VNR Vignana Jyothi Institute of Engineering and Technology
(Affiliated to J.N.T.U, Hyderabad)
Bachupally (v), Hyderabad, Telangana, India.

Implementatio LL(1) Parser

A course project submitted in complete requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

Ch.Lokendra Shiva Kumar (19071A05J8)
E.Shiva Sai(19071A05K1)
Parthiv Jasti(18071A05P3)
S.Sai Sathwik (19071A05Q2)
V.Sai Kowshil (19071A05R0)
S.Gaurav(2007580523)

Under the guidance of

Mrs. G.Laxmi Deepthi
Assistant Professor
CSE

VNR Vignana Jyothi Institute of Engineering and Technology
(Affiliated to J.N.T.U, Hyderabad)
Bachupally(v), Hyderabad, Telangana, India.

<u>CERTIFICATE</u>

This is to certify that **Ch. Lokendra Shiva Kumar(19071A05J8), E. Shiva Sai (19071A05K1), Parthiv Jasti (19071A05P3), S. Sai Sathwik (19071A05Q2), V. Sai Kowshil (19071A05R0), S. Gaurav (2007580523)** have completed their course project work at CSE Department of VNR VJIET, Hyderabad entitled **"Implementation of LL(1) Parser"** in complete fulfilment of the requirements for the award of B.Tech degree during the academic year 2021-2022. This work is carried out under my supervision and has not been submitted to any other University/Institute for award of any degree/diploma.

Mrs. G.Laxmi Deepthi	**Dr . S. Nagini**
Assistant Professor	Head of the Department
CSE Department	CSE Department
VNRVJIET	VNRVJIET

DECLARATION

This is to certify that our project report titled **"Implementation of LL(1) Parser"** submitted to Vallurupalli Nageswara Rao Institute of Engineering and Technology in complete fulfilment of requirement for the award of Bachelor of Technology in Computer Science and Engineering is a bona fide report to the work carried out by us under the guidance and supervision of Mrs. G.Laxmi Deepthi, Assistant Professor, Department of Computer Science and Engineering, Vallurupalli Nageswara Rao Institute of Engineering and Technology. To the best of our knowledge, this has not been submitted in any form to other university or institution for the award of any degree or diploma.

CH. Lokendra Shiva Kumar
(19071A05J8)
CSE 4

E. Shiva Sai
(19071A05K1)
CSE 4

Parthiv J
(19071A05P3)
CSE 4

S. Sai Sathwik
(19071A05Q2)
CSE 4

V. Sai Kowshil Reddy
(19071A05R0)
CSE 4

S. Gaurav
(2007580523)
CSE 4

ACKNOWLEDGEMENT

Over a span of three and a half years, VNRVJIET has helped us transform ourselves from mere amateurs in the field of Computer Science into skilled engineers capable of handling any given situation in real time. We are highly indebted to the institute for everything that it has given us. We would like to express our gratitude towards the principal of our institute, **Dr. Challa Dhanunjaya Naidu** and the Head of the Computer Science & Engineering Department, **Dr.S.Nagini** for their kind co-operation and encouragement which helped us complete the project in the stipulated time. Although we have spent a lot of time and put in a lot effort into this project, it would not have been possible without the motivating support and help of our project guide **Mrs. G. Laxmi Deepthi.** We thank her for her guidance, constant supervision and for providing necessary information to complete this project. Our thanks and appreciations also go to all the faculty members, staff members of VNRVJIET, and all our friends who have helped us put this project together.

INDEX

ABSTRACT

LL(1) parser is a top-down parser that uses a one-token lookahead .The first L indicates that the input is read from left to right. The second L says that it produces a left-to-right derivation. And the L says that it uses one lookahead token. LL(1) parsing is arguably the simplest form of context-free parsing. It parses a document top-down and creates a left-derivation tree from the resulting input. The chief advantage it has over recursive-descent parsing is that it's slightly easier for a machine to generate an LL(1) table than it is to produce recursive functions. In practice, it is generally faster to use tables than a bunch of recursive methods.

