

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных
систем



Лабораторная работа №3.1

по дисциплине: Алгоритмы и структуры данных
по теме: «Структуры данных «линейные списки» (Pascal/C)»

Выполнил: студент группы ПВ-223
Мелехов Артём Дмитриевич

Проверили:
асс. Солонченко Роман Евгеньевич

Белгород 2023 г.

Лабораторная работа №5. Структуры данных «линейные списки» (Pascal/C)

Цель работы: изучить СД типа «линейный список», научиться их программно реализовывать и использовать.

Содержание отчета:

Решения заданий. Для каждого задания указаны:

- Название задания;
- Условие задания;
- Решение задания;

Вывод;

Полный листинг программы (с названиями директорий и файлов) и ссылка на репозиторий GitHub с выполненной работой.

Вариант №7.

Модуль: 7

Задача: 7

Задание №1.

Для СД типа «линейный список» определить:

Пункт 1:

Абстрактный уровень представления СД.

Решение:

Характер организованности и изменчивости:

Линейный список представляет собой упорядоченную коллекцию данных.

Элементы списка связаны между собой последовательно посредством указателей.

Порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера.

Вся внутренняя структура такого типа скрыта от разработчика программного обеспечения — в этом и заключается суть абстракции.

Набор допустимых операций:

```
// Создает новый список с заданной начальной вместимостью.
SequentialLinearList *create_list(size_t initial_capacity);

// Удаляет список и освобождает выделенную под него память.
void delete_list(SequentialLinearList *list);

// Вставляет элемент в указанную позицию.
void insert(SequentialLinearList *list, base_type element,
size_t position);

// Удаляет элемент из указанной позиции.
void erase(SequentialLinearList *list, size_t position);

// Возвращает элемент из указанной позиции.
base_type get(SequentialLinearList *list, size_t position);

// Перемещает указатель на n позиций (может быть отрицательным).
void move_ptr(SequentialLinearList *list, base_type n);

// Перемещает указатель на указанную позицию.
void move_to(SequentialLinearList *list, size_t position);
```

```

// Возвращает количество элементов в списке.
size_t count(SequentialLinearList *list);

// Проверяет, является ли список полным.
bool full_list(SequentialLinearList *list);

// Проверяет, является ли текущий элемент последним в списке.
bool end_list(SequentialLinearList *list);

// Копирует все элементы из списка src в список dest
void copy_list(SequentialLinearList *dest, SequentialLinearList
*src);

```

Пункт 2:

Физический уровень представления СД.

Решение:

Схемы хранения: Элементы списка хранятся в динамическом массиве.

Объём памяти, занимаемый экземпляром СД: Объём памяти, занимаемый экземпляром структуры данных, зависит от размера базового типа и вместимости списка. Для каждого элемента выделяется память под базовый тип (в нашем случае это `long long`), а также дополнительная память для переменных `size`, `capacity` и `ptr`.

Формат внутреннего представления СД и способ его интерпретации: Внутреннее представление – это динамический массив, где каждый элемент массива – это элемент списка. Интерпретация осуществляется через функции, определенные в коде, такие как `insert`, `erase`, `get` и другие.

Характеристика допустимых значений: Допустимые значения определяются базовым типом, который выбран (`long long`). Это означает, что в список можно добавить любое значение этого типа.

Тип доступа к элементам: Доступ к элементам осуществляется через функции, такие как `insert` (для добавления элемента), `erase` (для удаления элемента), `get` (для получения элемента) и другие. Это означает, что доступ к элементам является прямым и индексированным.

Пункт 3:

Логический уровень представления СД.

Решение:

Схемы хранения: На логическом уровне, линейный список представляет собой упорядоченную коллекцию элементов, где каждый элемент имеет определенную позицию. В случае последовательного линейного списка, элементы хранятся последовательно друг за другом.

Формат внутреннего представления СД и способ его интерпретации: На логическом уровне список представляется как упорядоченная последовательность элементов. Мы можем интерпретировать эту структуру данных как коллекцию элементов, доступ к которым осуществляется по их позиции в списке.

