Final Report

CSE-0302 Summer - 2021

Koushik Biswas [UG02-50-19-017]
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
koushik.biswas.edu@gmail.com

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

n

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

I. Introduction

Assignment 4 : Detecting Simple Syntax Errors

Syntax errors are very common in source program.The main purpose of this session is to write programs to detect and report simple syntax errors.

Assignment 5 : Use of CFGs for Parsing

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.But CFGs are hard to apply practically. In this session,we implement a simple recursive descent parser to parse a number of types of statements after exercising with simpler CFGs.We note that a recursive decent parser can be construsted from a CFGs with reduced left recursion and ambiguity.

Assignment 6 : Predictive Parsing

Manual implementation of LL(1) and LR(1) parsing algorithms .

II. Literature Review

Assignment 4 : Detecting Simple Syntax Errors

A frustrating aspect of software development is that compiler error messages often fail to locate the actual cause of a syntax error.Syntax Errors Just Aren't Natural.Jashua Charles (Department of Computing Science),Abram Hindle (department of Computing Science),Jose Nelson Amaral(Department of Computing Science) Improving Error Reporting with Language Models.

Assignment 5 : Use of CFGs for Parsing

Context Free Grammars (CFG) can be classified on the basis of following two properties: 1) Based on number of strings it generates. During Compilation, the parser uses the grammar of the language to make a parse tree(or derivation tree) out of the source code. Vilhjálmur orsteinsson, Hulda Óladóttir,Hrafn Loftsson(Department of Computer Science) .Both present open-source,wide-coverage context-free grammer (CFG) for Icelandic and an accompanying parsing system.

Assignment 6 : Predictive Parsing

A predictive parser is a recursive descent parser with no backtracking or backup. It is a top-down parser that does not require backtracking. At each step, the choice of the rule to be expanded is made upon the next terminal symbol.

III. Proposed Methodology

IV. Conclusion and Future Work

Every Computer Engineer should learn compiler design so that an interpreted scripting language and interpreter.I think thatwhat is useful is how to :Parse an expression tree,Robust error handling,General-purpose text processing technique,Sanitize input,Schedule tasks in the future with cross-platform timers,Creation of virtual machines.

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.

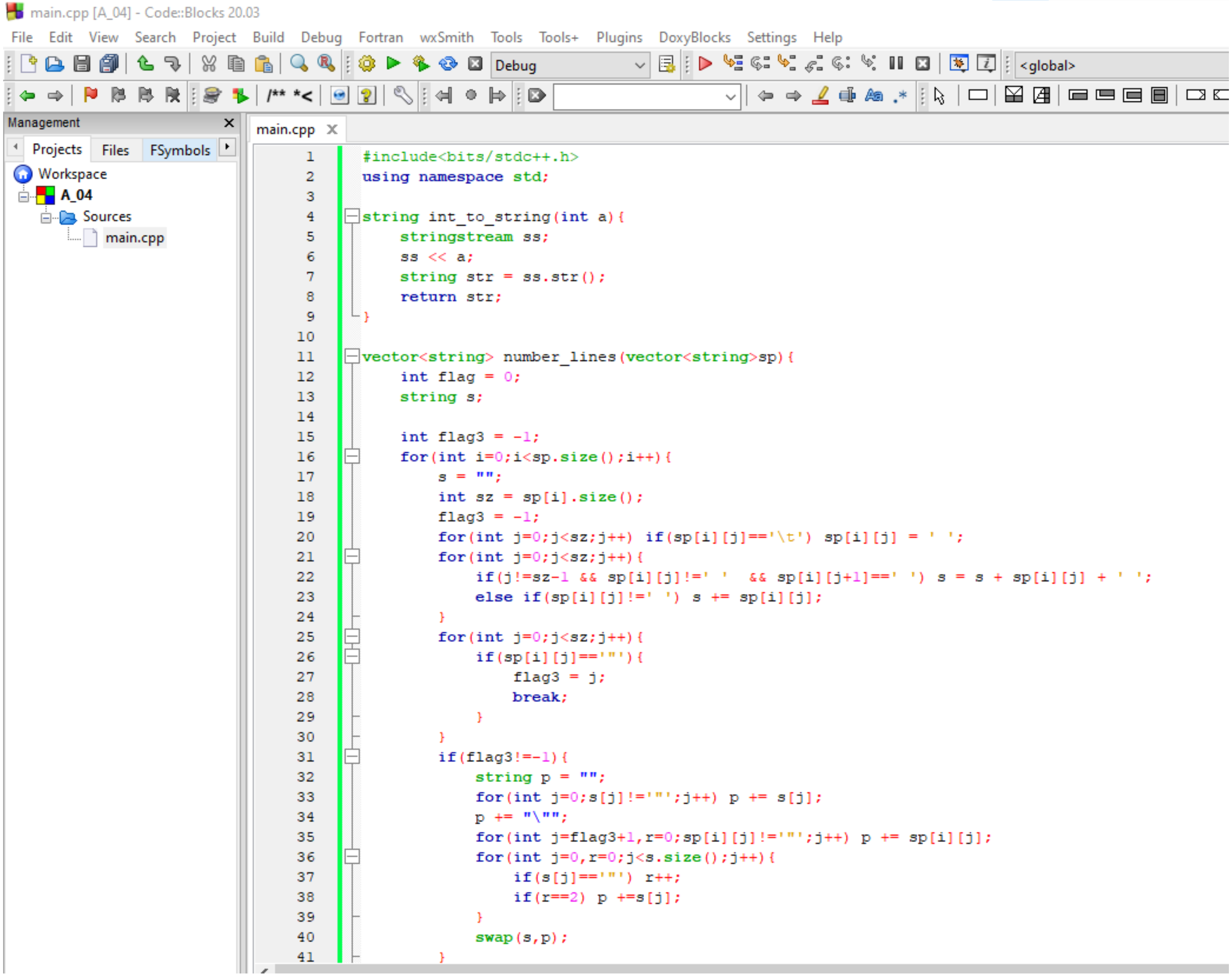
[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[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.

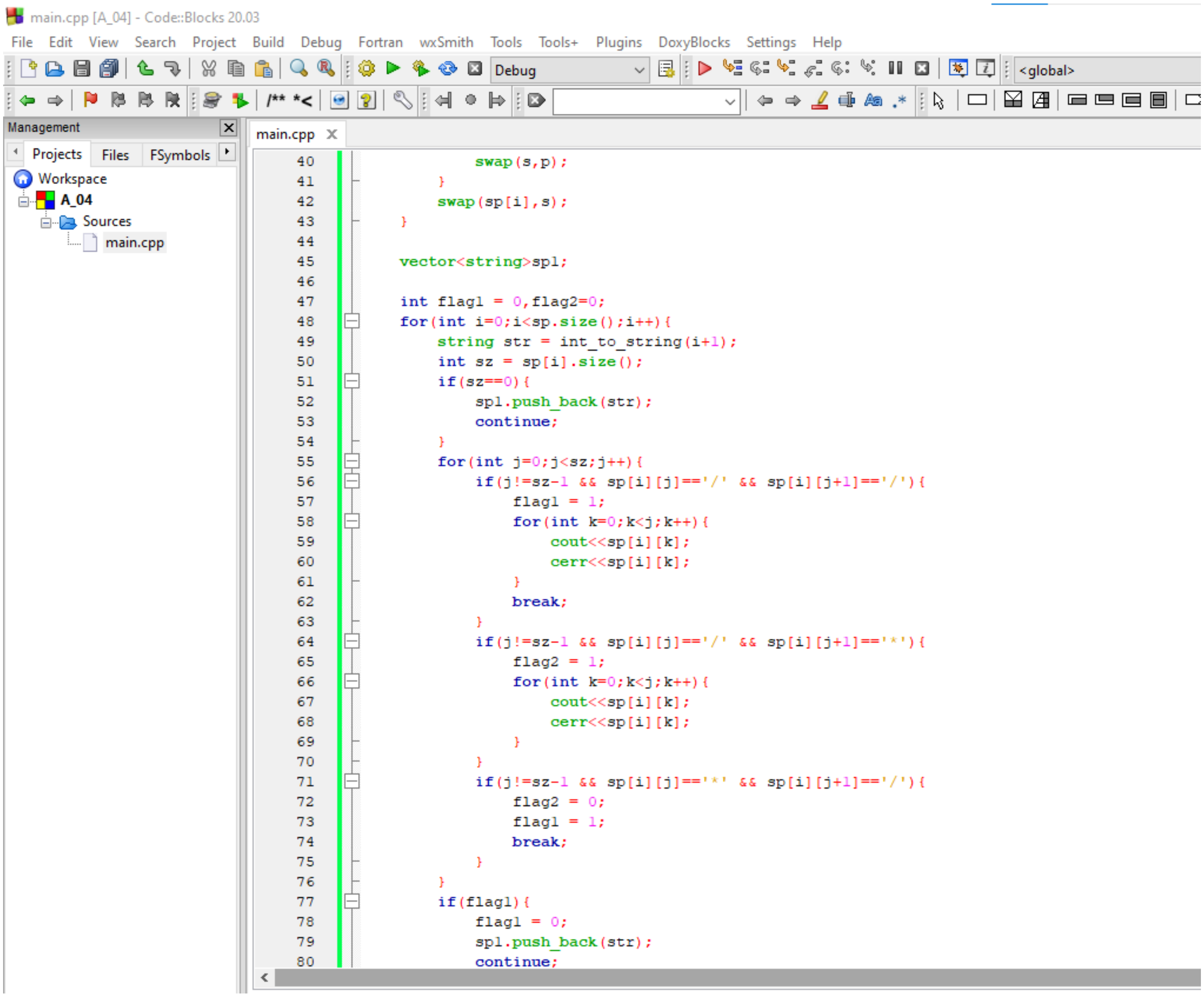
[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

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


main.cpp [A_04] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
<global>
Management
Projects Files FSymbols
Workspace
A_04
Sources
main.cpp
main.cpp

