

Compiler Design

Assignment – 6: Implement Predictive LL(1) Parser

 Points to be taken care of while designing Predictive LL(1) parser are as below :

- 1) Elimination of Left Recursion and Left Factoring if any
- 2) Find First() and Follow() of each non-terminal
- 3) Construct LL(1) parse table
- 4) Check whether the grammar is LL(1) or not
- 5) Check whether the given string is accepted by the grammar using LL(1) parse table

CODE:

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

void followfirst(char , int , int);
void findfirst(char , int , int);
void follow(char c);

int count,n=0;
char calc_first[10][100];
char calc_follow[10][100];
int m=0;
char production[10][10], first[10];
char f[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;
printf("How many productions ? :");
scanf("%d",&count);
printf("\nEnter %d productions in form A=B where A and B are
grammar symbols :\n\n",count);
for(i=0;i<count;i++)
{
    scanf("%s%c",production[i],&ch);
}
int kay;
char done[count];
int ptr = -1;
for(k=0;k<count;k++){
    for(kay=0;kay<100;kay++){
        calc_first[k][kay] = '!';
    }
}
int point1 = 0,point2,temp;
for(k=0;k<count;k++)
{
    c=production[k][0];
    point2 = 0;
    temp = 0;
    for(kay = 0; kay <= ptr; kay++)
        if(c == done[kay])
            temp = 1;
    if (temp == 1)
        continue;
    findfirst(c,0,0);
    ptr+=1;
    done[ptr] = c;
    printf("\n First(%c)= { ",c);
```

```

    calc_first[point1][point2++] = c;
    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;
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;

```

```

temp = 0;
for(kay = 0; kay <= ptr; kay++)
    if(ck == donee[kay])
        temp = 1;
if (temp == 1)
    continue;
land += 1;
follow(ck);
ptr+=1;
donee[ptr] = ck;
printf(" Follow(%c) = { ",ck);
calc_follow[point1][point2++] = ck;
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++;
}
char ter[10];
for(k=0;k<10;k++){
    ter[k] = '!';
}

```

```
int ap,vp,result = 0;
for(k=0;k<count;k++){
    for(kay=0;kay<count;kay++){
        if(!isupper(production[k][kay]) &&
production[k][kay] != '#' && production[k][kay] != '=' &&
production[k][kay] != '\0'){
            vp = 0;
            for(ap = 0;ap < result; ap++){
                if(production[k][kay] == ter[ap]){
                    vp = 1;
                    break;
                }
            }
            if(vp == 0){
                ter[result] = production[k][kay];
                result ++;
            }
        }
    }
}
ter[result] = '$';
result++;
printf("\n\t\t\t\t\t The LL(1) Parsing Table for the above
grammer :-");
printf("\n\t\t\t\t\t ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^\n");
printf("\n\t\t\t\t=====
=====
=====\\n");
printf("\\t\\t\\t|\\t");
for(ap = 0;ap < result; ap++){
    printf("%c\\t\\t",ter[ap]);
}
```

```
printf("\n\t\t\t=====
=====
=====\\n");
char first_prod[count][result];
for(ap=0;ap<count;ap++){
    int destiny = 0;
    k = 2;
    int ct = 0;
    char tem[100];
    while(production[ap][k] != '\\0'){
        if(!isupper(production[ap][k])){
            tem[ct++] = production[ap][k];
            tem[ct++] = '_';
            tem[ct++] = '\\0';
            k++;
            break;
        }
        else{
            int zap=0;
            int tuna = 0;
            for(zap=0;zap<count;zap++){
                if(calc_first[zap][0] == production[ap][k]){
                    for(tuna=1;tuna<100;tuna++){
                        if(calc_first[zap][tuna] != '!'){
                            tem[ct++] =
calc_first[zap][tuna];
                                }
                            else
                                break;
                        }
                    }
                    break;
                }
            }
        }
    }
}
```

