**Mini Project**

**AIM: Design a predictive parser for a given language.**

**CODE:**

*#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;

}

*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()

{

*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;

}

