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:

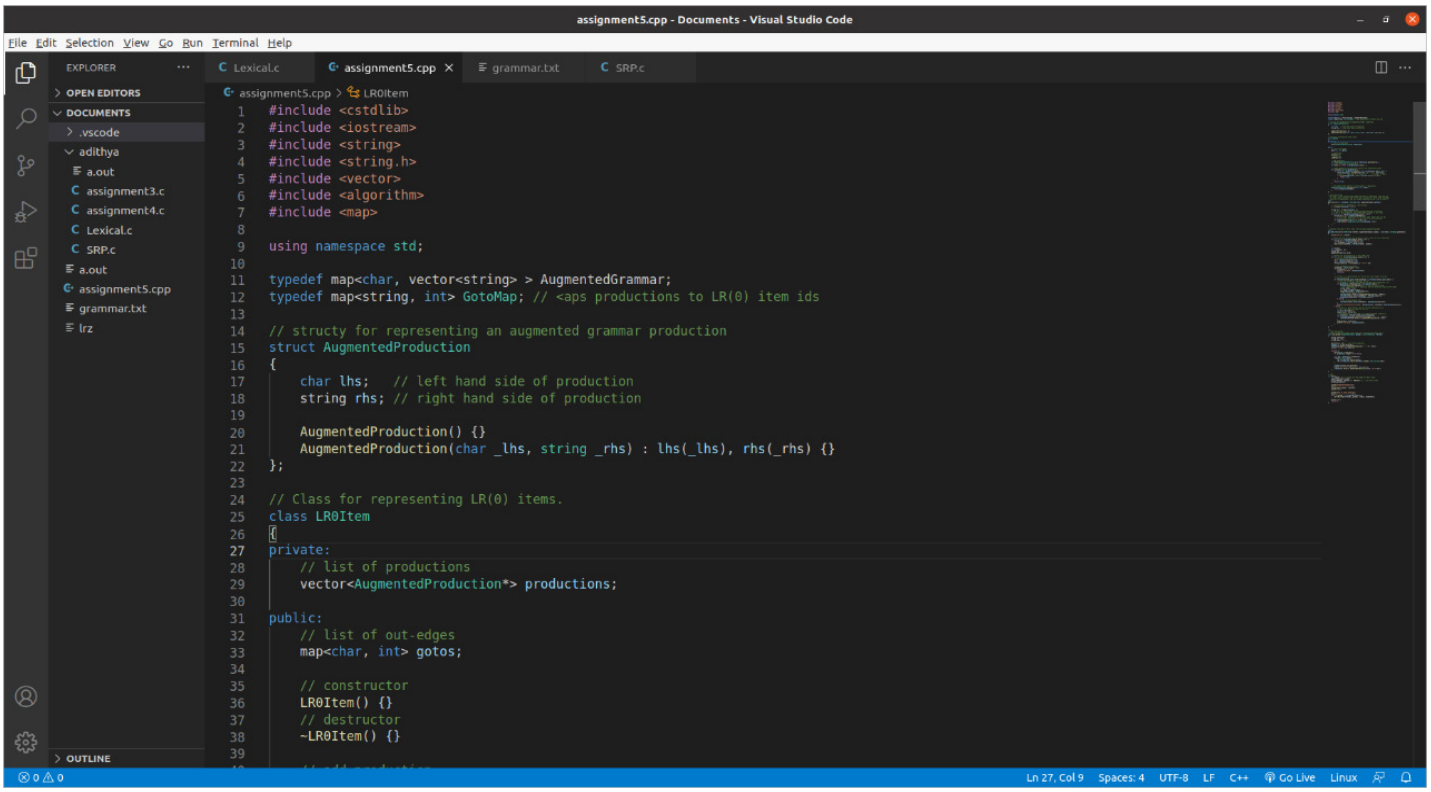
An LR parser makes shift-reduce decisions by maintaining states to keep track of where we are in a parse. States represent sets of "items." An LR(O) item (item for short) of a grammar G is a production of G with a dot at some position of the body. Thus, production A -> XYZ yields the four items

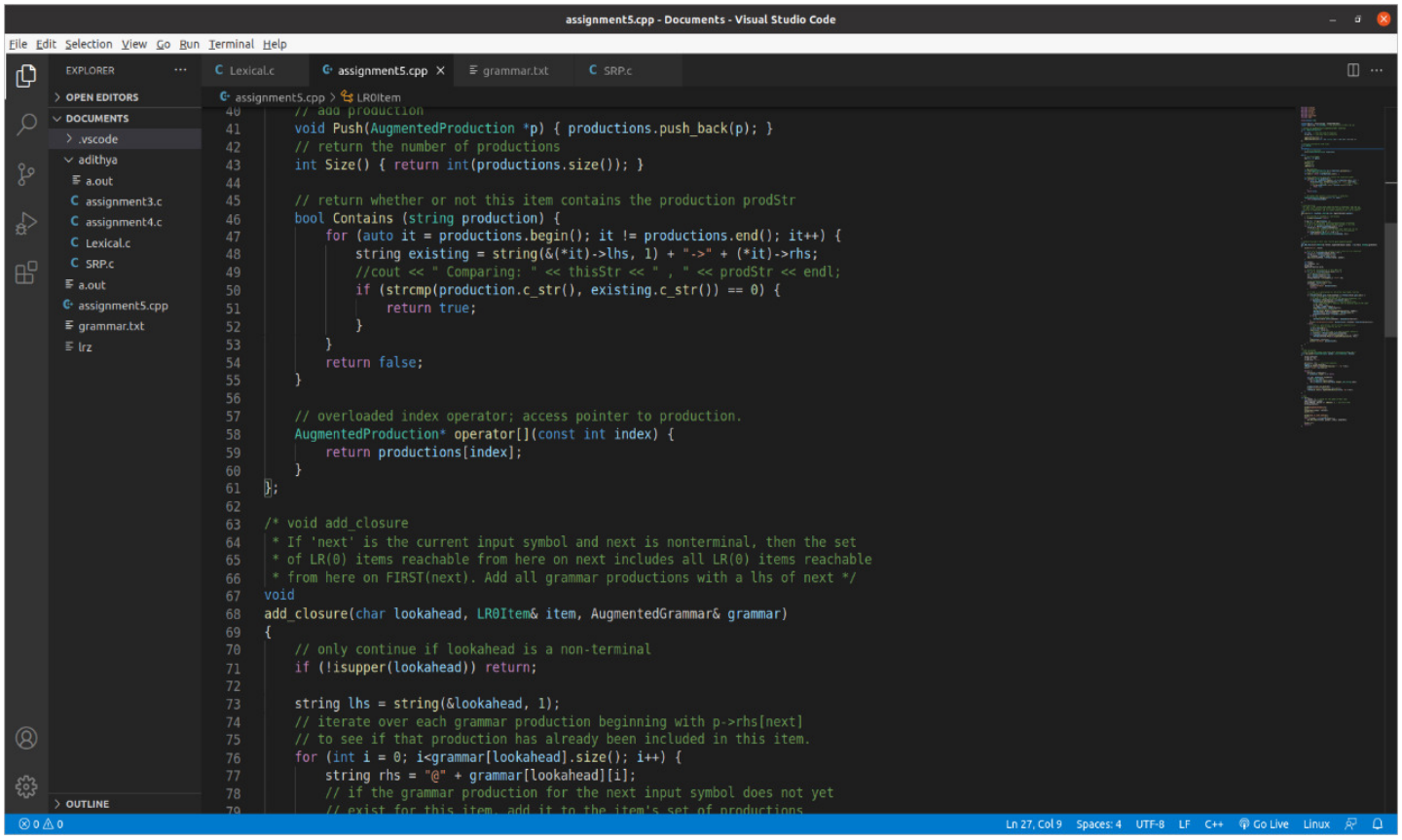
