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)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

РГЗ

по дисциплине: Алгоритмы и структуры данных
тема: «Структуры данных типа «таблица» (Pascal/C)»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили: асс. Солонченко Роман
Евгеньевич

Белгород 2023г.

РГЗ

«Структуры данных типа «таблица» (Pascal/C)»

Вариант 5

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

1. Для СД типа дерево список» определить:

   1.1. Абстрактный уровень представления СД:

      1.1.1. Характер организованности и изменчивости.
      Характер организованности - **множество**. Характер изменчивости - **динамический**.

      1.1.2. Набор допустимых операций.
      Инициализация, включение элемента, исключение, чтение элемента, изменение элемента, проверка таблицы на пустоту, уничтожение таблицы.

   1.2. Физический уровень представления СД:

      1.2.1. Схему хранения.
      Схема хранения - **зависит от реализации**, может быть как последовательной, так и связной.

      1.2.2. Объем памяти, занимаемый экземпляром СД.
      Объём памяти зависит от реализации. В текущей реализации таблица является ОЛС из 5 лаб. работы, объём которой равен $V = 13+N\cdot(sizeof(BaseTy$ $4)$. $BaseType$ - элемент таблицы, который содержит ключ и значение. По умолчанию они оба int, то есть занимают по 4 байт. Итого: $V = 13+N\cdot 12$.

      1.2.3. Формат внутреннего представления СД и способ его интерпретации.
      Элемент таблицы представляет из себя СД запись, содержащую ключ и значение. Расположение элементов таблицы зависит от реализации, в данном случае таблица реализована на ОЛС, элементы которой хранятся в динамической памяти.

      1.2.4. Характеристику допустимых значений.
      $Car(C) = Car(BaseType)^0+Car(BaseType)^1+Car(BaseType)^2+...+$ $Car(BaseType)^{max}$.

      1.2.5. Тип доступа к элементам.
      Тип доступа к элементам - **последовательный**.

   1.3. Логический уровень представления СД.

      1.3.1. Способ описания СД и экземпляра СД на языке программирования.

```
Table t = InitTable();
```

2. Реализовать СД типа «таблица» в соответствии с вариантом индивидуального задания (см. табл. 18) в виде модуля.
   main.c (тесты)

```c
#include "../../libs/alg/lab8/table.c"

#include <assert.h>

void testInitTable() {
    Table t = InitTable();
    assert(TableError == TableOk);
}

void testEmptyTable() {
    Table t = InitTable();

    assert(EmptyTable(&t) && TableError == TableOk);

    PutTable(&t, (BaseType){42, 42});
    assert(!EmptyTable(&t) && TableError == TableOk);
}

void testPutTable() {
    Table t = InitTable();
    assert(PutTable(&t, (BaseType){42, 42}) && TableError == TableOk && !EmptyTable(&t) && TableError ==
    ↪  TableOk);

    BaseType an;
    assert(GetTable(&t, &an, 42) && TableError == TableOk && an.Key == 42 && an.Value == 42 );
    PutTable(&t, (BaseType){42, 42});
    assert(!PutTable(&t, (BaseType){42, 42}));

    PutTable(&t, (BaseType){39, 39});
    PutTable(&t, (BaseType){41, 41});
    PutTable(&t, (BaseType){40, 40});
    PutTable(&t, (BaseType){38, 38});
    BeginPtr(&t);
    assert(t.ptr->next->data.Key == 38 && t.ptr->next->data.Value == 38);
    MovePtr(&t);
    assert(t.ptr->next->data.Key == 39 && t.ptr->next->data.Value == 39);
    MovePtr(&t);
    assert(t.ptr->next->data.Key == 40 && t.ptr->next->data.Value == 40);
    MovePtr(&t);
    assert(t.ptr->next->data.Key == 41 && t.ptr->next->data.Value == 41);
    MovePtr(&t);
    assert(t.ptr->next->data.Key == 42 && t.ptr->next->data.Value == 42);
}

void testGetTable() {
    Table t = InitTable();
    PutTable(&t, (BaseType){42, 42});
    PutTable(&t, (BaseType){39, 39});
    PutTable(&t, (BaseType){41, 41});
    PutTable(&t, (BaseType){40, 40});
    PutTable(&t, (BaseType){38, 38});

    BaseType an;
    assert(GetTable(&t, &an, 41) && an.Key == 41 && an.Value == 41 && TableError == TableOk);
```

```
    assert(!GetTable(&t, &an, 41));
    assert(!GetTable(&t, &an, 90));
}

void testReadTable() {
    Table t = InitTable();
    PutTable(&t, (BaseType){42, 42});
    PutTable(&t, (BaseType){39, 39});
    PutTable(&t, (BaseType){41, 41});
    PutTable(&t, (BaseType){40, 40});
    PutTable(&t, (BaseType){38, 38});

    BaseType an;
    assert(ReadTable(&t, &an, 40) && TableError == TableOk && an.Key == 40 && an.Value == 40);
    assert(ReadTable(&t, &an, 40) && TableError == TableOk && an.Key == 40 && an.Value == 40);
    assert(!ReadTable(&t, &an, 90) && TableError == TableOk);
}

void testWriteTable() {
    Table t = InitTable();
    PutTable(&t, (BaseType){42, 42});
    PutTable(&t, (BaseType){39, 39});
    PutTable(&t, (BaseType){41, 41});
    PutTable(&t, (BaseType){40, 40});
    PutTable(&t, (BaseType){38, 38});

    BaseType an;
    assert(WriteTable(&t, (BaseType){40, 55}) && TableError == TableOk);
    assert(ReadTable(&t, &an, 40) && TableError == TableOk && an.Key == 40 && an.Value == 55);
    assert(!WriteTable(&t, (BaseType){123, 55}) && TableError == TableOk);
}

void test() {
    testInitTable();
    testEmptyTable();
    testPutTable();
    testGetTable();
    testReadTable();
    testWriteTable();
}

int main() {
    test();
}
```