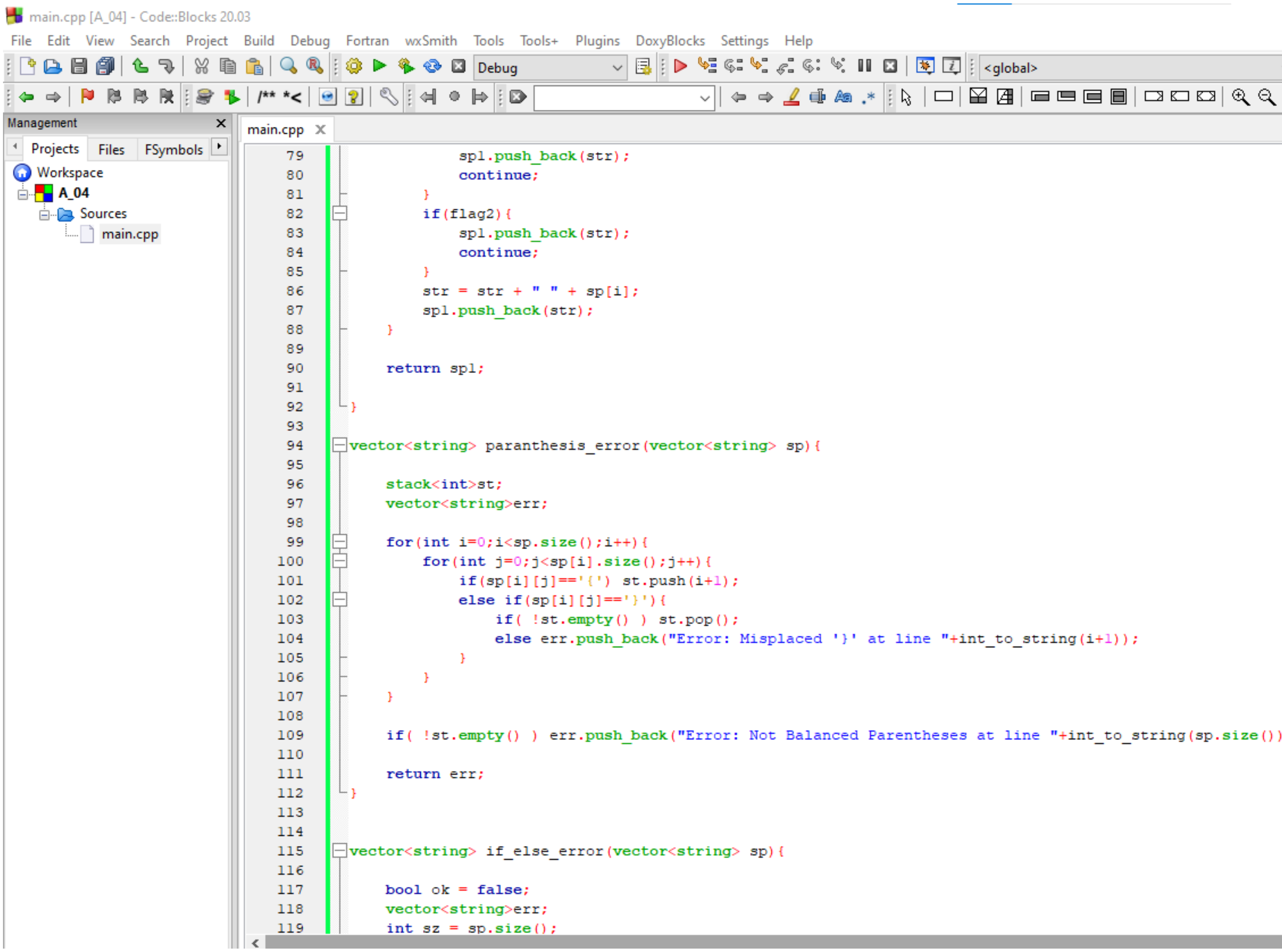
```
#include<bits/stdc++.h>
using namespace std;

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;
        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++;
                if(r==2) p +=s[j];
            }
            swap(s,p);
        }
```

Fig. 1. Constructing LR(0) automation for the grammar


main.cpp [A_04] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
<global>
Management
Projects Files FSymbols
Workspace
A_04
Sources
main.cpp
main.cpp

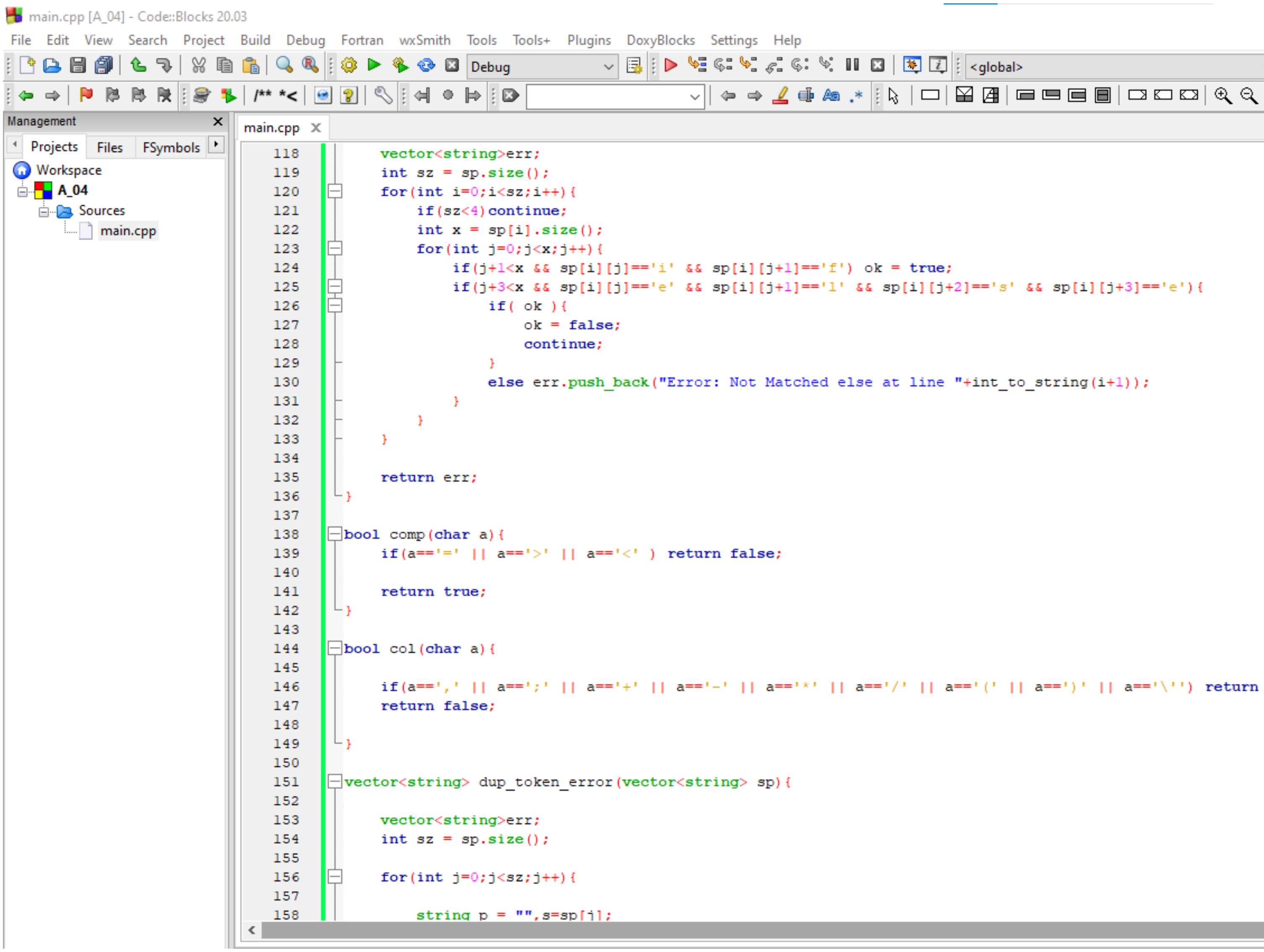
```cpp
            swap(s,p);
        }
        swap(sp[i],s);
    }

    vector<string>spl;

    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){
            spl.push_back(str);
            continue;
        }
        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;
            }
        }
        if(flag1){
            flag1 = 0;
            spl.push_back(str);
            continue;
```

Fig. 2. Constructing LR(0) automation for the grammar



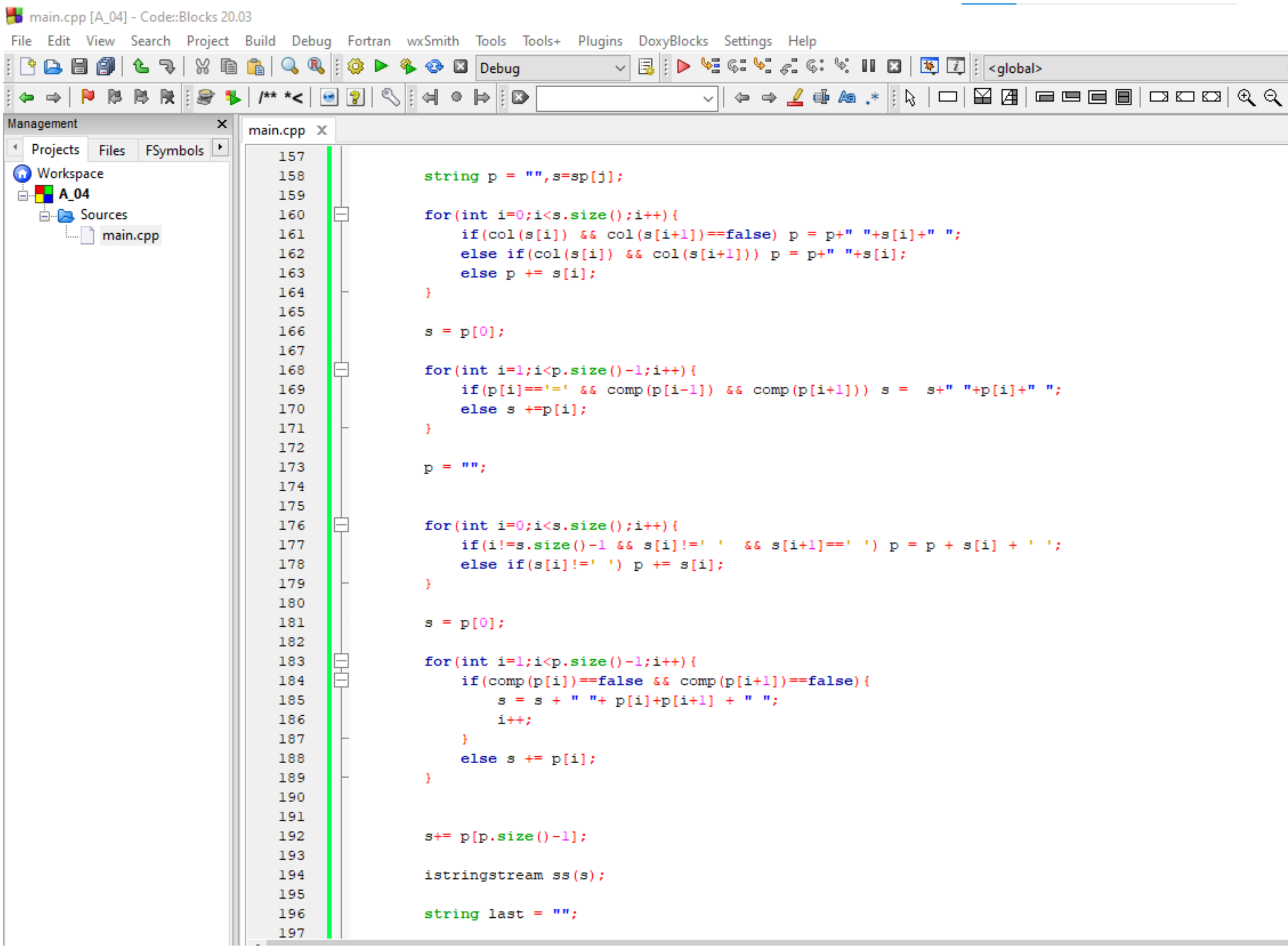
```
            spl.push_back(str);
            continue;
        }
        if(flag2){
            spl.push_back(str);
            continue;
        }
        str = str + " " + sp[i];
        spl.push_back(str);
    }

    return spl;

}

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;
}


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

    bool ok = false;
    vector<string>err;
    int sz = sp.size();
```

