Final Assessment : Compiler Lab

CSE-0302 Summer 2021

Md.Arif Hossen, UG02-50-19-005
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
arifsohag2500@gmail.com

***Abstract*—Main theme of your assignment or academic projects.**

n

***Index Terms*—The word mostly used in your report.**

I. Introduction

Compiler is a software which converts a program written in high level language (Source Language) to low level language. We know a computer is a logical assembly of Software and Hardware. The hardware knows a language, that is hard for us to grasp, consequently we tend to write programs in high-level language, that is much less complicated for us to comprehend and maintain in thoughts. Now these programs go through a series of transformation so that they can readily be used by machines. This is where language procedure systems come handy.

A syntactical error in Java code is one in which the language you use to create your code is incorrect. For example, if you try to create an if statement that doesn't include the condition in parentheses, even when the condition is present on the same line as the if statement, that's a syntax error. CFG stands for context-free grammar. It is is a formal grammar which is used to generate all possible patterns of strings in a given formal language.

Parser is a compiler that is used to break the data into smaller elements coming from lexical analysis phase. A parser takes input in the form of sequence of tokens and produces output in the form of parse tree. Parsing is of two types: top down parsing and bottom up parsing. Predictive parser is a recursive descent parser, which has the capability to predict which production is to be used to replace the input string. The predictive parser does not suffer from backtracking.

To accomplish its tasks, the predictive parser uses a look-ahead pointer, which points to the next input symbols. To make the parser back-tracking free, the predictive parser puts some constraints on the grammar and accepts only a class of grammar known as LL(k) grammar.

II. Literature Review

Before this project i read some books which wrote by **Alfred Aho (Author), Monica Lam (Author), Ravi Sethi (Author), Jeffrey Ullman (Author)** and Bangla compiler design book which wrote by**Tamim shahriar subin** .it helps me a lot ,because before i attend to compiler lab class i don't know how to use FILE,pointer ,Header etc.But my honourable teacher **Khan Md.Hasib** suggest me to basic of c language .and he also helps us to understand this topic.

III. Proposed Methodology

The methodology you work, explain here with code and other items. Syntax is the spelling and grammar of a programming language. Programming languages have rules for spelling, punctuation and grammar, just like the English language. In programming, a syntax error occurs when:
1.there is a spelling mistake.
2.there is a grammatical mistake.
Types of syntax error More than one type of syntax error may exist. There may be:
1.incorrectly spelled statements.
2.incorrectly spelled variables.
3.missing punctuation (quotes, brackets, etc).
Any one or more of these errors may exist in a program, and each will cause the program to crash or not run at all.

Definition A context-free grammar (CFG) consisting of a finite set of grammar rules is a quadruple (N, T, P, S) where

N is a set of non-terminal symbols.

T is a set of terminals where N T = NULL.

P is a set of rules, P: N $\rightarrow$ (N T)*, i.e., the left-hand side of the production rule P does have any right context or left context.

S is the start symbol.

Example

The grammar (A, a, b, c, P, A), P : A $\rightarrow$ aA, A $\rightarrow$ abc. The grammar (S, a, b, a, b, P, S), P: S $\rightarrow$ aSa, S $\rightarrow$ bSb, S $\rightarrow$ The grammar (S, F, 0, 1, P, S), P: S $\rightarrow$ 00S — 11F, F $\rightarrow$ 00F —

parse tree is an ordered rooted tree that graphically represents the semantic information a string derived from a context-free grammar.

Representation Technique
1.Root vertex Must be labeled by the start symbol.

2.Vertex Labeled by a non-terminal symbol.

3.Leaves Labeled by a terminal symbol or

Parsing, syntax analysis, or syntactic analysis is the process of analyzing a string of symbols, either in natural language, computer languages or data structures, conforming to the rules of a formal grammar. The term parsing comes from Latin pars (orationis), meaning part (of speech).

IV. Conclusion and Future Work

Firstly i learn c basic,pointer,file,structure then i start this project to do. i successfully run it and get the actual output .And hopefully i wil have done the allthis kind of problem solved.

Acknowledgment

I would like to thank my honourable**Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

References

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, “Title of paper if known,” unpublished.

