**Shift Reduce Parser OR Bottom-Up Parser**

//Including Libraries

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

//Global Variables

int z = 0, i = 0, j = 0, c = 0;

// Modify array size to increase

// length of string to be parsed

char a[16], ac[20], stk[15], act[10];

// This Function will check whether

// the stack contain a production rule

// which is to be Reduce.

// Rules can be E->2E2 , E->3E3 , E->4

void check()

{

// Copying string to be printed as action

strcpy(ac,"REDUCE TO E -> ");

// c=length of input string

for(z = 0; z < c; z++)

{

//checking for producing rule E->4

if(stk[z] == '4')

{

printf("%s4", ac);

stk[z] = 'E';

stk[z + 1] = '\0';

//printing action

printf("\n$%s\t%s$\t", stk, a);

}

}

for(z = 0; z < c - 2; z++)

{

//checking for another production

if(stk[z] == '2' && stk[z + 1] == 'E' &&

stk[z + 2] == '2')

{

printf("%s2E2", ac);

stk[z] = 'E';

stk[z + 1] = '\0';

stk[z + 2] = '\0';

printf("\n$%s\t%s$\t", stk, a);

i = i - 2;

}

}

for(z=0; z<c-2; z++)

{

//checking for E->3E3

if(stk[z] == '3' && stk[z + 1] == 'E' &&

stk[z + 2] == '3')

{

printf("%s3E3", ac);

stk[z]='E';

stk[z + 1]='\0';

stk[z + 1]='\0';

printf("\n$%s\t%s$\t", stk, a);

i = i - 2;

}

}

return ; //return to main

}

//Driver Function

int main()

{

printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");

// a is input string

strcpy(a,"32423");

// strlen(a) will return the length of a to c

c=strlen(a);

// "SHIFT" is copied to act to be printed

strcpy(act,"SHIFT");

// This will print Labels (column name)

printf("\nstack \t input \t action");

// This will print the initial

// values of stack and input

printf("\n$\t%s$\t", a);

// This will Run upto length of input string

for(i = 0; j < c; i++, j++)

{

// Printing action

printf("%s", act);

// Pushing into stack

stk[i] = a[j];

stk[i + 1] = '\0';

// Moving the pointer

a[j]=' ';

// Printing action

printf("\n$%s\t%s$\t", stk, a);

// Call check function ..which will

// check the stack whether its contain

// any production or not

check();

}

// Rechecking last time if contain

// any valid production then it will

// replace otherwise invalid

check();

// if top of the stack is E(starting symbol)

// then it will accept the input

if(stk[0] == 'E' && stk[1] == '\0')

printf("Accept\n");

else //else reject

printf("Reject\n");

}

**Recursive Descent Parser:**

**Example:**

**Grammar: E --> i E'**

**E' --> + i E' | e**

int main()

{

// E is a start symbol.

E();

// if lookahead = $, it represents the end of the string

// Here l is lookahead.

if (l == '$')

printf("Parsing Successful");

}

// Definition of E, as per the given production

E()

{

if (l == 'i') {

match('i');

E'();

}

}

// Definition of E' as per the given production

E'()

{

if (l == '+') {

match('+');

match('i');

E'();

}//The second condition of E'

else if ( l == 'e' )

{

match('e');

}

return ();

}

// Match function

match(char t)

{

if (l == t) {

l = getchar();

}

else

printf("Error");

}

**First and Follow Sets**

**Example :**

**Input :**

**E -> TR**

**R -> +T R| #**

**T -> F Y**

**Y -> \*F Y | #**

**F -> (E) | i**

**Output :**

**First(E)= { (, i, }**

**First(R)= { +, #, }**

**First(T)= { (, i, }**

**First(Y)= { \*, #, }**

**First(F)= { (, i, }**

**-----------------------------------------------**

**Follow(E) = { $, ), }**

**Follow(R) = { $, ), }**

**Follow(T) = { +, $, ), }**

**Follow(Y) = { +, $, ), }**

**Follow(F) = { \*, +, $, ), }**

// C program to calculate the First and

// Follow sets of a given grammar

#include<stdio.h>

#include<ctype.h>

#include<string.h>

// Functions to calculate Follow

void followfirst(char, int, int);

void follow(char c);

// Function to calculate First

void findfirst(char, int, int);

int count, n = 0;

// Stores the final result

// of the First Sets

char calc\_first[10][100];

// Stores the final result

// of the Follow Sets

char calc\_follow[10][100];

int m = 0;

// Stores the production rules

char production[10][10];

char f[10], first[10];

int k;

char ck;

int e;

int main(int argc, char \*\*argv)

{

int jm = 0;

int km = 0;

int i, choice;

char c, ch;

count = 8;

// The Input grammar

strcpy(production[0], "E=TR");

strcpy(production[1], "R=+TR");

strcpy(production[2], "R=#");

strcpy(production[3], "T=FY");

strcpy(production[4], "Y=\*FY");

strcpy(production[5], "Y=#");

strcpy(production[6], "F=(E)");

strcpy(production[7], "F=i");

int kay;

char done[count];

int ptr = -1;

