**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**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

дисциплина: Алгоритмы и Структуры данных

тема: «**Структуры данных типа «таблица» (Pascal/С)**»

Выполнил: ст. группы ПВ-21

Донцов Александр Алексеевич

Проверил: Синюк Василий Григорьевич

Белгород 2018

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

З а д а н и е

З а д а н и е

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:

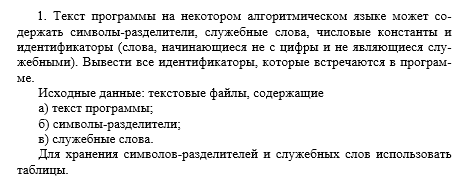
Способ описания СД и экземпляра СД на языке программирования.

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.

2. Реализовать СД типа «дерево» в соответствии с вариантом индивидуального (табл.17) задания в виде модуля.

3. Разработать программу для решения задачи в соответствии с вариантом индивидуального задания (см. табл.17) с использованием модуля, полученного в результате выполнения пункта 2 задания.





Модуль Хеш Таблицы

Заголовочный

#pragma once

#if !defined(\_\_TABLE8\_H)

#define \_\_TABLE8\_H

const short TableOk = 0;

const short TableNotMem = 1;

const short TableUnder = 2;

typedef void\* T\_Key; // Определить тип ключа

typedef void\* BaseType;

typedef struct {

int flag; /\* flag =-1 — элемент массива был занят

flag = 0 — элемент массива свободен

flag = 1 — элемент массива занят \*/

BaseType Data;

} ElTable;

typedef int(\*h\_f)(BaseType E, int index, int n);

typedef struct {

ElTable\* Buf;

unsigned n; // Количество элементов в таблице

unsigned SizeBuf; // Количество элементов в массиве

unsigned SizeEl; // Размер элемента таблицы

h\_f HashFunc;

} Table;

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

extern int TableError; // 0..2

Table\* TableInit(h\_f HashFunc, unsigned SizeBuf, unsigned SizeEl);

int TableEmpty(Table \*T); // Возвращает 1 , если таблица пуста, иначе — 0

int TablePut(Table \*T, BaseType E); // Включение элемента в таблицу. Возвращает 1 , если элемент включен в таблицу, иначе — 0 (если в таблице уже есть элемент с заданным ключем или нехватает памяти) \*/

int TableGet(Table \*T, BaseType E, func f); // Исключение элемен-та. Возвращает 1 , если элемент с ключем Key был в таблице, иначе — 0 \*/

int TableRead(Table \*T, BaseType E, func f); // Чтение элемента. Возвращает 1 , если элемент с ключем Key есть в таблице, иначе — 0 \*/

void TableDone(Table \*T); // Уничтожение таблицы

#endif

Основной

#include "HashTable.h"

#include <stdlib.h>

Table\* TableInit(h\_f HashFunc, unsigned SizeBuf, unsigned SizeEl) {

Table \*T = (Table\*)malloc(sizeof(Table));

T->Buf = (ElTable\*)malloc(sizeof(ElTable)\*SizeBuf);

for (int i = 0; i < SizeBuf; i++) {

T->Buf[i].Data = NULL;

T->Buf[i].flag = 0;

}

T->n = 0;

T->SizeBuf = SizeBuf;

T->SizeEl = SizeEl;

T->HashFunc = HashFunc;

TableError = TableOk;

return T;

}

// Возвращает 1 , если таблица пуста, иначе — 0

int TableEmpty(Table \*T) {

if (T == NULL) {

TableError = TableNotMem;

exit(TableError);

}

return (T->n) ? 1 : 0;

}

// Включение элемента в таблицу. Возвращает 1 , если элемент включен в таблицу, иначе — 0 (если в таблице уже есть элемент с заданным ключем или нехватает памяти)

int TablePut(Table \*T, BaseType E) {

if (T == NULL) {

TableError = TableNotMem;

return 0;

}

if (T->n >= T->SizeBuf) {

TableError = TableNotMem;

return 0;

