Assignment No: 02

CSE-0302 Summer 2021

Afrin Sultana
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
afrinsultana.su@gmail.com

***Abstract*—This work denotes we perform detection of simple syntax errors like duplication of tokens except parentheses or braces, unbalanced braces or parentheses problem, unmatched 'else'the problem, etc.and another one works Observe the C code segments that implement the non-terminals of CFG and last one works as Computation of the FIRST and FOLLOW functions.**

n

***Index Terms*—c,c++,Syntax Errors,CFG,Parsing.**

I. Introduction

In this program the main purpose of this session is to detect and report simple syntax errors.Syntax errors are very common in source programs like except parentheses or braces, unbalanced braces or parentheses problem, unmatched 'else' problem, etc. We can think of using CFGs to parse various language constructs in the token streams freed from simple syntactic and semantic errors, as it is easier to describe the constructs with CFGs. note that a recursive descent parser can be constructed from a CFG with reduced left recursion and ambiguity. Manual implementation of LL(1) parsing algorithms.our main purpose is find the FIRST and FOLLOW sets of LL(1) of the non-terminals.

II. Literature Review

Stormy Attaway, in MATLAB (Fifth Edition), 2019,MATLAB itself will flag syntax errors and give an error message.Program Development and Testing Andrew P. King, Paul Aljabar, in MATLAB Programming for Biomedical Engineers and Scientists, 2017.Language Reference Guide John Iovine, in PIC Projects for Non-Programmers, 2012 Prolog Programming Language Heimo H. Adelsberger, in Encyclopedia of Physical Science and Technology (Third Edition), 2003. Computational Analysis and Understanding of Natural Languages: Principles, Methods and Applications Akhil Gudivada, Dhana L. Rao, in Handbook of Statistics, 2018.for parsing John Backus, the principle designer of FORTRAN, and Peter Naur, a journalist for a computer magazine, both attend a conference on Algol in 1960 in Paris, France.

III. Proposed Methodology

IV. Conclusion and Future Work

In future, we try to make program for this project without calling file only use compiler.Infuture add some feature that simplify this project.we use frame work and use this for making another big projects.

Acknowledgment

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

References

[1] Santos, E. A., Campbell, J. C., Patel, D., Hindle, A., & Amaral, J. N. (2018, March). Syntax and sensibility: Using language models to detect and correct syntax errors. In 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 311-322). IEEE.

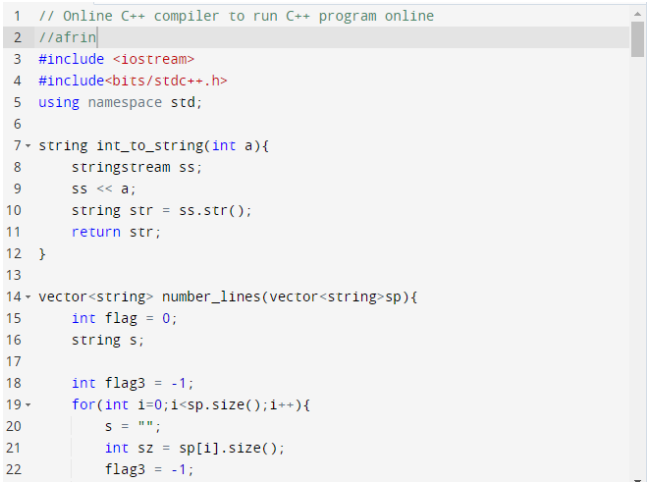
[2] Denny, P., Luxton-Reilly, A., & Tempero, E. (2012, July). All syntax errors are not equal. In Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education (pp. 75-80).

[3] Ford, B. (2004, January). Parsing expression grammars: a recognition-based syntactic foundation. In Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages (pp. 111-122).

[4] Demberg, V., Keller, F., & Koller, A. (2013). Incremental, predictive parsing with psycholinguistically motivated tree-adjoining grammar. Computational Linguistics, 39(4), 1025-1066.

[5] Klein, D., & Manning, C. D. (2003, July). Accurate unlexicalized parsing. In Proceedings of the 41st annual meeting of the association for computational linguistics (pp. 423-430).

