

Final Assignment : Compiler Lab

CSE-0302 Summer 2021

Ridwana Majumder, UG02-50-19-016

Department of Computer Science and Engineering

State University of Bangladesh (SUB)

Dhaka, Bangladesh

ridr471@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 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 $N \cap T = \text{NULL}$.

P is a set of rules, $P: N \rightarrow (N \cup T)^*$, 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) , $P: S \rightarrow aSa, S \rightarrow bSb, S \rightarrow \epsilon$
The grammar $(S, F, 0, 1, P, S)$, $P: S \rightarrow 00S \mid 11F, F \rightarrow 00F \mid \epsilon$

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. Magnetism Japan, p. 301, 1982].
- [7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

```

20.03
t Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 string int_to_string(int a){
5     stringstream ss;
6     ss << a;
7     string str = ss.str();
8     return str;
9 }
10
11 vector<string> number_lines(vector<string>sp){
12     int flag = 0;
13     string s;
14
15     int flag3 = -1;
16     for(int i=0;i<sp.size();i++){
17         s = "";
18         int sz = sp[i].size();
19         flag3 = -1;
20         for(int j=0;j<sz;j++) if(sp[i][j]=='\t') sp[i][j] = ' ';
21         for(int j=0;j<sz;j++){
22             if(j!=sz-1 && sp[i][j]!=' ' && sp[i][j+1]==' ') s = s + sp[i][j];
23             else if(sp[i][j]!=' ') s += sp[i][j];
24         }
25         for(int j=0;j<sz;j++){
26             if(sp[i][j]==' '){
27                 flag3 = j;
28                 break;
29             }
30         }
31         if(flag3!=-1){
32             string p = "";
33             for(int j=0;j<sz;j++) p += s[j];
34             p += "\n";
35             for(int j=flag3+1,r=0;j<sp[i].size();j++) p += sp[i][j];
36             for(int j=0,r=0;j<s.size();j++){
37                 if(s[j]==' ') r++;
38                 if(r==2) p += s[j];
39             }
40             swap(s,p);
41         }

```

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

```
Code::Blocks 20.03
rch Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
40 swap(s,p);
41 }
42 swap(sp[i],s);
43 }
44
45 vector<string> spl;
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         spl.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         spl.push_back(str);
80         continue;
```

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

```
Blocks 20.03
Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
79 spl.push_back(str);
80 continue;
81 }
82 if(flag2){
83     spl.push_back(str);
84     continue;
85 }
86 str = str + " " + sp[i];
87 spl.push_back(str);
88 }
89
90 return spl;
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();
```

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

```

cks 20.03
ject Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
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+2]=='t') ok = true;
126         if(ok){
127             ok = false;
128             continue;
129         }
130         else err.push_back("Error: Not Matched else at line " + to_string(i+1));
131     }
132 }
133 }
134 }
135 return err;
136 }
137 }
138 bool comp(char a){
139     if(a=='=' || a=='>' || a=='<' ) return false;
140     return true;
141 }
142 }
143 }
144 bool col(char a){
145     if(a==' ' || a=='\n' || a=='+' || a=='-' || a=='*' || a=='/' || a=='%')
146     return false;
147 }
148 }
149 }
150 }
151 vector<string> dup_token_error(vector<string> sp){
152     vector<string>err;
153     int sz = sp.size();
154     for(int j=0;j<sz;j++){
155         string n = " ";
156         s=sp[j];

```

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

```

cks 20.03
ject Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
157 string p = " ";
158 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 = s+" "+p[i];
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 = p+" "+s[i];
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 s+= p[p.size()-1];
192 }
193 istream ss(s);
194 }
195 string last = " ";
196 }
197 }

```

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

```
196         string last = "";
197
198         while(ss>>s){
199             if(s==last) err.push_back("Error: Duplicat
200                 last = s;
201             }
202         }
203     }
204
205     return err;
206 }
207
208
209
210 int main(){
211     freopen("input.txt", "r", stdin);
212     freopen("out.txt", "w", stdout);
213
214     string s;
215
216     vector<string> sp, paran_error, if_else_err, dup_token
217
218     cerr<<"input\n";
219
220     while(getline(cin, s)){
221         sp.push_back(s);
222         cerr<<s<<"\n";
223     }
224
225     cerr<<"\n";
226
227     sp = number_lines(sp);
228
229     cerr<<"\noutput:\n";
230
231     cerr<<"Recognized tokens in the lines of code:\n";
232
233     for(int i=0; i<sp.size(); i++){
234         cout<<sp[i]<<"\n";
235         cerr<<sp[i]<<"\n";
236     }
```

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

```
233
234     for(int i=0; i<sp.size(); i++){
235         cout<<sp[i]<<"\n";
236         cerr<<sp[i]<<"\n";
237     }
238
239     paran_error = parenthesis_error(sp);
240
241     if_else_err = if_else_error(sp);
242
243     dup_token_err = dup_token_error(sp);
244
245     paran_error.erase( unique( paran_error.begin(), paran_error.end()
246
247     if_else_err.erase( unique( if_else_err.begin(), if_else_err.end()
248
249     dup_token_err.erase( unique( dup_token_err.begin(), dup_token_err
250
251     cout<<"\n\nERROR: \n";
252     cerr<<"\n\nERROR: \n";
253
254     for(int i=0; i<paran_error.size(); i++){
255         cout<<paran_error[i]<<"\n";
256         cerr<<paran_error[i]<<"\n";
257     }
258
259     for(int i=0; i<if_else_err.size(); i++){
260         cout<<if_else_err[i]<<"\n";
261         cerr<<if_else_err[i]<<"\n";
262     }
263
264     for(int i=0; i<dup_token_err.size(); i++){
265         cout<<dup_token_err[i]<<"\n";
266         cerr<<dup_token_err[i]<<"\n";
267     }
268
269     return 0;
270 }
271
272
```

