CD LAB WEEK7

NAME : SAGNIK CHATTERJEE

ROLL NO: 61

REG NO :180905478

SEC :B

Q1.

CODE :

getNextToken.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#include <ctype.h>

#include <errno.h>

#define SZ 20

struct token{

char toktype[SZ];

char name[SZ];

int row,col,idx;

int sz;

};

struct ListElement{

struct token tok;

struct ListElement \*next;

};

struct ListElement \*TABLE[SZ];

int row=1,col=1,val=-1,TableLength = 0;

char prev[SZ];

bool filenotended=true;

char keyword[34][10]={"printf","scanf","auto","double","int",

"struct","break","else","long","switch","case","enum","register",

"typedef","char","extern","return","union","continue",

"for","signed","void","do","if","static","while","default","goto",

"sizeof","volatile","const","float","short","unsigned"};

bool iskeyword(char\* buf){

for(int i=0;i<34;i++){

if(strcmp(keyword[i],buf)==0)

return true;

}

return false;

}

bool isDelimiter(char ch){

if (ch == ',' || ch == ';' || ch == '(' || ch == ')' || ch == '[' || ch == ']' || ch == '{' || ch == '}')

return true;

return false;

}

bool isArithmetic\_operator(char ch)

{

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

ch == '/' )

return true;

return false;

}

void printtok(struct token t){

printf("<%s,%d,%d> ",t.name,t.row,t.col-1);

}

int SEARCH(struct token tk){

//printf("s\n");

struct ListElement \* cur;

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

cur = TABLE[i];

if(cur&&strcmp(tk.toktype,"func")==0){

if(strcmp((cur->tok).name,tk.name)==0){

return 1;

}

}

else{

while(cur){

if(strcmp((cur->tok).name,tk.name)==0&&strcmp((cur->tok).toktype,tk.toktype)==0&&(cur->tok).idx==tk.idx){

return 1;

}

cur=cur->next;

}

}

}

return 0;

}

void INSERT(struct token tk){

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

return;

}

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

cur->tok = tk;

cur->next = NULL;

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;

}

}

struct token getnextToken(FILE \*fa){

char ca,cb;

int i,j;

char buf[SZ],temp[SZ];

struct token s;

ca=fgetc(fa);

while(ca!=EOF){

//newline

if(ca=='\n'){

row++;

col=1;

//printf("\n");

}

//blank space and tabs

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

col++;//doubt

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

ca=fgetc(fa);

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

}

//comments

else if(ca=='/'){

col++;

cb=fgetc(fa);

if(cb=='/'){

while(ca!='\n')

ca=fgetc(fa);

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

}

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

do{

while(ca!='\*')

ca = fgetc(fa);

ca = fgetc(fa);

}while(ca!='/');

}

else{

i=0;

while(ca!='\n'){

temp[i++] = ca;

ca = fgetc(fa);

}

temp[i]='\0';

strcpy(s.name,"syntax error");

s.row=row;

s.col=col;

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

return s;

}

}

//preprocessor

else if(ca=='#'){

i=0;

while(ca!='\n'){

temp[i++]=ca;

ca=fgetc(fa);

}

temp[i]='\0';

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

if(strstr(temp,"#include")==NULL && strstr(temp,"#define")==NULL){//not working

printf("include\n");

strcpy(s.name,"syntax error");

s.row=row;

row++;

s.col=col;

return s;

}

}

//keywords and identifiers

else if(isalpha(ca)||ca=='\_'){

i=0;

while(isalnum(ca)||ca=='\_'){

buf[i++]=ca;

ca=fgetc(fa);

col++;

}

buf[i]='\0';

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

if(iskeyword(buf)){

strcpy(s.name,buf);

strcpy(prev,buf);

s.row=row;

s.col=col-strlen(buf)+1;

return s;

}

else{

if(ca=='('){

strcpy(s.name,buf);

strcpy(s.toktype,"func");

s.sz=-1;

if(SEARCH(s)==0){

val++;

}

s.idx = val;

INSERT(s);

return s;

}

char w[10]="";

strcat(w,"id ");

strcat(w,buf);

strcpy(s.name,w);

strcpy(s.toktype,prev);

s.row=row;

s.col=col-strlen(buf)+1;

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

s.sz=sizeof(int);

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

s.sz=sizeof(char);

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

s.sz=sizeof(bool);

else

s.sz=0;

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

return s;

s.idx=val;

INSERT(s);

return s;

}

}

//relational operator

else if(ca=='='||ca=='>'||ca=='<'||ca=='!'){

cb=fgetc(fa);

i=0;

temp[i++]=ca;

col++;

if(cb=='='){

temp[i++] = cb;

temp[i] = '\0';

strcpy(s.name,temp);

s.row=row;

s.col=col;

col++;

return s;

