BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Empowered Autonomous Institute Affiliated to University of Mumbai)
[Knowledge is Nectar]

Department of Computer Engineering

Course - System Programming and Compiler Construction (SPCC)

UID	2021300108
Name	Hatim Sawai
Class and Batch	TE Computer Engineering - Batch C
Date	26/2/2024
Lab #	3
Aim	Design syntax analyzer for various grammars and implement using different parsing techniques* (Top-down and Bottom-up).
Objective	To implement syntax analysis using various parsing techniques.
Theory	**Syntax Analysis:** Parsing, syntax analysis, or syntactic analysis is the process of analyzing a string of symbols, either in natural language, computer languages or data structures, conforming to the rules of a formal grammar. **Types of parser:** The parser is mainly classified into two categories, i.e. Top-down Parser, and Bottom-up Parser. **Top-Down Parser:** A top-down parser builds the parse tree from the top down, starting with the start non-terminal. There are two types of Top-Down Parsers: 1. Top-Down Parser with Backtracking 2. Top-Down Parsers without Backtracking **LL(1) Parser:** In LL1 Parser, 1st L represents that the scanning of the Input will be done from the Left to Right manner and the second L shows that in this parsing technique, we are going to use the Leftmost Derivation Tree. And finally, the 1 represents the number of look-aheads, which means how many symbols are you going to see when you want to make a decision. **FIRST:** The FIRST set of a non-terminal A is defined as the set of terminals that can appear as the first symbol in any string derived from A. If a non-terminal A can derive the empty string, then the empty string is also included in the FIRST set of A. **Rules to compute FIRST set:** 1. If x is a terminal, then FIRST(x) = { 'x' } 2. If x-> ?, is a production rule, then add ? to FIRST(x).

3. If X->Y1 Y2 Y3….Yn is a production,
   FIRST(X) = FIRST(Y1)
   If FIRST(Y1) contains ? then FIRST(X) = { FIRST(Y1) – ? } U { FIRST(Y2) }
   If FIRST (Yi) contains ? for all i = 1 to n, then add ? to FIRST(X).

FOLLOW:
FOLLOW set in compiler design are used to identify the terminal symbol immediately after a non-terminal in a given language. FOLLOW set is also used to avoid backtracking same as the FIRST set. The only difference is FOLLOW set works on vanishing non-terminal on the right-hand side so that decision-making gets easier for the compiler while parsing.

Rules to compute FOLLOW set:
1. FOLLOW(S) = { $ } // where S is the starting Non-Terminal
2. If A -> pBq is a production, where p, B and q are any grammar symbols, then everything in FIRST(q) except Є is in FOLLOW(B).
3. If A->pB is a production, then everything in FOLLOW(A) is in FOLLOW(B).
4. If A->pBq is a production and FIRST(q) contains Є, then FOLLOW(B) contains { FIRST(q) – Є } U FOLLOW(A)

Essential conditions:
The grammar is free from left recursion.
The grammar should not be ambiguous.
The grammar has to be left factored in so that the grammar is deterministic grammar.

Parse Table construction:
1. First check all the essential conditions mentioned above and go to 2.

2. 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.
3. 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.
4. If First(α) contains ε (epsilon) as terminal, then find the Follow(A) and for each terminal in Follow(A), make entry A –> ε in the table.
5. If the First(α) contains ε and Follow(A) contains $ as terminal, then make entry A –> ε in the table for the $.
6. To construct the parsing table, we have two functions: In the table, rows will contain the Non-Terminals and the column will contain the Terminal Symbols. All the Null Productions of the Grammars will go under the Follow elements and the remaining productions will lie under the elements of the First set.

Advantages of Construction of LL(1) Parsing Table:
1. Efficiency
2. Deterministic Parsing

