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 №8

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

Выполнил: студент группы ПВ-223
Мелехов Артём Дмитриевич

Проверили:
асс. Солонченко Роман Евгеньевич

Белгород 2023 г.

Лабораторная работа №7. Структуры данных типа «таблица» (C)

Цель работы: изучить СД типа «таблица», 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.

Номер модуля: 7

Задача: 7

Задание №1.

Для СД типа «таблица» определить:

Пункт 1:

Абстрактный уровень представления СД.

Решение:

Характер организованности: Таблицы обычно организованы в виде рядов и столбцов, где каждый ряд представляет собой отдельную запись, а каждый столбец представляет собой отдельное поле. Это позволяет легко и быстро получать доступ к данным и обрабатывать их.

Изменчивость: Таблицы могут быть динамическими, что означает, что они могут изменяться со временем.

Набор допустимых операций: Существует множество операций, которые можно выполнять с таблицами. Некоторые из наиболее распространенных включают:

Вставка: Добавление новой записи в таблицу.
Удаление: Удаление существующей записи из таблицы.
Изменение: Изменение значения в существующей записи.
Поиск: Поиск записи в таблице на основе определенных критериев.
Сортировка: Упорядочивание записей в таблице в определенном порядке.

Пункт 2:

Физический уровень представления СД.

Решение:

Схема хранения: Таблицы обычно хранятся в виде двумерных массивов, где каждый элемент массива представляет собой ячейку таблицы. В зависимости от типа данных, который хранится в таблице, могут использоваться различные схемы хранения.

Объем памяти, занимаемый экземпляром СД: Объем памяти, который занимает таблица, зависит от количества записей в таблице и количества полей в каждой записи. Каждое поле занимает определенное количество памяти, в зависимости от его типа данных.

Формат внутреннего представления СД и способ его интерпретации: Внутреннее представление таблицы обычно состоит из серии байтов, которые интерпретируются в соответствии с схемой хранения. Например, целочисленное значение может быть представлено в виде последовательности байтов, которые интерпретируются как целое число.

Характеристика допустимых значений: Допустимые значения в таблице зависят от типа данных каждого поля. Например, поле, предназначенное для хранения целых чисел, может принимать любое целое число в определенном диапазоне.

Тип доступа к элементам: Доступ к элементам таблицы обычно осуществляется по индексу, который представляет собой пару чисел: номер строки и номер столбца. Это позволяет быстро и эффективно получать доступ к любому элементу таблицы.

Пункт 3:

Логический уровень представления СД.

Решение:

На логическом уровне представления структуры данных типа “таблица” можно рассматривать как набор записей, каждая из которых состоит из набора полей. Каждое поле имеет имя и тип данных. Вот некоторые характеристики этого уровня:

Схема данных: Схема данных для таблицы определяет структуру каждой записи, включая имена полей и их типы данных. Схема данных обычно определяется заранее и используется для проверки целостности данных.

Отношения между данными: В таблице могут быть определены отношения между данными. Например, в таблице могут быть поля, которые ссылаются на записи в других таблицах, создавая таким образом отношения между различными наборами данных.

Операции над данными: На логическом уровне можно определить различные операции над данными, такие как выборка (извлечение подмножества данных на основе определенных критериев), проекция (извлечение подмножества полей), соединение (объединение данных из двух или более таблиц на основе общих полей) и другие.

Целостность данных: Могут быть определены ограничения целостности, которые обеспечивают корректность и согласованность данных. Например, могут быть ограничения на уникальность значений в определенных полях, ограничения на допустимые значения и т.д.

Задание №2.

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

Решение:

Файл data_structures/table.h:

```
#ifndef __TABLE_H
#define __TABLE_H

#include "tree.h"
#include "singly_linked_list.h"

#define TABLE_OK 0
#define TABLE_NOT_MEM 1
#define TABLE_UNDER 2

typedef Node* Table;
typedef int T_Key; // Определить тип ключа
typedef int (*func)(void *, void *);

int table_error; // 0..2

void init_table(Table *t, unsigned size_mem, unsigned size_el);

inline int empty_table(Table *t);

int put_table(Table *t, void *e, func f);

int get_table(Table *t, void *e, T_Key key, func f);

int read_table(Table *t, void *e, T_Key key, func f);

int write_table(Table *t, void *e, T_Key key, func f);

void done_table(Table *t);

#endif
```

