) {
  if (strncmp(command, "append ", 7) == 0)
    return 1;
  if (strcmp(command, "display") == 0)
    return 2;
  if (strncmp(command, "delete ", 7) == 0)
    return 3;
  if (strcmp(command, "clear") == 0)
    return 5;
  if (strcmp(command, "help") == 0)
    return 6;
  if (strcmp(command, "exit") == 0)
    return 4;
  return 0;
}

void clear_screen() { printf("\033[H\033[J"); }

int main() {
```

```
CDLLists list = {NULL};
char *input;

rl_attempted_completion_function = cli_completion;

printf("CDLL CLI. Введите 'help' для списка команд.\n");

while ((input = readline(">> ")) != NULL) {
  if (strlen(input) == 0) {
    free(input);
    continue;
  }

  add_history(input);

  int cmd = parse_command(input);
  switch (cmd) {
  case 1: { // append
    char data[256];
    if (sscanf(input + 7, "%255s", data) == 1) {
      append(&list, data);
    } else {
      printf("Ошибка формата: append <data>\n");
    }
    break;
  }

  case 2: // display
    display(&list);
    break;

  case 3: { // delete
    char data[256];
    if (sscanf(input + 7, "%255s", data) == 1) {
```

```
            deleteNode(&list, data);
        } else {
            printf("Ошибка формата: delete <data>\n");
        }
        break;
    }

    case 4: // exit
        freeList(&list);
        free(input);
        return 0;

    case 5: // clear
        clear_screen();
        break;

    case 6: // help
        print_help();
        break;

    default:
        printf("Неизвестная команда. Напишите 'help'.\n");
    }
    free(input);
  }

  freeList(&list);
  return 0;
}
```