**МИНОБРНАУКИ РОССИИ**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

Расчетно-графическое задание

Дисциплина: Алгоритмы и структуры данных

по теме «Структура данных таблица»

Выполнил: ст. группы ВТ-22  
Макаров Даниил Сергеевич

Проверил: Синюк В.Г.

Белгород 2018

#include "lex\_checker.h"

char \*op\_names[8] ={{"CLR"},{"IN"},{"OUT"},{"MOV"},{"SUB"},{"ADD"},{"DIV"},{"MLT"}};

char\* get\_word(const char \*str, unsigned \*k){

int j=0;

int i=\*k;

char\* word=(char\*)malloc(sizeof(char)\*10);

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

i++;

}

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

word[j]=str[i];

i++;

j++;

}

\*k=i;

word[j]='\0';

return word;

}

int token\_input(FILE \*f\_in, char \*str){

if ( fgets (str , 100 , f\_in) != NULL ){

return 1;

}

return 0;

}

token\_op\* get\_operator(char\* str){

unsigned current\_char=0;

token\_op \*temp=(token\_op\*)malloc(sizeof(token\_op));

char flag=1,i=0;

char\* op\_name=get\_word(str,&current\_char);

while((i<8)&&(flag)){

if((strcmp(op\_name,op\_names[i]))==0){

temp->op\_type=i;

flag=0;

}

i++;

}

if(flag){

printf("Sintax error\n");

temp->op\_type=-1;

}

if((temp->op\_type==1)||(temp->op\_type==2)){

temp->var\_1=(token\_var\*)malloc(sizeof(token\_var));

char\* str\_1=get\_word(str,&current\_char);

if(check\_var(str\_1)){

temp->var\_1\_state=1;

temp->var\_1->name=str\_1;

}

else{

temp->var\_1\_state=2;

temp->var\_1->value=atof(str\_1);

temp->var\_1->name=NULL;

}

}

if(temp->op\_type>2){

temp->var\_1=(token\_var\*)malloc(sizeof(token\_var));

char\* str\_1=get\_word(str,&current\_char);

if(check\_var(str\_1)){

temp->var\_1\_state=1;

temp->var\_1->name=str\_1;

}

else{

temp->var\_1\_state=2;

temp->var\_1->value=atof(str\_1);

temp->var\_1->name=NULL;

}

temp->var\_2=(token\_var\*)malloc(sizeof(token\_var));

char\* str\_2=get\_word(str,&current\_char);

if(check\_var(str\_2)){

temp->var\_2\_state=1;

temp->var\_2->name=str\_2;

}

else{

temp->var\_2\_state=2;

temp->var\_2->value=atof(str\_2);

temp->var\_2->name=NULL;

}

}

return temp;

}

char check\_var(char\* str){

int i=0;

int flag=1;

while((str[i]!='\0')&&flag){

if((str[i]<='0')&&(str[i]>='9')){

flag=0;

}

i++;

}

return flag;

}

void token\_run(token\_op\* op,table\* var\_table){

switch (op->op\_type) {

case OP\_ERROR: printf("Syntax error\n");

exit(-1);

break;

case OP\_CLR: printf("Reset all\n");

table\_done(var\_table);

var\_table=table\_init(VAR\_COUNT);

break;

case OP\_IN:

if(op->var\_1\_state=1){

double temp;

printf("Input value of %s - ",op->var\_1->name);

scanf("%f",&temp);

op->var\_1->value=temp;

if(table\_element\_search(var\_table,op->var\_1->name,hash\_func1,hash\_func2)){

table\_element\_edit(var\_table,op->var\_1->name,op->var\_1,hash\_func1,hash\_func2);

}

else{

table\_put(var\_table,op->var\_1,hash\_func1,hash\_func2);

}}

break;

case OP\_OUT:

if(op->var\_1\_state=1){

if(table\_element\_search(var\_table,op->var\_1->name,hash\_func1,hash\_func2)){

op->var\_1=table\_read(var\_table,op->var\_1->name,hash\_func1,hash\_func2);

printf("%s = %f\n",op->var\_1->name,op->var\_1->value);