}

int index = T->HashFunc(E, -1, T->SizeBuf);//Вычисление адреса

while (T->Buf[index].flag == 1) {

index = T->HashFunc(E, index, T->SizeBuf);//Перевычисление адреса

}

T->Buf[index].Data = E;

T->Buf[index].flag = 1;

T->n += 1;

return 1;

}

// Исключение элемен-та. Возвращает 1 , если элемент с ключем Key был в таблице, иначе — 0

int TableGet(Table \*T, BaseType E, func f) {

if (T == NULL) {

TableError = TableNotMem;

return 0;

}

int index = T->HashFunc(E, -1, T->SizeBuf);

while (T->Buf[index].flag != 0) {

if (!f(E, T->Buf[index].Data)) {

T->Buf[index].Data = NULL;

T->Buf[index].flag = -1;

T->n -= 1;

return 1;

}

index = T->HashFunc(E, index, T->SizeBuf);

}

return 0;

}

// Чтение элемента. Возвращает 1 , если элемент E есть в таблице, иначе — 0

int TableRead(Table \*T, BaseType E, func f) {

if (T == NULL) {

TableError = TableNotMem;

return 0;

}

int index = T->HashFunc(E, -1, T->SizeBuf);

while (T->Buf[index].flag != 0) {

if (f(E, T->Buf[index].Data))

return 1;

index = T->HashFunc(E, index, T->SizeBuf);

}

return 0;

}

// Уничтожение таблицы

void TableDone(Table \*T) {

for (unsigned i = 0; i < T->SizeBuf; i++)

free(T->Buf[i].Data);

free(T->Buf);

free(T);

}

Основной модуль

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include "hashTable.h"

#define MAXSTRLN 30

Table\* inputExceptions(char \*path, h\_f hash\_func, func equal);

void Identif(char \*path, Table \*T, func equal);

int equal(void\* a, void\* b);

int HashFunc(void\* str, int index, int n);

int TableError;

int main() {

char path1[MAXSTRLN] = "Sintaxys.txt";

char path2[MAXSTRLN] = "Programme.txt";

h\_f hash\_func = HashFunc;

func f = equal;

Table \*T = inputExceptions(path1, hash\_func, f);

Identif(path2, T, f);

getchar();

return 0;

}

//Ввод таблицы исключений

Table\* inputExceptions(char \*path, h\_f hash\_func, func equal) {

FILE \*f;

int i;

f = fopen(path, "r");

Table \*T;

T = TableInit(hash\_func, 50, 0);

while (!feof(f)) {

char \*word = (char\*)malloc(sizeof(char) \* MAXSTRLN);

i = 0;

fscanf(f, "%c", &word[i]);

while (word[i] != ' ' && word[i] != '\0' && word[i] != '\n') {

i++;

fscanf(f, "%c", &word[i]);

}

word[i] = '\0';

TablePut(T, word);

}

return T;

}

void Identif(char \*path, Table \*T, func equal) {

FILE \*f = fopen(path, "r");

int i;

char\* word;

while (!feof(f)) {

word = (char\*)malloc(sizeof(char) \* MAXSTRLN);

\*word = '\0';

i = 0;

fscanf(f, "%c", &word[i]);

while (word[i] != ' ' && word[i] != '\0' && word[i] != '\n') {

i++;

fscanf(f, "%c", &word[i]);

}

word[i] = '\0';

if (!TableRead(T, word, equal))

puts(word);

free(word);

}

printf("-------------The end-----------");

}

int equal(void\* a, void\* b) {

char\* path1 = (char\*)a;

char\* path2 = (char\*)b;

if (path1 == NULL || path2 == NULL || path1[0] == '\0')

return 0;

int i;

for (i = 0; path1[i] == path2[i] && (path1[i] != '\0' && path2[i] != '\0') && i < MAXSTRLN; i++)

;

if (path1[i] == '\0' && path2[i] == '\0')

return 1;

else

return 0;

}

int HashFunc(void\* str, int index, int n) {

char \*a = (char\*)str;

int i = 0;

while (a[i] != '\0' && a[i] != '\n')

i++;

return (i + index) % n;

}