[5] R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

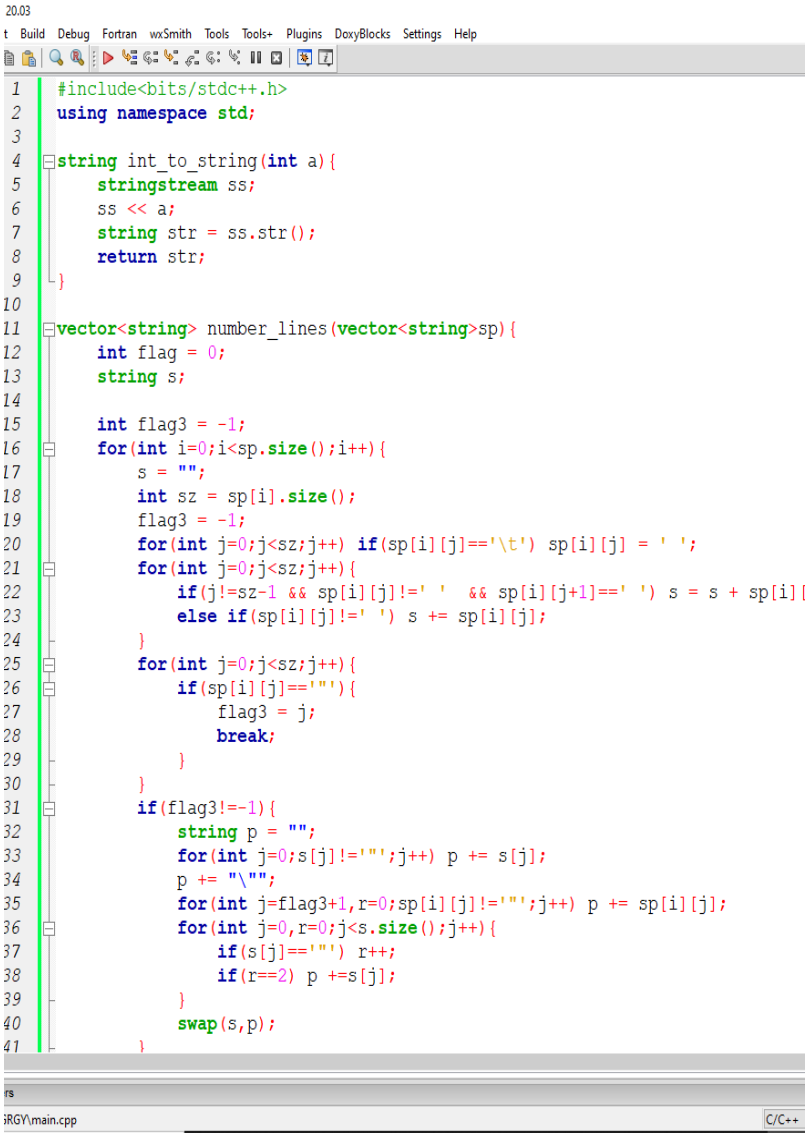


Fig. 1. Assignment 4 Code : Detecting Simple Syntax Errors

```
40              swap(s,p);
41          }
42          swap(sp[i],s);
43      }
44
45      vector<string>sp1;
46
47      int flag1 = 0,flag2=0;
48      for(int i=0;i<sp.size();i++){
49          string str = int_to_string(i+1);
50          int sz = sp[i].size();
51          if(sz==0){
52              sp1.push_back(str);
53              continue;
54          }
55          for(int j=0;j<sz;j++){
56              if(j!=sz-1 && sp[i][j]=='/' && sp[i][j+1]=='
57                  flag1 = 1;
58                  for(int k=0;k<j;k++){
59                      cout<<sp[i][k];
60                      cerr<<sp[i][k];
61                  }
62                  break;
63              }
64              if(j!=sz-1 && sp[i][j]=='/' && sp[i][j+1]=='
65                  flag2 = 1;
66                  for(int k=0;k<j;k++){
67                      cout<<sp[i][k];
68                      cerr<<sp[i][k];
69                  }
70              }
71              if(j!=sz-1 && sp[i][j]=='*' && sp[i][j+1]=='
72                  flag2 = 0;
73                  flag1 = 1;
74                  break;
75              }
76          }
77          if(flag1){
78              flag1 = 0;
79              sp1.push_back(str);
80              continue;
```

Logs & others

MAT-0201\GRGY\main.cpp

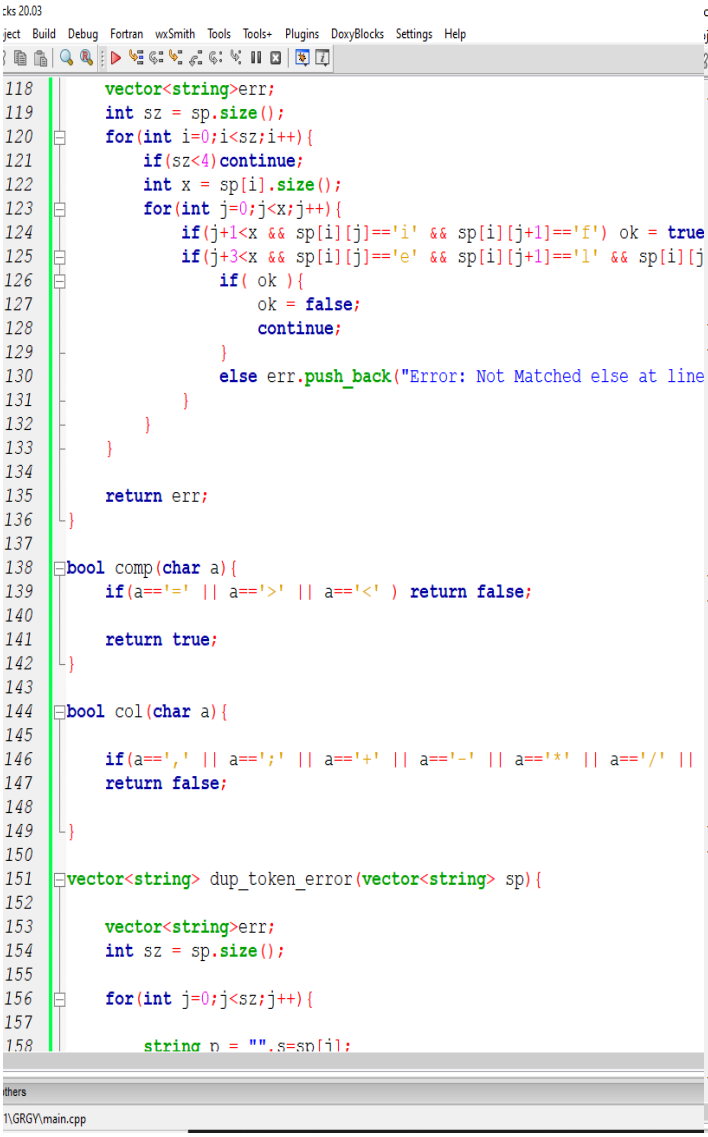
Fig. 2. Assignment 4 Code: Detecting Simple Syntax Errors

```
79              sp1.push_back(str);
80              continue;
81          }
82          if(flag2){
83              sp1.push_back(str);
84              continue;
85          }
86          str = str + " " + sp[i];
87          sp1.push_back(str);
88      }
89
90      return sp1;
91
92  }
93
94  vector<string> paranthesis_error(vector<string> sp){
95
96      stack<int>st;
97      vector<string>err;
98
99      for(int i=0;i<sp.size();i++){
100         for(int j=0;j<sp[i].size();j++){
101             if(sp[i][j]=='{') st.push(i+1);
102             else if(sp[i][j]=='}'){
103                 if( !st.empty() ) st.pop();
104                 else err.push_back("Error: Misplaced '}' at line "+int
105             }
106         }
107     }
108
109     if( !st.empty() ) err.push_back("Error: Not Balanced Parentheses a
110
111     return err;
112 }
113
114
115 vector<string> if_else_error(vector<string> sp){
116
117     bool ok = false;
118     vector<string>err;
119     int sz = sp.size();
```

