

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

РГЗ

по дисциплине: Алгоритмы и структуры данных
тема: «Структура данных типа «таблица»»

Выполнил: ст. группы ПВ-202
Буйвало Анастасия Андреевна

Проверил:
Синюк Василий Григорьевич

Белгород 2021 г.

«Структура данных «таблица» (Pascal/C)»

Цель работы:

Изучить СД типа «таблица», научиться их программно реализовывать и использовать.

Задание:

1. Для СД типа «таблица» 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.
 - а. Способ описания СД и экземпляра СД на языке программирования.
2. Реализовать СД типа «таблица» в соответствии с вариантом индивидуального задания в виде модуля.
3. Разработать программу для решения задачи в соответствии с вариантом индивидуального задания с использованием модуля, полученного в результате выполнения пункта 2 задания.

Выполнение работы:

Задание 1:

СД «Таблица»:

Абстрактный уровень представления СД:

1. Характер организованности – множество;
2. Изменчивость – динамическая СД;
3. Набор допустимых операций: инициализация, включение элемента, исключение элемента по ключу, чтение элемента по ключу, изменение элемента по ключу, проверка таблицы на пустоту, уничтожение таблицы.

Физический уровень представления СД:

1. Схема хранения: зависит от реализации.
2. Объем памяти, занимаемый экземпляром СД: зависит от реализации.
3. Формат внутреннего представления СД и способ его интерпретации:
В дескрипторе хранится число элементов, указатель на текущий элемент и указатель на первый элемент. Каждый элемент хранит данные и указатель на последующий элемент. Объем занимаемой памяти: $N * (4 + M) + 12$, где N – количество узлов списка, M – размер базового типа, 4 – размер указателя, 12 – размер дескриптора (Целое число + 2 указателя).
4. Характеристика допустимых значений: зависит от реализации.
5. Тип доступа к элементам: зависит от реализации.

Логический уровень представления СД:

Способ описания СД (Си):

```
typedef List Table;
```

Способ описания экземпляра СД (Си):

```
Table T
```

Задание

Написать интерпретатор языка арифметических вычислений. Язык содержит команды ввода и вывода значений переменных, команду пересылки константы или значения переменной в другую переменную, арифметические команды сложения, вычитания, умножения и деления. Команды ввода (*IN*) и вывода (*OUT*) имеют один операнд, команда пересылки (*MOV*) — два операнда, первый из которых — имя переменной, в которую пересылается второй операнд, арифметические команды (*ADD*, *SUB*, *MUL*, *DIV*) — два операнда, в первом сохраняется результат. В каждой строке программы — одна команда. Команды и операнды разделяются пробелами. Текст программы находится в текстовом файле. Значения переменных хранятся в таблице. Ключ элемента таблицы — имя переменной, информационная часть — значение переменной. Если операнда команды ввода или первого операнда арифметических команд и команды пересылки нет в таблице, то определить его значение и занести в таблицу. Если операнда команды вывода или второго операнда арифметических команд и команды пересылки нет в таблице, то выдать сообщение об ошибке.

Пример текста программы на языке арифметических вычислений:

```
IN a  
IN b  
IN c  
MOV d a  
MUL d b  
DIV c a  
SUB b c  
MUL b 3  
ADD d b  
OUT d
```

Таблица реализована на ОЛС, используется бинарный поиск.

