Compiler Construction

Lab 5

# Task1:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<string.h>

#define SIZE 10

int main() {

char non\_terminal;

int num;

char production[10][SIZE];

int index = 3;

printf("Enter Number of Production : ");

scanf("%d", &num);

printf("Enter the grammar as E->E-A :\n");

for (int i = 0; i<num; i++){

scanf("%s", production[i]);

}

for (int i = 0; i<num; i++){

printf("\nGRAMMAR : : : %s", production[i]);

non\_terminal = production[i][0];

if (non\_terminal == production[i][index]) {

printf(" is left recursive.\n");

}

else

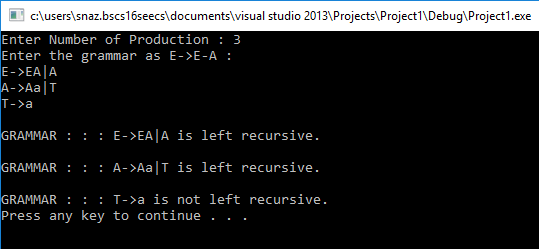
printf(" is not left recursive.\n");

index = 3;

}

system("pause");

}



# Task 2:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<string.h>

void main() {

char input[100], \*l, \*r, \*temp, tempprod[20], productions[25][50];

int i = 0, j = 0, flag = 0;

printf("Enter the productions: \n");

scanf("%s", input);

l = strtok(input, "->");

r = strtok(NULL, "->");

temp = strtok(r, "|");

while (temp) {

if (temp[0] == l[0]) {

flag = 1;

sprintf(productions[i++], "%s'->%s%s'\0\n", l, temp + 1, l);

}

else

sprintf(productions[i++], "%s->%s%s'\0\n", l, temp, l);

temp = strtok(NULL, "|");

}

sprintf(productions[i++], "%s->\356\0\n", l);

if (flag == 0)

printf("The given productions don't have Left Recursion\n");

else

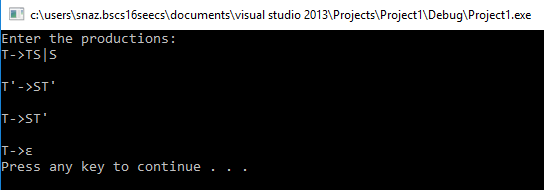
for (j = 0; j<i; j++) {

printf("\n%s\n", productions[j]);

}

system("pause");

}



# Task 3:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<conio.h>

void main()

{

char pro[10][10], first[10][10], follow[10][10], nt[10], ter[10], res[10][10][10], temp[10];

int npro, noter = 0, nont = 0, i, j, k, flag = 0, count[10][10], row, col, l, m, n, index;

for (i = 0; i < 10; i++)

{

for (j = 0; j < 10; j++)

{

count[i][j] = NULL;

for (k = 0; k < 10; k++) {

res[i][j][k] = NULL;

}

}

}

printf("Enter the no of productions:\n");

scanf("%d", &npro);

printf("Enter the productions:\n");

for (i = 0; i < npro; i++)

{

scanf("%s", pro[i]);

}

for (i = 0; i < npro; i++)

{

flag = 0;

for (j = 0; j < nont; j++)

{

if (nt[j] == pro[i][0])

{

flag = 1;

}

}

if (flag == 0)

{

nt[nont] = pro[i][0];

nont++;

}

}

printf("\nEnter the first values:\n");

for (i = 0; i < nont; i++)

{

printf("First value(%c):", nt[i]);

scanf("%s", first[i]);

}

printf("\nEnter the follow values:\n");

for (i = 0; i < nont; i++)

{

printf("Follow value(%c):", nt[i]);

scanf("%s", follow[i]);

}

for (i = 0; i < nont; i++)

{

flag = 0;

for (j = 0; j < strlen(first[i]); j++)

{

for (k = 0; k < noter; k++)

{

if (ter[k] == first[i][j])

{

flag = 1;

}

}

if (flag == 0)

{

if (first[i][j] != '#')

{

ter[noter] = first[i][j];

noter++;

}

}

}

}

for (i = 0; i < nont; i++)

{

flag = 0;

for (j = 0; j < strlen(follow[i]); j++)

{

for (k = 0; k < noter; k++)

{

if (ter[k] == follow[i][j])

{

flag = 1;

}

}

if (flag == 0)

{

ter[noter] = follow[i][j];

noter++;

}

}

}

for (i = 0; i < nont; i++)

{

for (j = 0; j < strlen(first[i]); j++)

{

flag = 0;

if (first[i][j] == '#')

{

col = i;

for (m = 0; m < strlen(follow[col]); m++)

{

for (l = 0; l < noter; l++)

{

if (ter[l] == follow[col][m])

{

row = l;

}

}

temp[0] = nt[col];

temp[1] = '-';

temp[2] = '>';

temp[3] = '#';

temp[4] = '\0';

printf("temp %s", temp);

strcpy(res[col][row], temp);

count[col][row] += 1;

for (k = 0; k < 10; k++) {

temp[k] = NULL;

}

}

}

else {

for (l = 0; l < noter; l++)

{

if (ter[l] == first[i][j])

{

row = l;

}

}

for (k = 0; k < npro; k++) {

if (nt[i] == pro[k][0])

{

col = i;

if ((pro[k][3] == first[i][j]) && (pro[k][0] == nt[col]))

{

strcpy(res[col][row], pro[k]);

count[col][row] += 1;

}

else

{

if ((isupper(pro[k][3])) && (pro[k][0] == nt[col]))

{

flag = 0;

for (m = 0; m < nont; m++)

{

if (nt[m] == pro[k][3]) { index = m; flag = 1; }

}

if (flag == 1) {

for (m = 0; m < strlen(first[index]); m++)

{

if (first[i][j] == first[index][m])

{

strcpy(res[col][row], pro[k]);

count[col][row] += 1;

}

}

}

}

}

}

}

}

}

}

printf("LL1 Table\n\n");

flag = 0;

for (i = 0; i < noter; i++)

{

printf("\t%c", ter[i]);

}

for (j = 0; j < nont; j++)

{

printf("\n\n%c", nt[j]);

for (k = 0; k < noter; k++)

{

printf("\t%s", res[j][k]);

if (count[j][k] > 1) { flag = 1; }

}

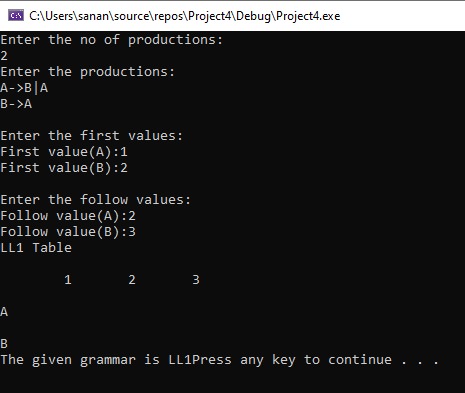
}

if (flag == 1) { printf("\nThe given grammar is not LL1"); }

else { printf("\nThe given grammar is LL1"); }

system("pause");

}



# Task 4:

#define \_CRT\_SECURE\_NO\_WARNINGS

#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;

count = 8;

// The Input grammar

strcpy(production[0], "E=TR");

strcpy(production[1], "R=+TR");

strcpy(production[2], "R=#");

strcpy(production[3], "T=FY");

strcpy(production[4], "Y=\*FY");

strcpy(production[5], "Y=#");

strcpy(production[6], "F=(E)");

strcpy(production[7], "F=i");

int kay;

char done[8];

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[8];

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++;

}

system("pause");

}

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++;

}

}

}

