**8. Write a C program to implement predictive parser**

**Description**:

Predictive parsing is one of the top-down parsing technique. In this parsing the parse table is used with input. The parse table doesn’t contain multiple entries. So the grammar which is acceptable by this parser is LL(1) grammar. There is no backtracking in predictive parsing.

**Algorithm:**

**Algorithm for function FIRST:**

1.if x is terminal then FIRST(x) is {x}.

2.if x is a non terminal and x ->a a(alpha) is a production then add a to FIRST(x). If x->e(epsilon) is a production then add e to FIRST(x).

3. If x->Y1Y2Y3…..Yk is a production, then for all I such that all of Y1,Y2,….Yi-1 are non terminals and FIRST(Yj) contains e for j=1,2…i-1(i.e Y1,Y2,…Yi=1\*=>E).add every non e-symbol in FIRST(Yi) to FIRST(x). If e is in FIRST(Yj) for all j=1,2…k, then add e to FIRST(x).

**Algorithm for function FOLLOW:**

1.$ IS IN follow(s), where S is the start symbol.

2. If there is a production A->a(alpha)Bb(beta),b not equal e, then everything in FIRST(b) but e is in FOLLOW(B).

3. If there is a production A->aB, or a production A->aBb where FIRST(B) contains e(i.e.b\*=>e), then everything in FOLLOW(A) is in FOLLOW(B).

**Algorithm for CONSTABLE(predictive parsing table):**

1. repeat
2. begin

let X be the top stack symbol and if a the next input symbol

3. if X is a terminal or $ then

4.if X=a then

Pop **X** from the stack and remove a from the input

1. else

ERROR()

1. else/\* X is a nonterminal\*/
2. if M[X,a]=X->Y1,Y2,……,Yk then
3. begin

pop X from the stack

push Yk,Yk-1,………..Y1 on to the stack,Y1 on top

1. end
2. else

ERROR()

1. end/\*first begin end\*/
2. until

X=$/\* stack has emptied\*/

**Algorithm for function PARSE:**

1. If X=a=$, the parser halts and announces successful completion of parsing.
2. If X=a=$, the parser pops the X off the stack and advances the input pointer to the next input symbol.

3. If X is a non terminal, the program consults entry M[X, a] of the parsing table M. This entry will be either an X production of the grammar or an error entry. If M[X, a]={X->UVW}, the parser replaces X on top of the stack by WVU(with U on top). As output, the grammar does the symmetric action associated with this production, which, for the time being, we shall assume is just printing the production used. If M[X, a]=error, the parser calls an error recovery routine.

**Program:**

/\*program to implement PREDICTIVE PARSER \*/

#include<stdio.h>

#include<stdlib.h>

int stack[20],top=-1;

void push(int item)

{

if(top>=20)

{

printf("STACK OVERFLOW");

exit(1);

}

stack[++top]=item;

}

int pop()

{

int ch;

if(top<=-1)

{

printf("underflow");

exit(1);

}

ch=stack[top--];

return ch;

}

char convert(int item)

{

char ch;

switch(item)

{

case 0:return('E');

case 1:return('e');

case 2:return('T');

case 3:return('t');

case 4:return('F');

case 5:return('i');

case 6:return('+');

case 7:return('\*');

case 8:return('(');

case 9:return(')');

case 10:return('$');

}

}

void main()

{

int m[10][10],i,j,k;

char ips[20];

int ip[10],a,b,t;

m[0][0]=m[0][3]=21;

m[1][1]=621;

m[1][4]=m[1][5]=-2;

m[2][0]=m[2][3]=43;

m[3][1]=m[3][4]=m[3][5]=-2;

m[3][2]=743;

m[4][0]=5;

m[4][3]=809;

printf("\n Given CFG: \n E->E+T/T \n T->T\*F/F \n F->(E)/i\n Enter the input string Ending with $(Ex:i+i$,i+(i+i)$ etc..):");

scanf("%s",ips);

for(i=0;ips[i];i++)

{

switch(ips[i])

{

case 'E':k=0;break;

case 'e':k=1;break;

case 'T':k=2;break;

case 't':k=3;break;

case 'F':k=4;break;

case 'i':k=5;break;

case '+':k=6;break;

case '\*':k=7;break;

case '(':k=8;break;

case ')':k=9;break;

case '$':k=10;break;

}

ip[i]=k;

}

ip[i]=-1;

push(10);

push(0);

i=0;

printf("\tstack\t\t input \n");

while(1)

{

printf("\t\t");

for(j=0;j<=top;j++)

printf("%c",convert(stack[j]));

printf("\t\t");

for(k=i;ip[k]!=-1;k++)

printf("%c",convert(ip[k]));

printf("\n");

if(stack[top]==ip[i])

{

if(ip[i]==10)

{

printf("\t\t SUCCESS");

return;

}

else

{

top--;

i++;

}

}

else if(stack[top]<=4&&stack[top]>=0)

{

a=stack[top];

b=ip[i]-5;

t=m[a][b];

top--;

while(t>0)

{

push(t%10);

t=t/10;

}

}

else

{

printf("ERROR");

return;

}

}

}

**OUTPUT:**

Given CFG:

E->E+T/T

T->T\*F/F

F->(E)/i

Enter the input string Ending with $(Ex:i+i$,i+(i+i)$ etc..):i+(i\*i)$

stack input

$E i+(i\*i)$

$eT i+(i\*i)$

$etF i+(i\*i)$

$eti i+(i\*i)$

$et +(i\*i)$

$e +(i\*i)$

$eT+ +(i\*i)$

$eT (i\*i)$

$etF (i\*i)$

$et)E( (i\*i)$

$et)E i\*i)$

$et)eT i\*i)$

$et)etF i\*i)$

$et)eti i\*i)$

$et)et \*i)$

$et)etF\* \*i)$

$et)etF i)$

$et)eti i)$

$et)et )$

$et)e )$

$et) )$

$et $

$e $

$ $

SUCCESS