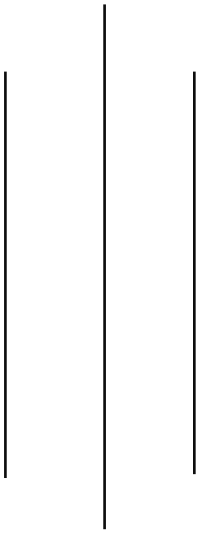
NAGAR JUNA COLLEGE OF IT

AFFILIATED TO TRIBHUWAN UNIVERSITY



LAB REPORT

OF

COMPILER DESIGN & CONSTRUCTION

BSc.CSIT 6th Semester

SUBMITTED BY:

Suravi Shrestha

Roll number: 33

Symbol no: 26472/077

Regd. No: 5-2-429-35-2020

SUBMITTED TO:

Bhim Bahadur Rawat

Subject: Compiler Design & Construction

Code: CSC 365

COMPILER DESIGN & CONSTRUCTION

Course: BSc.CSIT

Semester: 6th

Name: Suravi Shrestha

Roll No: 33

Symbol no. : 26472/077

LIST OF LAB WORKS

LAB. No	TITLE/QUESTION	Date of Submission	Signature
1	PREFIXES, SUFFIXES AND SUBSTRING	7/19/2024	
2	IDENTIFIER & KEYWORD	7/19/2024	
3	IMPLEMENTATION OF DFA	7/19/2024	
4	IMPLEMENTATION OF REGULAR EXPRESSION	7/19/2024	
5	COMMENT IN C	7/19/2024	
6	COMPUTATION OF FIRST	7/19/2024	
7	COMPUTATION OF FOLLOW	7/19/2024	
8	IDENTIFIER VALIDATION	7/19/2024	
9	LEXICAL ANALYSIS	7/19/2024	
10	TYPE CHECKING	7/19/2024	
11	SHIFT REDUCE PARSER	7/19/2024	

LAB. No	TITLE/QUESTION	Date of Submission	Signature
12	IMPLEMENTATION OF SYMBOL TABLE	7/19/2024	
13	IMPLEMENTATION OF SLR(1) GRAMMER	7/19/2024	
14	IMPLEMENTATION OF LL(1) GRAMMER	7/19/2024	
15	IMPLEMENTATION OF BRUTE FORCE GRAMMER	7/19/2024	

LAB: 1

TITLE: PREFIXES, SUFFIXES AND SUBSTRING

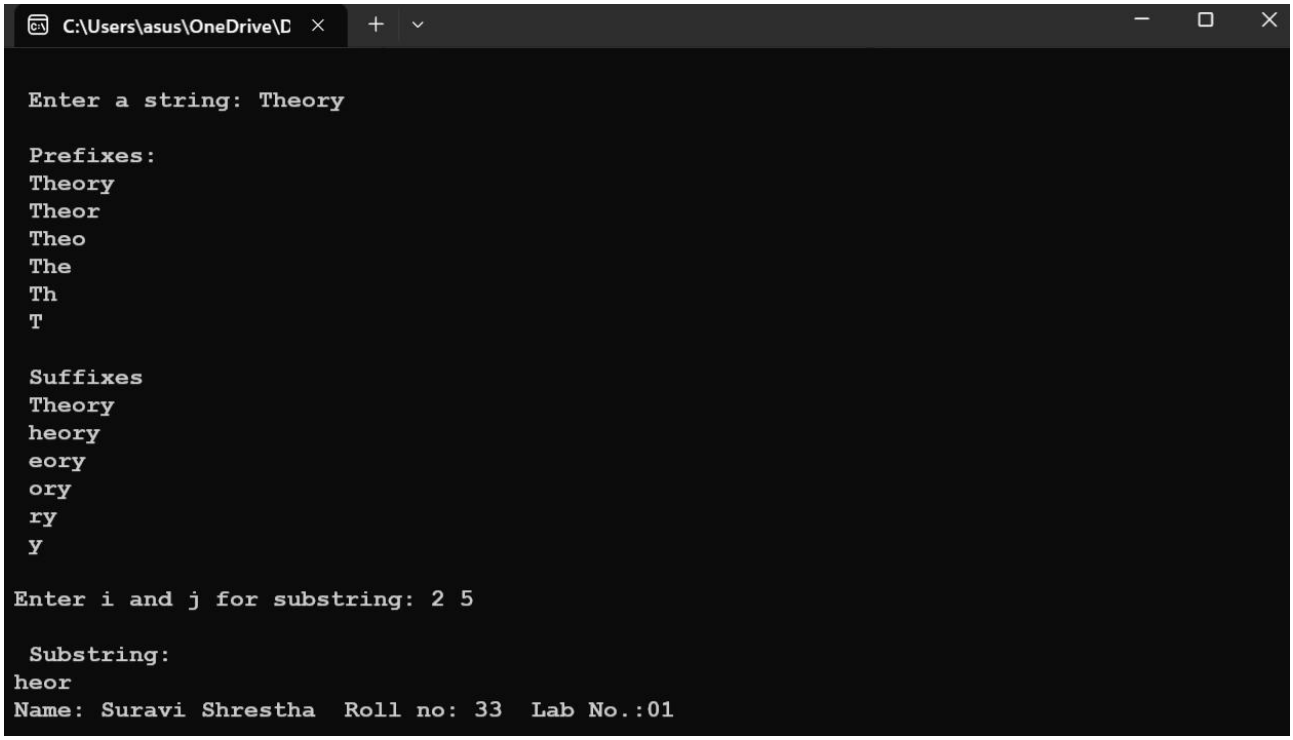
OBJECTIVE: Write a program to find prefixes, suffixes and substring from given string.

SOURCE CODE:

```
#include<stdio.h>
#include<string.h>
void find_prefix(char string[]);
void find_suffix(char string[]);
void find_substring(char string[],int,int);

int main()
{
        char string[20];
        int i,j;
        printf("\n Enter a string\t");
        gets(string);

        printf("\n Prefixes:");
        find_prefix(string);
        printf("\n Suffixes");
        find_suffix(string);

        printf("\nEnter i and j for substring");
        scanf("%d%d",&i,&j);
        find_substring(string,i,j);

        return 0;
}

void find_prefix(char string[])
{
        int i,j;
        char prefix[20];
        for(i=strlen(string);i>=0;i--)
        {
                for( j = 0; j<i;j++)
```

```
                {
                        prefix[j]= string[j];
                }
                prefix[j]='\0';
                printf("\n %s",prefix);
        }
}
void find_suffix(char string[])
{
        int i,j,k;
        char suffix[20];
        for(i=0;i<=strlen(string);i++)
        {
                k = i;
                for( j = 0; j<strlen(string);j++)
                {
                        suffix[j]= string[k];
                        k++;
                }
                suffix[j]='\0';
                printf("\n %s",suffix);
        }
}

void find_substring(char string[],int x, int y)
{
        char substr[20];

        int k=0;
        for(int i=x-1;i<y;i++)
        {
            substr[k]=string[i];
            k++;
        }
        substr[k]='\0';
        printf("\n Substring:\n%s",substr);

}
```

OUTPUT


C:\Users\asus\OneDrive\D
Enter a string: Theory
Prefixes:
Theory
Theor
Theo
The
Th
T
Suffixes
Theory
heory
eory
ory
ry
y
Enter i and j for substring: 2 5
Substring:
heor
Name: Suravi Shrestha Roll no: 33 Lab No.:01

LAB: 2

TITLE: IDENTIFIER & KEYWORD

OBJECTIVE: Write a program to validate C identifiers and keywords.

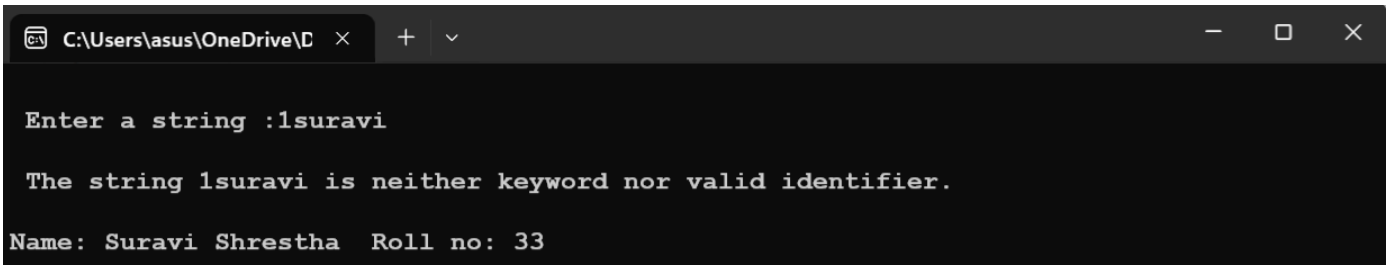
SOURCE CODE:

```
#include<stdio.h>
#include<string.h>
char keyword[32][10]= { "auto","double","int","struct","break","else","long",
switch","case", "enum", "register", "typedef","char",
"extern", "return", "union", "const", "float", "short",
"unsigned","continue","for","signed","void","default", "goto",
"sizeof", "volatile", "do","if","static","while"} ;
enum states { q0, qf, qd};
enum states delta(enum states, char);
int iskeyword(char []);

int main()
{
        enum states curr_state = q0;
        char string[20], ch;
        int i=0;

        printf("\n Enter a string \t");
        gets(string);

        ch = string[i];
        if(iskeyword(string))
          printf("\n The string %s is keyword.",string);
        else
        {
                while(ch!='\0')
                {
                        curr_state = delta(curr_state,ch);
                        ch = string[++i];
                }
                if(curr_state==qf)
                         printf("\n The string %s is valid indentifier.",string);
```

```
                else
                        printf("\n The string %s is neither keyword nor valid
identifier.",string);
        }
        return 0;
} //end of the main

//transition function
enum states delta(enum states s, char ch)
{
        enum states curr_state;
        switch(s)
        {
                case q0:
                        if(ch>='A' && ch<='Z'||ch>='a'&&ch<='z'||ch=='_')
                          curr_state = qf;
                        else
                          curr_state = qd;
                        break;
                case qf:
                        if(ch>='A' &&
ch<='Z'||ch>='a'&&ch<='z'||ch=='_'||ch>='0'&&ch<='9')
                          curr_state = qf;
                        else
                          curr_state = qd;
                        break;
                case qd:
                        curr_state = qd;
        }

                return curr_state;
}

int iskeyword(char str[])
{
        for(int i=0;i<32;i++)
        {
                if(strcmp(str,keyword[i])==0)
                  return 1;
        }
```

```
    return 0;
}
```

OUTPUT



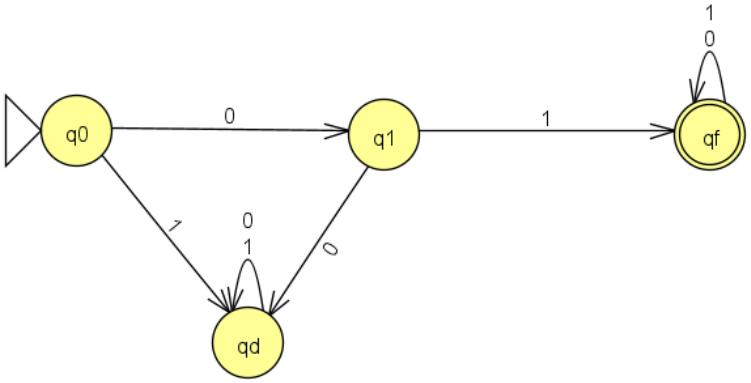
LAB: 3

TITLE: IMPLEMENTATION OF DFA

OBJECTIVE: Write a C program to implement following DFA's over alphabet L={ 0,1}

i. The DFA that accepts all strings that start with 01.