Fig. 3. Constructing LR(0) automation for the grammar



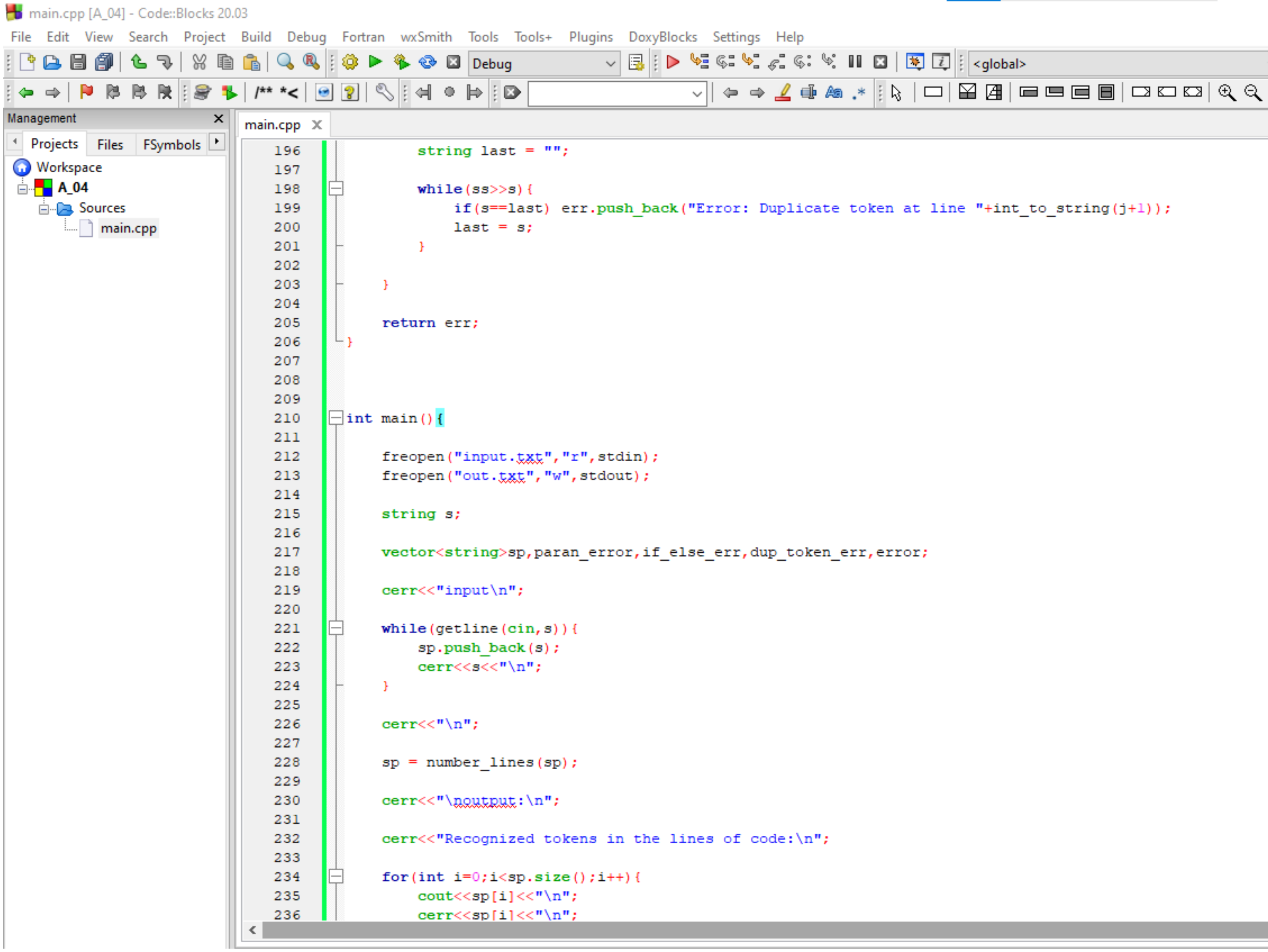
```cpp
    vector<string>err;
    int sz = sp.size();
    for(int i=0;i<sz;i++){
        if(sz<4)continue;
        int x = sp[i].size();
        for(int j=0;j<x;j++){
            if(j+1<x && sp[i][j]=='i' && sp[i][j+1]=='f') ok = true;
            if(j+3<x && sp[i][j]=='e' && sp[i][j+1]=='l' && sp[i][j+2]=='s'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

Fig. 4. Constructing LR(0) automation for the grammar



```cpp

        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];
            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];
        }

        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 = "";

```

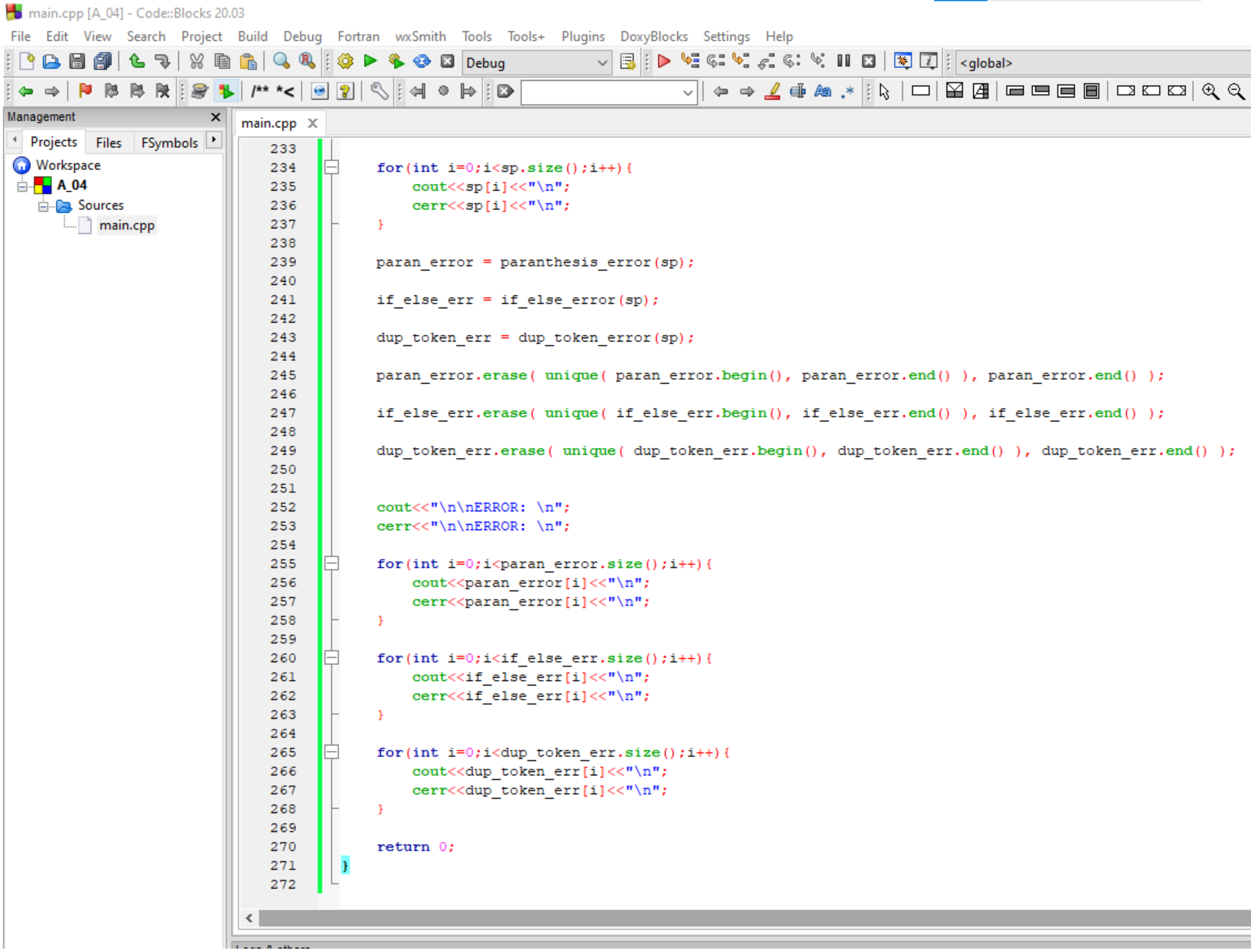
Fig. 5. Constructing LR(0) automation for the grammar



main.cpp [A_04] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
<global>
Management
Projects
Files
FSymbols
Workspace
A_04
Sources
main.cpp
main.cpp

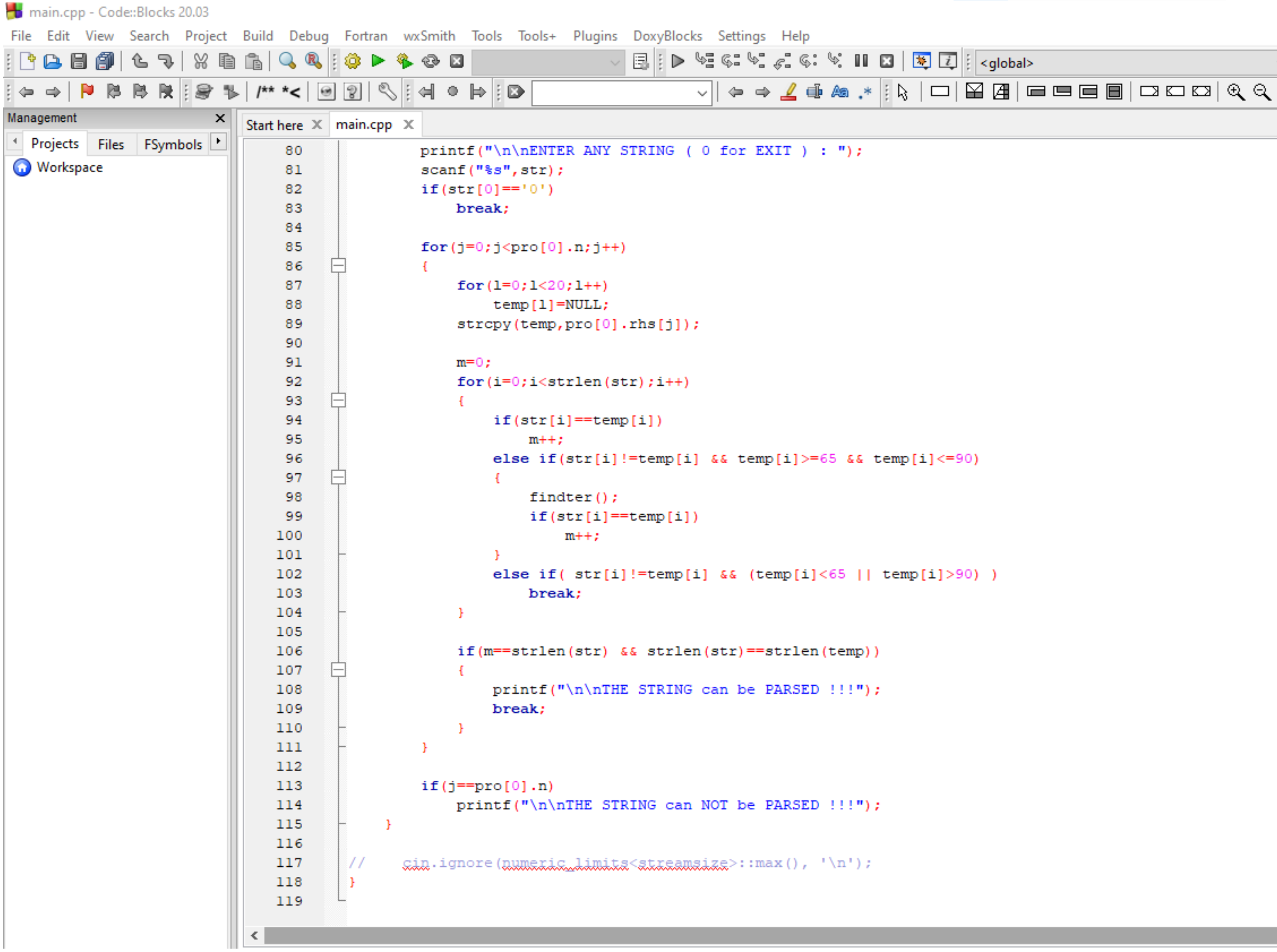
```
        string last = "";