[6] CFG, T., Parkinson, L., Griffiths, G. J., Garcia, A. F., & Marshall, E. J. (2001). Aggregation and temporal stability of carabid beetle distributions in field and hedgerow habitats. Journal of Applied Ecology, 38(1), 100-116.



```
// Online C++ compiler to run C++ program online
//afrin
#include <iostream>
#include<bits/stdc++.h>
using namespace std;

string int_to_string(int a){
    stringstream ss;
    ss << a;
    string str = ss.str();
    return str;
}

vector<string> number_lines(vector<string>sp){
    int flag = 0;
    string s;

    int flag3 = -1;
    for(int i=0;i<sp.size();i++){
        s = "";
        int sz = sp[i].size();
        flag3 = -1;
```

Fig. 1. Session 4: Detecting Simple Syntax Errors.

```
        s = "";
        int sz = sp[i].size();
        flag3 = -1;
        for(int j=0;j<sz;j++) if(sp[i][j]=='\t') sp[i][j] = ' ';
        for(int j=0;j<sz;j++){
            if(j!=sz-1 && sp[i][j]!=' '  && sp[i][j+1]==' ') s = s
                + sp[i][j] + ' ';
            else if(sp[i][j]!=' ') s += sp[i][j];
        }
        for(int j=0;j<sz;j++){
            if(sp[i][j]=='"'){
                flag3 = j;
                break;
            }
        }
        if(flag3!=-1){
            string p = "";
            for(int j=0;s[j]!='"';j++) p += s[j];
            p += "\"";
            for(int j=flag3+1,r=0;sp[i][j]!='"';j++) p += sp[i][j];
            for(int j=0,r=0;j<s.size();j++){
                if(s[j]=='"') r++;
```

Fig. 2. Session 4: Detecting Simple Syntax Errors.

```
            for(int j=flag3+1,r=0;sp[i][j]!='"';j++) p += sp[i][j];
            for(int j=0,r=0;j<s.size();j++){
                if(s[j]=='"') r++;
                if(r==2) p +=s[j];
            }
            swap(s,p);
        }
        swap(sp[i],s);
    }

    vector<string>sp1;

    int flag1 = 0,flag2=0;
    for(int i=0;i<sp.size();i++){
        string str = int_to_string(i+1);
        int sz = sp[i].size();
        if(sz==0){
            sp1.push_back(str);
            continue;
        }
        for(int j=0;j<sz;j++){
            if(j!=sz-1 && sp[i][j]=='/' && sp[i][j+1]=='/'){
```

Fig. 3. Session 4: Detecting Simple Syntax Errors.

```
        }
        for(int j=0;j<sz;j++){
            if(j!=sz-1 && sp[i][j]=='/' && sp[i][j+1]=='/'){
                flag1 = 1;
                for(int k=0;k<j;k++){
                    cout<<sp[i][k];
                    cerr<<sp[i][k];
                }
                break;
            }
            if(j!=sz-1 && sp[i][j]=='/' && sp[i][j+1]=='*'){
                flag2 = 1;
                for(int k=0;k<j;k++){
                    cout<<sp[i][k];
                    cerr<<sp[i][k];
                }
            }
            if(j!=sz-1 && sp[i][j]=='*' && sp[i][j+1]=='/'){
                flag2 = 0;
                flag1 = 1;
                break;
            }
        }
```

Fig. 4. Session 4: Detecting Simple Syntax Errors.

```
                break;
            }
        }
        if(flag1){
            flag1 = 0;
            sp1.push_back(str);
            continue;
        }
        if(flag2){
            sp1.push_back(str);
            continue;
        }
        str = str + " " + sp[i];
        sp1.push_back(str);
    }

    return sp1;

}

vector<string> paranthesis_error(vector<string> sp){
```

