LAB 4 SYMBOL TABLE

1.

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

int TableLength = 0;

#define SZ 20

struct token

{

char lexeme[SZ];

int index;

unsigned int row,col; //row number, column number.

char type[SZ];

int sz;

};

struct ListElement{

struct token tok;

struct ListElement \*next;

};

struct ListElement \*TABLE[SZ];

int row,col;

char ca,temp[20];

bool FILE\_NOT\_ENDED = true;

void print\_token(struct token s){

printf("<%s,%d,%d>",s.lexeme,s.row,s.col);

return;

}

bool is\_include(char\* temp){

if(strstr(temp,"#include")!=NULL){

return true;

}

return false;

}

bool is\_define(char \*temp){

if(strstr(temp,"#define")!=NULL){

return true;

}

return false;

}

char\* key[] = {

"auto","double","int","struct","break","else","long",

"switch","case","enum","register","typedef","char",

"extern","return","union","const","float","short",

"unsigned","continue","for","signed","void","default",

"goto","sizeof","voltile","do","if","static","while","printf","scanf","bool"

};

int isKeyword(char\* word){

// printf(" word in func %s\n",word );

for(int i = 0; i < 35; i++){

// printf(" key in func %s\n",key[i] );

if(strcmp(key[i], word) == 0) {

// printf("%s\n", "Keyworh hai ");

return 1;

}

// printf(" strcmp %d\n",strcmp(key[i], word) );

}

return 0;

}

//Following functions have been copied

bool isDelimiter(char ch)

{

if (ch == ' ' || ch == ',' || ch == ';' || ch=='='||

ch == '(' || ch == ')' || ch == '[' || ch == ']' || ch == '{' || ch == '}')

return true;

return false;

}

bool isRelational\_operator(char ch)

{

if (ch == '>' || ch == '<' || ch == '!')

return true;

return false;

}

bool isArithmetic\_operator(char ch)

{

if (ch == '%' || ch == '+' || ch == '-' || ch == '\*' ||

ch == '/' )

return true;

return false;

}

bool isRealNumber(char\* str)

{

int i, len = strlen(str);

bool hasDecimal = false;

if (len == 0)

return (false);

for (i = 0; i < len; i++) {

if (str[i] != '0' && str[i] != '1' && str[i] != '2'

&& str[i] != '3' && str[i] != '4' && str[i] != '5'

&& str[i] != '6' && str[i] != '7' && str[i] != '8'

&& str[i] != '9' && str[i] != '.' ||

(str[i] == '-' && i > 0))

return (false);

if (str[i] == '.')

hasDecimal = true;

}

return hasDecimal;

}

bool isInteger(char\* str)

{

int i, len = strlen(str);

if (len == 0)

return (false);

for (i = 0; i < len; i++) {

if (str[i] != '0' && str[i] != '1' && str[i] != '2'

&& str[i] != '3' && str[i] != '4' && str[i] != '5'

&& str[i] != '6' && str[i] != '7' && str[i] != '8'

&& str[i] != '9'|| str[i]=='.' || (str[i] == '-' && i > 0))

return (false);

}

return (true);

}

int val=-1;

int HASH(char \*str){

//Develop an OpenHash function on a string.

int sum=0;

for(int i=0;i<strlen(str);i++){

sum+=str[i];

}

return sum%SZ;

}

int SEARCH(struct token tk){

if(val<0)return 0;

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

struct ListElement \* cur = TABLE[i];

while(cur){

if(strcmp((cur->tok).lexeme,tk.lexeme)==0&&strcmp((cur->tok).type,tk.type)==0){

return 1;

}

cur=cur->next;

}

}

return 0;

}

void INSERT(struct token tk){

if(strcmp(tk.type,"func")!=0&&SEARCH(tk)==1){

return;

}

struct ListElement\* cur = malloc(sizeof(struct ListElement));

cur->tok = tk;

cur->next = NULL;

if(strcmp(tk.type,"func")==0){

if(SEARCH(tk)==0)

val++;

}

if(TABLE[val]==NULL){

TABLE[val] = cur; // No collosion.

}

else{

struct ListElement \* ele= TABLE[val];

while(ele->next!=NULL){

ele = ele->next; // Add the element at the End in the case of a collision.