- Machine:



- Steps for string: 0110

SOURCE CODE:

```
#include<stdio.h>
enum states { q0, q1, qf,qd};
enum states delta(enum states, char);
int main(){
        char input[20];
        enum states curr_state = q0;
        int i =0;
```

```
        printf("\n Enter a binary string\t");
        gets(input);
        char ch = input[i];
        while( ch !='\0'){
            curr_state = delta(curr_state,ch);
                ch = input[++i];
        }if(curr_state == qf)
           printf("\n The string %s is accepted.",input);
        else
            printf("\n The string %s is not accepted.",input);
   return 0;
}enum states delta(enum states s, char ch){
      enum states curr_state;
            switch(s){              case q0:
                                    if(ch=='0')
                                      curr_state = q1;
                                    else
                                      curr_state = qd;
                                    break;
                        case q1:
                                    if(ch=='1')
                                      curr_state = qf;
                                    else
                                      curr_state = qd;
                                    break;
                        case qf:
                                    if(ch=='0')
                                      curr_state = qf;
                                    else
                                      curr_state = qf;
                                    break;
                        case qd:
                                    if(ch=='0')
                                      curr_state = qd;
                                    else
                                      curr_state = qd;
                                    break;
                    }return curr_state;}
```

OUTPUT:

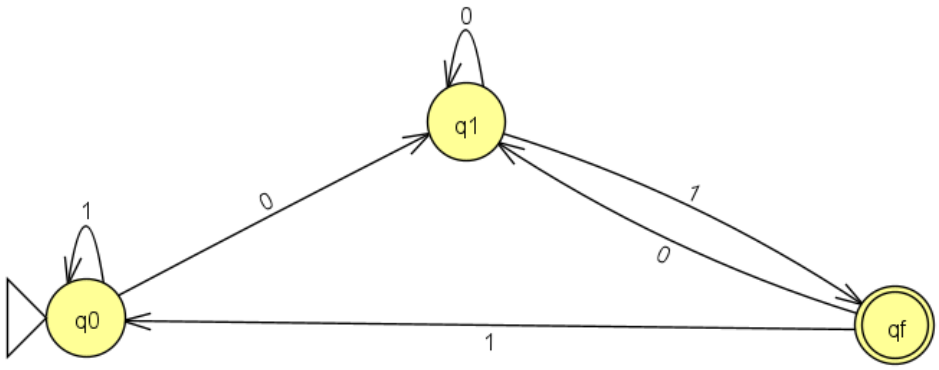
```
Enter a binary string  0110

The string 0110 is accepted.

Name: Suravi Shrestha  Roll no: 33
--------------------------------
```

ii. The DFA that accepts all the strings that end with 01.

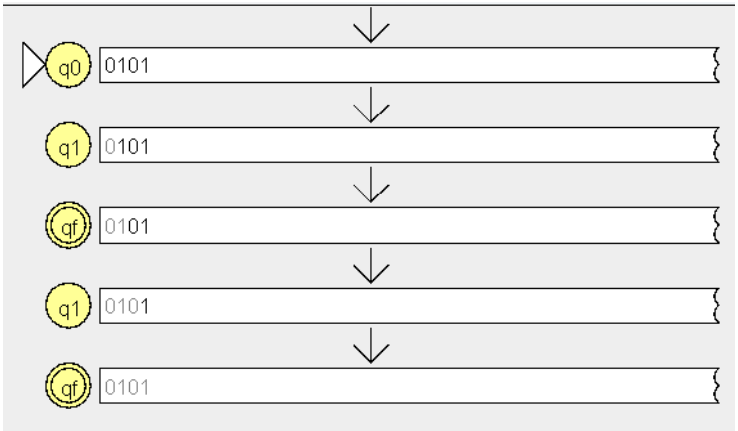
Q = { q0, q1, qf}
start state = q0,
Final state = qf
Transition function, δ is defined as:

$$\delta(q0,0) = q1$$
$$\delta(q0,1) = q0$$
$$\delta(q1,0) = q1$$
$$\delta(q1,1) = qf$$
$$\delta(qf,0) = q1$$
$$\delta(qf,1) = q0$$

- Machine:



- Steps for string: <u>0101</u>

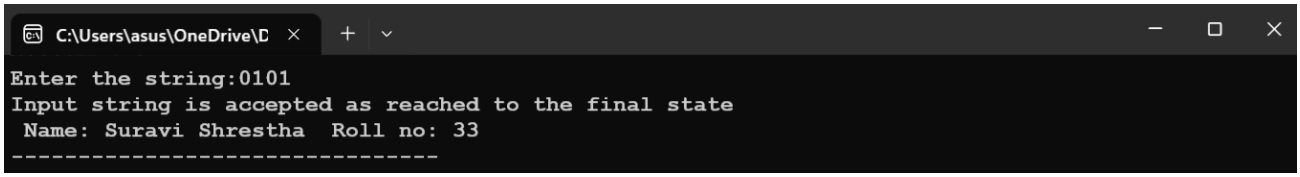


SOURCE CODE:

```
#include<stdio.h>
enum states { q0, q1, qf};
enum states delta(enum states, char);
int main()
{
        char input[20];
        enum states curr_state = q0;
        int i =0;

        printf("\n Enter a binary string\t");
        gets(input);
        char ch = input[i];
        while( ch !='\0')
        {
           curr_state = delta(curr_state,ch);
                ch = input[++i];
        }
                if(curr_state == qf)
          printf("\n The string %s is accepted.",input);
        else
           printf("\n The string %s is not accepted.",input);

  return 0;
}
enum states delta(enum states s, char ch)
{
     enum states curr_state;
          switch(s)
                {
                        case q0:
                                if(ch=='0')
                                  curr_state = q1;
                                else
                                  curr_state = q0;
                                break;
                        case q1:
                                if(ch=='1')
                                  curr_state = qf;
                                else
                                  curr_state = q1;
                                break;
                        case qf:
```

```
                                    if(ch=='0')
                                      curr_state = q1;
                                    else
                                      curr_state = q0;
                                    break;
                        }
            return curr_state;
    }
```

OUTPUT:

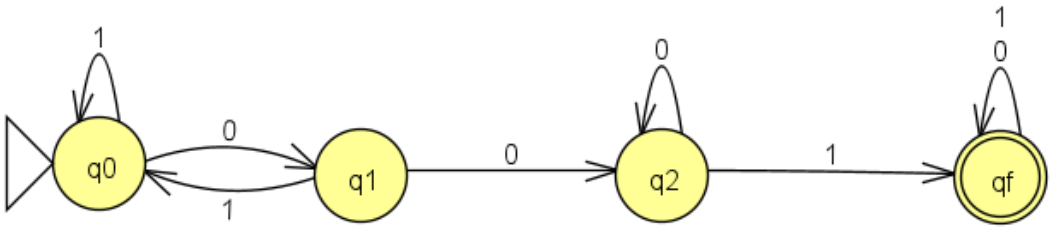


iii. The DFA that accepts all string that contains substring 001.

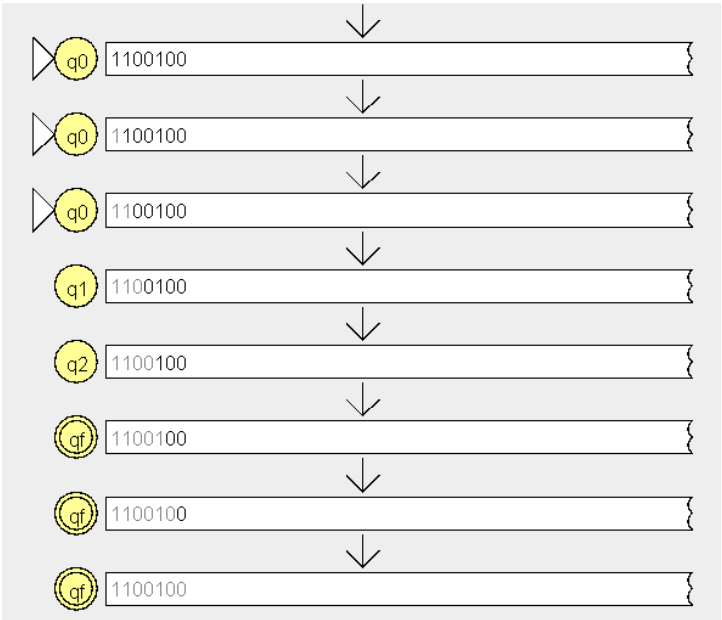
Q = {q0, q1, q2, qf}
start state = q0,
Final state = qf
Transition function, δ is defined as:

$$\delta(q0,0) = q1$$
$$\delta(q0,1) = q0$$
$$\delta(q1,0) = q2$$
$$\delta(q1,1) = q0$$
$$\delta(q2,0) = q2$$
$$\delta(q2,1) = qf$$
$$\delta(qf,0) = qf$$
$$\delta(qf,1) = qf$$

- Machine:



- Steps for string: <u>1100100</u>



SOURCE CODE:

```
#include<stdio.h>
enum states { q0,q1,q2,qf};
enum states delta(enum states, char);
int main()
{
	enum states curr_state = q0;
	char string[20], ch;
	int i=0;
	printf("\n Enter a string \t");
	gets(string);
		ch = string[i];
	while(ch!='\0')
	{
		curr_state = delta(curr_state,ch);
		ch = string[++i];
	}
	if(curr_state==qf)
	  printf("\n The string %s is valid.",string);
	else
	  printf("\n The string %s is not valid.",string);
		return 0;
}
enum states delta(enum states s, char ch)
{
	enum states curr_state;
	switch(s)
	{
		case q0:
			if(ch=='0')
			  curr_state = q1;
			else
			  curr_state = q0;
			break;
		case q1:
			if(ch=='0')
			  curr_state = q2;
			else
			  curr_state = q0;
			break;
		case q2:
			if(ch=='0')
			  curr_state = q2;
```

```
                else
                  curr_state = qf;
                break;
        case qf:
                if(ch=='0'||ch=='1')
                  curr_state = qf;
        }
        return curr_state;
}
```

OUTPUT:

```
 Enter a string         1010010

 The string 1010010 is valid.
Name: Suravi Shrestha  Roll no: 33
```

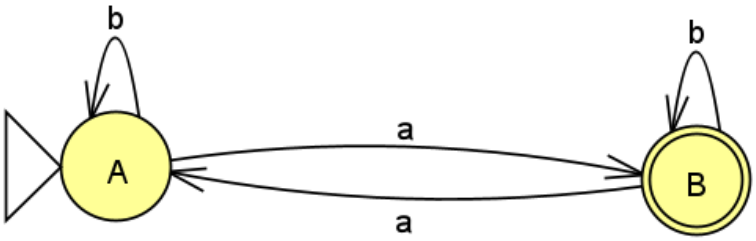
LAB: 4

TITLE: IMPLEMENTATION OF REGULAR EXPRESSION

OBJECTIVE: Write a C program to implement following DFA for L={ Set of all strings over $\sum = \{a, b\}$ such that string has odd no. of a's}.