        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;

    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. 6. Constructing LR(0) automation for the grammar



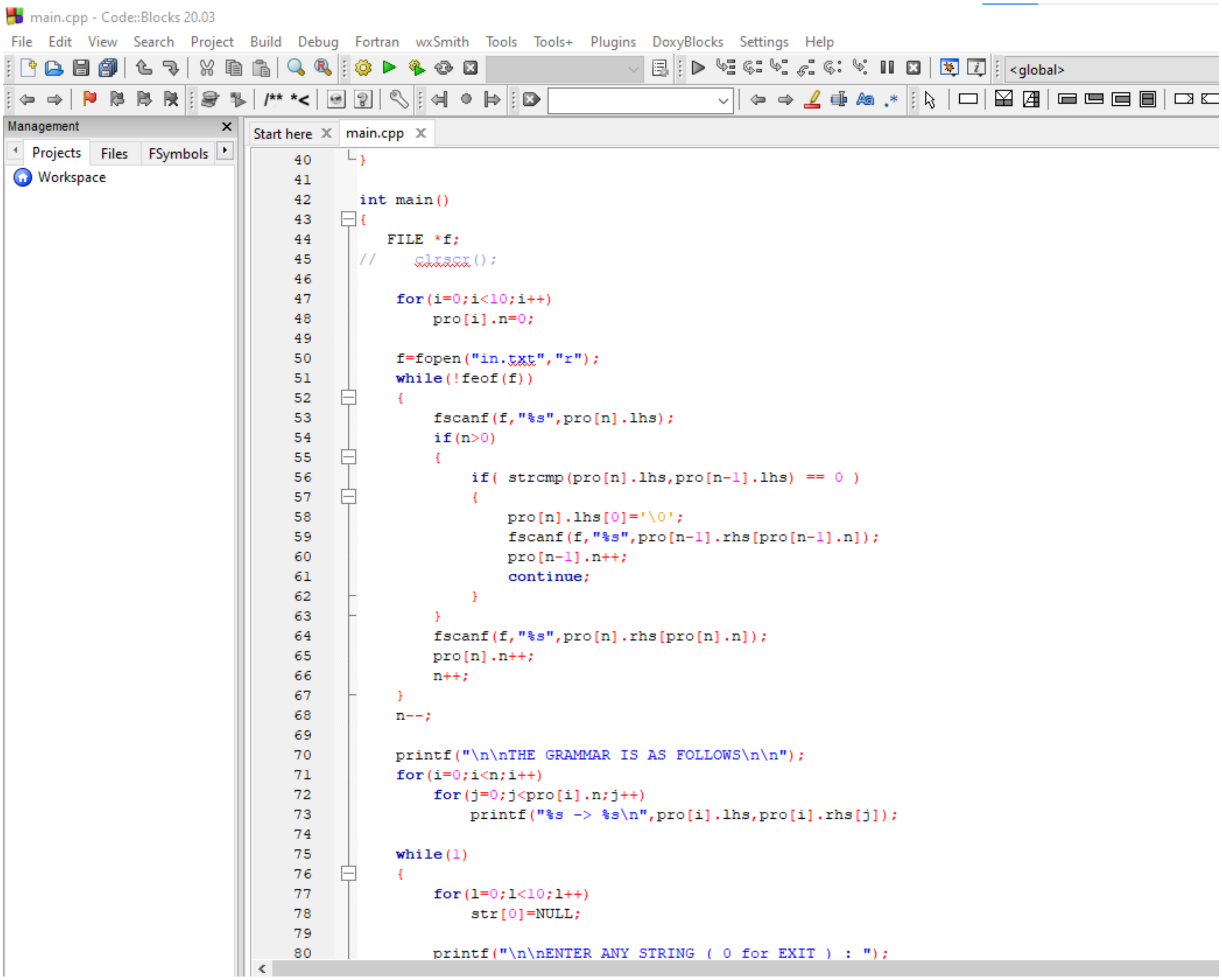
```

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

    paran_error = paranthesis_error(sp);

    if_else_err = if_else_error(sp);

    dup_token_err = dup_token_error(sp);

    paran_error.erase( unique( paran_error.begin(), paran_error.end() ), paran_error.end() );

    if_else_err.erase( unique( if_else_err.begin(), if_else_err.end() ), if_else_err.end() );

    dup_token_err.erase( unique( dup_token_err.begin(), dup_token_err.end() ), dup_token_err.end() );


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

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

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

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

    return 0;
}

```

Fig. 7. Constructing LR(0) automation for the grammar



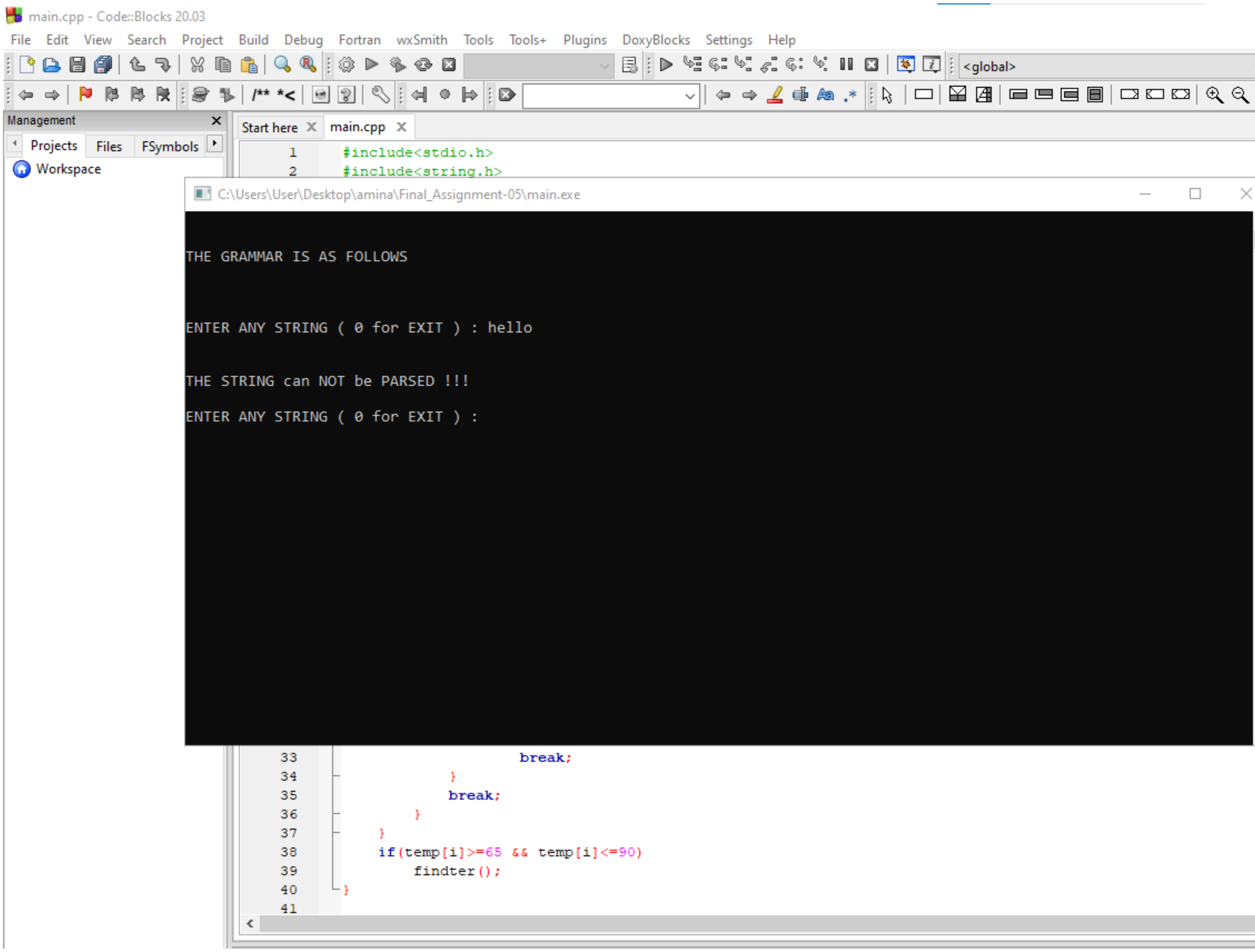
```
        printf("\n\nENTER ANY STRING ( 0 for EXIT ) : ");
        scanf("%s",str);
        if(str[0]=='0')
            break;

        for(j=0;j<pro[0].n;j++)
        {
            for(l=0;l<20;l++)
                temp[l]=NULL;
            strcpy(temp,pro[0].rhs[j]);

            m=0;
            for(i=0;i<strlen(str);i++)
            {
                if(str[i]==temp[i])
                    m++;
                else if(str[i]!=temp[i] && temp[i]>=65 && temp[i]<=90)
                {
                    findter();
                    if(str[i]==temp[i])
                        m++;
                }
                else if( str[i]!=temp[i]  && (temp[i]<65 || temp[i]>90) )
                    break;
            }

            if(m==strlen(str) && strlen(str)==strlen(temp))
            {
                printf("\n\nTHE STRING can be PARSED !!!");
                break;
            }
        }

        if(j==pro[0].n)
            printf("\n\nTHE STRING can NOT be PARSED !!!");
    }

//    cin.ignore(numeric_limits<streamsize>::max(), '\n');
}

