Ain Shams University
Faculty of Engineering
Computer and Systems Engineering Department

CSE 422: Systems Software
4th Year CSE
1st Semester 2021/2022
Mini Project 2
Parser

Submitted to:

Dr.Sahar Hagag

Drive link:

https://drive.google.com/drive/folders/1jFQW5MH4ODvJTrfQYrVnYfIpMWnWPS5W?usp=sharing

Submitted by:

Name	ID
Ahmed Mohamed Ahmed Abd El-Hamid	1700157
Ahmed Mohamed Abd El-Hakim	1700176
Aya Sameh	1700342
Bassant Yasser sultan	1700360
Basmala Magdy Ali Abu El Nasr	1700363
Sarah Mohamed Ahmed	1700593
Nourhan Ashraf El Sayed Ahmed	1701604

Table of Contents

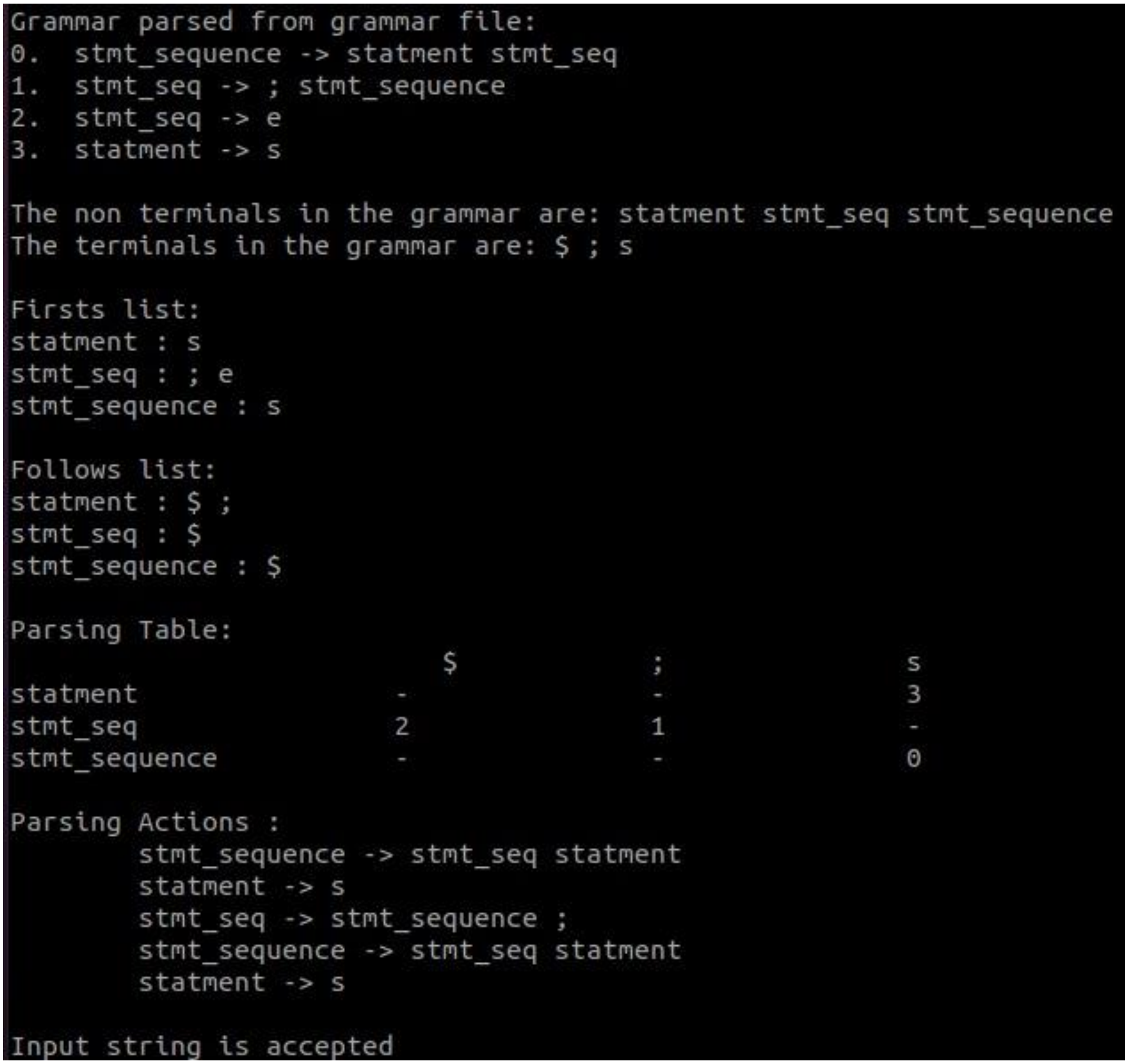
Parser

Bonus features:

Error handling:

Detect the error and it’s type.