```

        tem[ct++] = '_';
    }
    k++;
}
int zap = 0,tuna;
for(tuna = 0;tuna<ct;tuna++){
    if(tem[tuna] == '#'){
        zap = 1;
    }
    else if(tem[tuna] == '_'){
        if(zap == 1){
            zap = 0;
        }
        else
            break;
    }
    else{
        first_prod[ap][destiny++] = tem[tuna];
    }
}
}
char table[land][result+1];
ptr = -1;
for(ap = 0; ap < land ; ap++){
    for(kay = 0; kay < (result + 1) ; kay++){
        table[ap][kay] = '!';
    }
}
for(ap = 0; ap < count ; ap++){
    ck = production[ap][0];
    temp = 0;
    for(kay = 0; kay <= ptr; kay++)
        if(ck == table[kay][0])

```

```
        temp = 1;
    if (temp == 1)
        continue;
    else{
        ptr = ptr + 1;
        table[ptr][0] = ck;
    }
}
for(ap = 0; ap < count ; ap++){
    int tuna = 0;
    while(first_prod[ap][tuna] != '\0'){
        int to,ni=0;
        for(to=0;to<result;to++){
            if(first_prod[ap][tuna] == ter[to]){
                ni = 1;
            }
        }
        if(ni == 1){
            char xz = production[ap][0];
            int cz=0;
            while(table[cz][0] != xz){
                cz = cz + 1;
            }
            int vz=0;
            while(ter[vz] != first_prod[ap][tuna]){
                vz = vz + 1;
            }
            table[cz][vz+1] = (char)(ap + 65);
        }
        tuna++;
    }
}
for(k=0;k<result;k++){
```



```

    for(kay=0;kay<100;kay++){
        if(calc_first[k][kay] == '!'){
            break;
        }
        else if(calc_first[k][kay] == '#'){
            int fz = 1;
            while(calc_follow[k][fz] != '!'){
                char xz = production[k][0];
                int cz=0;
                while(table[cz][0] != xz){
                    cz = cz + 1;
                }
                int vz=0;
                while(ter[vz] != calc_follow[k][fz]){
                    vz = vz + 1;
                }
                table[k][vz+1] = '#';
                fz++;
            }
            break;
        }
    }
}

for(ap = 0; ap < land ; ap++){
    printf("\t\t\t %c\t\t",table[ap][0]);
    for(kay = 1; kay < (result + 1) ; kay++){
        if(table[ap][kay] == '!')
            printf("\t\t");
        else if(table[ap][kay] == '#')
            printf("%c=#\t\t",table[ap][0]);
        else{
            int mum = (int)(table[ap][kay]);
            mum -= 65;

```

```
        printf("%s\t\t",production[mum]);
    }
}
printf("\n");
printf("\t\t\t-----");
                                           );
    printf("\n");
}
int j;
printf("\n\nPlease enter the desired INPUT STRING = ");
char input[100];
scanf("%s%c",input,&ch);
printf("\n\t\t\t\t\t=====
=====\\n");
printf("\\t\\t\\t\\t\\tStack\\t\\tInput\\t\\tAction");
printf("\\n\\t\\t\\t\\t\t=====
=====\\n");
int i_ptr = 0,s_ptr = 1;
char stack[100];
stack[0] = '$';
stack[1] = table[0][0];
while(s_ptr != -1){
    printf("\\t\\t\\t\\t\\t");
    int vamp = 0;
    for(vamp=0;vamp<=s_ptr;vamp++){
        printf("%c",stack[vamp]);
    }
    printf("\\t\\t\\t");
    vamp = i_ptr;
    while(input[vamp] != '\\0'){
        printf("%c",input[vamp]);
        vamp++;
    }
}
```

```

printf("\t\t\t");
char her = input[i_ptr];
char him = stack[s_ptr];
s_ptr--;
if(!isupper(him)){
    if(her == him){
        i_ptr++;
        printf("POP ACTION\n");
    }
    else{
        printf("\nString Not Accepted by LL(1) Parser
!!\n");

        exit(0);
    }
}
else{
    for(i=0;i<result;i++){
        if(ter[i] == her)
            break;
    }
    char produ[100];
    for(j=0;j<land;j++){
        if(him == table[j][0]){
            if (table[j][i+1] == '#'){
                printf("%c=#\n",table[j][0]);
                produ[0] = '#';
                produ[1] = '\0';
            }
            else if(table[j][i+1] != '!'){
                int mum = (int)(table[j][i+1]);
                mum -= 65;
                strcpy(produ,production[mum]);
                printf("%s\n",produ);
            }
        }
    }
}

```