	LR parsers : It is an efficient bottom-up syntax analysis technique that can be used to parse large classes of context free grammar is called LR(0) parsing. L stands for the left to right scanning R stands for rightmost derivation in reverse 0 stands for no. of input symbols of lookahead **Advantages of LR parsing :** 1. It recognizes virtually all programming language constructs for which CFG can be written 2. It is able to detect syntactic errors 3. It is an efficient non-backtracking shift reducing parsing method. **Types of LR parsing methods :** 1. SLR 2. CLR 3. LALR **SLR Parser :** SLR is simple LR. It is the smallest class of grammar having few number of states. SLR is very easy to construct and is similar to LR parsing. The only difference between SLR parser and LR(0) parser is that in LR(0) parsing table, there's a chance of 'shift reduced' conflict because we are entering 'reduce' corresponding to all terminal states. We can solve this problem by entering 'reduce' corresponding to FOLLOW of LHS of production in the terminating state. This is called SLR(1) collection of items **Steps for constructing the SLR parsing table :** 1. Writing augmented grammar 2. LR(0) collection of items to be found 3. Find FOLLOW of LHS of production 4. Defining 2 functions:goto[list of terminals] and action[list of non-terminals] in the parsing table **What is the limitation of the SLR parser?** One of the key limitations of SLR is that it can only handle grammars that fall under the SLR(1) class, which implies that the grammar must be straightforward and have only one lookahead symbol.

Implementation / Code	**1. LL1 Parser**
	`#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");
    }
    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\t\tStack\t\t\tInput\t\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\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 < sid; 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\t\tYOUR STRING HAS BEEN ACCEPTED !!\n");
    }
    else
        printf("\n\t\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++;
        }
```

```
    }
}
```

2. SLR Parser

```
import java.util.Scanner;

public class LL1Parser {
    static int count, n = 0, m = 0;
    static char[][] calc_first = new char[10][100];
    static char[][] calc_follow = new char[10][100];
    static int k;
    static char ck;
    static int e;
    static char[][] production = new char[10][10];
    static char[] first = new char[10];
    static char[] f = new char[10];

    public static void main(String[] args) {
        int jm = 0;
        int km = 0;
        int i, choice;
        char c, ch;
        Scanner sc = new Scanner(System.in);
        System.out.print("How many productions ? :");
        count = sc.nextInt();
        System.out.printf("\nEnter %d productions in form A=B where A
and B are grammar symbols :\n\n", count);
        for (i = 0; i < count; i++) {
            production[i] = sc.next().toCharArray();
            ch = sc.next().charAt(0);
        }
        int kay;
        char[] done = new char[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;
            System.out.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) {
                    System.out.printf("%c, ", first[i]);
                    calc_first[point1][point2++] = first[i];
                }
            }
            System.out.println("}\n");
            jm = n;
            point1++;
        }
        System.out.println("\n");
```

```
        System.out.println("----------------------------------------
-----\n\n");
        char[] donee = new char[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;
            System.out.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) {
                    System.out.printf("%c, ", f[i]);
```

```
                    calc_follow[point1][point2++] = f[i];
                }
            }
            System.out.println(" }\n\n");
            km = m;
            point1++;
        }
        char[] ter = new char[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 (!Character.isUpperCase(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++;
        System.out.println("\n\t\t\t\t\t\t\t\t The LL(1) Parsing Table
for the above grammar :-");
```

```
System.out.println("\n\t\t\t\t\t\t\t^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^\n");

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

System.out.println("\n\t\t\t==========================================
======================================================================
=====\n");
        char[][] first_prod = new char[count][sid];
        for (ap = 0; ap < count; ap++) {
            int destiny = 0;
            k = 2;
            int ct = 0;
            char[] tem = new char[100];
            while (production[ap][k] != '\0') {
                if (!Character.isUpperCase(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 = new char[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++) {
            System.out.printf("\t\t\t   %c\t|\t", table[ap][0]);
            for (kay = 1; kay < (sid + 1); kay++) {
                if (table[ap][kay] == '!')
                    System.out.printf("\t\t");
                else if (table[ap][kay] == '#')
                    System.out.printf("%c=#\t\t", table[ap][0]);
                else {
                    int mum = (int) (table[ap][kay]);
                    mum -= 65;
                    System.out.printf("%s\t\t", new
String(production[mum]));
                }
            }
            System.out.println();
```

```
            System.out.println("\t\t\t--------------------------------
--------------------------------------------------------------------
---------------");
            System.out.println();
        }
        System.out.print("\n\nPlease enter the desired INPUT STRING =
");
        String input = sc.next();
        char[] inputArr = input.toCharArray();

System.out.println("\n\t\t\t\t\t=====================================
=====================================\n");

System.out.println("\t\t\t\t\tStack\t\t\tInput\t\t\tAction");