}

else{

temp[i]='\0';

strcpy(s.name,temp);

s.row=row;

s.col=col;

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

return s;

}

}

//string

else if(ca=='"'){

i=0;

do{

col++;

i++;

ca=fgetc(fa);

}while(ca!='"');

col++;

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

s.row=row;

s.col=col-i;

return s;

}

//delimiters

else if(isDelimiter(ca)){

i=0;

temp[i++]=ca;

temp[i]='\0';

col++;

strcpy(s.name,temp);

s.row=row;

s.col=col;

return s;

}

//numeric constants

else if(isdigit(ca)){

i=0;

while(isdigit(ca)){

col++;

i++;

ca=fgetc(fa);

}

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

strcpy(s.name,"num");

s.row=row;

s.col=col-i+1;

return s;

}

//arithmetic op

else if(isArithmetic\_operator(ca)){

i=0;

temp[i++]=ca;

temp[i]='\0';

col++;

strcpy(s.name,temp);

s.row=row;

s.col=col;

return s;

}

ca=fgetc(fa);

}

strcpy(s.name,"end");

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).name,(cur->tok).toktype);

cur=cur->next;

while(cur){

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

cur=cur->next;

}

printf("[STATUS] Done.\n");

}

}

parser.c

Code :

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "getNextToken.c"

//prototypes

void declarations();

void assign\_stat();

void assign\_stat\_prime();

void data\_type();

void identifier\_list();

void identifier\_list\_prime();

void untoken();

struct token s;

FILE \*fa;

void untoken(){

int len;

if(s.name[0]=='i'&&s.name[1]=='d'&&s.name[2]==' ')

len=strlen(s.name)-3;

else len=strlen(s.name);

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

}

void Program(){

s=getnextToken(fa);

if(strcmp(s.name,"main")==0){

s=getnextToken(fa);

if(strcmp(s.name,"(")==0){

s=getnextToken(fa);

if(strcmp(s.name,")")==0){

s=getnextToken(fa);

if(strcmp(s.name,"{")==0){

declarations();

assign\_stat();

s=getnextToken(fa);

if(strcmp(s.name,"}")==0){

return;

}

else{

printf("[ERROR] : missing '}' row : %d col :%d\n",s.row,s.col);

exit(1);

}

}

else{

printf("[ERROR] : missing '{' row : %d col :%d\n",s.row,s.col);

exit(1);

}

}

else{

printf("[ERROR]: missing ')' row : %d col :%d\n",s.row,s.col);

exit(1);

}

}

else{

printf("[ERROR]: missing '(' row : %d col :%d\n",s.row,s.col);

exit(1);

}

}

else{

printf("[ERROR]: missing main row : %d col :%d\n",s.row,s.col);

exit(1);

}

}

void declarations(){

s=getnextToken(fa);

//printf("dec %s\n",s.name);

if(strcmp(s.name,"int")==0||strcmp(s.name,"char")==0){

identifier\_list();

s=getnextToken(fa);

if(strcmp(s.name,";")==0){

declarations();

}

else{

printf("[ERROR]: expected ';' row : %d col :%d\n",s.row,s.col);

exit(1);

}

}

else{

untoken();

}

}

void identifier\_list(){

s=getnextToken(fa);

//printf("id %s\n",s.name);

if(s.name[0]=='i'&&s.name[1]=='d'&&s.name[2]==' '){

identifier\_list\_prime();

}

else{

printf("[ERROR] : expected identifier row : %d col :%d\n",s.row,s.col);

exit(1);

}

}

void identifier\_list\_prime(){

s=getnextToken(fa);

//printf("idprime %s\n",s.name);

if(strcmp(s.name,",")==0){

identifier\_list();

}

else{

if(strcmp(s.name,";")==0){

untoken();

}

else{

printf("[ERROR] : missing ',' row : %d col : %d\n",s.row,s.col);

exit(1);

}

}

//printf("error : expecting ',' in line %d\n",s.row);

}

void assign\_stat(){

s=getnextToken(fa);

// printf("as %s\n",s.name);

if(s.name[0]=='i'&&s.name[1]=='d'&&s.name[2]==' '){

s=getnextToken(fa);

if(strcmp(s.name,"=")==0)

assign\_stat\_prime();

else{

printf("[ERROR] : missing '=' row : %d col :%d\n",s.row,s.col);

exit(1);

}

}

else{

printf("[ERROR] : missing identifier row : %d col :%d\n",s.row,s.col);

exit(1);

}

}

void assign\_stat\_prime(){

s=getnextToken(fa);

// printf("aspp %s\n",s.name);

if((s.name[0]=='i'&&s.name[1]=='d'&&s.name[2]==' ')||strcmp(s.name,"num")==0){

s=getnextToken(fa);

if(strcmp(s.name,";")==0)

return;

else{

printf("[ERROR] : missing ';' row : %d col :%d\n",s.row,s.col);

exit(1);