}

}

else{

if(op->var\_1\_state=2){

printf("%f\n",op->var\_1->value);

}

}

break;

case OP\_MOV:

token\_var \*temp1,\*temp2;

if(op->var\_2\_state=1){

if(table\_element\_search(var\_table,op->var\_2->name,hash\_func1,hash\_func2)){

temp2=table\_read(var\_table,op->var\_1->name,hash\_func1,hash\_func2);

table\_element\_edit(var\_table,op->var\_1->name,temp2,hash\_func1,hash\_func2);

}

}

else{

if(op->var\_2\_state=2){

table\_element\_edit(var\_table,op->var\_1->name,op->var\_2,hash\_func1,hash\_func2);

}

}

break;

case OP\_SUB:

token\_var \*temp1,\*temp2;

if(op->var\_2\_state=1){

if(table\_element\_search(var\_table,op->var\_2->name,hash\_func1,hash\_func2)){

temp2=table\_read(var\_table,op->var\_1->name,hash\_func1,hash\_func2);

temp1=table\_read(var\_table,op->var\_2->name,hash\_func1,hash\_func2);

temp2->value=temp1->value-temp2->value;

table\_element\_edit(var\_table,op->var\_1->name,temp2,hash\_func1,hash\_func2);

}

}

else{

if(op->var\_2\_state=2){

temp1=table\_read(var\_table,op->var\_1->name,hash\_func1,hash\_func2);

temp1->value=temp1->value-op->var\_2->value;

table\_element\_edit(var\_table,op->var\_1->name,op->var\_2,hash\_func1,hash\_func2);

}

}

break;

case OP\_ADD:

token\_var \*temp1,\*temp2;

if(op->var\_2\_state=1){

if(table\_element\_search(var\_table,op->var\_2->name,hash\_func1,hash\_func2)){

temp2=table\_read(var\_table,op->var\_1->name,hash\_func1,hash\_func2);

temp1=table\_read(var\_table,op->var\_2->name,hash\_func1,hash\_func2);

temp2->value=temp1->value+temp2->value;

table\_element\_edit(var\_table,op->var\_1->name,temp2,hash\_func1,hash\_func2);

}

}

else{

if(op->var\_2\_state=2){

temp1=table\_read(var\_table,op->var\_1->name,hash\_func1,hash\_func2);

temp1->value=temp1->value+op->var\_2->value;

table\_element\_edit(var\_table,op->var\_1->name,op->var\_2,hash\_func1,hash\_func2);

}

}

break;

case OP\_DIV:

token\_var \*temp1,\*temp2;

if(op->var\_2\_state=1){

if(table\_element\_search(var\_table,op->var\_2->name,hash\_func1,hash\_func2)){

temp2=table\_read(var\_table,op->var\_1->name,hash\_func1,hash\_func2);

temp1=table\_read(var\_table,op->var\_2->name,hash\_func1,hash\_func2);

temp2->value=temp1->value/temp2->value;

table\_element\_edit(var\_table,op->var\_1->name,temp2,hash\_func1,hash\_func2);

}

}

else{

if(op->var\_2\_state=2){

temp1=table\_read(var\_table,op->var\_1->name,hash\_func1,hash\_func2);

temp1->value=temp1->value/op->var\_2->value;

table\_element\_edit(var\_table,op->var\_1->name,op->var\_2,hash\_func1,hash\_func2);

}

}

break;

case OP\_MLT:

token\_var \*temp1,\*temp2;

if(op->var\_2\_state=1){

if(table\_element\_search(var\_table,op->var\_2->name,hash\_func1,hash\_func2)){

temp2=table\_read(var\_table,op->var\_1->name,hash\_func1,hash\_func2);

temp1=table\_read(var\_table,op->var\_2->name,hash\_func1,hash\_func2);

temp2->value=temp1->value\*temp2->value;

table\_element\_edit(var\_table,op->var\_1->name,temp2,hash\_func1,hash\_func2);

