Compiler Design Lab – 4

Lex Programming

Name: SreeDananjay S

Reg no: 21BAI1807

Date: 26/02/2023

Program to find the number of vowels and consonants

Code:

```
%{
#include<stdio.h>
#include<string.h>
int vowels = 0;
int consonants = 0;
%}

%%
[aAeEiIoOuU]   { vowels++; }
[A-Za-z]      { consonants++; }
.           ;
"\n" {printf("Total Number of Vowels : %d\nTotal Number of Consonants : %d\n",
vowels,consonants); vowels = 0;consonants = 0;}
%%
int yywrap(void){}
int main() {
   yylex();
   return 0;
}
```

Output:

```
student@614:~/Desktop$ lex prog1.l
student@614:~/Desktop$ gcc lex.yy.c
student@614:~/Desktop$ ./a.out
This program counts the number of vowels and consonants
Total Number of Vowels : 15
Total Number of Consonants : 32
```

Program to count the number of spaces and words and print the vowels present.

Code:

```
%{
#include<stdio.h>
#include<string.h>
char vowels[50];
int i=0;
int j=0;
char consonants[50];
int spaces = 0;
int words = 0;
%}

%%



[aAeEiIoOuU]   { vowels[i] = yytext[0];i++;}
[A-Za-z]       { consonants[j] = yytext[0];j++;}
[ ]          { spaces++;}
.            ;
"\n" {printf("Total Number of Vowels : %s\nTotal Number of Consonants : %s\nTotal
Number of Spaces : %d\nTotal Number of Words : %d\n",
vowels,consonants,spaces,(spaces+1));spaces = 0;words = 0;}
%%
int yywrap(void){}
int main() {
   yylex();
   return 0;
}
```

Output:

```
student@614:~/Desktop$ lex prog2.l
student@614:~/Desktop$ gcc lex.yy.c
student@614:~/Desktop$ ./a.out
This program counts the prints the vowels and consonants and number of words and
 spaces
Total Number of Vowels : ioaoueieoeaooaaueooaae
Total Number of Consonants : Thsprgrmcntsthprntsthvwlsndcnsnntsndnmbrfwrdsndspcs
Total Number of Spaces : 14
Total Number of Words : 15
```

Program to create a lexical analyzer using lex

Code:

```
%{
#include <stdio.h>
%}
%%
```

```
[0-9]+                    { printf("NUMBER: %s\n", yytext); }
"int"|"printf"|"float"          { printf("KEYWORD: %s\n", yytext); }
[a-zA-Z][a-zA-Z0-9_]*       { printf("IDENTIFIER: %s\n", yytext); }
"=="|"!="|"<="|">="|"="|"<"|">" { printf("OPERATOR: %s\n", yytext); }
"+"|"-"|"*"|"/"             { printf("OPERATOR: %s\n", yytext); }
"("|")"|";"|","             { printf("DELIMITER: %s\n", yytext); }
[ \t\n]                   ; // ignore whitespace
.                         { printf("INVALID CHARACTER: %s\n", yytext); }
%%
int yywrap(void){}
int main() {
  yylex();
  return 0;
}
```

Output:

```
student@614:~/Desktop$ lex prog3.l
student@614:~/Desktop$ gcc lex.yy.c
student@614:~/Desktop$ ./a.out
int a=(a+b)*23;
KEYWORD: int
IDENTIFIER: a
OPERATOR: =
DELIMITER: (
IDENTIFIER: a
OPERATOR: +
IDENTIFIER: b
DELIMITER: )
OPERATOR: *
NUMBER: 23
DELIMITER: ;
```

Q4) Program to calculate First() and Follow() function:

Code:

```
#include <ctype.h>

#include <stdio.h>

#include <string.h>


// Functions used to find Follow

void grammarfollow(char, int, int);

void follow(char c);
```

```
// Function used to find First
void find_first(char, int, int);

int count, n = 0;

// Stores the final resultof the First Sets
char final_first[10][100];

// Stores the final resultof the Follow Sets
char final_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;
   count = 3;

   // Initialize the production rules for the context-free grammar
   strcpy(production[0], "E=AB");
   strcpy(production[1], "A=ilove");
   strcpy(production[2], "B=jtptutorials");
   int ff;
   char done[count];
```

```
    int ptr = -1;

    // Initializing the final_first array
    for (k = 0; k < count; k++) {
        for (ff= 0; ff< 100; ff++) {
final_first[k][ff] = '!';
        }
    }
    int point1 = 0, point2, xxx;

    for (k = 0; k < count; k++) {
        c = production[k][0];
        point2 = 0;
        xxx = 0;

        // Checking if the first of c has already been calculated
        for (ff= 0; ff<= ptr; ff++)
            if (c == done[ff])
                xxx = 1;

        if (xxx == 1)
            continue;

        // Calling function
find_first(c, 0, 0);
        ptr += 1;

        // Adding c to the calculated list
        done[ptr] = c;
        printf("\n First(%c) = { ", c);
final_first[point1][point2++] = c;

        // Printing the First Sets of the given grammar
```

```
        for (i = 0 + jm; i < n; i++) {
            int fs = 0, chk = 0;

            for (fs = 0; fs< point2; fs++) {

                if (first[i] == final_first[point1][fs]) {
                    chk = 1;
                    break;
                }
            }
            if (chk == 0) {
                printf("%c, ", first[i]);
final_first[point1][point2++] = first[i];
            }
        }
        printf("}\n");
        jm = n;
        point1++;
    }
    printf("\n");
    printf("==================================""\n\n");
    char donee[count];
    ptr = -1;

    // Initializing the final_follow array
    for (k = 0; k < count; k++) {
        for (ff= 0; ff< 100; ff++) {
final_follow[k][ff] = '!';
        }
    }
    point1 = 0;
    int land = 0;
    for (e = 0; e < count; e++) {
```

```
        ck = production[e][0];
        point2 = 0;
        xxx = 0;

        // Checking if Follow of ckhas already been calculated
        for (ff= 0; ff<= ptr; ff++)
           if (ck == donee[ff])
              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);
final_follow[point1][point2++] = ck;

        // Printing the Follow Sets of the given grammar
        for (i = 0 + km; i < m; i++) {
           int fs= 0, chk = 0;
           for (fs= 0; fs< point2; fs++) {
              if (f[i] == final_follow[point1][fs]) {
                 chk = 1;
                 break;
              }
           }
           if (chk == 0) {
              printf("%c, ", f[i]);
```