System.out.println("\n\t\t\t\t\t=====================================
=====================================\n");
        int i_ptr = 0, s_ptr = 1;
        char[] stack = new char[100];
        stack[0] = '$';
        stack[1] = table[0][0];
        while (s_ptr != -1) {
            System.out.print("\t\t\t\t\t\t");
            for (int vamp = 0; vamp <= s_ptr; vamp++) {
                System.out.print(stack[vamp]);
            }
            System.out.print("\t\t\t");
            int vamp = i_ptr;
            while (vamp < inputArr.length) {
                System.out.print(inputArr[vamp]);
                vamp++;
            }
            System.out.print("\t\t\t");
            char her = inputArr[i_ptr];
            char him = stack[s_ptr];
            s_ptr--;
```

```
            if (!Character.isUpperCase(him)) {
                if (her == him) {
                    i_ptr++;
                    System.out.println("POP ACTION");
                } else {
                    System.out.println("\nString Not Accepted by LL(1)
Parser !!\n");
                    System.exit(0);
                }
            } else {
                for (i = 0; i < sid; i++) {
                    if (ter[i] == her)
                        break;
                }
                char[] produ = new char[100];
                for (int j = 0; j < land; j++) {
                    if (him == table[j][0]) {
                        if (table[j][i + 1] == '#') {
                            System.out.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;
                            produ = production[mum];
                            System.out.println(new String(produ));
                        } else {
                            System.out.println("\nString Not Accepted
by LL(1) Parser !!\n");
                            System.exit(0);
                        }
                    }
                }
                int le = produ.length;
                le = le - 1;
                if (le == 0) {
```

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

System.out.println("\n\t\t\t=========================================
=============================================================================
=======\n");
        if (i_ptr == inputArr.length) {
            System.out.println("\t\t\t\t\t\t\t\tYOUR STRING HAS BEEN
ACCEPTED !!\n");
        } else
            System.out.println("\n\t\t\t\t\t\t\t\tYOUR STRING HAS BEEN
REJECTED !!\n");

System.out.println("\t\t\t=========================================
=============================================================================
=====\n");
    }

    static 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]);
                    }
                }
            }
        }
    }

    static void findfirst(char c, int q1, int q2) {
        int j;
        if (!(Character.isUpperCase(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 (!Character.isUpperCase(production[j][2])) {
                    first[n++] = production[j][2];
                } else {
                    findfirst(production[j][2], j, 3);
                }
            }
        }
    }

    static void followfirst(char c, int c1, int c2) {
        int k;
        if (!(Character.isUpperCase(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];
                    m++;
                } else {
                    if (production[c1][c2] == '\0') {
                        follow(production[c1][0]);
                    } else {
                        followfirst(production[c1][c2], c1, c2 + 1);
                    }
                }
                j++;
            }
        }
    }
}
```

Grammar.txt
E -> E + T | T
T -> T * F | F
F -> (E) | id

BHARATIYA VIDYA BHAVAN'S

SARDAR PATEL INSTITUTE OF TECHNOLOGY

(Empowered Autonomous Institute Affiliated to University of Mumbai)

[Knowledge is Nectar]

Department of Computer Engineering

Output

1. LL1 Parser

```
Hitstar53 at …\SPCC Practicals\Exp3 on main (⚠✓) via C v9.2.0-gcc
→ cd "d:\SEM_6\SPCC Practicals\Exp3\" ; if ($?) { gcc ll1parser.c -o ll1parser } ; if ($?) { .\ll1parser }
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
----------------------------------------------------------------------------------------------------------------
```

```
Please enter the desired INPUT STRING = aaaaab$

                                ========================================================================
                                        Stack                   Input                   Action
                                ========================================================================
                                        $S                      aaaaab$                 S=Bb
                                        $bB                     aaaaab$                 B=aB
                                        $bBa                    aaaaab$                 POP ACTION
                                        $bB                     aaaab$                  B=aB
                                        $bBa                    aaaab$                  POP ACTION
                                        $bB                     aaab$                   B=aB
                                        $bBa                    aaab$                   POP ACTION
                                        $bB                     aab$                    B=aB
                                        $bBa                    aab$                    POP ACTION
                                        $bB                     ab$                     B=aB
                                        $bBa                    ab$                     POP ACTION
                                        $bB                     b$                      B=#
                                        $b                      b$                      POP ACTION
                                        $                       $                       POP ACTION

                        ================================================================================================================
                                                        YOUR STRING HAS BEEN ACCEPTED !!
                        ================================================================================================================
Hitstar53 at …\SPCC Practicals\Exp3 on main (⚠✓) via C v9.2.0-gcc took 59s
```