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

```

/tmp/o20xtVztTm.o
input

output:
Recognized tokens in the

ERROR:
/* A program fragment*/

float x1 = 3.125;;
/* Definition of function
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
|

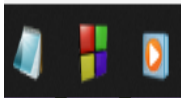


Fig. 10. output Assignment 4 : Detecting Simple Syntax Errors

Code::Blocks 20.03

File Edit View Compiler Project Build Debug Fortran wxSmith Tools Window Help

```
1  #include<bits/stdc++.h>
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

Code::Blocks 20.03

Project Build Debug Fortran wxSmith Tools Tools+ Plugins Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlock

1

#include<bits/stdc++.h>

2

using namespace std;

3

4

int i=0,f=0,l;

5

6

string s;

7

8

void X() {

9

10

if(s[i]=='b') {

11

i++;

12

f = 1;

13

14

else{

15

f = 0;

16

return;

17

18

if(s[i]=='b') {

19

20

i++;

21

f = 1;

22

if(i!=l-1) X();

23

24

else if(s[i]=='c') {

25

26

i++;

27

f = 1;

28

if(i!=l-1) X();

29

30

else{

31

f = 0;

32

return;

33

34

}

35

36

void A() {

37

38

if(s[i]=='a') {

39

40

i++;

41

f = 1;

42

}

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

if(s[i]=='d') {

f = 1;

i++;

return;

}

else{

f = 0;

return;

}

}

int main(){

freopen("i2.txt","r",stdin);

freopen("o2.txt","w",stdout);

while(getline(cin,s)) {

f = 0;

i = 0;

l = s.size();

A();

if(l==i && f) {

cout<<"valid\n";

}

else{

cout<<"invalid\n";

}

}

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

```
Code::Blocks 20.03
rch Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Hel

1
2 #include<bits/stdc++.h>
3 using namespace std;
4
5 vector<string>sp,ke,ri;
6 map<string,string>mp,mpp;
7 string ans;
8
9 bool isTERMINAL(char a){
10     if(a>='A' && a<='Z') return true;
11     return false;
12 }
13
14 void FIRST(string key){
15
16     string val = mp[key];
17
18     if(isTERMINAL(val[0])){
19         string p = "";
20         p += val[0];
21         FIRST(p);
22     }
23     else{
24         ans += val[0];
25         ans += ",";
26         int flag = 0;
27         for(int i=0;i<val.size();i++){
28             if(val[i]=='|'){
29                 flag = 1;
30                 continue;
31             }
32             if(flag){
33                 ans += val[i];
34             }
35         }
36     }
37 }
38
39 }
40
41 void FOLLOW(string key,int z){
```

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<<"}\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     for(int i=0;i<ke.size();i++){
168         ans = "";
169
170         FOLLOW(ke[i],0);
171         cerr<<"FOLLOW("<<ke[i]<<")"<<" = {"<<ans<<"}\n";
172         cout<<"FOLLOW("<<ke[i]<<")"<<" = {"<<ans<<"}\n";
173     }
174
175     }
176
177

```

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     for(int i=0;i<ke.size();i++){
168         ans = "";
169
170         FOLLOW(ke[i],0);
171         cerr<<"FOLLOW("<<ke[i]<<")"<<" = {"<<ans<<"}\n";
172         cout<<"FOLLOW("<<ke[i]<<")"<<" = {"<<ans<<"}\n";
173     }
174
175     }
176
177

```

Fig. 23. Assignment 6 Code:Predictive Parsing

$$\begin{aligned}
 E &= TE' \\
 E' &= +TE' \mid \# \\
 T &= FT' \\
 T' &= *FT' \mid \# \\
 F &= (E) \mid id
 \end{aligned}$$

Fig. 24. Input Assignment 6 :Predictive Parsing

FIRST:

$$\begin{aligned}
 \text{FIRST}(E) &= \{ (, id \} \\
 \text{FIRST}(E') &= \{ +, \# \} \\
 \text{FIRST}(T) &= \{ (, id \} \\
 \text{FIRST}(T') &= \{ *, \# \} \\
 \text{FIRST}(F) &= \{ (, id \}
 \end{aligned}$$

FOLLOW:

$$\begin{aligned}
 \text{FOLLOW}(E) &= \{ \$,) \} \\
 \text{FOLLOW}(E') &= \{), \$ \} \\
 \text{FOLLOW}(T) &= \{ +, \$,) \} \\
 \text{FOLLOW}(T') &= \{ +,), \$ \} \\
 \text{FOLLOW}(F) &= \{ *, \$, +,) \}
 \end{aligned}$$

Fig. 25. OUTPUT Assignment 6 : Predictive Parsing