COMPILERS LAB PROJECT REPORT

Parser for C Language

Submitted by:

Shreyas G 11CO88

Sourabh S 11CO92

Objective:

1. To write a recursive descent parser for C language which reads the tokens from the scanner of Project 1 written using C/C++.
2. To modify the parser such that it generates the abstract syntax tree for C language.

Tools Used: The project was implemented using Dev-C++ IDE.

---

1. INTRODUCTION:

A recursive descent parser is a parser which uses top-down approach. In this method, the parse tree is created from the root to the leaves and the input string is scanned from left to right.

Recursive Descent parser involves backtracking, i.e. if a production rule does not work, it traces back to try other alternatives. Since it is not very efficient, it is not widely used but it is relatively easy to implement.

2. THE ALGORITHM:

This algorithm involves clearly modeling a finite state machine for each of the non-terminals starting off from the start symbol of the grammar. After each of the DFA's have been modeled, the parser then scans the input string and based on the character which is read, it follows the corresponding DFA and progresses by recursively calling the corresponding DFA as required.

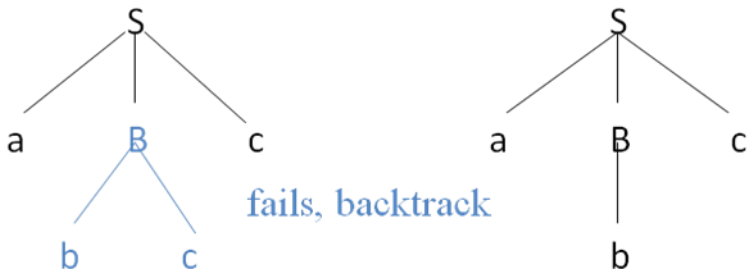
The recursion can have many sub-levels and is dependent on the input string. This is the main reason for the inefficiency of the above method. However, since it is recursive, it is easier to code and understand.

Consider the example in the following page, it clearly shows the nature of the backtracking used by this method.

$S \rightarrow aBc$

$B \rightarrow bc \mid b$

input: abc



The basic algorithm for recursive descent parsing is clearly explained in the below:

$A \rightarrow aBb \mid bAB$

```
proc A {
        case of the current token {
                'a':  - match the current token with a,move to the next token;
                      - call 'B';
                      - match the current token with b, and move to the next token;
                'b':  - match the current token with b, and move to the next token;
                      - call 'A';
                 - call 'B';
        }
}
```

(All the remaining production rules of the grammar are modeled along the same lines.)

To verify the validity of each of the tokens, we have an explicit stack data structure defined along with the pointer to the current input symbol. This procedure uses a non-backtracking approach and uses a parsing table to identify accurately which production rule should be used to get the correct left-most derivation in the parse tree.

Initially, $ is appended to end of input string as well as pushed on top of stack. Then, for each input symbol scanned, the production rule is obtained after looking up the parsing table and then inserting them in the stack in reverse order of the right-hand side of the production rule. If input symbol read is same as top of stack, then the input symbol is popped. If the entire string is read and both input symbol and stack top have $, it then indicates success.