Характеристика допустимых значений: Допустимые значения определяются базовым типом, который выбран (`long long`). Это означает, что в список можно добавить любое значение этого типа.

Тип доступа к элементам: Доступ к элементам осуществляется через функции, такие как `insert` (для добавления элемента), `erase` (для удаления элемента), `get` (для получения элемента) и другие. Это означает, что доступ к элементам является прямым и индексированным.

Задание №2.

Реализовать СД типа «линейный список» в соответствии с вариантом индивидуального задания в виде модуля.

Решение:

Файл `sequential_linear_list.h`:

```
// sequential_linear_list.h
#ifndef SEQUENTIAL_LINEAR_LIST_H
#define SEQUENTIAL_LINEAR_LIST_H

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef long long base_type; // Замените на нужный тип данных

typedef struct {
    base_type *array; // Массив для хранения элементов списка
    size_t size; // Текущий размер списка
    size_t capacity; // Текущая вместимость списка
    size_t ptr; // Указатель на текущий элемент
} SequentialLinearList;

const short list_ok = 0;
const short list_not_mem = 1; // Ошибка выделения памяти
const short list_under = 2;
const short list_eEnd = 3;
short list_error;

// Создает новый список с заданной начальной вместимостью.
SequentialLinearList *create_list(size_t initial_capacity);

// Удаляет список и освобождает выделенную под него память.
void delete_list(SequentialLinearList *list);

// Вставляет элемент в указанную позицию.
void insert(SequentialLinearList *list, base_type element,
size_t position);

// Удаляет элемент из указанной позиции.
void erase(SequentialLinearList *list, size_t position);
```

```
// Возвращает элемент из указанной позиции.
base_type get(SequentialLinearList *list, size_t position);

// Перемещает указатель на n позиций (может быть отрицательным).
void move_ptr(SequentialLinearList *list, base_type n);

// Перемещает указатель на указанную позицию.
void move_to(SequentialLinearList *list, size_t position);

// Возвращает количество элементов в списке.
size_t count(SequentialLinearList *list);

// Проверяет, является ли список полным.
bool full_list(SequentialLinearList *list);

// Проверяет, является ли текущий элемент последним в списке.
bool end_list(SequentialLinearList *list);

// Копирует все элементы из списка src в список dest
void copy_list(SequentialLinearList *dest, SequentialLinearList
*src);

#endif // SEQUENTIAL_LINEAR_LIST_H
```

Файл sequential_linear_list.c:

```
//
// Created by Artyom on 31.10.2023.
//

#include "sequential_linear_list.h"
```

```

SequentialLinearList *create_list(size_t initial_capacity) {
    SequentialLinearList *list = (SequentialLinearList *)
    malloc(sizeof(SequentialLinearList));

    if (list == NULL) {
        //Ошибка выделения памяти для списка
        list_error = list_not_mem;

        return NULL;
    }

    list->array = (base_type *) malloc(initial_capacity *
    sizeof(base_type));

    if (list->array == NULL) {
        // Ошибка выделения памяти для массива
        free(list);
        list_error = list_not_mem;

        return NULL;
    }

    list->size = 0;
    list->capacity = initial_capacity;
    list->ptr = 0; // Устанавливаем указатель на начало списка
    list_error = list_ok;

    return list;
}

void delete_list(SequentialLinearList *list) {
    free(list->array);
    free(list);
}

```

```

void insert(SequentialLinearList *list, base_type element,
size_t position) {
    if (position > list->size) {
        // Позиция вставки выходит за пределы списка
        list_error = list_under;

        return;
    }

    if (list->size == list->capacity) {
        list->capacity *= 2;
        base_type *newArray = (base_type *) realloc(list->array,
list->capacity *
sizeof(base_type));

        if (newArray == NULL) {
            // Ошибка выделения памяти для расширения списка
            list_error = list_not_mem;

            return;
        }

        list->array = newArray;
    }

    for (size_t i = list->size; i > position; --i)
        list->array[i] = list->array[i - 1];

    list->array[position] = element;
    ++list->size;

    if (position <= list->ptr)
        ++list->ptr; // Обновляем указатель, если вставка была
перед ним

    list_error = list_ok;
}

