AMERICAN INTERNATIONAL UNIVERSITY–BANGLADESH (AIUB)

Final Term Assignment

Spring 2022-2023

Name : Punam Das

ID : 21-44946-2

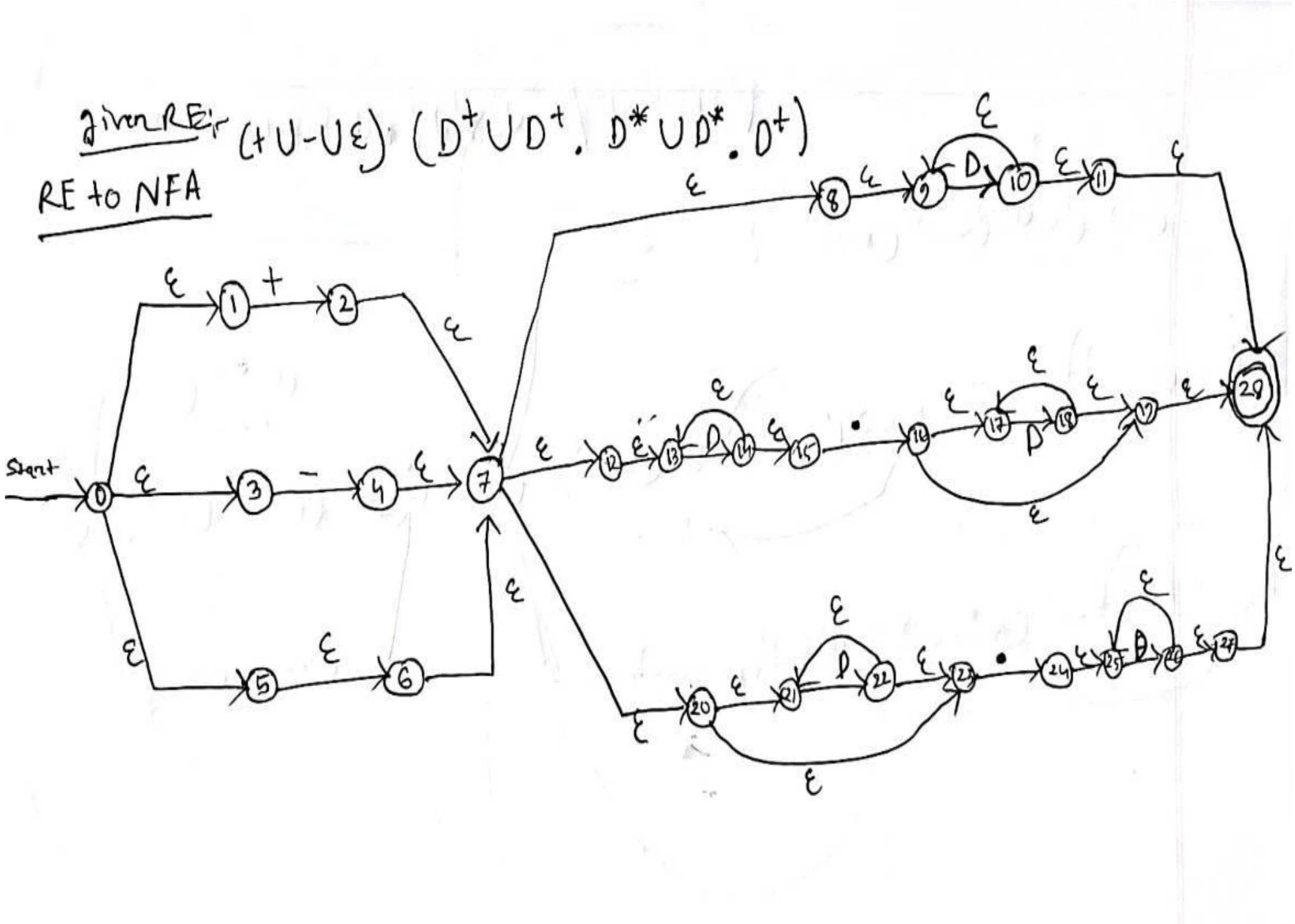