Fig. 5. Session 4: Detecting Simple Syntax Errors.

```
}

vector<string> paranthesis_error(vector<string> sp){

    stack<int>st;
    vector<string>err;

    for(int i=0;i<sp.size();i++){
        for(int j=0;j<sp[i].size();j++){
            if(sp[i][j]=='{') st.push(i+1);
            else if(sp[i][j]=='}'){
                if( !st.empty() ) st.pop();
                else err.push_back("Error: Misplaced '}' at line "
                    +int_to_string(i+1));
            }
        }
    }

    if( !st.empty() ) err.push_back("Error: Not Balanced
        Parentheses at line "+int_to_string(sp.size()));

    return err;
}
```

Fig. 6. Session 4: Detecting Simple Syntax Errors.

```

    return err;
}


vector<string> if_else_error(vector<string> sp){

    bool ok = false;
    vector<string>err;
    int sz = sp.size();
    for(int i=0;i<sz;i++){
        if(sz<4)continue;
        int x = sp[i].size();
        for(int j=0;j<x;j++){
            if(j+1<x && sp[i][j]=='i' && sp[i][j+1]=='f') ok = true
                ;
            if(j+3<x && sp[i][j]=='e' && sp[i][j+1]=='l' && sp[i][j
                +2]=='s' && sp[i][j+3]=='e'){
                if( ok ){
                    ok = false;
                    continue;
                }
```

Fig. 7. Session 4: Detecting Simple Syntax Errors.

```
                    ok = false;
                    continue;
                }
                else err.push_back("Error: Not Matched else at line
                    "+int_to_string(i+1));
            }
        }
    }

    return err;
}

bool comp(char a){
    if(a=='=' || a=='>' || a=='<' ) return false;

    return true;
}

bool col(char a){

    if(a==',' || a==';' || a=='+' || a=='-' || a=='*' || a=='/' ||
        a=='(' || a==')' || a=='\'') return true;
```

Fig. 8. Session 4: Detecting Simple Syntax Errors.

```
}

bool col(char a){

    if(a==',' || a==';' || a=='+' || a=='-' || a=='*' || a=='/' ||
        a=='(' || a==')' || a=='\'') return true;
    return false;

}

vector<string> dup_token_error(vector<string> sp){

    vector<string>err;
    int sz = sp.size();

    for(int j=0;j<sz;j++){

        string p = "",s=sp[j];

        for(int i=0;i<s.size();i++){
            if(col(s[i]) && col(s[i+1])==false) p = p+" "+s[i]+" ";
            else if(col(s[i]) && col(s[i+1])) p = p+" "+s[i];
```

Fig. 9. Session 4: Detecting Simple Syntax Errors.

```
        for(int i=0;i<s.size();i++){
            if(col(s[i]) && col(s[i+1])==false) p = p+" "+s[i]+" ";
            else if(col(s[i]) && col(s[i+1])) p = p+" "+s[i];
            else p += s[i];
        }

        s = p[0];

        for(int i=1;i<p.size()-1;i++){
            if(p[i]=='=' && comp(p[i-1]) && comp(p[i+1])) s =  s+"
                "+p[i]+" ";
            else s +=p[i];
        }

        p = "";


        for(int i=0;i<s.size();i++){
            if(i!=s.size()-1 && s[i]!=' '  && s[i+1]==' ') p = p +
                s[i] + ' ';
            else if(s[i]!=' ') p += s[i];
        }
```

Fig. 10. Session 4: Detecting Simple Syntax Errors.

```
            else if(s[i]!=' ') p += s[i];
        }

        s = p[0];

        for(int i=1;i<p.size()-1;i++){
            if(comp(p[i])==false && comp(p[i+1])==false){
                s = s + " "+ p[i]+p[i+1] + " ";
                i++;
            }
            else s += p[i];
        }


        s+= p[p.size()-1];

        istringstream ss(s);

        string last = "";

        while(ss>>s){
```

Fig. 11. Session 4: Detecting Simple Syntax Errors.

```
        string last = "";

        while(ss>>s){
            if(s==last) err.push_back("Error: Duplicate token at
                line "+int_to_string(j+1));
            last = s;
        }

    }

    return err;
}


int main(){

    freopen("input.txt","r",stdin);
    freopen("out.txt","w",stdout);

    string s;

    vector<string>sp,paran_error,if_else_err,dup_token_err,error;
```