Текст программы:

```
main.c  
  
#include <stdio.h>  
#include <string.h>  
#include <locale.h>  
#include "_TABLE7.h"  
  
#define TABLE_EL_SIZE sizeof(TableEl)  
  
#define N 7  
  
//создание массива служебных слов  
void in_service(char serv_words[N][N])
```

```

{
    char w1[N] = "IN";
    strcpy(serv_words[0], w1);
    char w2[N] = "OUT";
    strcpy(serv_words[1], w2);
    char w3[N] = "MOV";
    strcpy(serv_words[2], w3);
    char w4[N] = "ADD";
    strcpy(serv_words[3], w4);
    char w5[N] = "SUB";
    strcpy(serv_words[4], w5);
    char w6[N] = "MUL";
    strcpy(serv_words[5], w6);
    char w7[N] = "DIV";
    strcpy(serv_words[6], w7);
}

//функция ввода
void in_func(FILE *fp, Table *T)
{
    struct TableEl el;
    char c;
    fscanf(fp, "%c", &c);
    int i;
    for(i = 0; c != ' ' && c != '\n'; i++) { //считывание ключа
        el.name[i] = c;
        fscanf(fp, "%c", &c);
    }
    el.name[i] = '\0';
    printf("in variable %s: ", el.name);
    scanf_s("%d", &(el.data));
    if(!PutTable(T, &el))
        printf("не удалось включить элемент в таблицу\n");
}

//функция вывода
void out_func(FILE *fp, Table *T)
{
    char c;
    struct TableEl el;
    fscanf(fp, "%c", &c);
    int i;
    for (i = 0; c != ' ' && c != '\n'; i++) { //считывание ключа
        el.name[i] = c;
        fscanf(fp, "%c", &c);
    }
    el.name[i] = '\0';
    struct TableEl el1;
    struct TableEl* el2 = &el1;
    if (!ReadTable(T, &el2, el.name))
        printf("не удалось считать элемент с таким ключом\n");
    else
        printf("%s %d\n", el2->name, el2->data);
}

//функция пересылки
void mov_func(FILE *fp, Table *T)
{
    char c;
    struct TableEl el1;
    struct TableEl el2;
    fscanf(fp, "%c", &c);

```

```

int i;
for (i = 0; c != ' '; i++) { //считывание первого ключа
    el1.name[i] = c;
    fscanf(fp, "%c", &c);
}
el1.name[i] = '\0';

fscanf(fp, "%c", &c);
for (i = 0; c != ' ' && c != '\n'; i++) { //считывание второго ключа
    el2.name[i] = c;
    fscanf(fp, "%c", &c);
}
el2.name[i] = '\0';

struct TableEl el3;
struct TableEl* el4 = &el3;

struct TableEl el5;
struct TableEl* el6 = &el5;

if (!ReadTable(T, &el4, el1.name) || !ReadTable(T, &el6, el2.name)) //проверка на
возможность чтения из таблицы
    printf("ошибка чтения из таблицы\n");
el4->data = el6->data;
if (!WriteTable(T, el4, el4->name))
    printf("ошибка записи в таблицу");
}

//функция сложения
void add_func(FILE *fp, Table *T) {
    char c;
    struct TableEl el1;
    struct TableEl el2;
    fscanf(fp, "%c", &c);
    int i;
    for (i = 0; c != ' '; i++) { //считывание первого ключа
        el1.name[i] = c;
        fscanf(fp, "%c", &c);
    }
    el1.name[i] = '\0';

    fscanf(fp, "%c", &c);
    for (i = 0; c != ' ' && c != '\n'; i++) { //считывание второго ключа
        el2.name[i] = c;
        fscanf(fp, "%c", &c);
    }
    el2.name[i] = '\0';

    struct TableEl el3;
    struct TableEl* el4 = &el3;

    struct TableEl el5;
    struct TableEl* el6 = &el5;

    if (!ReadTable(T, &el4, el1.name) || !ReadTable(T, &el6, el2.name)) //проверка на
возможность чтения из таблицы
        printf("ошибка чтения из таблицы\n");
    el4->data += el6->data;
    if (!WriteTable(T, el4, el4->name))
        printf("ошибка записи в таблицу");
}