}

}

else{

printf("[ERROR] : missing identifier or numeric constant row : %d col :%d\n",s.row,s.col);

exit(1);

}

}

void data\_type(){

s=getnextToken(fa);

if(strcmp(s.name,"int")==0||strcmp(s.name,"char")==0)

return;

else{

printf("[ERROR] : data type not available row : %d col :%d\n",s.row,s.col);

exit(1);

}

}

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

{

if(argc!=2){

printf("[ERROR] Usage : %s <filename>",argv[0]);

exit(1);

}

fa=fopen(argv[1],"r");

if(fa==NULL){

printf("[ERROR] Could not open file for reading.");

exit(1);

}

Initialize();

Program();

s=getnextToken(fa);

if(strcmp(s.name,"end")==0)

printf("[STATUS] Successfully parsed.\n");

Display();

return 0;

}

Input file :

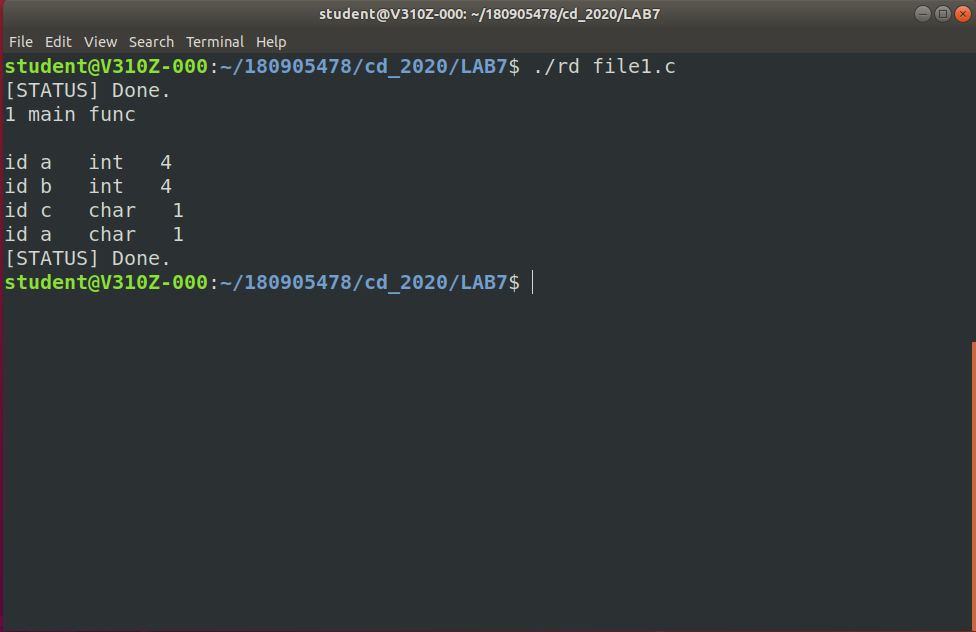
1. main(){

int a,b ;

char c;

a=25;

}



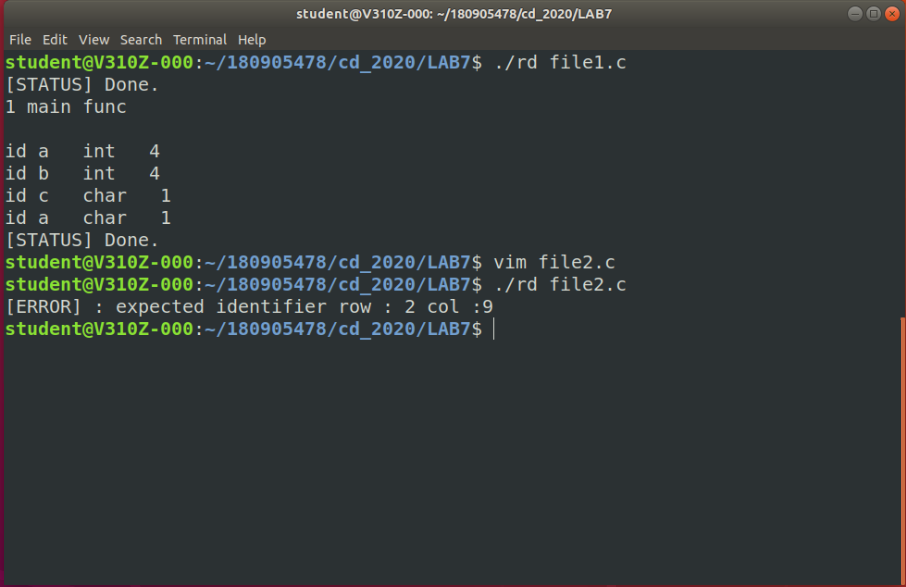
1. main(){

int a ,,b;

char c ;

a=24;

}



3.int a b ;

