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 № 5

по дисциплине: Алгоритмы и структуры данных

тема: «Структуры данных «линейные списки» С»

Выполнил: ст. группы ПВ-223
Игнатьев Артур Олегович

Проверил:
асс. Солонченко Роман Евгеньевич

Белгород 2023г.

Лабораторная работа №5

«Структуры данных «линейные списки» С»

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

Содержание отчета:

1. Тема лабораторной работы.
2. Цель работы.
3. Характеристика СД типа «линейный список» (п.1 задания).
4. Индивидуальное задание.
5. Текст модуля для реализации СД типа «линейный список», текст программы для отладки модуля, тестовые данные результат работы программы.
6. Текст программы для решения задачи с использованием модуля, тестовые данные, результат работы программы.

Задание к лабораторной работе :

1. Для СД типа «линейный список» определить:
   1.1. Абстрактный уровень представления СД:
      1.1.1. Характер организованности и изменчивости.
      1.1.2. Набор допустимых операций.
   1.2. Физический уровень представления СД:
      1.2.1. Схему хранения.
      1.2.2. Объем памяти, занимаемый экземпляром СД.
      1.2.3. Формат внутреннего представления СД и способ его интерпретации.
      1.2.4. Характеристику допустимых значений.
      1.2.5. Тип доступа к элементам.
   1.3. Логический уровень представления СД.

   Способ описания СД и экземпляра СД на языке программирования.

2. Реализовать СД типа «линейный список» в соответствии с вариантом индивидуального задания (см. табл.14) в виде модуля.

3. Разработать программу для решения задачи в соответствии с вари-антом индивидуального задания (см. табл.14) с использованием модуля, полученного в результате выполнения пункта 2 задания.

Выполнение заданий:

1. Для СД типа «линейный список» определить:

1.1. Абстрактный уровень представления СД:

1.1.1. Характер организованности и изменчивости.

Представляет собой линейную структуру – последовательность, где каждый элемент содержит ссылку на следующий элемент. Порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера

1.1.2. Набор допустимых операций:

1. Инициализация.
2. Включение элемента.
3. Исключение элемента.
4. Чтение текущего элемента.
5. Переход в начало списка.
6. Переход в конец списка.
7. Переход к следующему элементу.
8. Переход к i-му элементу.
9. Определение длины списка.
10. Уничтожение списка.

1.2. Физический уровень представления СД:

1.2.1. Схема хранения может быть реализована в последовательной или связной схемой хранения. Располагаться может в статической или динамической памяти.

1.2.2. Объем памяти, занимаемый экземпляром СД зависит от размера базового типа и вместимости списка. Для каждого элемента выделяется память под базовый тип, а также дополнительная память для переменных.

1.2.3. Формат внутреннего представления СД и способ его интерпретации.

Каждый элемент содержит данные и указатель (ссылку) на следующий элемент. Проход по списку начиная с первого элемента и следование по ссылкам до достижения конечного элемента.

1.2.4. Характеристику допустимых значений.

Допустимые значения определяются базовым типом, который выбран. Это означает, что в список можно добавить любое значение этого типа.

1.2.5. Тип доступа к элементам.

Элементы доступны последовательно, начиная с первого и двигаясь по списку к конечному элементу.

1.3. Логический уровень представления СД.

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. Это означает, что в список можно добавить любое значение этого типа.

Тип доступа к элементам:

Доступ к элементам является прямым и индексированным.

2. Реализовать СД типа «линейный список» в соответствии с вариантом индивидуального задания (см. табл.14) в виде модуля.

Вариант 3

Модуль: ОЛС в динамической памяти (базовый тип — pointer). Выделение памяти под информационную часть элемента ОЛС и запись в нее значения происходит при выполнении процедуры PutList. При выполнении процедуры GetList память, занимаемая элементом, освобождается. Размер информационной части элемента задается при инициализации ОЛС и сохраняется в дескрипторе.

```
#if !defined(__LIST3_H)
const    ListOk = 0;
const    ListNotMem = 1;
const    ListUnder = 2;
const    ListEnd  = 3;
typedef void* BaseType;
typedef  struct element *ptrel;
typedef  struct element {basetype data;
                         ptrel next;};
typedef  struct List {ptrel Start;
                      ptrel ptr;
                      unsigned int N;//размер списка
unsigned int size;}//размер информационной части эле-мента
short ListError;
void InitList(List *L);
void PutList(List *L, BaseType E);
void GetList(List *L, BaseType *E);
void ReadList(List *L,BaseType *E);
int FullList(List *L);
int EndList(List *L);
usigned int Count(List *L);
```

void BeginPtr(List *L);

void EndPtr(List *L);

void MovePtr(List *L);

void MoveTo(List *L, unsigned int n);

void DoneList(List *L);

void CopyList(List *L1,List *L2);

