**Mini project**

**Experiment 10 :Design and development of LR(0) Parser**

**Learning Objective**: Student should be able to apply LR(0) parsing technique and implement it.

**Tools:**  Dev C++,Notepad

**Theory:**

**LR(0) parser:** The LR parser is an efficient bottom-up syntax analysis technique that can be used for a large class of context-free grammar. This technique is also called LR(0) parsing.

L stands for the left to right scanning R stands for rightmost derivation in reverse

0 stands for no. of input symbols of lookahead.

**LR(0) parser Algorithm:**

1. Construct the given grammar G into augmented grammar
2. Find Item I
3. Compute / Find Closure (I)
4. Apply Goto function to find LR(0) collection
5. Draw DFA Graph
6. Construct LR(0) Table
7. Derive i/p string

**Code**:

#include<iostream>

#include<conio.h>

#include<string.h>

using namespace std;

char prod[20][20], listofvar[26] = "ABCDEFGHIJKLMNOPQR";

int novar = 1, i = 0, j = 0, k = 0, n = 0, m = 0, arr[30];

int noitem = 0;

struct Grammar {

char lhs;

char rhs[8];

}

g[20], item[20], clos[20][10];

int isvariable(char variable) {

for (int i = 0; i < novar; i++)

if (g[i].lhs == variable)

return i + 1;

return 0;

}

void findclosure(int z, char a) {

int n = 0, i = 0, j = 0, k = 0, l = 0;

for (i = 0; i < arr[z]; i++) {

for (j = 0; j < strlen(clos[z][i].rhs); j++) {

if (clos[z][i].rhs[j] == '.' && clos[z][i].rhs[j + 1] == a) {

clos[noitem][n].lhs = clos[z][i].lhs;

strcpy(clos[noitem][n].rhs, clos[z][i].rhs);

char temp = clos[noitem][n].rhs[j];

clos[noitem][n].rhs[j] = clos[noitem][n].rhs[j + 1];

clos[noitem][n].rhs[j + 1] = temp;

n = n + 1;

}

}

}

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

for (j = 0; j < strlen(clos[noitem][i].rhs); j++) {

if (clos[noitem][i].rhs[j] == '.' &&

isvariable(clos[noitem][i].rhs[j + 1]) > 0) {

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

if (clos[noitem][i].rhs[j + 1] == clos[0][k].lhs) {

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

if (clos[noitem][l].lhs == clos[0][k].lhs &&

strcmp(clos[noitem][l].rhs, clos[0][k].rhs) == 0)

break;

if (l == n) {

clos[noitem][n].lhs = clos[0][k].lhs;

strcpy(clos[noitem][n].rhs, clos[0][k].rhs);

n = n + 1;

}

}

}

}

}

}

arr[noitem] = n;

int flag = 0;

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

if (arr[i] == n) {

for (j = 0; j < arr[i]; j++) {

int c = 0;

for (k = 0; k < arr[i]; k++)

if (clos[noitem][k].lhs == clos[i][k].lhs &&

strcmp(clos[noitem][k].rhs, clos[i][k].rhs) == 0)

c = c + 1;

if (c == arr[i]) {

flag = 1;

goto exit;

}

}

}

}

exit: ;

if (flag == 0)

arr[noitem++] = n;

}

int main() {

cout << "ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :\n";

do {

cin >> prod[i++];

} while (strcmp(prod[i - 1], "0") != 0);

for (n = 0; n < i - 1; n++) {

m = 0;

j = novar;

g[novar++].lhs = prod[n][0];

for (k = 3; k < strlen(prod[n]); k++) {

if (prod[n][k] != '|')

g[j].rhs[m++] = prod[n][k];

if (prod[n][k] == '|') {

g[j].rhs[m] = '\0';

m = 0;

j = novar;

g[novar++].lhs = prod[n][0];

}

}

}

for (i = 0; i < 26; i++)

if (!isvariable(listofvar[i]))

break;

g[0].lhs = listofvar[i];

char temp[2] = {

g[1].lhs,

'\0'

};

strcat(g[0].rhs, temp);

cout << "\n\n augumented grammar \n";

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

cout << endl << g[i].lhs << "->" << g[i].rhs << " ";