```

```

//функция вычитания
void sub_func(FILE *fp, Table *T)
{
    char c;
    struct TableEl el1;
    struct TableEl el2;
    fscanf(fp, "%c", &c);
    int i;
    for (i = 0; c != ' '; i++) { //считывание первого ключа
        el1.name[i] = c;
        fscanf(fp, "%c", &c);
    }
    el1.name[i] = '\0';

    fscanf(fp, "%c", &c);
    for (i = 0; c != ' ' && c != '\n'; i++) { //считывание второго ключа
        el2.name[i] = c;
        fscanf(fp, "%c", &c);
    }
    el2.name[i] = '\0';

    struct TableEl el3;
    struct TableEl* el4 = &el3;

    struct TableEl el5;
    struct TableEl* el6 = &el5;

    if (!ReadTable(T, &el4, el1.name) || !ReadTable(T, &el6, el2.name)) //проверка на
возможность чтения из таблицы
        printf("ошибка чтения из таблицы\n");
    el4->data -= el6->data;
    if (!WriteTable(T, el4, el4->name))
        printf("ошибка записи в таблицу");
}

//функция умножения
void mul_func(FILE *fp, Table *T)
{
    char c;
    struct TableEl el1;
    struct TableEl el2;
    fscanf(fp, "%c", &c);
    int i;
    for (i = 0; c != ' '; i++) { //считывание первого ключа
        el1.name[i] = c;
        fscanf(fp, "%c", &c);
    }
    el1.name[i] = '\0';

    fscanf(fp, "%c", &c);
    for (i = 0; c != ' ' && c != '\n'; i++) { //считывание второго ключа
        el2.name[i] = c;
        fscanf(fp, "%c", &c);
    }
    el2.name[i] = '\0';

    struct TableEl el3;
    struct TableEl* el4 = &el3;

    struct TableEl el5;
    struct TableEl* el6 = &el5;

```

```

        if (!ReadTable(T, &el4, el1.name) || !ReadTable(T, &el6, el2.name)) //проверка на
возможность чтения из таблицы
            printf("ошибка чтения из таблицы\n");
        el4->data *= el6->data;
        if (!WriteTable(T, el4, el4->name))
            printf("ошибка записи в таблицу");
    }

```

//функция деления

```

void div_func(FILE *fp, Table *T)
{

```

```

    char c;
    struct TableEl el1;
    struct TableEl el2;
    fscanf(fp, "%c", &c);
    int i;
    for (i = 0; c != ' '; i++) { //считывание первого ключа
        el1.name[i] = c;
        fscanf(fp, "%c", &c);
    }
    el1.name[i] = '\0';

```

```

    fscanf(fp, "%c", &c);
    for (i = 0; c != ' ' && c != '\n'; i++) { //считывание второго ключа
        el2.name[i] = c;
        fscanf(fp, "%c", &c);
    }
    el2.name[i] = '\0';

```

```

    struct TableEl el3;
    struct TableEl* el4 = &el3;

```

```

    struct TableEl el5;
    struct TableEl* el6 = &el5;

```

```

        if (!ReadTable(T, &el4, el1.name) || !ReadTable(T, &el6, el2.name)) //проверка на
возможность чтения из таблицы
            printf("ошибка чтения из таблицы\n");
        el4->data /= el6->data;
        if (!WriteTable(T, el4, el4->name))
            printf("ошибка записи в таблицу");
    }

```

//осуществляет вызов требующейся функции, согласно служебному слову

```

void distrib_func(FILE *fp, Table *T, int i)
{

```

```

    switch (i) {
        case 0: in_func(fp, T);
            break;
        case 1: out_func(fp, T);
            break;
        case 2: mov_func(fp, T);
            break;
        case 3: add_func(fp, T);
            break;
        case 4: sub_func(fp, T);
            break;
        case 5: mul_func(fp, T);
            break;
        case 6: div_func(fp, T);
            break;
    }