Fig. 12. Session 4: Detecting Simple Syntax Errors.

```
    string s;

    vector<string>sp,paran_error,if_else_err,dup_token_err,error;

    cerr<<"input\n";

    while(getline(cin,s)){
        sp.push_back(s);
        cerr<<s<<"\n";
    }

    cerr<<"\n";

    sp = number_lines(sp);

    cerr<<"\noutput:\n";

    cerr<<"Recognized tokens in the lines of code:\n";

    for(int i=0;i<sp.size();i++){
        cout<<sp[i]<<"\n";
        cerr<<sp[i]<<"\n";
```

Fig. 13. Session 4: Detecting Simple Syntax Errors.

```
239         cerr<<sp[i]<<"\n";
240     }
241
242     paran_error = paranthesis_error(sp);
243
244     if_else_err = if_else_error(sp);
245
246     dup_token_err = dup_token_error(sp);
247
248     paran_error.erase( unique( paran_error.begin(), paran_error.end
            () ), paran_error.end() );
249
250     if_else_err.erase( unique( if_else_err.begin(), if_else_err.end
            () ), if_else_err.end() );
251
252     dup_token_err.erase( unique( dup_token_err.begin(),
            dup_token_err.end() ), dup_token_err.end() );
253
254
255     cout<<"\n\nERROR: \n";
256     cerr<<"\n\nERROR: \n";
257
```

Fig. 14. Session 4: Detecting Simple Syntax Errors.

```
253
254
255     cout<<"\n\nERROR: \n";
256     cerr<<"\n\nERROR: \n";
257
258     for(int i=0;i<paran_error.size();i++){
259         cout<<paran_error[i]<<"\n";
260         cerr<<paran_error[i]<<"\n";
261     }
262
263     for(int i=0;i<if_else_err.size();i++){
264         cout<<if_else_err[i]<<"\n";
265         cerr<<if_else_err[i]<<"\n";
266     }
267
268     for(int i=0;i<dup_token_err.size();i++){
269         cout<<dup_token_err[i]<<"\n";
270         cerr<<dup_token_err[i]<<"\n";
271     }
272
273     return 0;
274 }
```

Fig. 15. Session 4: Detecting Simple Syntax Errors.

Output

```
/tmp/Q2hBuCeRv9.o
input
/* A program fragment*/
float x1 = 3.125;;;
/* Definition of function f1 */
double f1(float a, int int x)
{if(x<x1)
double z;;
else z = 0.01;}}
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 code:
```

Fig. 16. Session 4:Input.

```
output:
Recognized tokens in the lines of code:


ERROR:
tion f1 */

double f1(float a, int int x)

{if(x<x1)

double z;;

else z = 0.01;}}

else return z;

}

/* Beginning of 'main' */
```

Fig. 17. Session 4:Output.

```
/* Beginning of 'main' */

int main(void)

{{{{

int n1; double z;

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



output:

Recognized tokens in the lines of code:




ERROR:
```

Fig. 18. Session 4:Output.

```
ERROR:


dash: 2: tion: not found
dash: 4: Syntax error: "(" unexpected
dash: 5: Syntax error: word unexpected (expecting ")")
dash: 6: Syntax error: newline unexpected
dash: 6: Syntax error: ";;" unexpected
dash: 7: Syntax error: "else" unexpected
dash: 8: Syntax error: "else" unexpected
dash: 9: Syntax error: "}" unexpected
dash: 11: /app: Permission denied
dash: 13: Syntax error: "(" unexpected
dash: 14: {{{{: not found
dash: 16: int: not found
dash: 16: double: not found
dash: 18: Syntax error: "(" unexpected
dash: 21: output:: not found
dash: 23: Recognized: not found
dash: 29: ERROR:: not found
```

Fig. 19. Session 4:Output.

```cpp
// Online C++ compiler to run C++ program online
//afrin
#include<bits/stdc++.h>
using namespace std;

int i=0,f=0,l;

string st;

void A() {
    if (st[i] == 'a') {
        i++;
        f=1;
    }
    else {
        f=0;
        return;
    }
    if (i<l-1) A();
}

void B() {
```

