CD PROGRAM NO :1

#include <stdbool.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

bool isValidDelimiter(char ch) {

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

ch == '/' || ch == ',' || ch == ';' || ch == '>' ||

ch == '<' || ch == '=' || ch == '(' || ch == ')' ||

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

return (true);

return (false);

}

bool isValidOperator(char ch){

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

ch == '/' || ch == '>' || ch == '<' ||

ch == '=')

return (true);

return (false);

}

// Returns 'true' if the string is a VALID IDENTIFIER.

bool isvalidIdentifier(char\* str){

if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||

str[0] == '3' || str[0] == '4' || str[0] == '5' ||

str[0] == '6' || str[0] == '7' || str[0] == '8' ||

str[0] == '9' || isValidDelimiter(str[0]) == true)

return (false);

return (true);

}

bool isValidKeyword(char\* str) {

if (!strcmp(str, "if") || !strcmp(str, "else") || !strcmp(str, "while") || !strcmp(str, "do") || !strcmp(str, "break") || !strcmp(str, "continue") || !strcmp(str, "int")

|| !strcmp(str, "double") || !strcmp(str, "float") || !strcmp(str, "return") || !strcmp(str, "char") || !strcmp(str, "case") || !strcmp(str, "char")

|| !strcmp(str, "sizeof") || !strcmp(str, "long") || !strcmp(str, "short") || !strcmp(str, "typedef") || !strcmp(str, "switch") || !strcmp(str, "unsigned")

|| !strcmp(str, "void") || !strcmp(str, "static") || !strcmp(str, "struct") || !strcmp(str, "goto"))

return (true);

return (false);

}

bool isValidInteger(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] == '-' && i > 0))

return (false);

}

return (true);

}

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);

}

char\* subString(char\* str, int left, int right) {

int i;

char\* subStr = (char\*)malloc( sizeof(char) \* (right - left + 2));

for (i = left; i <= right; i++)

subStr[i - left] = str[i];

subStr[right - left + 1] = '\0';

return (subStr);

}

void detectTokens(char\* str) {

int left = 0, right = 0;

int length = strlen(str);

while (right <= length && left <= right) {

if (isValidDelimiter(str[right]) == false)

right++;

if (isValidDelimiter(str[right]) == true && left == right) {

if (isValidOperator(str[right]) == true)

printf("Valid operator : '%c'\n", str[right]);

right++;

left = right;

} else if (isValidDelimiter(str[right]) == true && left != right || (right == length && left != right)) {

char\* subStr = subString(str, left, right - 1);

if (isValidKeyword(subStr) == true)

printf("Valid keyword : '%s'\n", subStr);

else if (isValidInteger(subStr) == true)

printf("Valid Integer : '%s'\n", subStr);

else if (isRealNumber(subStr) == true)

printf("Real Number : '%s'\n", subStr);

else if (isvalidIdentifier(subStr) == true

&& isValidDelimiter(str[right - 1]) == false)

printf("Valid Identifier : '%s'\n", subStr);

else if (isvalidIdentifier(subStr) == false

&& isValidDelimiter(str[right - 1]) == false)

printf("Invalid Identifier : '%s'\n", subStr);

left = right;

}

}

return;

}

int main(){

char str[100] = "l\*m+k";

printf("The Program is : '%s' \n", str);

printf("All Tokens are : \n");

detectTokens(str);

return (0);

}

CD PROGRAM NO :2

#include<stdio.h>

#include<ctype.h>

#include<string.h>

#include<stdlib.h>

#define SIZE 128

#define NONE -1

#define EOS '\0'

#define NUM 257

#define KEYWORD 258

#define ID 259

#define DONE 260

#define MAX 999

char lexemes[MAX];

char buffer[SIZE];

int lastchar=-1;

int lastentry=0;

int tokenval=DONE;

int lineno=1;

int lookahead;

struct entry

{

char \*lexptr;

int token;

}

symtable[100];

struct entry

keywords[]= {"if",KEYWORD,"else",KEYWORD,"for",KEYWORD,"int",KEYWORD,"float",KEYWORD,

"double",KEYWORD,"char",KEYWORD,"struct",KEYWORD,"return",KEYWORD,0,0

};

void Error\_Message(char \*m)

{

fprintf(stderr,"line %d, %s \n",lineno,m);

exit(1);

}

int look\_up(char s[ ])

{

int k;

for(k=lastentry; k>0; k--)

if(strcmp(symtable[k].lexptr,s)==0)

return k;

return 0;

}

int insert(char s[ ],int tok)

{

int len;

len=strlen(s);

if(lastentry+1>=MAX)

Error\_Message("Symbpl table is full");

if(lastchar+len+1>=MAX)

Error\_Message("Lexemes array is full");

lastentry=lastentry+1;

symtable[lastentry].token=tok;

symtable[lastentry].lexptr=&lexemes[lastchar+1];

lastchar=lastchar+len+1;

strcpy(symtable[lastentry].lexptr,s);

return lastentry;

}

/\*void Initialize()

{

struct entry \*ptr;

for(ptr=keywords;ptr->token;ptr+1)

insert(ptr->lexptr,ptr->token);

}\*/

int lexer()

{

int t;

int val,i=0;

while(1)

{

t=getchar();

if(t==' '||t=='\t');

else if(t=='\n')

lineno=lineno+1;

else if(isdigit(t))

{

ungetc(t,stdin);

scanf("%d",&tokenval);

return NUM;

}

else if(isalpha(t))

{

while(isalnum(t))

{

buffer[i]=t;

t=getchar();

i=i+1;

if(i>=SIZE)

Error\_Message("Compiler error");

}

buffer[i]=EOS;

if(t!=EOF)

ungetc(t,stdin);

val=look\_up(buffer);

if(val==0)

val=insert(buffer,ID);

tokenval=val;

return symtable[val].token;

}

else if(t==EOF)

return DONE;

else

{

tokenval=NONE;

return t;

}

}

}

void Match(int t)

{

if(lookahead==t)

lookahead=lexer();

else

Error\_Message("Syntax error");

}

void display(int t,int tval)

{

if(t=='+'||t=='-'||t=='\*'||t=='/')

printf("\nArithmetic Operator: %c",t);

else if(t==NUM)

printf("\n Number: %d",tval);

else if(t==ID)

printf("\n Identifier: %s",symtable[tval].lexptr);

else

printf("\n Token %d tokenval %d",t,tokenval);

}

void F()

{

//void E();

switch(lookahead)

{

case '(' :

Match('(');

E();

Match(')');

break;

case NUM :

display(NUM,tokenval);

Match(NUM);

break;

case ID :

display(ID,tokenval);

Match(ID);

break;

default :

Error\_Message("Syntax error");

}

}

void T()

{

int t;

F();

while(1)

{

switch(lookahead)

{

case '\*' :

t=lookahead;

Match(lookahead);

F();

display(t,NONE);

continue;

case '/' :

t=lookahead;

Match(lookahead);

display(t,NONE);

continue;

default :

return;

}

}

}

void E ()

{

int t;

T();

while(1)

{

switch(lookahead)

{

case '+' :

t=lookahead;

Match(lookahead);

T();

display(t,NONE);

continue;

case '-' :

t=lookahead;

Match(lookahead);

T();

display(t,NONE);

continue;

default :

return;

}

}

}

void parser()

{

lookahead=lexer();

while(lookahead!=DONE)

{

E();

Match(';');

}

}

int main()

{

char ans[10];

printf("\n Program for recursive descent parsing ");

printf("\n Enter the expression ");

printf("And place ; at the end\n");

printf("Press Ctrl-Z to terminate\n");

parser();

return 0;

}