1. Introduction

Algorithm to construct LL(1) Parsing Table:

Step 1: First check for left recursion in the grammar.
A production of grammar is said to have left recursion if the leftmost variable of RHS is same as the variable of its LHS. Eg:- A -> A a|b
A grammar containing a production having left recursion is called left recursive grammar Left recursion is eliminated because top down parsing methods cannot handle left recursive grammar

Step 2: Check for left factoring in the grammar.
A grammar is said to be left recursive if it is of the form A → αβ1 / αβ2 / αβ3/……/αβn/γ

The productions start with the same terminals
When the choice between two alternative A- productions is not clear, we may be able to rewrite the productions to defer the decision until enough of the input has been seen to make the right choice.
During parsing FIRST and FOLLOW helps us to choose which production rule to apply based on the next input symbol. We know that we need backtracking in syntax analysis which is really a complex process to implement. There is an easier way to sort out this problem by using FIRST and FOLLOW.
If the compiler would come to know in advance , that what is the "first character of the string produced when a production is applied ", and comparing it to the current character or token in the input string it sees, it can wisely take decision on which production rule to apply.
Step 3: Calculate First() and Follow() for all non-terminals.

First(): If there is a variable, and from that variable, if we try to drive all the strings then the beginning Terminal Symbol is called the First.

Follow(): What is the Terminal Symbol which follows a variable in the process of derivation.

Step 4: For each production A –> α. (A tends to alpha)
Find First(α) and for each terminal in First(α), make entry A –> α in the table.
If First(α) contains ε (epsilon) as terminal than, find the Follow(A) and for each terminal in Follow(A), make entry A –> α in the table.
If the First(α) contains ε and Follow(A) contains $ as terminal, then make entry A –> α the table for the $.

2. Requirements

SOFTWARE REQUIREMENTS

- Windows 7 or Higher
- GCC Compiler
- Microsoft VS Code Editor

HARDWARE REQUIREMENTS:

- System with minimum 250GB
- Harddisk System with Minimum 2GB RAM.
- Intel i3 processor or above.

3. MODEL IMPLEMENTATION

We have implemented the LL(1) parsing table using C language with ctype and string libraries.

Program:

```
#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,xxx;
        for(k=0;k<count;k++)
        {
```

```
		c=production[k][0];
		point2 = 0;
		xxx = 0;
		for(kay = 0; kay <= ptr; kay++)
			if(c == done[kay])
				xxx = 1;
		if (xxx == 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;
		xxx = 0;
		for(kay = 0; kay <= ptr; kay++)
			if(ck == donee[kay])
				xxx = 1;
		if (xxx == 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,sid = 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 < sid; ap++){
						if(production[k][kay] == ter[ap]){
							vp = 1;
							break;
						}
					}
					if(vp == 0){
						ter[sid] = production[k][kay];
						sid ++;
					}
				}
			}
		}
		ter[sid] = '$';
		sid++;
		printf("\n\t\t\t\t\t\t\t The LL(1) Parsing Table for the above grammer :-");
```