Section : E

Course : Compiler Design

Theory Part

From page 66 of the book we get The Regular Expression

RE TO NFA

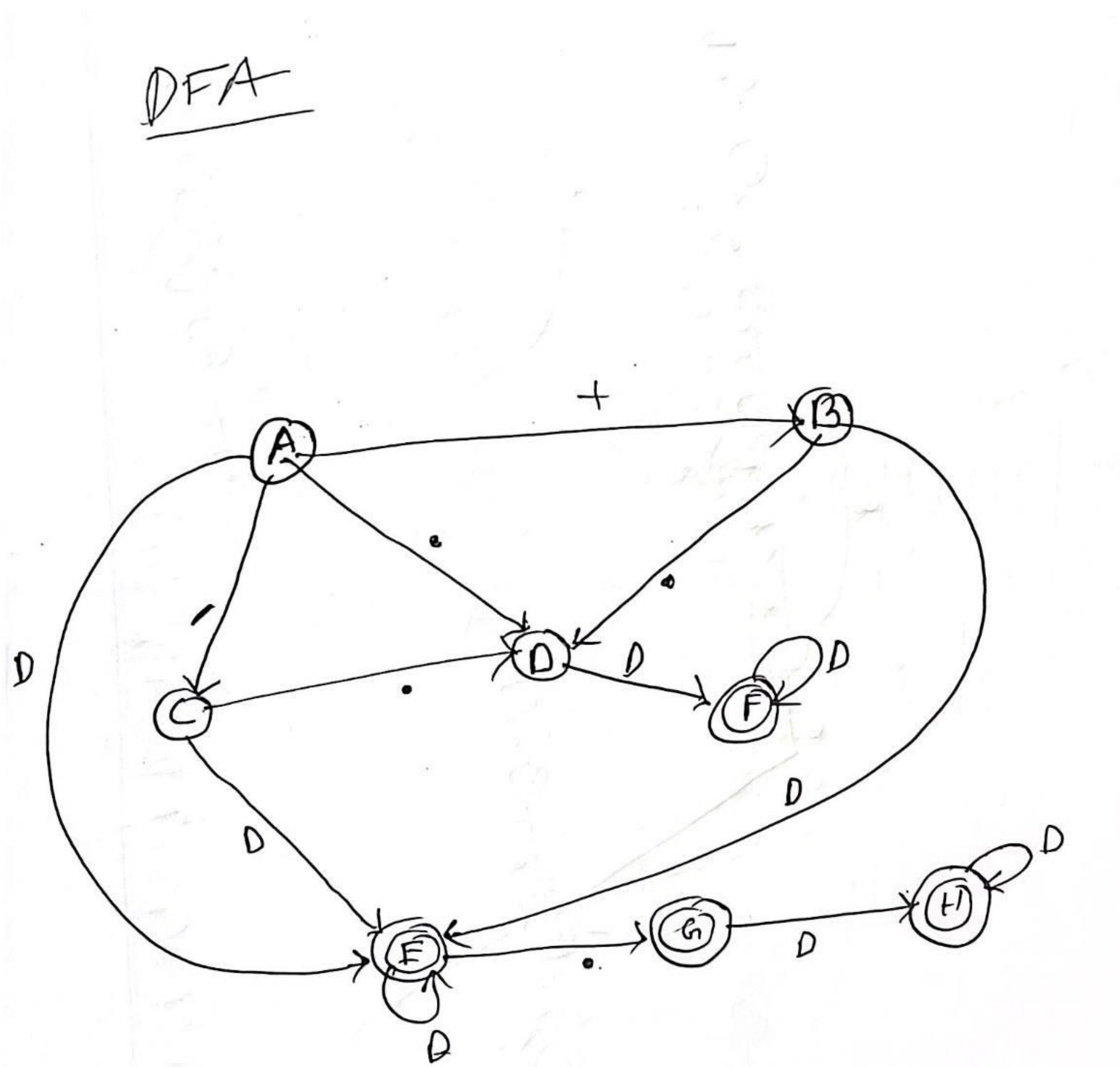


8(1)