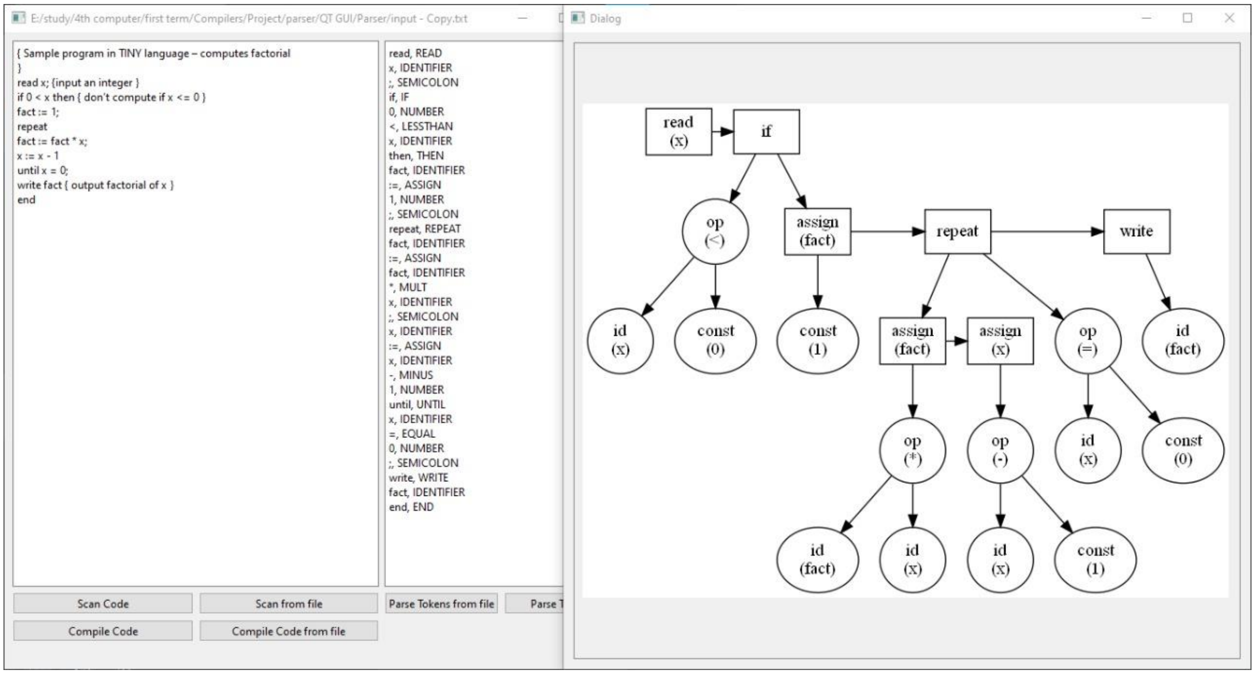
LL(1) Parser



```
Grammar parsed from grammar file:
0.  stmt_sequence -> statment stmt_seq
1.  stmt_seq -> ; stmt_sequence
2.  stmt_seq -> e
3.  statment -> s

The non terminals in the grammar are: statment stmt_seq stmt_sequence
The terminals in the grammar are: $ ; s

Firsts list:
statment : s
stmt_seq : ; e
stmt_sequence : s

Follows list:
statment : $ ;
stmt_seq : $
stmt_sequence : $

Parsing Table:
                           $            ;               s
statment                -               -               3
stmt_seq                2               1               -
stmt_sequence           -               -               0

Parsing Actions :
        stmt_sequence -> stmt_seq statment
        statment -> s
        stmt_seq -> stmt_sequence ;
        stmt_sequence -> stmt_seq statment
        statment -> s

Input string is accepted
```

Test cases:

Parser:

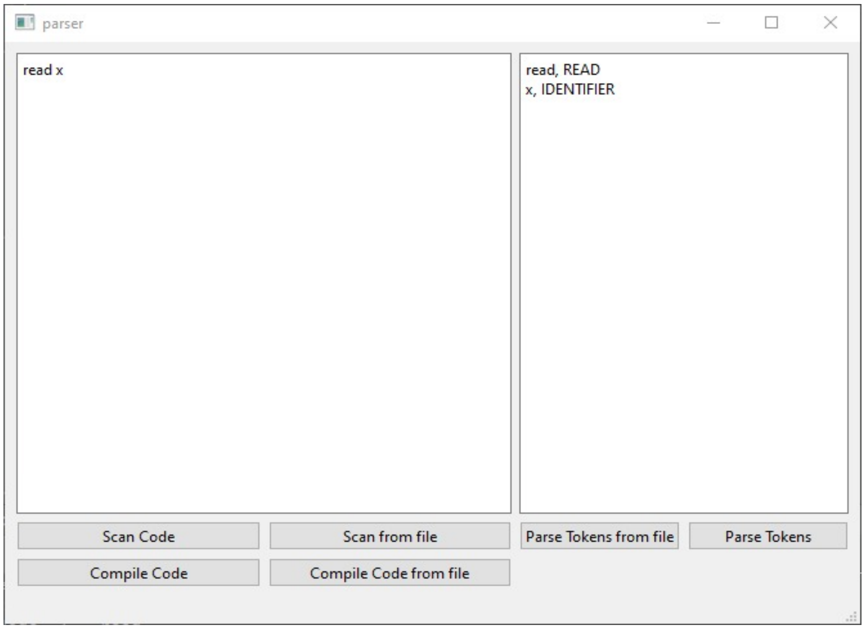


E:/study/4th computer/first term/Compilers/Project/parser/QT GUI/Parser/input - Copy.txt
Dialog

```
{ Sample program in TINY language – computes factorial
}
read x; {input an integer }
if 0 < x then { don't compute if x <= 0 }
fact := 1;
repeat
fact := fact * x;
x := x - 1
until x = 0;
write fact { output factorial of x }
end
```

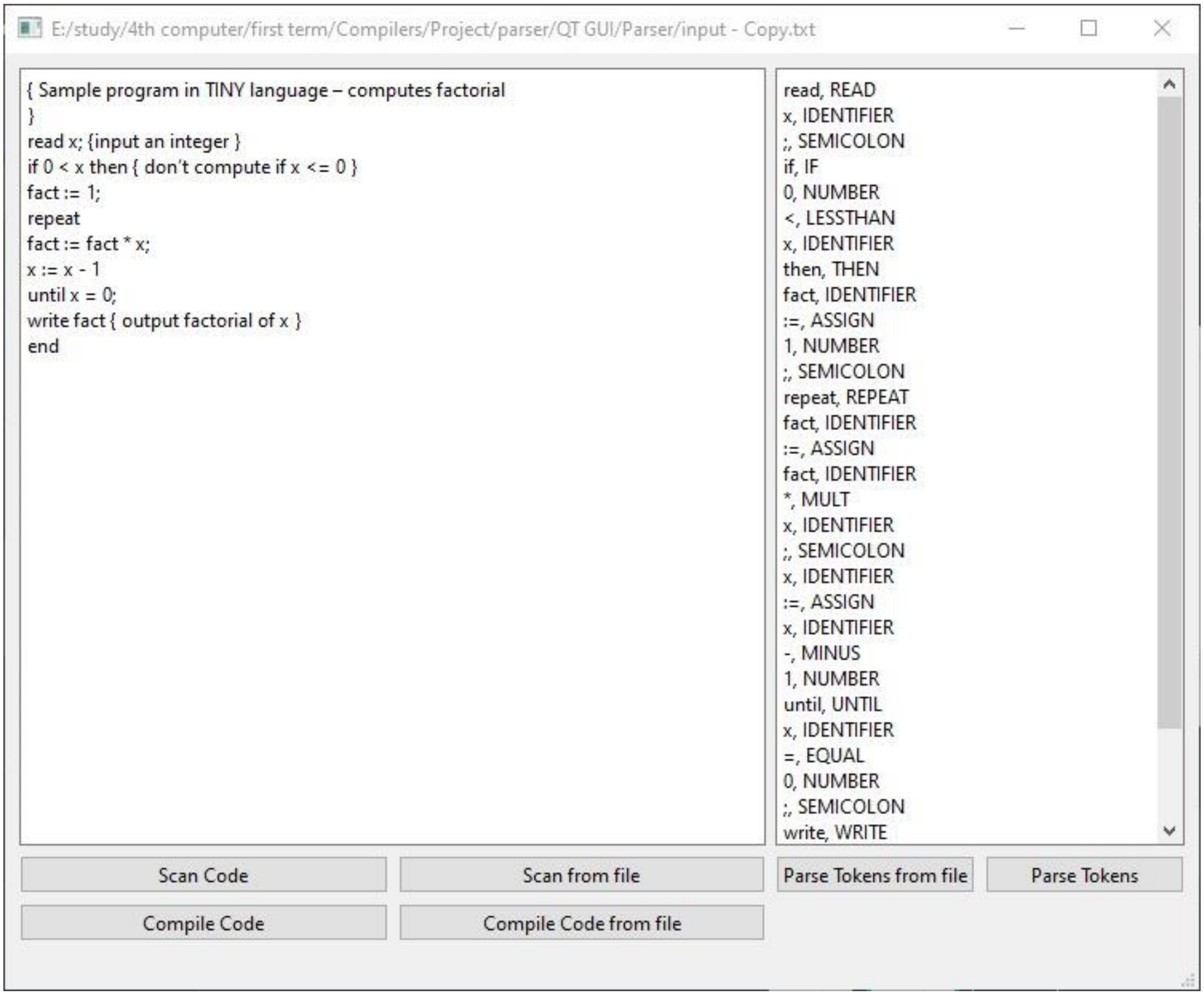
```
read, READ
x, IDENTIFIER
;, SEMICOLON
if, IF
0, NUMBER
<, LESSTHAN
x, IDENTIFIER
then, THEN
fact, IDENTIFIER
:=, ASSIGN
1, NUMBER
;, SEMICOLON
repeat, REPEAT
fact, IDENTIFIER
:=, ASSIGN
fact, IDENTIFIER
*, MULT
x, IDENTIFIER
;, SEMICOLON
x, IDENTIFIER
:=, ASSIGN
x, IDENTIFIER
-, MINUS
1, NUMBER
until, UNTIL
x, IDENTIFIER
=, EQUAL
0, NUMBER
;, SEMICOLON
write, WRITE
fact, IDENTIFIER
end, END
```