Regular Expression: b*a(b*ab*a)*b*.

- Machine:



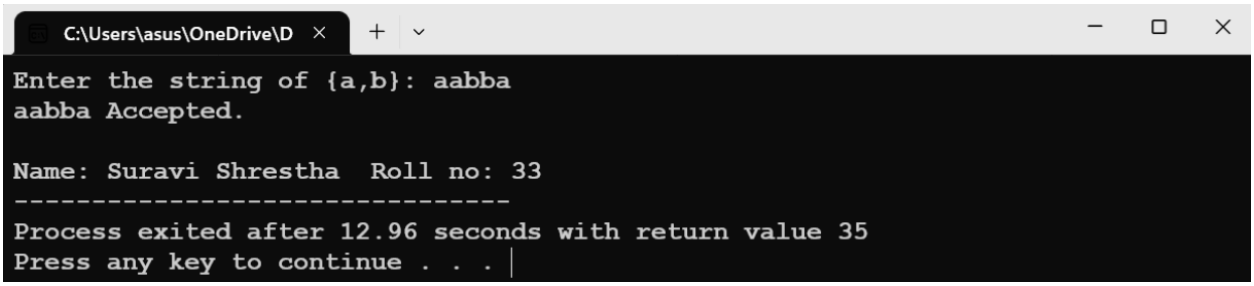
- Steps for string: aabba

SOURCE CODE:

```
#include <stdio.h>
#include <string.h>
int main()
{
   char str[100], state= 'A';
   int i;
```

```
    printf("Enter the string of {a,b}: ");
    scanf("%s",str);
    for(i=0;str[i]!='\0';i++)
    {
       switch(state)
       {
          case 'A':
          if(str[i]=='a')
             state='B';
          else if(str[i]=='b')
             state='A';
          break;
          case 'B':
          if(str[i]=='a')
             state='A';
          else if(str[i]=='b')
             state='B';
          break;
       }
    }   if(state=='B')  {
       printf("%s Accepted.\n",str);
    }
    else {
       printf("%s Rejected.\n", str);
    }  printf("\nName: Suravi Shrestha  Roll no: 33");
}
```

OUTPUT:



LAB: 5

TITLE: COMMENT IN C.

OBJECTIVE: Write a C program to identify weather a given line is a comment.

ALGORITHM

1. Read the input string.

2. Check wheather the string is starting '/' and check next character is '/'or '*'

3. If condition satisfies print comment.

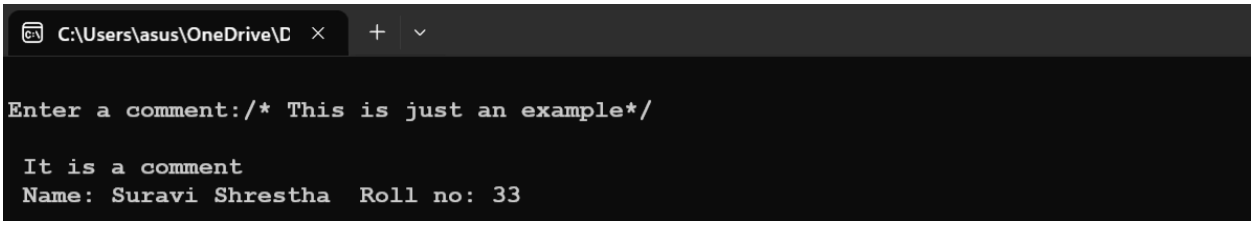
4. Else not a comment.

SOURCE CODE:

```
#include<stdio.h>
void main()
{
  char comment[30];
  int i = 2, a = 0;
  //clrscr();
  printf("\nEnter a comment:");
  gets(comment);
  if(comment[0] == '/')
  {
   if(comment[1] == '/')
     printf("\n It is a comment");
   else if(comment[1]=='*')
      {
        for(i=2; i<=30; i++)
        {
         if(comment[i]== '*' && comment[i+1] == '/')
           {
             printf("\n It is a comment ");
             a = 1;
             break;
```

```
            }
           else
            continue;
          }
          if(a == 0)
           printf("\n It is not a comment");
        }
     else
      printf("\n It is not a comment");
   }
   else
    printf("\n It is not a comment");
    //getch();
     printf("\n Name: Suravi Shrestha  Roll no: 33");
}
```

OUTPUT:



LAB: 6

TITLE: COMPUTATION OF FIRST

OBJECTIVE: Write a C program to illustrate computation of FIRST.

SOURCE CODE

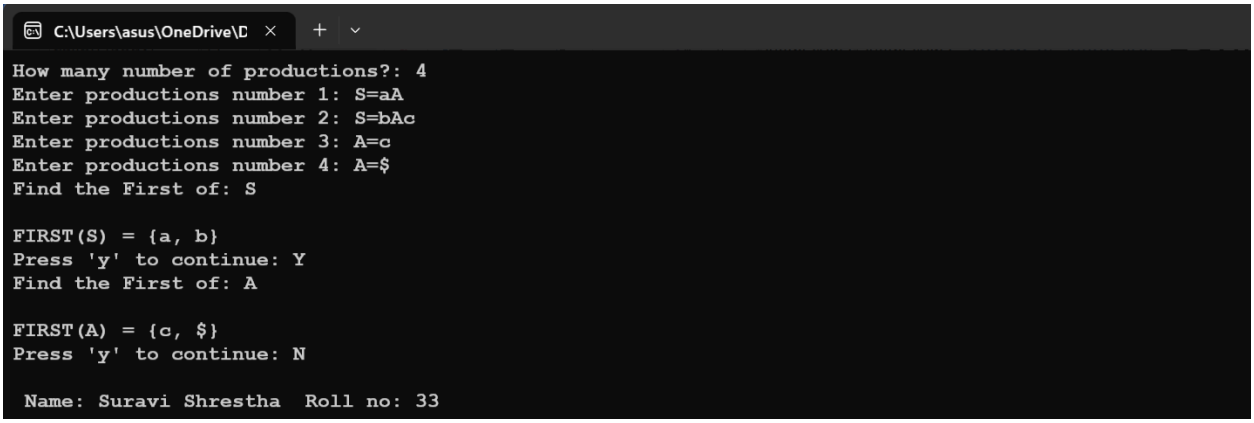
```
include <stdio.h>
#include <ctype.h>
#include <string.h>
void FIRST(char[], char);
void addToResultSet(char[], char);
int numOfProductions;
char productionSet[10][10];

int main() {
   int i;
   char choice;
   char ch;
   char result[20];

   printf("How many number of productions?: ");
   scanf("%d", &numOfProductions);

   // Ensure the newline character after the number is consumed
   getchar();

   for (i = 0; i < numOfProductions; i++) {
      printf("Enter productions number %d: ", i + 1);
      scanf("%s", productionSet[i]);
   }

   do {
      printf("Find the First of: ");
      scanf(" %c", &ch);  // Note the space before %c to consume any leftover whitespace

      // Initialize result to an empty string
```

```
        result[0] = '\0';

        FIRST(result, ch);

        printf("\nFIRST(%c) = {", ch);
        for (i = 0; result[i] != '\0'; i++) {
            printf("%c", result[i]);
            if (result[i + 1] != '\0') {
                printf(", ");
            }
        }
        printf("}\n");

        printf("Press 'y' to continue: ");
        scanf(" %c", &choice);  // Note the space before %c to consume any leftover whitespace
    } while (choice == 'y' || choice == 'Y');
            printf("\n Name: Suravi Shrestha  Roll no: 33");
    return 0;
}

void FIRST(char* Result, char ch) {
    int i, j, k;
    char subResult[20];
    int foundEpsilon;

    // Initialize subResult to an empty string
    subResult[0] = '\0';

    // If X is terminal, FIRST(X) = {X}
    if (!isupper(ch)) {
        addToResultSet(Result, ch);
        return;
    }

    // For each production
    for (i = 0; i < numOfProductions; i++) {
        // Check if the production is of the form X -> ...
        if (productionSet[i][0] == ch) {
```

```
        // If X -> epsilon, add epsilon to FIRST(X)
        if (productionSet[i][2] == '$') {
           addToResultSet(Result, '$');
        } else {
           // For each symbol in the production body
           j = 2;
           while (productionSet[i][j] != '\0') {
              foundEpsilon = 0;

              // Recursively calculate FIRST
              FIRST(subResult, productionSet[i][j]);

              // Add FIRST(Y) to FIRST(X)
              for (k = 0; subResult[k] != '\0'; k++) {
                 addToResultSet(Result, subResult[k]);
              }

              // Check if epsilon is in FIRST(Y)
              for (k = 0; subResult[k] != '\0'; k++) {
                 if (subResult[k] == '$') {
                    foundEpsilon = 1;
                    break;
                 }
              }

              // If epsilon is not in FIRST(Y), stop
              if (!foundEpsilon) {
                 break;
              }

              j++;
           }
        }
     }
  }
}

void addToResultSet(char Result[], char val) {
```

```
    int k;

    // Check if val is already in Result
    for (k = 0; Result[k] != '\0'; k++) {
        if (Result[k] == val) {
            return;
        }
    }

    // Add val to Result
    Result[k] = val;
    Result[k + 1] = '\0';
}
```

OUTPUT



LAB: 7

TITLE: COMPUTATION OF FOLLOW

OBJECTIVE: Write a C program to illustrate computation of FOLLOW.

SOURCE CODE:

```
#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;

   printf("Enter the number of productions: ");
```

```
scanf("%d", &count);

// Ensure the newline character after the number is consumed
getchar();

printf("Enter the productions (e.g., E=TR| #=EPSILON):\n");
for (i = 0; i < count; i++) {
   scanf("%s", production[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 alredy 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++;
     }
        printf("\n Name: Suravi Shrestha  Roll no: 33");
}

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++;
        }
      }
    }
```

OUTPUT:

```
Enter the number of productions: 4
Enter the productions (e.g., E=TR| #=EPSILON):
S=aA
S=bAc
A=c
A=#

 First(S) = { a, b, }

 First(A) = { c, #, }

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

 Follow(S) = { $,  }

 Follow(A) = { $, c,  }


 Name: Suravi Shrestha  Roll no: 33
```

LAB: 8

TITLE: IDENTIFIER VALIDATION

OBJECTIVE: Write a C program to test a given identifier is valid or not.

ALGORITHM/ LOGIC:

1. Read the input string.

2. Check the start character of the string is numerical or any special character except '_'

then print it is not a valid identifier.