& others

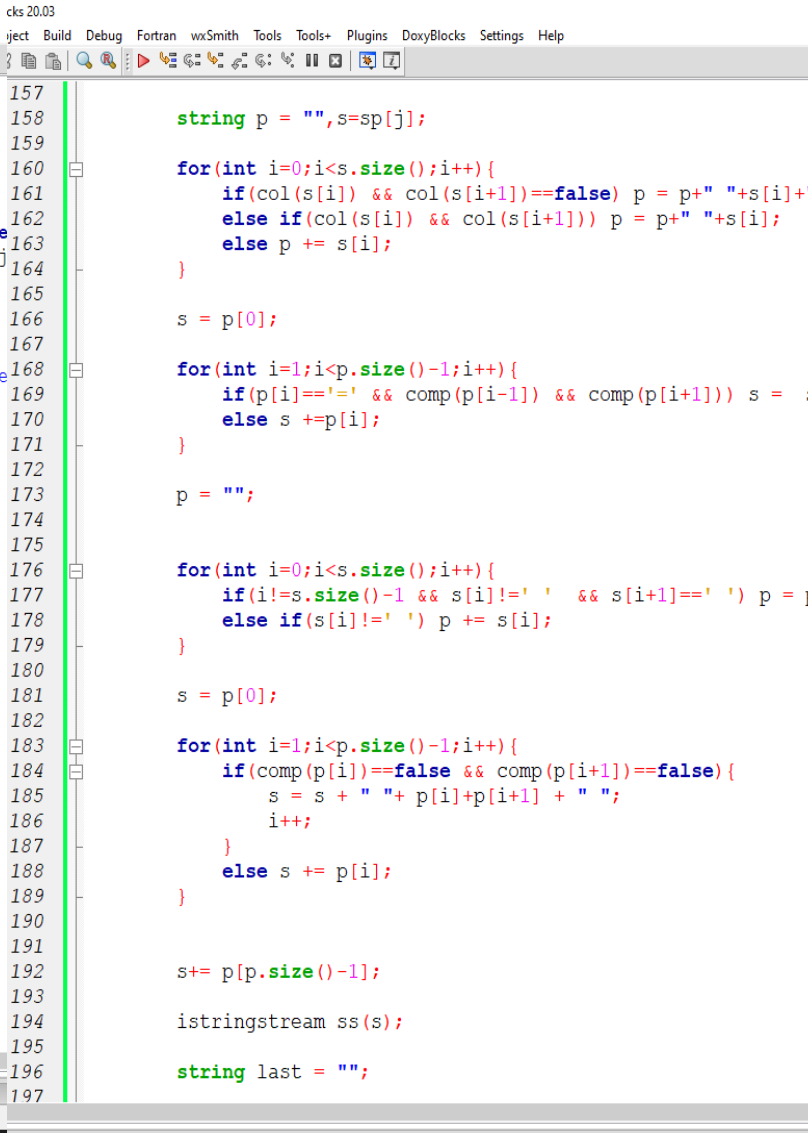
0201\GRGY\main.cpp

Fig. 3. Assignment 4 Code: Detecting Simple Syntax Errors



```
118     vector<string>err;
119     int sz = sp.size();
120     for(int i=0;i<sz;i++){
121         if(sz<4)continue;
122         int x = sp[i].size();
123         for(int j=0;j<x;j++){
124             if(j+1<x && sp[i][j]=='i' && sp[i][j+1]=='f') ok = true
125             if(j+3<x && sp[i][j]=='e' && sp[i][j+1]=='l' && sp[i][j
126                 if( ok ){
127                     ok = false;
128                     continue;
129                 }
130                 else err.push_back("Error: Not Matched else at line
131             }
132         }
133     }
134
135     return err;
136 }
137
138 bool comp(char a){
139     if(a=='=' || a=='>' || a=='<' ) return false;
140
141     return true;
142 }
143
144 bool col(char a){
145
146     if(a==',' || a==';' || a=='+' || a=='-' || a=='*' || a=='/' ||
147     return false;
148
149 }
150
151 vector<string> dup_token_error(vector<string> sp){
152
153     vector<string>err;
154     int sz = sp.size();
155
156     for(int j=0;j<sz;j++){
157
158         string p = "",s=sp[j];
```

Fig. 4. Assignment 4 Code: Detecting Simple Syntax Errors



```
157
158         string p = "",s=sp[j];
159
160         for(int i=0;i<s.size();i++){
161             if(col(s[i]) && col(s[i+1])==false) p = p+" "+s[i]+
162             else if(col(s[i]) && col(s[i+1])) p = p+" "+s[i];
163             else p += s[i];
164         }
165
166         s = p[0];
167
168         for(int i=1;i<p.size()-1;i++){
169             if(p[i]=='=' && comp(p[i-1]) && comp(p[i+1])) s =
170             else s +=p[i];
171         }
172
173         p = "";
174
175
176         for(int i=0;i<s.size();i++){
177             if(i!=s.size()-1 && s[i]!=' '  && s[i+1]==' ') p =
178             else if(s[i]!=' ') p += s[i];
179         }
180
181         s = p[0];
182
183         for(int i=1;i<p.size()-1;i++){
184             if(comp(p[i])==false && comp(p[i+1])==false){
185                 s = s + " "+ p[i]+p[i+1] + " ";
186                 i++;
187             }
188             else s += p[i];
189         }
190
191
192         s+= p[p.size()-1];
193
194         istringstream ss(s);
195
196         string last = "";
197
```

