Practical-10

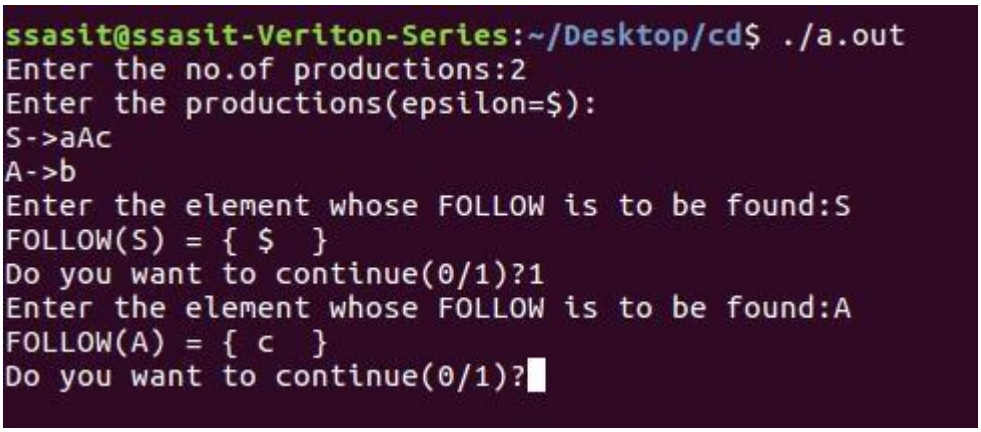
Aim: Finding "Follow" set Input: The string consists of grammar symbols. Output: The Follow set for a given string. Explanation: The student has to assume a typical grammar. The program when run will ask for the string to be entered. The program will find the Follow set of the given string.

Code:

```
#include<stdio.h> #include<string.h> int n,m=0,p,i=0,j=0; char a[10][10],f[10]; void follow(char c); void
first(char c); int main()
{
int i,z; char
c,ch;
printf("Enter the no.of productions:");
scanf("%d",&n);
printf("Enter the productions(epsilon=$):\n");
for(i=0;i<n;i++)
  scanf("%s%c",a[i],&ch);

do{
m=0;
printf("Enter the element whose FOLLOW is to be found:");
  scanf("%c",&c);
  follow(c);
  printf("FOLLOW(%c) = { ",c);
  for(i=0;i<m;i++)
   printf("%c ",f[i]);
  printf(" }\n");
  printf("Do you want to continue(0/1)?");
  scanf("%d%c",&z,&ch);
}
while(z==1);
}
void follow(char c)
{

if(a[0][0]==c)f[m++]='$'; for(i=0;i<n;i++)
{
  for(j=2;j<strlen(a[i]);j++)
  {
   if(a[i][j]==c)
   {
    if(a[i][j+1]!='\0')first(a[i][j+1]);
    if(a[i][j+1]=='\0'&&c!=a[i][0])
     follow(a[i][0]);

   }
  }
}
}
void first(char c)
{ int k; if(!(isupper(c)))f[m++]=c;
     for(k=0;k<n;k++)
```

```
        {
        if(a[k][0]==c)
        {
        if(a[k][2]=='$') follow(a[i][0]); else
        if(islower(a[k][2]))f[m++]=a[k][2];
        else first(a[k][2]);
        }
        }

}
```

Output:



Practical-11

Aim: Implement a C program for constructing LL(1) parsing.