A -> ·XYZ
A -> X ·YZ
A -> XY· Z
A -> XYZ·

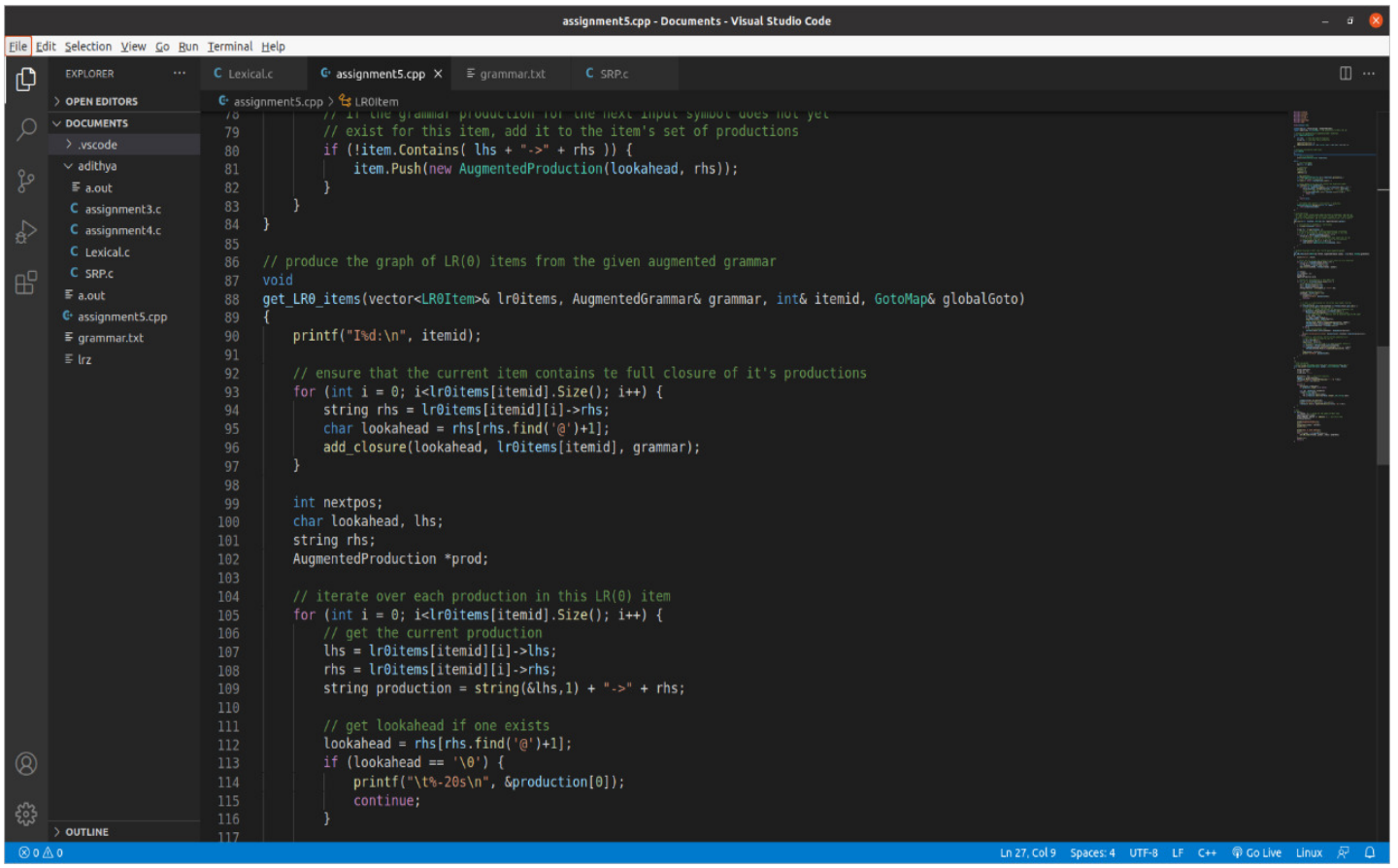
The production A ->ϵ generates only one item, A -> . _
Intuitively, an item indicates how much of a production we have seen at a given point in the parsing process. For example, the item A -> ·XY Z indicates that we hope to see a string derivable from XY Z next on the input. Item A -> X · Y Z indicates that we have