Fig. 5. Assignment 4 Code: Detecting Simple Syntax Errors

Fig. 6. Assignment 4 Code: Detecting Simple Syntax Errors

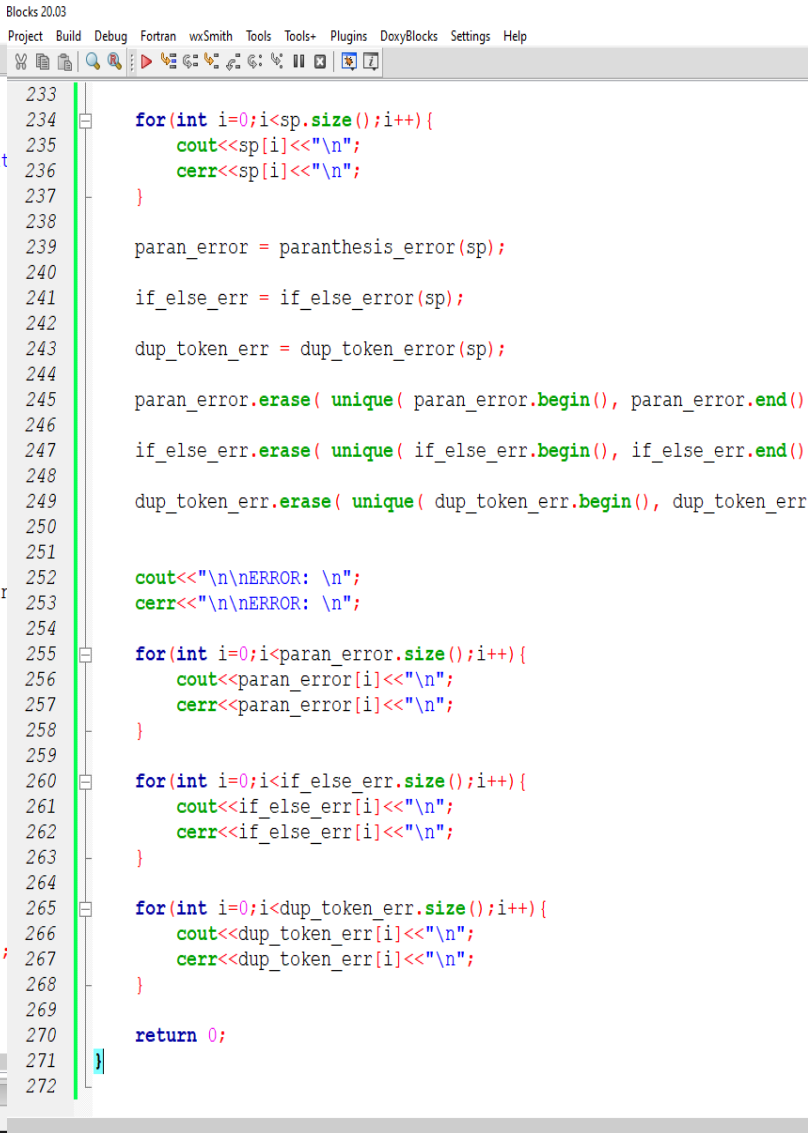


Fig. 7. Assignment 4 Code: Detecting Simple Syntax Errors

```
/tmp/o2OxtVztTm.o
input

output:
Recognized tokens in the


ERROR:
/* A program fragment*/

float x1 = 3.125;;
/* Definition of functio
double f1(int int x)
{if(x<x1)
    double z;
else z =      0.01+x*5.5;
  else return z;
}
/* Beginning of 'main' *
int main(void)
{
int n1; double z;
{{ n1=25; z=f1(n1);}
```

Fig. 8. Input Assignment 4 : Detecting Simple Syntax Errors

```
output:
Recognized tokens in the lines of
ERROR:
of function f1 */

double f1(int int x)

{if(x<x1)

    double z;

else z =      0.01+x*5.5;}}

  else return z;

}

/* Beginning of 'main' */

int main(void)

{

int n1; double z;

{{ n1=25; z=f1(n1);}

output:

Recognized tokens in the lines of
```

Fig. 9. Output Assignment 4 : Detecting Simple Syntax Errors

```
ERROR:


dash: 2: of: not found
dash: 4: Syntax error: "(" unexpected
dash: 5: Syntax error: word unexpected (expecting ")")
dash: 6: Syntax error: newline unexpected
dash: 6: double: not found
dash: 8: Syntax error: "else" unexpected
dash: 9: Syntax error: "else" unexpected
dash: 10: Syntax error: "}" unexpected
dash: 12: /app: Permission denied
dash: 14: Syntax error: "(" unexpected
> > > > dash: 19: Syntax error: "(" unexpected (expecting "}")
dash: 20: output:: not found
dash: 22: Recognized: not found
dash: 24: ERROR:: not found
```

Fig. 10. output Assignment 4 : Detecting Simple Syntax Errors

Code::Blocks 20.03

arch Project Build Debug Fortran wxSmith Tools To

```
1   #include<bits/stdc++
2   using namespace std;
3
4   int i=0,f=0,l;
5
6   string st;
7
8   void A() {
9       if (st[i] == 'a'
10          i++;
11          f=1;
12      }
13      else {
14          f=0;
15          return;
16      }
17      if (i<l-1) A();
18  }
19
20  void B() {
21      if (st[i] == 'b'
22          i++;
23          f=1;
24          return;
25      }
26      else {
27          f=0;
28          return;
29      }
30  }
31
32  void S() {
33      if (st[i] == 'b'
34          i++;
35          f = 1;
36          return;
37      }
38      else {
39          A();
40          if (f) { B()
41      }
```

Fig. 11. Assignment 5 Code: Use of CFGs for Parsing

```
40          if (f) { B(); r
41      }
42  }
43
44  int main(){
45
46      freopen("i1.txt","r
47      freopen("o1.txt","w
48
49      while(getline(cin,s
50
51          f = 0;
52          i = 0;
53
54          l = st.size();
55
56          S();
57
58          if(l==i && f){
59              cout<<"vali
60          }
61          else{
62              cout<<"inva
63          }
64      }
65
66  }
67
68
69
```

Fig. 12. Assignment 5 Code: Use of CFGs for Parsing

```
b
ab
aab
aaab
```

Fig. 13. Input Assignment 5 : Use of CFGs for Parsing

```
1    valid
2    valid
3    valid
4    valid
```

