ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

COMPILER DESIGN LAB

EXP 9 - COMPUTATION OF LR(0) ITEMS

17-3-22

EXP 9 - LR(0) PARSER

AIM

To build a C++ program which produces LR(0) items as the output.

REQUIREMENTS

1. Knowledge of the concepts of parsing [botton-up]
2. Knowledge of the concepts of LR(0) items and how to build them.
3. An online C++ compiler to execute the code.

THEORY

LR(0) Parser:

The LR parser is an efficient botton up syntax analysis technique that can be used to large class of context free grammar.

L stands for left to right scanning.
R stands for rightmost derivation in reverse.
0 stands for no. of input symbols of lookahead.

Augmented Grammar:

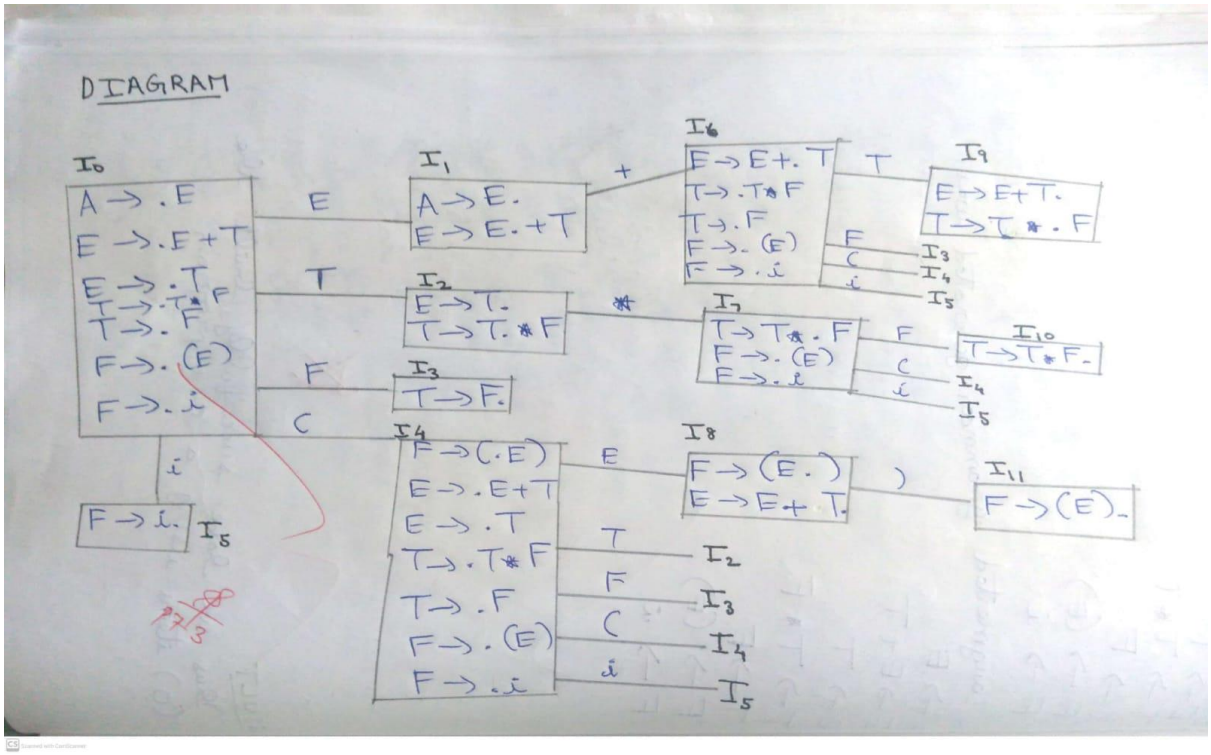
If P is a grammar with starting symbol S, then augmented grammar S' is given by

$S \rightarrow .S'$

The purpose of this new starting production is to indicate the parser when it should stop parsing. The '.' before S indicates the left side of S has been read by compiler and right

ALGORITHM

1. Start
2. Create structure for production with LHS and RHS
3. Open file and read input from file
4. Build state 0 from extra grammar law $S' \rightarrow S\ \$$ that is all start symbol of grammar and one Dot (.) before S symbol
5. If dot symbol is before a non-terminal, add grammar laws that this non-terminal is in LHS of the law and set dot in before first part of RHS.
6. If state exists, [a state with this laws and same dot position], use that instead.
7. Now find set of terminals and non-terminals in which dot exists before.
8. If step 7 set is non-empty, go to 9 or else go to 10.
9. For each terminal / non-terminal in step 7, create new state by using all grammar law that dot position is before of that terminal / non-terminal in reference state by increasing dot point to next part in RHS of that laws.
10. Go to step 5.
11. End of state building
12. Display the output
13. END



SCREENSHOTS :

main.cpp

```
#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++)
    {
```

input

```
...Program finished with exit code 0
Press ENTER to exit console.
```

```
T->F.

 I4

F->(.E)
E->.E+T
E->.T
T->.T*F
T->.F
F->.(E)
F->.I

 I5

F->I.

 I6

E->E+.T
T->.T*F
T->.F
F->.(E)
F->.I

 I7

T->T*.F
F->.(E)
F->.I

 I8

F->(E.)
E->E.+T

 I9

E->E+T.
T->T.*F

 I10

T->T*F.

 I11

F->(E).


..Program finished with exit code 0
Press ENTER to exit console.
```

```
ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :
E->E+T
E->T
T->T*F
T->F
F->(E)
F->I
0


 augumented grammar

A->E
E->E+T
E->T
T->T*F
T->F
F->(E)
F->I
 THE SET OF ITEMS ARE


 I0

A->.E
E->.E+T
E->.T
T->.T*F
T->.F
F->.(E)
F->.I

 I1

A->E.
E->E.+T

 I2

E->T.
T->T.*F

 I3

T->F.

 I4

F->(.E)
E->.E+T
E->.T
```

OBSERVATION

The productions of the grammar entered were:

E → E+T
E → T
T → T*F
T → F
F → (E)
F → i

The augmented grammar generated was:

A → E
E → E+T
E → T
T → T*F
T → F
F → (E)
F → i

RESULT

Thus we have succesfully built the LR(0) items using a C++ program.