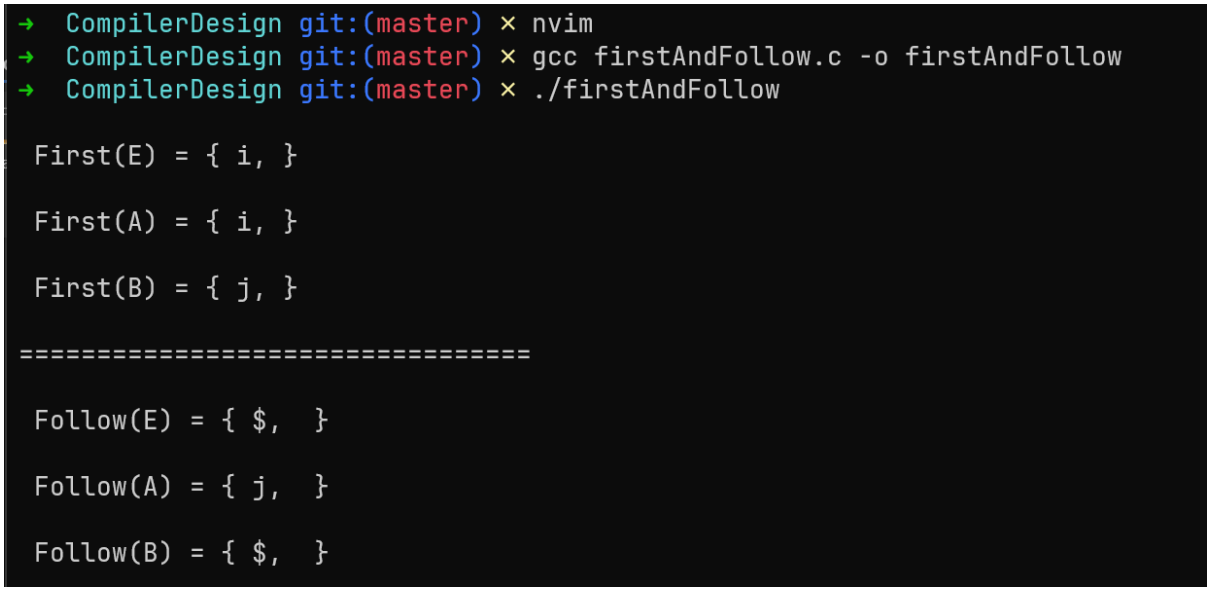
```
final_follow[point1][point2++] = f[i];
          }
       }
       printf(" }\n\n");
       km = m;
       point1++;
   }
}

void follow(char c)
{
   int i, j;

   // Adding "$" to the Follow Setof the start symbol
   if (production[0][0] == c) {
      f[m++] = '$';
   }
   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
grammarfollow(production[i][j + 1], i,
                        (j + 2));
            }

            if (production[i][j + 1] == '\0'
&& c != production[i][0]) {
 // Calculate the Follow of thenon-terminal in the L.H.S. of theproduction
               follow(production[i][0]);
            }
         }
      }
```

```
    }
}

void find_first(char c, int q1, int q2)
{
    int j;

    // The case where we will 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 the First new non-terminal we encounter after
                    // epsilon
                    find_first(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 ofthe new non-terminal we encounterat the beginning
                find_first(production[j][2], j, 3);
            }
```

```
        }
    }
}

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

    // The case where we will encountera terminal
    if (!(isupper(c)))
        f[m++] = c;
    else {
        int i = 0, j = 1;
        for (i = 0; i < count; i++) {
            if (final_first[i][0] == c)
                break;
        }

// Including the First set of the non-terminal in the Follow ofthe original query
        while (final_first[i][j] != '!') {
            if (final_first[i][j] != '#') {
                f[m++] = final_first[i][j];
            }
            else {
                if (production[c1][c2] == '\0') {
            // The case where we will reach theend of the production
                    follow(production[c1][0]);
                }
                else {
                    // Recursion to the next symbolin case we encounter a "#"
grammarfollow(production[c1][c2], c1,
                            c2 + 1);
                }
```

```
      }
      j++;
    }
  }
}
```

Output:


➜ CompilerDesign git:(master) ✗ nvim
➜ CompilerDesign git:(master) ✗ gcc firstAndFollow.c -o firstAndFollow
➜ CompilerDesign git:(master) ✗ ./firstAndFollow
First(E) = { i, }
First(A) = { i, }
First(B) = { j, }
==========
Follow(E) = { $, }
Follow(A) = { j, }
Follow(B) = { $, }