just seen on the input a string derivable from X and that we hope next to see a string derivable from Y Z. Item A ->XY Z· indicates that we have seen the body XY Z and that it may be time to reduce XYZ to A.

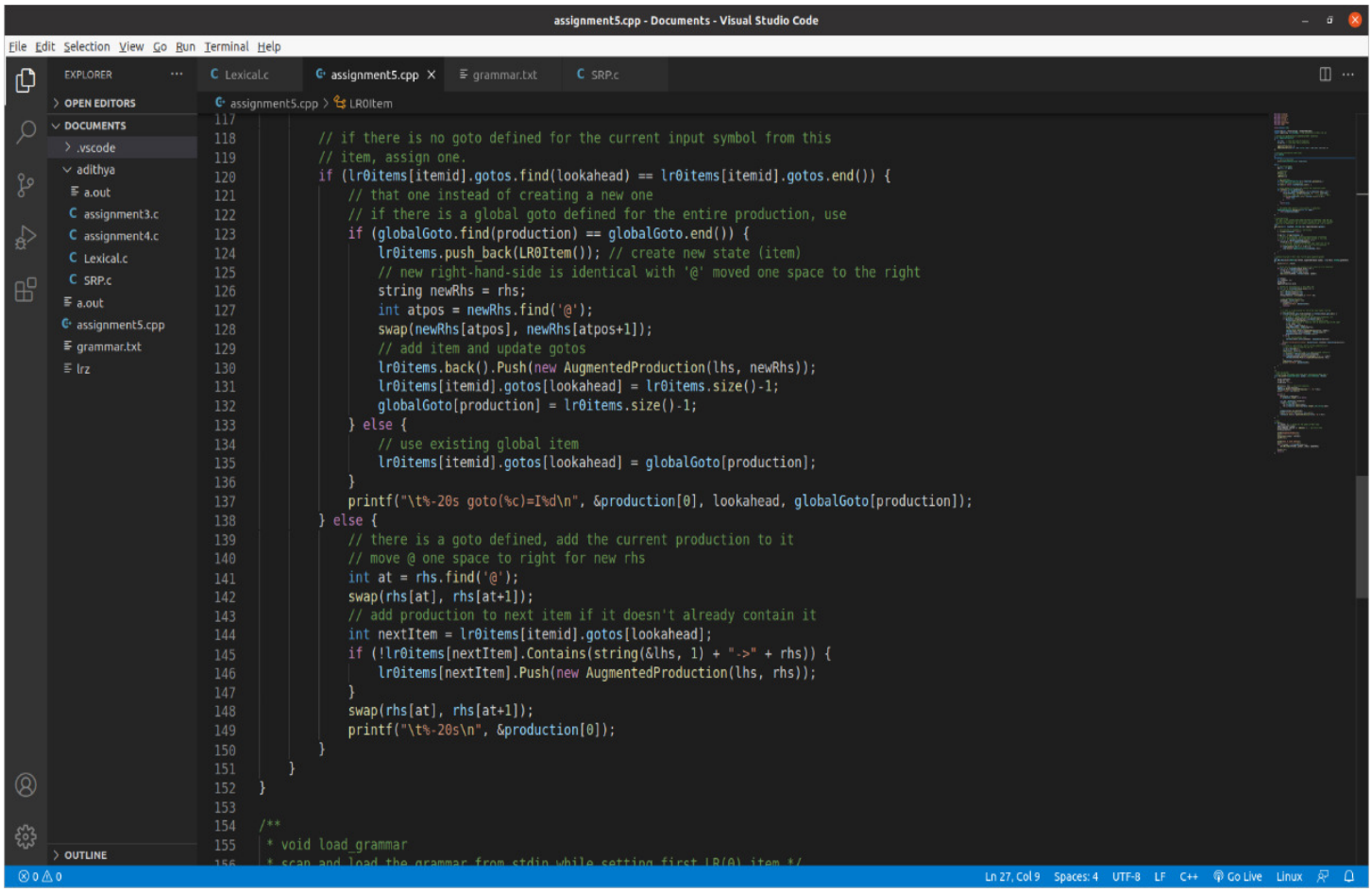
RESULTS :

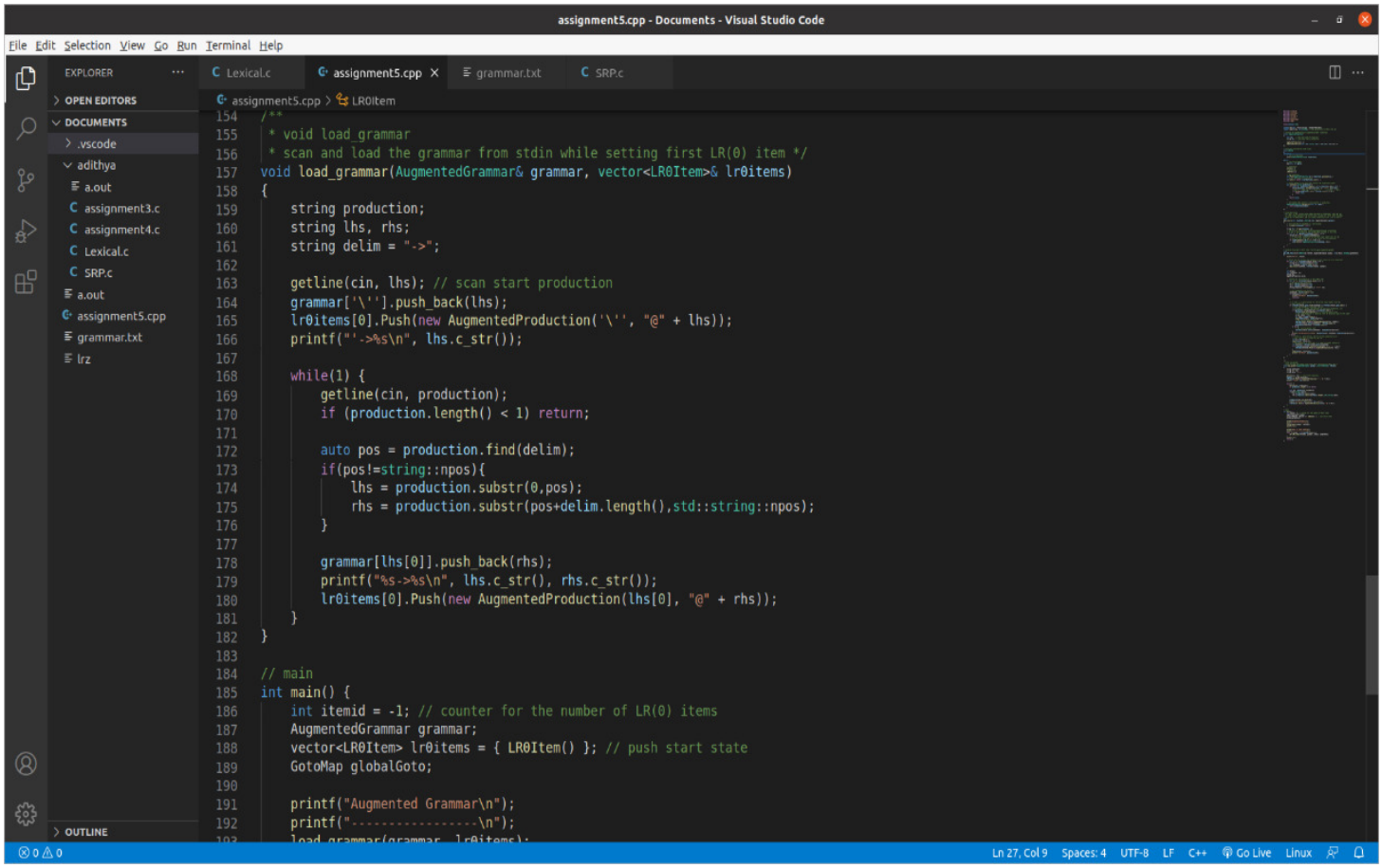
Screenshot of the Code :

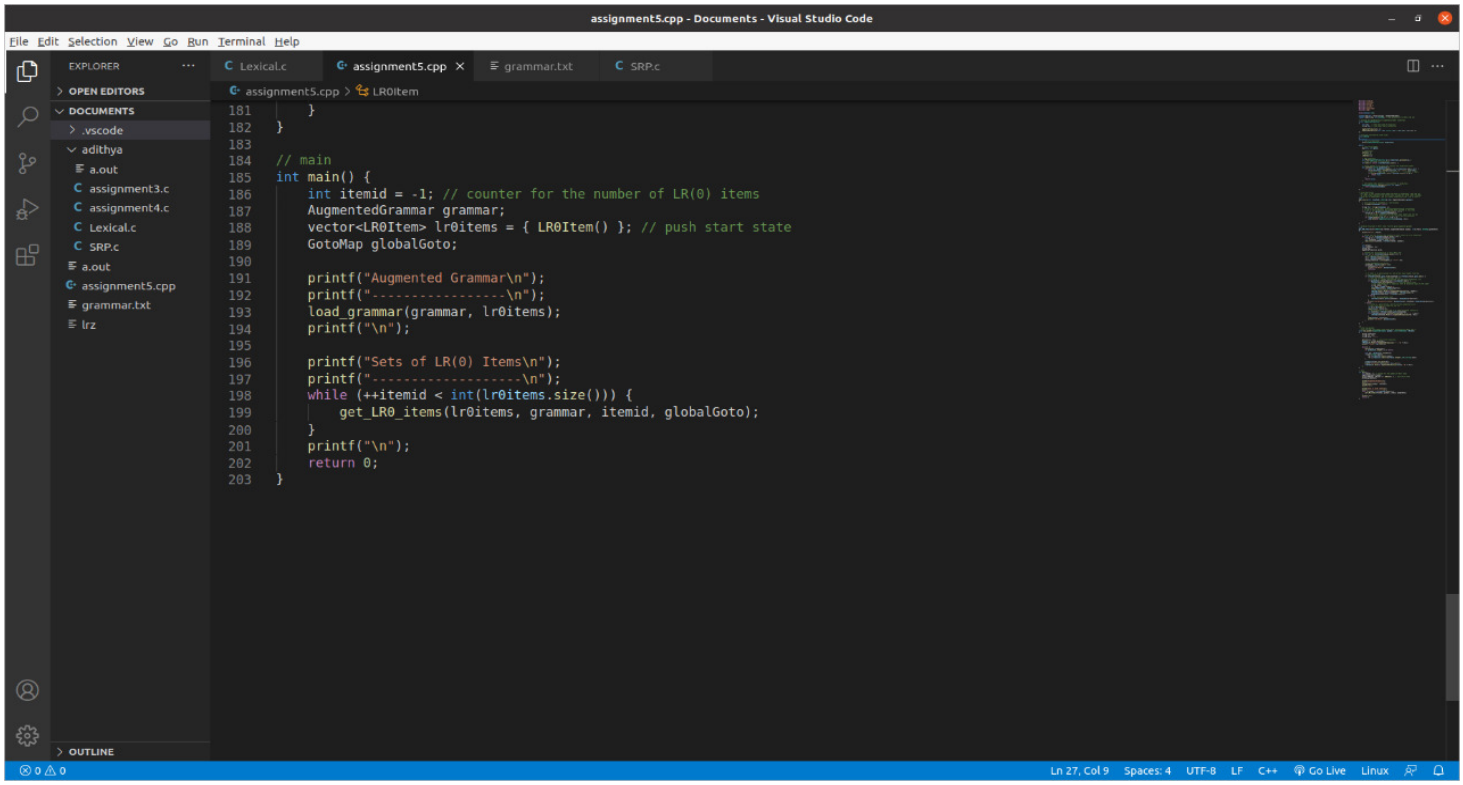



assignment5.cpp - Documents - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
DOCUMENTS
.vscode
adithya
a.out
assignment3.c
assignment4.c
Lexical.c
SRP.c
a.out
assignment5.cpp
grammar.txt
lrz
Lexical.c
assignment5.cpp
grammar.txt
SRP.c
assignment5.cpp > LR0Item
// add production
void Push(AugmentedProduction *p) { productions.push_back(p); }
// return the number of productions
int Size() { return int(productions.size()); }
// return whether or not this item contains the production prodStr
bool Contains (string production) {
for (auto it = productions.begin(); it != productions.end(); it++) {
string existing = string(&(*it)->lhs, 1) + "->" + (*it)->rhs;
//cout << " Comparing: " << thisStr << " , " << prodStr << endl;
if (strcmp(production.c_str(), existing.c_str()) == 0) {
return true;
}
}
return false;
}
// overloaded index operator; access pointer to production.