}

}

else{

if(op->var\_2\_state=2){

temp1=table\_read(var\_table,op->var\_1->name,hash\_func1,hash\_func2);

temp1->value=temp1->value\*op->var\_2->value;

table\_element\_edit(var\_table,op->var\_1->name,op->var\_2,hash\_func1,hash\_func2);

}

}

break;

default: printf("Error\n");

exit(-2);

}

}

#ifndef LEX\_CHECKER\_H

#define LEX\_CHECKER\_H

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "hash\_table.h"

#define VAR\_NAME\_SIZE 50

enum op\_types {OP\_ERROR=-1,OP\_CLR=0,OP\_IN=1,OP\_OUT=2,OP\_MOV=3,OP\_SUB=4,OP\_ADD=5,OP\_DIV=6,OP\_MLT=7};

typedef struct Operation{

char op\_type;

token\_var\* var\_1;

char var\_1\_state;//0 - не нужна для операции; 1 - в var записана переменная; 2 - в var записана числовая константа без имени

token\_var\* var\_2;

char var\_2\_state;

}token\_op;

token\_op\* get\_operator(char\* str);

token\_var\* get\_var(char\* str);

char check\_var(char\* str);

void token\_run(token\_op\* op,table\* var\_table);

int token\_input(FILE \*f\_in, char \*str);

#endif

#ifndef HASH\_TABLE\_H

#define HASH\_TABLE\_H

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define VAR\_COUNT 255

extern const char table\_ok;

extern const char table\_mem\_err;

extern const char table\_under;

typedef struct var{

char\* name;

double value;

}token\_var;

typedef struct el{

token\_var\* e;

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

flag = 0 — элемент массива свободен

flag = 1 — элемент массива занят \*/

}table\_el;

typedef struct t{

table\_el \*table\_buffer;

unsigned table\_size;

unsigned table\_current\_size;

}table;

typedef unsigned (\* hash\_func)(char\*);

unsigned hash\_func1(char\* a);

unsigned hash\_func2(char\* a);

table\* table\_init(unsigned table\_size);

int table\_empty(table \*t); // Возвращает 1 , если таблица пуста, иначе — 0

int table\_put(table \*t, token\_var \*e, hash\_func f1,hash\_func f2); // Включение элемента в таблицу.

token\_var\* table\_get(table \*t, char \*key, hash\_func f1,hash\_func f2); // Исключение элемента.

token\_var\* table\_read(table \*t, char \*key, hash\_func f1,hash\_func f2);// Чтение элемента.

int table\_element\_edit(table \*t, char \*key,token\_var \*e, hash\_func f1,hash\_func f2); // Изменение элемента таблицы

int table\_element\_search(table \*t,char \*key, hash\_func f1,hash\_func f2);

void table\_done(table \*t);

#endif

#include "hash\_table.h"

#include <math.h>

int table\_error;

const char table\_ok = 0;

const char table\_mem\_err = 1;

const char table\_under = 2;

unsigned hash\_func2(char\* a){

int i=0,c1=3,c2=5,hash\_result;

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

hash\_result+=c1\*a[i]+c2;

i++;

}

hash\_result=abs(hash\_result % VAR\_COUNT);

return hash\_result;

}

unsigned hash\_func1(char\* a){

int i=0,C=10,hash\_result;

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

hash\_result+=a[i]+C % VAR\_COUNT;

i++;

}

hash\_result=abs(hash\_result % VAR\_COUNT);

return hash\_result;

}

table\* table\_init(unsigned table\_size){

table\* temp=(table\*)malloc(sizeof(table));

if(temp!=NULL){

temp->table\_size=table\_size;

temp->table\_current\_size=0;

temp->table\_buffer=(table\_el\*)malloc(sizeof(table\_el)\*table\_size);

if(temp->table\_buffer==NULL){

table\_error=table\_mem\_err;

return;

}

table\_error=table\_ok;

return temp;

}

else{

table\_error=table\_mem\_err;

return;