```

Fig. 8. Constructing parsing table LR(1) parsing with the grammar


main.cpp - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
Management
Projects Files FSymbols
Workspace
Start here
main.cpp

```
}

int main()
{
    FILE *f;
//    clrscr();

    for(i=0;i<10;i++)
        pro[i].n=0;

    f=fopen("in.txt","r");
    while(!feof(f))
    {
        fscanf(f,"%s",pro[n].lhs);
        if(n>0)
        {
            if( strcmp(pro[n].lhs,pro[n-1].lhs) == 0 )
            {
                pro[n].lhs[0]='\0';
                fscanf(f,"%s",pro[n-1].rhs[pro[n-1].n]);
                pro[n-1].n++;
                continue;
            }
        }
        fscanf(f,"%s",pro[n].rhs[pro[n].n]);
        pro[n].n++;
        n++;
    }
    n--;

    printf("\n\nTHE GRAMMAR IS AS FOLLOWS\n\n");
    for(i=0;i<n;i++)
        for(j=0;j<pro[i].n;j++)
            printf("%s -> %s\n",pro[i].lhs,pro[i].rhs[j]);

    while(1)
    {
        for(l=0;l<10;l++)
            str[0]=NULL;

        printf("\n\nENTER ANY STRING ( 0 for EXIT ) : ");
```

Fig. 9. Constructing parsing table LR(1) parsing with the grammar

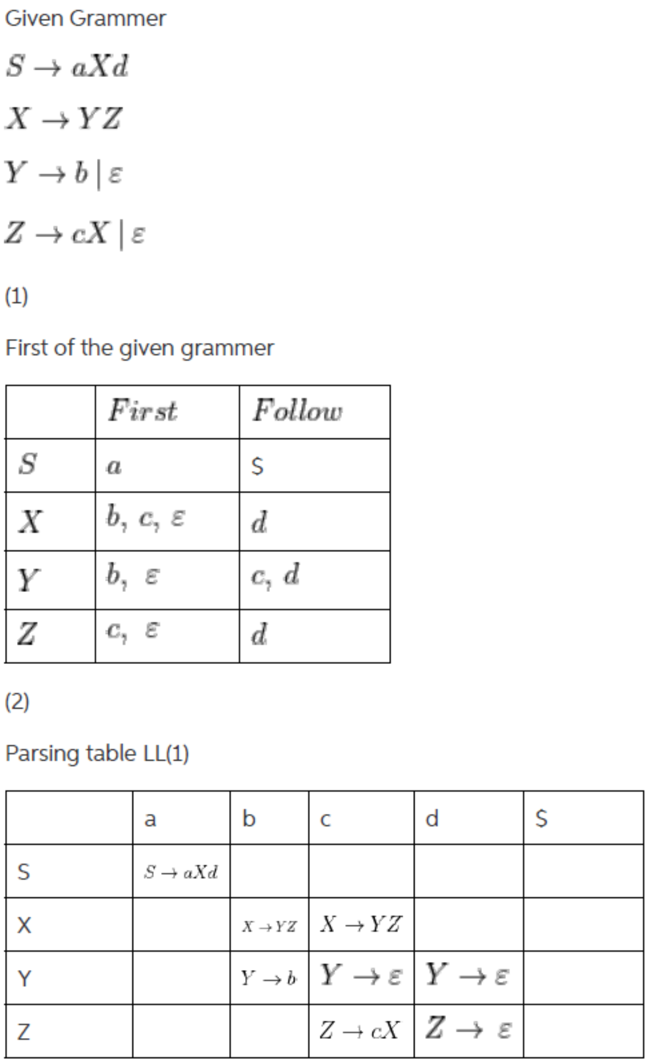

main.cpp - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
Management
Projects Files FSymbols
Workspace
Start here
main.cpp
C:\Users\User\Desktop\amina\Final_Assignment-05\main.exe

```
#include<stdio.h>
#include<string.h>
```

```
THE GRAMMAR IS AS FOLLOWS


ENTER ANY STRING ( 0 for EXIT ) : hello


THE STRING can NOT be PARSED !!!

ENTER ANY STRING ( 0 for EXIT ) :
```

```
                    break;
                }
                break;
            }
        }
        if(temp[i]>=65 && temp[i]<=90)
            findter();
}

```

Fig. 10. Constructing parsing table LR(1) parsing with the grammar



Given Grammer

$S \to aXd$

$X \to YZ$

$Y \to b \mid \varepsilon$

$Z \to cX \mid \varepsilon$

(1)

First of the given grammer

	$First$	$Follow$
S	a	\$
X	$b,\ c,\ \varepsilon$	d
Y	$b,\ \varepsilon$	$c,\ d$
Z	$c,\ \varepsilon$	d

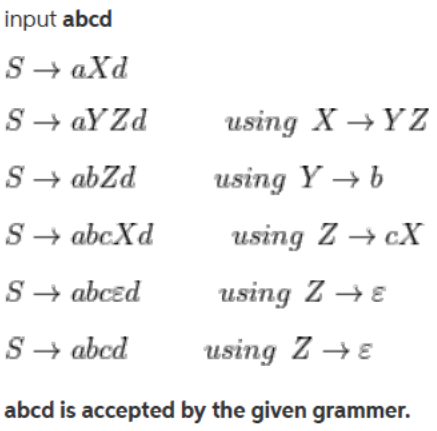
(2)

Parsing table LL(1)

	a	b	c	d	\$
S	$S \to aXd$				
X		$X \to YZ$	$X \to YZ$		
Y		$Y \to b$	$Y \to \varepsilon$	$Y \to \varepsilon$	
Z			$Z \to cX$	$Z \to \varepsilon$	

Fig. 11. Demonstrating moves of the LR(1) parser on the given input.

Assignment 6



input **abcd**

$S \to aXd$

$S \to aYZd \quad using\ X \to YZ$

$S \to abZd \quad using\ Y \to b$

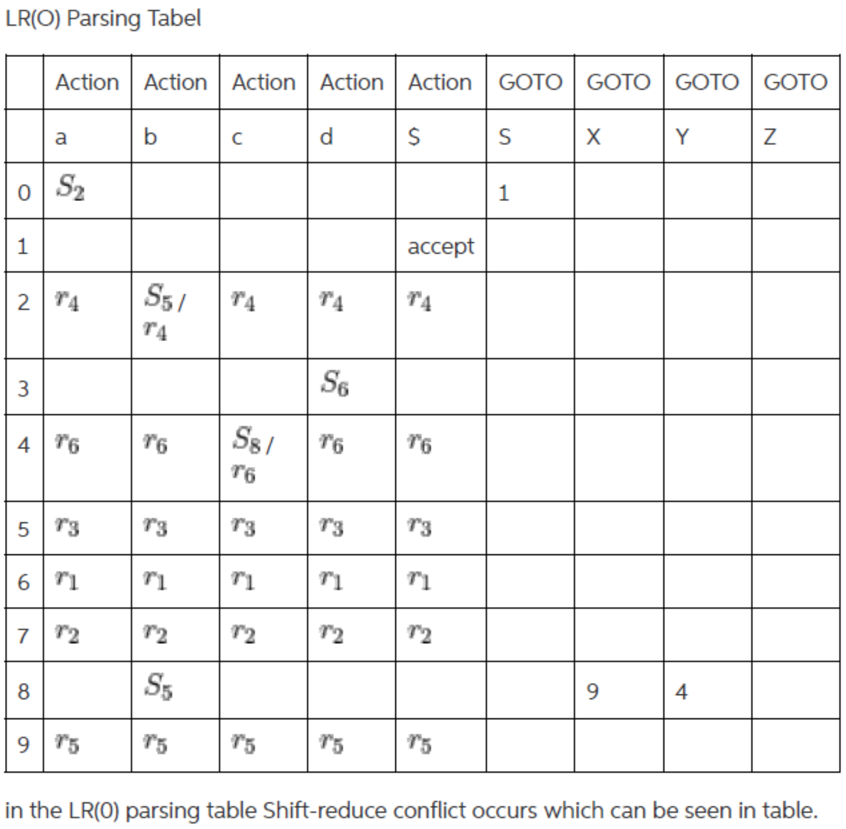
$S \to abcXd \quad using\ Z \to cX$

$S \to abc\varepsilon d \quad using\ Z \to \varepsilon$

$S \to abcd \quad using\ Z \to \varepsilon$

abcd is accepted by the given grammer.

Fig. 12. Demonstrating moves of the LR(1) parser on the given input.



LR(O) Parsing Tabel

	Action	Action	Action	Action	Action	GOTO	GOTO	GOTO	GOTO
	a	b	c	d	$	S	X	Y	Z
0	S_2					1			
1					accept				
2	r_4	S_5 / r_4	r_4	r_4	r_4				
3				S_6					
4	r_6	r_6	S_8 / r_6	r_6	r_6				
5	r_3	r_3	r_3	r_3	r_3				
6	r_1	r_1	r_1	r_1	r_1				
7	r_2	r_2	r_2	r_2	r_2				
8		S_5					9	4	
9	r_5	r_5	r_5	r_5	r_5				

in the LR(0) parsing table Shift-reduce conflict occurs which can be seen in table.

Fig. 13. Demonstrating moves of the LR(1) parser on the given input.

(4)

LR(0) grammar

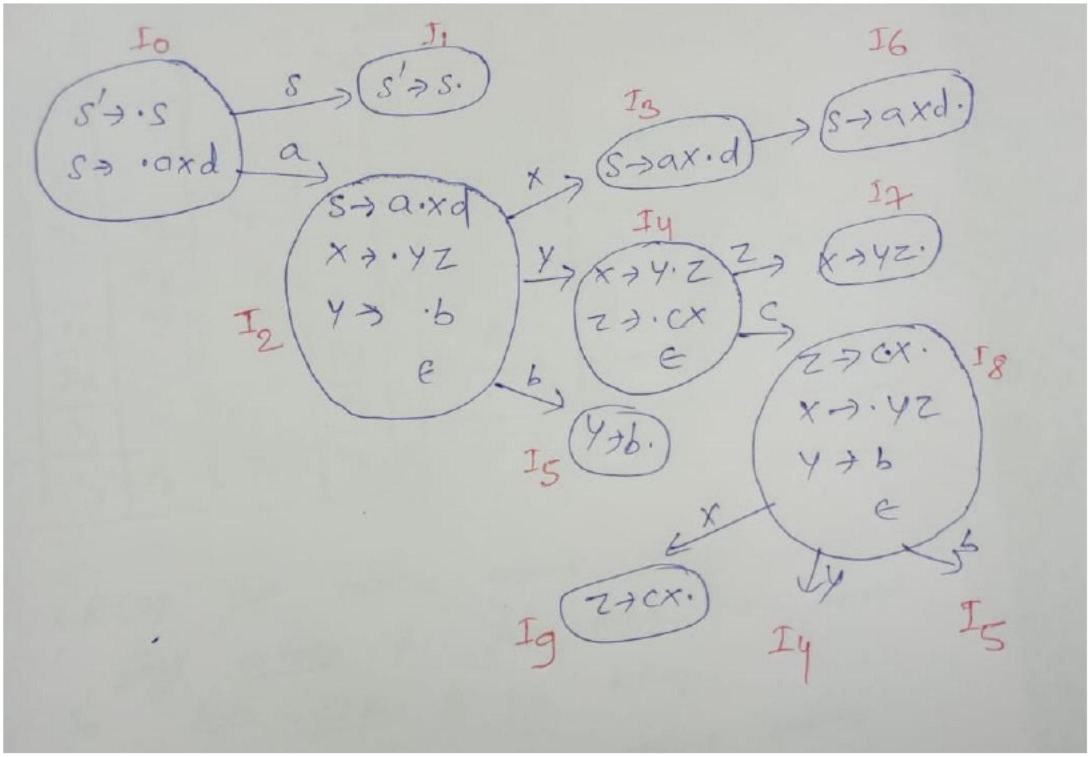


Fig. 14. Demonstrating moves of the LR(1) parser on the given input.

