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: «Структуры данных типа «таблица» C»

Выполнил: ст. группы ПВ-223

Игнатьев Артур Олегович

Проверил:

асс. Солонченко Роман Евгеньевич

Белгород 2023г.

**РГЗ**

**«Структуры данных типа «таблица» C»**

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

**Содержание отчета:**

1. Тема лабораторной работы.

2. Цель работы.

3. Характеристика СД типа «стек» и «очередь» (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 «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: Способ описания СД и экземпляра СД на языке программирования.

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

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.

Выполнение заданий:

1. Для СД типа «таблица» определить:

1.1. Абстрактный уровень представления СД:

1.1.1. Характер организованности и изменчивости.

Множество, динамическая структура

1.1.2. Набор допустимых операций.

* Инициализация
* Включение элемента
* Исключение элемента с заданным ключом
* Чтение элемента с заданным ключом
* Изменение элемента с заданным ключом
* Проверка пустоты
* Уничтожение.

1.2. Физический уровень представления СД:

1.2.1. Схема хранения – последовательная или связная.

1.2.2. Объем памяти, занимаемый экземпляром СД – зависит от базового типа элемента таблицы.

1.2.3. Формат внутреннего представления СД и способ его интерпретации – в динамической памяти (каждый элемент таблицы – СД типа запись из ключа и информативной части) или на массиве (статическом или динамическом)

1.2.4. Характеристику допустимых значений.CAR(БД)= CAR(BaseType)0 +CAR(BaseType)1+… +CAR(BaseType)max

1.2.5. Тип доступа к элементам: в хэш-таблице - прямой

1.3. Логический уровень представления СД.

Способ описания СД и экземпляра СД на языке программирования: Table\* T;

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

Файл hash\_table.h

#ifndef ALGORITHMS\_AND\_DATA\_STRUCTURES\_HASH\_TABLE\_H  
#define ALGORITHMS\_AND\_DATA\_STRUCTURES\_HASH\_TABLE\_H  
  
extern const short TableOk;  
extern const short TableNotMem;  
extern const short TableUnder;  
  
typedef char T\_Key; //Определить тип ключа  
  
typedef struct ElTable {  
 int flag; //flag =-1 — элемент массива был занят  
 //flag = 0 — элемент массива свободен  
 //flag = 1 — элемент массива занят  
 float E;  
 T\_Key key;  
} ElTable;  
  
typedef struct Table {  
 ElTable \*Buf;  
 unsigned n; //Количество элементов в таблице  
 unsigned SizeBuf; //Количество элементов в массиве  
 unsigned SizeEl; //Размер элемента таблицы  
} Table;  
  
extern short TableError; // 0..2  
  
Table \*InitTable(unsigned SizeBuf, unsigned SizeEl);  
  
int HashFun(Table \*T, T\_Key Key, int i);  
  
//Возвращает 1, если таблица пуста, иначе — 0  
int EmptyTable(Table \*T);  
  
//Включение элемента в таблицу. Возвращает 1, если элемент включен в таблицу, иначе — 0  
//(если в таблице уже есть элемент с заданным ключем или не хватает памяти)  
int PutTable(Table \*T, float \*E, T\_Key Key);  
  
//Исключение элемен-та. Возвращает 1, если элемент с ключем Key был в таблице, иначе — 0  
int GetTable(Table \*T, float \*E, T\_Key Key);  
  
//Чтение элемента. Возвращает 1, если элемент с ключем Key есть в таблице, иначе — 0  
int ReadTable(Table \*T, float \*E, T\_Key key);  
  
// Изменение элемен-та. Возвращает 1, если элемент с ключем Key есть в таблице, иначе — 0  
int WriteTable(Table \*T, float \*E, T\_Key key);  
  
// Уничтожение таблицы  
void DoneTable(Table \*T);  
  
void printTable(Table \*T);  
  