Code:

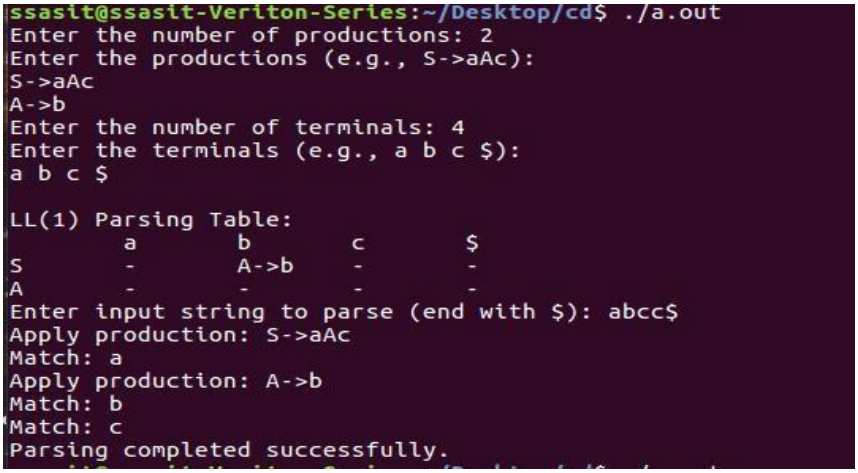
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h> #include
<ctype.h>

#define MAX 10
#define MAX_TERMINALS 10
#define MAX_NON_TERMINALS 10
#define MAX_PRODUCTIONS 10 #define
MAX_INPUT 100

char productions[MAX_PRODUCTIONS][MAX]; char
first[MAX_NON_TERMINALS][MAX_TERMINALS],
follow[MAX_NON_TERMINALS][MAX_TERMINALS]; char
parsing_table[MAX_NON_TERMINALS][MAX_TERMINALS][MAX]; char
non_terminals[MAX_NON_TERMINALS], terminals[MAX_TERMINALS]; int
num_productions, num_non_terminals, num_terminals;

// Helper function to find if a symbol is terminal int
isTerminal(char symbol) {
   return !(symbol >= 'A' && symbol <= 'Z');
}

// Function to add a symbol to the set (to prevent duplicates)
void addToSet(char *set, char symbol) {
 if (!strchr(set, symbol)) {
          int len = strlen(set);
          set[len] = symbol;
          set[len + 1] = '\0';
   }
}

// Function to find FIRST set for a non-terminal void
findFirst(char non_terminal, char *result) {
   for (int i = 0; i < num_productions; i++) {
   if (productions[i][0] == non_terminal) {
   char *rhs = productions[i] + 3; if
      (isTerminal(rhs[0])) {
            addToSet(result, rhs[0]);
         } else {
            findFirst(rhs[0], result);
         }
      }
   }
}
// Function to find FOLLOW set for a non-terminal
void findFollow(char non_terminal, char *result) {
if (non_terminal == non_terminals[0]) {
   addToSet(result, '$');
   }
```

```
    for (int i = 0; i < num_productions; i++) {
     char *rhs = productions[i] + 3;
        for (int j = 0; rhs[j] != '\0'; j++) {
        if (rhs[j] == non_terminal) {
           if (rhs[j + 1] != '\0') {
                   if (isTerminal(rhs[j + 1])) {
                  addToSet(result, rhs[j + 1]);
                  } else {
                   char temp[MAX] = "";
                      findFirst(rhs[j + 1], temp);
                       for (int k = 0; temp[k] != '\0'; k++) {
                          if (temp[k] != 'e') {
                                addToSet(result, temp[k]);
                          }
                      }
                  }
              } else {
                 findFollow(productions[i][0], result);
              }
           }
        }
    }
}

// Function to construct LL(1) parsing table void
constructParsingTable() {
    for (int i = 0; i < num_non_terminals; i++) {
        for (int j = 0; j < num_terminals; j++) {
           strcpy(parsing_table[i][j], "");
        }
    }

    for (int i = 0; i < num_productions; i++)
        { char non_terminal = productions[i][0];
        char *rhs = productions[i] + 3; char
        first_set[MAX] = "";

        if (isTerminal(rhs[0])) {
           addToSet(first_set, rhs[0]);
        } else {
           findFirst(rhs[0], first_set);
        }