Fig. 20. Session 5.1: Use of CFGs for Parsing.

```cpp

void B() {
    if (st[i] == 'b') {
        i++;
        f=1;
        return;
    }
    else {
        f=0;
        return;
    }
}

void S() {
    if (st[i] == 'b'){
        i++;
        f = 1;
        return;
    }
    else {
        A();
```

Fig. 21. Session 5.1: Use of CFGs for Parsing.

```cpp
        f = 1;
        return;
    }
    else {
        A();
        if (f) { B(); return;}
    }
}

int main(){

    freopen("i1.txt","r",stdin);
    freopen("o1.txt","w",stdout);

    while(getline(cin,st)){

        f = 0;
        i = 0;

        l = st.size();

        S();
```

Fig. 22. Session 5.1: Use of CFGs for Parsing.

```cpp
    freopen("i1.txt","r",stdin);
    freopen("o1.txt","w",stdout);

    while(getline(cin,st)){

        f = 0;
        i = 0;

        l = st.size();

        S();

        if(l==i && f){
            cout<<"valid\n";
        }
        else{
            cout<<"invalid\n";
        }
    }

}
```

Fig. 23. Session 5.1: Use of CFGs for Parsing.

```
1  b
2  ab
3  aab
4  aaab
```

Fig. 24. Session 5.1:Input.

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

Fig. 25. Session 5.1:Output.

```cpp
// Online C++ compiler to run C++ program online
//afrin
#include<bits/stdc++.h>
using namespace std;

int i=0,f=0,l;

string s;

void X(){

    if(s[i]=='b'){
        i++;
        f = 1;
    }
    else{
        f = 0;
        return;
    }

```

Fig. 26. Session 5.2: Use of CFGs for Parsing.

```
        f = 0;
        return;
    }

    if(s[i]=='b'){
        i++;
        f = 1;
        if(i!=l-1) X();
    }
    else if(s[i]=='c'){
        i++;
        f = 1;
        if(i!=l-1) X();
    }
    else{
        f = 0;
        return;
    }

}
```

Fig. 27. Session 5.2: Use of CFGs for Parsing.

```

void A(){

    if(s[i]=='a'){
        i++;
        f = 1;
    }
    else return;

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

    if(i==l-1 && f){
        if(s[i]=='d'){
            f = 1;
            i++;
            return;
        }
```

Fig. 28. Session 5.2: Use of CFGs for Parsing.

```
            i++;
            return;
        }
        else{
            f = 0;
            return;
        }
    }

}

int main(){

    freopen("i2.txt","r",stdin);
    freopen("o2.txt","w",stdout);
    while(getline(cin,s)){


        f = 0;
```

Fig. 29. Session 5.2: Use of CFGs for Parsing.

```

        f = 0;
        i = 0;

        l = s.size();

        A();

        if(l==i && f){
            cout<<"valid\n";
        }
        else{
            cout<<"invalid\n";
        }

    }

}
```

Fig. 30. Session 5.2: Use of CFGs for Parsing.

```
asasfas
bba
ba
abbd
```

Fig. 31. Session 5.2:Input.

```
invalid
invalid
invalid
valid
```

Fig. 32. Session 5.2:Output.

```
// Online C++ compiler to run C++ program online
//afrin
#include<bits/stdc++.h>
using namespace std;

vector<string>sp,ke,ri;
map<string,string>mp,mpp;
string ans;

bool isTERMINAL(char a){
    if(a>='A' && a<='Z') return true;
    return false;
}

void FIRST(string key){

    string val = mp[key];

    if(isTERMINAL(val[0])){
        string p = "";
        p += val[0];
        FIRST(p);
```

Fig. 33. Session 6: Predictive Parsing.

```
        string p = "";
        p += val[0];
        FIRST(p);
    }
    else{
        ans += val[0];
        ans += ",";
        int flag = 0;
        for(int i=0;i<val.size();i++){
            if(val[i]=='|'){
                flag = 1;
                continue;
            }
            if(flag){
                ans += val[i];
            }
        }

    }

}