#endif

Листинг программы:

Файл linear_list.h

```
#ifndef ALGORITHMS_AND_DATA_STRUCTURES_LINEAR_LIST_H
#define ALGORITHMS_AND_DATA_STRUCTURES_LINEAR_LIST_H

#include "../../alg.h"

typedef void* BaseType;

typedef struct element *ptrel;
typedef struct element {
    BaseType data;
    ptrel next;
} Element;

typedef struct List {
    ptrel Start;
    ptrel ptr;
    unsigned int N; // размер списка
    unsigned int size; // размер информационной части элемента
    short ListError;
} List;

extern const short ListOk;
extern const short ListNotMem; // Ошибка выделения памяти
extern const short ListUnder;
extern const short ListEnd;
extern short ListError;

//Устанавливает указатели Start и ptr в NULL.
//Устанавливает размер списка (N) и размер информационной части элемента
(size) в 0.
//Инициализирует ListError значением -1.
void InitList(List *L);

//Выделяет память под новый элемент списка.
//Проверяет успешность выделения памяти.
//Записывает переданное значение в информационную часть нового элемента.
//Устанавливает указатель на следующий элемент как NULL (новый элемент стано-
вится последним).
//Если список пуст, устанавливает начало списка на новый элемент.
//Если список не пуст, добавляет новый элемент в конец списка.
//Обновляет указатель на текущий элемент.
//Увеличивает размер списка.
//Устанавливает код ошибки в ListOk.
void PutList(List *L, BaseType E);
```

```
//Проверяет, не является ли список пустым.
//Сохраняет значение текущего элемента для возвращения.
//Сохраняет указатель на текущий элемент для последующего освобождения па-
мяти.
//Перемещает указатель начала списка на следующий элемент.
//Освобождает память, занимаемую предыдущим первым элементом.
//Уменьшает размер списка.
//Устанавливает код ошибки в ListOk.
void GetList(List *L, BaseType *E);

//Проверяет, не является ли список пустым.
//Сохраняет значение текущего элемента для возвращения.
//Устанавливает код ошибки в ListOk.
//Возвращает значение текущего элемента.
void ReadList(List *L, BaseType *E);

//Функция возвращает 1, если текущий размер списка превысил максимальный раз-
мер, и 0 в противном случае.
int FullList(List *L);

//Эта функция возвращает 1, если текущий элемент является последним в списке,
и 0 в противном случае.
int EndList(List *L);

//Возвращает текущее количество элементов в списке.
unsigned int Count(List *L);

//Устанавливает указатель на текущий элемент списка в начало списка.
void BeginPtr(List *L);

//Устанавливает указатель на текущий элемент списка в NULL, текущий элемент
теперь считается последним в списке.
void EndPtr(List *L);

//Перемещает указатель на текущий элемент списка к следующему элементу, если
текущий элемент не является последним.
//Если текущий элемент уже последний, устанавливается код ошибки ListEnd.
void MovePtr(List *L);

//Перемещает указатель на текущий элемент списка к элементу с номером n (ну-
мерация начинается с 0).
//Если n выходит за пределы списка, устанавливается код ошибки ListUnder.
void MoveTo(List *L, unsigned int n);

//Освобождает память, занимаемую всеми элементами списка, а затем сбрасывает
параметры списка в исходное состояние.
void DoneList(List *L);

//Создает новый список L2 и копирует все элементы из списка L1 в L2.
//Если выделение памяти для новых элементов не удается, функция очищает L2 и
завершает свою работу.
void CopyList(List *L1, List *L2);

#endif //ALGORITHMS_AND_DATA_STRUCTURES_LINEAR_LIST_H
```

