

# **NATIONAL INSTITUTE OF TECHNOLOGY** **DURGAPUR**

Department of **Computer Science and Engineering**

## **COMPILER DESIGN SESSIONAL** **(CSS-651)**

### **REPORT-3**

**Section : CS-Y**

**Group No. - 5**

**Group Members :**

- 1) SAPTARSHI MONDAL (18CS8076)
- 2) PRIYESH DEEP KUMAR (18CS8077)
- 3) KALA YADUVEERA CHOWDAIAH (18CS8078)
- 4) KALIVARAPU ADITHYA (18CS8079)
- 5) DEBANANDA DAS (18CS8080)

## **Assignment-5**

**Q. Implement one LR(0) parser without and with error detection capacity. The input to the parser is the input sentence to be parsed.**

**(i) The grammar is for LR(0) parser is fixed.**

**(ii) The grammar for the LR(0) parser is another input along with the input text.**

### **SOFTWARE USED :**

**Visual Studio Code**(for C program), **Ubuntu Terminal** (for Compiling)

### **THEORY :**

#### **LR PARSERS:**

The 'most prevalent type of bottom-up parser today is based on a concept called LR(k) parsing; the "L" is for left-to-right scanning of the input, the "R" for constructing a rightmost derivation in reverse, and the k for the number of input symbols of lookahead that are used in making parsing decisions.

LR parsing is attractive because of variety of reasons:

- LR parsers can be constructed to recognize virtually all programming language constructs for which context-free grammars can be written. Non LR context-free grammars exist, but these can generally be avoided for typical programming-language constructs.
- The LR-parsing method is the most general non back tracking shift-reduce parsing method known, yet it can be implemented as efficiently as other, more primitive shift-reduce methods .
- An LR parser can detect a syntactic error as soon as it is possible to do so on a left-to-right scan of the input.
- LR grammars can describe more languages than LL grammars.

## ITEMS AND THE LR(0) AUTOMATION:

An LR parser makes shift-reduce decisions by maintaining states to keep track of where we are in a parse. States represent sets of "items." An LR(0) item (item for short) of a grammar  $G$  is a production of  $G$  with a dot at some position of the body. Thus, production  $A \rightarrow XYZ$  yields the four items