algc.h (заголовки)

```
#ifndef TABLE
#define TABLE

#ifndef CUSTOM_TYPES_TABLE
typedef int T_Key;
typedef int T_Value;
```

```
#endif

typedef struct {
    T_Key Key;
    T_Value Value;
} TableElement;

#define CUSTOM_BASE_TYPE
typedef TableElement BaseType;

#include <lab5/singlyconnectedlist.h>

#include <stdbool.h>

#define TableOk 0
#define TableNotMem 1
#define TableUnder 2

extern int TableError;

typedef List Table;

Table InitTable();

bool EmptyTable(Table *T);

bool PutTable(Table *T, BaseType E);

bool GetTable(Table *T, BaseType *E, T_Key Key);

bool ReadTable(Table *T, BaseType *E, T_Key Key);

bool WriteTable(Table *T, BaseType E);

void DoneTable(Table *T);

#endif
```

table.c (реализации функций)

```
#include <lab8/table.h>

#include "../lab5/task2.c"

#include <stddef.h>

int TableError = TableOk;

Table InitTable() {
    TableError = TableOk;
    Table result;
    InitList(&result);
```

```
    return result;
}

bool EmptyTable(Table *T) {
    TableError = TableOk;
    bool result = (Count(T) == 0);

    if (ListError == ListNotMem) TableError = TableNotMem;
    if (ListError == ListUnder) TableError = TableUnder;

    return result;
}

bool PutTable(Table *T, BaseType E) {
    TableError = TableOk;
    BeginPtr(T);
    while (!EndList(T) && T->ptr->next->data.Key < E.Key)
        MovePtr(T);

    if (!EndList(T) && T->ptr->next->data.Key == E.Key) {
        return false;
    }

    PutList(T, E);

    if (ListError == ListNotMem) {
        TableError = TableNotMem;
        return false;
    }

    return true;
}

bool GetTable(Table *T, BaseType *E, T_Key Key) {
    TableError = TableOk;
    BeginPtr(T);
    while (!EndList(T) && T->ptr->next->data.Key < Key)
        MovePtr(T);

    if (EndList(T)) return false;

    if (T->ptr->next->data.Key == Key) {
        GetList(T, E);

        return true;
    }

    return false;
}

bool ReadTable(Table *T, BaseType *E, T_Key Key) {
    TableError = TableOk;
    BeginPtr(T);
    while (!EndList(T) && T->ptr->next->data.Key < Key)
```

```
        MovePtr(T);

    if (EndList(T)) return false;

    if (T->ptr->next->data.Key == Key) {
        ReadList(T, E);

        return true;
    }

    return false;
}

bool WriteTable(Table *T, BaseType E) {
    TableError = TableOk;
    BeginPtr(T);
    while (!EndList(T) && T->ptr->next->data.Key < E.Key)
        MovePtr(T);

    if (EndList(T)) return false;

    if (T->ptr->next->data.Key == E.Key) {
        MovePtr(T);
        T->ptr->data = E;

        return true;
    }

    return false;
}

void DoneTable(Table *T) {
    TableError = TableOk;
    DoneList(T);
}
```

3. Разработать программу для решения задачи в соответствии с вариантом индивидуального задания (см. табл.18) с использованием модуля, полученного в результате выполнения пункта 2 задания.
main.c (основная программа)

```
#include <stdint.h>

#define CUSTOM_TYPES_TABLE
typedef uint64_t T_Key;
typedef float T_Value;

#include "../../libs/alg/lab8/table.c"
#include "../../libs/alg/lab6/stack.c"

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