```



```

    }
}

int main()
{
    setlocale(LC_ALL, "RUS");

    FILE *fp;
    char fname[256];
    printf("input filename ");
    scanf("%s", &fname);
    fp = fopen(fname, "r");

    Table value;
    InitTable(&value, TABLE_EL_SIZE);

    int n; //количество действий в файле
    fscanf(fp, "%d", &n);

    char serv_words[N][N]; //массив служебных слов(действий)
    in_service(serv_words); //заполнение

    int i;
    char c;
    char word[N];
    while (n) {
        fscanf(fp, "%c", &c);
        while(c == '\n')
            fscanf(fp, "%c", &c);
        for (i = 0; c != ' '; i++) { //считывание служебного слова
            word[i] = c;
            fscanf(fp, "%c", &c);
        }
        word[i] = '\0';
        int j = 0;
        while (j < N && strcmp(word, serv_words[j])) //поиск служебного слова в
массиве сл.слов
            j++;
        if (j != N) //слово найдено
            distrib_func(fp, &value, j);
        else {
            printf("Допущена ошибка в введенных данных");
            return -1;
        }
        n--;
    }
    return 0;
}

```

_TABLE7.h

```

//
// Created by настя буйвало on 17/12/2021.
//

```

```

#ifndef AISD8__TABLE7_H
#define AISD8__TABLE7_H

```

```

#include "list6.h" // Смотреть лаб.раб. No5

#define TABLE_KEY_SIZE 256

// Константы ошибок
extern const int TableOk;
extern const int TableNotMem;
extern const int TableUnder;
// Переменная ошибки
extern int TableError;

// Тип элемента таблицы
typedef struct TableEl {
    char name[TABLE_KEY_SIZE]; //имя переменной
    int data; // значение переменной
} TableEl;

typedef List Table;
typedef char *T_Key;

typedef int (* func)(void*, void*);
/* Сравнивает ключи элементов таблицы,
адреса которых находятся в параметрах а и b.
Возвращает -1, если ключ элемента по адресу а меньше ключа
элемента по адресу b, 0 – если ключи равны и +1 – если ключ
элемента по адресу а больше ключа элемента по адресу b */

// Инициализация таблицы
void InitTable(Table *T, unsigned sizeEl);
// Предикат (пуста ли таблица). Возвращает 1, если таблица пуста, иначе - 0
int EmptyTable(Table *T);
// Включение элемента в таблицу. Возвращает 1, если элемент был включен в таблицу.
Иначе - 0
int PutTable(Table *T, BaseType E);
// Исключение элемента по ключу key из таблицы. Элемент записывается в E. Возвращает
1, если элемент с данным ключом есть в таблице. Иначе - 0
int GetTable(Table *T, BaseType *E, T_Key key);
// Чтение элемента по ключу key в переменную E (если элемент с таким ключом есть в
таблице).
// Возвращает 1, если он есть в таблице. Иначе - 0
int ReadTable(Table *T, BaseType *E, T_Key key);
// Изменение (перезапись) элемента по ключу key. Возвращает 1, если он есть в
таблице. Иначе - 0
int WriteTable(Table *T, BaseType E, T_Key key);
// Возвращает 1, если элемент с ключом key есть в таблице T, иначе - 0
int KeyInTable(Table *T, T_Key key);
// Очистка таблицы
void DoneTable(Table *T);

#endif //AISD8__TABLE7_H

_TABLE7.c

//
// Created by настя буйвало on 17/12/2021.
//