AugmentedProduction* operator[](const int index) {
return productions[index];
}
};
/* void add_closure
* If 'next' is the current input symbol and next is nonterminal, then the set
* of LR(0) items reachable from here on next includes all LR(0) items reachable
* from here on FIRST(next). Add all grammar productions with a lhs of next */
void
add_closure(char lookahead, LR0Item& item, AugmentedGrammar& grammar)
{
// only continue if lookahead is a non-terminal
if (!isupper(lookahead)) return;
string lhs = string(&lookahead, 1);
// iterate over each grammar production beginning with p->rhs[next]
// to see if that production has already been included in this item.
for (int i = 0; i<grammar[lookahead].size(); i++) {
string rhs = "@" + grammar[lookahead][i];
// if the grammar production for the next input symbol does not yet
// exist for this item, add it to the item's set of productions
OUTLINE
Ln 27, Col 9 Spaces: 4 UTF-8 LF C++ Go Live Linux

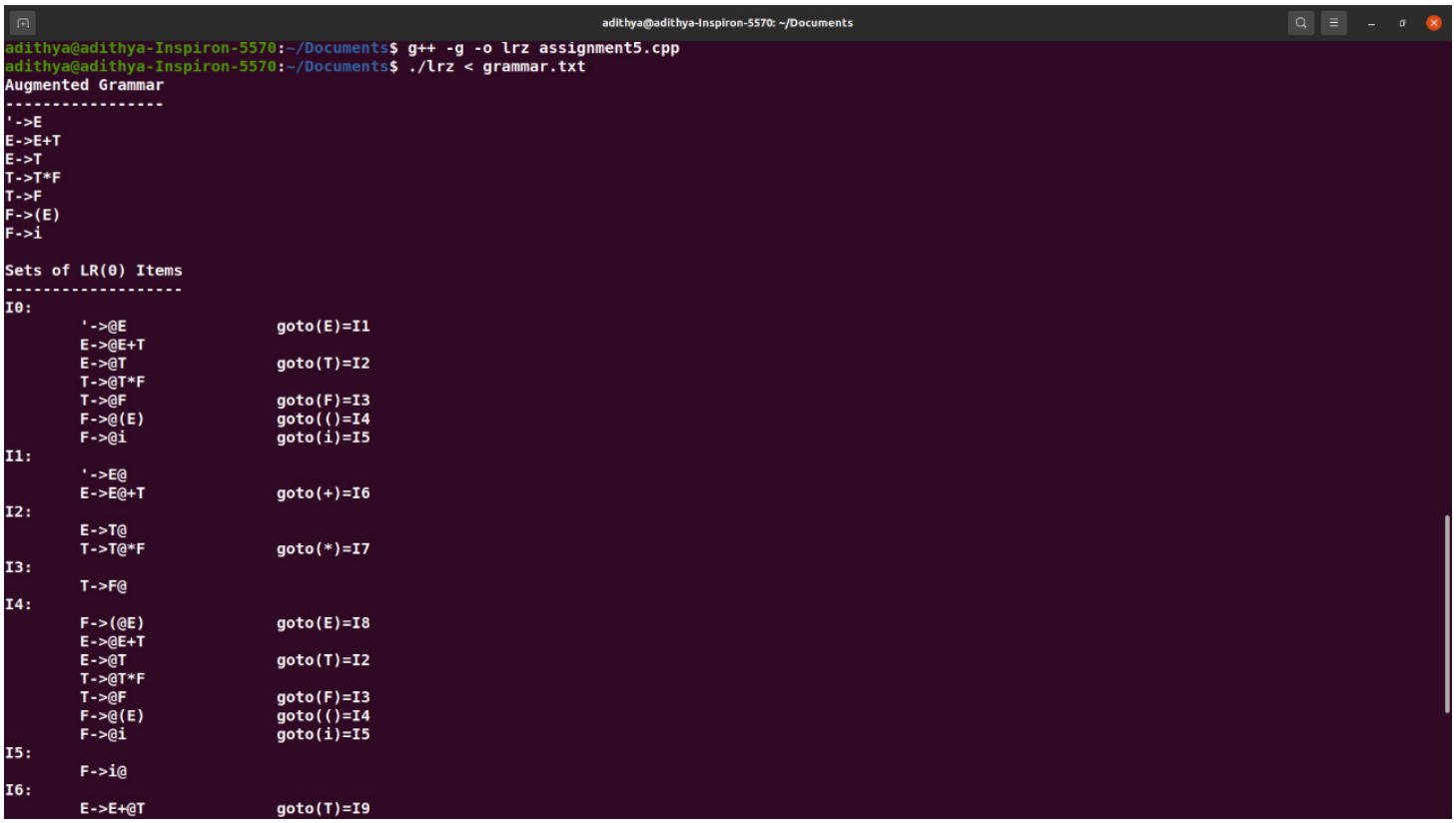

assignment5.cpp - Documents - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
DOCUMENTS
.vscode
adithya
a.out
assignment3.c
assignment4.c
Lexical.c
SRP.c
a.out
assignment5.cpp
grammar.txt
lrz
Lexical.c
assignment5.cpp
grammar.txt
SRP.c
assignment5.cpp > LR0Item
// if the grammar production for the next input symbol does not yet
// exist for this item, add it to the item's set of productions
if (!item.Contains(lhs + "->" + rhs)) {
item.Push(new AugmentedProduction(lookahead, rhs));
}
}
}
// produce the graph of LR(0) items from the given augmented grammar
void
get_LR0_items(vector<LR0Item>& lr0items, AugmentedGrammar& grammar, int& itemid, GotoMap& globalGoto)
{
printf("I%d:\n", itemid);
// ensure that the current item contains te full closure of it's productions
for (int i = 0; i<lr0items[itemid].Size(); i++) {
string rhs = lr0items[itemid][i]->rhs;
char lookahead = rhs[rhs.find('@')+1];