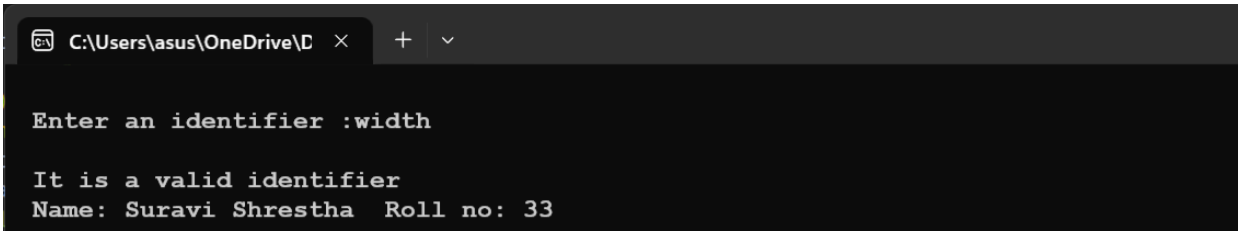
3.Otherwise: print it is valid identifier if remaining character of string does't contain any special character except '_'.

SOURCE CODE:

```
#include<stdio.h>
//#include<conio.h>
#include<ctype.h>

void main()
{
 char identifier[20];
 int flag, i = 1;
 //clrscr();
 printf("\n Enter an identifier :");
 gets(identifier);
 if(isalpha(identifier[0]))
  flag = 1;
 else
  printf("\n It is not valid identifier");

 while(identifier[i] != '\0')
   {
    if(!isdigit(identifier[i])&&!isalpha(identifier[i]))
     {
```

```
        flag = 0;
        break;
       }
      i++;
    }
  if(flag == 1)
    printf("\n It is a valid identifier");

          printf("\n Name: Suravi Shrestha  Roll no: 33");
  //getch();
}
```

OUTPUT:



LAB: 9

TITLE: LEXICAL ANALYSIS

OBJECTIVE: Write a C program to simulate lexical analysis for validating operators.

SOURCE CODE

```
#include <stdio.h>
#include <string.h>
int main() {
   char arithmetic[5] = {'+', '-', '*', '/', '%'};
   char relational[4] = {'<', '>', '!', '='};
   char bitwise[5] = {'&', '^', '~', '|'};
   char str[2] = {' ', ' '};
   int valid = 0; // Flag to indicate if a valid operator is found
   printf("Enter value to be identified: ");
   scanf("%s", &str);
   int i;
   if (((str[0] == '&' || str[0] == '|') && str[0] == str[1]) || (str[0] == '!' && str[1] == '\0')) {
      printf("\nIt is a Logical operator");
      valid = 1;
   }
   for (i = 0; i < 4; i++) {
      if (str[0] == relational[i] && (str[1] == '=' || str[1] == '\0')) {
         printf("\nIt is a Relational Operator");
         valid = 1;
         break;
      }
   }
   for (i = 0; i < 4; i++) {
      if ((str[0] == bitwise[i] && str[1] == '\0') || ((str[0] == '<' || str[0] == '>') && str[1] ==
str[0])) {
         printf("\nIt is a Bitwise Operator");
         valid = 1;
         break;
      }
   }
```

```
    if (str[0] == '?' && str[1] == ':') {
       printf("\nIt is a Ternary operator");
       valid = 1;
    }
          for (i = 0; i < 5; i++) {
       if ((str[0] == '+' || str[0] == '-') && str[0] == str[1]) {
          printf("\nIt is a Unary operator");
          valid = 1;
          break;
       } else if ((str[0] == arithmetic[i] && str[1] == '=') || (str[0] == '=' && str[1] == ' ')) {
          printf("\nIt is an Assignment operator");
          valid = 1;
          break;
       } else if (str[0] == arithmetic[i] && str[1] == '\0') {
          printf("\nIt is an Arithmetic operator");
          valid = 1;
          break;
       }
    }
    if (!valid) {
       printf("\nThe input is not a valid operator");
    }
    printf("\n Name: Suravi Shrestha  Roll no: 33");
    return 0;
}
```

OUTPUT:

```
Enter value to be identified: $

The input is not a valid operator
 Name: Suravi Shrestha  Roll no: 33
```

```
Enter value to be identified: ++

It is a Unary operator
 Name: Suravi Shrestha  Roll no: 33
```

LAB: 10

TITLE: TYPE CHECKING

OBJECTIVE: Write a C program to implement a symbol table to check type.

SOURCE CODE

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

int main() {
   int i = 0, j = 0, x = 0, n, flag = 0;
   void *p, *add[15];
   char srch, b[15], d[15], c;

   printf("Enter expression terminated by $: ");
   while((c = getchar()) != '$') {
      b[i] = c;
      i++;
   }
   b[i] = '\0'; // Null-terminate the string
   n = i;

   printf("Given expression is: %s\n", b);
   printf("Symbol table:\n");
   printf("Symbol\t Address\t\t Type\n");

   for (j = 0; j < n; j++) {
      c = b[j];
      if (isalpha(c)) {
         if (j == n - 1 || b[j + 1] == '+' || b[j + 1] == '-' || b[j + 1] == '*' || b[j + 1] == '=') {
            p = malloc(sizeof(char));
            add[x] = p;
            d[x] = c;
            printf("%c\t %p\t identifier\n", c, p);
            x++;
```

```
            }
        }
    }

    printf("Enter the symbol to be searched: ");
    scanf(" %c", &srch);

    for (i = 0; i < x; i++) {
        if (srch == d[i]) {
            printf("Symbol found in table:\n");
            printf("Symbol\t Address\t\t Type\n");
            printf("%c\t %p\t identifier\n", srch, add[i]);
            flag = 1;
            break;
        }
    }

    if (flag == 0)
        printf("Symbol not found in the table.\n");

    for (i = 0; i < x; i++) {
        free(add[i]);
    }
printf("\n Name: Suravi Shrestha  Roll no: 33\n");
    return 0;
}
```

OUTPUT:

```
Enter expression terminated by $: a+b*c+d$
Given expression is: a+b*c+d
Symbol table:
Symbol   Address                 Type
a        0000000000D19DD0        identifier
b        0000000000D19DF0        identifier
c        0000000000D19E10        identifier
d        0000000000D19E30        identifier
Enter the symbol to be searched: b
Symbol found in table:
Symbol   Address                 Type
b        0000000000D19DF0        identifier

 Name: Suravi Shrestha  Roll no: 33
```

LAB: 11

TITLE: SHIFT REDUCE PARSER

OBJECTIVE: Write a C program to implement Shift Reduce Parser.

SOURCE CODE

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

char ip_sym[20], stack[20];
int ip_ptr = 0, st_ptr = 0, len, i;
char temp[2], temp2[2];
char act[20];

void check();

void main() {
   printf("\n\t\t Shift Reduce Parser\n");
   printf("\n Grammar\n");
   printf("\n E->E+E \n E->E/E");
   printf("\n E->E*E \n E->a/b");

   printf("\n Enter the input symbol:\t");
   gets(ip_sym);

   printf("\n\t Stack implementation table");
   printf("\n Stack \t\t input symbol\t\taction");
   printf("\n-----------\t\t-------------\t\t---------\n");
   printf("\n$\t\t%s$\t\t\t---", ip_sym);

   strcpy(act, "Shift ");
   temp[0] = ip_sym[ip_ptr];
   temp[1] = '\0';
   strcat(act, temp);
   len = strlen(ip_sym);
```

```
    for (i = 0; i < len; i++) {
        stack[st_ptr] = ip_sym[ip_ptr];
        stack[st_ptr + 1] = '\0';
        ip_sym[ip_ptr] = ' ';
        ip_ptr++;
        printf("\n $%s\t\t%s$\t\t\t%s", stack, ip_sym, act);
        strcpy(act, "Shift ");
        temp[0] = ip_sym[ip_ptr];
        temp[1] = '\0';
        strcat(act, temp);
        check();
        st_ptr++;
    }

    st_ptr++;
    check();

}

void check() {
    int flag = 0;
    temp2[0] = stack[st_ptr];
    temp2[1] = '\0';
    if ((!strcmp(temp2, "a")) || (!strcmp(temp2, "b"))) {
        stack[st_ptr] = 'E';
        if (!strcmp(temp2, "a"))
            printf("\n $%s\t\t%s\t\t\tE->a", stack, ip_sym);
        else
            printf("\n $%s\t\t%s$\t\t\tE->b", stack, ip_sym);
        flag = 1;
    }

    if ((!strcmp(temp2, "+")) || (!strcmp(temp2, "*")) || (!strcmp(temp2, "/"))) {
        flag = 1;
    }

    if ((!strcmp(stack, "E+E")) || (!strcmp(stack, "E/E")) || (!strcmp(stack, "E*E"))) {
        strcpy(stack, "E");
```

```
        st_ptr = 0;
        if (!strcmp(stack, "E+E"))
          printf("\n $%s\t\t%s\t\t\tE->E+E", stack, ip_sym);
        else if (!strcmp(stack, "E/E"))
          printf("\n $%s\t\t%s\t\t\tE->E/E", stack, ip_sym);
        else if (!strcmp(stack, "E*E"))
          printf("\n $%s\t\t%s\t\t\tE->E*E", stack, ip_sym);
        flag = 1;
      }

      if (!strcmp(stack, "E") && ip_ptr == len) {
        printf("\n $%s\t\t%s$\t\t\tAccept", stack, ip_sym);
        printf("\n Name: Suravi Shrestha  Roll no: 33\n");
        exit(0);
      }

      if (flag == 0) {
        printf("\n%s\t\t\t%s\t\t Reject", stack, ip_sym);
      }
      return;
    }
```

OUTPUT:

```
                Shift Reduce Parser

 Grammar

 E->E+E
 E->E/E
 E->E*E
 E->a/b
 Enter the input symbol:        a+a*b/a

        Stack implementation table
 Stack           input symbol           action
-----------             -------------          ---------

$               a+a*b/a$                       ---
 $a              +a*b/a$                       Shift a
 $E              +a*b/a                E->a
 $E+              a*b/a$                       Shift +
 $E+a              *b/a$                       Shift a
 $E+E              *b/a                E->a
 $E*                b/a$                       Shift *
 $E*b                /a$                       Shift b
 $E*E                /a$                       E->b
 $E/                  a$                       Shift /
 $E/a                  $                       Shift a
 $E/E                                 E->a
 $E                    $                       Accept
 Name: Suravi Shrestha  Roll no: 33
```

LAB: 12

TITLE: IMPLEMENTATION OF SYMBOL TABLE

OBJECTIVE: Write a C program to implement Symbol Table.

LOGIC/ALGORITHM

1. start the program for performing insert, display, delete, search and modify option in symbol table
2. Define the structure of the symbol table.
3. Enter the choice for performing the operation in the symbol table.
4. If the entered choice is 1, search the symbol table for the symbol to inserted. if the symbol table is already present, it displays "Duplicate symbol". Else, insert the symbol table and the corresponding address in the symbol table.
5. If the entered choice is 2, the symbol present in the symbol table are displayed.
6. if the entered choice is 3, the symbol to be deleted is searched in the symbol table.
7. if it is not found in the symbol table it displays "Label not found". Else the symbol is deleted.
8. if the entered choice is 5, the symbol to be modified is searched in the symbol table. the label or address or both can be modified.