```

```

#include "_TABLE7.h"
#include "search.h"
#include <string.h>

const int TableOk = 0;
const int TableNotMem = 1;
const int TableUnder = 2;

int TableError = 0;

void InitTable(Table *T, unsigned sizeEl) {
    // Инициализация списка
    InitList(T, sizeEl);
}

int EmptyTable(Table *T) {
    // Таблица пуста если список пуст
    return EmptyList(T);
}

int PutTable(Table *T, BaseType E) {
    // Сдвигаю указатель в начало списка
    BeginPtr(T);

    // Если это был первый элемент на добавление в таблицу
    if (T->N == 0) {
        PutListToEnd(T, E);
        TableError = ListError;
        return 1;
    }

    // void* приводится к указателю на элемент таблицы
    TableEl *elToAdd = E;

    // Получение номера первого ключа >= ключа elToAdd
    int key = binarySearch(T, elToAdd->name);

    if (key == -1) {
        // Если такого ключа нет,
        // то элемент будет последним
        // Добавление элемента в конец списка
        PutListToEnd(T, E);
        TableError = ListError;
        return 1;
    } else {
        // Получение элемента по ключу с номером key
        MoveTo(T, key);
        TableEl *el;
        ReadList(T, &el);

        //printf("strcmp %s %s\n", elToAdd->name, el->name);

        if (strcmp(elToAdd->name, el->name) != 0) {
            // Элемент будет вставлен добавлен в таблицу
            // лексикографический порядок будет сохранен
            // Добавление элемента в список по текущему указателю
            PutList(T, E);
            TableError = ListError;
            return 1;
        } else {
            // Элемент не будет включен в таблицу,
            // поскольку уже есть элемент с таким же ключом

```

```

        TableError = ListError;
        return 0;
    }
}

}

int GetTable(Table *T, BaseType *E, T_Key key) {
    // Чтение из T по ключу key, запись в E
    // в keyFound сохраняется 1 или 0 (был ли найден ключ)
    int keyFound = ReadTable(T, E, key);

    // Если ключ был найден, то соответствующий элемент
    // следует удалить из списка
    if (keyFound == 1)
        GetList(T, E);

    TableError = ListError;
    return keyFound;
}

int ReadTable(Table *T, BaseType *E, T_Key key) {
    // Сдвигаю указатель в начало списка
    BeginPtr(T);

    // Получение номера первого ключа >= ключа key
    int keyN = binarySearch(T, key);

    if (keyN == -1) {
        // Элемента с ключом >= key нет в таблице
        TableError = ListError;
        return 0;
    }

    // Получение элемента по ключу с номером keyN
    MoveTo(T, keyN);
    TableEl *el;
    ReadList(T, &el);

    if (strcmp(key, el->name) != 0) {
        // Элемента с ключом >= key нет в таблице
        TableError = ListError;
        return 0;
    } else {
        // Элемент найден
        // Запись элемента с ключом key по адресу E
        ReadList(T, E);
        TableError = ListError;
        return 1;
    }
}

int WriteTable(Table *T, BaseType E, T_Key key) {
    // Сдвигаю указатель в начало списка
    BeginPtr(T);

    // Получение номера первого ключа >= ключа key
    int keyN = binarySearch(T, key);

    if (keyN == -1) {
        // Элемента с ключом >= key нет в таблице
        TableError = ListError;
    }
}

```

```

        return 0;
    }

    // Получение элемента по ключу с номером keyN
    MoveTo(T, keyN);
    TableEl *el;
    ReadList(T, &el);

    if (strcmp(key, el->name) == 0) {
        // Перезапись значения по ключу key
        T->ptr->data = E;
        TableError = ListError;
        return 1;
    } else
        return 0;
}

int KeyInTable(Table *T, T_Key key) {
    ptrel save = T->ptr;
    // Сдвигаю указатель в начало списка
    BeginPtr(T);

    // Получение номера первого ключа >= ключа key
    int keyN = binarySearch(T, key);

    if (keyN == -1) {
        // Элемента с ключом >= key нет в таблице
        TableError = ListError;
        return 0;
    }

    // Получение элемента по ключу с номером keyN
    MoveTo(T, keyN);
    TableEl *el;
    ReadList(T, &el);

    T->ptr = save;

    return strcmp(key, el->name) == 0;
}

void DoneTable(Table *T) {
    // Очистить список
    DoneList(T);
}

```

list6.h

```

//
// Created by настя буйвало on 17/12/2021.
//

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>

#if !defined(__LIST3_H)
#define __LIST3_H