$A \rightarrow \cdot XYZ$

$A \rightarrow X \cdot YZ$

$A \rightarrow XY \cdot Z$

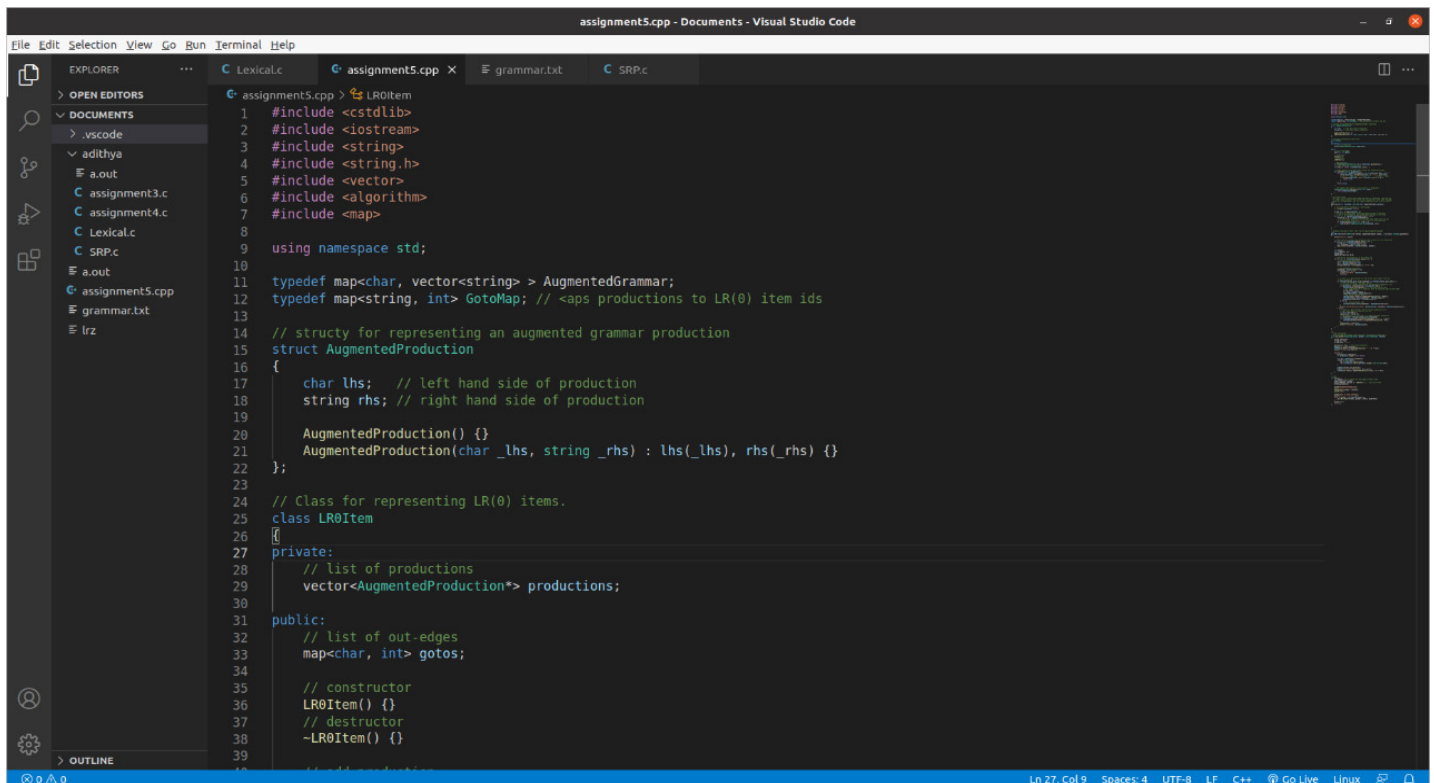
$A \rightarrow XYZ \cdot$

The production  $A \rightarrow \epsilon$  generates only one item,  $A \rightarrow \cdot$ .

Intuitively, an item indicates how much of a production we have seen at a given point in the parsing process. For example, the item  $A \rightarrow \cdot XYZ$  indicates that we hope to see a string derivable from  $XYZ$  next on the input. Item  $A \rightarrow X \cdot YZ$  indicates that we have just seen on the input a string derivable from  $X$  and that we hope next to see a string derivable from  $YZ$ . Item  $A \rightarrow XY Z \cdot$  indicates that we have seen the body  $XYZ$  and that it may be time to reduce  $XYZ$  to  $A$ .