add_closure(lookahead, lr0items[itemid], grammar);
}
int nextpos;
char lookahead, lhs;
string rhs;
AugmentedProduction *prod;
// iterate over each production in this LR(0) item
for (int i = 0; i<lr0items[itemid].Size(); i++) {
// get the current production
lhs = lr0items[itemid][i]->lhs;
rhs = lr0items[itemid][i]->rhs;
string production = string(&lhs,1) + "->" + rhs;
// get lookahead if one exists
lookahead = rhs[rhs.find('@')+1];
if (lookahead == '\0') {
printf("\t%-20s\n", &production[0]);
continue;
}
OUTLINE
Ln 27, Col 9 Spaces: 4 UTF-8 LF C++ Go Live Linux

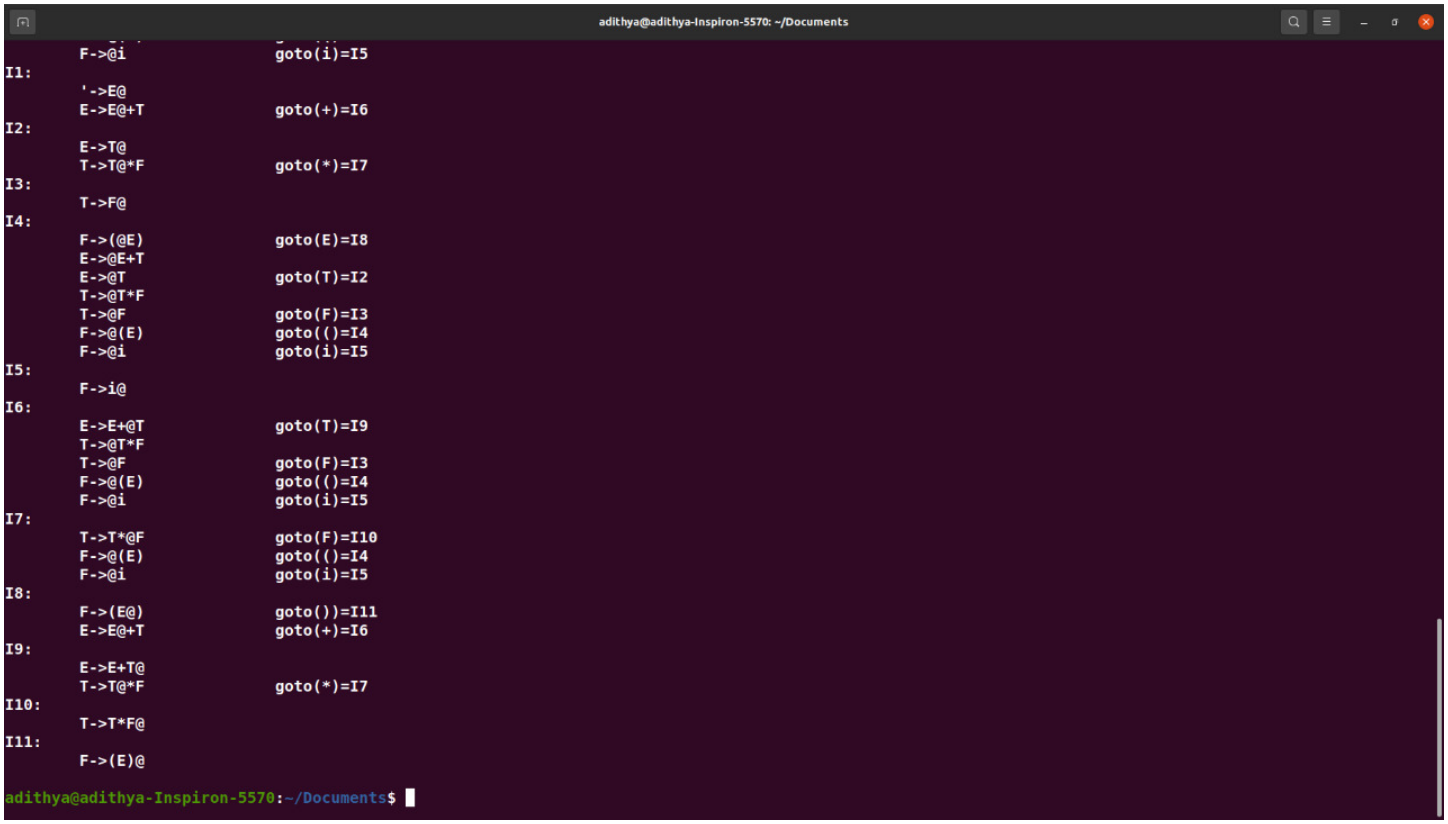

assignment5.cpp - Documents - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
DOCUMENTS
.vscode
adithya
a.out
assignment3.c
assignment4.c
Lexical.c
SRP.c
a.out
assignment5.cpp
grammar.txt
lrz
OUTLINE
Lexical.c
assignment5.cpp
grammar.txt
SRP.c
assignment5.cpp > LR0Item
// if there is no goto defined for the current input symbol from this
// item, assign one.
if (lr0items[itemid].gotos.find(lookahead) == lr0items[itemid].gotos.end()) {
// that one instead of creating a new one
// if there is a global goto defined for the entire production, use
if (globalGoto.find(production) == globalGoto.end()) {
lr0items.push_back(LR0Item()); // create new state (item)
// new right-hand-side is identical with '@' moved one space to the right
string newRhs = rhs;
int atpos = newRhs.find('@');
swap(newRhs[atpos], newRhs[atpos+1]);
// add item and update gotos
lr0items.back().Push(new AugmentedProduction(lhs, newRhs));
lr0items[itemid].gotos[lookahead] = lr0items.size()-1;
globalGoto[production] = lr0items.size()-1;
} else {
// use existing global item
lr0items[itemid].gotos[lookahead] = globalGoto[production];
}
printf("\t%-20s goto(%c)=I%d\n", &production[0], lookahead, globalGoto[production]);
} else {
// there is a goto defined, add the current production to it
// move @ one space to right for new rhs
