



LIMBAJE FORMALE SI TRANSLATOARE

PROIECT LEX + YACC

Name: Fazakas Edina and Ballai Levente

Group: 30233

Teaching assistant: Timotei Molcut



Contents

1	Introducere	3
2	Descriere teoretica	4
2.1	Analizorul lexical (lex.l)	4
2.2	Analizorul sintactic (yacc.y)	5
3	Decizii de implementare	6
4	Exemple de implementare	7
5	Link catre proiectul incarcat pe github	18
5.1	Tool-uri folosite	18
5.2	Link catre proiect	18

Chapter 1

Introducere

Proiectul nostru este o implementare a unui compilator simplu pentru un limbaj de programare definit de noi. Scopul proiectului este de a crea un compilator care poate analiza și interpreta programe scrise în acest limbaj, generând rezultatele corespunzătoare. Pentru a realiza acest proiect am folosit analizorul lexical LEX si analizorul sintactic YACC.

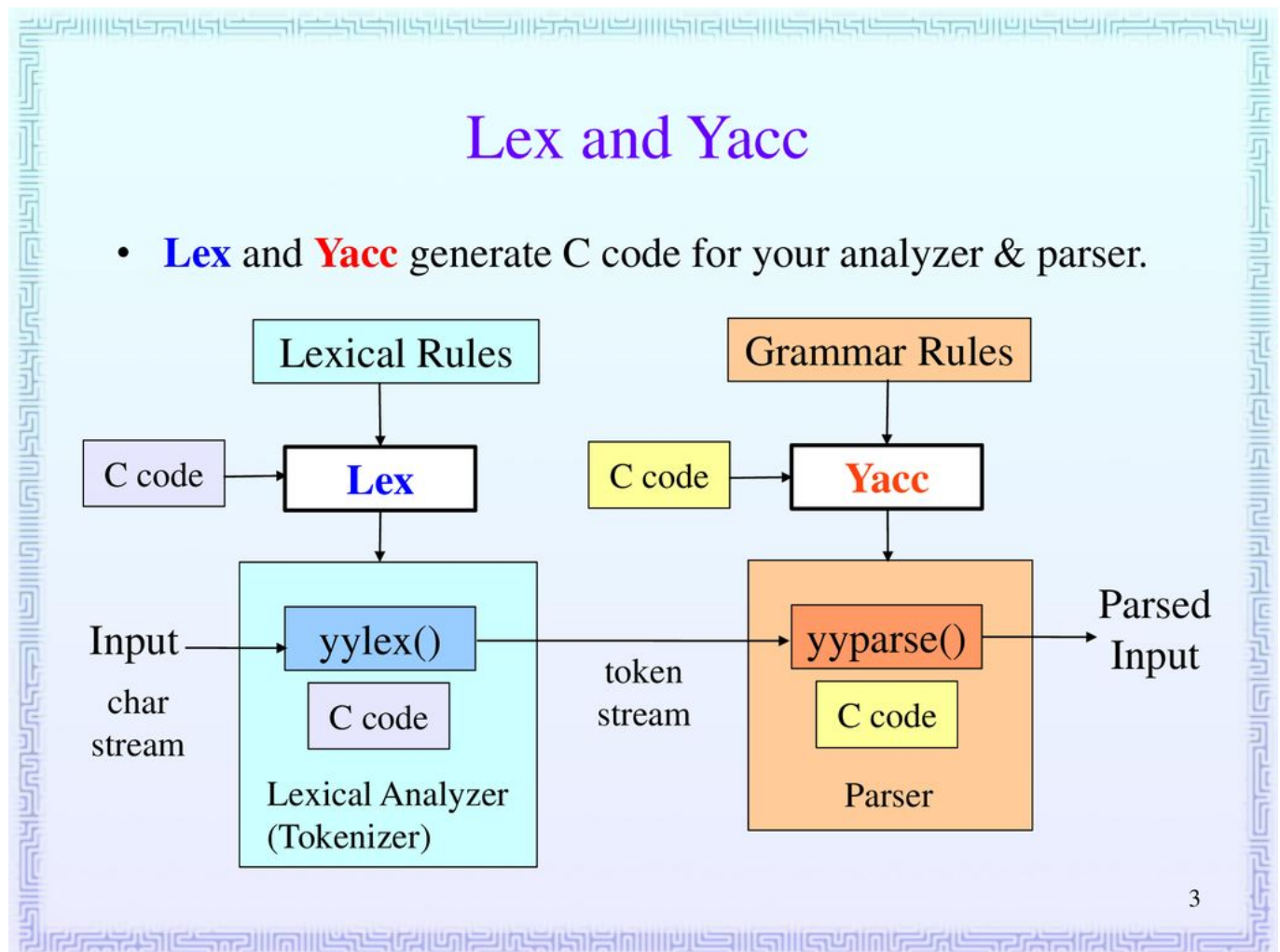


Figure 1.1

Chapter 2

Descriere teoretica

2.1 Analizorul lexical (lex.1)

În secțiunea de declarații, sunt incluse bibliotecile necesare și fișierele de antet pentru tipurile de date și simbolurile utilizate. Secțiunea de reguli definește regulile de analiză lexicală, utilizând expresii regulate și acțiuni asociate. Regulile definește cum sunt recunoscute și clasificate simbolurile din codul sursă, cum ar fi variabilele, numerele întregi, operatorii și cuvintele cheie și stringuri. Limbajul nostru poate parse aceste tipuri de operații:

- WHILE
- DO WHILE
- IF, IF ELSE
- DISPLAY
- FOR
- PLOT
- SAVE
- CMMMC
- CMMDC
- SHOW

Regulile returnează valori și atribuie valori unor variabile pentru utilizarea ulterioară în analizorul sintactic.