## RESULTS :

### Screenshot of the Code :



```
assignment5.cpp - Documents - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
  OPEN EDITORS
  DOCUMENTS
    .vscode
  adithya
    a.out
    assignment3.c
    assignment4.c
    Lexical.c
    SRP.c
    a.out
    assignment5.cpp
    grammar.txt
    lrz

C Lexical.c
C assignment5.cpp X
E grammar.txt
C SRP.c

C assignment5.cpp > LR0Item
1 #include <cstdlib>
2 #include <iostream>
3 #include <string>
4 #include <string.h>
5 #include <vector>
6 #include <algorithm>
7 #include <map>
8
9 using namespace std;
10
11 typedef map<char, vector<string> > AugmentedGrammar;
12 typedef map<string, int> GotoMap; // <aps productions to LR(0) item ids
13
14 // structy for representing an augmented grammar production
15 struct AugmentedProduction
16 {
17     char lhs; // left hand side of production
18     string rhs; // right hand side of production
19
20     AugmentedProduction() {}
21     AugmentedProduction(char _lhs, string _rhs) : lhs(_lhs), rhs(_rhs) {}
22 };
23
24 // Class for representing LR(0) items.
25 class LR0Item
26 {
27 private:
28     // list of productions
29     vector<AugmentedProduction*> productions;
30
31 public:
32     // list of out-edges
33     map<char, int> gotos;
34
35     // constructor
36     LR0Item() {}
37     // destructor
38     ~LR0Item() {}
39
40     // methods
41     void add_production(AugmentedProduction* p) { productions.push_back(p); }
```

```
assignment5.cpp - Documents - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
  OPEN EDITORS
  DOCUMENTS
    .vscode
    adithya
      a.out
      assignment3.c
      assignment4.c
      Lexical.c
      SRP.c
      a.out
      assignment5.cpp
      grammar.txt
      lrz

assignment5.cpp
40 // add production
41 void Push(AugmentedProduction *p) { productions.push_back(p); }
42 // return the number of productions
43 int Size() { return int(productions.size()); }
44
45 // return whether or not this item contains the production prodStr
46 bool Contains(string production) {
47     for (auto it = productions.begin(); it != productions.end(); it++) {
48         string existing = string(&(*it)->lhs, 1) + "->" + (*it)->rhs;
49         //cout << " Comparing: " << thisStr << " , " << prodStr << endl;
50         if (strcmp(production.c_str(), existing.c_str()) == 0) {
51             return true;
52         }
53     }
54     return false;
55 }
56
57 // overloaded index operator; access pointer to production.
58 AugmentedProduction* operator[](const int index) {