DFA State	E-closure of	E-closure outcome states	+	-	.	D
A	{0}	{0,1,3,5,6,7,8,12,13, 9,20,21,23}	{2}	{4}	{24}	{10,14,22}
B	{2}	{2,7,8,9,12,13,20,21, 23}	{}	{}	{24}	{10,14,22}
C	{4}	{4,7,8,9,12,13,20, 21,23}	{}	{}	{24}	{10,14,22}
D	{24}	{24,25}	{}	{}	{}	{26}
E*	{10,14,22}	{9,10,11,13,14,15, 21,22,23,28}	{}	{}	{16,24}	{10,14,22}
F*	{26}	{26,25,27,28}	{}	{}	{}	{26}
G*	{16,24}	{16,17,18,19,~~24~~,25,28}	{}	{}	{}	{18,26}
H*	{18,26}	{17,18,19,25,26,27, 28}	{}	{}	{}	{18,26}

NFA States	DFA States	+	-	.	D
{0,1,3,5,6,7,8,12,13, 9,20,21,23}	A	B	C	D	E
{2,7,8,9,12,13 20,21,23}	B			D	E
{4,7,8,9,12, 13,20,21,23}	C			D	E
{24,25}	D				F
{9,10,11,13,14,15, 21,22,23,28}	E*			G	E
{26,25}	F*				F
{16,17,18,19,24, 25,28}	G*				H
{17,18,19,25 26,27,28}	H*				H

NFA To DFA



Using this DFA we have constructed to verify whether the input string "+183.47" is a valid number. We will simulate the process a Lexical Analyzer would perform:

1. **Initial State - A**: The Lexical Analyzer starts at the initial state A.
2. **Input '+':** There is no explicit transition for '+' in our DFA because we simplified the DFA by merging states q1 and q2 into B. In a complete lexical analyzer, transitions for '+' and '-' would be explicitly defined. For this example, assume the '+' transitions to the state that expects digits, which would be B.
3. **Input '1':** From B, upon reading a digit '1', the DFA move to E, which is the state for a sequence of digits.
4. **Input '8':** Reading the next digit '8', the DFA still stays in E.
5. **Input '3':** Reading the digit '3', the DFA continues to remain in E.
6. **Input '.':** Encountering the decimal point '.', the DFA transitions from E to G, the state that expects at least one digit following the decimal point.
7. **Input '4':** From G, on reading the digit '4', the DFA moves to H.
8. **Input '7':** Upon reading the digit '7', the DFA remains in H.

Since the string ends and the DFA is in the accepting state H, the Lexical Analyzer concludes that "+183.47" is a valid number according to the regular expression and the constructed DFA.

Let's walk through this step-by-step with a visual representation.

It appears there was a misunderstanding in the simulation. The constructed DFA does not include transitions for the **+** symbol, as it was not explicitly defined in the DFA's transition table. This is because in the regular expression to NFA conversion, the **+** and **-** were considered as optional leading characters and were consumed in the NFA's initial ε-transitions to simplify the diagram.

In a complete lexical analyzer, we would have explicit transitions for both **+** and **-** to account for signed numbers. These transitions would lead to the state expecting digits (B in our case).

To correct this, we should update our DFA to include transitions for the **+** and **-** symbols from the initial state. Since we cannot modify the DFA directly at this point, let's conceptually include these transitions and simulate the DFA manually:

1. Start at **A**.
2. The **+** sign would transition us to the state that expects the first digit, which would be **B** conceptually.
3. The first digit **1** then move in to **DFA state E**.
4. The second digit **8** also keeps us in E.
5. The third digit **3** keeps us in **E**.
6. The **.** symbol transitions us to **G**, expecting digits after the decimal point.
7. The digit **4** transitions us to **H**, where we can have additional digits after the decimal point.
8. The digit **7** keeps us in **H**.

Since the input string ends while we are in **H**, which is an accepting state, the Lexical Analyzer would determine that "+183.47" is indeed a valid number according to the DFA's rules.