Файл data_structures/table.c:

```c
#include "table.h"

void init_table(Table *t, unsigned size_mem, unsigned size_el) {
    // Инициализация таблицы
    t = (Table *) malloc(size_mem * sizeof(Table));

    if (t == NULL) {
        table_error = TABLE_NOT_MEM;

        return;
    }

    // Инициализация каждого элемента в таблице
    for (unsigned i = 0; i < size_mem; i++) {
        t[i] = (Node *) malloc(size_el * sizeof(Node));

        if (t[i] == NULL) {
            table_error = TABLE_NOT_MEM;

            return;
        }
    }

    table_error = TABLE_OK;
}

int empty_table(Table *t) {
    // Проверка, является ли указатель на таблицу NULL
    if (t == NULL)
        return 1;

    // Проверка, является ли корень дерева NULL
    if (*t == NULL)
        return 1;

    // В противном случае таблица не пуста
    return 0;
}
```

```c
int put_table(Table *t, void *e, func f) {
    // Проверка, является ли указатель на таблицу NULL
    if (t == NULL)
        return 1;

    // Создание нового узла с данными e
    Node *new_node = create_node((base_type) e);

    // Проверка, был ли узел успешно создан
    if (new_node == NULL)
        return 1;

    // Если таблица пуста, вставляем новый узел в качестве корня
    if (*t == NULL) {
        *t = new_node;

        return 0;
    }

    // В противном случае используем функцию f для определения
позиции вставки
    Node *current = *t;

    while (current != NULL)
        if (f((void *) &current->data, e) <= 0) {
            if (current->next == NULL) {
                current->next = new_node;

                return 0;
            }

            current = current->next;
        } else {
            new_node->next = current->next;
            current->next = new_node;

            return 0;
        }

    // В случае ошибки возвращаем 1
    return 1;
}
```

```
int get_table(Table *t, void *e, T_Key key, func f) {
    // Проверка, является ли указатель на таблицу NULL
    if (t == NULL)
        return 1;

    // Поиск элемента с ключом key
    Node *current = *t;

    while (current != NULL) {
        if (!f((void *) &current->data, &key)) {
            // Если элемент найден, копируем его в e и
возвращаем 0
            *(base_type *) e = current->data;

            return 0;
        }

        current = current->next;
    }

    // Если элемент не найден, возвращаем 1
    return 1;
}

int read_table(Table *t, void *e, T_Key key, func f) {
    // Проверка, является ли указатель на таблицу NULL
    if (t == NULL)
        return 1;

    // Поиск элемента с ключом key
    Node *current = *t;

    while (current != NULL) {
        if (!f((void *) &current->data, &key)) {
            // Если элемент найден, копируем его в e и
возвращаем 0
            *(base_type *) e = current->data;

            return 0;
        }

        current = current->next;
    }

    // Если элемент не найден, возвращаем 1
    return 1;
}
```

```
int write_table(Table *t, void *e, T_Key key, func f) {
    // Проверка, является ли указатель на таблицу NULL
    if (t == NULL)
        return 1;

    // Поиск элемента с ключом key
    Node *current = *t;

    while (current != NULL) {
        if (!f((void *) &current->data, &key)) {
            // Если элемент найден, заменяем его значение на e и
возвращаем 0
            current->data = *(base_type *) e;

            return 0;
        }

        current = current->next;
    }

    // Если элемент не найден, возвращаем 1
    return 1;
}

void done_table(Table *t) {
    // Проверка, является ли указатель на таблицу NULL
    if (t == NULL)
        return;

    // Освобождение памяти, занятой каждым узлом в таблице
    Node *current = *t;

    while (current != NULL) {
        Node *next = current->next;

        free(current);
        current = next;
    }

    // Установка указателя на таблицу в NULL
    *t = NULL;
}
```

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 lab8/lab8.h:

```
//
// Created by Artyom on 16.11.2023.
//

#ifndef ALGORITMS_AND_DS_LAB8_H
#define ALGORITMS_AND_DS_LAB8_H

#include "../data_structure/table.h"

#define MAX_LINE_LENGTH 100
#define MAX_VAR_NAME_LENGTH 50

void process_line(char *line, Table *t);

#endif //ALGORITMS_AND_DS_LAB8_H
```

Файл lab8/lab8.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "lab8.h"

int strcmp_wrapper(void *a, void *b) {
    return strcmp((const char *) a, (const char *) b);
}
```