59     return productions[index];
60 }
61 };
62
63 /* void add_closure
64  * If 'next' is the current input symbol and next is nonterminal, then the set
65  * of LR(0) items reachable from here on next includes all LR(0) items reachable
66  * from here on FIRST(next). Add all grammar productions with a lhs of next */
67 void
68 add_closure(char lookahead, LR0Item& item, AugmentedGrammar& grammar)
69 {
70     // only continue if lookahead is a non-terminal
71     if (!isupper(lookahead)) return;
72
73     string lhs = string(&lookahead, 1);
74     // iterate over each grammar production beginning with p->rhs[next]
75     // to see if that production has already been included in this item.
76     for (int i = 0; i < grammar[lookahead].size(); i++) {
77         string rhs = "@" + grammar[lookahead][i];
78         // if the grammar production for the next input symbol does not yet
79         // exist for this item, add it to the item's set of productions
```

```
assignment5.cpp - Documents - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
  OPEN EDITORS
  DOCUMENTS
    .vscode
    adithya
      a.out
      assignment3.c
      assignment4.c
      Lexical.c
      SRP.c
      a.out
      assignment5.cpp
      grammar.txt
      lrz

assignment5.cpp
78 // if the grammar production for the next input symbol does not yet
79 // exist for this item, add it to the item's set of productions
80 if (!item.Contains( lhs + "->" + rhs )) {
81     item.Push(new AugmentedProduction(lookahead, rhs));
82 }
83 }
84 }
85
86 // produce the graph of LR(0) items from the given augmented grammar
87 void
88 get_LR0_items(vector<LR0Item>& lr0items, AugmentedGrammar& grammar, int& itemid, GotoMap& globalGoto)
89 {
90     printf("I%d:\n", itemid);
91
92     // ensure that the current item contains the full closure of it's productions
93     for (int i = 0; i < lr0items[itemid].Size(); i++) {
94         string rhs = lr0items[itemid][i]->rhs;
95         char lookahead = rhs[rhs.find('@')+1];
96         add_closure(lookahead, lr0items[itemid], grammar);
97     }
98
99     int nextpos;
100     char lookahead, lhs;
101     string rhs;
102     AugmentedProduction *prod;
103
104     // iterate over each production in this LR(0) item
105     for (int i = 0; i < lr0items[itemid].Size(); i++) {
106         // get the current production
107         lhs = lr0items[itemid][i]->lhs;
108         rhs = lr0items[itemid][i]->rhs;