#endif //ALGORITHMS\_AND\_DATA\_STRUCTURES\_HASH\_TABLE\_H

Файл hash\_table.c

#include "hash\_table.h"  
#include <stdio.h>  
#include <stdlib.h>  
  
const short TableOk = 0;  
const short TableNotMem = 1;  
const short TableUnder = 2;  
  
short TableError; // 0..2  
  
//Инициализация таблицы  
Table \*InitTable(unsigned SizeBuf, unsigned SizeEl) {  
//Выделим память под таблицу  
 Table \*T = (Table \*) malloc(sizeof(Table));  
//Память под указатель на массив  
 T->Buf = (ElTable \*) malloc(SizeBuf \* sizeof(ElTable));  
  
 for (int i = 0; i < SizeBuf; i++) {  
//Все элементы массива свободны  
 T->Buf[i].flag = 0;  
 }  
//Размер таблицы - 0  
 T->n = 0;  
 T->SizeBuf = SizeBuf;  
 return T;  
}  
  
//Возвращает 1, если таблица пуста, иначе 0  
int EmptyTable(Table \*T) {  
 return (T->n == 0) ? 1 : 0;  
}  
  
int HashFun(Table \*T, T\_Key key, int i) {  
//Вычислим первую Хэш-функцию  
 int e = key;  
 int H1 = e % (T->SizeBuf);  
//Вычислим вторую Хэш-функцию  
 //Она не должна возвращать 0  
 int H2 = 1 + ((int) e % (T->SizeBuf - 1));  
 int H = (H1 + i \* H2) % T->SizeBuf;  
 return H;  
}  
  
//Включение элемента в таблицу. Возвращает 1 , если элемент включен в таблицу,  
//иначе — 0 (если в таблице уже есть элемент с заданным ключем или не хватает памяти)  
int PutTable(Table \*T, float \*E, T\_Key key) {  
//Если не хватает памяти  
 if (T->n == T->SizeBuf) {  
 TableError = TableNotMem;  
 return 0;  
 }  
  
//Пока не найдется свободная ячейка  
 for (int i = 0; i < T->SizeBuf; i++) {  
 int H = HashFun(T, key, i);  
//Если ячейка не занята, осуществляем включение  
 // Иначе, продолжаем поиск  
 if (T->Buf[H].flag != 1) {  
 T->Buf[H].E = \*E;  
//Увеличиваем счетчик  
 T->n++;  
//Позиция становится занятой  
 T->Buf[H].flag = 1;  
 T->Buf[H].key = key;  
 return 1;  
 }  
//Если такой элемент уже есть в таблице  
 //Т.е ячейка по данному ключу занята  
 if (T->Buf[H].flag == 1)  
 return 0;  
 }  
}  
  
//Возвращает 1 - если удаление успешно, иначе 0  
int GetTable(Table \*T, float \*E, T\_Key key) {  
//Если пустая таблица  
 if (EmptyTable(T)) {  
 return 0;  
 }  
  
 for (int i = 0; i < T->SizeBuf; i++) {  
 int H = HashFun(T, key, i);  
//Если такой элемент есть в таблице  
 if (T->Buf[H].flag == 1) {  
 T->Buf[H].E = 0;  
//Флаг устанавливаем в состояние "Элемент массива был занят"  
 T->Buf[H].flag = -1;  
//Уменьшаем счетчик  
 T->n--;  
 T->Buf[H].key = 0;  
 return 1;  
 }  
 }  
//Все места заняты  
 return 0;  
}  
  
  
//Чтение элемента с ключом key: 1-если успешно, иначе 0  
int ReadTable(Table \*T, float \*E, T\_Key key) {  
//Если пустая таблица  
 if (EmptyTable(T)) {  
 return 0;  
 }  
  
  
 for (int i = 0; i < T->SizeBuf; i++) {  
 int H = HashFun(T, key, i);  
 if ((T->Buf[H].flag == 1) && (T->Buf[H].key == key)) {  
 \*E = T->Buf[H].E;  
 return 1;  
 }  
 }  
  
//Ключ не нашелся, неудачный поиск  
 return 0;  
}  
  
//Изменение значения ключа key  
int WriteTable(Table \*T, float \*E, T\_Key key) {  
//Если пустая таблица  
 if (EmptyTable(T)) {  
 return 0;  
 }  
//Найдем позицию элемента через хэш функцию  
 for (int i = 0; i < T->SizeBuf; i++) {  
 int H = HashFun(T, key, i);  
 if (T->Buf[H].flag == 1) {  
 T->Buf[H].E = \*E;  
 return 1;  
 }  
 }  
//Изменить не удалось  
 return 0;  
}  
  