3. CODE:

```
# include<stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>

char str[20];
char TOKEN[20];
int lineno=1;
int p0=0;
char in[100][100];
char ptree[50];
int i=0,n,p=0,q;
char token[20];

int B();
int D();
int S();
int T();
int I();
int A();
int E();
int Q();
int R();
int U();
int M();
int N();
int id();
int constant();
int S1();
int E1();
void ntoken();
void btoken();

#define DO 1
#define IF    2
#define ELSE  3
#define INT   4
#define RETURN 5
#define VOID  6
#define WHILE 7
#define PLUS  8        // +
#define MINUS 9        // -
#define MUL   10       // *
#define DIVIDE 11      // /
#define LT    12       // <
#define LE    13       // <=
#define GT        14       // >
#define GE    15       // >=
#define EE    16       // ==
#define NE    17       // !=
#define EQ    18   // =
#define SEMECOLON 19                  // ;
#define COMMA    20             // ,
#define L_PARENTHESIS 21    // (
#define R_PARENTHESIS 22   // )
#define LS_BRACKET  23                // [
#define RS_BRACKET  24    // ]
#define LB_RBRACKET  25               // {
#define RB_RBRACKET  26    // }
#define DM       27                // /*
#define MD       28                // */
#define NUM      29
#define ID       30
#define MOD      31               // %
#define AND      32               // &&
#define OR       33               // ||
#define NOT      34               // !
#define BITW1    35               // &
#define BITW2    36               // |
#define BITW3    37               // ^
#define PRE_PROCESSOR 61
#define STRING 62

int lookup(char* p)
{
        strcpy(str,p);
        if(strcmp(str,"if")==0)
                return 2;
        else if(strcmp(str,"else")==0)
                return 3;
        else if(strcmp(str,"int")==0)
                return 4;
        else if(strcmp(str,"return")==0)
                return 5;
        else if(strcmp(str,"void")==0)
                return 6;
        else if(strcmp(str,"while")==0)
                return 7;
        else if(strcmp(str,"do")==0)
                return 1;
        else if(strcmp(str,"break")==0)
                return 38;
        else if(strcmp(str,"case")==0)
                return 39;
        else if(strcmp(str,"char")==0)
                return 40;
        else if(strcmp(str,"const")==0)
                return 41;
        else if(strcmp(str,"continue")==0)
                return 42;
```

```
	else if(strcmp(str,"default")==0)
		return 43;
	else if(strcmp(str,"double")==0)
		return 45;
	else if(strcmp(str,"enum")==0)
		return 46;
	else if(strcmp(str,"extern")==0)
		return 47;
	else if(strcmp(str,"float")==0)
		return 48;
	else if(strcmp(str,"for")==0)
		return 49;
	else if(strcmp(str,"goto")==0)
		return 50;
	else if(strcmp(str,"long")==0)
		return 51;
	else if(strcmp(str,"short")==0)
		return 52;
	else if(strcmp(str,"signed")==0)
		return 53;
	else if(strcmp(str,"sizeof")==0)
		return 54;
	else if(strcmp(str,"static")==0)
		return 55;
	else if(strcmp(str,"struct")==0)
		return 56;
	else if(strcmp(str,"switch")==0)
		return 57;
	else if(strcmp(str,"typedef")==0)
		return 58;
	else if(strcmp(str,"union")==0)
		return 59;
	else if(strcmp(str,"unsigned")==0)
		return 60;
	else
		return 0;
}
void out(int sortnum,char*p)
{
	char str2[20];
	strcpy(str2,p);
	if(sortnum<8&&sortnum>0||sortnum
<61&&sortnum>37)
		printf("%d is a
Keyword\t\t%s\n",lineno,str2);
	else
if(sortnum<19&&sortnum>7||sortnum<38&&s
ortnum>30)
		printf("%d is an
Operator\t%s\n",lineno,str2);
	else if(sortnum<29&&sortnum>18)
		printf("%d is a
Symbol\t\t%s\n",lineno,str2);
	else if(sortnum==29)
		printf("%d is a
Constant\t%s\n",lineno,str2);
	else if(sortnum==30)
		printf("%d is an
Identifier\t%s\n",lineno,str2);
	else if(sortnum==61)
		printf("%d is a
Proprocessor\t%s\n",lineno,str2);
	else if(sortnum==62)
		printf("%d is a
String\t\t%s\n",lineno,str2);

	strcpy(in[i++],p);

}
void report_error()
{
	printf("%d\terror : Illegal
Identifier\n",lineno);
	po=1;
}
void report_error1()
{
	printf("%d\terror : Illegal
Character\n",lineno);
	po=1;
}
int ischar(char * p)
{
	char str3[20];
	int m,i;
	strcpy(str3,p);
	m=strlen(p);
	for(i=1;i<m;i++)
	{
		if(str3[i]>='a'&&str3[i]<='z' ||
str3[i]>='A'&&str3[i]<='Z')
			return 0;
	}


	return 1;
}
void scanner(FILE *fp,FILE *fp1)
{
	char ch;
	int i,c;
	ch=fgetc(fp);
```

```
//          printf("%c",ch);
            if(isalpha(ch))
            {
                        TOKEN[0]=ch;
                        i=1;
                        ch=fgetc(fp);
                        while(isalnum(ch))
                        {
                                    TOKEN[i]=ch;i++;
                                    ch=fgetc(fp);
                        }
      TOKEN[i]='\0';
                        fseek(fp,-1,1);
                        c=lookup(TOKEN);
                        if(c==0)
                        {
                                    fputs(TOKEN,fp1);
                                    fputs("\n",fp1);
                                    out(ID,TOKEN);
                        }
                        else
                        {
                                    out(c,str);

                        }
            }
            else
            {
                        if(isdigit(ch))
                        {
                                    TOKEN[0]=ch;
                                    ch=fgetc(fp); i=1;

                                    while(isalnum(ch))
                                    {
                                                TOKEN[i]=ch;
i++;
                                                ch=fgetc(fp);

                                    }
                                    TOKEN[i]='\0';
                                    fseek(fp,-1,1);
                                    int c=ischar(TOKEN);
                                    if(c==1)

            out(NUM,TOKEN);
                                    else

            report_error();
                        }
                        else
                        {
                                    switch(ch)
                                    {
                                    case '+':

            out(PLUS,"+");

                                                break;
                                    case '-':
                                                ch=fgetc(fp);
                                                out(MINUS,"-
");


                                                break;
                                    case '*':
                                                ch=fgetc(fp);
                                                if(ch=='/')
                                                {

            out(MD,"*/");
                                                }

                                                else
                                                {

            fseek(fp,-1,1);

            out(MUL,"*");

                                                }
                                                break;
                                    case '/':
                                                ch=fgetc(fp);
                                                //
            printf("%c",ch);
                                                if(ch=='*')
                                                {

            out(DM,"/*");

                                                }

                                                else
                                                {

            fseek(fp,-1,1);

            out(DIVIDE,"/");
```

```
                        }
                        break;