2. SLR Parser

```
vboxuser@spit:~/Desktop/spcc/spcc3$ java LR_parser
Tokens: [E + T ,  T]
Tokens: [T * F ,  F]
Tokens: [( E ) ,  id]
{E'=[[E]], T=[[T, *, F], [F]], E=[[E, +, T], [T]], F=[[(, E, )], [id]]}
The first set:
E'->[(, id]
T->[(, id]
E->[(, id]
F->[(, id]
The follow set:
E'->[$]
T->[$, ), *, +]
E->[$, ), +]
F->[$, ), *, +]
[Id: 0 , Rules: [T -> . F , F -> . ( E ) , E' -> . E , F -> . id , T -> . T * F , E -> . E + T , E -> . T ]
, Id: 1 , Rules: [E -> T . , T -> T . * F ]
, Id: 2 , Rules: [E' -> E . , E -> E . + T ]
, Id: 3 , Rules: [T -> F . ]
, Id: 4 , Rules: [F -> ( . E ) , T -> . F , F -> . ( E ) , F -> . id , E -> . E + T , E -> . T , T -> . T * F ]
, Id: 5 , Rules: []
, Id: 6 , Rules: [F -> id . ]
, Id: 7 , Rules: [F -> . ( E ) , T -> T * . F , F -> . id ]
, Id: 8 , Rules: [T -> . F , F -> . ( E ) , E -> E + . T , F -> . id , T -> . T * F ]
, Id: 9 , Rules: [F -> ( E . ) , E -> E . + T ]
, Id: 10 , Rules: [T -> T * F . ]
, Id: 11 , Rules: [E -> E + T . , T -> T . * F ]
, Id: 12 , Rules: [F -> ( E ) . ]
]
Map for state: Id: 0 , Rules: [T -> . F , F -> . ( E ) , E' -> . E , F -> . id , T -> . T * F , E -> . E + T , E -> . T ]

Transitions:
T->[E -> T . , T -> T . * F ] ( S1 )
E->[E' -> E . , E -> E . + T ] ( S2 )
F->[T -> F . ] ( S3 )
(->[F -> ( . E ) , T -> . F , F -> . ( E ) , F -> . id , E -> . E + T , E -> . T , T -> . T * F ] ( S4 )
)->[] ( S5 )
*->[] ( S5 )
+->[] ( S5 )
id->[F -> id . ] ( S6 )
-->[] ( S5 )
/->[] ( S5 )

Map for state: Id: 1 , Rules: [E -> T . , T -> T . * F ]
```