Fig. 14. OUTPUT Assignment 5 : Use of CFGs for Parsing

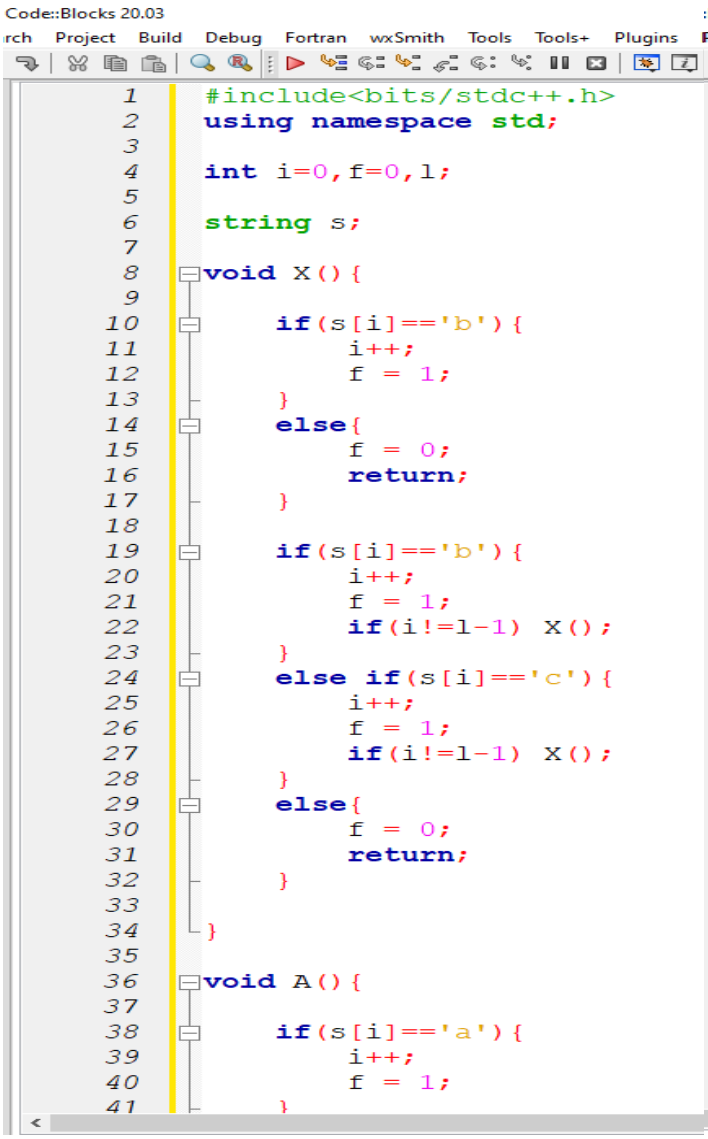


Fig. 15. Assignment 5 Code: Use of CFGs for Parsing 2

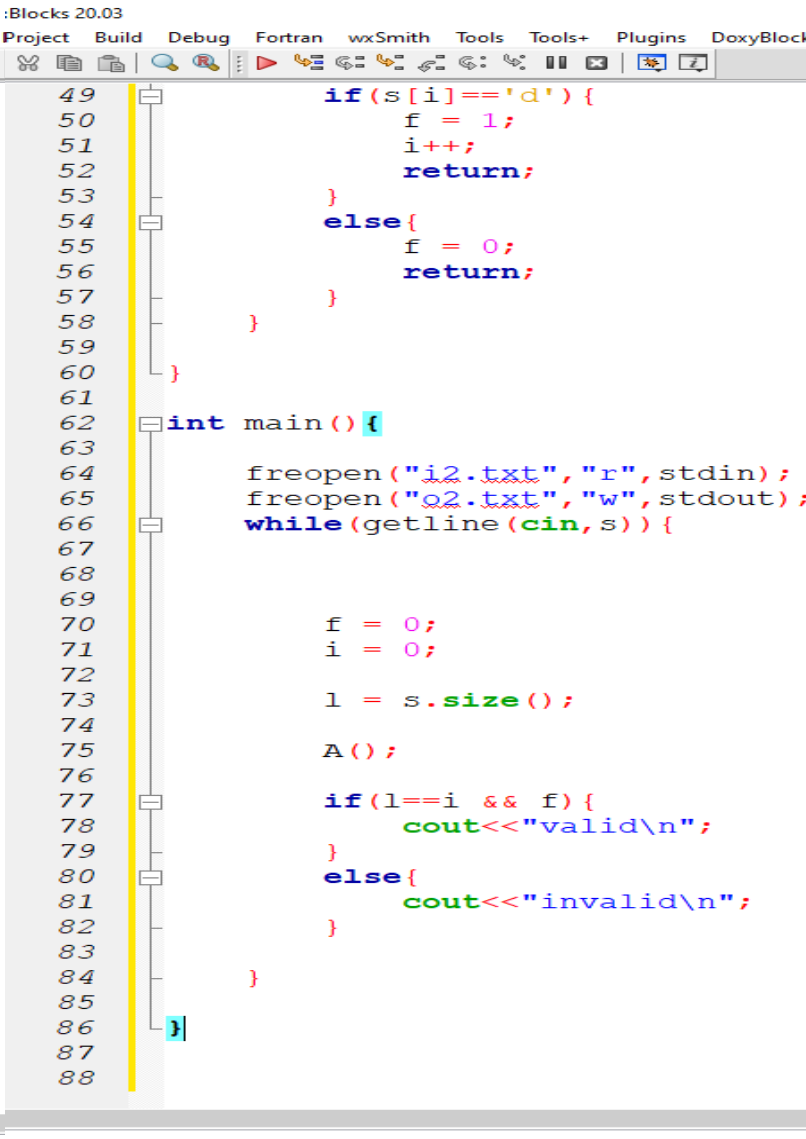


Fig. 16. Assignment 5 Code: Use of CFGs for Parsing 2

```
asasfas
bba
ba
abbd
```

Fig. 17. Input Assignment 5 : Use of CFGs for Parsing 2

```
invalid
invalid
invalid
valid
```