                case '<':
                        ch=fgetc(fp);
                        if(ch=='=')
                        {

out(LE,"<=");

                        }
                        else
                        {

fseek(fp,-1,1);

out(LT,"<");

                        }
                        break;
                case '>':
                        ch=fgetc(fp);
                        if(ch=='=')
                        {

out(GE,">=");

                        }
                        else
                        {

fseek(fp,-1,1);

out(GT,">");

                        }
                        break;
                case '=':
                        ch=fgetc(fp);
                        if(ch=='=')
                        {

out(EE,"==");

                        }
                        else
                        {

fseek(fp,-1,1);

out(EQ,"=");

                        }
                        break;
                case '!':
                        ch=fgetc(fp);
                        if(ch=='=')
                        {

        out(NE,"!=");

                        }
                        else
                        {

        fseek(fp,-1,1);

        report_error1();
                        }
                        break;
                case ';':

        out(SEMECOLON,";");
                        break;
                case ',':

        out(COMMA,",");

                        break;
                case '(':

        out(L_PARENTHESIS,"(");
                        break;
                case ')':
                        out(
R_PARENTHESIS,")");
                        break;
                case '{':
                        out(
LB_RBRACKET,"{");
                        break;
                case '}':

        out(RB_RBRACKET,"}");
                        break;
                case '[':
                        out(
LS_BRACKET,"[");
                        break;
                case ']':
                        out(
RS_BRACKET,"]");
                        break;
```

```
                        case '#':

        TOKEN[o]=ch;
                                i=1;
                                ch=fgetc(fp);
                                while(ch!='>')
                                {

        TOKEN[i++]=ch;

        ch=fgetc(fp);

                                }

        TOKEN[i++]=ch;

        TOKEN[i]='\o';
                                out(
PRE_PROCESSOR,TOKEN);
                                break;

                        case ' ':

                                break;
                        case '\n':
                                lineno++;
                                break;
                        case '\r':

                                break;
                        case '\t':

                                break;
                        case '"':

        TOKEN[o]=ch;
                                i=o;
                                ch=fgetc(fp);
                                while(ch!='"')
                                {

        TOKEN[i++]=ch;

        ch=fgetc(fp);

                                }

        //TOKEN[i++]=ch;

        TOKEN[i]='\o';
                                out(
STRING,TOKEN);
                                break;

                        default:

                report_error1();
                                break;

                        }

                        return;
                }
        }

}
void ntoken()
{
        strcpy(token,in[i]);
        i++;
        if(i<=n)
        {
        //      printf("%s\n",token);
        }

}

void btoken(int j)
{
        i=j-1;
        ntoken();


}



int B()
{
        ptree[p++]='}';
        ptree[p++]='S';
        ptree[p++]='D';
        ptree[p]='{';
        printf("\n");
        for(q=o;q<=p;q++)
                printf(" %c ",ptree[q]);


        if(!strcmp(token,"{"))
        {
                p--;
```

```
                printf("\n");
                for(q=0;q<=p;q++)
                        printf(" %c
",ptree[q]);

                ntoken();
                D();
                S();
                if(!strcmp(token,"}"))
                {
                        p--;
                        printf("\n");
                        for(q=0;q<=p;q++)
                                printf(" %c
",ptree[q]);

                        ntoken();

                        if(i==n)
                        {

        for(q=0;q<=p;q++)

        printf(" %c ",ptree[q]);
                        //      printf("\nright
syntax\n");
                                return 1;
                        }



                }
                else
                {printf("error1\n");
                po=1;}

        }
        else
        {
                //printf("error2\n");
                return 0;
        }

return 1;
}

int D()
{
        int j=i,r=p;

        ptree[p++]='D';
        ptree[p++]=';';
        ptree[p++]='l';
        ptree[p]='T';
        printf("\n");
        for(q=0;q<=p;q++)
                printf(" %c ",ptree[q]);


        if(T())
                if(l())
                        if(!strcmp(token,";"))
                        {
                                p--;
                                printf("\n");

        for(q=0;q<=p;q++)

        printf(" %c ",ptree[q]);

                                ntoken();

                                D();
                                return 1;
                        }

        btoken(j);
        p=r-1;
        printf("\n");
        for(q=0;q<=p;q++)
                printf(" %c ",ptree[q]);

        return 1;
}

int T()
{
        if(strcmp(token,"void"))
                if(strcmp(token,"int"))

        if(strcmp(token,"float"))

        if(strcmp(token,"string"))
                                {
                        //
        printf("error4\n");

        return 0;
                                }

        ntoken();
        return 1;
```

```
}

int I()
{
	ptree[p++]='A';
	ptree[p]='i';
	printf("\n");
	for(q=0;q<=p;q++)
		printf(" %c ",ptree[q]);


	if(id())
	{
		p--;
		printf("\n");
		for(q=0;q<=p;q++)
			printf(" %c
",ptree[q]);

		if(A())
		{
			return 1;
		}
		else
		{
			printf("Error5\n");
			p0=1;
		}
	}

	return 0;
}

int A()
{
	int j=i,r=p;
	ptree[p]='E';
	printf("\n");
	for(q=0;q<=p;q++)
		printf(" %c ",ptree[q]);



	if(!strcmp(token,"="))
	{
		p--;
		printf("\n");
		for(q=0;q<=p;q++)
			printf(" %c
",ptree[q]);
		ntoken();
		if(E())
			return 1;
	}

	btoken(j);
	p=r-1;
	printf("\n");
	for(q=0;q<=p;q++)
		printf(" %c ",ptree[q]);

	return 1;
}

int E()
{
	ptree[p++]='U';
	ptree[p++]='R';
	ptree[p]='Q';
	printf("\n");
	for(q=0;q<=p;q++)
		printf(" %c ",ptree[q]);


	if(Q())
		if(R())
			if(U())
				return 1;

	return 0;
}
int Q()
{
	int j=i,r=p;
	ptree[p++]='Q';
	ptree[p++]='M';
	ptree[p]='U';
	printf("\n");
	for(q=0;q<=p;q++)
		printf(" %c ",ptree[q]);


	if(U())
		if(M())
		{
			Q();
			return 1;
		}

	btoken(j);
```

```
	p=r-1;
	printf("\n");
	for(q=0;q<=p;q++)
		printf(" %c ",ptree[q]);
	return 1;
}

int M()
{
	if(strcmp(token,"="))
		if(strcmp(token,"*="))
			if(strcmp(token,"/="))

	if(strcmp(token,"+="))

	if(strcmp(token,"-="))

	if(strcmp(token,"%="))

		return 0;

	p--;
		printf("\n");
		for(q=0;q<=p;q++)
			printf(" %c
",ptree[q]);

	ntoken();
	return 1;
}

int R()
{
	int j=i,r=p;
	ptree[p++]='R';
	ptree[p++]='N';
	ptree[p]='U';
	printf("\n");
	for(q=0;q<=p;q++)
		printf(" %c ",ptree[q]);
	if(U())
		if(N())
		{
			R();
			return 1;
		}

	btoken(j);
	p=r-1;
	printf("\n");
	for(q=0;q<=p;q++)
		printf(" %c ",ptree[q]);
	return 1;
}

int N()
{