// Initializing the calc\_first array

for(k = 0; k < count; k++) {

for(kay = 0; kay < 100; kay++) {

calc\_first[k][kay] = '!';

}

}

int point1 = 0, point2, xxx;

for(k = 0; k < count; k++)

{

c = production[k][0];

point2 = 0;

xxx = 0;

// Checking if First of c has

// already been calculated

for(kay = 0; kay <= ptr; kay++)

if(c == done[kay])

xxx = 1;

if (xxx == 1)

continue;

// Function call

findfirst(c, 0, 0);

ptr += 1;

// Adding c to the calculated list

done[ptr] = c;

printf("\n First(%c) = { ", c);

calc\_first[point1][point2++] = c;

// Printing the First Sets of the grammar

for(i = 0 + jm; i < n; i++) {

int lark = 0, chk = 0;

for(lark = 0; lark < point2; lark++) {

if (first[i] == calc\_first[point1][lark])

{

chk = 1;

break;

}

}

if(chk == 0)

{

printf("%c, ", first[i]);

calc\_first[point1][point2++] = first[i];

}

}

printf("}\n");

jm = n;

point1++;

}

printf("\n");

printf("-----------------------------------------------\n\n");

char donee[count];

ptr = -1;

// Initializing the calc\_follow array

for(k = 0; k < count; k++) {

for(kay = 0; kay < 100; kay++) {

calc\_follow[k][kay] = '!';

}

}

point1 = 0;

int land = 0;

for(e = 0; e < count; e++)

{

ck = production[e][0];

point2 = 0;

xxx = 0;

// Checking if Follow of ck

// has already been calculated

for(kay = 0; kay <= ptr; kay++)

if(ck == donee[kay])

xxx = 1;

if (xxx == 1)

continue;

land += 1;

// Function call

follow(ck);

ptr += 1;

// Adding ck to the calculated list

donee[ptr] = ck;

printf(" Follow(%c) = { ", ck);

calc\_follow[point1][point2++] = ck;

// Printing the Follow Sets of the grammar

for(i = 0 + km; i < m; i++) {

int lark = 0, chk = 0;

for(lark = 0; lark < point2; lark++)

{

if (f[i] == calc\_follow[point1][lark])

{

chk = 1;

break;

}

}

if(chk == 0)

{

printf("%c, ", f[i]);

calc\_follow[point1][point2++] = f[i];

}

}

printf(" }\n\n");

km = m;

point1++;

}

}

void follow(char c)

{

int i, j;

// Adding "$" to the follow

// set of the start symbol

if(production[0][0] == c) {

f[m++] = '$';

}

for(i = 0; i < 10; i++)

{

for(j = 2;j < 10; j++)

{

if(production[i][j] == c)

{

if(production[i][j+1] != '\0')

{

// Calculate the first of the next

// Non-Terminal in the production

followfirst(production[i][j+1], i, (j+2));

}

if(production[i][j+1]=='\0' && c!=production[i][0])

{

// Calculate the follow of the Non-Terminal

// in the L.H.S. of the production

follow(production[i][0]);

}

}

}

}

}

void findfirst(char c, int q1, int q2)

{

int j;

// The case where we

// encounter a Terminal

if(!(isupper(c))) {

first[n++] = c;

}

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

{

if(production[j][0] == c)

{

if(production[j][2] == '#')

{

if(production[q1][q2] == '\0')

first[n++] = '#';

else if(production[q1][q2] != '\0'

&& (q1 != 0 || q2 != 0))

{

// Recursion to calculate First of New

// Non-Terminal we encounter after epsilon

findfirst(production[q1][q2], q1, (q2+1));

}

else

first[n++] = '#';

}

else if(!isupper(production[j][2]))

{

first[n++] = production[j][2];

}

else

{

// Recursion to calculate First of

// New Non-Terminal we encounter

// at the beginning

findfirst(production[j][2], j, 3);

}

}

}

}

void followfirst(char c, int c1, int c2)

{

int k;

// The case where we encounter

// a Terminal

if(!(isupper(c)))

f[m++] = c;

else

{

int i = 0, j = 1;

for(i = 0; i < count; i++)

{

if(calc\_first[i][0] == c)

break;

}

//Including the First set of the

// Non-Terminal in the Follow of

// the original query

while(calc\_first[i][j] != '!')

{

if(calc\_first[i][j] != '#')

{

f[m++] = calc\_first[i][j];

}

else

{

if(production[c1][c2] == '\0')

{

// Case where we reach the

// end of a production

follow(production[c1][0]);

}

else

{

// Recursion to the next symbol

// in case we encounter a "#"

followfirst(production[c1][c2], c1, c2+1);

}

}

j++;

}

}

}

**Finding and Removing Left Recursion**

Left Recursion:

Consider,

E->E+T

E=a

T=b

In it's parse tree E will grow left indefinitely, so to remove it

E=Ea | b

we take as

E=bE'