109         string production = string(&lhs, 1) + "->" + rhs;
110
111         // get lookahead if one exists
112         lookahead = rhs[rhs.find('@')+1];
113         if (lookahead == '\0') {
114             printf("\t%-20s\n", &production[0]);
115             continue;
116         }
117     }
```



```
assignment5.cpp - Documents - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
> OPEN EDITORS
  C Lexical.c
  assignment5.cpp x
  grammar.txt
  SRP.c

DOCUMENTS
> .vscode
  adithya
    a.out
    assignment3.c
    assignment4.c
    Lexical.c
    SRP.c
    a.out
    assignment5.cpp
    grammar.txt
    lrz

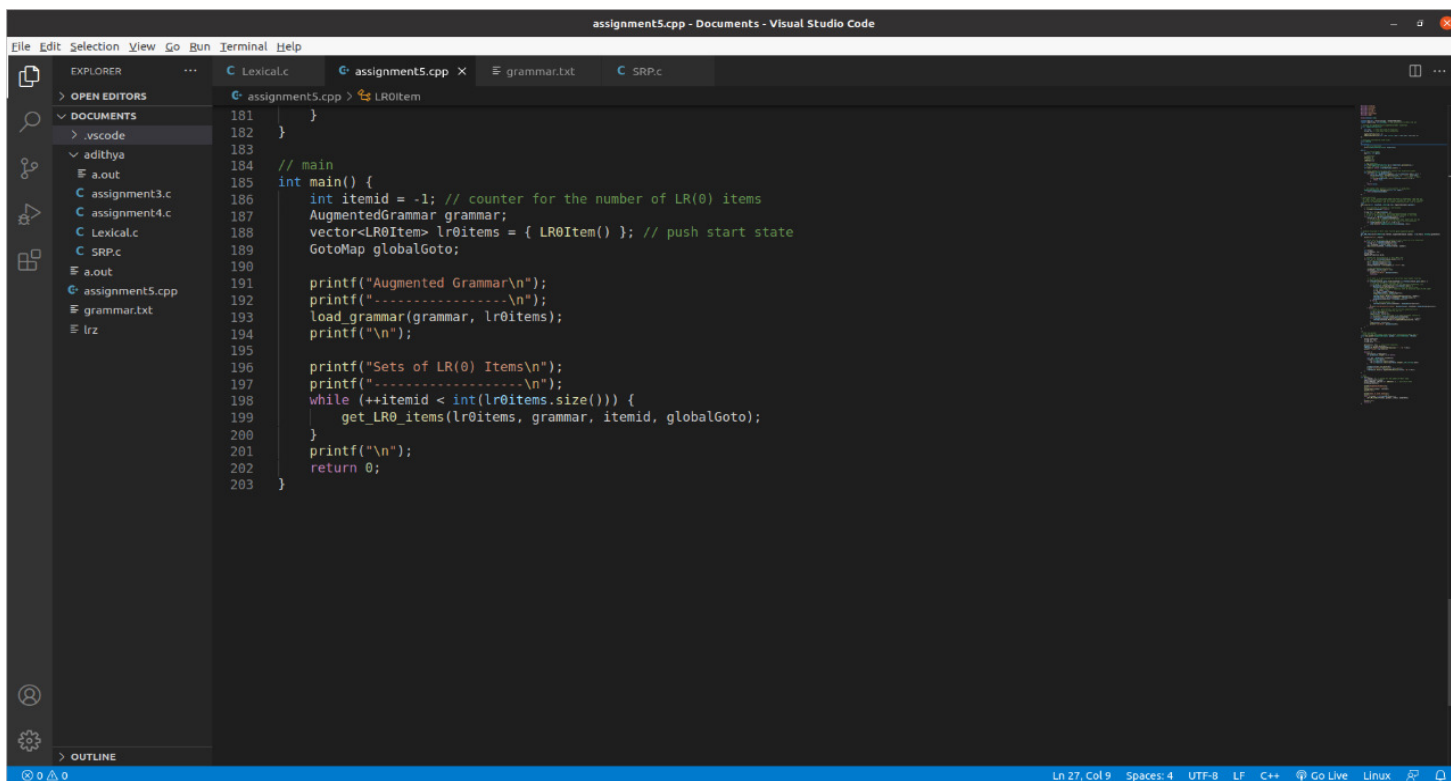
OUTLINE
  117
  118 // if there is no goto defined for the current input symbol from this
  119 // item, assign one.
  120 if (lr0Items[itemid].gotos.find(lookahead) == lr0Items[itemid].gotos.end()) {
  121     // that one instead of creating a new one
  122     // if there is a global goto defined for the entire production, use
  123     if (globalGoto.find(production) == globalGoto.end()) {
  124         lr0Items.push_back(LR0Item()); // create new state (item)
  125         // new right-hand-side is identical with '@' moved one space to the right
  126         string newRhs = rhs;
  127         int atpos = newRhs.find('@');
  128         swap(newRhs[atpos], newRhs[atpos+1]);
  129         // add item and update gotos
  130         lr0Items.back().Push(new AugmentedProduction(lhs, newRhs));
  131         lr0Items[itemid].gotos[lookahead] = lr0Items.size()-1;
  132         globalGoto[production] = lr0Items.size()-1;
  133     } else {
  134         // use existing global item
  135         lr0Items[itemid].gotos[lookahead] = globalGoto[production];
  136     }
  137     printf("\t%-20s goto(%c)=%d\n", &production[0], lookahead, globalGoto[production]);
  138 } else {
  139     // there is a goto defined, add the current production to it
  140     // move @ one space to right for new rhs
  141     int at = rhs.find('@');
  142     swap(rhs[at], rhs[at+1]);
  143     // add production to next item if it doesn't already contain it
  144     int nextItem = lr0Items[itemid].gotos[lookahead];
  145     if (!lr0Items[nextItem].Contains(string(&lhs, 1) + "->" + rhs)) {
  146         lr0Items[nextItem].Push(new AugmentedProduction(lhs, rhs));
  147     }
  148     swap(rhs[at], rhs[at+1]);
  149     printf("\t%-20s\n", &production[0]);
  150 }
  151 }
  152 }
  153
  154 /**
  155  * void load_grammar