(5)

augumented grammar for LR(1) Parsing table

Fig. 15. Demonstrating moves of the LR(1) parser on the given input.

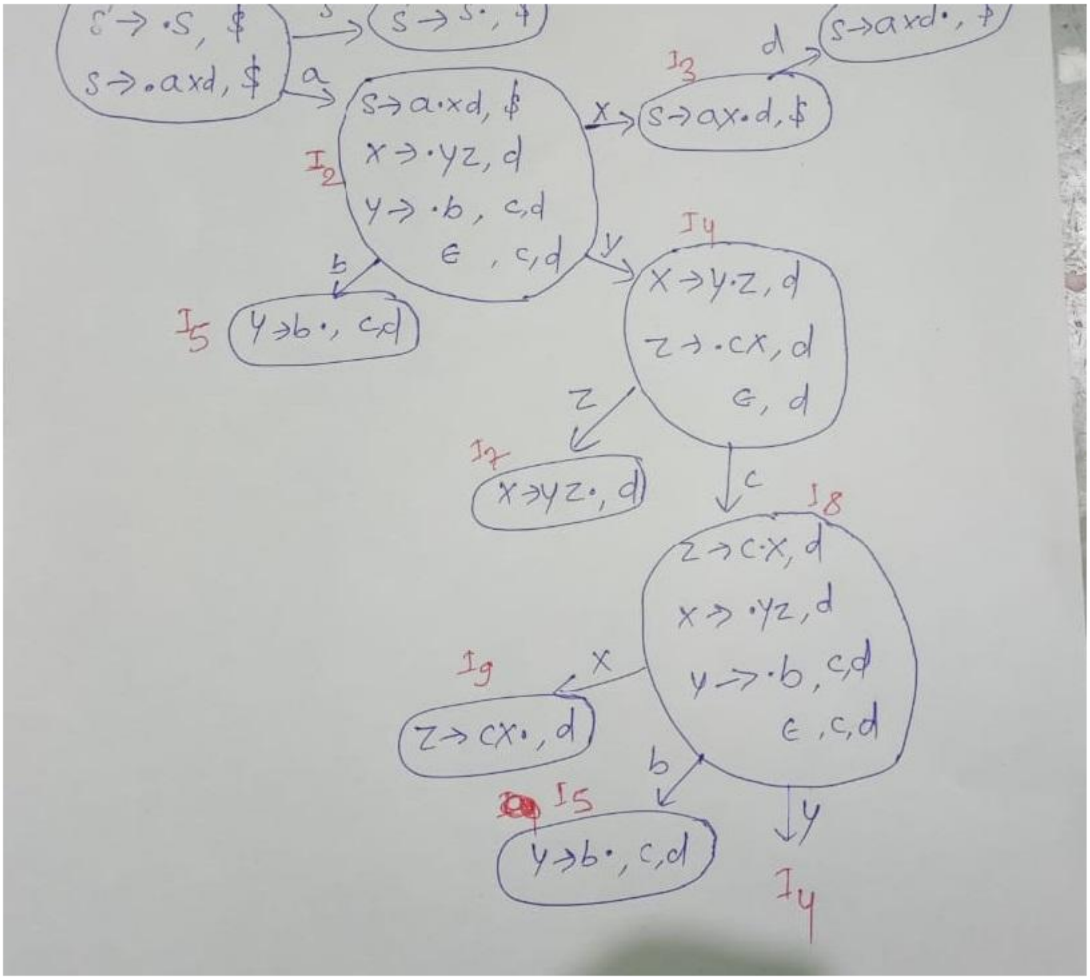
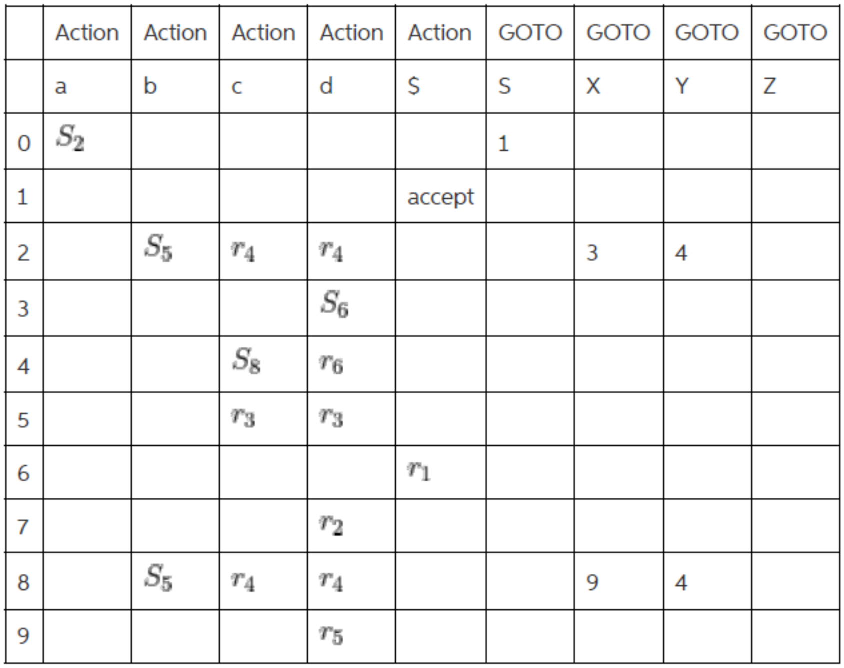
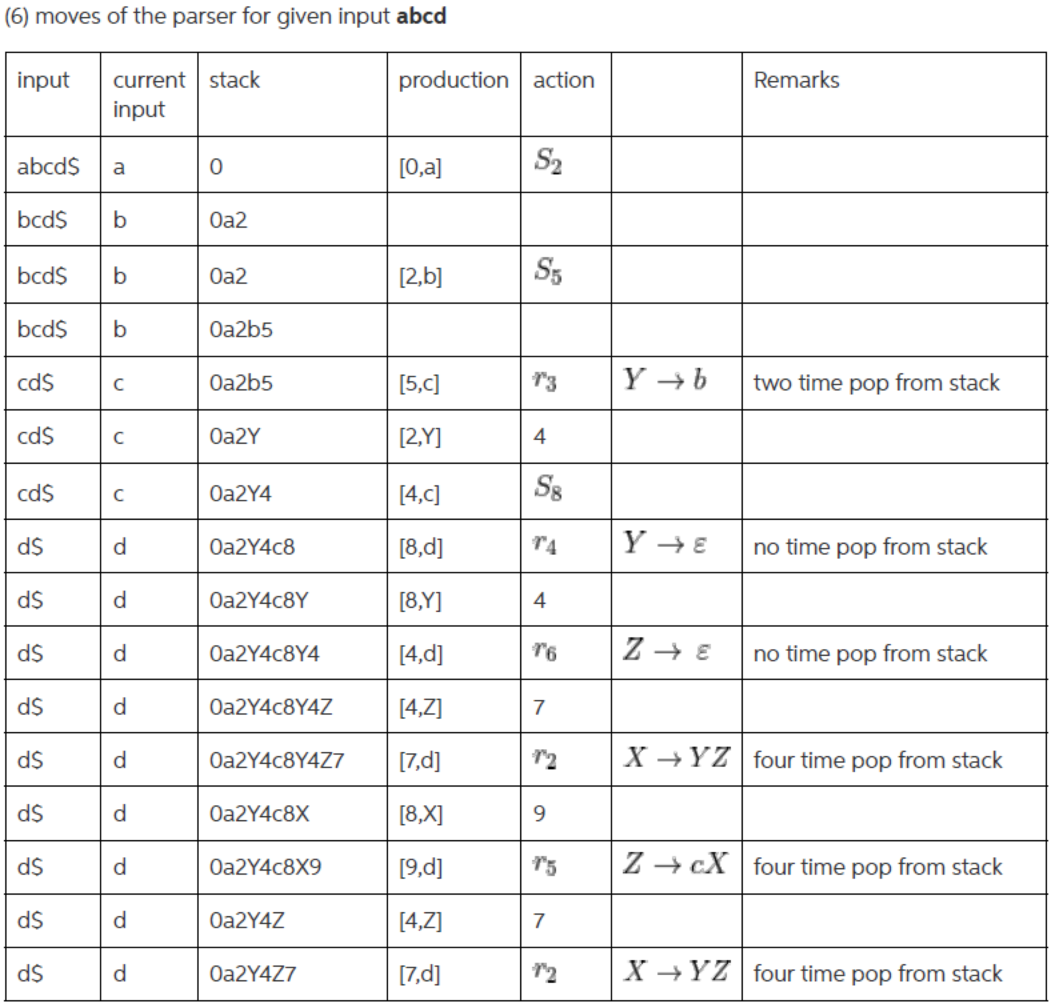


Fig. 16. Demonstrating moves of the LR(1) parser on the given input.



	Action	Action	Action	Action	Action	GOTO	GOTO	GOTO	GOTO
	a	b	c	d	$	S	X	Y	Z
0	S_2					1			
1					accept				
2		S_5	r_4	r_4			3	4	
3				S_6					
4			S_8	r_6					
5			r_3	r_3					
6					r_1				
7				r_2					
8		S_5	r_4	r_4			9	4	
9				r_5					

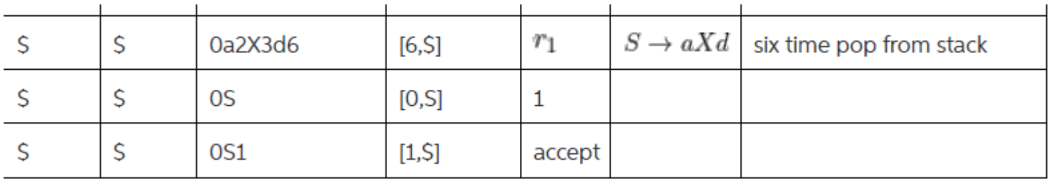
Fig. 17. Demonstrating moves of the LR(1) parser on the given input.



