**Федеральное государственное автономное образовательное учреждение высшего образования**

**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Инженерная академия**

**Кафедра механики и процессов управления**

КУРСОВАЯ РАБОТА

«РЕАЛИЗАЦИЯ ОДНОСВЯЗНОГО СПИСКА НА ЯЗЫКЕ СИ»

направление подготовки «Управление в технических системах». Квалификация «бакалавр».

Разработчик **Лещенко В.А.**

Государство РФ

Студент группы ИУСбд-01-23

Студенческий билет № 1132231954

Научный руководитель: Жуков В.И.

Оценка. ECTS\_\_\_\_\_ Балл \_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| Оригинальность в "Антиплагиат" |  | % |

*проставляется научным руководителем*

Москва

2024

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 3](#_Toc514781219)

[**ГЛАВА I. ТЕОРИЯ** 4](#_Toc514781220)

[**1. Основные понятия** 4](#_Toc514781221)

[**2. Односвязный список** 4](#_Toc514781222)

[**3. Сравнение с массивами** 4](#_Toc514781223)

[**ГЛАВА II. ПРАКТИКА** 5](#_Toc514781224)

[**1. Алгоритмы основных операций** 5](#_Toc514781225)

[**2. Методы тестирования** 5](#_Toc514781226)

[**ЗАКЛЮЧЕНИЕ** 7](#_Toc514781232)

[**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ** 9](#_Toc514781234)

**ВВЕДЕНИЕ**

**ССЫЛКА НА ГИТХАБ:** **https://github.com/LeschV/proga/blob/main/Laba/kurs.c**

Современные информационные технологии требуют эффективной организации и обработки данных. Производительность и надежность программного обеспечения во многом зависят от выбора правильной и качественной структуры данных. Среди множества структур данных связанный список выделяется своей гибкостью и простотой реализации, что делает его незаменимым инструментом в различных областях программирования. В этой курсовой работе рассматривается реализация связанного списка на языке программирования Си.

Актуальность темы заключается в том, что связанные списки широко используются для решения задач, требующих динамического изменения размера данных. В отличие от массивов, связанные списки обладают преимуществом динамического распределения памяти, что позволяет эффективно работать с данными переменного размера.

Цель работы — разработка и реализация библиотеки функций для работы с односвязным списком на языке Си, включающей операции создания, добавления, удаления, поиска и вывода элементов.

Основные задачи:

• Создание структуры данных для узла списка

• Реализация алгоритмов основных операций

• Тестирование реализованных функций

• Оценка эффективности

Результаты работы могут послужить основой для дальнейшей разработки более сложных структур данных, а также для решения практических задач, требующих использования динамических структур данных.**ГЛАВА I. ТЕОРИЯ**

**1. Основные понятия**

Структура данных – это программная единица, описывающая способ организации ячеек памяти, хранящих данные одного и того же типа.

Временная сложность – зависимость выполнения операции от количества элементов в структуре данных (обычно выражается в нотации "Большое O"). Например, O(1) означает постоянное время, независимое от размера данных, O(n) — линейное время, зависящее от размера данных, O(log n) — логарифмическое время и т.д.

Классификация структур данных:

По сложности:

• Простые (базовые, примитивные);

• Интегрированные (композитные, сложные).

По связи между элементами:

• Связные (связные списки);

• Несвязные (векторы, массивы, строки, очереди).

Линейные структуры

по расположению элементов в памяти:

• Последовательные (векторы, строки, массивы, стеки, очереди);

• Произвольные (односвязные, двусвязные списки).

Нелинейные структуры

• Многосвязные списки, деревья, графы.

**2. Односвязный список**

Односвязный список — это линейная динамическая структура данных, элементы которой (узлы) не хранятся в непрерывной области памяти, а связаны между собой с помощью указателей. Каждый узел списка содержит два основных компонента:

• Данные: Информация, хранящаяся в узле. Тип данных может быть любым (integer, float, char, struct и т.д.).

• Указатель (ссылка): Адрес памяти следующего узла в списке. Последний узел списка имеет указатель, равный NULL (или 0), сигнализирующий об окончании списка.

Преимущества:

Динамическое распределение памяти: память выделяется по мере необходимости, что позволяет эффективно работать с данными переменного размера.

Быстрая вставка и удаление элементов: вставка или удаление узла в произвольном месте списка требует изменения указателей только соседних узлов.

Недостатки:

Доступ к элементам по индексу медленный: для доступа к i-ому элементу необходимо пройти по списку от начала до i-го элемента.

Дополнительная память на указатели: для хранения указателей требуется дополнительная память.

Невозможность обратного прохода: Перемещение по списку возможно только в одном направлении.

**3. Сравнение с массивами**

Массивы и односвязные списки — две основные линейные структуры данных. Они отличаются по способу хранения данных и, следовательно, по эффективности выполнения различных операций.

|  |  |  |
| --- | --- | --- |
| **Характеристика** | **Массив** | **Односвязный список** |
| Размер | фиксированный | динамический |
| Использование памяти | непрерывный блок памяти | непрерывный блок памяти не требуется |
| Доступ к элементам | О(1) | О(n) |
| Вставка/удаление элемента | О(n) | О(1), при условии знания позиции |

Выбор между массивом и односвязным списком зависит от конкретной задачи. Если необходим частый доступ к элементам по индексу, то предпочтительнее массив. Если требуется частое добавление и удаление элементов, то лучше использовать односвязный список.

**ГЛАВА II. ПРАКТИКА**

**1. Алгоритмы основных операций**

В этой части опишем алгоритмы основных операций над односвязным списком. Предполагается, что узел списка описывается структурой:

1. Создание списка (create()):

Алгоритм тривиален: возвращается указатель на NULL, представляющий пустой список.