Fig. 18. OUTPUT Assignment 5 : Use of CFGs for Parsing 2

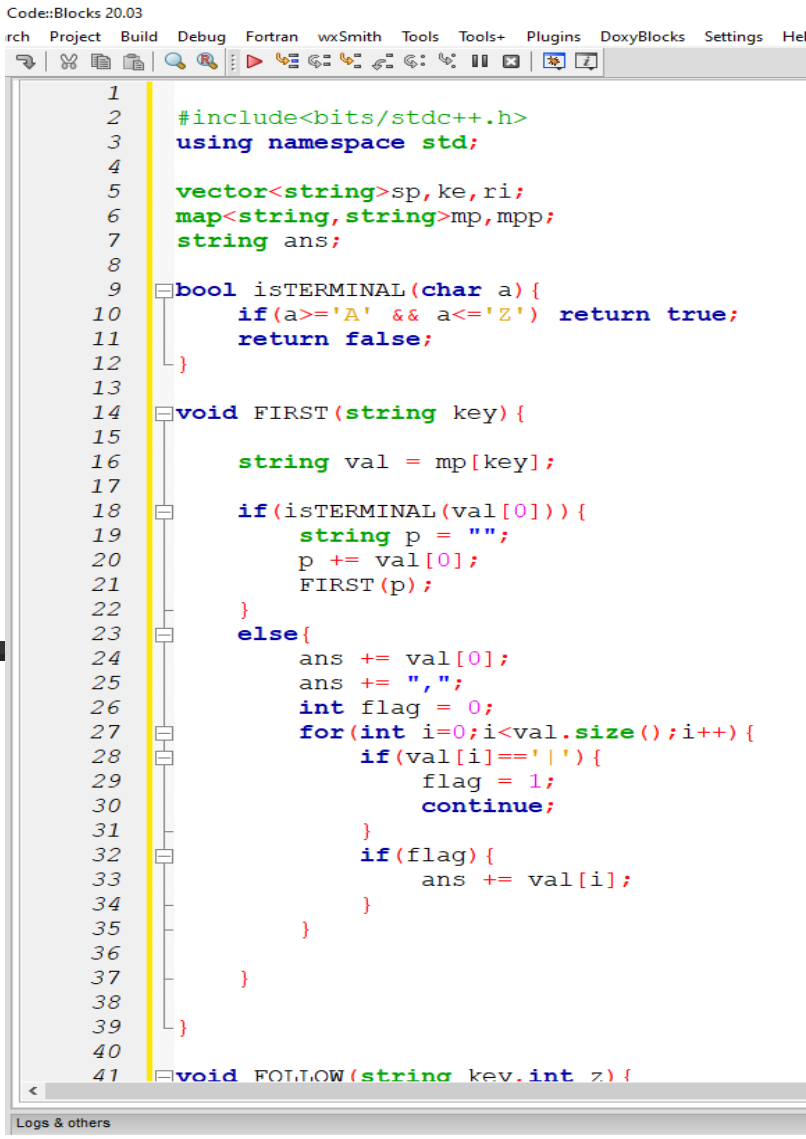
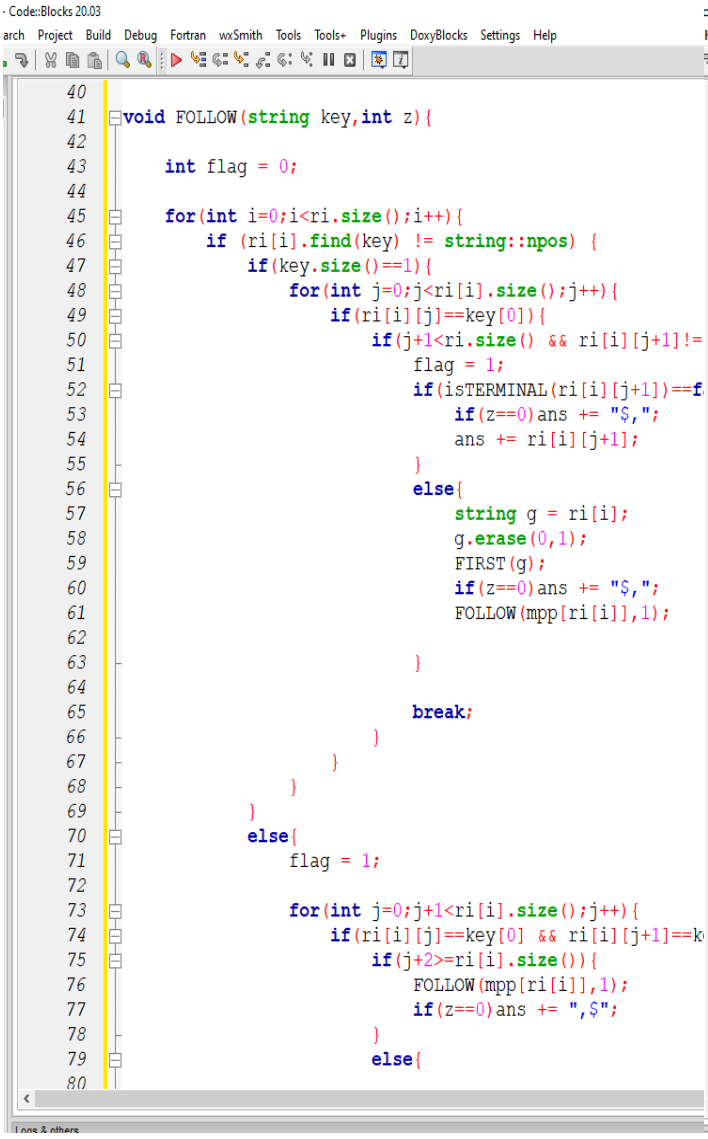
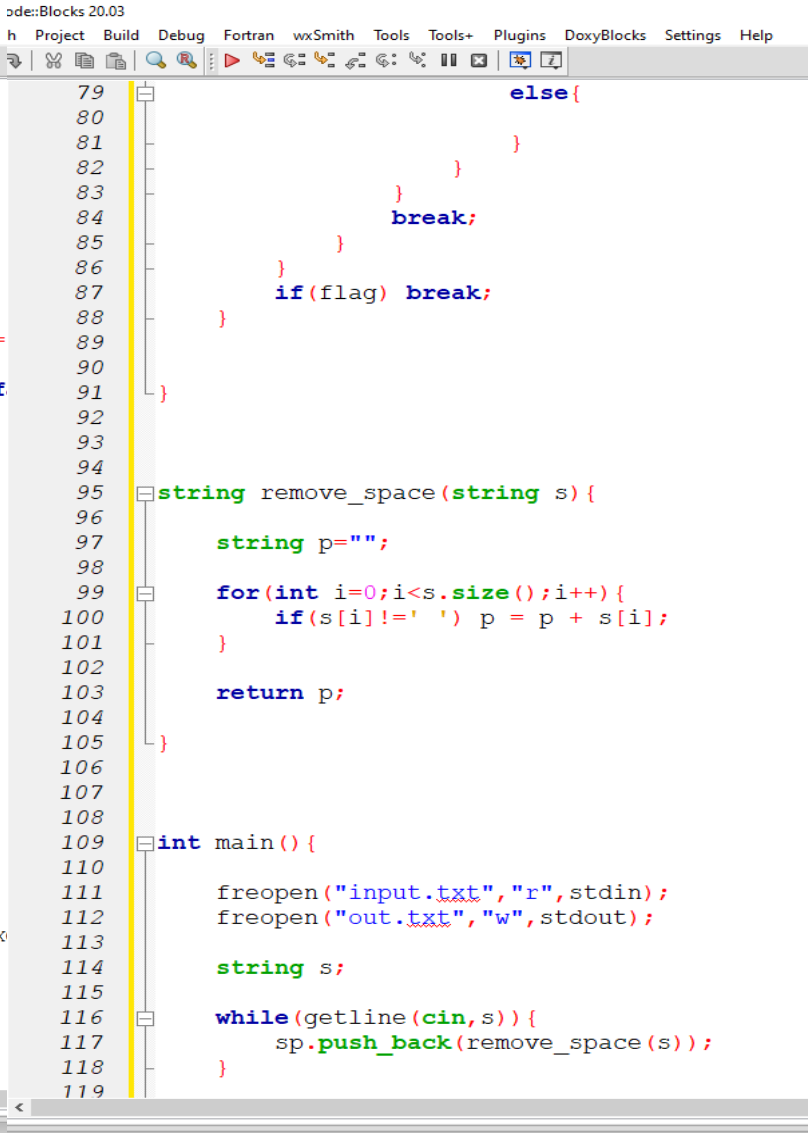