```

    }
    else{
        printf("\nString Not Accepted by
LL(1) Parser !!\n");

        exit(0);
    }
}

}

int le = strlen(produ);
le = le - 1;
if(le == 0){
    continue;
}
for(j=le;j>=2;j--){
    s_ptr++;
    stack[s_ptr] = produ[j];
}
}

}

printf("\n\t\t\t=====
=====
=====\\n");

if (input[i_ptr] == '\\0'){
    printf("\\t\\t\\t\\t\\t\\t\\tYOUR STRING HAS BEEN ACCEPTED
!!\\n");
}
else
    printf("\\n\\t\\t\\t\\t\\t\\t\\tYOUR STRING HAS BEEN REJECTED
!!\\n");

printf("\\t\\t\\t=====
=====
=====\\n");
}

```

```
void follow(char c)
{
    int i ,j;
    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'){
                    followfirst(production[i][j+1],i,(j+2));
                }
                if(production[i][j+1]=='\0'&& c!=production[i][0]){
                    follow(production[i][0]);
                }
            }
        }
    }
}

void findfirst(char c ,int q1 , int q2)
{
    int j;
    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))
            {
                findfirst(production[q1][q2], q1, (q2+1));
            }
            else
                first[n++]='#';
        }
        else if(!isupper(production[j][2])){
            first[n++] = production[j][2];
        }
        else {
            findfirst(production[j][2], j, 3);
        }
    }
}

```

```

void followfirst(char c, int c1 , int c2)
{
    int k;
    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;
        }
    }
}

```

```
    }  
    while(calc_first[i][j] != '!')  
    {  
        if(calc_first[i][j] != '#'){  
            f[m++] = calc_first[i][j];  
        }  
        else{  
            if(production[c1][c2] == '\\0'){  
                follow(production[c1][0]);  
            }  
            else{  
                followfirst(production[c1][c2],c1,c2+1);  
            }  
        }  
        j++;  
    }  
}
```

Output:

Select C:\Users\Arjun Vankani\Desktop\CE SEM 7\ASS\CD\Lab6\LL(1).exe

How many productions ? : 8

Enter 8 productions in form A=B where A and B are grammar symbols :

E=TR

R=+TR

R=#

T=FY

Y=*FY

Y=#

F=(E)

F=i

First(E) = { (, i, }

First(R) = { +, #, }

First(T) = { (, i, }

First(Y) = { *, #, }

First(F) = { (, i, }

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

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

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

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

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

The LL(1) Parsing Table for the above grammar :-

	\$	+	*	()	i
E				E=TR		E=TR
R		R=+TR				R=#
T				T=FY		T=FY
Y		Y=#	Y=*FY			Y=#
F				F=(E)		F=i

Please enter the desired INPUT STRING = _

Please enter the desired INPUT STRING = i+i*i\$

Stack	Input	Action
\$E	i+i*i\$	E=TR
\$RT	i+i*i\$	T=FY
\$RYF	i+i*i\$	F=i
\$RYi	i+i*i\$	POP ACTION
\$RY	+i*i\$	Y=#
\$R	+i*i\$	R=+TR
\$RT+	+i*i\$	POP ACTION
\$RT	i*i\$	T=FY
\$RYF	i*i\$	F=i
\$RYi	i*i\$	POP ACTION
\$RY	*i\$	Y=*FY
\$RYF*	*i\$	POP ACTION
\$RYF	i\$	F=i
\$RYi	i\$	POP ACTION
\$RY	\$	Y=#
\$R	\$	R=#
\$	\$	POP ACTION

YOUR STRING HAS BEEN ACCEPTED !!

Process exited after 110 seconds with return value 0

Press any key to continue . . .