Compiler Mini Project

Name:Ravina yadav

Roll No. : 32

Subject:Msc-C.S part-1

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;
```

```c
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);
```

```c
    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
    {
```

```c
            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);
```
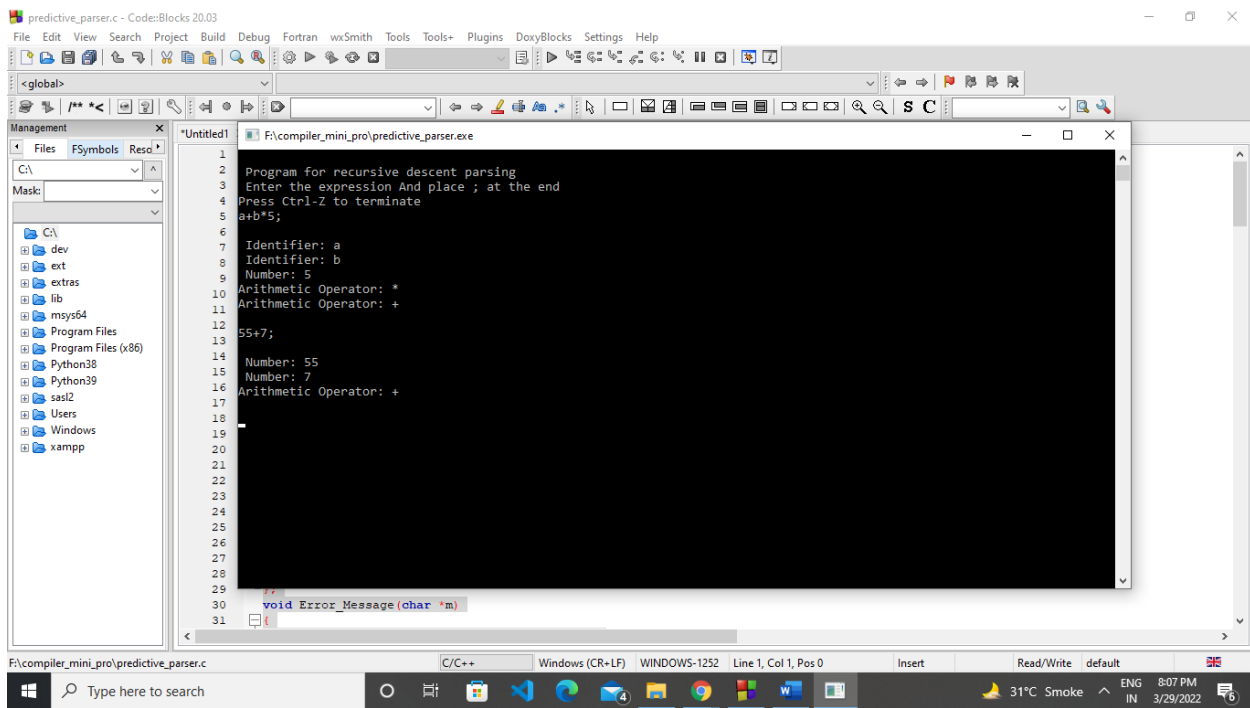
```c
            F();
            display(t,NONE);
            continue;
        case '/' :
            t=lookahead;
            Match(lookahead);
            display(t,NONE);
            continue;
        default :
            return;
        }
    }
}
void E()
{
    int t;
    T();
    while(1)
    {
        switch(lookahead)
        {
        case '+' :
            t=lookahead;
            Match(lookahead);
            T();
            display(t,NONE);
            continue;
        case '-' :
```

```c
            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;
```

}

OUTPUT:

Program for recursive descent parsing

 Enter the expression And place ; at the end

Press Ctrl-Z to terminate

a+b*5;

 Identifier: a

 Identifier: b

 Number: 5

Arithmetic Operator: *

Arithmetic Operator: +

55+7;

 Number: 55

 Number: 7

Arithmetic Operator: +