		if(strcmp(token,"*"))
			if(strcmp(token,"/"))

	if(strcmp(token,"+"))

	if(strcmp(token,"-"))

	if(strcmp(token,"%"))

		return 0;

	p--;
		printf("\n");
		for(q=0;q<=p;q++)
			printf(" %c
",ptree[q]);

	ntoken();
	return 1;
}

int U()
{
	ptree[p]='i';
	printf("\n");
	for(q=0;q<=p;q++)
		printf(" %c ",ptree[q]);

	if(id()||constant())
	{
		p--;
		printf("\n");
		for(q=0;q<=p;q++)
			printf(" %c
",ptree[q]);
		return 1;
	}

//	printf("error3\n");
	return 0;
}

int id()
{
```

```
	if((!strcmp(token,"a"))||(!strcmp(toke
n,"b"))||(!strcmp(token,"c"))||(!strcmp(token,"
d")))
	{
		ntoken();
		return 1;
	}

	return 0;

}


int constant()
{
	if((!strcmp(token,"0"))||(!strcmp(toke
n,"1"))||(!strcmp(token,"2"))||(!strcmp(token,"
3")))
	{
		ntoken();
		return 1;
	}

	return 0;


}

int S()
{
	int j=i,r=p;
	ptree[p++]='S';
	ptree[p]='1';
	printf("\n");
	for(q=0;q<=p;q++)
		printf(" %c ",ptree[q]);


	if(S1())
	{
		S();
		return 1;
	}

	btoken(j);
	p=r-1;
	printf("\n");
	for(q=0;q<=p;q++)
		printf(" %c ",ptree[q]);
	return 1;
}
```

```
int S1()
{
	int j=i,r=p;

	ptree[p++]=';';
	ptree[p]='1';
	printf("\n");
	for(q=0;q<=p;q++)
		printf(" %c ",ptree[q]);


	if(E1())
	{
		if(!strcmp(token,";"))
		{
			p--;
			printf("\n");
			for(q=0;q<=p;q++)
				printf(" %c
",ptree[q]);

			ntoken();
			return 1;
		}
	}

	p=r;
	ptree[p]='B';
	printf("\n");
	for(q=0;q<=p;q++)
		printf(" %c ",ptree[q]);

	if(B())
	{
		return 1;
	}

	p=r;
	ptree[p++]='1';
	ptree[p++]=')';
	ptree[p++]='E';
	ptree[p++]='(';
	ptree[p]='f';
	printf("\n");
	for(q=0;q<=p;q++)
		printf(" %c ",ptree[q]);

	if(!strcmp(token,"if"))
	{
		p--;
```

```
                printf("\n");
                for(q=0;q<=p;q++)
                        printf(" %c
",ptree[q]);

                ntoken();
                if(!strcmp(token,"("))
                {
                        p--;
                        printf("\n");
                        for(q=0;q<=p;q++)
                                printf(" %c
",ptree[q]);

                        ntoken();
                        if(E())
                        {

        if(!strcmp(token,")"))
                                {
                                        p--;

        printf("\n");

        for(q=0;q<=p;q++)

        printf(" %c ",ptree[q]);


        ntoken();

        if(S1())
                                        {

        return 1;
                                        }
                                }
                        }
                }
        }

        p=r;
        ptree[p++]='1';
        ptree[p++]='e';
        ptree[p++]='1';
        ptree[p++]=')';
        ptree[p++]='E';
        ptree[p++]='(';
        ptree[p]='f';
        printf("\n");
        for(q=0;q<=p;q++)
                printf(" %c ",ptree[q]);

        if(!strcmp(token,"if"))
        {
                p--;
                printf("\n");
                for(q=0;q<=p;q++)
                        printf(" %c
",ptree[q]);

                if(!strcmp(token,"("))
                {
                        p--;
                        printf("\n");
                        for(q=0;q<=p;q++)
                                printf(" %c
",ptree[q]);

                        ntoken();
                        if(E())
                        {

        if(!strcmp(token,")"))
                                {
                                        p--;

        printf("\n");

        for(q=0;q<=p;q++)

        printf(" %c ",ptree[q]);


        ntoken();

        if(S1())
                                {

        if(!strcmp(token,"else"))

        {

                printf("\n");

                for(q=0;q<=p;q++)

                        printf(" %c
",ptree[q]);
```

```
                ntoken();

                if(S1())

                {

                        return 1;

                }

        }
                                                }
                                        }
                        }
                }
        }

        p=r;
        ptree[p++]='1';
        ptree[p++]=')';
        ptree[p++]='E';
        ptree[p++]='(';
        ptree[p]='w';
        printf("\n");
        for(q=0;q<=p;q++)
                printf(" %c ",ptree[q]);
        if(!strcmp(token,"do"))
        {
                p--;
                printf("\n");
                for(q=0;q<=p;q++)
                        printf(" %c
",ptree[q]);

                ntoken();
                if(S())
                {
                        return 1;
                }

        }

        if(!strcmp(token,"while"))
        {
                p--;
                printf("\n");
                for(q=0;q<=p;q++)
                        printf(" %c
",ptree[q]);

                ntoken();
                if(!strcmp(token,"("))
                {
                        p--;
                        printf("\n");
                        for(q=0;q<=p;q++)
                                printf(" %c
",ptree[q]);

                        ntoken();
                        if(E())
                        {

        if(!strcmp(token,")"))
                                {
                                        p--;

        printf("\n");

        for(q=0;q<=p;q++)

        printf(" %c ",ptree[q]);


        ntoken();

        if(S1())
                                        {

        return 1;
                                        }
                                }
                        }
                }
        }

        p=r;
        ptree[p++]='1';
        ptree[p++]=')';
        ptree[p++]='1';
        ptree[p++]=';';
        ptree[p++]='1';
        ptree[p++]=';';
        ptree[p++]='1';
        ptree[p++]='(';
        ptree[p]='0';
        printf("\n");
        for(q=0;q<=p;q++)
```

```
                printf(" %c ",ptree[q]);


        if(!strcmp(token,"for"))
        {
                p--;
                printf("\n");
                for(q=0;q<=p;q++)
                        printf(" %c
",ptree[q]);

                ntoken();
                if(!strcmp(token,"("))
                {
                        p--;
                        printf("\n");
                        for(q=0;q<=p;q++)
                                printf(" %c
",ptree[q]);

                        ntoken();
                        if(E1())
                        {

        if(!strcmp(token,";"))
                                {
                                        p--;

        printf("\n");

        for(q=0;q<=p;q++)

        printf(" %c ",ptree[q]);


        ntoken();

        if(E1())
                                                {

        if(!strcmp(token,";"))

        {

                p--;

                printf("\n");

                for(q=0;q<=p;q++)
```

```
                                printf(" %c
",ptree[q]);


                        ntoken();

                        if(E1())

                        {

                                if(!strcmp(token,")"))

                                {

                                        p--;

                                        printf("\n");


        for(q=0;q<=p;q++)


        printf(" %c ",ptree[q]);

                                        ntoken();

                                        if(S1())

                                        {


        return 1;

                                        }

                                }

                        }

        }
                                                }
                                        }
                                }
                        }
        }


        p=r;
        ptree[p++]=';';
```