```



```

void erase(SequentialLinearList *list, size_t position) {
    if (position >= list->size) {
        // Позиция удаления выходит за пределы списка
        list_error = list_under;

        return;
    }

    for (size_t i = position; i < list->size - 1; ++i)
        list->array[i] = list->array[i + 1];

    --list->size;

    if (position < list->ptr)
        --list->ptr; // Обновляем указатель, если удаление было
    перед ним

    list_error = list_ok;
}

base_type get(SequentialLinearList *list, size_t position) {
    if (position >= list->size) {
        // Позиция получения выходит за пределы списка
        list_error = list_under;

        return list_error;
    }

    list_error = list_ok;

    return list->array[position];
}

void move_ptr(SequentialLinearList *list, base_type n) {
    base_type new_position = (base_type) list->ptr + n; //
    Вычисляем новую позицию указателя

    if (new_position < 0 || new_position >= (int) list->size) {
        // Новая позиция указателя выходит за пределы списка
        list_error = list_under;

        return;
    }

    list->ptr = (size_t) new_position;
    list_error = list_ok;
}

```

```

void move_to(SequentialLinearList *list, size_t position) {
    if (position >= list->size) {
        // Позиция перемещения указателя выходит за пределы
        // списка
        list_error = list_under;

        return;
    }

    list->ptr = position;
    list_error = list_ok;
}

size_t count(SequentialLinearList *list) {
    return list->size;
}

bool full_list(SequentialLinearList *list) {
    return list->size == list->capacity ? 1 : 0;
}

bool end_list(SequentialLinearList *list) {
    return list->ptr == list->size - 1 ? 1 : 0;
}

void copy_list(SequentialLinearList *dest, SequentialLinearList
*src) {
    if (dest->capacity < src->size) {
        base_type *new_array = (base_type *) realloc(dest-
>array, src->size *
sizeof(base_type));

        if (new_array == NULL) {
            // Ошибка выделения памяти для копирования списка
            list_error = list_not_mem;

            return;
        }

        dest->array = new_array;
        dest->capacity = src->size;
    }

    for (size_t i = 0; i < src->size; ++i)
        dest->array[i] = src->array[i];

    dest->size = src->size;
    dest->ptr = src->ptr;
    list_error = list_ok;
}

```

Задание №3.

Разработать программу для решения задачи в соответствии с вариантом индивидуального задания с использованием модуля, полученного в результате выполнения пункта 2 задания.

Решение:

Файл lab5.c:

```
//  
// Created by Artyom on 01.11.2023.  
//  
  
#include "lab5.h"  
  
SequentialLinearList *sum_polynomials(SequentialLinearList *q,  
SequentialLinearList *r) {  
    // Создаем новый список для хранения результата.  
    SequentialLinearList *p = create_list(q->capacity > r-  
>capacity ? q->capacity :  
                                            r->capacity);  
  
    // Суммируем коэффициенты для каждой степени x.  
    for (size_t i = 0; i < p->capacity; i++) {  
        base_type q_coef = i < q->size ? get(q, i) : 0;  
        base_type r_coef = i < r->size ? get(r, i) : 0;  
        base_type sum_coef = q_coef + r_coef;  
  
        // Если суммарный коэффициент не равен нулю, добавляем  
        его в результат.  
        if (sum_coef != 0)  
            insert(p, sum_coef, i);  
    }  
  
    return p;  
}
```

Файл lab5.h:

```
//  
// Created by Artyom on 01.11.2023.  
//  
  
#ifndef ALGORITHMS_AND_DS_LAB5_H  
#define ALGORITHMS_AND_DS_LAB5_H  
  
#include "../data_structure/sequential_linear_list.h"  
  
// Суммирует два многочлена, представленных в виде списков.  
SequentialLinearList *sum_polynomials(SequentialLinearList *q,  
SequentialLinearList *r);  
  
#endif //ALGORITHMS_AND_DS_LAB5_H
```