Файл linear_list.c

```
#include "linear_list.h"

const short ListOk = 0;
const short ListNotMem = 1;
const short ListUnder = 2;
const short ListEnd  = 3;
```

```
short ListError;

void InitList(List *L) {
    L->Start = NULL;        // Устанавливаем начало списка как NULL
    L->ptr = NULL;          // Устанавливаем указатель на текущий элемент как
NULL
    L->N = 0;               // Устанавливаем размер списка как 0
    L->size = 0;            // Устанавливаем размер информационной части эле-
мента как 0
    L->ListError = ListOk; // Инициализируем код ошибки значением ListOk
}

void PutList(List *L, BaseType E) {
    // Выделяем память под новый элемент списка
    ptrel newElement = (ptrel)malloc(sizeof(Element));

    if (newElement == NULL) {
        // Если выделение памяти не удалось, устанавливаем код ошибки
        L->ListError = ListNotMem;
        return;
    }

    // Записываем переданное значение в информационную часть нового элемента
    newElement->data = E;

    // Устанавливаем указатель на следующий элемент как NULL (новый элемент
становится последним)
    newElement->next = NULL;

    // Если список пуст, устанавливаем начало списка на новый элемент
    if (L->Start == NULL) {
        L->Start = newElement;
    } else {
        // Иначе добавляем новый элемент в конец списка
        L->ptr->next = newElement;
    }

    // Обновляем указатель на текущий элемент
    L->ptr = newElement;

    // Увеличиваем размер списка
    L->N++;

    // Устанавливаем код ошибки в ListOk
    L->ListError = ListOk;
}

void GetList(List *L, BaseType *E) {
    // Проверяем, не является ли список пустым
    if (L->Start == NULL) {
        L->ListError = ListUnder;
        return;
    }

    // Сохраняем значение текущего элемента для возвращения
    *E = L->Start->data;

    // Сохраняем указатель на текущий элемент для освобождения памяти
    ptrel temp = L->Start;

    // Перемещаем указатель начала списка на следующий элемент
    L->Start = L->Start->next;

    // Освобождаем память, занимаемую предыдущим первым элементом
```

```
    free(temp);

    // Уменьшаем размер списка
    L->N--;

    // Устанавливаем код ошибки в ListOk
    L->ListError = ListOk;
}

void ReadList(List *L, BaseType *E) {
    // Проверяем, не является ли список пустым
    if (L->Start == NULL) {
        L->ListError = ListUnder;
        return;
    }

    // Сохраняем значение текущего элемента для возвращения
    *E = L->Start->data;

    // Устанавливаем код ошибки в ListOk
    L->ListError = ListOk;
}

int FullList(List *L) {
    const unsigned int MAX_SIZE = 4294967295;

    // Проверяем, достиг ли список максимального размера
    // 1 - список полон
    // 0 - список не полон
    return L->N >= MAX_SIZE ? 1 : 0;
}

int EndList(List *L) {
    // Проверяем, является ли текущий элемент последним
    // 1 - Текущий элемент последний
    // 0 - Текущий элемент не последний
    return L->ptr == NULL ? 1 : 0;
}

unsigned int Count(List *L) {
    // Возвращаем текущее количество элементов в списке
    return L->N;
}

void BeginPtr(List *L) {
    // Устанавливаем указатель на текущий элемент в начало списка
    L->ptr = L->Start;

    // Устанавливаем код ошибки в ListOk
    L->ListError = ListOk;
}

void EndPtr(List *L) {
    // Устанавливаем указатель на текущий элемент в NULL (конец списка)
    L->ptr = NULL;

    // Устанавливаем код ошибки в ListOk
    L->ListError = ListOk;
}

void MovePtr(List *L) {
    // Проверяем, не является ли текущий элемент последним
    if (L->ptr != NULL) {
        // Перемещаем указатель на следующий элемент
```

```
        L->ptr = L->ptr->next;

        // Устанавливаем код ошибки в ListOk
        L->ListError = ListOk;
    } else {
        // Если текущий элемент уже последний, устанавливаем код ошибки в
ListEnd
        L->ListError = ListEnd;
    }
}

void MoveTo(List *L, unsigned int n) {
    // Проверяем, не выходит ли индекс за пределы списка
    if (n < L->N) {
        // Ищем элемент с номером n и устанавливаем указатель на текущий эле-
мент
        L->ptr = L->Start;
        for (L->ptr = L->Start; n > 0 && L->ptr != NULL; --n) {
            L->ptr = L->ptr->next;
        }

        // Устанавливаем код ошибки в ListOk
        L->ListError = ListOk;
    } else {
        // Если индекс выходит за пределы списка, устанавливаем код ошибки в
ListUnder
        L->ListError = ListUnder;
    }
}

void DoneList(List *L) {
    // Освобождаем память, занимаемую всеми элементами списка
    ptrel current = L->Start;
    ptrel next;

    while (current != NULL) {
        next = current->next;
        free(current);
        next = NULL;
        current = next;
    }

    // Сбрасываем параметры списка в исходное состояние
    L->Start = NULL;
    L->ptr = NULL;
    L->N = 0;
    L->size = 0;
    L->ListError = -1;
}

void CopyList(List *L1, List *L2) {
    // Очищаем существующий список L2, если он не пуст
    DoneList(L2);

    // Создаем новый список L2 с теми же параметрами размера элемента
    L2->size = L1->size;

    // Копируем элементы из L1 в L2
    ptrel currentL1 = L1->Start;
    ptrel previousL2 = NULL;

    while (currentL1 != NULL) {
        // Выделяем память под новый элемент списка
        ptrel newElement = (ptrel)malloc(sizeof(Element));
```

```
        if (newElement == NULL) {
            // Если выделение памяти не удалось, очищаем L2 и завершаем функ-
цию
            DoneList(L2);
            return;
        }

        // Копируем данные из текущего элемента L1 в новый элемент L2
        newElement->data = currentL1->data;
        newElement->next = NULL;

        // Если L2 пуст, устанавливаем начало списка
        if (previousL2 == NULL) {
            L2->Start = newElement;
        } else {
            // Иначе добавляем новый элемент в конец списка L2
            previousL2->next = newElement;
        }

        // Обновляем указатель на предыдущий элемент L2
        previousL2 = newElement;

        // Переходим к следующему элементу L1
        currentL1 = currentL1->next;
    }

    L2->ptr = previousL2;

    // Копируем остальные параметры
    L2->N = L1->N;
    L2->ptr = L1->ptr;

    // Устанавливаем код ошибки в ListOk
    L2->ListError = ListOk;
}
```