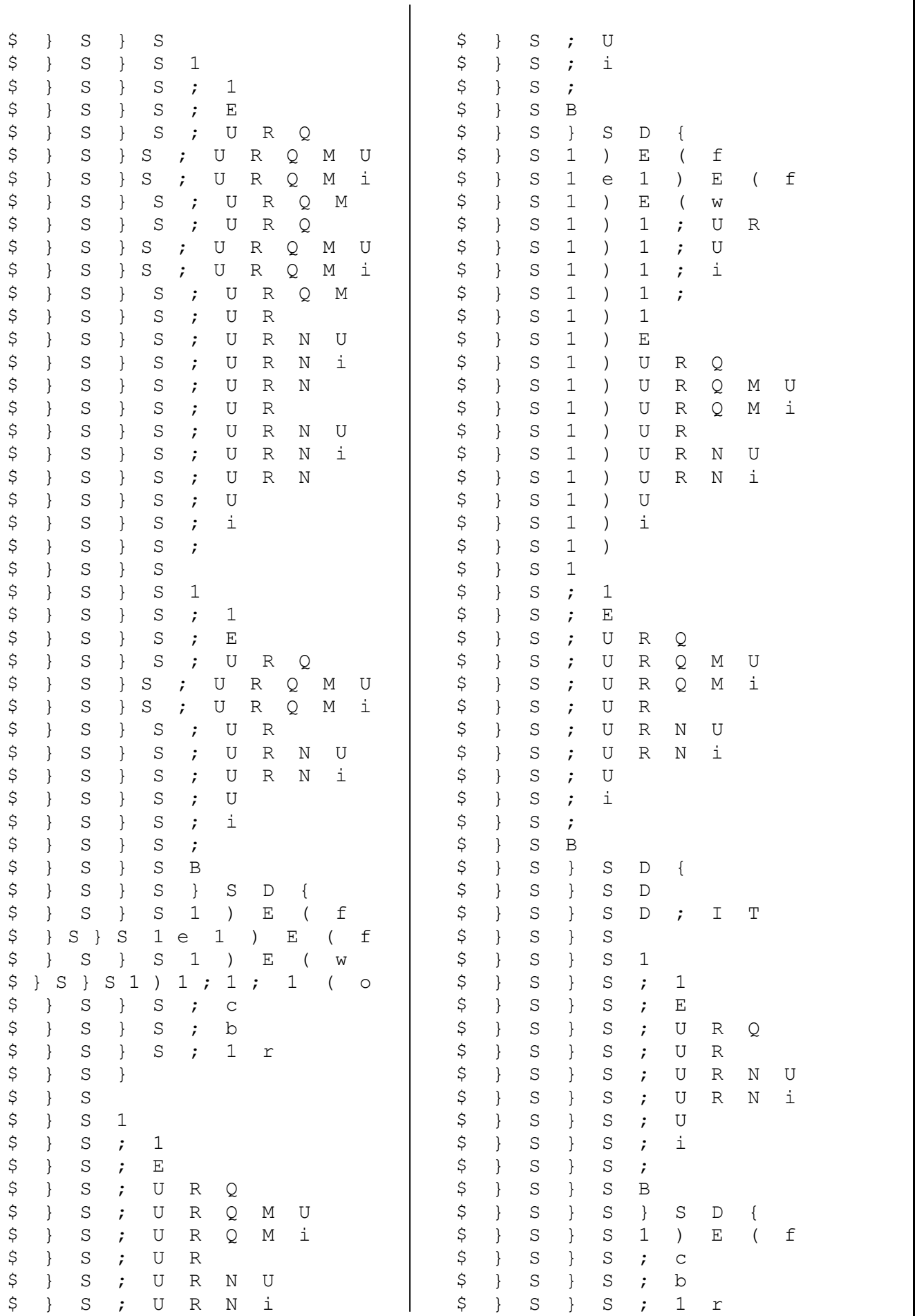
```
		ptree[p]='c';
		printf("\n");
		for(q=0;q<=p;q++)
			printf(" %c ",ptree[q]);

		if(!strcmp(token,"continue"))
		{
			p--;
			printf("\n");
			for(q=0;q<=p;q++)
				printf(" %c
",ptree[q]);

			ntoken();
			if(!strcmp(token,";"))
			{
				p--;
				printf("\n");
				for(q=0;q<=p;q++)
					printf(" %c
",ptree[q]);

				ntoken();
				return 1;
			}
		}

		p=r;
		ptree[p++]=';';
		ptree[p]='b';
		printf("\n");
		for(q=0;q<=p;q++)
			printf(" %c ",ptree[q]);

		if(!strcmp(token,"break"))
		{
			p--;
			printf("\n");
			for(q=0;q<=p;q++)
				printf(" %c
",ptree[q]);

			ntoken();
			if(!strcmp(token,";"))
			{
				p--;
				printf("\n");
				for(q=0;q<=p;q++)
					printf(" %c
",ptree[q]);

				ntoken();
				return 1;
			}
		}

		p=r;
		ptree[p++]=';';
		ptree[p++]='1';
		ptree[p]='r';
		printf("\n");
		for(q=0;q<=p;q++)
			printf(" %c ",ptree[q]);

		if(!strcmp(token,"return"))
		{
			p--;
			printf("\n");
			for(q=0;q<=p;q++)
				printf(" %c
",ptree[q]);

			ntoken();
			if(E1())
			{
				return 1;
			}
		}
		return 0;
}


int E1()
{
		int j=i,r=p;
		ptree[p]='E';
		printf("\n");
		for(q=0;q<=p;q++)
			printf(" %c ",ptree[q]);

		if(E())
			return 1;

		btoken(j);
		p=r-1;
		printf("\n");
		for(q=0;q<=p;q++)
			printf(" %c ",ptree[q]);
		return 1;
}
main(int argc, char* argv[])
{
```

```
        char s[100];
        int k=0,x;

        FILE* fp;
        FILE* fp1;

        if(argc!=2)
        {
                printf("You forgot to enter a
filename\n");
                exit(0);
        }

        if((fp=fopen(argv[1],"r"))==NULL)
        {
                printf("Can not open the
file\n",argv[1]);
                exit(0);
        }

        else
                {


        fp1=fopen("abc.txt","w");

                        while(fgetc(fp)!=EOF)
                                {
                                        fseek(fp,-1,1);

                scanner(fp,fp1);

                                }

                                fclose(fp);
                                fclose(fp1);
                        }
                n=i;
                i=0;
                ntoken();
                printf("\nS -> ");
                ptree[p++]='$';
                B();

                printf("\n");
                if(p!=0 && p0==0)
                {
                printf(" $ } \n");
                printf(" $\n");
                }


        }
```