2.2 Analizorul sintactic (yacc.y)

În secțiunea de declarații, sunt incluse bibliotecile necesare, fișierele de antet pentru tipurile de date și simbolurile utilizate și sunt declarate variabile globale. Secțiunea de declarații union specifică tipurile de date utilizate în analizorul sintactic. Sunt definite simbolurile terminale și neterminale utilizate în gramatică. Sunt specificate regulile gramaticale ale limbajului de programare și acțiunile asociate acestora. Sunt definite funcții auxiliare pentru crearea și eliberarea nodurilor arborelui sintactic, crearea nodurilor de exemplu. Sunt definite funcții pentru evaluarea și executarea instrucțiunilor și expresiilor din codul sursă. Funcția main inițializează analizorul sintactic și apelează funcția `yyparse()` pentru începerea analizei sintactice.

Instrucțiunile WHILE, DO WHILE au sintaxe similare sintaxei din C și aceleași funcționalități. FOR ca sintaxa arată un pic altfel, având forma: "FOR VARIABLE = init_value TO end_value STEP step_value:". If, și if else sunt implementate la fel ca în C, însă display este un cuvânt cheie care este folosit pentru afișare, având sintaxa "DISPLAY expression;". PLOT este folosit pentru a crea dintr-un FOR un grafic folosind doi vectori care proiectează axa x și y. Acesta are ca valorile de pe axa x valorile contorului din buclă. SAVE are rolul de a initializa valori în vectorul pentru axa y. Ea are sintaxa "SAVE expression;"; și salvează valoarea din expresie în următorul element din vectorul y. CMMMC și CMMDC au rolul de a calcula cel mai mare divizor comun și cel mai mic multiplu comun dintre două numere. Sintaxele lor arată așa: "CMMMC EXPR EXPR;", "CMMDC EXPR EXPR;".

Chapter 3

Decizii de implementare

- In acest proiect avem 2 tipuri de afisari, acestea fiind instructiunile DISPLAY si SHOW. DISPLAY este folosit pentru a afisa expresii, respectiv SHOW pentru a afisa stringurile.
- Bucla FOR are sintaxa "FOR VARIABLE = init_value TO end_value STEP step_value:", ceea ce este o sintaxa neobisnuita fata de alte limbaje de programare. Am ales aceasta varianta de sintaxa pentru ca am vrut sa definim propria noastra bucla for si sa fie cat mai ampla-am adaugat cuvintele cheie TO si STEP.
- SAVE este o instructiune pe care o folosim pentru a salva anumite rezultate din expresii intr-un vector cu elemente intregi. Vectorul acesta este definit global si il folosim pentru a reprezenta valorile de pe axa y din graful creat(plot).
- Folosim o singura litera mica pentru a memora valorile intregi a expresiilor si o singura litera mare pentru a stoca stringuri in ghilimele. Pentru a asigna valori intregi unei variabile folosim operatorul "=", iar pentru asignarea stringurilor cuvantul cheie "IS".
- Am implementat cate o functie pentru cel mai mic multiplu comun(CMMC) si cel mai mare divizor comun(CMMD) pe care le-am folosit intr-un caz de testare pentru crearea graficului.
- Daca un utilizator doreste sa creeze un grafic, trebuie sa foloseasca o bucla FOR prin care salveaza explicit, cu instructiunea SAVE, rezultatele expresiilor care vor reprezenta axa y de pe grafic. Bucla for salveaza implicit contorul intr-un alt vector care va reprezenta axa x de pe grafic.

Chapter 4

Exemple de implementare

WHILE:

Programul de testare, file-ul "WHILE", afiseaza de la 1 la 100 elementele care sunt divizibile cu 5, sau in caz contrar 1:

```
i=1;
j=1000;
while (i<=100) {
    if ((i-(5*(i/5)))==0){
        display i;
    }
    else{
        display j;
    };
    i=i+1;
}
```

Figure 4.1

Rularea programului:

```
edina@edina-VirtualBox:~/Downloads/lft/project$ ./PROJECT <while
1000
1000
1000
1000
1000
5
1000
1000
1000
1000
1000
10
1000
1000
1000
1000
15
1000
1000
1000
1000
20
1000
1000
1000
1000
25
1000
1000
1000
1000
30
1000
1000
1000
1000
35
1000
1000
1000
1000
40
1000
1000
1000
1000
45
1000
1000
1000
1000
```

(a)

```
1000
1000
1000
55
1000
1000
1000
60
1000
1000
1000
65
1000
1000
1000
70
1000
1000
1000
75
1000
1000
1000
80
1000
1000
1000
85
1000
1000
1000
90
1000
1000
1000
95
1000
1000
1000
100
1000
```

(b)

Figure 4.2

DO WHILE:

Acest program face acelasi lucru ca cel anterior cu diferenta ca este bucla do while

```
t > project > while
1  i=1;
2  j=1000;
3  do {
4      if ((i-(5*(i/5)))==0){
5          display i;
6      }
7      else{
8          display j;
9      };
10     i=i+1;
11 }while(i<=100);
12 .
```

Figure 4.3

Rularea programului:

```
edina@edina-VirtualBox:~/Downloads/lft/project$ ./PROJECT <while
1000
1000
1000
1000
5
1000
1000
1000
1000
10
1000
1000
1000
15
1000
1000
1000
20
1000
1000
1000
25
1000
1000
1000
30
1000
1000
1000
35
1000
1000
1000
40
1000
1000
1000
45
1000
1000
1000
50
1000
1000
1000
```