3. Разработать программу для решения задачи в соответствии с вариантом индивидуального задания (см. табл.14) с использованием модуля, полученного в результате выполнения пункта 2 задания.

Задача: дано натуральное число n целые числа $a_1, a_2, \dots, a_n$. Вычислить $\min|a_i - \bar{a}|$, где $\bar{a}$ среднее арифметическое чисел $a_1, \dots, a_n$.

Листинг программы:

Файл lab5.c

```
int task_linear_list() {
    unsigned int n;
    printf("Введите количество элементов: ");
    scanf("%u", &n);

    List myList;
    InitList(&myList);

    printf("Введите элементы:\n");
    for (unsigned int i = 0; i < n; ++i) {
        int* element = (int*)malloc(sizeof(int));
        if (element == NULL) {
            // Обработка ошибки выделения памяти
            return 1;
        }
        scanf("%d", element);
        PutList(&myList, (BaseType)element);
    }

    int sum = 0;
    BeginPtr(&myList);
    while (!EndList(&myList)) {
        BaseType currentElement;
        ReadList(&myList, &currentElement);
        int* currentIntElement = (int*)currentElement;
        sum += *currentIntElement;
        MovePtr(&myList);
    }
    double average = (double)sum / n;

    double minDifference = -1;
    BeginPtr(&myList);
    while (!EndList(&myList)) {
        BaseType currentElement;
        ReadList(&myList, &currentElement);
        int* currentIntElement = (int*)currentElement;
        double difference = fabs((double)*currentIntElement - average);
        if (minDifference == -1 || difference < minDifference) {
            minDifference = difference;
        }
        MovePtr(&myList);
    }

    printf("Минимальная абсолютная разница: %lf\n", minDifference);

    // Освобождаем память, занимаемую списком
    BeginPtr(&myList);
    while (!EndList(&myList)) {
        BaseType currentElement;
```

```
        ReadList(&myList, &currentElement);
        free(currentElement);
        MovePtr(&myList);
    }
    DoneList(&myList);

    return 0;
}
```

Вывод: в ходе выполнения лабораторной работы были изучины СД типа «линейный список», научился их программно реализовывать и использовать.