|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.** | **Practical Name** | **Date** | **Page No.** | **Remarks** |
| 6. | 1. Write a C program for implementing the functionalities of predictive parser for the mini language. 2. Write a program in C to create LL (1) parsing table for the grammar and shoe the processing of any string that belongs to that grammar. E--> TE’ E’--> +TE'/€ T--> FT’ T’--> \*FT’/€ F--> (E)/id | 18-10-2024 | 25-39 |  |
| 7. | 1. Check which of the following Grammar is LR(0), justify your answer. (a) E--> T+E/T, T--> a (b) E--> E+T/T, T--> a | 26-10-2024 | 40-43 |  |
| 8. | 1. Construct an AST for expression a+a\*(b-c)+(b-c)\*d | 7-11-2024 | 44-46 |  |

**LAB 06**

**Objective (1):** Write a C program for implementing the functionalities of predictive parser for the mini language.

**Source Code:**

#include <stdio.h>

#include <string.h>

char prol[7][10] = {"S", "A", "A", "B", "B", "C", "C"};

char pror[7][10] = {"A", "Bb", "Cd", "aB", "@", "Cc", "@"};

char prod[7][10] = {"S->A", "A->Bb", "A->Cd", "B->aB", "B->@", "C->Cc", "C->@"};

char first[7][10] = {"abcd", "ab", "cd", "a@", "@", "c@", "@"};

char follow[7][10] = {"$", "$", "$", "a$", "b$", "c$", "d$"};

char table[5][6][10];

int numr(char c)

{

switch (c)

{

case 'S':

return 0;

case 'A':

return 1;

case 'B':

return 2;

case 'C':

return 3;

case 'a':

return 0;

case 'b':

return 1;

case 'c':

return 2;

case 'd':

return 3;

case '$':

return 4;

}

return 2;

}

int main()

{

int i, j, k;

// Initialize the parsing table with empty strings

for (i = 0; i < 5; i++)

{

for (j = 0; j < 6; j++)

{

strcpy(table[i][j], " ");

}

}

// Print the grammar rules

printf("\nGiven Grammar: \n");

for (i = 0; i < 7; i++)

{

printf("%s\n", prod[i]);

}

printf("\nPredictive parsing table: \n");

// Populate the parsing table using first and follow sets

for (i = 0; i < 7; i++)

{

k = strlen(first[i]);

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

{

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

{

strcpy(table[numr(prol[i][0]) + 1][numr(first[i][j]) + 1], prod[i]);

}

}

}

for (i = 0; i < 7; i++)

{

if (strlen(pror[i]) == 1)

{

if (pror[i][0] == '@')

{

k = strlen(follow[i]);

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

{

strcpy(table[numr(prol[i][0]) + 1][numr(follow[i][j]) + 1], prod[i]);

}

}

}

}

strcpy(table[0][0], " ");

strcpy(table[0][1], "a");

strcpy(table[0][2], "b");

strcpy(table[0][3], "c");

strcpy(table[0][4], "d");

strcpy(table[0][5], "$");

strcpy(table[1][0], "S");

strcpy(table[2][0], "A");

strcpy(table[3][0], "B");

strcpy(table[4][0], "C");

for (i = 0; i < 5; i++)

{

for (j = 0; j < 6; j++)

{

printf("%-10s", table[i][j]);

if (j == 5)

}

}

return 0;

}

**Output:**

A screenshot of a computer

Description automatically generated

**Objective (2):** Write a program in C to create LL (1) parsing table for the grammar and show= the processing of any string that belongs to that grammar.

E --> TE’

E’ --> +TE'/€

T --> FT’

T’ --> \*FT’/€

F --> (E)/id

**Source Code:**

#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 jm = 0;

int km = 0;

int i, choice;

char c, ch;

printf("Total no. of productions: ");

scanf("%d", &count);

printf("Enter %d productions in form A=B (A->B) where A and B are grammar symbols :\n", count);

printf("Use # for Epsilon\n\n");

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++;

}

}

}

**Output:**

A screenshot of a computer program

Description automatically generated

The LL(1) Parsing Table for the above grammer :-

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

======================================================================================================

| + \* ( ) i $

======================================================================================================

E | E=TR E=TR

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

R | R=+TR R=# R=#

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

T | T=FY T=FY

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

Y | Y=# Y=\*FY Y=# Y=#

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

F | F=(E) F=i

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



===========================================================================

Stack Input Action

===========================================================================

$E i+i\*i$ E=TR

$RT i+i\*i$ T=FY

$RYF i+i\*i$ F=i

$RYi i+i\*i$ POP ACTION

$RY +i\*i$ Y=#

$R +i\*i$ R=+TR

$RT+ +i\*i$ POP ACTION

$RT i\*i$ T=FY

$RYF i\*i$ F=i

$RYi i\*i$ POP ACTION

$RY \*i$ Y=\*FY

$RYF\* \*i$ POP ACTION

$RYF i$ F=i

$RYi i$ POP ACTION

$RY $ Y=#

$R $ R=#

$ $ POP ACTION

=======================================================================================

YOUR STRING HAS BEEN ACCEPTED !