Файл main.h:

```
#include "data_structure/sequential_linear_list.h"
#include "lab5/lab5.h"

typedef long long base_type; // Замените на нужный тип данных

// Здесь происходит запуск последней выполненной лабораторной
// работы
int main() {
    size_t n = 5;

    base_type arr_q[] = {1, 2, 3, 4, 5};
    SequentialLinearList *q = create_list(n);
    init_list(q, arr_q, n);

    base_type arr_r[] = {6, 7, 8, 9, 10};
    SequentialLinearList *r = create_list(n);
    init_list(r, arr_r, n);

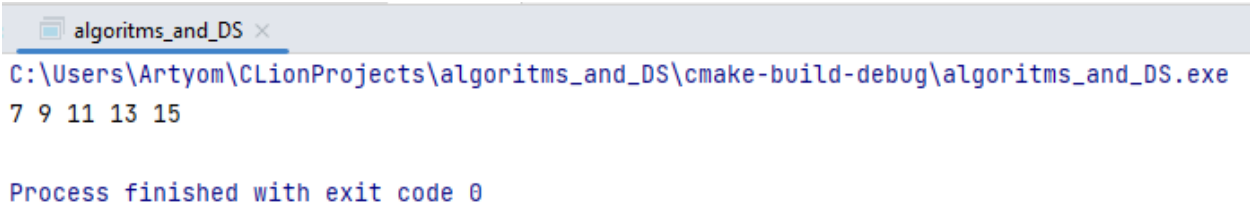
    SequentialLinearList *p = sum_polynomials(q, r);

    output_list(p);

    delete_list(q);
    delete_list(r);
    delete_list(p);

    return 0;
}
```

Результат работы программы:



```
algoritms_and_DS x
C:\Users\Artyom\CLionProjects\algoritms_and_DS\cmake-build-debug\algoritms_and_DS.exe
7 9 11 13 15

Process finished with exit code 0
```

Вывод: в результате выполнения лабораторной работы изучены СД типа «линейный список», написана их программная реализация.

Ссылка на GitHub (lab5): https://github.com/SStaryi/algorithms_and_DS

Полный листинг программы:

Файл `data_structure/sequential_linear_list.c`:

```
//  
// Created by Artyom on 31.10.2023.  
//  
  
#include "sequential_linear_list.h"  
  
const short list_ok = 0;  
const short list_not_mem = 1; // Ошибка выделения памяти  
const short list_under = 2;  
const short list_end = 3;  
short list_error = -1;  
  
SequentialLinearList *create_list(size_t initial_capacity) {  
    SequentialLinearList *list = (SequentialLinearList *)  
    malloc(sizeof(SequentialLinearList));  
  
    if (list == NULL) {  
        // Ошибка выделения памяти для списка  
        list_error = list_not_mem;  
  
        return NULL;  
    }  
  
    list->array = (base_type *) malloc(initial_capacity *  
    sizeof(base_type));  
  
    if (list->array == NULL) {  
        // Ошибка выделения памяти для массива  
        free(list);  
        list_error = list_not_mem;  
  
        return NULL;  
    }  
  
    list->size = 0;  
    list->capacity = initial_capacity;  
    list->ptr = 0; // Устанавливаем указатель на начало списка  
    list_error = list_ok;  
  
    return list;  
}  
  
void delete_list(SequentialLinearList *list) {  
    free(list->array);  
    free(list);  
}
```

```

void insert(SequentialLinearList *list, base_type element,
size_t position) {
    if (position > list->size) {
        // Позиция вставки выходит за пределы списка
        list_error = list_under;

        return;
    }

    if (list->size == list->capacity) {
        list->capacity *= 2;
        base_type *newArray = (base_type *) realloc(list->array,
list->capacity *
sizeof(base_type));

        if (newArray == NULL) {
            // Ошибка выделения памяти для расширения списка
            list_error = list_not_mem;

            return;
        }

        list->array = newArray;
    }

    for (size_t i = list->size; i > position; --i)
        list->array[i] = list->array[i - 1];

    list->array[position] = element;
    ++list->size;

    if (position <= list->ptr)
        ++list->ptr; // Обновляем указатель, если вставка была
перед ним

    list_error = list_ok;
}