(6) moves of the parser for given input **abcd**

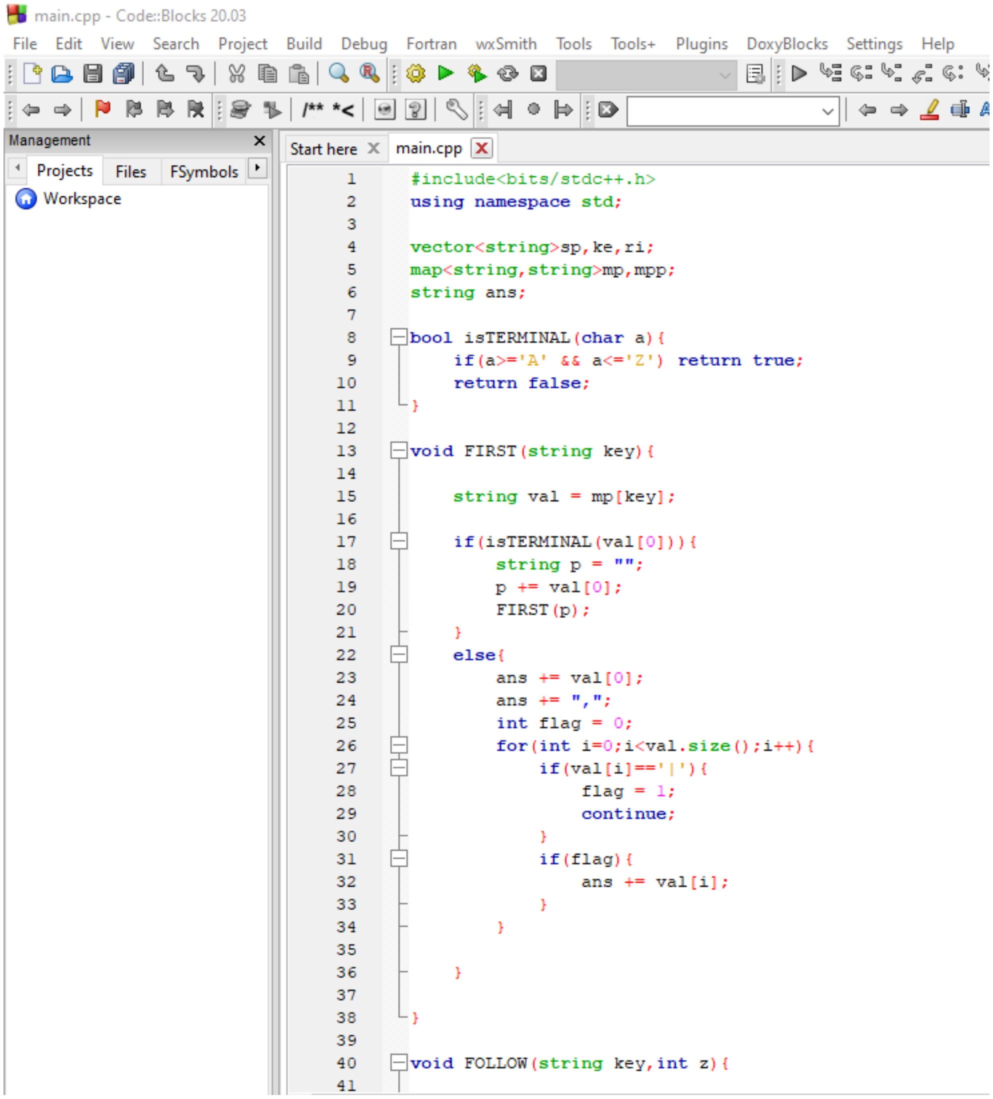
input	current input	stack	production	action		Remarks
abcd$	a	0	[0,a]	S_2		
bcd$	b	0a2				
bcd$	b	0a2	[2,b]	S_5		
bcd$	b	0a2b5				
cd$	c	0a2b5	[5,c]	r_3	$Y \rightarrow b$	two time pop from stack
cd$	c	0a2Y	[2,Y]	4		
cd$	c	0a2Y4	[4,c]	S_8		
d$	d	0a2Y4c8	[8,d]	r_4	$Y \rightarrow \varepsilon$	no time pop from stack
d$	d	0a2Y4c8Y	[8,Y]	4		
d$	d	0a2Y4c8Y4	[4,d]	r_6	$Z \rightarrow \varepsilon$	no time pop from stack
d$	d	0a2Y4c8Y4Z	[4,Z]	7		
d$	d	0a2Y4c8Y4Z7	[7,d]	r_2	$X \rightarrow YZ$	four time pop from stack
d$	d	0a2Y4c8X	[8,X]	9		
d$	d	0a2Y4c8X9	[9,d]	r_5	$Z \rightarrow cX$	four time pop from stack
d$	d	0a2Y4Z	[4,Z]	7		
d$	d	0a2Y4Z7	[7,d]	r_2	$X \rightarrow YZ$	four time pop from stack

Fig. 18. Demonstrating moves of the LR(1) parser on the given input.



$	$	0a2X3d6	[6,$]	r_1	$S \rightarrow aXd$	six time pop from stack
$	$	0S	[0,S]	1		
$	$	0S1	[1,$]	accept		

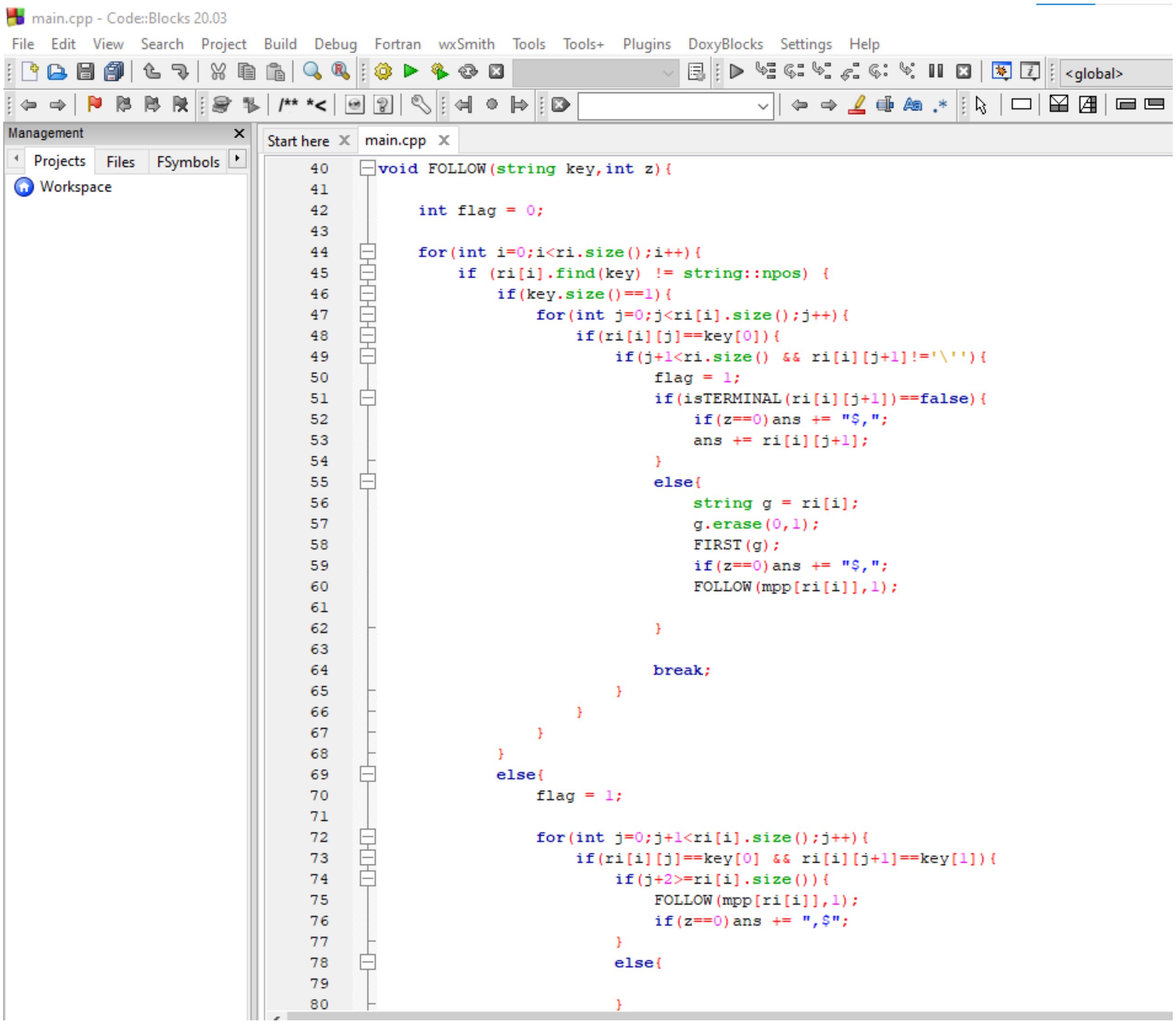
Fig. 19. Demonstrating moves of the LR(1) parser on the given input.


main.cpp - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects
Files
FSymbols
Workspace
Start here
main.cpp

```cpp
#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);
    }
    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];
            }
        }

    }

}

void FOLLOW(string key,int z){
```