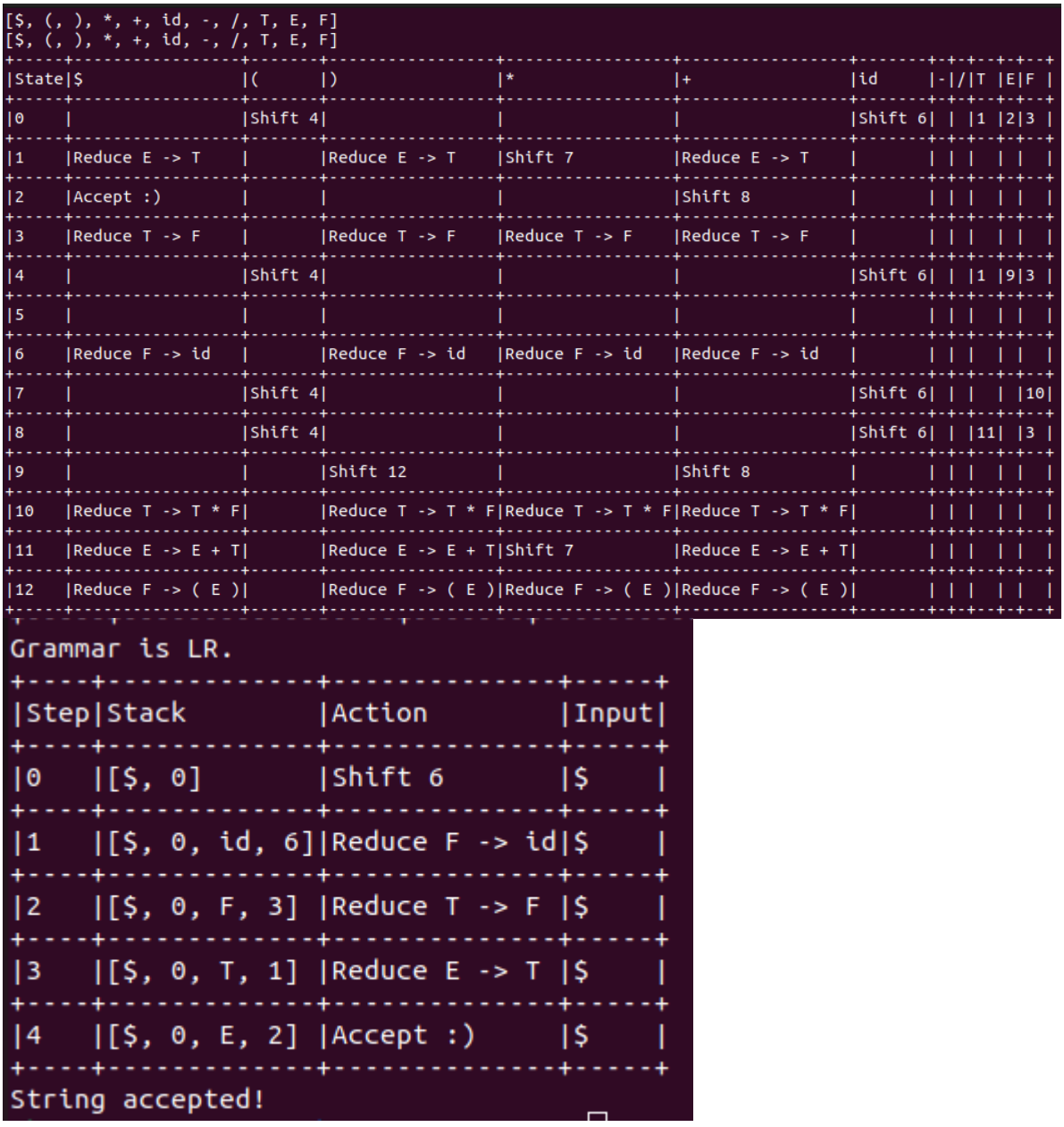
```
Map for state: Id: 1 , Rules: [E -> T . , T -> T . * F ]

Transitions:
T->[] ( S5 )
E->[] ( S5 )
F->[] ( S5 )
(->[] ( S5 )
)->[] ( S5 )
*->[F -> . ( E ) , T -> T * . F , F -> . id ] ( S7 )
+->[] ( S5 )
id->[] ( S5 )
-->[] ( S5 )
/->[] ( S5 )

Map for state: Id: 2 , Rules: [E' -> E . , E -> E . + T ]

Transitions:
T->[] ( S5 )
E->[] ( S5 )
F->[] ( S5 )
(->[] ( S5 )
)->[] ( S5 )
*->[] ( S5 )
+->[T -> . F , F -> . ( E ) , E -> E + . T , F -> . id , T -> . T * F ] ( S8 )
id->[] ( S5 )
-->[] ( S5 )
/->[] ( S5 )

Map for state: Id: 3 , Rules: [T -> F . ]

Transitions:
T->[] ( S5 )
E->[] ( S5 )
F->[] ( S5 )
(->[] ( S5 )
)->[] ( S5 )
*->[] ( S5 )
+->[] ( S5 )
id->[] ( S5 )
-->[] ( S5 )
/->[] ( S5 )

Map for state: Id: 4 , Rules: [F -> ( . E ) , T -> . F , F -> . ( E ) , F -> . id , E -> . E + T , E -> . T , T -> . T * F ]

Transitions:
T->[E -> T . , T -> T . * F ] ( S1 )
E->[F -> ( E . ) , E -> E . + T ] ( S9 )
F->[T -> F . ] ( S3 )
(->[F -> ( . E ) , T -> . F , F -> . ( E ) , F -> . id , E -> . E + T , E -> . T , T -> . T * F ] ( S4 )
)->[] ( S5 )
*->[] ( S5 )
+->[] ( S5 )
id->[F -> id . ] ( S6 )
-->[] ( S5 )
/->[] ( S5 )
```

```
Map for state: Id: 5 , Rules: []

Transitions:
T->[] ( S5 )
E->[] ( S5 )
F->[] ( S5 )
(->[] ( S5 )
)->[] ( S5 )
*->[] ( S5 )
+->[] ( S5 )
id->[] ( S5 )
-->[] ( S5 )
/->[] ( S5 )

Map for state: Id: 6 , Rules: [F -> id . ]

Transitions:
T->[] ( S5 )
E->[] ( S5 )
F->[] ( S5 )
(->[] ( S5 )
)->[] ( S5 )
*->[] ( S5 )
+->[] ( S5 )
id->[] ( S5 )
-->[] ( S5 )
/->[] ( S5 )
```

```
Map for state: Id: 7 , Rules: [F -> . ( E ) , T -> T * . F , F -> . id ]

Transitions:
T->[] ( S5 )
E->[] ( S5 )
F->[T -> T * F . ] ( S10 )
(->[F -> ( . E ) , T -> . F , F -> . ( E ) , F -> . id , E -> . E + T , E -> . T , T -> . T * F ] ( S4 )
)->[] ( S5 )
*->[] ( S5 )
+->[] ( S5 )
id->[F -> id . ] ( S6 )
-->[] ( S5 )
/->[] ( S5 )

Map for state: Id: 8 , Rules: [T -> . F , F -> . ( E ) , E -> E + . T , F -> . id , T -> . T * F ]

Transitions:
T->[E -> E + T . , T -> T . * F ] ( S11 )
E->[] ( S5 )
F->[T -> F . ] ( S3 )
(->[F -> ( . E ) , T -> . F , F -> . ( E ) , F -> . id , E -> . E + T , E -> . T , T -> . T * F ] ( S4 )
)->[] ( S5 )
*->[] ( S5 )
+->[] ( S5 )
id->[F -> id . ] ( S6 )
-->[] ( S5 )
/->[] ( S5 )
```

```
Map for state: Id: 9 , Rules: [F -> ( E . ) , E -> E . + T ]

Transitions:
T->[] ( S5 )
E->[] ( S5 )
F->[] ( S5 )
(->[] ( S5 )
)->[F -> ( E ) . ] ( S12 )
*->[] ( S5 )
+->[T -> . F , F -> . ( E ) , E -> E + . T , F -> . id , T -> . T * F ] ( S8 )
id->[] ( S5 )
-->[] ( S5 )
/->[] ( S5 )

Map for state: Id: 10 , Rules: [T -> T * F . ]

Transitions:
T->[] ( S5 )
E->[] ( S5 )
F->[] ( S5 )
(->[] ( S5 )
)->[] ( S5 )
*->[] ( S5 )