  156  * scan and load the grammar from stdin while setting first LR(0) item */
```

```
assignment5.cpp - Documents - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
> OPEN EDITORS
  C Lexical.c
  assignment5.cpp x
  grammar.txt
  SRP.c

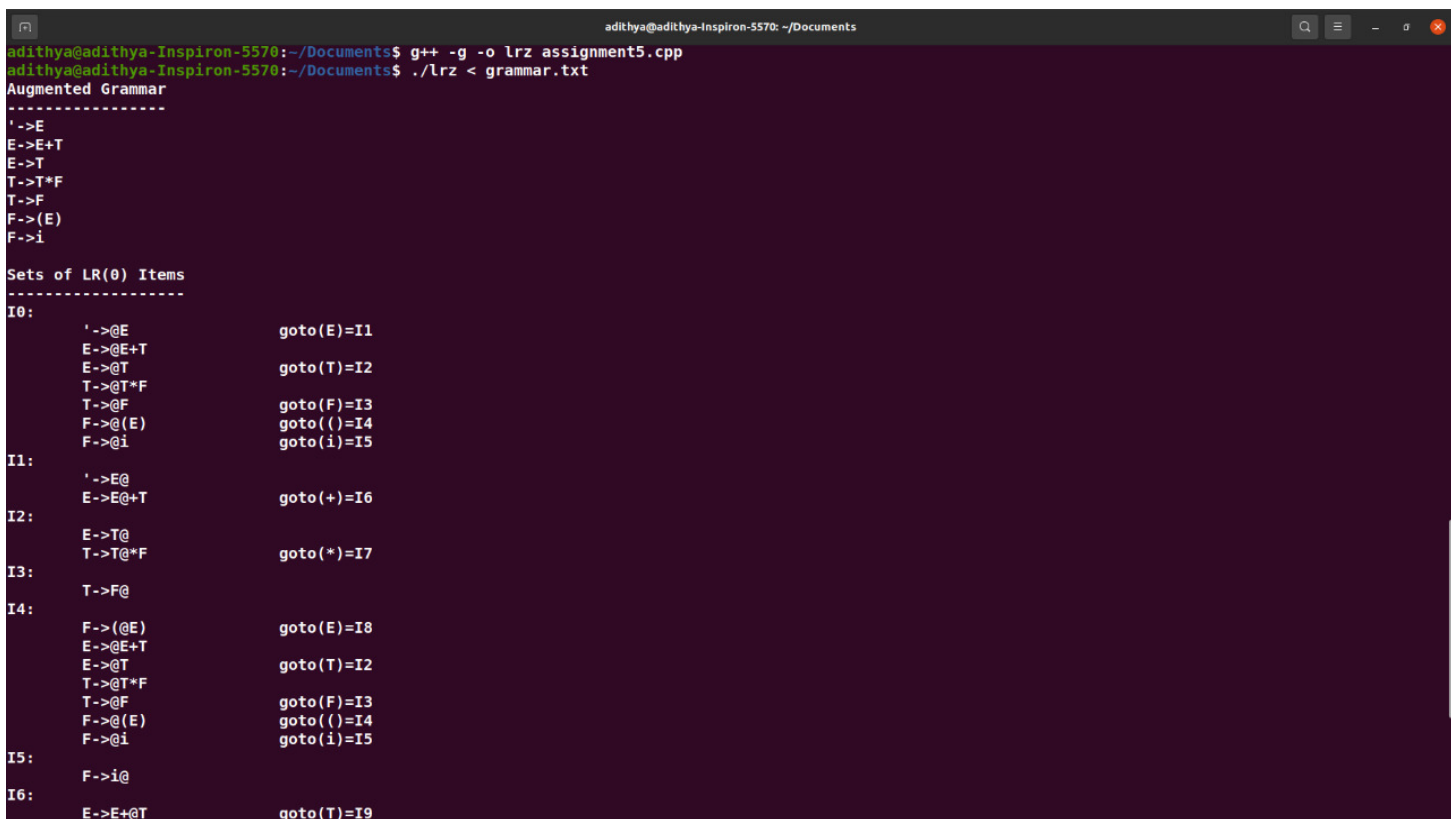
DOCUMENTS
> .vscode
  adithya
    a.out
    assignment3.c
    assignment4.c
    Lexical.c
    SRP.c
    a.out
    assignment5.cpp
    grammar.txt
    lrz

OUTLINE
  154 /**
  155  * void load_grammar
  156  * scan and load the grammar from stdin while setting first LR(0) item */
  157 void load_grammar(AugmentedGrammar& grammar, vector<LR0Item>& lr0Items)
  158 {
  159     string production;
  160     string lhs, rhs;
  161     string delim = "->";
  162
  163     getline(cin, lhs); // scan start production
  164     grammar['\'].push_back(lhs);
  165     lr0Items[0].Push(new AugmentedProduction('\', "@" + lhs));
  166     printf("%s\n", lhs.c_str());
  167
  168     while(1) {
  169         getline(cin, production);
  170         if (production.length() < 1) return;
  171
  172         auto pos = production.find(delim);
  173         if (pos != string::npos) {
  174             lhs = production.substr(0, pos);
  175             rhs = production.substr(pos+delim.length(), std::string::npos);
  176         }
  177
  178         grammar[lhs[0]].push_back(rhs);
  179         printf("%s->%s\n", lhs.c_str(), rhs.c_str());
  180         lr0Items[0].Push(new AugmentedProduction(lhs[0], "@" + rhs));
  181     }
  182 }
  183
  184 // main
  185 int main() {
  186     int itemid = -1; // counter for the number of LR(0) items
  187     AugmentedGrammar grammar;
  188     vector<LR0Item> lr0Items = { LR0Item() }; // push start state
  189     GotoMap globalGoto;
  190
  191     printf("Augmented Grammar\n");
  192     printf("-----\n");
  193     load_grammar(grammar, lr0Items);
```



```
181 }
182 }
183
184 // main
185 int main() {
186     int itemid = -1; // counter for the number of LR(0) items
187     AugmentedGrammar grammar;
188     vector<LR0Item> lr0items = { LR0Item() }; // push start state
189     GotoMap globalGoto;
190
191     printf("Augmented Grammar\n");
192     printf("-----\n");
193     load_grammar(grammar, lr0items);
194     printf("\n");
195
196     printf("Sets of LR(0) Items\n");
197     printf("-----\n");
198     while (++itemid < int(lr0items.size())) {
199         get_LR0_items(lr0items, grammar, itemid, globalGoto);
200     }
201     printf("\n");
202     return 0;
203 }
```