LAB Part

Syntax Analyzer Code

Global Variables:

```
int children_counter[50];
//An array to store the count of children for each node in the syntax tree.
int main() {
```

Variable Declarations:

```
int syntax_tree_root, start_variable, top_rule, left_variable;
int input_token_index, statement_index;
string input_token;
char statement[7]; // Declaration of various integer variables and arrays.
```

Grammar Rules:

```
char grammar_rules[6][11] = {"A->BCDEDF.","B->i.","C->=.","D->i.","E-
>+.","F->;."};
```

‘A’	‘-’	‘>’	‘B’	‘C’	‘D’	‘E’	‘D’	‘F’	‘.’	‘\0’
0	1	2	3	4	5	6	7	8	9	10

```
//Array of strings representing grammar rules for the language.
```

Syntax Tree and Variable Replacer Initialization:

```
char variable_replacer[50];
char syntax_tree[50];
//Initialization of arrays to store the variable replacer and syntax tree.
```

Input Token Prompt and Display:

```
cout<< "Enter the input token:";
```

```
getline (cin, input_token);
//Prompt the user to enter an input token and display the entered token.
<id,0><=><id,1><+><id,2><;>
```

Input Token:

<	i	d	,	0	>	<	=	>	<	i	d	,	1	>	<	+	>	<	i	d	,	2	>	<	;	>
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

```
cout<< "The entered token was:"<< <id,0><=><id.1><+><id.2><;>
<< endl;
```

Statement Extraction from Input Token:

```
for(input_token_index=0;input_token_index<input_token.length();input_tok
en_index++){
   // Code extracts characters between '<' and '>'
}
//for loop up to 27
statment_index=0; // initially statement_index 0 input_token
index.length()=27:// for loop up to 27
statement[statement_index] =input_token[input_token_index+1];
input token index=0;(‘<’)
statement[0] = input_token[0+1];
statement[0]=’i’; statement_index++;

input_token_index=6; （’<’）
statement[1] = input_token[6+1];
statement[1]=’=’; statement_index ++;
input_token_index=9;(‘<’)
```

statement[2] = input token[9+1];
statement[2] =’i’;
statement _index++;

input_token_index=15;(‘<’)
statement[3] = input token[15+1];
statement[3] =’+’;
statement _index++;

input_token_index=18;(‘<’)
statement[4] = input token[18+1];
statement[4] =’+’;
statement _index++;

input_token_index=24;(‘<’)
statement[5] = input token[24+1];
statement[5] =’+’;
statement _index++;

for statement array for token

i	=	i	+	i	;	
0	1	2	3	4	5	6

Initialization of Variables for Syntax Tree Construction:

syntax_tree_root=0;
start_variable=0;
top_rule=0;
left_variable=0;
//Initialization of variables used in the syntax tree construction.

Initial Syntax Tree and Variable Replacer Assignment:

syntax_tree[syntax_tree_root]=grammar_rules[top_rule][left_variable];
syntax tree[0]= grammar rules[0][0]
syntax tree[0]='A'
//This line assigns a value to the element at the index 'syntax_tree_root' in the ' syntax_tree array'. The assigned value comes from the ' grammar_rules' array at the position specified by 'top_rule 'and 'left_variable'. It seems to represent the initial assignment of a variable or non-terminal symbol to the root of the syntax tree.
variable_replacer[start_variable]=grammar_rules[top_rule][left_variable];
variable_replacer [0]= grammar rules[0][0]
variable_replacer [0]='A'

//This line assigns a value to the element at the index 'start_variable' in the 'variable_replacer' array. The assigned value also comes from the 'grammar_rules' array at the position specified by 'top_rule' and 'left_variable'. This appears to represent the initial assignment of a variable or non-terminal symbol to a specific position in the variable replacer.

Main Loop for Syntax Analysis:

```
while(true){
   // Code for syntax analysis
}
```

//The main loop for syntax analysis. It iteratively constructs the syntax tree based on the grammar rules and checks for syntax errors.

In the WHILE LOOP:

1. while(true) //When our statement array current index and the vanable replacer terminal state will be same until then statement- variable replacer

(break the while loop) and if statement array current index and the variable replacer terminal state is not same then it will be error (break the whileloop).

2.Grammar Rule Matching:

```
for(grammar_rule_number=0;grammar_rule_number<6;grammar_rule_num
ber++){
   // Code for matching grammar rules
}
```

//This loop finds a grammar rule that matches the current variable in the variable replacer.