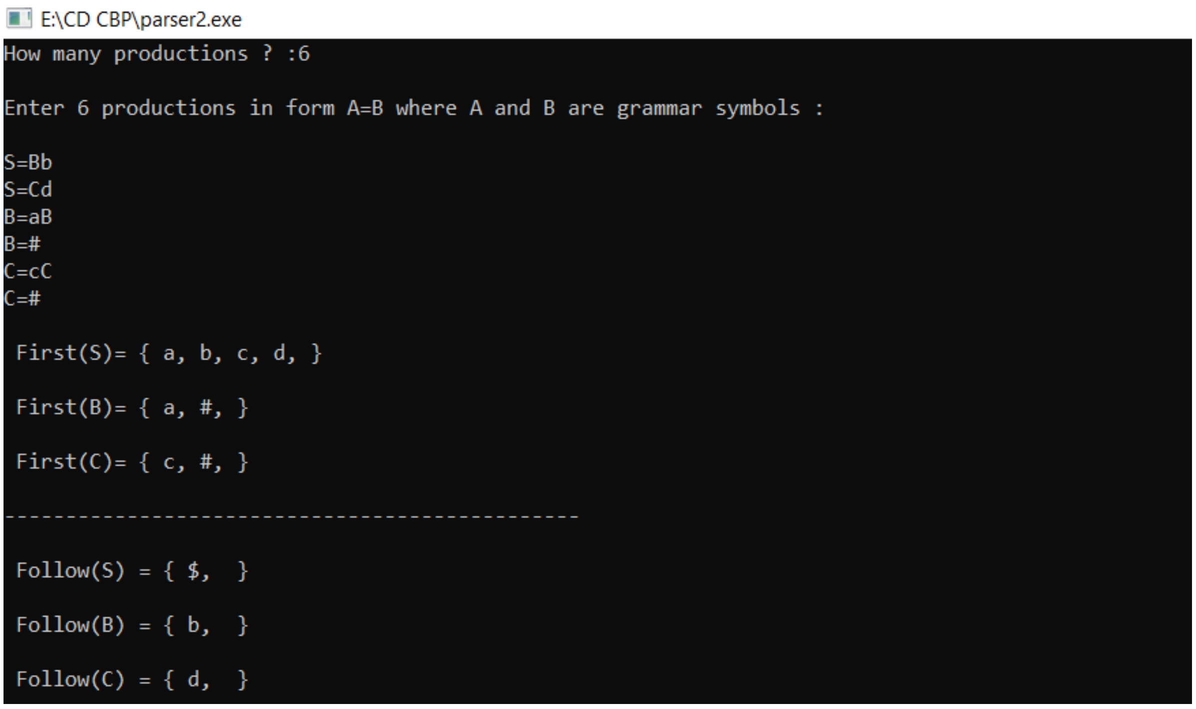
```
    printf("\n\t\t\t\t\t\t\t^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n");

printf("\n\t\t\t==================================================
=========================================================\n");
    printf("\t\t\t\t|\t");
    for(ap = 0;ap < sid; ap++){
        printf("%c\t\t",ter[ap]);
    }

printf("\n\t\t\t==================================================
=========================================================\n");
    char first_prod[count][sid];
    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][sid+1];
ptr = -1;
for(ap = 0; ap < land ; ap++){
        for(kay = 0; kay < (sid + 1) ; kay++){
                table[ap][kay] = '!';
        }
}
for(ap = 0; ap < count ; ap++){
        ck = production[ap][0];
        xxx = 0;
        for(kay = 0; kay <= ptr; kay++)
                if(ck == table[kay][0])
                        xxx = 1;
        if (xxx == 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<sid;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<sid;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 < (sid + 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");
	}
```

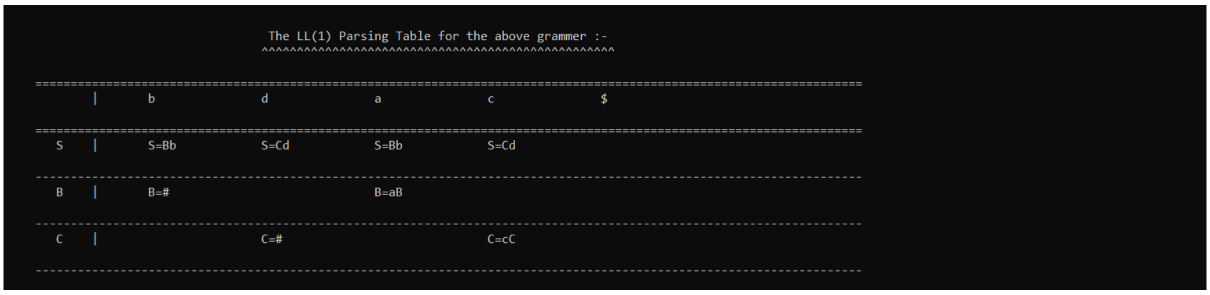
```
}

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++;
                }
        }
}
```

4.Output:



E:\CD CBP\parser2.exe

```
How many productions ? :6

Enter 6 productions in form A=B where A and B are grammar symbols :

S=Bb
S=Cd
B=aB
B=#
C=cC
C=#

 First(S)= { a, b, c, d, }

 First(B)= { a, #, }

 First(C)= { c, #, }

-----------------------------------------------

 Follow(S) = { $,  }

 Follow(B) = { b,  }

 Follow(C) = { d,  }
```



```
                              The LL(1) Parsing Table for the above grammer :-
                             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

=================================================================================================
       |      b             d             a             c             $

=================================================================================================
   S   |      S=Bb          S=Cd          S=Bb          S=Cd

-------------------------------------------------------------------------------------------------
   B   |      B=#                         B=aB

-------------------------------------------------------------------------------------------------
   C   |                    C=#                         C=cC

-------------------------------------------------------------------------------------------------
```

5.CONCLUSION

Thus we used first and follow techniques to construct LL(1) parsing table in C++ using ctype and string libraries.

6.REFERENCES

https://www.geeksforgeeks.org/construction-of-ll1-parsing-table/