Figure 4.4

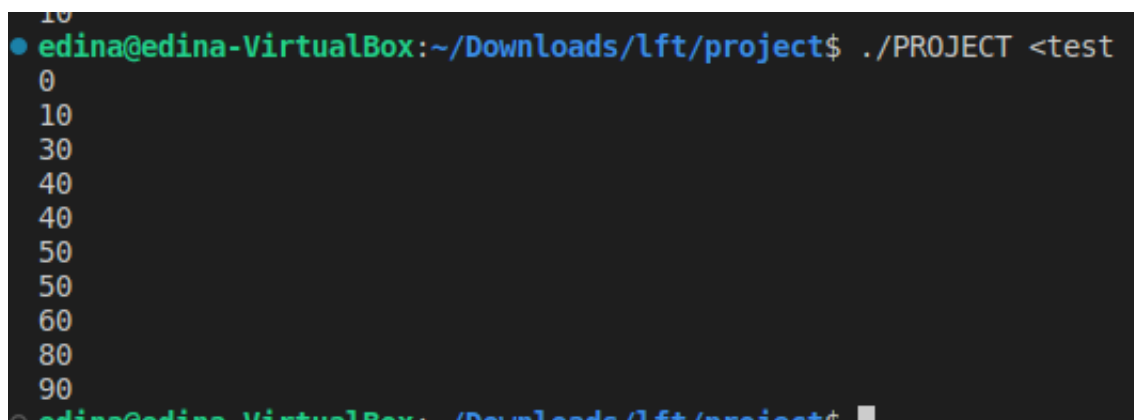
FOR:

Acest program in cazul in care elementul de la 1 la 10 este divizibil cu 2, inmulteste 10 cu j si o afiseaza, daca e divizibil cu 3 o inmulteste cu 20 si o afiseaza, iar in caz contrar doar afiseaza j

```
> project > test
1  j=0;
2  for i=1 to 10 step 1 : {
3      if ((i-(2*(i/2)))==0) {
4          j=j+10;
5          display j;
6      }
7      else if ((i-(3*(i/3)))==0) {
8          j=j+20;
9          display j;
10     }
11     else {
12         display j;
13     };
14 }
15 .
16
```

Figure 4.5

Rularea programului:

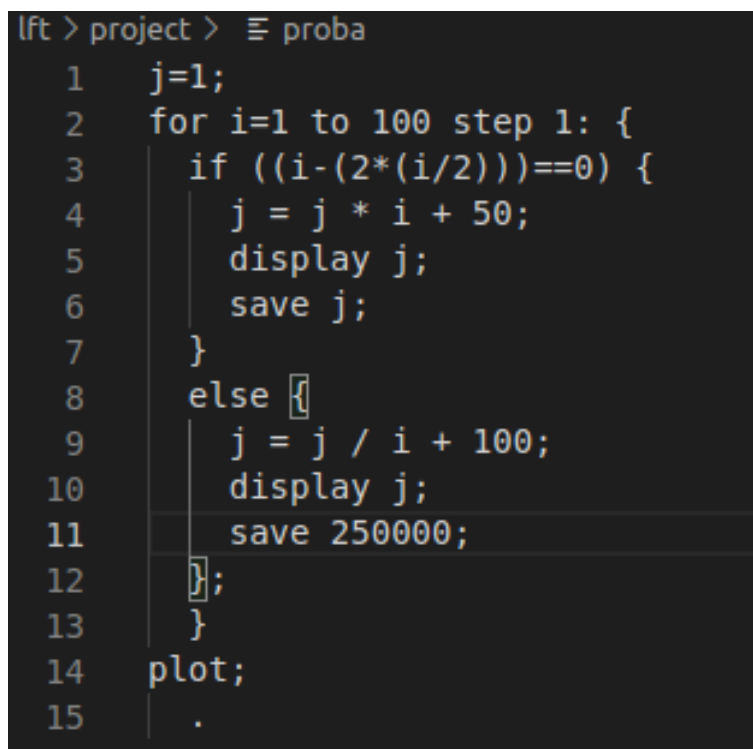


```
edina@edina-VirtualBox:~/Downloads/lft/project$ ./PROJECT <test
10
0
10
30
40
40
50
50
60
80
90
edina@edina-VirtualBox:~/Downloads/lft/project$
```

Figure 4.6

PLOT:

In acest cod se va salva si se va afisa rezultatul unei operatii pe o variabila in cazul in care contorul e par, sau in caz contrar daca e impar. Se vor salva valorile in cele 2 vectori cu care vom crea un grafic cu ajutorul instructiunii plot. Vom putea vizualiza cateva variatii a crearii plotului prin bucla for:



```
lft > project > ≡ proba
1  j=1;
2  for i=1 to 100 step 1: {
3      if ((i-(2*(i/2)))==0) {
4          j = j * i + 50;
5          display j;
6          save j;
7      }
8      else {
9          j = j / i + 100;
10         display j;
11         save 250000;
12     };
13 }
14 plot;
15 .
```

Figure 4.7

Rularea programului:

```
edina@edina-VirtualBox:~/Downloads/lft/project$ ./PROJECT <proba
101
252
184
786
257
1592
327
2666
396
4010
464
5618
532
7498
599
9634
666
12038
733
14710
800
17650
867
20858
934
24334
1001
28078
1068
32090
1135
36370
1202
40918
1269
45734
1336
50818
1403
56170
1470
61790
1536
67634
1602
73742
1668
80114
1734
86750
1800
93650
```