Fig. 19. Assignment 6 Code:Predictive Parsing



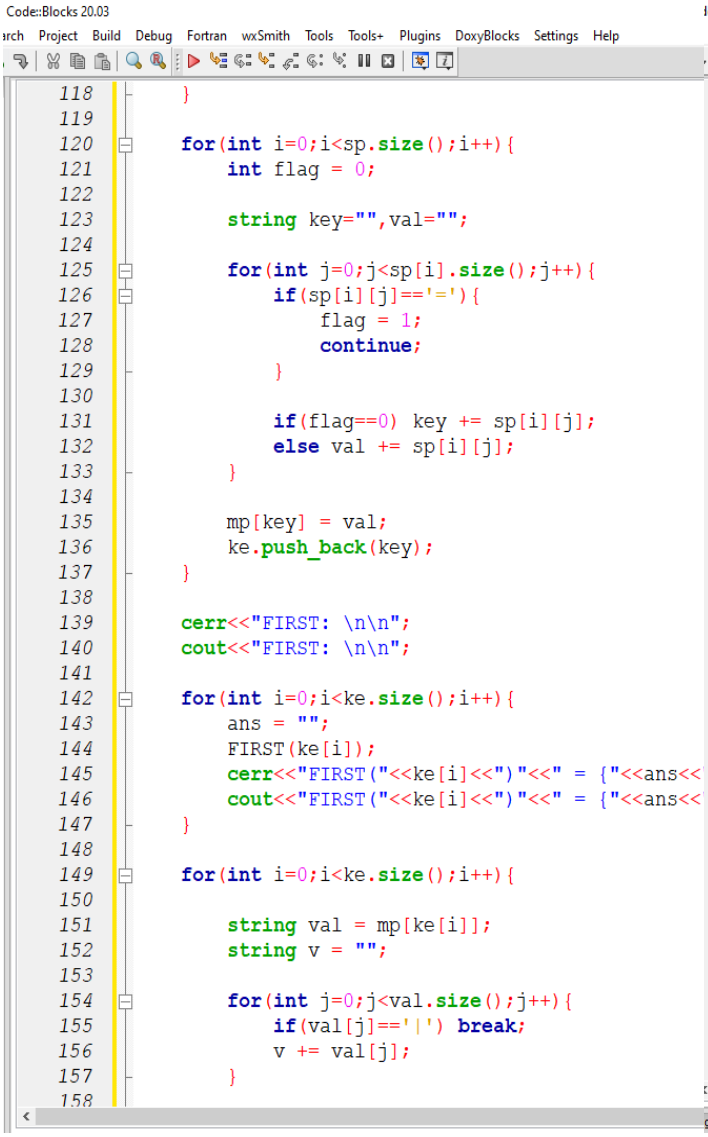
```
40
41  void FOLLOW(string key,int z){
42
43      int flag = 0;
44
45      for(int i=0;i<ri.size();i++){
46          if (ri[i].find(key) != string::npos) {
47              if(key.size()==1){
48                  for(int j=0;j<ri[i].size();j++){
49                      if(ri[i][j]==key[0]){
50                          if(j+1<ri.size() && ri[i][j+1]!=
51                              flag = 1;
52                              if(isTERMINAL(ri[i][j+1])==f
53                                  if(z==0)ans += "$,";
54                                  ans += ri[i][j+1];
55                              }
56                              else{
57                                  string g = ri[i];
58                                  g.erase(0,1);
59                                  FIRST(g);
60                                  if(z==0)ans += "$,";
61                                  FOLLOW(mpp[ri[i]],1);
62
63                              }
64
65                              break;
66                          }
67                      }
68                  }
69              }
70              else{
71                  flag = 1;
72
73                  for(int j=0;j+1<ri[i].size();j++){
74                      if(ri[i][j]==key[0] && ri[i][j+1]==k
75                          if(j+2>=ri[i].size()){
76                              FOLLOW(mpp[ri[i]],1);
77                              if(z==0)ans += ",$";
78                          }
79                          else{
80
```