4. OUTPUT:

Sample Program:

```
1 is a Symbol          {
2 is a Keyword         int
2 is an Identifier     a
2 is an Operator=
2 is a Constant 2
2 is a Symbol          ;
3 is a Keyword         if
3 is a Symbol          (
3 is an Identifier     a
3 is a Symbol          )
4 is a Symbol          {
5 is an Identifier     a
5 is an Operator=
5 is an Identifier     a
5 is an Operator+
5 is a Constant 3
5 is a Symbol          ;
6 is a Symbol          }
8 is a Keyword         for
8 is a Symbol          (
8 is a Symbol          ;
8 is a Symbol          ;
8 is a Symbol          )
8 is a Symbol          {
8 is a Symbol          }
10 is a Keyword        while
10 is a Symbol         (
10 is an Identifier    a
10 is a Symbol         )
11 is a Symbol         {
11 is a Symbol         }
14 is a Keyword        if
14 is a Symbol         (
14 is an Identifier    a
14 is a Symbol         )
14 is a Symbol         ;
17 is a Keyword        do
18 is a Symbol         {
18 is a Symbol         }
19 is a Keyword        while
19 is a Symbol         (
19 is an Identifier    a
19 is a Symbol         )
19 is a Symbol         ;
22 is a Symbol         }

S ->
 $ } S D {
 $ } S D
 $ } S D ; I T
 $ } S D ; I A i
 $ } S D ; I A
 $ } S D ; I E
 $ } S D ; I
 $ } S D ; U R Q
 $ } S D ; U R Q M U
 $ } S D ; U R Q M i
 $ } S D ; U R Q M
 $ } S D ; U R
 $ } S D ; U R N U
 $ } S D ; U R N i
 $ } S D ; U R N
 $ } S D ; U
 $ } S D ; i
 $ } S D ;
 $ } S D
 $ } S D ; I T
 $ } S
 $ } S 1
 $ } S ; 1
 $ } S ; E
 $ } S ; U R Q
 $ } S ; U R Q M U
 $ } S ; U R Q M i
 $ } S ; U R
 $ } S ; U R N U
 $ } S ; U R N i
 $ } S ; U
 $ } S ; i
 $ } S ;
 $ } S B
 $ } S } S D {
 $ } S } S D
 $ } S } S D ; I T
 $ } S 1 ) E ( f
 $ } S 1 ) E (
 $ } S 1 ) E
 $ } S 1 ) U R Q
 $ } S 1 ) U R Q M U
 $ } S 1 ) U R Q M i
 $ } S 1 ) U R Q M
 $ } S 1 ) U R
 $ } S 1 ) U R N U
 $ } S 1 ) U R N i
 $ } S 1 ) U R N
 $ } S 1 ) U
 $ } S 1 ) i
 $ } S 1 )
 $ } S 1
 $ } S ; 1
 $ } S ; E
 $ } S ; U R Q
 $ } S ; U R Q M U
 $ } S ; U R Q M i
 $ } S ; U R
 $ } S ; U R N U
 $ } S ; U R N i
 $ } S ; U
 $ } S ; i
 $ } S ;
 $ } S B
 $ } S } S D {
 $ } S } S D
 $ } S } S D ; I T
```