(a)

```
86750
1800
93650
1866
100814
1932
108242
1998
115934
2064
123890
2130
132110
2196
140594
2262
149342
2328
158354
2394
167630
2460
177170
2526
186974
2592
197042
2658
207374
2724
217970
2790
228830
2856
239954
2922
251342
2988
262994
3054
274910
3120
287090
3186
299534
3252
312242
3318
325214
3384
338450
edina@edina-VirtualBox:~/Downloads/lft/project$
```

(b)

Figure 4.8

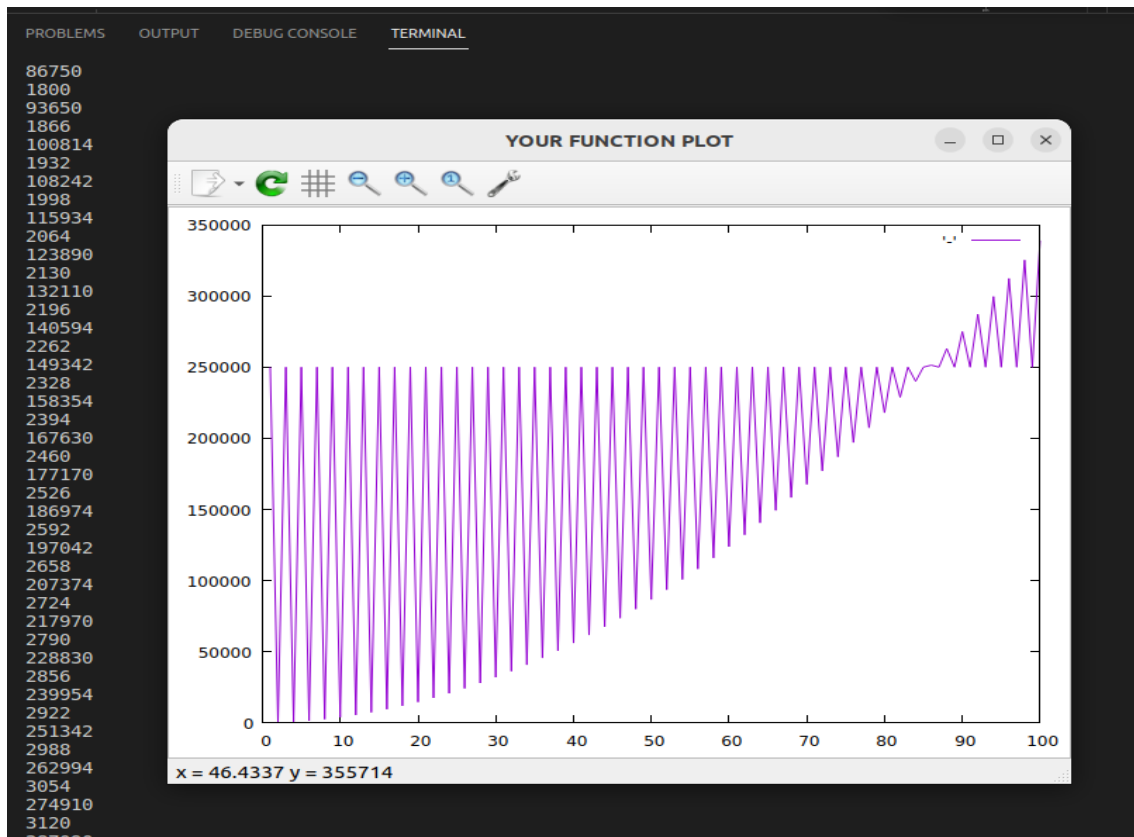


Figure 4.9

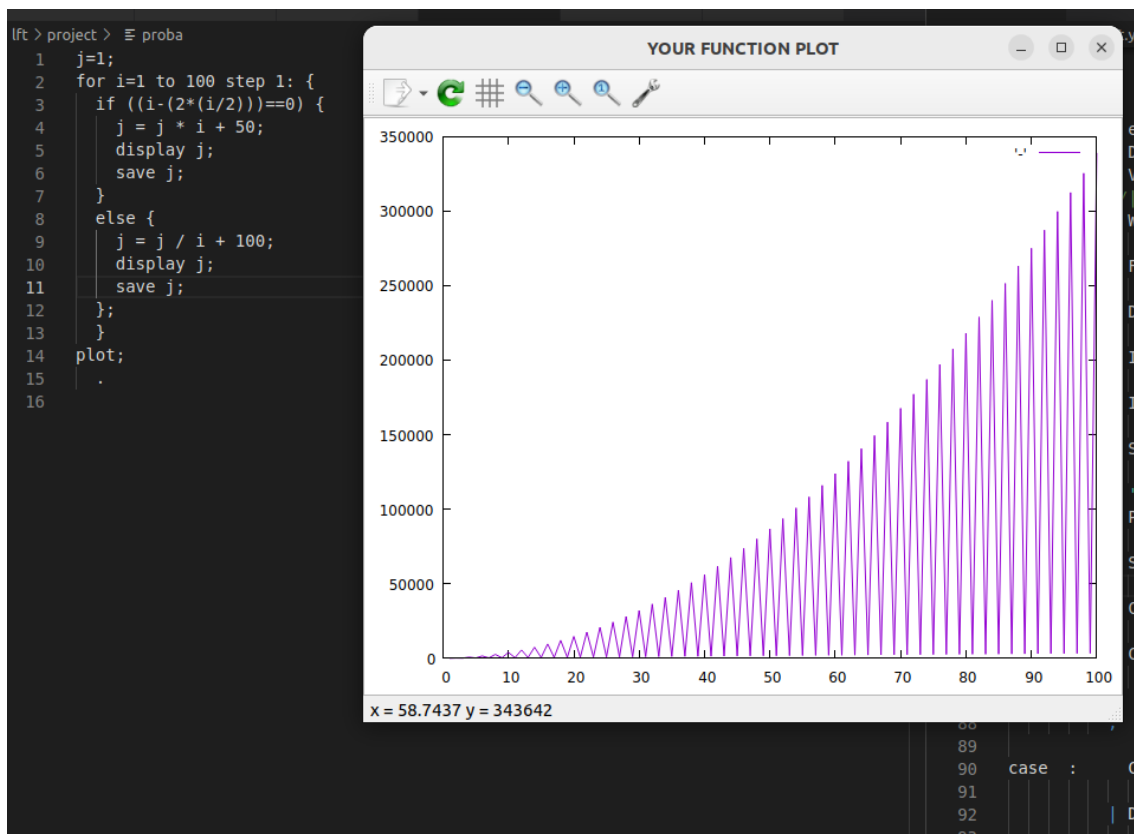


Figure 4.10

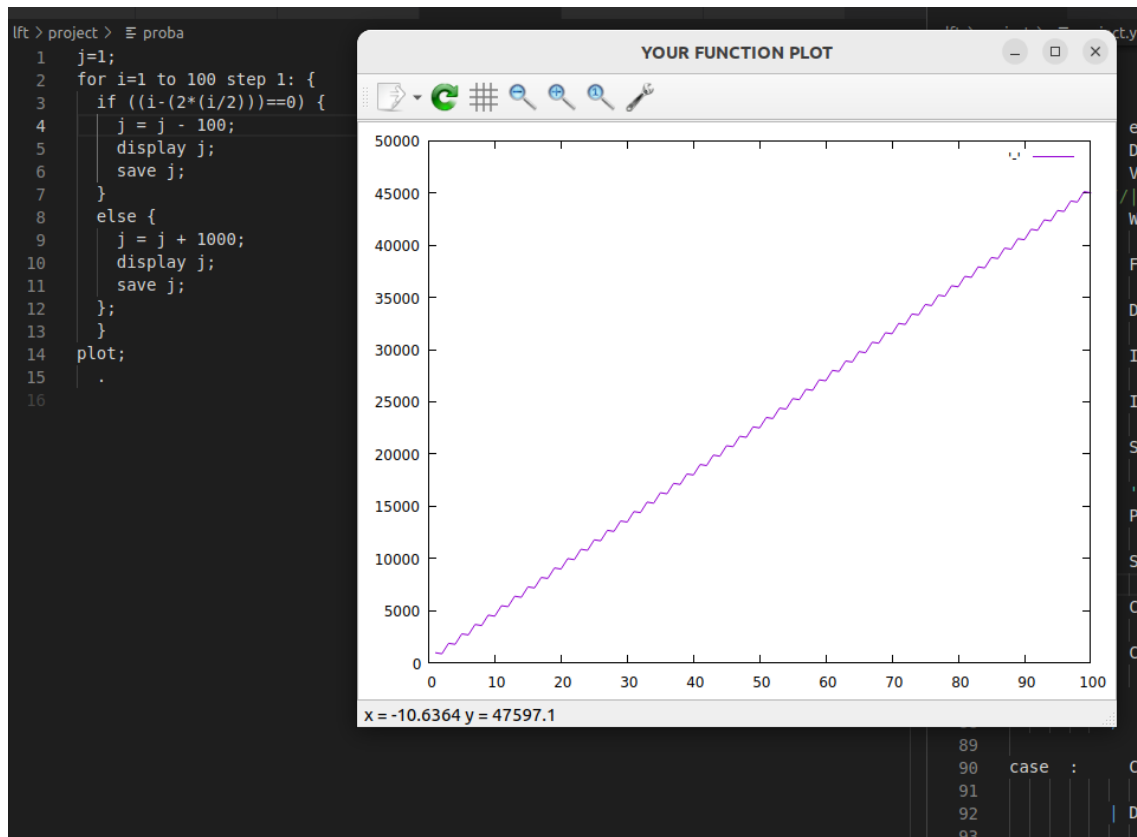


Figure 4.11

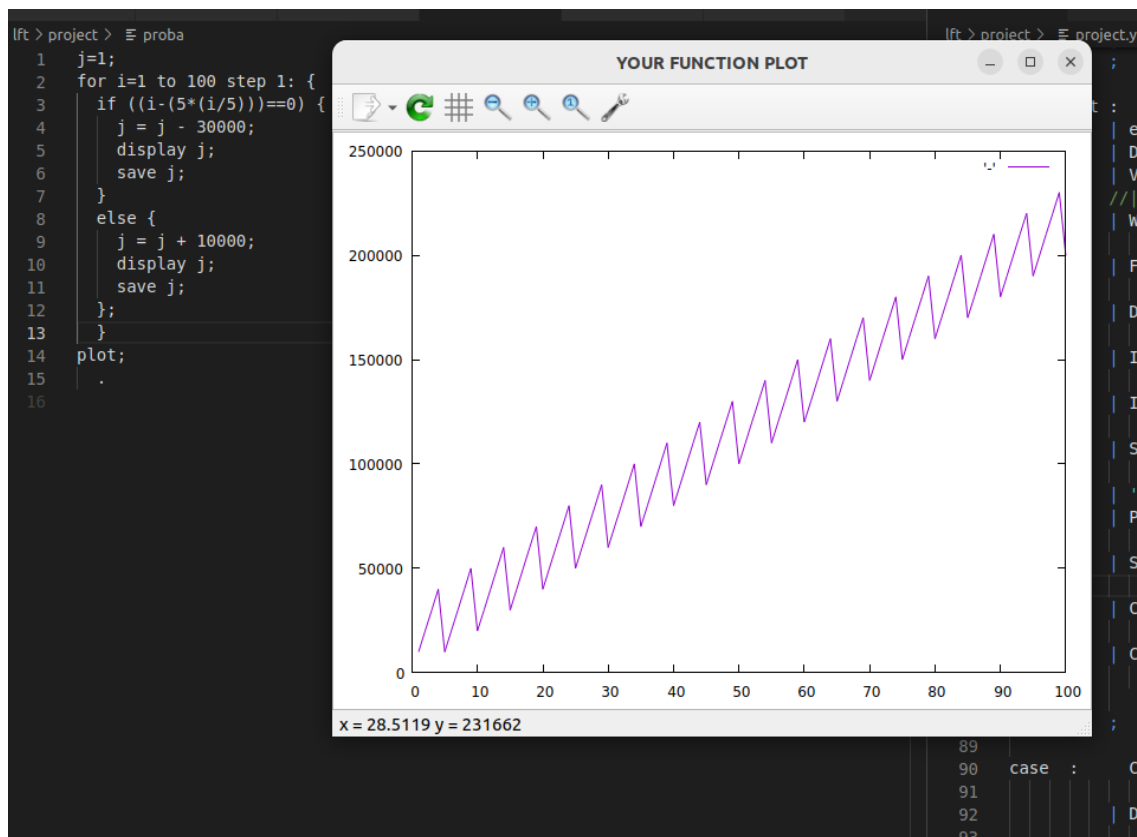
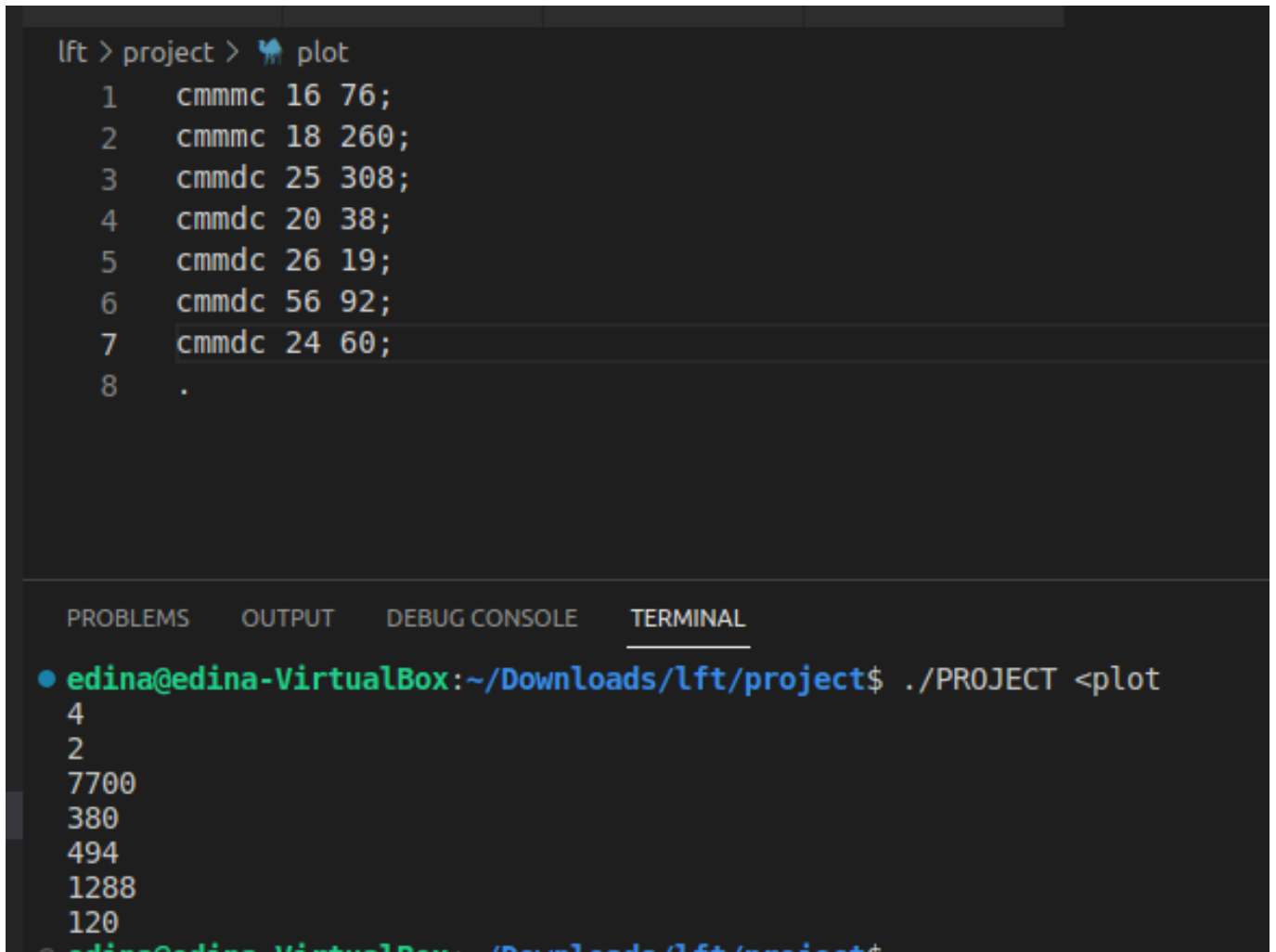


Figure 4.12

CMMMC + CMMDC:

Aici vedem cateva calcule de cmmmc si cmmdc:



The image shows a code editor with a dark theme. The top part of the editor displays a list of commands for a 'plot' operation. The bottom part shows the output of these commands in a terminal window.

```
lft > project > plot
1  cmmmc 16 76;
2  cmmmc 18 260;
3  cmmdc 25 308;
4  cmmdc 20 38;
5  cmmdc 26 19;
6  cmmdc 56 92;
7  cmmdc 24 60;
8  .
```