SOURCE CODE

```
#include<stdio.h>
#include<string.h>
#define null 0
int size = 0;
void insert();
void del();
int search(char lab[]);
void modify();
void display();
struct symbtab
 {
  char label[10];
  int addr;
  struct symtab *next;
 };
```

```
struct symbtab *first, *last;
void main()
{
  int op;
  int y;
  char la[10];
  //clrscr();
  do
   {
     printf("\n symbol table implementation\n");
     printf("1.Insert\n");
     printf("2. Display\n");
     printf("3. Delete\n");
     printf("4. Search\n");
     printf("5. Modify\n");
     printf("6. End\n");
     printf("Enter your option :");
     scanf("%d",&op);
     switch(op)
        {
          case 1:
          insert();
          display();
          break;
          case 2:
          display();
          break;
          case 3:
          del();
          display();
          break;
          case 4:
          printf("Enter the label to be searched");
          scanf("%s",la);
          y=search(la);
          if(y==1)
            {
             printf("The label is already in the symbol table");
```

```
          }
         else
          {
          printf("The label is not found in the symbol table");
          }
         break;
         case 5:
          modify();
          display();
          break;
         case 6:
                printf("\n Name: Suravi Shrestha  Roll no: 33");
         break;
      }
    }
    while(op < 6);
    //getch();
}
void insert()
   {
    int n;
    char l[10];
    printf("Enter the label:");
    scanf("%s",l);
    n=search(l);
    if(n==1)
     {
      printf("The label already exits.Duplicate can't be inserted.");
     }
     else
     {
       struct symbtab *p;
       p=malloc(sizeof(struct symbtab));
       strcpy(p->label,l);
       printf("Enter the address :");
       scanf("%d",&p->addr);
       p->next=null;
       if(size==0 )
```

```
        {
         first = p;
         last =p;
        }
       else
        {
          last->next=p;
          last=p;
        }
       size++;        }
  }
 void display()
  {
  int i;
  struct symbtab *p;
  p=first;
  printf("Label\t Address\n");
  for(i=0;i<size;i++)
    {
     printf("%s\t%d\n",p->label,p->addr);
     p=p->next;
    }
  }
int search(char lab[])
  {
   int i, flag=0;
   struct symbtab *p;
   p=first;
   for(i=0; i<size; i++)
     {
      if(strcmp(p->label,lab)==0)
       {
        flag = 1;
       }
       p=p->next;
     }
     return flag;
  }
```

```
void modify()
  {
   char l[10], nl[10];
   int add, choice, i, s;
   struct symbtab *p;
   p=first;
   printf("What do you want to modify?");
   printf("1. Only the label\n");
   printf("2. Only the address of a particular label");
   printf("3. Both the label and address\n");
   printf("Enter your choice: ");
   scanf("%d", &choice);
   switch(choice)
       {
        case 1:
           printf("Enter the old label\n");
           scanf("%s",l);
           printf("Enter the new label\n");
           scanf("%s",nl);
           s=search(l);
           if(s==0)
            {
             printf("No such label");
            }
           else
             {
             for(i=0;i<size;i++)
               {       if(strcmp(p->label,l)==0)
                  {
                   strcpy(p->label,nl);
                  }
                 p=p->next;
               }
             }
             break;
            case 2:
              printf("Enter the label whose address is to be modified\n");
              scanf("%s",l);
```

```
printf("Enter the new address\n");
scanf("%d",&add);
s=search(l);
if(s==0)
 {
  printf("No such label");
 }
else
 {
  for(i=0; i<size;i++)
   {
    if(strcmp(p->label,l)==0)
      {
       p->addr=add;
      }
     p=p->next;
   }
 }
 break;

 case 3:
 printf("Enter the old label:");
 scanf("%s",l);
 printf("Enter the new label");
 scanf("%s",nl);
 printf("Enter the new address :");
 scanf("%d",&add);
 s=search(l);
 if(s==0)
   {
    printf("No such label");
   }
  else
   {
    for(i=0; i<size; i++)
      {
       if(strcmp(p->label,l)==0)
         {
```

```
                    strcpy(p->label,nl);
                    p->addr=add;
                   }
                  p=p->next;
                 }
               }
              break;
        }
   }
void del()
  {
    int a;
    char l[10];
    struct symbtab *p, *q;
    p=first;
    printf("Enter the label to be deleted\n");
    scanf("%s",l);
    a = search(l);
    if(a==0)
     {
      printf("Label not found\n");
     }
     else
      {
       if(strcmp(first->label,l)==0)
        {
         first = first->next;
        }
       else if(strcmp(last->label,l)==0)
        {
         q = p->next;
         while(strcmp(q->label,l)!=0)
          {
           p= p->next;
           q= q->next;
          }
         p->next=null;
         last=p;
```

```
        }

      else
       {
        q=p->next;
        while(strcmp(q->label,l)!=0)
        {
          p =p->next;
          q =q->next;
        }
        p->next = q->next;
       }
       size--;
     }

}
```

OUTPUT:

- **Insert**

```
 symbol table implementation
1.Insert
2. Display
3. Delete
4. Search
5. Modify
6. End
Enter your option :1
Enter the label:A
Enter the address :100
Label    Address
A       100

 symbol table implementation
1.Insert
2. Display
3. Delete
4. Search
5. Modify
6. End
Enter your option :1
Enter the label:B
Enter the address :200
Label    Address
A       100
B       200
```

- **Display**

```
Enter your option :2
Label    Address
A       100
B       200
```

- **Search**

```
Enter your option :4
Enter the label to be searchedB
The label is already in the symbol table
 symbol table implementation
```

- **Delete**

```
Enter your option :3
Enter the label to be deleted
B
Label    Address
A       100
```

- **Modify**

```
Enter your option :5
What do you want to modify?1. Only the label
2. Only the address of a particular label3. Both the label and address
Enter your choice: 3
Enter the old label:A
Enter the new labelD
Enter the new address :400
Label    Address
D       400
```

- **End**

```
Enter your option :6

 Name: Suravi Shrestha  Roll no: 33
```

LAB: 13

TITLE: IMPLEMENTATION OF SLR(1) GRAMMER.

OBJECTIVE: Implement the given grammer in JFLAP and perform stack implementation.

i. Grammer:

E->E+T|t

T->T*F|F

F->(E)|a

Augumented grammer

E'->E

E->E+T|t

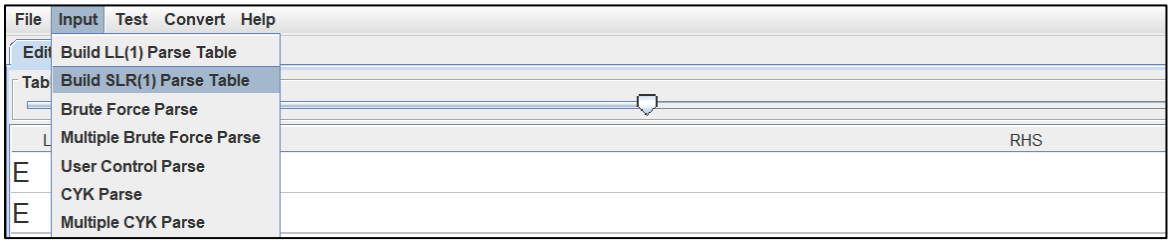
T->T*F|F

F->(E)|a

Input in jflap:

LHS		RHS
E	→	E+T
E	→	T
T	→	T*F
T	→	F
F	→	(E)
F	→	a

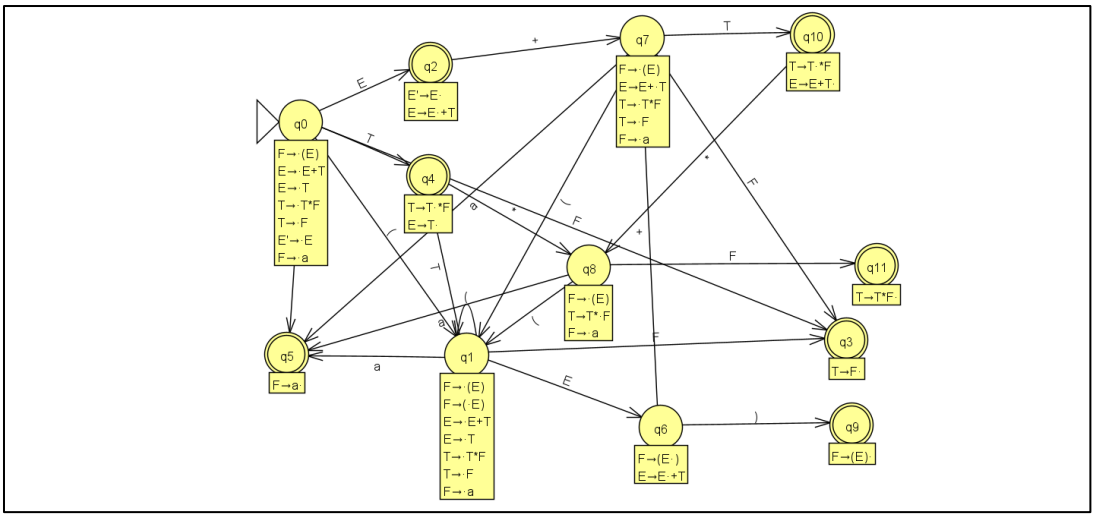
GOTO: Input->Build SLR(1) Parse Table



Computing FIRST and FOLLOW

	FIRST	FOLLOW
E	{ a, (}	{ $,), + }
F	{ a, (}	{ $,), *, + }
T	{ a, (}	{ $,), *, + }

Cannonical collection :



SLR parsing table:

	(	)	*	+	a	$	E	F	T
0	s1				s5		2	3	4
1	s1				s5		6	3	4
2				s7		acc			
3		r4	r4	r4		r4			
4		r2	s8	r2		r2			
5		r6	r6	r6		r6			
6		s9		s7					
7	s1				s5			3	10
8	s1				s5			11	
9		r5	r5	r5		r5			
10		r1	s8	r1		r1			
11		r3	r3	r3		r3			