For loop// For every child can use this 6-grammar rules

Grammar:

"A->BCDEDE.",//grammar rule number 0;
"B->i.", //grammar rule number I;
“C->=", //grammar rule number 2;
"D->i", //grammar rule number 3;
“E->+”,//grammar rule number4;
"F->;" ,//grammar_rule number 5;

3. If Condition:

/grammar_rules|grammar_rule _number][left_variable]variable replacer[variable replacer current index]
grammar_ rules[0][0] = variable replacer[0]: //’A==A’

4.Copying Grammar Rule to Syntax Tree and Variable Replacer:

```
for(rule_right_side_copier_index=0;grammar_rules[grammar_rule_number]
[rule_right_side_copier_index]!='.';rule_right_side_copier_index++){
   // Code for copying the right- side of the grammar rule to syntax tree and
variable replacer
}
```

//The loop copies the right- side of the matched grammar rule to the syntax tree and variable replacer. And this loop will continue until we find in a rule

5. If Condition// it will visit until we don't find this symbol '>'
grammar rules[grammar rule number][rule right side copier index] ==’>’
grammar rules [0][0]= 'A'
gramiar_rules[0][1]=’-‘
grammar_rules[0][2]=’<’
copy_flag = 1: continue.
//until this for loop find ‘>’ It can't enter this loop;

6
If condition: copy flag ==1
children counter[current parent]++;
syntax_tree[syntax_tree_current_index] =
grammar rules[grammar_rule number][rule right side copier index]:

variable replacer[variable replacer curent_index]=
grammar rules|grammar rule number|[rule right side copier index];

Children Count

Children Count=1;++
syntax tree[1] = grammar rules[0][3]’//’B’
syntax_tree_current_index++;
variable replacer[0] = grammar _rules[0][3];//’B’
variable replacer current index++;
Children Count=2++
syntax tree[2] = grammar rules[0][4]’//’C’
syntax_tree_current_index++;
variable replacer[1] = grammar _rules[0][4];//’C’
variable replacer current index++;

Children Count=3;
syntax tree[3] = grammar rules[0][5]'//'D'
syntax_tree_current_index++;
variable replacer[2] = grammar _rules[0][5];//'D'
variable replacer current index++;

Children Count=4;
syntax tree[4] = grammar rules[0][6]'//'E'
syntax_tree_current_index++;
variable replacer[3] = grammar _rules[0][6];//'E'
variable replacer current index++;

Children Count=5
syntax tree[5] = grammar rules[0][7]'//'D'
syntax_tree_current_index++;
variable replacer[4] = grammar _rules[0][7];//'D'
variable replacer current index++;
Children Count=6
syntax tree[6] = grammar rules[0][8]'//'F'
syntax_tree_current_index++;
variable replacer[5] = grammar _rules[0][8];//'F'
variable replacer current index++;

CHECK GRAMER RULE

grammar_rules[0]

'A'	'-'	'>'	'B'	'C'	'D'	'E'	'D'	'F'	'.'	'\0'
0	1	2	3	4	5	6	7	8	9	10

grammar rules[1]

'B'	'-'	'>'	'i'	'\0'						
0	1	2	3	4	5	6	7	8	9	10

grammar rule_Number 1;
grammar_ rules|1][0] = variable_replacer[5]
B="F"/FALSE
grammar_rule_number ++

grammar_rule number 2;
grammar rules[2][0] = variable replacer[5]
C="F"//FALSE
grammar_rule_number ++;

grammar rule number 3;
grammar rules[3][0] = variable replacer[5]
D="F'//FALSE
grammar_rule_number++;

grammar rule_number 4;
grammar rules|4][0]= variable replacer[5]
E="F//FALSE
grammar rule number++;

grammar_rule_number 5:

grammar rules[5][0] = vanable replacer[5]
D="E"// FALSE
grammar rule_number++.