**LAB 07**

**Objective : Check which of the following Grammar is LR(0), justify your answer.**

**E--> T+E/T,**

**T--> a**

**E--> E+T/T,**

**T--> a**

**Source Code (a):**

%{

#include <stdio.h>

%}

%token a '+'

%%

E : T '+' E

| T;

T : 'a';

%%

int main() {

return yyparse();

}

int yyerror(char \*s) {

printf("%s\n", s);

return 0;

}

**Output:**

**Lab7a.output**

Terminals which are not used

a

Grammar

0 $accept: E $end

1 E: T '+' E

2 | T

3 T: 'a'

Terminals, with rules where they appear

$end (0) 0

'+' (43) 1

'a' (97) 3

error (256)

a (258)

Nonterminals, with rules where they appear

$accept (6)

on left: 0

E (7)

on left: 1 2, on right: 0 1

T (8)

on left: 3, on right: 1 2

state 0

0 $accept: . E $end

'a' shift, and go to state 1

E go to state 2

T go to state 3

state 1

3 T: 'a' .

$default reduce using rule 3 (T)

state 2

0 $accept: E . $end

$end shift, and go to state 4

state 3

1 E: T . '+' E

2 | T .

'+' shift, and go to state 5

$default reduce using rule 2 (E)

state 4

0 $accept: E $end .

$default accept

state 5

1 E: T '+' . E

'a' shift, and go to state 1

E go to state 6

T go to state 3

state 6

1 E: T '+' E .

$default reduce using rule 1 (E)

**Result (a):** state 3 have shift-reduce conflict. Therefore, it is not a LR(0) Grammar.

**Source Code (b):**

%{

#include <stdio.h>

%}

%token a '+'

%%

E : E '+' T

| T;

T : 'a';

%%

int main() {

return yyparse();

}

int yyerror(char \*s) {

printf("%s\n", s);

return 0;

}

**Output:**

**Lab7b.output**

Terminals which are not used

a

Grammar

0 $accept: E $end

1 E: E '+' T

2 | T

3 T: 'a'

Terminals, with rules where they appear

$end (0) 0

'+' (43) 1

'a' (97) 3

error (256)

a (258)

Nonterminals, with rules where they appear

$accept (6)

on left: 0

E (7)

on left: 1 2, on right: 0 1

T (8)

on left: 3, on right: 1 2

state 0

0 $accept: . E $end

'a' shift, and go to state 1

E go to state 2

T go to state 3

state 1

3 T: 'a' .

$default reduce using rule 3 (T)

state 2

0 $accept: E . $end

1 E: E . '+' T

$end shift, and go to state 4

'+' shift, and go to state 5

state 3

2 E: T .

$default reduce using rule 2 (E)

state 4

0 $accept: E $end .

$default accept

state 5

1 E: E '+' . T

'a' shift, and go to state 1

T go to state 6

state 6

1 E: E '+' T .

$default reduce using rule 1 (E)

**Result (b):** None of the states have shift-reduce or reduce-reduce conflict. Therefore, it’s a LR(0) Grammar.

**LAB 08**

**Objective : Write a C program to implement program semantic rules to calculate the expression, that takes an expression with digit “+” & “\*” and compute the value.**

**Source Code :**

#include <bits/stdc++.h>

#define MAX\_SIZE 50

int numeral\_stack[MAX\_SIZE];

int top\_numeral = -1;

void pushNumeral(int numeral)

{

top\_numeral++;

numeral\_stack[top\_numeral]=numeral;

}

int popNumeral(){

int temp=numeral\_stack[top\_numeral];

top\_numeral--;

return temp;

}

char operator\_stack[MAX\_SIZE];

int top\_operator = -1;

void pushOperator(char op)

{

top\_operator++;

operator\_stack[top\_operator] = op;

}

int popOperator(){

int temp = operator\_stack[top\_operator];

top\_operator--;

return temp;

}

int isOperator\_stack\_empty(){

return top\_operator==-1;

}

int precedence(char op){

if(op=='\*') return 2;

if(op=='+') return 1;

return 0;

}

int calculate(int a, int b, char op){

if(op=='\*') return a\*b;

if(op=='+') return a+b;

return 0;

}

int calculateExpression(char \*character){

int i=0;

while(character[i]!='\0'){// TILL IT REACHES END OF THE STRING

if(character[i]>='0' && character[i]<='9'){// CHECK IF ITS A NUMERAL

int num=0;

while(character[i]>='0' && character[i]<='9'){ // CONVERT THE NUMERAL TO INTEGER FOR >1 DIGITS

num=num\*10+(character[i]-'0');

i++;

}

pushNumeral(num);

}

else if(character[i]=='\*' || character[i]=='+'){

while (!isOperator\_stack\_empty() && precedence(operator\_stack[top\_operator]) >= precedence(character[i])){

int a = popNumeral();

int b = popNumeral();

char op = popOperator();

pushNumeral(calculate(a,b,op));

}

pushOperator(character[i]);

i++;

}

else i++; // IGNORE ELSEWISE

}

// REMAINED OPERATORS IN STACK

while (!isOperator\_stack\_empty()){

int a = popNumeral();

int b = popNumeral();

char op = popOperator();

pushNumeral(calculate(a,b,op));

}

return popNumeral();

}

int main(){

char expression[MAX\_SIZE];

printf("Enter your expression: ");

scanf("%s",expression);

printf("Value: %d\n",calculateExpression(expression));

return 0;

}

**Output:**