```

```

// Константы ошибок
extern const short ListOk;
extern const short ListNotMem;
extern const short ListUnder;
extern const short ListEnd;
extern short ListError;

typedef void* BaseType;

typedef struct element* ptrel;

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

typedef struct List {
    ptrel start; // Указатель на первый элемент
    ptrel ptr; // Указатель на текущий элемент
    unsigned N; // Размер списка
    unsigned size; // Размер информационной части элемента
} List;

void InitList(List *L, unsigned size);
void PutListToEnd(List *L, BaseType E);
void PutList(List *L, BaseType E);
void GetList(List *L, BaseType *E);
void ReadList(List *L, BaseType *E);
int EmptyList(List *L);
int FullList(List *L);
int EndList(List *L);
unsigned 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

```

list6.c

```

//
// Created by настя буйвало on 17/12/2021.

#include "list6.h"

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

void InitList(List *L, unsigned size) {
    L->N = 0;
    L->size = size;
    L->start = NULL;
    L->ptr = NULL;
}

```

```

void writeBytes(void* data, void* place, unsigned size) {
    char* data_ptr = data;
    char* place_ptr = place;

    for (size_t i = 0; i < size; i++)
        place_ptr[i] = data_ptr[i];
}

```

```

void PutListToEnd(List *L, BaseType E) {
    ptrel *top;
    if (L->ptr != NULL)
        top = &(L->ptr);
    else
        top = &(L->start);

    while(*top)
        top = &((*top)->next);

    *top = (ptrel)calloc(1, sizeof(element));
    (*top)->data = (void *)calloc(1, L->size);
    // Запись данных
    writeBytes(E, (*top)->data, L->size);

    if (L->start == NULL)
        L->start = *top;

    // (*top)->data = E;
    (*top)->next = NULL;
    L->N++;
}

```

```

// Включение элемента в список
void PutList(List *L, BaseType E) {
    if (L->ptr == NULL)
        PutListToEnd(L, E);
    else {
        ptrel ptr = L->ptr;
        L->ptr = (ptrel)calloc(1, sizeof(element));
        L->ptr->data = (void *)calloc(1, L->size);
        // Запись данных
        writeBytes(E, L->ptr->data, L->size);
        //L->ptr->data = E;
        L->ptr->next = ptr;
        L->N++;

        if (ptr == L->start)
            L->start = L->ptr;
        else {
            ptrel prev = L->start;

            while(prev->next != ptr)
                prev = prev->next;

            prev->next = L->ptr;
        }
    }
}

```

```

// Исключение элемента из списка
void GetList(List *L, BaseType *E){
    if (L->ptr == NULL)
        EndPtr(L);
}

```

```

    *E = L->ptr->data;
    // Если ptr указывает на начало списка
    if (L->start == L->ptr){
        ptrel top = L->start;
        L->start = L->ptr->next;
        L->ptr = L->start;
        free(top);
    } else {
        ptrel *top = &(L->start);

        if (*top != L->ptr)
            while ((*top)->next != L->ptr)
                top = &(*top)->next;

        (*top)->next = L->ptr->next;
        // Элемент, подлежащий удалению
        ptrel elemToRemove = L->ptr;
        L->ptr = (*top)->next;
        // Освободить память
        free(elemToRemove);
    }
    // Декремент счетчика элементов
    L->N--;
}

// Чтение элемента из списка
void ReadList(List *L, BaseType *E) {
    // Если указывает на NULL (данных нет)
    if (L->ptr == NULL) {
        ListError = ListNotMem;
        return;
    }

    // Запись данных по указателю E
    *E = L->ptr->data;
}

// Предикат (полон ли список)
int FullList(List *L) {
    return L->N == UINT_MAX;
}

// Предикат (пуст ли список)
int EmptyList(List *L) {
    return L->N == 0;
}

// Предикат (указатель сейчас в конце списка)
int EndList(List *L) {
    return L->ptr->next == NULL;
}

// Возвращает число элементов в списке L
unsigned int Count(List *L) {
    return L->N;
}

// Устанавливает указатель на начало списка
void BeginPtr(List *L) {
    L->ptr = L->start;
}