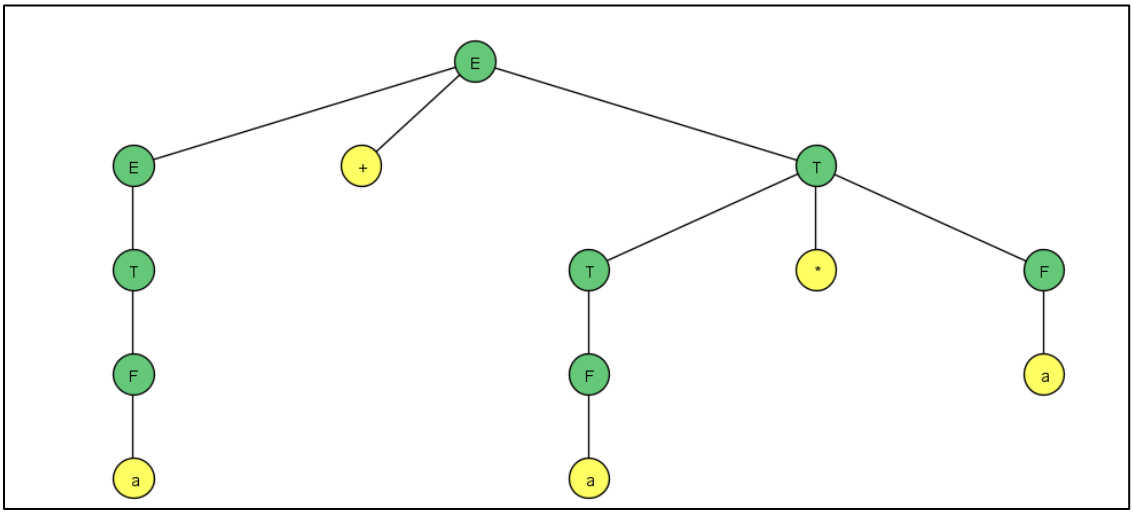
Stack implementation: a+a*a

STACK	INPUT	ACTION
$0	a+a*a$	Shift->S5
$0a5	+a*a$	Reduce by F->a

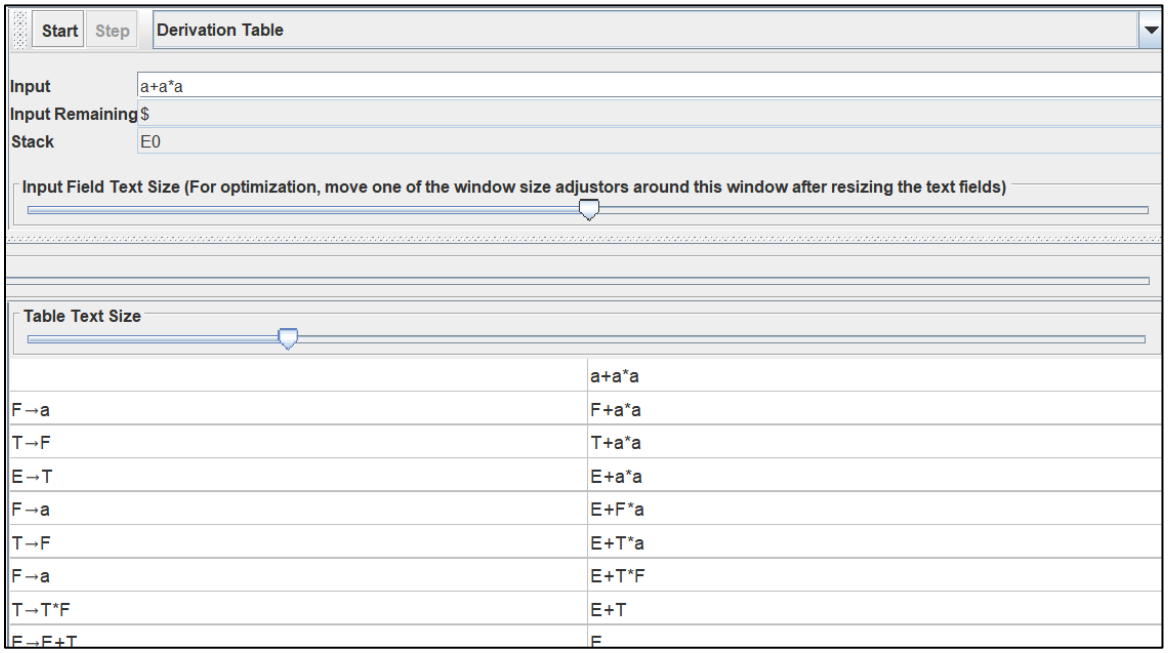
$0F3	+a*a$	Reduce by T->F
$0T4	+a*a $	Reduce by E->T
$0E2	+a*a $	Shift->S6
$0E2+7	a*a $	Shift->S5
$0E2+7a5	*a $	Reduce by F->a
$0E2+73	*a $	Reduce by T->F
$0E2+7T10	*a $	Shift->S7
$0E2+7T10*8	a$	Shift->S5
$0E2+7T10*8a5	$	Reduce by F->a
$0E2+7T10*8F11	$	Reduce by T->T*F
$0E2+7T10	$	Reduce by E->E+T
$0E2	$	Accepted

Parse tree

input: a+a*a



Derivation Table:



ii. Grammar:

S->AA
A->Aa
a->b

Augmented grammar

S'->S
S->AA
S->Aa
S->b

Input in jflap:

Editor

Table Text Size

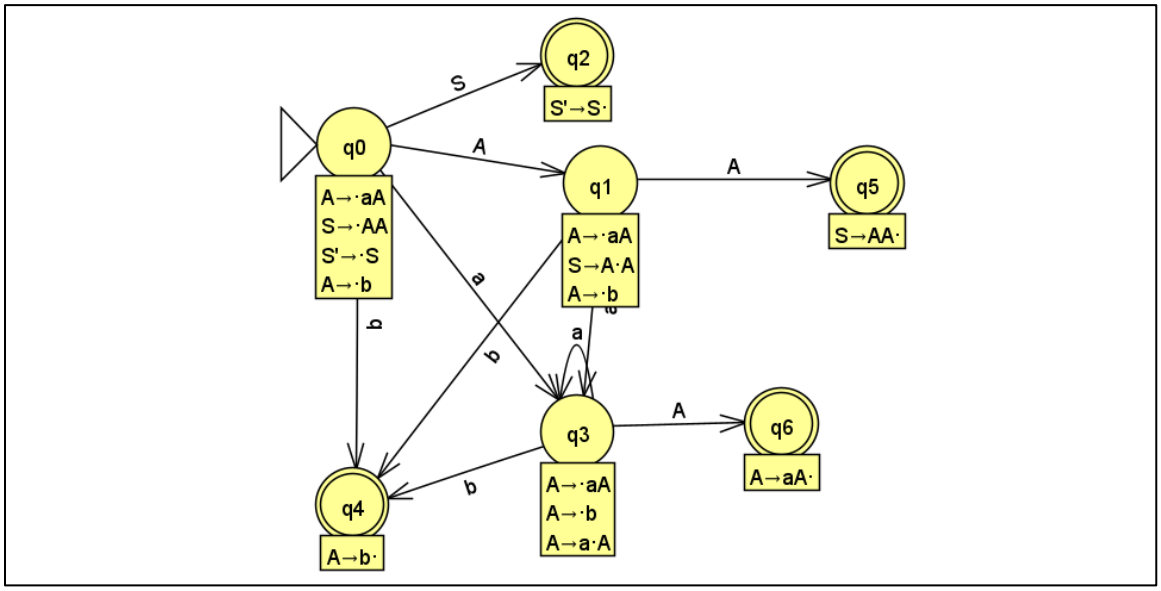
LHS		RHS
S	→	AA
A	→	aA
A	→	b

Computing first and follow

Parse table complete. Press "parse" to use it.

	FIRST	FOLLOW
A	{ a, b }	{ a, b, $ }
S	{ a, b }	{ $ }

Cannonical colln :



SLR(1) parsing table:

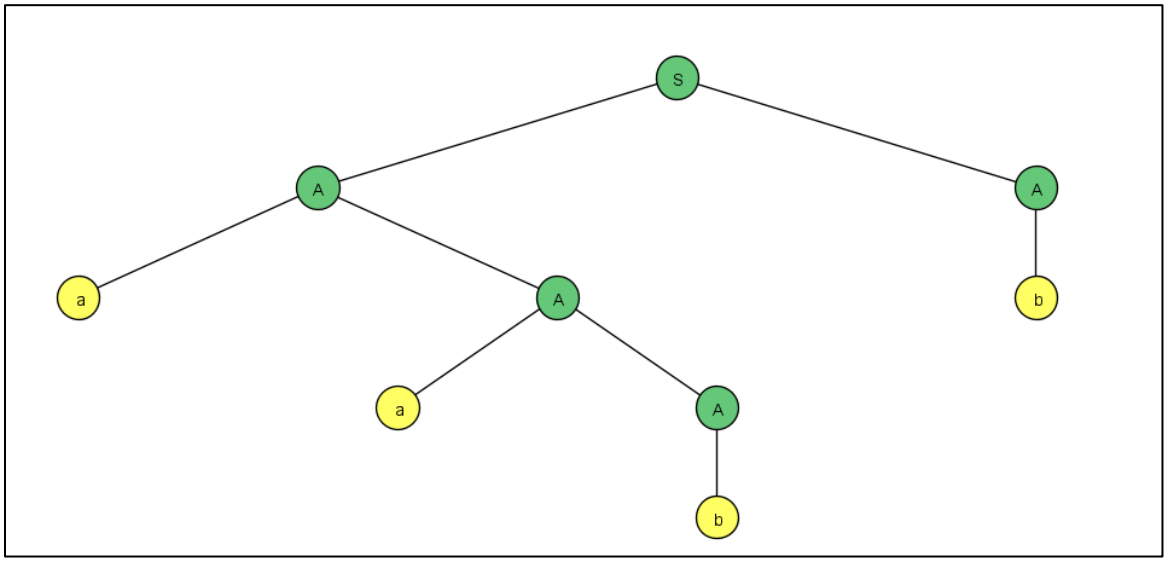
	a	b	$	A	S
0	s3	s4		1	2
1	s3	s4		5	
2			acc		
3	s3	s4		6	
4	r3	r3	r3		
5			r1		
6	r2	r2	r2		

Stack Implementation for aabb

STACK	INPUT	ACTION
$0	aabb$	Shift =>S3
$0a3	abb$	Shift =>S3
$0a3a3	bb$	Shift =>S4
$0a3a3b4	b$	Reduce by A->b
$0a3a3A6	b$	Reduce by A->aA
$0a3A6	b$	Reduce by A->aA
$0A1	b$	Shift =>S4
$0A1b4	$	Reduce by A->b
$0A1A5	$	Reduce by S->AA
$0S2	$	Accept
S0		Accept

Parse tree

Input: aabb



Derivation table:

Table Text Size

	aabb
A→b	aaAb
A→aA	aAb
A→aA	Ab
A→b	AA
S→AA	S

LAB: 14

TITLE: IMPLEMENTATION OF LL(1) GRAMMER.

OBJECTIVE: Implement the given grammar in JFLAP and perform stack implementation.