int at = rhs.find('@');
swap(rhs[at], rhs[at+1]);
// add production to next item if it doesn't already contain it
int nextItem = lr0items[itemid].gotos[lookahead];
if (!lr0items[nextItem].Contains(string(&lhs, 1) + "->" + rhs)) {
lr0items[nextItem].Push(new AugmentedProduction(lhs, rhs));
}
swap(rhs[at], rhs[at+1]);
printf("\t%-20s\n", &production[0]);
}
}
}
/**
* void load_grammar
* scan and load the grammar from stdin while setting first LR(0) item */
Ln 27, Col 9 Spaces: 4 UTF-8 LF C++ Go Live Linux


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
OUTLINE
Lexical.c
assignment5.cpp
grammar.txt
SRP.c
assignment5.cpp > LR0Item
/**
* void load_grammar
* scan and load the grammar from stdin while setting first LR(0) item */
void load_grammar(AugmentedGrammar& grammar, vector<LR0Item>& lr0items)
{
string production;
string lhs, rhs;
string delim = "->";
getline(cin, lhs); // scan start production
grammar['\''].push_back(lhs);
lr0items[0].Push(new AugmentedProduction('\'', "@" + lhs));
printf("'->%s\n", lhs.c_str());
while(1) {
getline(cin, production);
if (production.length() < 1) return;
auto pos = production.find(delim);
if(pos!=string::npos){
lhs = production.substr(0,pos);
rhs = production.substr(pos+delim.length(),std::string::npos);
}
grammar[lhs[0]].push_back(rhs);
printf("%s->%s\n", lhs.c_str(), rhs.c_str());
lr0items[0].Push(new AugmentedProduction(lhs[0], "@" + rhs));
}
}
// main
int main() {
int itemid = -1; // counter for the number of LR(0) items
AugmentedGrammar grammar;
vector<LR0Item> lr0items = { LR0Item() }; // push start state
GotoMap globalGoto;
printf("Augmented Grammar\n");
printf("----------------\n");
Ln 27, Col 9 Spaces: 4 UTF-8 LF C++ Go Live Linux



Screenshot of the Results in the Terminal :




adithya@adithya-Inspiron-5570: ~/Documents
F->@i goto(i)=I5
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
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$