  
//Удалить таблицу  
void DoneTable(Table \*T) {  
//Если таблица пуста, ее все равно необходимо удалить  
 free(T->Buf);  
 free(T);  
}  
  
void printTable(Table \*T) {  
 for (int i = 0; i < T->SizeBuf; i++) {  
 if (T->Buf[i].flag == 1) {  
 printf("Key: %c, Value: %2.1f\n", T->Buf[i].key, T->Buf[i].E);  
 }  
 }  
}

3. Написать интерпретатор языка арифметических вычислений. Язык содержит команды ввода и вывода значений вещественных 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

ADD d b

OUT d

Файл main.c

#include <stdlib.h>  
#include <stdio.h>  
#include <windows.h>  
#include "../../libs/alg/labs/lab8/hash\_table.h"  
  
//Ввод (inout = 1)/вывод(inout = 2) по ключу  
int tryIN\_OUT(Table \*T, const char buffer[128], int i, int inout, int j) {  
//Проверка числа операндов  
 //Пропустим пробелы  
 while (buffer[i] == ' ') {  
 i++;  
 }  
//Считаем операнд  
 char key = buffer[i];  
 i++;  
//Если операнд, не последний символ в строке, то ошибка  
 if ((buffer[i] != '\n') && (buffer[i] != '\0')) {  
 printf("Ошибка операнда в строке%d", j);  
 return 0;  
 }  
  
 if (inout == 1) {  
//Осуществление включение операнда в таблицу  
 //Запрос ввода у пользователя  
 float E, TEMP;  
 scanf("%f", &E);  
//Был ли элемент с данным ключом ранее  
 //Если нет, то обновляем значение по ключу  
 if (!ReadTable(T, &TEMP, key))  
 PutTable(T, &E, key);  
 else {  
//Иначе обновляем  
 WriteTable(T, &E, key);  
 }  
 } else {  
 float TEMP;  
 if (ReadTable(T, &TEMP, key))  
 printf("%c = %2.1f\n", key, TEMP);  
 else {  
 printf("Неизвестный операнд из строки%d", j);  
 return 0;  
 }  
 }  
}  
  
//Реализация арифметических операций  
int tryArithmetics(Table \*T, const char buffer[128], char key1, char key2, int number) {  
 float TEMP2;  
//Проверка, есть ли второй операнд в таблице  
 if (!ReadTable(T, &TEMP2, key2)) {  
 return 0;  
 }  
  
//Если первого операнда нет, то добавить его  
 float TEMP1;  
 if (!ReadTable(T, &TEMP1, key1)) {  
 PutTable(T, 0, key1);  
 }  
  
//MUL  
 if (number == 2) {  
 float mul = TEMP1 \* TEMP2;  
 WriteTable(T, &mul, key1);  
 }  
  
//DIV  
 if (number == 3) {  
 float div = TEMP1 / TEMP2;  
 WriteTable(T, &div, key1);  
 }  
  
//SUB  
 if (number == 4) {  
 float sub = TEMP1 - TEMP2;  
 WriteTable(T, &sub, key1);  
 }  
  
//ADD  
 if (number == 5) {  
 float add = TEMP1 + TEMP2;  
 WriteTable(T, &add, key1);  
 }  
 return 1;  
}  
  
//Реализация обработка двух операндов  
//number = 1, при операции MOV  
//2, если MUL  
//3, если DIV  
//4, если SUB  
//5, если ADD  
int tryTwoOperands(Table \*T, const char buffer[128], int i, int number, int j) {  
//Пропуск пробелов  
 while (buffer[i] == ' ') {  
 i++;  
 }  
//Пробуем считать операнды  
 char key1 = buffer[i];  
 i++;  
 if (buffer[i] == ' ')  
 i++;  
 else {  
 printf("Ошибка операнда 1 в строке %d", j);  
 return 0;  
 }  
 char key2 = buffer[i];  
  
 i++;  
 if ((buffer[i] != '\n') && (buffer[i] != '\0')) {  
 printf("Ошибка операнда 2 в строке %d", j);  
 return 0;  
 }  
  
//Если команда не mov, то выполнить арифметическую команду  
 if (number != 1) {  
 int flag = tryArithmetics(T, buffer, key1, key2, number);  
 return flag;  
 }  
  