```

```

void erase(SequentialLinearList *list, size_t position) {
    if (position >= list->size) {
        // Позиция удаления выходит за пределы списка
        list_error = list_under;

        return;
    }

    for (size_t i = position; i < list->size - 1; ++i)
        list->array[i] = list->array[i + 1];

    --list->size;

    if (position < list->ptr)
        --list->ptr; // Обновляем указатель, если удаление было
    перед ним

    list_error = list_ok;
}

void init_list(SequentialLinearList *list, base_type *array,
size_t size) {
    // Удаляем все текущие элементы из списка
    while (list->size > 0) {
        erase(list, 0);

        if (list_error != list_ok) {
            list_error = list_under;

            return;
        }
    }

    // Добавляем элементы из массива в список
    for (size_t i = 0; i < size; i++) {
        insert(list, array[i], i);

        if (list_error != list_ok) {
            list_error = list_not_mem;

            return;
        }
    }

    list_error = list_ok;
}

```

```

base_type get(SequentialLinearList *list, size_t position) {
    if (position >= list->size) {
        // Позиция получения выходит за пределы списка
        list_error = list_under;

        return list_error;
    }

    list_error = list_ok;

    return list->array[position];
}

void move_ptr(SequentialLinearList *list, base_type n) {
    base_type new_position = (base_type) list->ptr + n; //
    Вычисляем новую позицию указателя

    if (new_position < 0 || new_position >= (int) list->size) {
        // Новая позиция указателя выходит за пределы списка
        list_error = list_under;

        return;
    }

    list->ptr = (size_t) new_position;
    list_error = list_ok;
}

void move_to(SequentialLinearList *list, size_t position) {
    if (position >= list->size) {
        // Позиция перемещения указателя выходит за пределы
        списка
        list_error = list_under;

        return;
    }

    list->ptr = position;
    list_error = list_ok;
}

size_t count(SequentialLinearList *list) {
    return list->size;
}

bool full_list(SequentialLinearList *list) {
    return list->size == list->capacity ? 1 : 0;
}

bool end_list(SequentialLinearList *list) {
    return list->ptr == list->size - 1 ? 1 : 0;
}

```



```

void copy_list(SequentialLinearList *dest, SequentialLinearList
*src) {
    if (dest->capacity < src->size) {
        base_type *new_array = (base_type *) realloc(dest-
>array, src->size *
sizeof(base_type));

        if (new_array == NULL) {
            // Ошибка выделения памяти для копирования списка
            list_error = list_not_mem;

            return;
        }

        dest->array = new_array;
        dest->capacity = src->size;
    }

    for (size_t i = 0; i < src->size; ++i)
        dest->array[i] = src->array[i];

    dest->size = src->size;
    dest->ptr = src->ptr;
    list_error = list_ok;
}

```

```

void input_list(SequentialLinearList *list) {
    if (list == NULL) {
        // Список не существует
        list_error = list_under;

        return;
    }

    size_t size = list->size;

    base_type *array = (base_type *) malloc(size *
sizeof(base_type));

    if (array == NULL) {
        // Ошибка выделения памяти
        list_error = list_not_mem;

        return;
    }

    // Ввод элементов
    for (size_t i = 0; i < size; i++)
        scanf("%lld", &array[i]);

    init_list(list, array, size);

    if (list_error != list_ok) {
        free(array);

        return;
    }

    free(array);
    list_error = list_ok;
}

```

```

void output_list(SequentialLinearList *list) {
    if (list == NULL) {
        // Список не существует
        list_error = list_under;

        return;
    }

    if (list->size == 0) {
        // Список пуст
        list_error = list_under;

        return;
    }

    // Элементы списка
    for (size_t i = 0; i < list->size; i++)
        printf("%lld ", list->array[i]);

    printf("\n");

    list_error = list_ok;
}