+->[] ( S5 )
id->[] ( S5 )
-->[] ( S5 )
/->[] ( S5 )
```

```
Map for state: Id: 11 , Rules: [E -> E + T . , T -> T . * F ]

Transitions:
T->[] ( S5 )
E->[] ( S5 )
F->[] ( S5 )
(->[] ( S5 )
)->[] ( S5 )
*->[F -> . ( E ) , T -> T * . F , F -> . id ] ( S7 )
+->[] ( S5 )
id->[] ( S5 )
-->[] ( S5 )
/->[] ( S5 )

Map for state: Id: 12 , Rules: [F -> ( E ) . ]

Transitions:
T->[] ( S5 )
E->[] ( S5 )
F->[] ( S5 )
(->[] ( S5 )
)->[] ( S5 )
*->[] ( S5 )
+->[] ( S5 )
id->[] ( S5 )
-->[] ( S5 )
/->[] ( S5 )
```



```
[$, (, ), *, +, id, -, /, T, E, F]
[$, (, ), *, +, id, -, /, T, E, F]
+-----+----------------+-------+----------------+----------------+----------------+-------+-+-+--+-+--+
|State|$               |(      |)               |*               |+               |id     |-|/|T |E|F |
+-----+----------------+-------+----------------+----------------+----------------+-------+-+-+--+-+--+
|0    |                |Shift 4|                |                |                |Shift 6| | |1 |2|3 |
+-----+----------------+-------+----------------+----------------+----------------+-------+-+-+--+-+--+
|1    |Reduce E -> T   |       |Reduce E -> T   |Shift 7         |Reduce E -> T   |       | | |  | |  |
+-----+----------------+-------+----------------+----------------+----------------+-------+-+-+--+-+--+
|2    |Accept :)       |       |                |                |Shift 8         |       | | |  | |  |
+-----+----------------+-------+----------------+----------------+----------------+-------+-+-+--+-+--+
|3    |Reduce T -> F   |       |Reduce T -> F   |Reduce T -> F   |Reduce T -> F   |       | | |  | |  |
+-----+----------------+-------+----------------+----------------+----------------+-------+-+-+--+-+--+
|4    |                |Shift 4|                |                |                |Shift 6| | |1 |9|3 |
+-----+----------------+-------+----------------+----------------+----------------+-------+-+-+--+-+--+
|5    |                |       |                |                |                |       | | |  | |  |
+-----+----------------+-------+----------------+----------------+----------------+-------+-+-+--+-+--+
|6    |Reduce F -> id  |       |Reduce F -> id  |Reduce F -> id  |Reduce F -> id  |       | | |  | |  |
+-----+----------------+-------+----------------+----------------+----------------+-------+-+-+--+-+--+
|7    |                |Shift 4|                |                |                |Shift 6| | |  | |10|
+-----+----------------+-------+----------------+----------------+----------------+-------+-+-+--+-+--+
|8    |                |Shift 4|                |                |                |Shift 6| | |11| |3 |