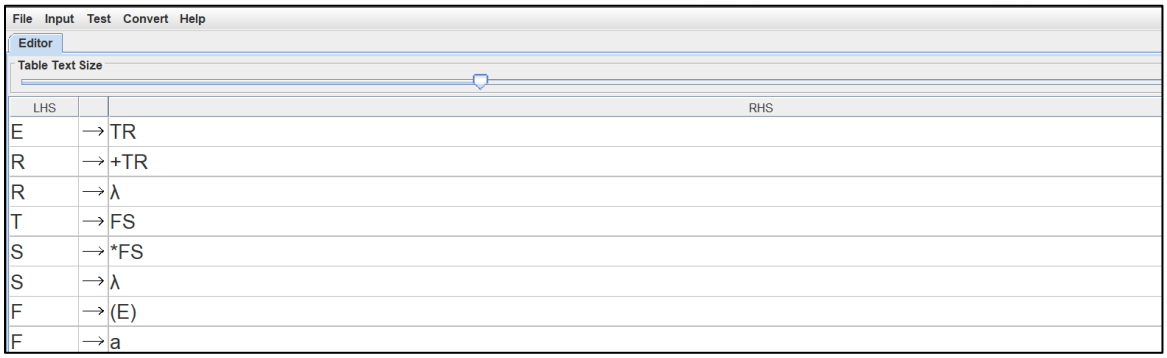
i. Grammar:

E->E+T |T
T->T*F |F
F->(E) |a

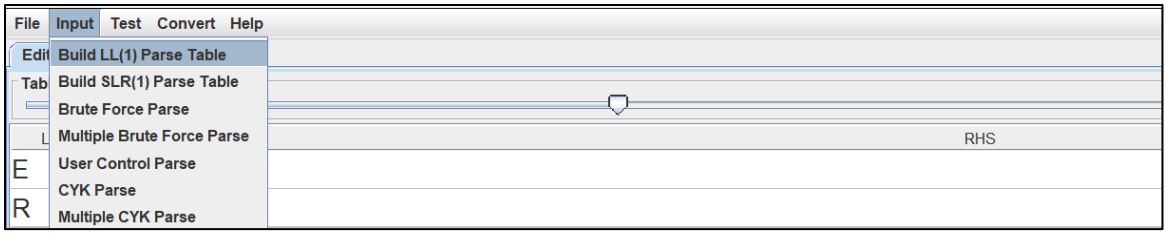
Removing left recursion

E->TR
R->+TR | λ
R->FS
S->*FS| λ
F->(E) | a

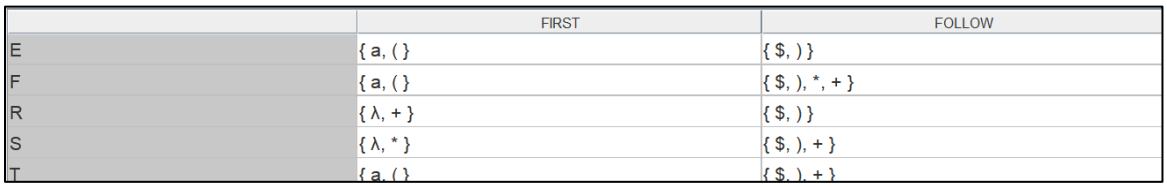
Input in JFLAP:



GOTO: Input->Build LL(1) Parse Table:



Computing first and follow:



LL(1) Parsing Table:

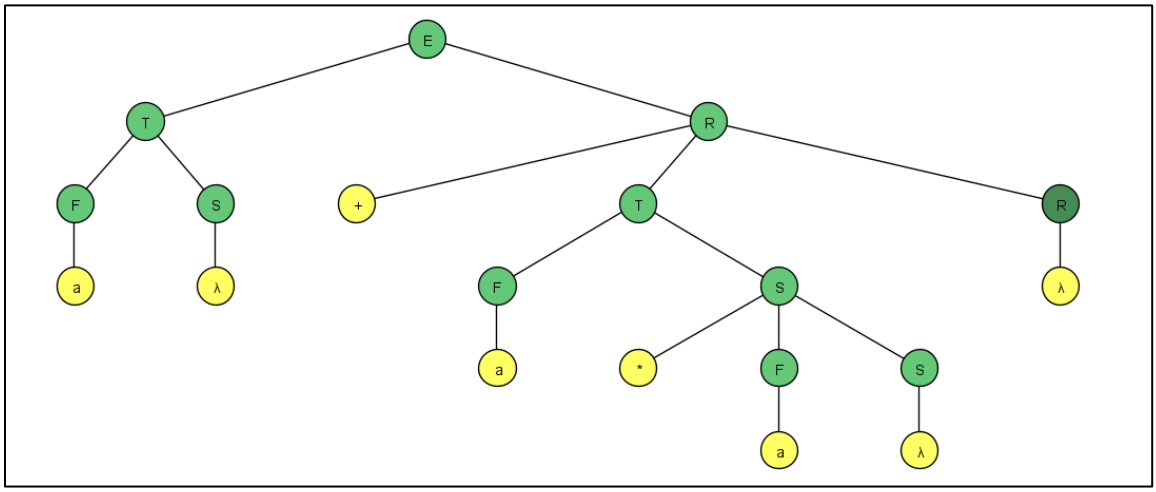
	(	)	*	+	a	$
E	TR				TR	
F	(E)				a	
R		λ		+TR		λ
S		λ	*FS	λ		λ
T	FS				FS	

Stack Implementation

Input: a+a*a

STACK	INPUT	OUTPUT
E$	a+a*a$	
TE'$	a+a*a$	E→TE'
FT'E'$	a+a*a$	T→FT'
aT'E'$	a+a*a$	F→a
T'E'$	+a*a$	—
E'$	+a*a$	T'→ϵ
+TE'$	+a*a$	E'→+TE'
TE'$	a*a$	—
FT'E'$	a*a$	T→FT'
aT'E'$	a*a$	F→a
T'E'$	*a$	—
*FT'E'$	*a$	T'→*FT'
FT'E'$	a$	—
aT'E'$	a$	F→a
T'E'$	$	—
E'$	$	T'→ϵ
$	$	T'→ϵ
$	$	Accept.

Parse Tree:



Derivation Table:

Start Step Derivation Table

Input a+a*a

Input Remaining $

Stack

Input Field Text Size (For optimization, move one of the window size adjustors around this window after resizing the text fields)

Table Text Size

	E
E→TR	TR
T→FS	FSR
F→a	aSR
S→λ	aR
R→+TR	a+TR
T→FS	a+FSR
F→a	a+aSR
S→*FS	a+a*FSR
F→a	a+a*aSR
S→λ	a+a*aR
R→λ	a+a*a

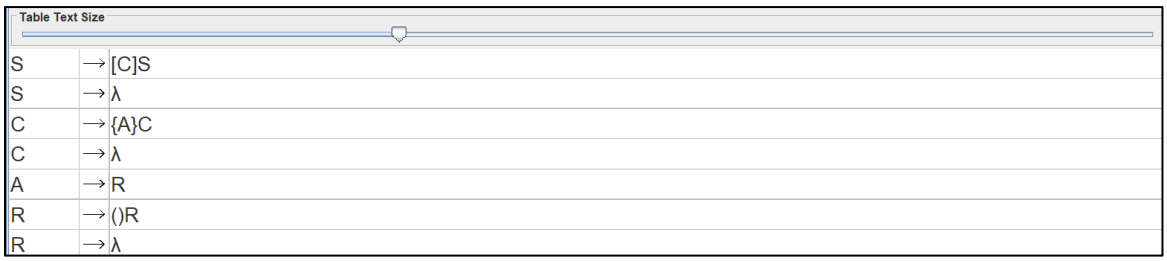
ii. Grammar:

S->[C]S|e
C->{A}C|e
A->A()|E

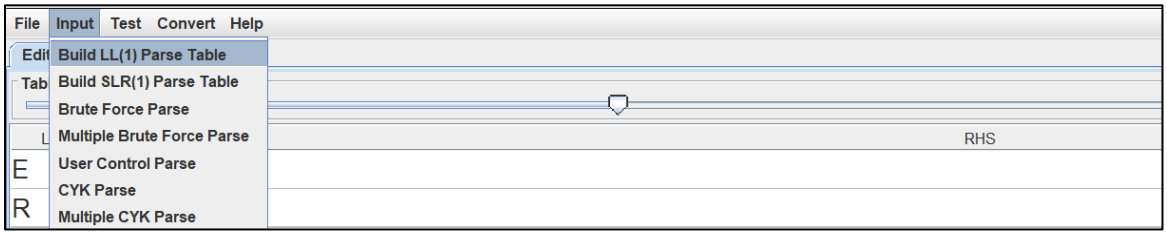
Removing left recursion

S->[C]S
S->LAMDA
C->{A}C
C->LAMDA
A->R
R->()R
R->LAMDA

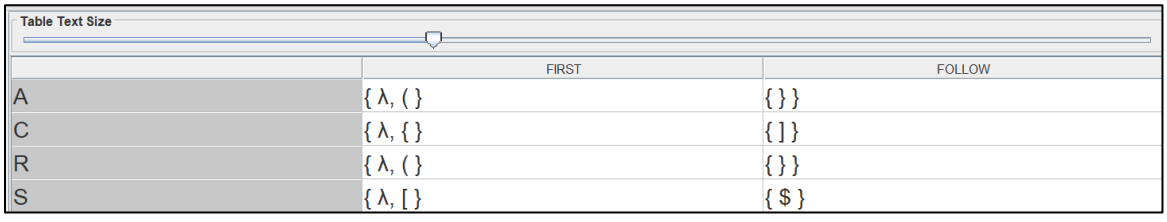
Input in jflap:



GOTO : Input->Build LL(1) Parse Table



Computing first and follow:



LL(1) Parsing Table:

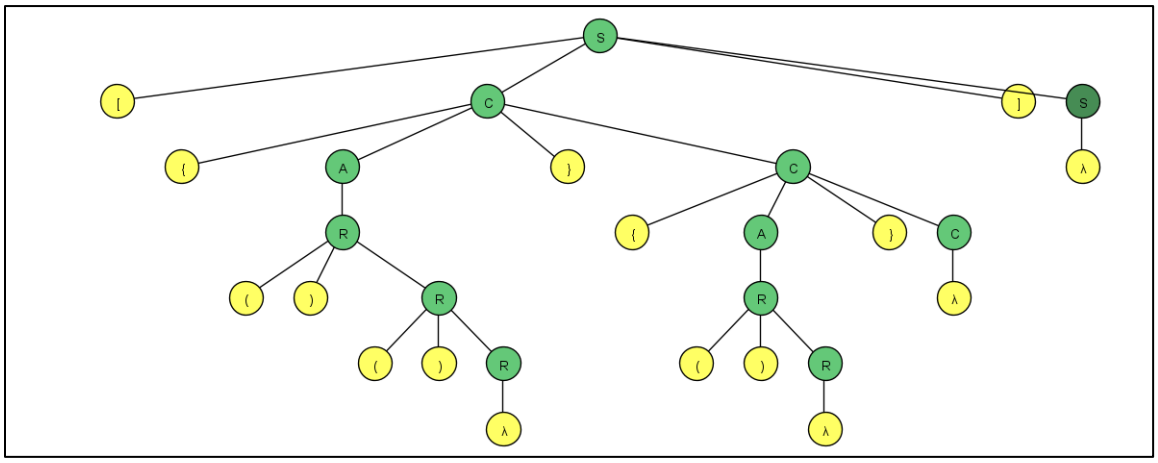
	(	)	[	]	{	}	$
A	R					R	
C				λ	{A}C		
R	()R					λ	
S			[C]S				λ

Stack Implementation of : [{()()}{()}]