//Если оба символа буквы, то выполним операцию MOV  
 int key1temp = (int) key1;  
 int key2temp = (int) key2;  
 if (number == 1) {  
 if ((key1temp >= 97) && (key1temp <= 123) && (key2temp >= 97) && (key2temp <= 123)) {  
  
 float TEMP;  
 if (ReadTable(T, &TEMP, key2))  
 PutTable(T, &TEMP, key1);  
 else {  
 printf("Неизвестный операнд из строки %d", i);  
 return 0;  
 }  
 } else {  
 printf("Ошибка названия операнда в строке %d", i);  
 return 0;  
 }  
 }  
}  
  
//Распознавание команды  
int checkingComand(Table \*T, const char comand[128], const char buffer[128], int i, int j) {  
 if ((comand[0] == 'I') && (comand[1] == 'N') && (comand[2] == ' ')) {  
 int flag = tryIN\_OUT(T, buffer, i, 1, j);  
 return (!flag) ? -1 : 1;  
 }  
 if ((comand[0] == 'M') && (comand[1] == 'O') && (comand[2] == 'V') && (comand[3] == ' ')) {  
 int flag = tryTwoOperands(T, buffer, i, 1, j);  
 return (!flag) ? -1 : 1;  
 }  
 if ((comand[0] == 'M') && (comand[1] == 'U') && (comand[2] == 'L') && (comand[3] == ' ')) {  
 int flag = tryTwoOperands(T, buffer, i, 2, j);  
 return (!flag) ? -1 : 1;  
 }  
 if ((comand[0] == 'D') && (comand[1] == 'I') && (comand[2] == 'V') && (comand[3] == ' ')) {  
 int flag = tryTwoOperands(T, buffer, i, 3, j);  
 return (!flag) ? -1 : 1;  
 }  
 if ((comand[0] == 'S') && (comand[1] == 'U') && (comand[2] == 'B') && (comand[3] == ' ')) {  
 int flag = tryTwoOperands(T, buffer, i, 4, j);  
 return (!flag) ? -1 : 1;  
 }  
 if ((comand[0] == 'A') && (comand[1] == 'D') && (comand[2] == 'D') && (comand[3] == ' ')) {  
 int flag = tryTwoOperands(T, buffer, i, 5, j);  
 return (!flag) ? -1 : 1;  
 }  
 if ((comand[0] == 'O') && (comand[1] == 'U') && (comand[2] == 'T') && (comand[3] == ' ')) {  
 int flag = tryIN\_OUT(T, buffer, i, 2, j);  
 return (!flag) ? -1 : 1;  
 }  
//Неизвестная команда  
 return 0;  
}  
  
//Обработка очередной строки программы  
int update(Table \*T, const char buffer[128], int j) {  
//Позиция каретки  
 int i = 0;  
 char comand[128];  
  
//Пока не пробел, считываем команду  
 while ((buffer[i] != ' ') && (i < 128)) {  
 comand[i] = buffer[i];  
 i++;  
 }  
 comand[i] = buffer[i];  
 comand[++i] = '\0';  
  
//Распознавание команды  
 int flag = checkingComand(T, comand, buffer, i, j);  
 if (flag == 0) {  
 printf("Неизвестная команда в строке: %d", j);  
 return 0;  
 }  
 if (flag == -1) {  
 return 0;  
 }  
  
//Строка обработана успешно  
 return 1;  
}  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
  
 Table \*T = InitTable(113, 0);  
  
//Открытие файла, в котором содержится программа  
 FILE \*file;  
 file = fopen("TextProgramm.txt", "r");  
 char buffer[128];  
  
//Пока не конец файла  
 int i = 0;  
 while (!feof(file)) {  
 fgets(buffer, 127, file);  
 i++;  
//Обработка i - строки  
 int flag = update(T, buffer, i);  
//Выйти из программы при обнаружении ошибки  
 if (!flag)  
 break;  
 }  
  
 fclose(file);  
 puts("");  
 system("pause");  
 return 0;  
}

Тестовые данные

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Проверим вручную:

a = 2.1

b = 2.3

c = 4.5

d = 2.1

d = d\*b = 2.1\*2.3 = 4.83

c = c/a = 4.5/2.1 = 2.14

b = b – c = 2.3 – 2.14 = 0.16

d = d + b = 4.83+0.16 = 4.99

Работа программы:  
Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

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