        for (int j = 0; first_set[j] != '\0'; j++) {
           if (first_set[j] != 'e') {
              for (int k = 0; k < num_terminals; k++) {
                 if (first_set[j] == terminals[k]) {
                    strcpy(parsing_table[non_terminal - 'A'][k], productions[i]); }
              }
           }
        }

        if (strchr(first_set, 'e')) {
           char follow_set[MAX] = "";
```

```
findFollow(non_terminal, follow_set);
            for (int j = 0; follow_set[j] != '\0'; j++) {
                for (int k = 0; k < num_terminals; k++) {
                if (follow_set[j] == terminals[k]) {
                    strcpy(parsing_table[non_terminal - 'A'][k], productions[i]); }
                }
            }
        }
    }
}

    // Function to print LL(1) parsing table
    void printParsingTable() {
    printf("\nLL(1) Parsing Table:\n\t");
    for (int i = 0; i < num_terminals; i++)
{printf("%c\t", terminals[i]);
    }
    printf("\n");
    for (int i = 0; i < num_non_terminals; i++)
        { printf("%c\t", non_terminals[i]);
    for (int j = 0; j < num_terminals; j++) {
     if (strcmp(parsing_table[i][j], "") != 0)
        { printf("%s\t", parsing_table[i][j]);
            } else {
                printf("-\t");
            }
        }
        printf("\n");
    }
}

// Function to parse the input string using the LL(1) parsing table void
parseInputString(char *input) {
    char stack[MAX_INPUT];
    int top = -1; stack[++top] = '$'; stack[++top] =
    non_terminals[0]; // Start symbol

    int i = 0; while (top >= 0) { char
    stackTop = stack[top]; char
    currInput = input[i];
        if (stackTop == currInput)
            { printf("Match: %c\n", currInput);
             top--;
            i++;
        } else if (stackTop == '$') {
             printf("Parsing completed successfully.\n");
             return;
        } else { int row = stackTop -
            'A'; int col = -1;
            for (int j = 0; j < num_terminals; j++) {
            if (currInput == terminals[j]) { col = j;
                break;
                }
            }
```

```
        if (col == -1 || strcmp(parsing_table[row][col], "") == 0)
           { printf("Error: Invalid input at %c\n", currInput);
                return;
        } else {
         printf("Apply production: %s\n", parsing_table[row][col]);
         top--;
            char *prod = parsing_table[row][col] + 3;
             for (int j = strlen(prod) - 1; j >= 0; j--) {
            if (prod[j] != 'e')
            // Ignore epsilon (empty production) stack[++top] =
            prod[j];
            }
        }
      }
   }
}

int main() {
   printf("Enter the number of productions: ");
   scanf("%d", &num_productions);

   printf("Enter the productions (e.g., S->aAc):\n");
    for (int i = 0; i < num_productions; i++) {
    scanf("%s", productions[i]);
   non_terminals[i] = productions[i][0];
   }

   num_non_terminals = num_productions;

   // Input the terminals
   printf("Enter the number of terminals: ");
   scanf("%d", &num_terminals);
   printf("Enter the terminals (e.g., a b c $):\n");
   for (int i = 0; i < num_terminals; i++) {
   scanf(" %c", &terminals[i]);
   }

   // Construct the parsing table
   constructParsingTable();

   // Print the parsing table printParsingTable();

   // Input the string to parse char
   input[MAX_INPUT];
   printf("Enter input string to parse (end with $): ");
   scanf("%s", input);
   // Parse the input string parseInputString(input);

   return 0;
}
```

Output:


ssasit@ssasit-Veriton-Series:~/Desktop/cd$ ./a.out
Enter the number of productions: 2
Enter the productions (e.g., S->aAc):
S->aAc
A->b
Enter the number of terminals: 4
Enter the terminals (e.g., a b c $):
a b c $
LL(1) Parsing Table:
a b c $
S - A->b - -
A - - - -
Enter input string to parse (end with $): abcc$
Apply production: S->aAc
Match: a
Apply production: A->b
Match: b
Match: c
Parsing completed successfully.