}

ele->next = cur;

}

}

char buff[40];

struct token getNextToken(FILE \*fa){

char cb;

char word[20], num[20];

int i = 0;

num[0]='\0';

while(ca != EOF){

struct token s;

if(ca == '\n'){

row++;

col = 1;

printf("\n");

}

else if(ca=='#'){

int x=0;

while(ca!='\n'){

temp[x++] = ca;

ca = getc(fa);

col++;

}

temp[x] = '\0';

if(!(is\_define(temp)||is\_include(temp))){

strcpy((s.lexeme),temp);

s.row=row;

s.col=col-strlen(temp);

strcpy(s.type,"unknown");

s.sz=sizeof(temp);

col=1;

return s;

}

col=1;

}

// remove comments , blankspaces

else if(ca==' '||ca=='\t'){

ca=fgetc(fa);

while(ca==' '||ca=='\t'){

ca=fgetc(fa);

}

fseek(fa,-1,SEEK\_CUR);

//printf("space\n");

}

else if (ca=='/'){

cb = getc(fa);

if (cb == '/'){

while(ca != '\n')

ca = getc(fa);

col=0;

}

else if (cb == '\*'){

do{

while(ca != '\*')

ca = getc(fa);

ca = getc(fa);

}while (ca != '/');;

}

else{

fseek(fa, -2, SEEK\_CUR);

}

}

// check string

else if(ca == '"'){

// printf(" 5 \n");

strcpy(s.lexeme,"string literal");

s.row=row;

s.col=col;

print\_token(s);

ca = getc(fa);

while(ca != '"'){

col++;

ca = getc(fa);

}

col++;

}

// is a word -> keyword / variable

else if(isalpha(ca)) {

i=0;

while(isalpha(ca) || isdigit(ca) || ca == '\_'){

word[i++] = ca;

ca = getc(fa);

col++;

}

word[i]='\0';

fseek(fa,-1,SEEK\_CUR);

col--;

//printf(" word hai %s\n",word );

if(isKeyword(word)){

strcpy(s.lexeme,word);

strcpy(buff,word);

s.row=row;

s.col=col-(int)(strlen(word))+1;

return s;

}

else{

// printf("iddd\n");

char name[20]="";

strcat(name,"id ");

strcat(name,word);

strcpy(s.lexeme,name);

ca=fgetc(fa);

if(strcmp(buff,"func")==0)

buff[0]='\0';

if(ca=='(')

strcpy(buff,"func");

fseek(fa,-1,SEEK\_CUR);

strcpy(s.type,buff);

s.row=row;

s.col=col-(int)(strlen(word))+1;

if(strcmp(buff,"int")==0)

s.sz=sizeof(int);

else if(strcmp(buff,"char")==0)

s.sz=sizeof(char);

else if(strcmp(buff,"bool")==0)

s.sz=sizeof(bool);

else if(strcmp(buff,"func")==0)

s.sz=-1;

else

s.sz=0;

//printf("bef\n");

if(strcmp(buff,"return")==0||strcmp(buff,"if")==0||strcmp(buff,"scanf")==0||strcmp(buff,"printf")==0||strcmp(buff,"for")==0)

return s;

INSERT(s);

//printf("after\n");

//buff[0]='\0';

return s;

}

}

// is an Delimeter

else if(isDelimiter(ca)){

char c[10];

c[0]=ca;

c[1]='\0';

strcpy(s.lexeme,c);

s.row=row;

s.col=col;

col++;

return s;

}

// is a relational op

else if(isRelational\_operator(ca)){

char c[10];

c[0]=ca;

c[1]='\0';

strcpy(s.lexeme,c);

ca=getc(fa);

col++;

s.row=row;

if(ca=='=')

s.col=col-1;

else{

s.col=col;

fseek(fa,-1,SEEK\_CUR);

}

return s;

}

else if(isArithmetic\_operator(ca)){

char c[10];

c[0]=ca;

c[1]='\0';

strcpy(s.lexeme,c);

s.col=col;

s.row=row;

return s;

}

// is a number of any sort

else if(isdigit(ca)){

i=0;

num[i++] = ca;

while(isdigit(ca)|| ca == '.'){

num[i++] = ca;

ca = getc(fa);

col++;

}

num[i]='\0';

if(isRealNumber(num) || isInteger(num)){

strcpy(s.lexeme,"num");

s.row=row;

s.col=col- (int)(strlen(num))+1;

return s;

}

i = 0;

num[0]='\0';

continue;

}

//col++;

ca = getc(fa);

//end of while

}

FILE\_NOT\_ENDED = false;

struct token s;

strcpy(s.lexeme,"null");

strcpy(s.type,"null");

s.row=-1;

s.col=-1;

return s;

}

void Initialize(){

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

TABLE[i] = NULL;

}

}

void Display(){

//iterate through the linked list and display

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

struct ListElement \* cur = TABLE[i];

printf("%d %s %s\n\n",i+1,(cur->tok).lexeme,(cur->tok).type);

cur=cur->next;

while(cur){

printf("%s %s %d\n",(cur->tok).lexeme, (cur->tok).type,(cur->tok).sz);

cur=cur->next;

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

}

int main(int argc, char const \*argv[])

{

FILE \*fa=fopen("input.txt","r");

struct token s;

row=1;

col=1;

ca=fgetc(fa);

Initialize();

while(FILE\_NOT\_ENDED&&ca!=EOF){

//printf("dd\n");

s=getNextToken(fa);

ca=fgetc(fa);

print\_token(s);

//printf("dsda\n");

}

printf("\nSYMBOL TABLE\n");

Display();

fclose(fa);

return 0;

}