Scanner:



```
read x
```

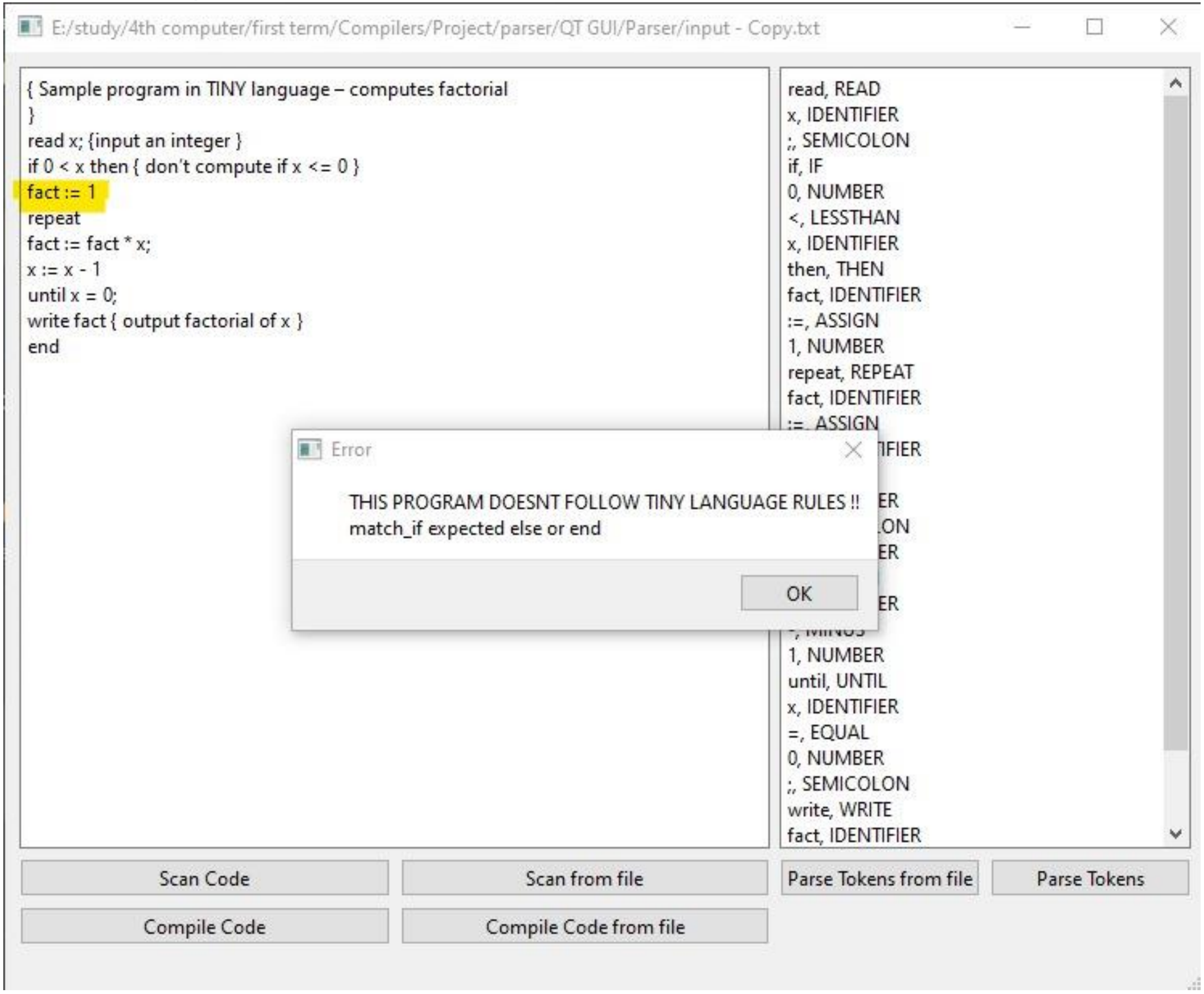
```
read, READ
x, IDENTIFIER
```



```
{ Sample program in TINY language – computes factorial
}
read x; {input an integer }
if 0 < x then { don't compute if x <= 0 }
fact := 1;
repeat
fact := fact * x;
x := x - 1
until x = 0;
write fact { output factorial of x }
end
```

```
read, READ
x, IDENTIFIER
;, SEMICOLON
if, IF
0, NUMBER
<, LESSTHAN
x, IDENTIFIER
then, THEN
fact, IDENTIFIER
:=, ASSIGN
1, NUMBER
;, SEMICOLON
repeat, REPEAT
fact, IDENTIFIER
:=, ASSIGN
fact, IDENTIFIER
*, MULT
x, IDENTIFIER
;, SEMICOLON
x, IDENTIFIER
:=, ASSIGN
x, IDENTIFIER
-, MINUS
1, NUMBER
until, UNTIL
x, IDENTIFIER
=, EQUAL
0, NUMBER
;, SEMICOLON
write, WRITE
```

Error handling:



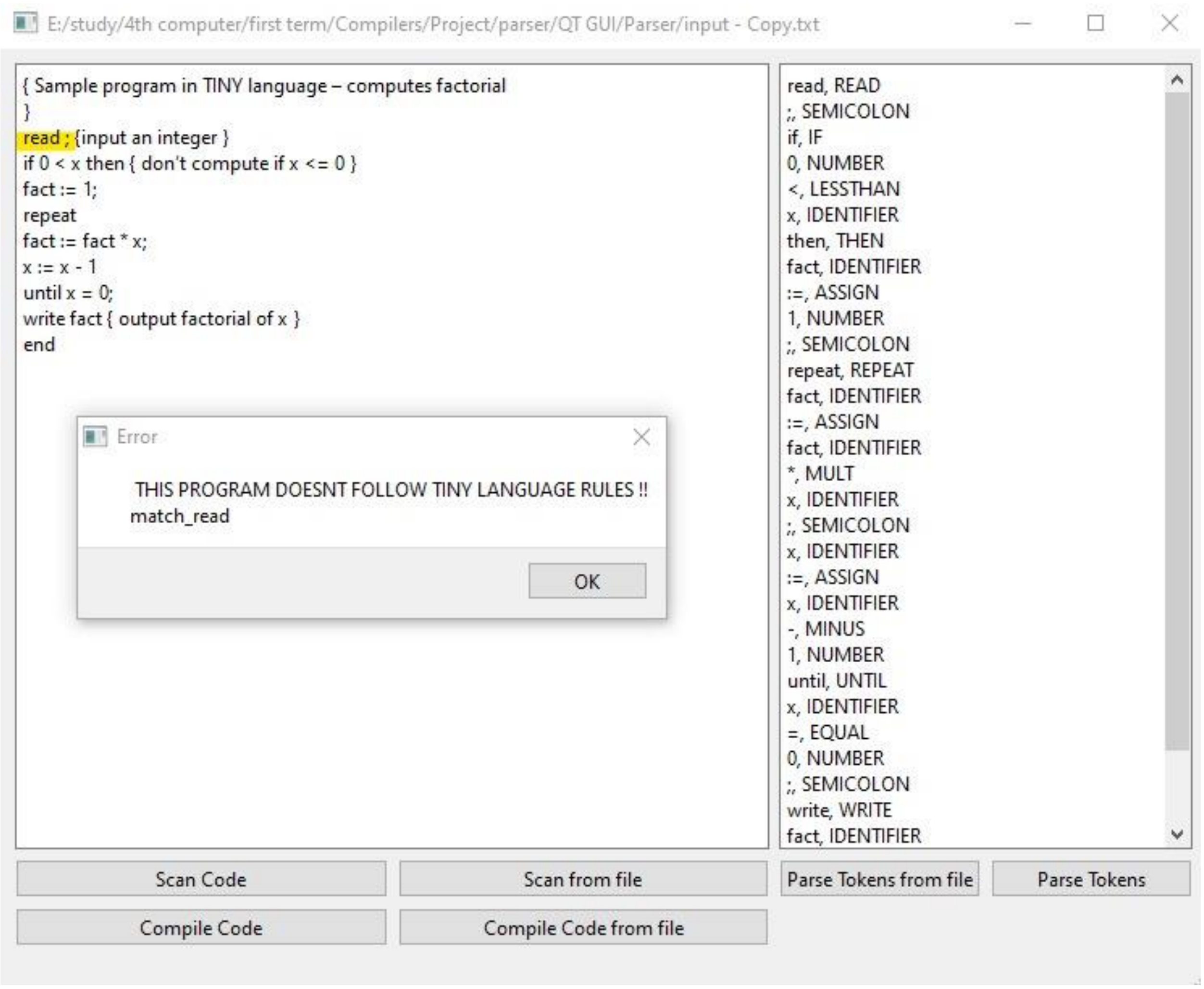
E:/study/4th computer/first term/Compilers/Project/parser/QT GUI/Parser/input - Copy.txt

```
{ Sample program in TINY language – computes factorial
}
read x; {input an integer }
if 0 < x then { don't compute if x <= 0 }
fact := 1
repeat
fact := fact * x;
x := x - 1
until x = 0;
write fact { output factorial of x }
end
```

```
read, READ
x, IDENTIFIER
;, SEMICOLON
if, IF
0, NUMBER
<, LESSTHAN
x, IDENTIFIER
then, THEN
fact, IDENTIFIER
:=, ASSIGN
1, NUMBER
repeat, REPEAT
fact, IDENTIFIER
:=, ASSIGN
          IFIER

          ER
          ON
          ER

          ER
-, MINUS
1, NUMBER
until, UNTIL
x, IDENTIFIER
=, EQUAL
0, NUMBER
;, SEMICOLON
write, WRITE
fact, IDENTIFIER
```