```

Файл `data_structure/sequential_linear_list.h`:

```

// sequential_linear_list.h
#ifndef SEQUENTIAL_LINEAR_LIST_H
#define SEQUENTIAL_LINEAR_LIST_H

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <malloc.h>

typedef long long base_type; // Замените на нужный тип данных

typedef struct SequentialLinearList {
    base_type *array; // Массив для хранения элементов списка
    size_t size; // Текущий размер списка
    size_t capacity; // Текущая вместимость списка
    size_t ptr; // Указатель на текущий элемент
} SequentialLinearList;

extern const short list_ok;
extern const short list_not_mem; // Ошибка выделения памяти
extern const short list_under;
extern const short list_end;
extern short list_error;

// Создает новый список с заданной начальной вместимостью.
SequentialLinearList *create_list(size_t initial_capacity);

```

```

// Удаляет список и освобождает выделенную под него память.
void delete_list(SequentialLinearList *list);

// Вставляет элемент в указанную позицию.
void insert(SequentialLinearList *list, base_type element,
size_t position);

// Удаляет элемент из указанной позиции.
void erase(SequentialLinearList *list, size_t position);

// Функция для инициализации списка значениями из массива
void init_list(SequentialLinearList *list, base_type *array,
size_t size);

// Возвращает элемент из указанной позиции.
base_type get(SequentialLinearList *list, size_t position);

// Перемещает указатель на n позиций (может быть отрицательным).
void move_ptr(SequentialLinearList *list, base_type n);

// Перемещает указатель на указанную позицию.
void move_to(SequentialLinearList *list, size_t position);

// Возвращает количество элементов в списке.
size_t count(SequentialLinearList *list);

// Проверяет, является ли список полным.
bool full_list(SequentialLinearList *list);

// Проверяет, является ли текущий элемент последним в списке.
bool end_list(SequentialLinearList *list);

// Копирует все элементы из списка src в список dest
void copy_list(SequentialLinearList *dest, SequentialLinearList
*src);

// Заполняет список элементами, введенными пользователем
void input_list(SequentialLinearList *list);

// Выводит все элементы списка
void output_list(SequentialLinearList *list);

#endif // SEQUENTIAL_LINEAR_LIST_H

```

Файл lab5/lab5.c:

```

//
// Created by Artyom on 01.11.2023.
//

#include "lab5.h"

```

```

SequentialLinearList *sum_polynomials(SequentialLinearList *q,
SequentialLinearList *r) {
    // Создаем новый список для хранения результата.
    SequentialLinearList *p = create_list(q->capacity > r-
>capacity ? q->capacity :
                                                r->capacity);

    // Суммируем коэффициенты для каждой степени x.
    for (size_t i = 0; i < p->capacity; i++) {
        base_type q_coef = i < q->size ? get(q, i) : 0;
        base_type r_coef = i < r->size ? get(r, i) : 0;
        base_type sum_coef = q_coef + r_coef;

        // Если суммарный коэффициент не равен нулю, добавляем
его в результат.
        if (sum_coef != 0)
            insert(p, sum_coef, i);
    }

    return p;
}

```

Файл lab5/lab5.h:

```

//
// Created by Artyom on 01.11.2023.
//

#ifndef ALGORITMS_AND_DS_LAB5_H
#define ALGORITMS_AND_DS_LAB5_H

#include "../data_structure/sequential_linear_list.h"

// Суммирует два многочлена, представленных в виде списков.
SequentialLinearList *sum_polynomials(SequentialLinearList *q,
SequentialLinearList *r);

#endif //ALGORITMS_AND_DS_LAB5_H

```

Файл main.c:

```

#include "data_structure/sequential_linear_list.h"
#include "lab5/lab5.h"

typedef long long base_type; // Замените на нужный тип данных

```

// Здесь происходит запуск последней выполненной лабораторной работы

```
int main() {
    size_t n = 5;

    base_type arr_q[] = {1, 2, 3, 4, 5};
    SequentialLinearList *q = create_list(n);
    init_list(q, arr_q, n);

    base_type arr_r[] = {6, 7, 8, 9, 10};
    SequentialLinearList *r = create_list(n);
    init_list(r, arr_r, n);

    SequentialLinearList *p = sum_polynomials(q, r);

    output_list(p);

    delete_list(q);
    delete_list(r);
    delete_list(p);

    return 0;
}
```