## Screenshot of the Results in the Terminal :



```
adithya@adithya-Inspiron-5570: ~/Documents
adithya@adithya-Inspiron-5570:~/Documents$ g++ -g -o lrz assignment5.cpp
adithya@adithya-Inspiron-5570:~/Documents$ ./lrz < grammar.txt
Augmented Grammar
-----
' -> E
E -> E+T
E -> T
T -> T*F
T -> F
F -> (E)
F -> i

Sets of LR(0) Items
-----
I0:
    ' -> @E          goto(E)=I1
    E -> @E+T        goto(T)=I2
    E -> @T
    T -> @T*F        goto(F)=I3
    T -> @F          goto(()=I4
    F -> @(E)        goto(i)=I5
    F -> @i

I1:
    ' -> E@
    E -> E@+T        goto(+)=I6

I2:
    E -> T@
    T -> T@*F        goto(*)=I7

I3:
    T -> F@

I4:
    F -> (@E)        goto(E)=I8
    E -> @E+T        goto(T)=I2
    E -> @T
    T -> @T*F        goto(F)=I3
    T -> @F          goto(()=I4
    F -> @(E)        goto(i)=I5
    F -> @i

I5:
    F -> i@

I6:
    E -> E+@T        goto(T)=I9
```

```
adithya@adithya-Inspiron-5570: ~/Documents

I1:      F->@i      goto(i)=I5
      ' ->E@
      E->E@+T      goto(+)=I6
I2:
      E->T@
      T->T@*F      goto(*)=I7
I3:
      T->F@
I4:
      F->(@E)      goto(E)=I8
      E->@E+T
      E->@T      goto(T)=I2
      T->@T*F
      T->@F      goto(F)=I3
      F->@ (E)      goto(()=I4
      F->@i      goto(i)=I5
I5:
      F->i@
I6:
      E->E+@T      goto(T)=I9
      T->@T*F
      T->@F      goto(F)=I3
      F->@ (E)      goto(()=I4
      F->@i      goto(i)=I5
I7:
      T->T*@F      goto(F)=I10
      F->@ (E)      goto(()=I4
      F->@i      goto(i)=I5
I8:
      F-> (E@)      goto())=I11
      E->E@+T      goto(+)=I6
I9:
      E->E+T@
      T->T@*F      goto(*)=I7
I10:
      T->T*F@
I11:
      F->(E)@

adithya@adithya-Inspiron-5570:~/Documents$
```