E'= aE'|E

**Program:**

**#include<stdio.h>**

**2: #include<string.h>**

**3: #define SIZE 10**

**4: int main () {**

**5: char non\_terminal;**

**6: char beta,alpha;**

**7: int num;**

**8: char production[10][SIZE];**

**9: int index=3; /\* starting of the string following "->" \*/**

**10: printf("Enter Number of Production : ");**

**11: scanf("%d",&num);**

**12: printf("Enter the grammar as E->E-A :\n");**

**13: for(int i=0;i<num;i++){**

**14: scanf("%s",production[i]);**

**15: }**

**16: for(int i=0;i<num;i++){**

**17: printf("\nGRAMMAR : : : %s",production[i]);**

**18: non\_terminal=production[i][0];**

**19: if(non\_terminal==production[i][index]) {**

**20: alpha=production[i][index+1];**

**21: printf(" is left recursive.\n");**

**22: while(production[i][index]!=0 && production[i][index]!='|')**

**23: index++;**

**24: if(production[i][index]!=0) {**

**25: beta=production[i][index+1];**

**26: printf("Grammar without left recursion:\n");**

**27: printf("%c->%c%c\'",non\_terminal,beta,non\_terminal);**

**28: printf("\n%c\'->%c%c\'|E\n",non\_terminal,alpha,non\_terminal);**

**29: }**

**30: else**

**31: printf(" can't be reduced\n");**

**32: }**

**33: else**

**34: printf(" is not left recursive.\n");**

**35: index=3;**

**36: }**

**37: }**

**Removing Direct and Indirect Recursion:**

| #include<stdio.h> |  |
| --- | --- |
|  | #include<string.h> |
|  | void findNonRec(char str[]){ |
|  | int i,j=0,k=0,len,l; |
|  | char alpha[20],beta[20]; |
|  | len=strlen(str); |
|  | l=str[0]; |
|  | for(i=3;i<len;i++){ |
|  | if(str[i]==l){ |
|  | int c=i+1; |
|  | while(str[c]!='|' && str[c]!='\0'){ |
|  | alpha[j]=str[c]; |
|  | j++; |
|  | c++; |
|  | } |
|  | alpha[j]='|'; |
|  | j++; |
|  | i=c; |
|  | alpha[j]='\0'; |
|  | } |
|  | else if(str[i]!=l){ |
|  | int n=i; |
|  | while(str[n]!='\0' && str[n]!='|'){ |
|  | beta[k++]=str[n]; |
|  | n++; |
|  | } |
|  | beta[k++]='|'; |
|  | i=n; |
|  | beta[k]='\0'; |
|  | } |
|  | } |
|  | printf("------Grammar after removing left recursion is----- \n"); |
|  | printf("%c->",l); |
|  | for(i=0;beta[i]!='\0';i++){ |
|  | if(beta[i]!='|') |
|  | printf("%c",beta[i]); |
|  | else |
|  | printf("%c'|",l); |
|  | } |
|  | printf("\n"); |
|  | printf("%c'->",l); |
|  | for(i=0;alpha[i]!='\0';i++){ |
|  | while(alpha[i]!='|'){ |
|  | printf("%c",alpha[i]); |
|  | i++; |
|  | } |
|  | printf("%c'|",l); |
|  | } |
|  | printf("0"); |
|  | } |
|  | int main(){ |
|  | int n,count=0,i,j,m=0,l=0,k; |
|  | char a[10],b[10],temp1[20],temp2[20]; |
|  | printf("Enter no of production"); |
|  | scanf("%d",&n); |
|  | if(n>1){ |
|  | printf("Enter %d productions\n",n); |
|  | scanf("%s",a); |
|  | scanf("%s",b); |
|  | int len1=strlen(a); |
|  | int len2=strlen(b); |
|  | char l1=a[0]; |
|  | char l2=b[0]; |
|  | temp2[l]=a[0]; |
|  | l++; |
|  | temp2[l]=a[1]; |
|  | l++; |
|  | temp2[l]=a[2]; |
|  | l++; |
|  | for(i=3;i<len1;i++){ |
|  | if(a[i]==l2){ |
|  | int n=i+1; |
|  | while(a[n]!='|' && a[n]!='\0'){ |
|  | temp1[m]=a[n]; |
|  | m++; |
|  | n++; |
|  | } |
|  | temp1[m]='\0'; |
|  | //printf("temp%s \n",temp2); |
|  | for(j=3;j<len2;j++){ |
|  | while(b[j]!='\0' && b[j]!='|'){ |
|  | temp2[l]=b[j]; |
|  |  |
|  | l++; |
|  | j++; |
|  | } |
|  | for(k=0;k<strlen(temp1);k++){ |
|  | temp2[l]=temp1[k]; |
|  | l++; |
|  | } |
|  | temp2[l]='|'; |
|  | l++; |
|  | } |
|  | i=n; |
|  | } |
|  | else if(a[i]!=l2){ |
|  | temp2[l]=a[i]; |
|  | l++; |
|  | } |
|  | } |
|  | temp2[l]='\0'; |
|  | findNonRec(temp2); |
|  | } |
|  | else{ |
|  | printf("Enter production"); |
|  | scanf("%s",a); |
|  | findNonRec(a); |
|  | } |
|  | return 0; |
|  | } |