$ } S } S
$ } S } S 1
$ } S } S ; 1
$ } S } S ; E
$ } S } S ; U R Q
$ } S } S ; U R Q M U
$ } S } S ; U R Q M i
$ } S } S ; U R Q M
$ } S } S ; U R Q
$ } S } S ; U R Q M U
$ } S } S ; U R Q M i
$ } S } S ; U R Q M
$ } S } S ; U R
$ } S } S ; U R N U
$ } S } S ; U R N i
$ } S } S ; U R N
$ } S } S ; U
$ } S } S ; i
$ } S } S ;
$ } S } S
$ } S } S 1
$ } S } S ; 1
$ } S } S ; E
$ } S } S ; U R Q
$ } S } S ; U R Q M U
$ } S } S ; U R Q M i
$ } S } S ; U R
$ } S } S ; U R N U
$ } S } S ; U R N i
$ } S } S ; U
$ } S } S ; i
$ } S } S ;
$ } S } S B
$ } S } S } S D {
$ } S } S 1) E (f
$ } S } S 1 e 1) E (f
$ } S } S 1) E (w
$ } S } S 1) 1 ; 1 ; 1 (o
$ } S } S ; c
$ } S } S ; b
$ } S } S ; 1 r
$ } S }
$ } S
$ } S 1
$ } S ; 1
$ } S ; E
$ } S ; U R Q
$ } S ; U R Q M U
$ } S ; U R Q M i
$ } S ; U R
$ } S ; U R N U
$ } S ; U R N i
$ } S ; U
$ } S ; i
$ } S ;
$ } S B
$ } S } S D {
$ } S } S D
$ } S } S D ; I T
$ } S } S
$ } S } S 1
$ } S } S ; 1
$ } S } S ; E
$ } S } S ; U R Q
$ } S } S ; U R
$ } S } S ; U R N U
$ } S } S ; U R N i
$ } S } S ; U
$ } S } S ; i
$ } S } S ;
$ } S } S B
$ } S } S } S D {
$ } S } S 1) E (f
$ } S } S ; c
$ } S } S ; b
$ } S } S ; 1 r

```
$ } S }
$ } S
$ } S 1
$ } S ; 1
$ } S ; E
$ } S ; U R Q
$ } S ; U R Q M U
$ } S ; U R Q M i
$ } S ; U R
$ } S ; U R N U
$ } S ; U R N i
$ } S ; U
$ } S ; i
$ } S ;
$ } S B
$ } S } S D {
$ } S 1 ) E ( f
$ } S 1 e 1 ) E ( f
$ } S 1 ) E ( w
$ } S 1 ) E (
$ } S 1 ) E
$ } S 1 ) U R Q
$ } S 1 ) U R Q M U
$ } S 1 ) U R Q M i
$ } S 1 ) U R Q M
$ } S 1 ) U R
$ } S 1 ) U R N U
$ } S 1 ) U R N i
$ } S 1 ) U R N
$ } S 1 ) U
$ } S 1 ) i
$ } S 1 )
$ } S 1
$ } S ; 1
$ } S ; E
$ } S ; U R Q
$ } S ; U R Q M U
$ } S ; U R Q M i
$ } S ; U R
$ } S ; U R N U
$ } S ; U R N i
$ } S ; U
$ } S ; i
$ } S ;
$ } S B
$ } S } S D {
$ } S } S D
$ } S } S D ; I T
$ } S } S
$ } S } S 1
$ } S } S ; 1
$ } S } S ; E
$ } S } S ; U R Q
$ } S } S ; U R N U
$ } S } S ; U R N i
$ } S } S ; U
```

```
$ } S } S ; i
$ } S } S ;
$ } S } S B
$ } S } S } S D {
$ } S } S 1 ) E ( f
$ } S } S ; c
$ } S } S ; b
$ } S } S ; 1 r
$ } S } S }
$ } S } S
$ } S 1
$ } S ; 1
$ } S ; E
$ } S ; U R Q
$ } S ; U R Q M U
$ } S ; U R Q M i
$ } S ; U R
$ } S ; U R N U
$ } S ; U R N i
$ } S ; U
$ } S ; i
$ } S ;
$ } S B
$ } S } S D {
$ } S 1 ) E ( f
$ } S 1 ) E (
$ } S 1 ) E
$ } S 1 ) U R Q
$ } S 1 ) U R Q M U
$ } S 1 ) U R Q M i
$ } S 1 ) U R Q M
$ } S 1 ) U R
$ } S 1 ) U R N U
$ } S 1 ) U R N i
$ } S 1 ) U R N
$ } S 1 ) U
$ } S 1 ) i
$ } S 1 )
$ } S 1
$ } S ; 1
$ } S ; E
$ } S ; U R Q
$ } S ; U R Q M U
$ } S ; U R Q M i
$ } S ; U R
$ } S ; U R N U
$ } S ; U R N i
$ } S ; U
$ } S ; i
$ } S ;
$ } S
$ } S 1
$ } S ; 1
$ } S ; E
$ } S ; U R Q
$ } S ; U R Q M U
```

```
$ } S ; U R Q M i
$ } S ; U R
$ } S ; U R N U
$ } S ; U R N i
$ } S ; U
$ } S ; i
$ } S ;
$ } S B
$ } S } S D {
$ } S 1 ) E ( f
$ } S 1 e 1 ) E ( f
$ } S 1 ) E ( w
$ } S 1 ) E (
$ } S 1 ) E S 1
$ } S 1 ) E S ; 1
$ } S 1 ) E S ; E
$ } S 1 ) E S } E S
$ } S 1 ) E S } S 1
$ } S 1 ) E S } S ; 1
$ } S 1 ) E S } S ; E
$ } S 1 ) E S } S ; U
$ } S 1 ) E S } S ; i
$ } S 1 ) E S } S ;
$ } S 1 ) E S } S B
$ } S 1 ) E S } ; 1 r
$ } S 1 ) E S }
$ } S 1 ) E S
$ } S 1 ) E S 1
$ } S 1 ) E S ; 1
$ } S 1 ) E S ; E
$ } S 1 ) E S ; U
$ } S 1 ) E S ; i
$ } S 1 ) E S ;
$ } S 1 ) E S B
$ } S 1 ) E S 1
$ } S 1 ) E S 1 )
$ } S 1 ) E S 1
$ } S 1 ) E S ; 1
$ } S 1 ) E S ; U
$ } S 1 ) E S ; i
```

```
$ } S 1 ) E S ;
$ } S 1 ) E S
$ } S 1 ) E S 1
$ } S 1 ) E S ; 1
$ } S 1 ) E S ; E
$ } S 1 ) E S ; U
$ } S 1 ) E S ; i
$ } S 1 ) E S ;
$ } S 1 ) E S B
$ } S 1 ) E S } S D {
$ } S 1 ) E S 1 ) E ( f
$ } S 1 ) E S ; c
$ } S 1 ) E S ; b
$ } S 1 ) E S ; 1 r
$ } S 1 ) E
$ } S 1 ) S 1
$ } S 1 ) S ; 1
$ } S 1 ) S ; E
$ } S 1 ) S ; U R Q
$ } S 1 ) S ; U R Q M U
$ } S 1 ) S ; U R
$ } S 1 ) S ; U R N U
$ } S 1 ) S ; U R N i
$ } S 1 ) S ; U
$ } S 1 ) S ; i
$ } S 1 ) S ;
$ } S 1 ) S B
$ } S 1 ) S } S D {
$ } S 1 ) S 1 ) E ( f
$ } S 1 ) S 1 e 1 ) E ( f
$ } S 1 ) S 1 ) E ( w
$ } S 1 ) S 1 ) 1 ; 1 ; 1( o
$ } S 1 ) S ; c
$ } S 1 ) S ; b
$ } S 1 ) S ; 1 r
$ } S 1 )
$ } S 1
$ }
$
//ends at $,successful parsing
```