Fig. 20. Assignment 6 Code:Predictive Parsing



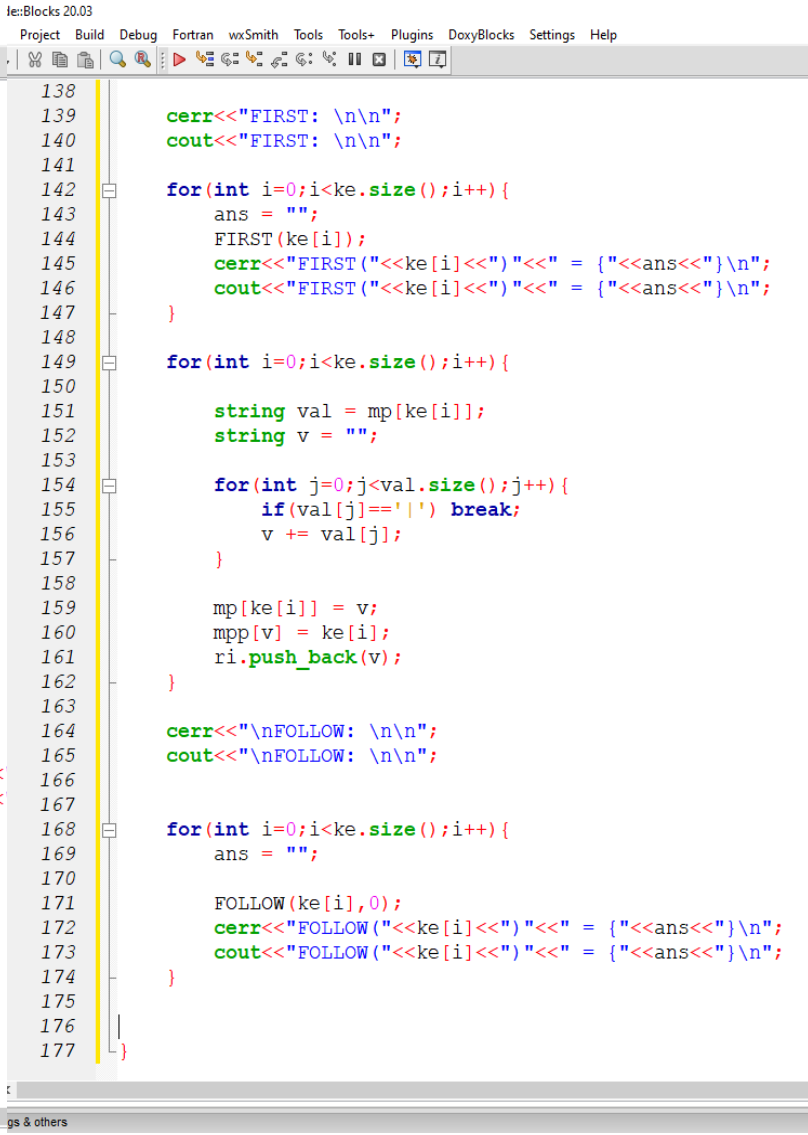
```
79                              else{
80
81                              }
82                          }
83                      }
84                      break;
85                  }
86              }
87              if(flag) break;
88          }
89
90
91  }
92
93
94
95  string remove_space(string s){
96
97      string p="";
98
99      for(int i=0;i<s.size();i++){
100         if(s[i]!=' ') p = p + s[i];
101     }
102
103     return p;
104
105 }
106
107
108
109 int main(){
110
111     freopen("input.txt","r",stdin);
112     freopen("out.txt","w",stdout);
113
114     string s;
115
116     while(getline(cin,s)){
117         sp.push_back(remove_space(s));
118     }
119
```

Fig. 21. Assignment 6 Code:Predictive Parsing




```
118     }
119
120     for(int i=0;i<sp.size();i++){
121         int flag = 0;
122
123         string key="",val="";
124
125         for(int j=0;j<sp[i].size();j++){
126             if(sp[i][j]=='='){
127                 flag = 1;
128                 continue;
129             }
130
131             if(flag==0) key += sp[i][j];
132             else val += sp[i][j];
133         }
134
135         mp[key] = val;
136         ke.push_back(key);
137     }
138
139     cerr<<"FIRST: \n\n";
140     cout<<"FIRST: \n\n";
141
142     for(int i=0;i<ke.size();i++){
143         ans = "";
144         FIRST(ke[i]);
145         cerr<<"FIRST("<<ke[i]<<")"<<" = {"<<ans<<
146         cout<<"FIRST("<<ke[i]<<")"<<" = {"<<ans<<
147     }
148
149     for(int i=0;i<ke.size();i++){
150
151         string val = mp[ke[i]];
152         string v = "";
153
154         for(int j=0;j<val.size();j++){
155             if(val[j]=='|') break;
156             v += val[j];
157         }
158
```

Fig. 22. Assignment 6 Code:Predictive Parsing




```
138
139     cerr<<"FIRST: \n\n";
140     cout<<"FIRST: \n\n";
141
142     for(int i=0;i<ke.size();i++){
143         ans = "";
144         FIRST(ke[i]);
145         cerr<<"FIRST("<<ke[i]<<")"<<" = {"<<ans<<"}\n";
146         cout<<"FIRST("<<ke[i]<<")"<<" = {"<<ans<<"}\n";
147     }
148
149     for(int i=0;i<ke.size();i++){
150
151         string val = mp[ke[i]];
152         string v = "";
153
154         for(int j=0;j<val.size();j++){
155             if(val[j]=='|') break;
156             v += val[j];
157         }
158
159         mp[ke[i]] = v;
160         mpp[v] = ke[i];
161         ri.push_back(v);
162     }
163
164     cerr<<"\nFOLLOW: \n\n";
165     cout<<"\nFOLLOW: \n\n";
166
167
168     for(int i=0;i<ke.size();i++){
169         ans = "";
170
171         FOLLOW(ke[i],0);
172         cerr<<"FOLLOW("<<ke[i]<<")"<<" = {"<<ans<<"}\n";
173         cout<<"FOLLOW("<<ke[i]<<")"<<" = {"<<ans<<"}\n";
174     }
175
176
177 }
```

Fig. 23. Assignment 6 Code:Predictive Parsing

```
E = TE'
E' = +TE' | #
T = FT'
T' = *FT' | #
F = (E) | id
```

Fig. 24. Input Assignment 6 :Predictive Parsing

```
FIRST:

FIRST(E) = {(,id}
FIRST(E') = {+,#}
FIRST(T) = {(,id}
FIRST(T') = {*,#}
FIRST(F) = {(,id}

FOLLOW:

FOLLOW(E) = {$,)}
FOLLOW(E') = {),$}
FOLLOW(T) = {+,$,)}
FOLLOW(T') = {+,),$}
FOLLOW(F) = {*,$,+,)}
```

Fig. 25. OUTPUT Assignment 6 : Predictive Parsing