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 реализованы алгоритмы сортировки вставками и пирамидальной сортировки с поддержкой настраиваемого компаратора и обработки текстовых файлов, что позволило оценить их эффективность и сравнить скорость работы.

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) {
  Node *newNode = createNode(data);
  if (!list->head) {
    list->head = newNode;
  } else {
    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)
    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);
      }
```

```
      return;
    }
    current = current->next;
  } while (current != list->head);

  printf("Такого элемента нет в списке: '%s'\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

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

#ifdef _WIN32
```

```
#include <conio.h>
#include <windows.h>
#else
#include <termios.h>
#include <unistd.h>
#endif

#define MAX_HISTORY 100
#define MAX_COMMAND 256

typedef struct {
  char items[MAX_HISTORY][MAX_COMMAND];
  int count;
  int position;
} History;

History history = {0};

#ifdef _WIN32
HANDLE hStdin;
DWORD fdwSaveOldMode;
#else
struct termios orig_termios;
#endif

void enable_raw_mode() {
#ifdef _WIN32
  // Сохраняем текущий режим консоли и включаем расширенный ввод
  hStdin = GetStdHandle(STD_INPUT_HANDLE);
  GetConsoleMode(hStdin, &fdwSaveOldMode);
  SetConsoleMode(hStdin, ENABLE_EXTENDED_FLAGS | ENABLE_WINDOW_INPUT);
#else
  // Отключаем эхо и канонический режим для терминала
```

```
  struct termios raw = orig_termios;
  raw.c_lflag &= ~(ECHO | ICANON);
  tcsetattr(STDIN_FILENO, TCSAFLUSH, &raw);
#endif
}

void disable_raw_mode() {
#ifdef _WIN32
  SetConsoleMode(hStdin, fdwSaveOldMode);
#else
  tcsetattr(STDIN_FILENO, TCSAFLUSH, &orig_termios);
#endif
}

void init_terminal() {
#ifndef _WIN32
  tcgetattr(STDIN_FILENO, &orig_termios);
  atexit(disable_raw_mode);
#endif
}

void clear_screen() {
#ifdef _WIN32
  system("cls");
#else
  printf("\033[H\033[J");
#endif
}

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 *readline(const char *prompt) {
  static char line[MAX_COMMAND] = {0};
  printf("%s", prompt);
  fflush(stdout);

  enable_raw_mode();
  int pos = 0;
  char seq[3];

  while (1) {
    char c;
#ifdef _WIN32
    if (!_kbhit())
      continue;
    c = _getch();
#else
    read(STDIN_FILENO, &c, 1);
#endif

    // Обработка автодополнения
    if (c == '\t') {
      const char *commands[] = {"append", "display", "delete",
                                "clear",  "help",    "exit"};
      int matches = 0;
      char *match = NULL;

      for (int i = 0; i < 6; i++) {
```

```
    if (strncmp(commands[i], line, pos) == 0) {
      if (!match)
        match = (char *)commands[i];
      matches++;
    }
  }

  if (matches == 1) {
    strcpy(line, match);
    pos = strlen(match);
    printf("\r>> %s", line);
    fflush(stdout);
  } else if (matches > 1) {
    printf("\nВозможные команды:\n");
    for (int i = 0; i < 6; i++) {
      if (strncmp(commands[i], line, pos) == 0) {
        printf("  %s\n", commands[i]);
      }
    }
    printf(">> %s", line);
    fflush(stdout);
  }
  continue;
}

// Обработка ввода Enter
if (c == '\r' || c == '\n') {
  line[pos] = '\0';
  printf("\n");
  disable_raw_mode();
  return line;
}
```

```
    // Обработка удаления символа
    if (c == '\b' || c == 127) {
      if (pos > 0) {
        pos--;
        line[pos] = '\0';
        printf("\b \b");
        fflush(stdout);
      }
      continue;
    }

#ifdef _WIN32
    if (c == '\0') {
      switch (_getch()) {
      case 72:
        if (history.position > 0) {
          history.position--;
          strcpy(line, history.items[history.position]);
          pos = strlen(line);
          printf("\r>> %s", line);
          fflush(stdout);
        }
        break;
      case 80:
        if (history.position < history.count - 1) {
          history.position++;
          strcpy(line, history.items[history.position]);
          pos = strlen(line);
          printf("\r>> %s", line);
          fflush(stdout);
        }
        break;
      }
```

```
      continue;
    }
#else
    if (c == '\x1B') {
      if (read(STDIN_FILENO, &seq[0], 1) != 1 || seq[0] != '[')
        continue;
      if (read(STDIN_FILENO, &seq[1], 1) != 1)
        continue;

      switch (seq[1]) {
      case 'A':
        if (history.position > 0) {
          history.position--;
          strcpy(line, history.items[history.position]);
          pos = strlen(line);
          printf("\r>> %s", line);
          fflush(stdout);
        }
        break;
      case 'B':
        if (history.position < history.count - 1) {
          history.position++;
          strcpy(line, history.items[history.position]);
          pos = strlen(line);
          printf("\r>> %s", line);
          fflush(stdout);
        }
        break;
      }
      continue;
    }
#endif
```

```
    if (pos < MAX_COMMAND - 1 && isprint(c)) {
      line[pos++] = c;
      printf("%c", c);
      fflush(stdout);
    }
  }
}

void add_history(const char *cmd) {
  if (strlen(cmd) == 0 ||
      (history.count > 0 &&
       strcmp(history.items[history.count - 1], cmd) == 0)) {
    return; // Avoid adding empty or duplicate commands
  }
  if (history.count >= MAX_HISTORY) {
    memmove(history.items[0], history.items[1],
            sizeof(history.items[0]) * (MAX_HISTORY - 1));
    history.count--;
  }
  strncpy(history.items[history.count++], cmd, MAX_COMMAND - 1);
  history.items[history.count - 1][MAX_COMMAND - 1] =
      '\0'; // Ensure null-termination
  history.position = history.count;
}

int parseCommand(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;
}

int main() {
  CDLLists list = {NULL};
  init_terminal();

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

  while (1) {
    char *input = readline(">> ");
    if (strlen(input) == 0)
      continue;

    add_history(input);

    int cmd = parseCommand(input);
    switch (cmd) {
    case 1: {
      char data[MAX_COMMAND];
      if (sscanf(input + 7, "%s", data) != 1) {
        printf("Ошибка: Неверный ввод для 'append'. append <data>\n");
        break;
      }
      append(&list, data);
      printf("Добавлено: %s\n", data);
      break;
    }
```

```
    case 2:
      display(&list);
      break;
    case 3: {
      char data[MAX_COMMAND];
      if (sscanf(input + 7, "%s", data) != 1) {
        printf("Ошибка: Неверный ввод для 'delete'. delete <data>\n");
        break;
      }
      deleteNode(&list, data);
      printf("Удалено: %s\n", data);
      break;
    }
    case 4:
      freeList(&list);
      return 0;
    case 5:
      clear_screen();
      break;
    case 6:
      print_help();
      break;
    default:
      printf("'help' для списка доступных команд.\n");
    }
  }
}
```