```

Fig. 34. Session 6: Predictive Parsing.

```

}

void FOLLOW(string key,int z){

    int flag = 0;

    for(int i=0;i<ri.size();i++){
        if (ri[i].find(key) != string::npos) {
            if(key.size()==1){
                for(int j=0;j<ri[i].size();j++){
                    if(ri[i][j]==key[0]){
                        if(j+1<ri.size() && ri[i][j+1]!='\''){
                            flag = 1;
                            if(isTERMINAL(ri[i][j+1])==false){
                                if(z==0)ans += "$,";
                                ans += ri[i][j+1];
                            }
                            else{
                                string g = ri[i];
                                g.erase(0,1);
                                FIRST(g);
```

Fig. 35. Session 6: Predictive Parsing.

```
                                string g = ri[i];
                                g.erase(0,1);
                                FIRST(g);
                                if(z==0)ans += "$,";
                                FOLLOW(mpp[ri[i]],1);

                            }

                            break;
                        }
                    }
                }
            }
            else{
                flag = 1;

                for(int j=0;j+1<ri[i].size();j++){
                    if(ri[i][j]==key[0] && ri[i][j+1]==key[1]){
                        if(j+2>=ri[i].size()){
                            FOLLOW(mpp[ri[i]],1);
                            if(z==0)ans += ",$";
                        }
                        else{
```

Fig. 36. Session 6: Predictive Parsing.

```
                        else{

                        }
                    }
                }
            }
        }
    }

}


string remove_space(string s){

    string p="";

    for(int i=0;i<s.size();i++){
        if(s[i]!=' ') p = p + s[i];
```

Fig. 37. Session 6: Predictive Parsing.

```
    for(int i=0;i<s.size();i++){
        if(s[i]!=' ') p = p + s[i];
    }

    return p;

}


int main(){

    freopen("input.txt","r",stdin);
    freopen("out.txt","w",stdout);

    string s;

    while(getline(cin,s)){
        sp.push_back(remove_space(s));
    }

```

Fig. 38. Session 6: Predictive Parsing.

```
    }

    for(int i=0;i<sp.size();i++){
        int flag = 0;

        string key="",val="";

        for(int j=0;j<sp[i].size();j++){
            if(sp[i][j]=='='){
                flag = 1;
                continue;
            }

            if(flag==0) key += sp[i][j];
            else val += sp[i][j];
        }

        mp[key] = val;
        ke.push_back(key);
    }

    cerr<<"FIRST: \n\n";
```

Fig. 39. Session 6: Predictive Parsing.

```
    }

    cerr<<"FIRST: \n\n";
    cout<<"FIRST: \n\n";

    for(int i=0;i<ke.size();i++){
        ans = "";
        FIRST(ke[i]);
        cerr<<"FIRST("<<ke[i]<<")"<<" = {"<<ans<<"}\n";
        cout<<"FIRST("<<ke[i]<<")"<<" = {"<<ans<<"}\n";
    }

    for(int i=0;i<ke.size();i++){

        string val = mp[ke[i]];
        string v = "";

        for(int j=0;j<val.size();j++){
            if(val[j]=='|') break;
            v += val[j];
        }
```

Fig. 40. Session 6: Predictive Parsing.

```
            v += val[j];
        }

        mp[ke[i]] = v;
        mpp[v] = ke[i];
        ri.push_back(v);
    }

    cerr<<"\nFOLLOW: \n\n";
    cout<<"\nFOLLOW: \n\n";


    for(int i=0;i<ke.size();i++){
        ans = "";

        FOLLOW(ke[i],0);
        cerr<<"FOLLOW("<<ke[i]<<")"<<" = {"<<ans<<"}\n";
        cout<<"FOLLOW("<<ke[i]<<")"<<" = {"<<ans<<"}\n";
    }


}
```

Fig. 41. Session 6: Predictive Parsing.

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

Fig. 42. Session 6: Input.

```
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. 43. Session 6: Output.