```c
#include <ctype.h>
#include <string.h>
#include <math.h>

#define TOKEN_BUFFER_SIZE sizeof(T_Key)

bool isStringFloat(const char *s) {
    int len;
    float ignore;
    int ret = sscanf(s, "%f %n", &ignore, &len);
    return ret == 1 && !s[len];
}

int main() {
    Stack tokens;
    InitStack(&tokens);
    Table tokenValues = InitTable();
    char formula[1024];
    int fIndex = 0;
    gets(formula);

    int input;
    char buffer[TOKEN_BUFFER_SIZE] = {0};
    int bufferCurrentIndex = 0;
    while (1) {
        input = formula[fIndex++];

        if (input == ' ' || input == '\0') {
            if (isStringFloat(buffer)) {
                float val = atof(buffer);
                PutStack(&tokens, (BaseType) {0, val});
            } else if (!strcmp(buffer, "+") || !strcmp(buffer, "-") || !strcmp(buffer, "*") || !strcmp(buffer,
            ↪  "/")) {
                BaseType leftOperand, rightOperand;
                GetStack(&tokens, &leftOperand);
                if (StackError != StackOk) {
                    fprintf(stderr, "Unable to parse");
                    return 1;
                }

                GetStack(&tokens, &rightOperand);
                if (StackError != StackOk) {
                    fprintf(stderr, "Unable to parse");
                    return 1;
                }

                if (!strcmp(buffer, "+"))
                    PutStack(&tokens, (BaseType){0, leftOperand.Value + rightOperand.Value});
                else if (!strcmp(buffer, "-"))
                    PutStack(&tokens, (BaseType){0, rightOperand.Value - leftOperand.Value});
                else if (!strcmp(buffer, "*"))
                    PutStack(&tokens, (BaseType){0, leftOperand.Value * rightOperand.Value});
                else if (!strcmp(buffer, "/"))
                    PutStack(&tokens, (BaseType){0, rightOperand.Value / leftOperand.Value});
```

```
        } else if (!strcmp(buffer, "sin") || !strcmp(buffer, "cos") || !strcmp(buffer, "arctan") ||
                   !strcmp(buffer, "abs") || !strcmp(buffer, "exp") || !strcmp(buffer, "ln") ||
                   !strcmp(buffer, "sqr") || !strcmp(buffer, "sqrt")) {
            BaseType operand;
            GetStack(&tokens, &operand);
            if (StackError != StackOk) {
                fprintf(stderr, "Unable to parse");
                return 1;
            }

            if (!strcmp(buffer, "sin"))
                PutStack(&tokens, (BaseType){0, sinf(operand.Value)});
            if (!strcmp(buffer, "cos"))
                PutStack(&tokens, (BaseType){0, cosf(operand.Value)});
            if (!strcmp(buffer, "arctan"))
                PutStack(&tokens, (BaseType){0, atanf(operand.Value)});
            if (!strcmp(buffer, "abs"))
                PutStack(&tokens, (BaseType){0, fabs(operand.Value)});
            if (!strcmp(buffer, "exp"))
                PutStack(&tokens, (BaseType){0, expf(operand.Value)});
            if (!strcmp(buffer, "ln"))
                PutStack(&tokens, (BaseType){0, log2f(operand.Value) / log2f(expf(1))});
            if (!strcmp(buffer, "sqr"))
                PutStack(&tokens, (BaseType){0, operand.Value * operand.Value});
            if (!strcmp(buffer, "sqrt"))
                PutStack(&tokens, (BaseType){0, sqrtf(operand.Value)});
        } else {
            T_Key key;
            memcpy(&key, buffer, sizeof(T_Key));

            BaseType token = (BaseType) {key, 0};
            if (!GetTable(&tokenValues, &token, key)) {
                printf("Input value of variable %s: ", buffer);
                scanf("%f", &token.Value);

                PutTable(&tokenValues, token);
            }

            PutStack(&tokens, token);
        }

        memset(buffer, 0, TOKEN_BUFFER_SIZE);
        bufferCurrentIndex = 0;

        if (input == '\0') break;
    } else {
        if (bufferCurrentIndex >= TOKEN_BUFFER_SIZE - 1) {
            fprintf(stderr, "Unable to parse, %s is too long, you can use only 7 symbols long varialbes.",
            ↪  buffer);
            return 1;
        }

        buffer[bufferCurrentIndex++] = input;
    }
```

```
    }

    BaseType result;
    GetStack(&tokens, &result);
    if (StackError != StackOk || !EmptyStack(tokens)) {
        fprintf(stderr, "Unable to parse");
        return 1;
    }

    printf("Result: %f", result.Value);

    return 0;
}
```

Ссылка на репозиторий

Вывод: в ходе лабораторной работы изучили СД типа «таблица», научились их программно реализовывать и использовать.