Check for syntax errors :

```
                    if(variable_replacer[variable_replacer_current_index]=='i' ||
variable_replacer[variable_replacer_current_index]=='=' ||
variable_replacer[variable_replacer_current_index]=='+' ||
variable_replacer[variable_replacer_current_index]==';'){
Variable Replacer[0]='B'
Variable Replacer[1]='C'
```

All are non terminal

B	C	D	E	D	F				
0	1	2	3	4	5	6	7	8	9

Break flag=0;

```
if(syntax_error_break==1)
    break;
```

Checks if syntax_error_break is equal to 1.
//If true, it breaks out of the loop. This is likely used to exit the loop in case a syntax error is detected.

```
cout << "variable replacer after copy: " << variable_replacer_current_index << endl;
```

//Prints the value of variable_replacer_current_index to the console. This is likely for debugging or informational purposes.

```
variable_replacer_current_index = last_index;
```

//Updates variable_replacer_current_index with the value of last_index. It seems to reset the current index to a certain point.

```
int variable_replacer_traverser;
for(variable_replacer_traverser=last_index; variable_replacer_traverser<50;
variable_replacer_traverser++){
   if (variable_replacer[variable_replacer_traverser]=='i' ||
variable_replacer[variable_replacer_traverser]=='=' ||
variable_replacer[variable_replacer_traverser]=='+' ||
variable_replacer[variable_replacer_traverser]==';'){
        last_index = last_index + 1;
        variable_replacer_current_index = last_index;
   }
```

```
    if(variable_replacer[variable_replacer_traverser]!='i' &&
variable_replacer[variable_replacer_traverser]!='=' &&
variable_replacer[variable_replacer_traverser]!='+' &&
variable_replacer[variable_replacer_traverser]!=';')
        break;
}
//A loop that traverses the variable_replacer array starting from last_index.
//If the current element is 'i', '=', '+', or ';', it increments last_index and updates
//variable_replacer_current_index.
//The loop breaks if the current element is not 'i', '=', '+', or ';'.


current_parent = current_parent + 1;//B
cout << endl << "-----" << endl;
cout << endl << "Variable Replacer Current Index" << variable_replacer_current_index;
cout << endl << "Current Parent" << current_parent;
cout << endl << "Syntax Tree Current Index" << syntax_tree_current_index;
//Increments current_parent by 1.
//Prints information about variable_replacer_current_index, current_parent, and
//syntax_tree_current_index to the console.


int m, break_flag=1;
for(m=0; variable_replacer[m]!='.'; m++){
    if(variable_replacer[m]!='i' && variable_replacer[m]!='=' && variable_replacer[m]!='+'
&& variable_replacer[m]!=';')
        break_flag=0;
}

if(break_flag==1)
    break;
//A loop that iterates through variable_replacer until it finds a '.' (end marker).
Sets break_flag to 0 if an element other than 'i', '=', '+', or ';' is encountered.
//If break_flag remains 1 (meaning all elements are 'i', '=', '+', or ';'), it breaks out of the
loop.
iteration = iteration + 1;
Increments the iteration variable by 1.

int loop_counter;
```

```
cout << endl << "Syntax Tree :" << endl;
for(loop_counter=0; loop_counter<50; loop_counter++){
   cout << syntax_tree[loop_counter] << endl;
}
```

The syntax tree will be

A	B	C	D	E	D	F	i	=	i	+	i	;
0	1	2	3	4	5	6	7	8	9	10	11	12

```
//Prints the content of the syntax_tree array to the console.
cout << "Variable Replacer :" << endl;
for(loop_counter=0; loop_counter<50; loop_counter++){
   cout << variable_replacer[loop_counter] << endl;
}
//Prints the content of the variable_replacer array to the console.
```

Variable replacer:

Initially it's 0 indes it has first rule's parent which is A

A												
0	1	2	3	4	5	6	7	8	9	10	11	12

Veriable replacer(All are non terminal)

B	C	D	E	D	F				
0	1	2	3	4	5	6	7	8	9

Veriable replacer(All are terminal)

i	=	i	+	i	;				
0	1	2	3	4	5	6	7	8	9

```
cout << "Children Counter :" << endl;
for(loop_counter=0; loop_counter<50; loop_counter++){
```

```
cout << children_counter[loop_counter] << endl;
```

Children Count:

6	1	1	1	1	1	1			
0	1	2	3	4	5	6	7	8	9

For A child is -6;
For B child is =1:
For C child is =1;
For D child is =1;
For E child is =1;
For D child is =1:
For F child is =1;
Here,
statement is = [i=i+i;]