Below the code editor, a terminal window is open, showing the execution of the commands. The prompt is 'edina@edina-VirtualBox:~/Downloads/lft/project\$' and the command is './PROJECT <plot'. The output is as follows:

```
4
2
7700
380
494
1288
120
```

Figure 4.13

Aici putem vizualiza cateva grafici utilizand ca elemente cmmmc si cmmdc a doua numere dupa conditie

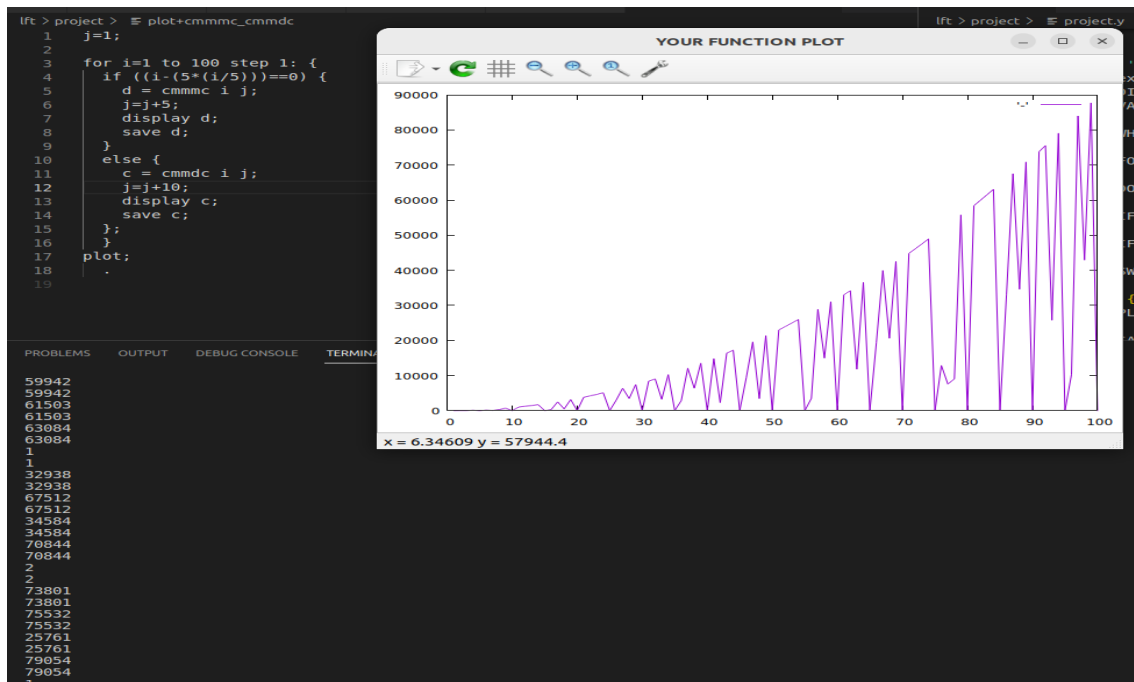


Figure 4.14

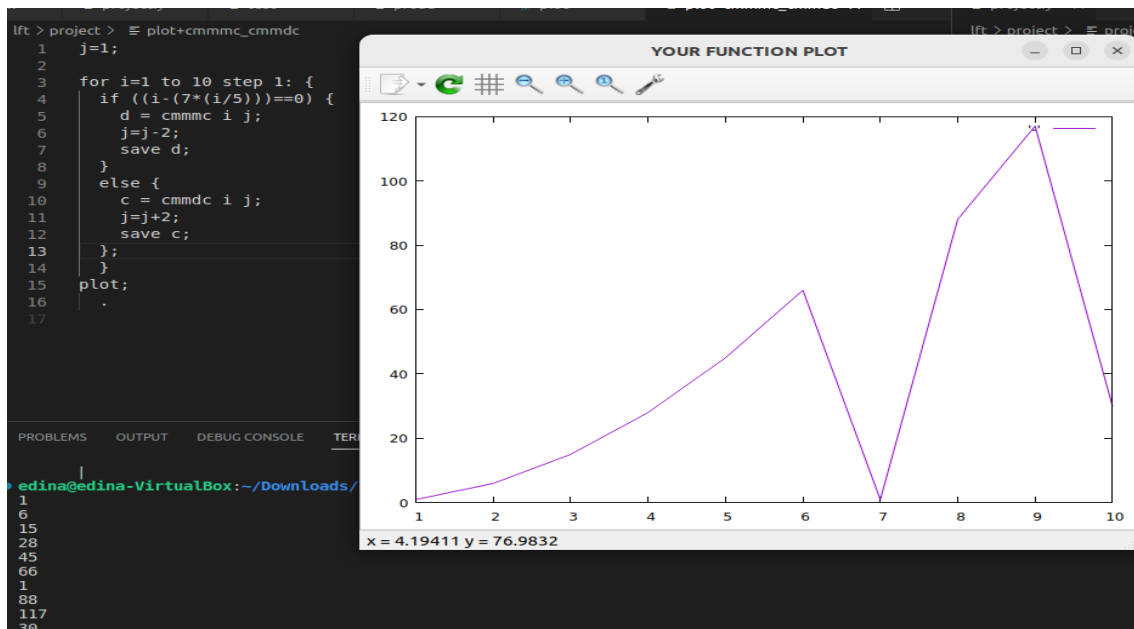


Figure 4.15

STRING:

Afisarea unor string-uri cu sau fara conditie + plot

```
lft > project > string
1 B is "HELLO";
2 A is "asd";
3 show A;
4 show B;
5 Z is "HEY";
6 Y is "ALL";
7 show Z;
8 .

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
● edina@edina-VirtualBox:~/Downloads/lft/project$ ./PROJECT <string
"asd"
"HELLO"
"HEY"
```

Figure 4.16

```
lft > project > string
1 A is "HELLO";
2 B is "HELLO WORLD";
3
4 for i=1 to 10 step 1 : {
5     if ((i-(3*(i/3)))==0) {
6         show A;
7     }
8     else {
9         show B;
10    };
11 }
12 .

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
● edina@edina-VirtualBox:~/Downloads/lft/project$ ./PROJECT <string
"HELLO WORLD"
"HELLO WORLD"
"HELLO"
"HELLO WORLD"
"HELLO WORLD"
"HELLO"
"HELLO WORLD"
"HELLO WORLD"
"HELLO"
"HELLO WORLD"
```

Figure 4.17

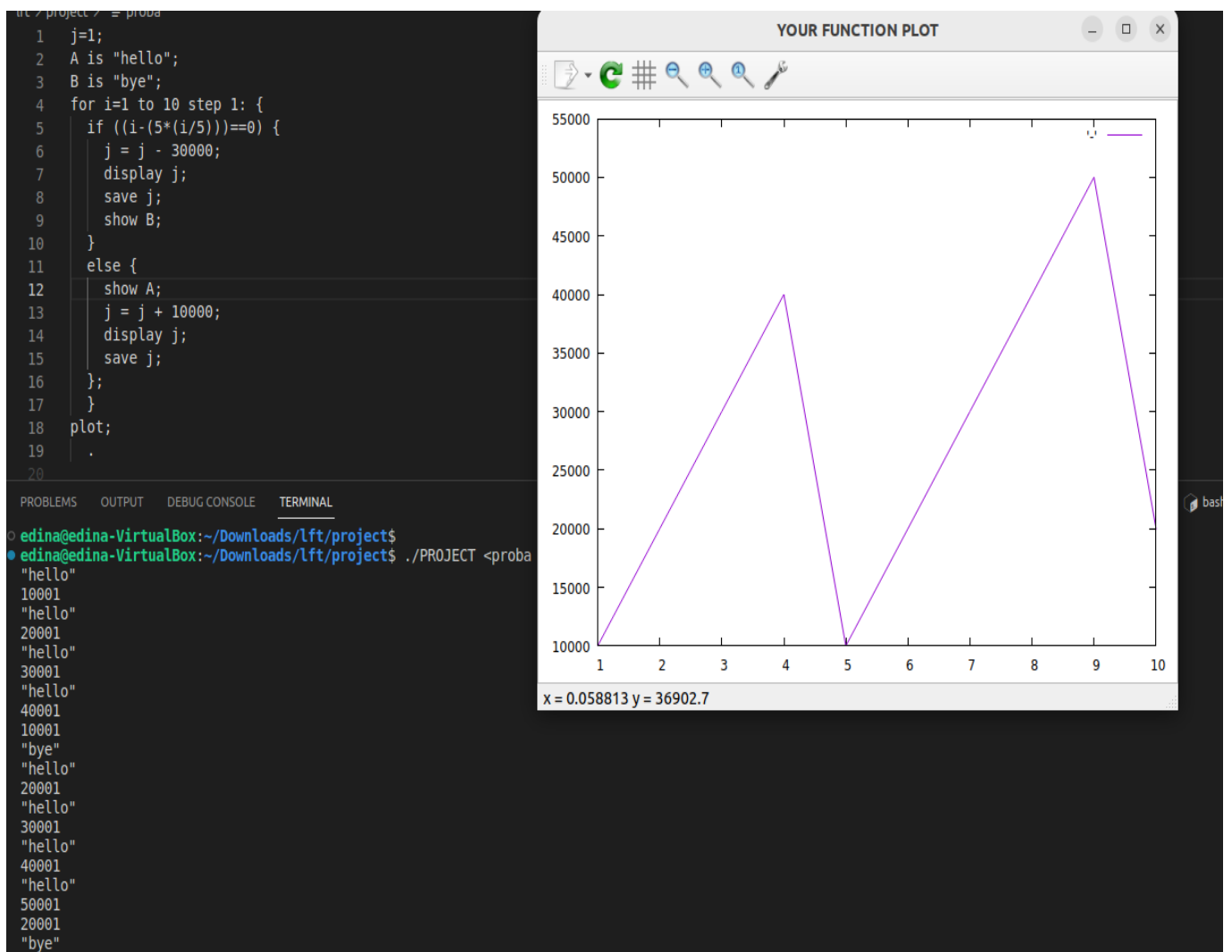


Figure 4.18

Chapter 5

Link catre proiectul incarcat pe github

5.1 Tool-uri folosite

Am folosit pentru parsarea limbajului analizorul lexical LEX si analizorul sintactic YACC. Pe proiect am lucrat in Visual Studio Code pe Linux. Pentru grafici am folosit libraria GNUPlot si extensiile corespunzatoare in VS Code.

5.2 Link catre proiect

LINK TO PROJECT: https://github.com/BallaiLevente/LFTproject_LEX_YACC.git

Bibliography

<https://arcb.csc.ncsu.edu/~mueller/codeopt/codeopt00/yman.pdf>

<https://www.youtube.com/watch?v=54bo1qaHAfk>

https://www.youtube.com/watch?v=_wUHG2rfM

<https://www.geeksforgeeks.org/flex-fast-lexical-analyzer-generator/> <https://www.geeksforgeeks.org/yacc-to-yacc/>