STACK	INPUT	OUTPUT
S$	[{()()}{()}]$	—
[C]S$	[{()()}{()}]$	S→ [C]S
C]S$	{()()}{()}]$	—
{A}C]S$	{()()}{()}]$	C→ {A}C
A}C]S$	()()}{()}]$	—
A'}C]S$	()()}{()}]$	A→ A'
()A'}C]S$	()()}{()}]$	A'→ ()A'
)A'}C]S$	)()}{()}]$	—
A'}C]S$	()}{()}]$	—
()A'}C]S$	()}{()}]$	A'→ ∈
)A'}C]S$	)}{()}]$	—
A'}C]S$	}{()}]$	C→ {A}C
}C]S$	{()}]$	—
C]S$	()}]$	A→ A'
{A}C]S$	()}]$	A'→ ()A'
()A'}C]S$	)}]$	—
)A'}C]S$	}]$	—
A'}C]S$	}]$	A'→ ∈
}C]S$	]$	—
C]S$	]$	C→ ∈
]S$	]$	—
S$	$	S→ ∈
$	$	Accept
$	$	

Parse Tree:

Input: [{()()}{()}]



Derivation Table:

Start Step Derivation Table

Input [{()()}{()}]

Input Remaining $

Stack

Input Field Text Size (For optimization, move one of the window size adjustors around this window after resizing the text fields)

Table Text Size

	S
S→[C]S	[C]S
C→{A}C	[{A}C]S
A→R	[{R}C]S
R→()R	[{()R}C]S
R→()R	[{()()R}C]S
R→λ	[{()()}C]S
C→{A}C	[{()()}{A}C]S
A→R	[{()()}{R}C]S
R→()R	[{()()}{()R}C]S
R→λ	[{()()}{()}C]S
C→λ	[{()()}{()}]S
S→λ	[{()()}{()}]

LAB: 15

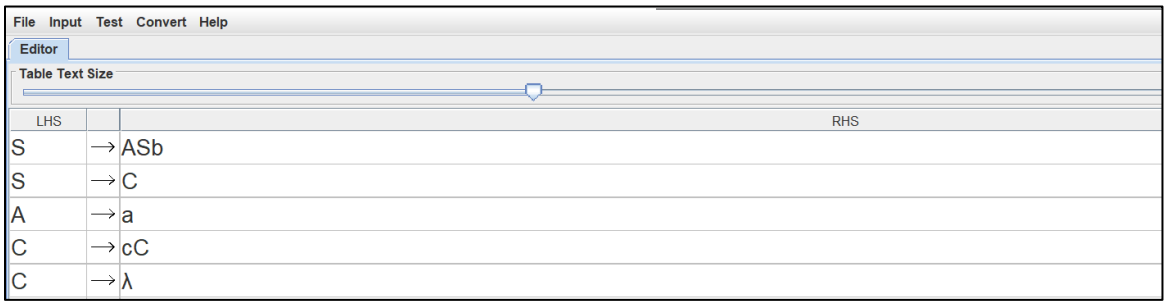
TITLE: IMPLEMENTATION OF BRUTE FORCE PARSER

OBJECTIVE: Illustrate Brute Force Parser of following grammar in JFLAP.

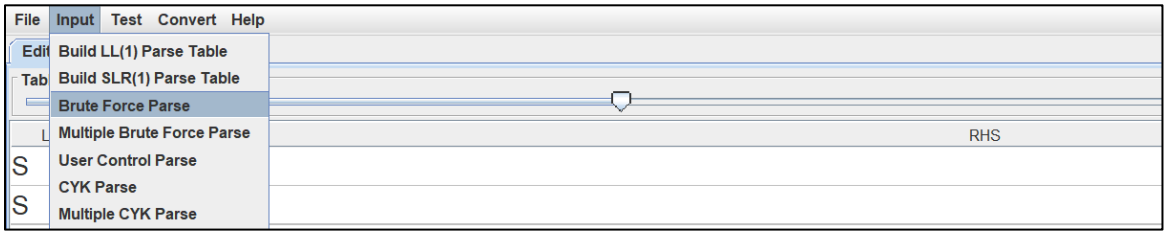
i. Grammar

S->ASb
S->C
A->a
C->cC
C->Epsilon

Input in jflap:



Goto: Input-> Brute Force Parse



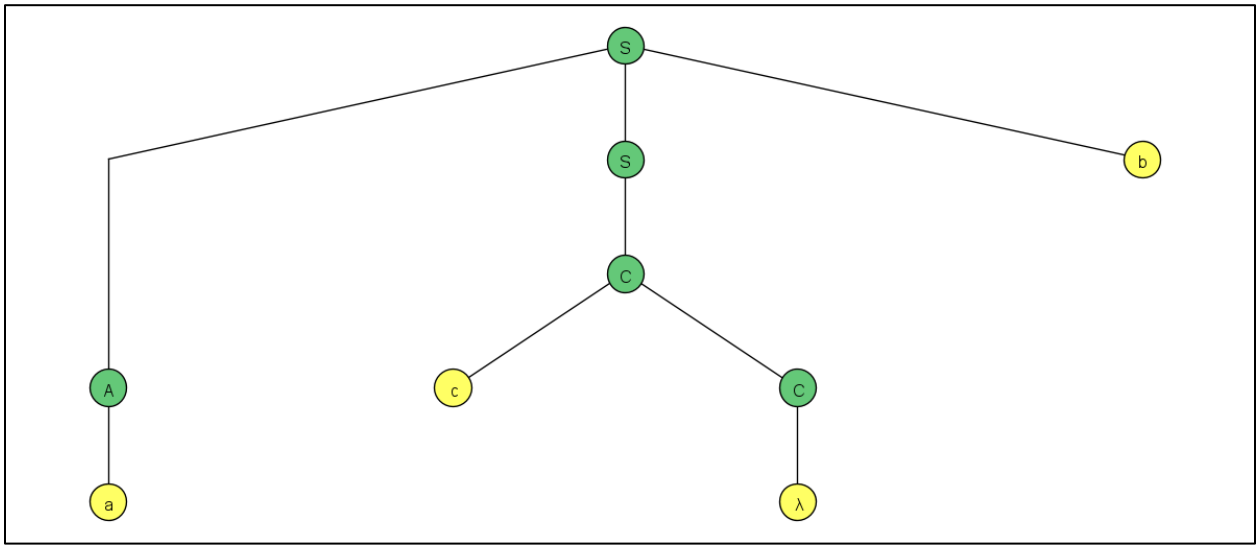
Stack Implementation: acb

STACK	INPUT	OUTPUT
S$	acb$	-
ASb$	acb$	S->ASb

aSb$	acb$	A->a
Sb$	cb$	-
Cb$	cb$	S->C
cCb$	cb$	C->cC
Cb$	b$	-
b$	b$	C->λ
$	$	-
$	$	Accepted

Parse tree

Input: acb



Derivation Table

Table Text Size

	S
S→ASb	ASb
S→C	ACb
C→cC	AcCb
A→a	acCb
C→λ	acb

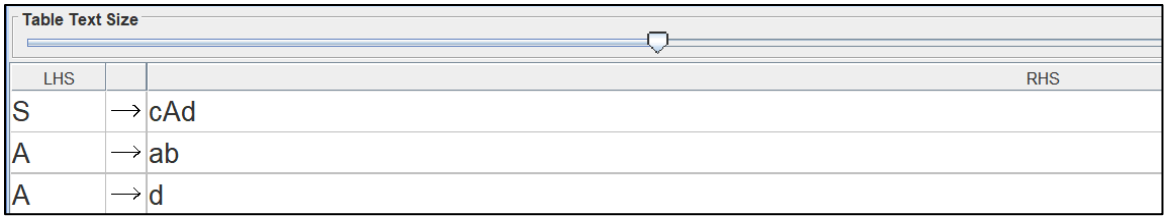
ii. Grammar:

S->cAd

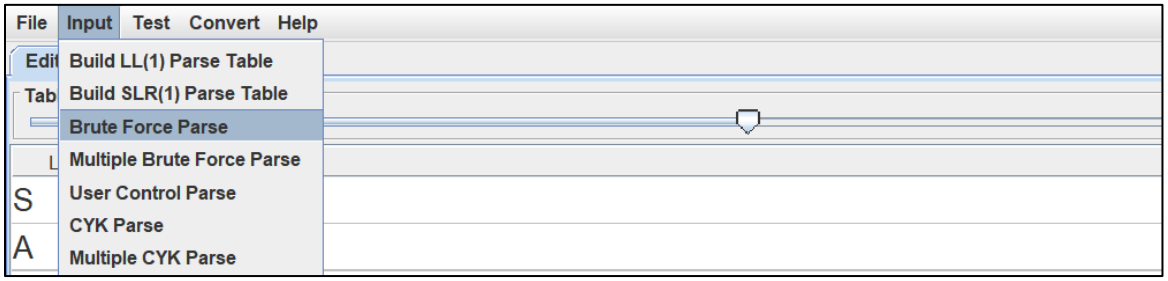
A->ab

A->d

Input grammar in jflap:



Goto: Input-> Brute Force Parse

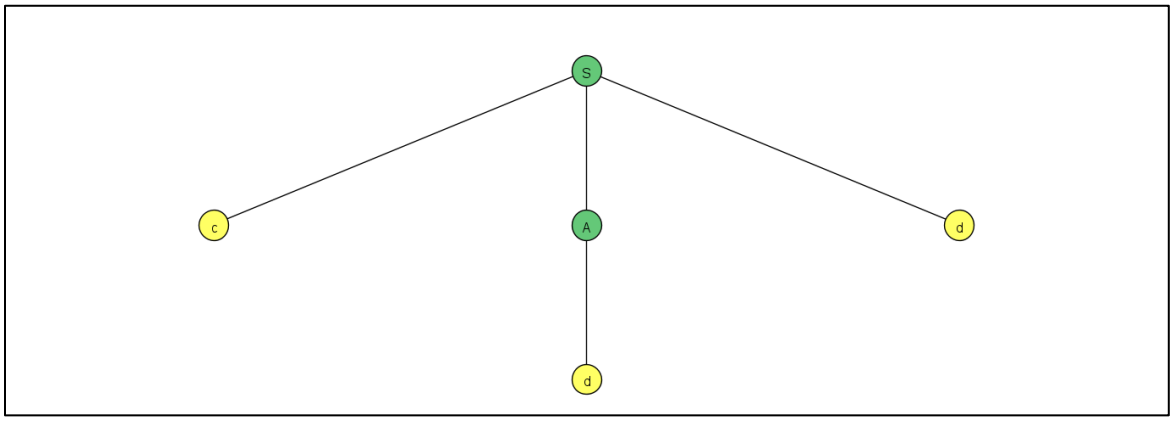


Stack Implementation for cdd

STACK	INPUT	OUTPUT
S$	Cdd$	-
cAd$	Cdd$	S->cAd
Ad$	Dd$	-
dd$	Dd$	A->d
d$	d$	-
$	$	-
$	$	Accepted

Parse Tree

Input: cdd



Derivation Tree:

Table Text Size

	S
S→cAd	cAd
A→d	cdd