```



```

// Устанавливает указатель на конец списка
void EndPtr(List *L) {
    if (L->start == NULL) {
        ListError = ListNotMem;
        return;
    }

    if (L->ptr == NULL)
        L->ptr = L->start;

    while (L->ptr->next != NULL)
        L->ptr = L->ptr->next;
}

// Сдвигает указатель на след. элемент
void MovePtr(List* L) {
    if (L->ptr != NULL)
        L->ptr = L->ptr->next;
}

// Сдвигает указатель на n-ый элемент
void MoveTo(List *L, unsigned int n) {
    if (L->start == NULL) {
        ListError = ListNotMem;
        return;
    } else if (L->N < n) {
        ListError = ListEnd;
        return;
    }

    L->ptr = L->start;
    while (n) {
        L->ptr = L->ptr->next;
        n--;
    }
}

// Очистка (удаление) списка
void DoneList(List *L) {
    BeginPtr(L);
    ptrel temp;

    while (L->ptr != NULL) {
        temp = L->ptr->next;
        free(L->ptr);
        L->ptr = temp;
    }
}

// Копирование списка L1 в L2
void CopyList(List *L1, List *L2) {
    // Указатель L1 в начало списка
    BeginPtr(L1);
    // Указатель L2 в конец списка
    EndPtr(L2);

    for (size_t i = 0; i < L1->N; i++) {
        PutList(L2, L1->ptr->data);
        L1->ptr = L1->ptr->next;
    }
}

```

```

search.c
//
// Created by настя буйвало on 17/12/2021.
//
#include "search.h"

// Модифицированный бинарный поиск для
// нахождения первого ключа >= k в упорядоченной
// лексикографически по возрастанию таблице T
int binarySearch(Table *T, T_Key k) {
    int i = 0;
    int j = T->N - 1;

    while (i <= j) {
        // Получение номера центрального ключа между i и j
        int m = (j + i) / 2;
        MoveTo(T, m);

        // Получение элемента по m-му ключу
        TableEl *el;
        ReadList(T, &el);
        // printf("k: %s, el->name: %s\n", k, el->name);
        int cmpRes = strcmp(k, el->name);

        if (cmpRes > 0)
            i = m + 1; // Сдвиг левой границы поиска
        else
            j = m - 1; // Сдвиг правой границы поиска
    }

    MoveTo(T, i);

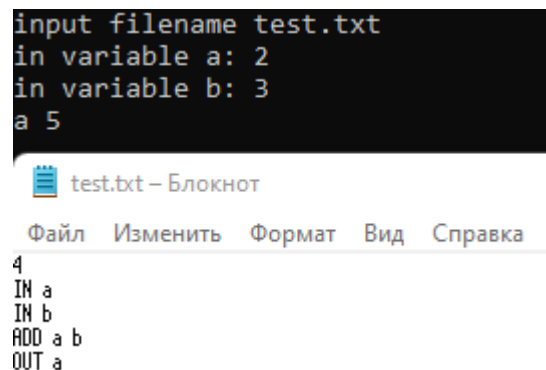
    if (T->ptr == NULL)
        return -1;

    TableEl *el;
    ReadList(T, &el);
    int cmpRes = strcmp(k, el->name);

    return cmpRes <= 0? i: -1;
}

```

Тестовые данные:



```

input filename test.txt
in variable a: 2
in variable b: 3
a 5

```

test.txt – Блокнот

Файл Изменить Формат Вид Справка

```

4
IN a
IN b
ADD a b
OUT a

```

```
input filename test.txt
in variable a: 9
in variable b: 5
a 4
b 4
```



*test.txt – Блокнот

Файл Изменить Формат Вид

```
6
IN a
IN b
SUB a b
OUT a
MOV b a
OUT b
```

```
input filename test.txt
in variable a: 2
in variable b: 3
a 6
a 2
```



test.txt – Блокнот

Файл Изменить Формат Вид

```
6
IN a
IN b
MUL a b
OUT a
DIV a b
OUT a
```