Sample Program 2 (with error):

```
{
        int a=2;
        if(a)
        {
                a=a+3;
        }
        float b=5                       // missing semicolon
        b++;
}
```

```
1 is a Symbol          {
2 is a Keyword         int
2 is an Identifier     a
2 is an Operator=
2 is a Constant 2
2 is a Symbol          ;
3 is a Keyword         if
3 is a Symbol          (
3 is an Identifier     a
3 is a Symbol          )
4 is a Symbol          {
5 is an Identifier     a
5 is an Operator=
5 is an Identifier     a
5 is an Operator+
5 is a Constant 3
5 is a Symbol          ;
6 is a Symbol          }
8 is a Keyword         float
8 is an Identifier     b
8 is an Operator=
8 is a Constant 5
9 is an Identifier     b
9 is an Operator+
9 is an Operator+
9 is a Symbol          ;
10 is a Symbol         }
S ->
 $ } S D {
 $ } S D
 $ } S D ; I T
 $ } S D ; I A i
 $ } S D ; I A
 $ } S D ; I E
 $ } S D ; I
 $ } S D ; U R Q
 $ } S D ; U R Q M U
 $ } S D ; U R Q M i
 $ } S D ; U R Q M
 $ } S D ; U R
 $ } S D ; U R N U
 $ } S D ; U R N
 $ } S D ; U
 $ } S D ; i
 $ } S D ;
 $ } S D
 $ } S D ; I T
 $ } S
 $ } S 1
 $ } S ; 1
 $ } S ; E
 $ } S ; U R Q
 $ } S ; U R Q M U
 $ } S ; U R Q M i
 $ } S ; U R
 $ } S ; U R N U
 $ } S ; U R N i
 $ } S ; U
 $ } S ; i
 $ } S ;
 $ } S B
 $ } S } S D {
 $ } S 1 ) E ( f
 $ } S 1 ) E (
 $ } S 1 ) E
 $ } S 1 ) U R Q
 $ } S 1 ) U R Q M U
 $ } S 1 ) U R Q M i
 $ } S 1 ) U R Q M
 $ } S 1 ) U R
 $ } S 1 ) U R N U
 $ } S 1 ) U R N i
 $ } S 1 ) U R N
 $ } S 1 ) U
 $ } S 1 ) i
 $ } S 1 )
 $ } S 1
 $ } S ; 1
 $ } S ; E
 $ } S ; U R Q
 $ } S ; U R Q M U
 $ } S ; U R Q M i
 $ } S ; U R
 $ } S ; U R N U
 $ } S ; U R N i
 $ } S ; U
 $ } S ; i
 $ } S ;
 $ } S B
 $ } S } S D {
 $ } S } S D
 $ } S } S D ; I T
 $ } S } S D ; I T Q M
 $ } S } S ; U R
 $ } S } S ; U R N U
 $ } S } S ; U R N i
 $ } S } S ; U R N
 $ } S } S ; U R
 $ } S } S ; U R N U
 $ } S ; U R Q M i
 $ } S ; U R
 $ } S ; U R N U
 $ } S ; U R N i
 $ } S ; U
 $ } S ; i
 $ } S ;
 $ } S B
 $ } S } S D {
 $ } S ; b
 $ } S ; 1 r
 $ } error1
```