```
void process_line(char *line, Table *t) {
    char command[MAX_VAR_NAME_LENGTH];
    char var1[MAX_VAR_NAME_LENGTH];
    char var2[MAX_VAR_NAME_LENGTH];
    double value;

    if (sscanf(line, "%s %s %s", command, var1, var2) == 3) {
        if (!strcmp(command, "MOV")) {
            // Получение значения из переменной var2
            double value;

            if (get_table(t, &value, var2, strcmp_wrapper)) {
                printf("Error: variable %s not found\n", var2);

                return;
            }

            // Копирование значения в переменную var1
            if (put_table(t, &value, var1, strcmp_wrapper)) {
                printf("Mistake: failed to write value to %s
variable\n", var1);

                return;
            }
        } else if (!strcmp(command, "ADD")) {
            // Получение значений из переменных var1 и var2
            double value1, value2;

            if (get_table(t, &value1, var1, strcmp_wrapper)) {
                printf("Error: variable %s not found\n", var1);

                return;
            }

            if (get_table(t, &value2, var2, strcmp_wrapper)) {
                printf("Error: variable %s not found\n", var2);

                return;
            }

            // Сложение значений и сохранение результата в var1
            value1 += value2;

            if (put_table(t, &value1, var1, strcmp_wrapper)) {
                printf("Mistake: failed to write value to %s
variable\n", var1);

                return;
            }
        } else if (!strcmp(command, "SUB")) {
            // Получение значений из переменных var1 и var2
            double value1, value2;
```

```
            if (get_table(t, &value1, var1, strcmp_wrapper)) {
                printf("Error: variable %s not found\n", var1);

                return;
            }

            if (get_table(t, &value2, var2, strcmp_wrapper)) {
                printf("Error: variable %s not found\n", var2);

                return;
            }

            // Вычитание значения var2 из var1 и сохранение
результата в var1
            value1 -= value2;

            if (put_table(t, &value1, var1, strcmp_wrapper)) {
                printf("Mistake: failed to write value to %s
variable\n", var1);

                return;
            }
        } else if (!strcmp(command, "MUL")) {
            // Получение значений из переменных var1 и var2
            double value1, value2;

            if (get_table(t, &value1, var1, strcmp_wrapper)) {
                printf("Error: variable %s not found\n", var1);

                return;
            }

            if (get_table(t, &value2, var2, strcmp_wrapper)) {
                printf("Error: variable %s not found\n", var2);

                return;
            }

            // Умножение значения var1 на var2 и сохранение
результата в var1
            value1 *= value2;

            if (put_table(t, &value1, var1, strcmp_wrapper)) {
                printf("Mistake: failed to write value to %s
variable\n", var1);

                return;
            }
        } else if (!strcmp(command, "DIV")) {
            // Получение значений из переменных var1 и var2
            double value1, value2;

            if (get_table(t, &value1, var1, strcmp_wrapper)) {
```

```
                printf("Error: variable %s not found\n", var1);

                return;
            }

            if (get_table(t, &value2, var2, strcmp_wrapper)) {
                printf("Error: variable %s not found\n", var2);

                return;
            }

            // Проверка деления на ноль
            if (!value2) {
                printf("Error: Division by zero\n");

                return;
            }

            // Деление значения var1 на var2 и сохранение
результата в var1
            value1 /= value2;

            if (put_table(t, &value1, var1, strcmp_wrapper)) {
                printf("Mistake: failed to write value to %s
variable\n", var1);

                return;
            }
        }
    } else if (sscanf(line, "%s %s", command, var1) == 2) {
        if (!strcmp(command, "IN")) {
            printf("Enter a value for %s: ", var1);

            scanf("%lf", &value);

            // Добавить значение в таблицу
            if (put_table(t, &value, var1, strcmp_wrapper))
                printf("Mistake: failed to write value to %s
variable\n", var1);
        } else if (!strcmp(command, "OUT")) {
            // Получить значение из таблицы
            double value;

            if (get_table(t, &value, var1, strcmp_wrapper))
                printf("Error: variable %s not found\n", var1);
            else
                // Вывести значение
                printf("%s = %lf\n", var1, value);
        }
    }
}
```

Файл `main.c`:

```c
#include <stdio.h>

#include "data_structure/table.h"
#include "lab8/lab8.h"

// Здесь находится реализация последней выполненной лабораторной работы
int main() {
    Table t;
    init_table(&t, 100, sizeof(double));

    char line[MAX_LINE_LENGTH];

    while (fgets(line, sizeof(line), stdin))
        process_line(line, &t);

    done_table(&t);

    return 0;
}
```

Вывод: в ходе выполнения лабораторной работы изучена и программно реализована СД типа «таблица».

Ссылка на GitHub (lab8): https://github.com/SStaryi/algorithms_and_DS