+-----+----------------+-------+----------------+----------------+----------------+-------+-+-+--+-+--+
|9    |                |       |Shift 12        |                |Shift 8         |       | | |  | |  |
+-----+----------------+-------+----------------+----------------+----------------+-------+-+-+--+-+--+
|10   |Reduce T -> T * F|      |Reduce T -> T * F|Reduce T -> T * F|Reduce T -> T * F|      | | |  | |  |
+-----+----------------+-------+----------------+----------------+----------------+-------+-+-+--+-+--+
|11   |Reduce E -> E + T|      |Reduce E -> E + T|Shift 7         |Reduce E -> E + T|      | | |  | |  |
+-----+----------------+-------+----------------+----------------+----------------+-------+-+-+--+-+--+
|12   |Reduce F -> ( E )|      |Reduce F -> ( E )|Reduce F -> ( E )|Reduce F -> ( E )|      | | |  | |  |
+-----+----------------+-------+----------------+----------------+----------------+-------+-+-+--+-+--+
Grammar is LR.
+----+-------------+--------------+-----+
|Step|Stack        |Action        |Input|
+----+-------------+--------------+-----+
|0   |[$, 0]       |Shift 6       |$    |
+----+-------------+--------------+-----+
|1   |[$, 0, id, 6]|Reduce F -> id|$    |
+----+-------------+--------------+-----+
|2   |[$, 0, F, 3] |Reduce T -> F |$    |
+----+-------------+--------------+-----+
|3   |[$, 0, T, 1] |Reduce E -> T |$    |
+----+-------------+--------------+-----+
|4   |[$, 0, E, 2] |Accept :)     |$    |
+----+-------------+--------------+-----+
String accepted!
```

Conclusion	In this experiment we learned how to implement a LL1 parser and SLR Parser. We also learnt how to find the first and follow of a given production and also made a parse table for the same. We also parsed a string to check its validity and made a stack for the same.
References	[1] siddhantbajaj1 (no date) *Siddhantbajaj1/LL-1-parsing-table*, *GitHub*. Available at: https://github.com/siddhantbajaj1/LL-1-Parsing-Table (Accessed: 14 February 2024). [2] *First set in Syntax Analysis* (2023) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/first-set-in-syntax-analysis/ (Accessed: 14 February 2024).

[3] *Follow set in Syntax Analysis* (2023) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/follow-set-in-syntax-analysis/?ref=lbp (Accessed: 14 February 2024).

[4] *Construction of LL(1) parsing table* (2023) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/construction-of-ll1-parsing-table/ (Accessed: 14 February 2024).

[5] GfG. (2022, February 25). *SLR parser (with examples)*. GeeksforGeeks. https://www.geeksforgeeks.org/slr-parser-with-examples/

[6] PalAditya. (n.d.-a). *Paladitya/CompilerDesignLab: Code for Compiler Design Lab, semester 7*. GitHub. https://github.com/PalAditya/CompilerDesignLab