Coding section

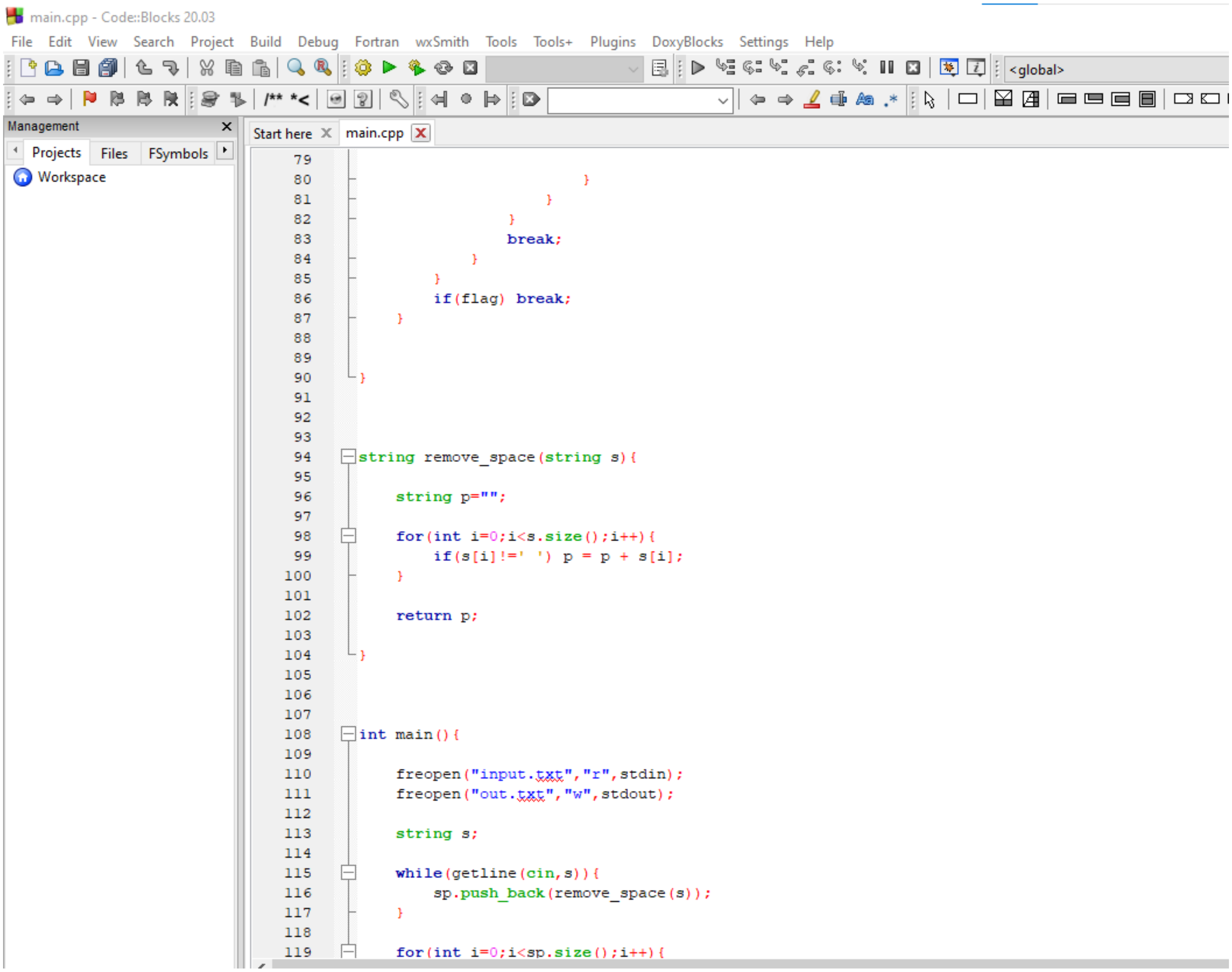
Fig. 20. Demonstrating moves of the LR(1) parser on the given input.



main.cpp - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
Management
Projects Files FSymbols
Workspace
Start here
main.cpp

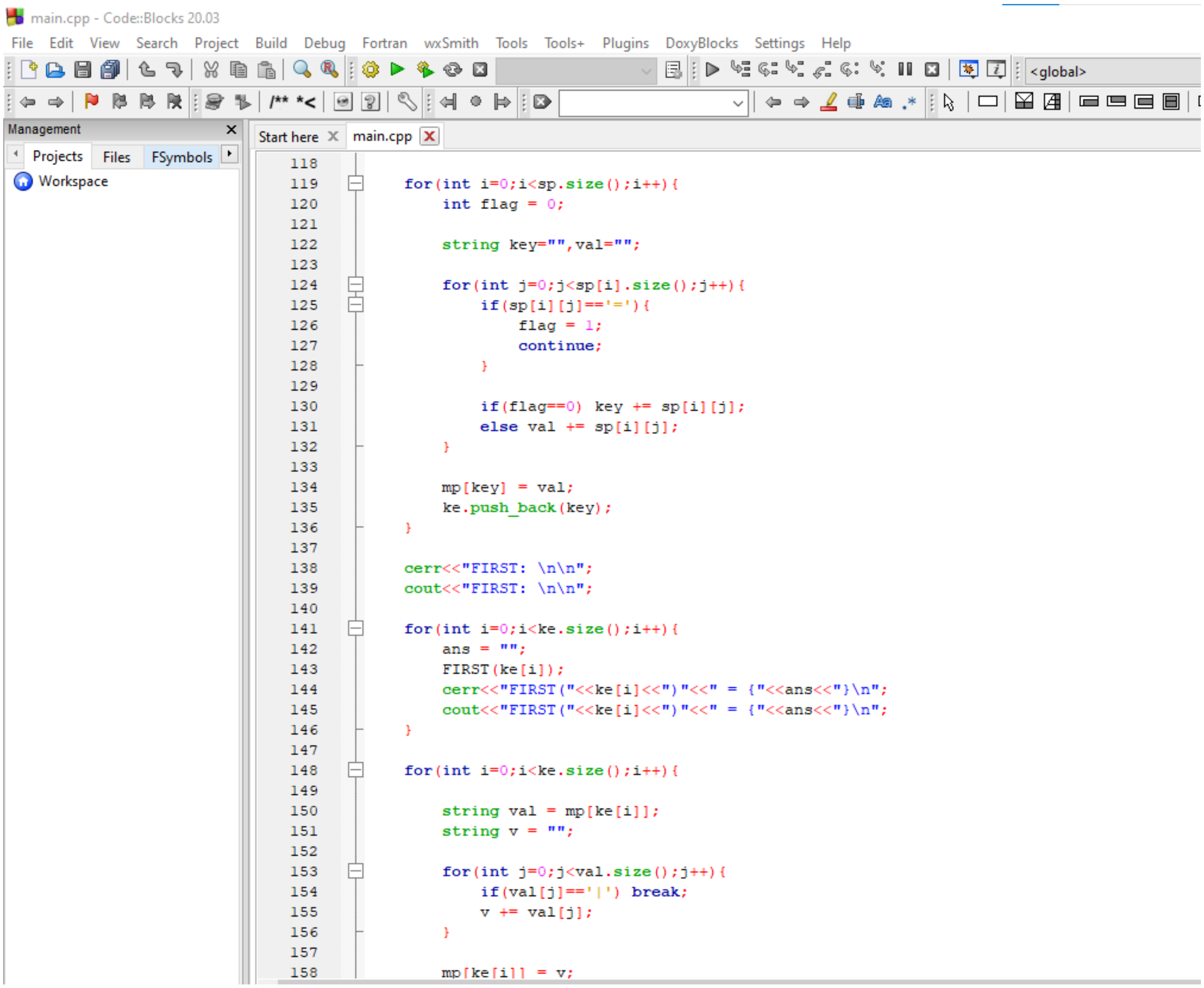
```cpp
void FOLLOW(string key,int z){

    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 += "$,";
                                ans += ri[i][j+1];
                            }
                            else{
                                string g = ri[i];
                                g.erase(0,1);
                                FIRST(g);
                                if(z==0)ans += "$,";
                                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. 21. Demonstrating moves of the LR(1) parser on the given input.



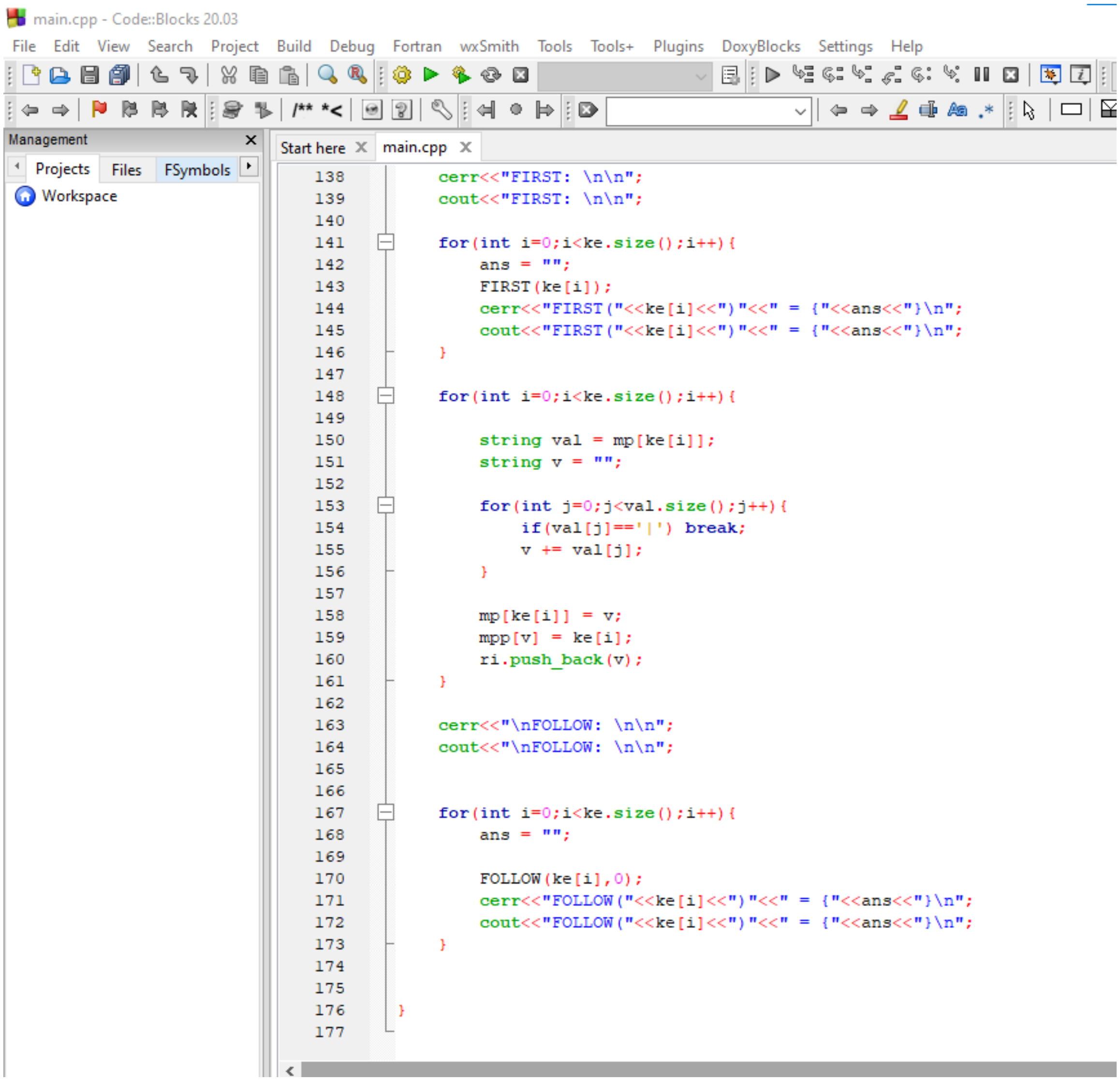
```

                        }
                    }
                }
                break;
            }
        }
        if(flag) break;
    }


}



string remove_space(string s){

    string p="";

    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));
    }

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

Fig. 22. Demonstrating moves of the LR(1) parser on the given input.



```

    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";
    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];
        }

        mp[ke[i]] = v;
```

Fig. 23. Demonstrating moves of the LR(1) parser on the given input.



```
    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];
        }

        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. 24. Demonstrating moves of the LR(1) parser on the given input.