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

clos[noitem][i].lhs = g[i].lhs;

strcpy(clos[noitem][i].rhs, g[i].rhs);

if (strcmp(clos[noitem][i].rhs, "ε") == 0)

strcpy(clos[noitem][i].rhs, ".");

else {

for (int j = strlen(clos[noitem][i].rhs) + 1; j >= 0; j--)

clos[noitem][i].rhs[j] = clos[noitem][i].rhs[j - 1];

clos[noitem][i].rhs[0] = '.';

}

}

arr[noitem++] = novar;

for (int z = 0; z < noitem; z++) {

char list[10];

int l = 0;

for (j = 0; j < arr[z]; j++) {

for (k = 0; k < strlen(clos[z][j].rhs) - 1; k++) {

if (clos[z][j].rhs[k] == '.') {

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

if (list[m] == clos[z][j].rhs[k + 1])

break;

if (m == l)

list[l++] = clos[z][j].rhs[k + 1];

}

}

}

for (int x = 0; x < l; x++)

findclosure(z, list[x]);

}

cout << "\n THE SET OF ITEMS ARE \n\n";

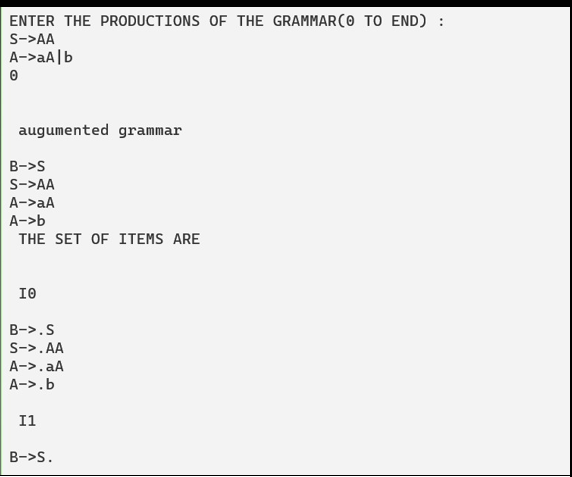
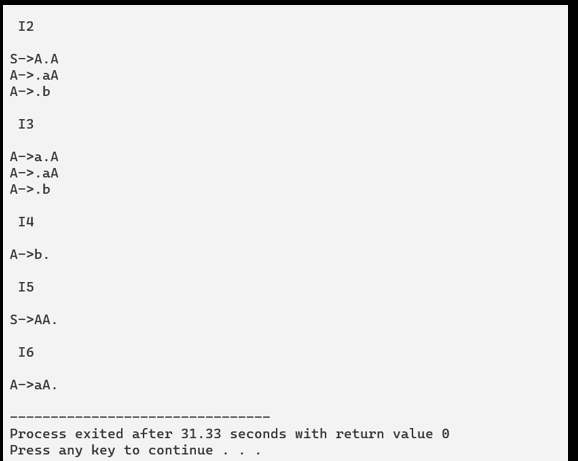
for (int z = 0; z < noitem; z++) {

cout << "\n I" << z << "\n\n";

for (j = 0; j < arr[z]; j++)

cout << clos[z][j].lhs << "->" << clos[z][j].rhs << "\n";

}



**Learning Outcomes:** The student should have the ability to

LO1 **Define** the role of Code Generator in Compiler design.

LO2: ***Apply*** the code generator algorithm to generate the machine code.

LO3: ***Generate*** target code for the optimized code, considering the target machines.

**Course Outcomes**: Upon completion of the course students will be able to evaluate the synthesis phase to produce object code optimized in terms of high execution speed and less memory usage.

**Conclusion:** From this experiment we will be able to apply LR(0) parsing technique and implement it. We understood the LR(O) parser algorithm . Now we have the ability to Define the role of Code Generator in Compiler design , *Apply* the code generator algorithm to generate the machine code. and *Generate* target code for the optimized code, considering the target machines.

For Faculty Use

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Correction Parameters** | **Formative**  **Assessment**  **[40%]** | **Timely completion of Practical [ 40%]** | **Attendance /**  **Learning**  **Attitude**  **[20%]** |  |
| **Marks Obtained** |  |  |  |