2. Добавление элемента (add\_element()):

Существуют три основных варианта добавления элемента: в начало, в конец и по индексу.

* Добавление в начало: создается новый узел, его указатель указывает на текущую голову списка, а новый узел становится новой головой.
* Добавление в конец: необходимо пройти по всему списку до последнего узла, а затем добавить новый узел после него.
* Добавление по индексу: требуется найти узел с указанным индексом и вставить новый узел перед ним. Обработка случаев, когда индекс вне диапазона, необходима.

3. Поиск элемента (search\_element()):

Проходим по списку и сравниваем данные каждого узла с искомым значением. Выводится информация о том,найден объект или нет.

4. Вывод списка (print()):

Проходим по списку и выводим данные каждого узла.

5. Освобождение памяти (free\_list()):

Необходимо освободить память, выделенную для каждого узла списка, чтобы избежать утечек памяти. Проходим по списку, освобождая память для каждого узла.

**2. Методы тестирования**

1. Модульное тестирование:

Этот метод предполагает тестирование каждой функции библиотеки независимо от остальных. Цель — убедиться, что каждая функция работает правильно в изоляции. Для этого для каждой функции будут разработаны тестовые случаи, покрывающие различные сценарии её использования:

• Тестирование функции create(): Проверка успешного создания пустого списка.

• Тестирование функции add\_element(): Добавление элемента в начало, в конец и в середину списка; проверка корректности связей между узлами после добавления; обработка случая добавления в пустой список.

• Тестирование функции print(): Вывод содержимого списка различной длины; вывод пустого списка; проверка корректности вывода элементов в нужном порядке.

• Тестирование функции free\_list(): Проверка освобождения памяти, занимаемой всеми узлами списка; предотвращение утечек памяти.

Для каждого тестового случая будут определены ожидаемые результаты, и результаты выполнения функции будут сравниваться с ожидаемыми. В случае несовпадения будет регистрироваться ошибка.

2. Интеграционное тестирование:

После успешного модульного тестирования выполняется интеграционное тестирование, которое проверяет взаимодействие между различными функциями библиотеки. Цель — убедиться, что функции корректно работают вместе. Тестирование будет включать в себя:

• Последовательность операций: Проверка корректной работы последовательности операций над списком (например, добавление, затем поиск).

• Обработка ошибок: Проверка корректной обработки ошибок (например, попытка удалить несуществующий элемент, доступ к элементу за пределами списка).

3. Системное тестирование (опционально):

Если разработанная библиотека является частью более крупной системы, то выполняется системное тестирование, которое проверяет работу библиотеки в контексте всей системы. Это тестирование выходит за рамки данной курсовой работы, но может быть упомянуто в качестве перспективы дальнейшего развития.

**ЗАКЛЮЧЕНИЕ**

В данной курсовой работе была успешно реализована библиотека функций для работы с односвязным списком на языке программирования Си. Были разработаны и протестированы функции создания списка, добавления, удаления и поиска элементов, а также функция вывода содержимого списка и освобождения занимаемой им памяти. В ходе работы были изучены основные принципы работы с динамическими структурами данных, особенности реализации односвязного списка и методы тестирования программного обеспечения.

Разработанная библиотека прошла модульное и интеграционное тестирование, подтвердившее корректность работы всех функций в различных сценариях, включая граничные условия. Результаты тестирования показали эффективность реализованных алгоритмов, подтвердив теоретические оценки временной сложности операций.

В процессе работы над курсовой были получены практические навыки программирования на языке Си, углублено понимание принципов работы динамических структур данных и методов их тестирования. Разработанная библиотека может быть использована в качестве основы для создания более сложных структур данных или при решении различных практических задач, требующих использования динамических списков.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Кнут, Д. Искусство программирования. Том 1. Основные алгоритмы. – М.: Вильямс, 2006.
2. Вирт, Н. Алгоритмы и структуры данных. – СПб.: Невский Диалект, 2006.
3. Седжвик, Р. Алгоритмы на языке C. Части 1-4. – 3-е изд. – СПб.: Невский Диалект, 2002.
4. Овсянников А. В., Пикман Ю. А. Алгоритмы и структуры данных – Минск: БГУ, 2015.

# **ПРИЛОЖЕНИЕ**

Программный код основных функций:

#include <stdio.h>

#include <stdlib.h>

typedef struct Node \*link;

typedef char elemtype;

typedef struct Node

{

    elemtype elem;

    link next;

} node;

typedef link list;

list lst; /\* переменная типа список \*/

list create(char \*s)

{

    link cur;

    list res;

    if (\*s == '\0')

        return NULL;

    res = cur = (link)malloc(sizeof(node));

    cur->elem = \*s++;

    while (\*s != '\0')

    {

        cur = cur->next = (list)malloc(sizeof(node));

        cur->elem = \*s++;

    }

    cur->next = NULL;

    return res;

}

void print(list p)

{

    while (p != NULL)

    {

        putchar(p->elem);

        p = p->next;

    }

    putchar('\n');

}

void add\_element(elemtype elem, list \*lp)

{

    list cur = (list)malloc(sizeof(list));

    cur->elem = elem;

    cur->next = \*lp;

    \*lp = cur;

}

void free\_list(list ls)

{

    link q;

    while (ls != NULL)

    {

        q = ls;

        ls = ls->next;

        free(q);

    }

}

int main()

{

    char buffer[81];

    printf("Input sth(<80):");

    scanf("%s", &buffer);

    printf("\n");

    lst = create(buffer);

    add\_element('a', &lst);

    print(lst);

    free\_list(lst);

}