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:

«__»_____ 2024 г.

Проверено:

«__»_____ 2024 г.

РЕАЛИЗАЦИЯ И РАБОТА С ДВУСВЯЗНЫМ СПИСКОМ

Отчёт по лабораторной работе №3

по дисциплине

«Программирование»

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

(подпись)

Заместитель кафедры ЭВМ _____/Долженкова М. Л./

(подпись)

Работа защищена «__»_____ 2024 г.

Киров

2024

Цель

Цель работы: Изучить структуру данных "двусвязный список"и принципы работы с ним. Реализовать операции добавления, удаления и вывода элементов списка. Разработать программу с интерактивным вводом данных и поддержкой управления списком через команды.

Задание

- Реализовать двусвязный список с помощью структур: Узел списка должен содержать значение (символ) и указатели на следующий и предыдущий узлы. Список должен содержать указатели на первый и последний узлы, а также размер списка.
- Реализовать функции для создания узлов и списка, добавления символов в конец списка, удаления узлов, вывода содержимого списка.
- Организовать интерактивную работу программы с командами управления: ввод символов, удаление символов (с помощью '&'), завершение ввода (с помощью '.') и выход из программы ("exit").

Теоретическая часть

Двусвязный список

Двусвязный список — это структура данных, состоящая из элементов, называемых узлами, где каждый узел хранит:

- **Значение (данные)** — информация, которую хранит узел (например, символ или число).

– **Ссылка на следующий узел (next)** — указатель на следующий элемент в списке.

– **Ссылка на предыдущий узел (prev)** — указатель на предыдущий элемент в списке.

Основное преимущество двусвязного списка — возможность проходить как в одну, так и в другую сторону, что даёт гибкость при реализации алгоритмов.

Преимущества

– **Добавление/удаление**: Быстрое добавление и удаление элементов как в начале, так и в конце списка, а также в середине. Операции вставки и удаления не требуют сдвига элементов, как это происходит в массивах.

– **Доступ**: Доступ к элементам двусвязного списка может быть как в прямом, так и в обратном направлении (вперёд/назад), что расширяет возможности для обхода и манипуляций с данными.

Недостатки

– **Память**: Каждый узел требует дополнительной памяти для хранения ссылки на предыдущий и следующий узел. Это увеличивает расход памяти по сравнению с односвязными списками или массивами.

– **Сложность управления**: Необходимо тщательно управлять ссылками на предыдущие и следующие узлы, особенно при удалении элементов, что требует дополнительных проверок и вычислений.

Сравнение

Таблица 1 – Сравнение структур.

Критерий	Двусвязный список	Односвязный список
Память	Больше	Меньше
Добавление/удаление	Быстро	Быстро
Доступ по индексу	Медленно (O(n))	Медленно (O(n))
Обход	Двунаправленный	Однонаправленный
	Массив	Стек
Память	Определена заранее	Столько же, сколько и в списке
Добавление/удаление	Медленно	Быстро
Доступ по индексу	Быстро (O(1))	Нет
Обход	Нет	Нет

Решение

Схема алгоритма решения задачи представлена на рисунках ниже. Исходный код представлен в Приложении А1.

Рисунок 1.1 - Схема алгоритма.

Рисунок 1.2 - Схема алгоритма.

Рисунок 1.3 - Схема алгоритма.

Вывод

В ходе выполнения работы изучены основы работы с двусвязным списком, реализованы операции добавления и удаления элементов, а также организован интерактивный пользовательский интерфейс для управления списком.

Приложение А1. Исходный код

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

// Структура узла списка
typedef struct Node {
    char value;
    struct Node* next;
    struct Node* prev;
} Node;

// Структура списка
typedef struct LList {
    size_t size;
    Node* head;
    Node* tail;
} LList;

// Создает новый узел
Node* createNode(char value) {
    Node* node = (Node*)malloc(sizeof(Node));
    node->value = value;
    node->prev = node->next = NULL;
    return node;
}

// Создает новый список
LList* createList() {
    LList* list = (LList*)malloc(sizeof(LList));
    list->size = 0;
    list->head = list->tail = NULL;
    return list;
```

```
}

// Удаляет список
void deleteList(LList* list) {
    Node* tmp = list->head;
    Node* next = NULL;
    while (tmp) {
        next = tmp->next;
        free(tmp);
        tmp = next;
    }
    free(list);
}

// Добавляет символ в список
void append(LList* list, char value) {
    Node* node = createNode(value);
    node->prev = list->tail;
    if (list->tail)
        list->tail->next = node;
    list->tail = node;
    if (list->head == NULL)
        list->head = node;
    list->size++;
}

// Удаляет узел из списка
void deleteNode(LList* list, Node* node) {
    if (node == NULL) return;
    if (node->prev)
        node->prev->next = node->next;
    if (node->next)
        node->next->prev = node->prev;
```

```
    if (list->head == node)
        list->head = node->next;
    if (list->tail == node)
        list->tail = node->prev;
    list->size--;
    free(node);
}

// Печатает список
void printList(LList* list) {
    Node* current = list->head;
    while (current) {
        printf("%c", current->value);
        current = current->next;
    }
    printf("\n");
}

int main() {
    LList* list = createList();
    char ch;
    char exitBuffer[6] = {0};
    int index = 0;

    printf("Введите последовательность лат. символов оканчивающюся '.',\
 '&' - для удаления предыдущего символа, 'exit' - для выхода.\n");

    int exit = 1;

    while (exit) {
        ch = getchar();

        exitBuffer[index % 5] = ch;
```

```
        index++;

        // Проверка на "exit"
        if (index >= 5) {
            if (exitBuffer[(index - 5) % 5] == 'e' &&
                exitBuffer[(index - 4) % 5] == 'x' &&
                exitBuffer[(index - 3) % 5] == 'i' &&
                exitBuffer[(index - 2) % 5] == 't') {

                printf("Вы ввели 'exit'. Выход...\n");
                exit = 0;
            }
        }

        // Завершение ввода на '.'
        if (ch == '.') {
            printList(list);
            deleteList(list);
            list = createList(); // Очистка и создание нового списка
            continue;
        }

        if (ch == '&') {
            deleteNode(list, list->tail); // Удаление последнего символа
        } else {
            append(list, ch); // Добавление символа
        }
    }

    deleteList(list);
    return 0;
}
```