}

}

int table\_empty(table \*t){

return t->table\_current\_size==0;}

int table\_put(table \*t, token\_var \*e, hash\_func f1,hash\_func f2){

int i=0;

//хеш ключа

int hash\_val=f1(e->name);

//смещение в массиве

int step\_val=f2(e->name);

while((t->table\_buffer[hash\_val].e!=NULL)&&(t->table\_buffer[hash\_val].flag!=-1)&&(i<100)){

hash\_val+=step\_val % VAR\_COUNT;

hash\_val%=VAR\_COUNT;

i++;}

if(i>=100){

printf("can't find empty space for element\n");

return 1;}

t->table\_buffer[hash\_val].e=e;

t->table\_buffer[hash\_val].flag=1;

return 0;

}

token\_var\* table\_read(table \*t, char \*key, hash\_func f1,hash\_func f2){

int i=0;

token\_var \*temp;

int hash\_val=f1(key);

int step\_val=f2(key);

while((t->table\_buffer[hash\_val].e!=NULL)&&(i<100)){

if(!strcmp(key,t->table\_buffer[hash\_val].e->name)){

printf("t[key] - %f\n",t->table\_buffer[hash\_val].e->value );

return t->table\_buffer[hash\_val].e;

}

hash\_val+=step\_val % VAR\_COUNT;

hash\_val%=VAR\_COUNT;

i++;

}

if(i>=100){

printf("can't find element\n");

return NULL;

}

printf("not match\n");

return NULL;

}

token\_var\* table\_get(table \*t, char \*key, hash\_func f1,hash\_func f2){

int i=0;

token\_var \*temp;

int hash\_val=f1(key);

int step\_val=f2(key);

while((t->table\_buffer[hash\_val].e!=NULL)&&(i<100)){

if(!strcmp(key,t->table\_buffer[hash\_val].e->name)){

temp=t->table\_buffer[hash\_val].e;

t->table\_buffer[hash\_val].e==NULL;

t->table\_buffer[hash\_val].flag=-1;

return temp;

}

hash\_val+=step\_val % VAR\_COUNT;

hash\_val%=VAR\_COUNT;

i++;

}

if(i>=100){

printf("can't find element\n");

return NULL;

}

return NULL;

}

int table\_element\_edit(table \*t, char \*key,token\_var \*e, hash\_func f1,hash\_func f2){

int i=0;

token\_var \*temp;

int hash\_val=f1(key);

int step\_val=f2(key);

while((t->table\_buffer[hash\_val].e!=NULL)&&(i<100)){

if(!strcmp(key,t->table\_buffer[hash\_val].e->name)){

t->table\_buffer[hash\_val].e=e;

return 0;

}

hash\_val+=step\_val % VAR\_COUNT;

hash\_val%=VAR\_COUNT;

i++;

}

if(i>=100){

printf("can't find element\n");

return 1;

}

return 1;

}

void table\_done(table \*t){

free(t->table\_buffer);

free(t);

}

int table\_element\_search(table \*t, char \*key,hash\_func f1,hash\_func f2){

int i=0;

token\_var \*temp;

int hash\_val=f1(key);

int step\_val=f2(key);

while((t->table\_buffer[hash\_val].e!=NULL)&&(i<100)){

if(!strcmp(key,t->table\_buffer[hash\_val].e->name)){

return 1;

}

hash\_val+=step\_val;

hash\_val%=VAR\_COUNT;

i++;

}

if(i>=100){

printf("can't find element\n");

return 0;

}

return 0;

}

#include <stdio.h>

#include <stdlib.h>

#include "lex\_checker.h"

int main(){

FILE \*f\_in;

char src\_name[50];

char token\_str[50];

table\* var\_table;

var\_table=table\_init(VAR\_COUNT);

printf("Input file path:\n");

scanf("%s",&src\_name);

f\_in=fopen(src\_name,"rw+");

if (f\_in == NULL){

return 1;

}

while(token\_input(f\_in,token\_str)){

token\_run(get\_operator(token\_str),var\_table);

}

table\_done(var\_table);

return 0;

}