Error

THIS PROGRAM DOESNT FOLLOW TINY LANGUAGE RULES !!
match_if expected else or end



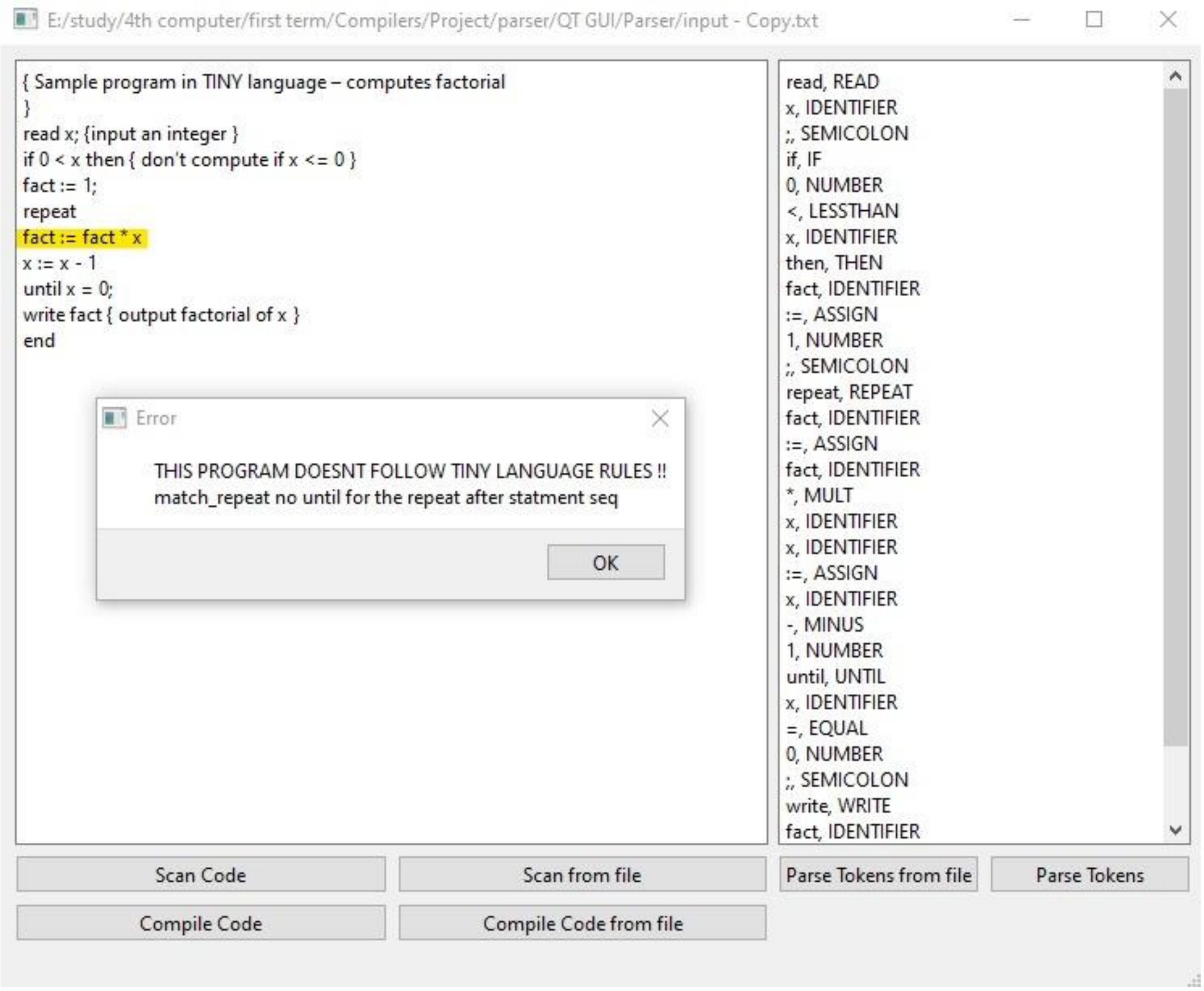
E:/study/4th computer/first term/Compilers/Project/parser/QT GUI/Parser/input - Copy.txt

```
{ Sample program in TINY language – computes factorial
}
read ; {input an integer }
if 0 < x then { don't compute if x <= 0 }
fact := 1;
repeat
fact := fact * x;
x := x - 1
until x = 0;
write fact { output factorial of x }
end
```

```
read, READ
;, SEMICOLON
if, IF
0, NUMBER
<, LESSTHAN
x, IDENTIFIER
then, THEN
fact, IDENTIFIER
:=, ASSIGN
1, NUMBER
;, SEMICOLON
repeat, REPEAT
fact, IDENTIFIER
:=, ASSIGN
fact, IDENTIFIER
*, MULT
x, IDENTIFIER
;, SEMICOLON
x, IDENTIFIER
:=, ASSIGN
x, IDENTIFIER
-, MINUS
1, NUMBER
until, UNTIL
x, IDENTIFIER
=, EQUAL
0, NUMBER
;, SEMICOLON
write, WRITE
fact, IDENTIFIER
```

Error

THIS PROGRAM DOESNT FOLLOW TINY LANGUAGE RULES !!
match_read



```
{ Sample program in TINY language – computes factorial
}
read x; {input an integer }
if 0 < x then { don't compute if x <= 0 }
fact := 1;
repeat
fact := fact * x
x := x - 1
until x = 0;
write fact { output factorial of x }
end
```

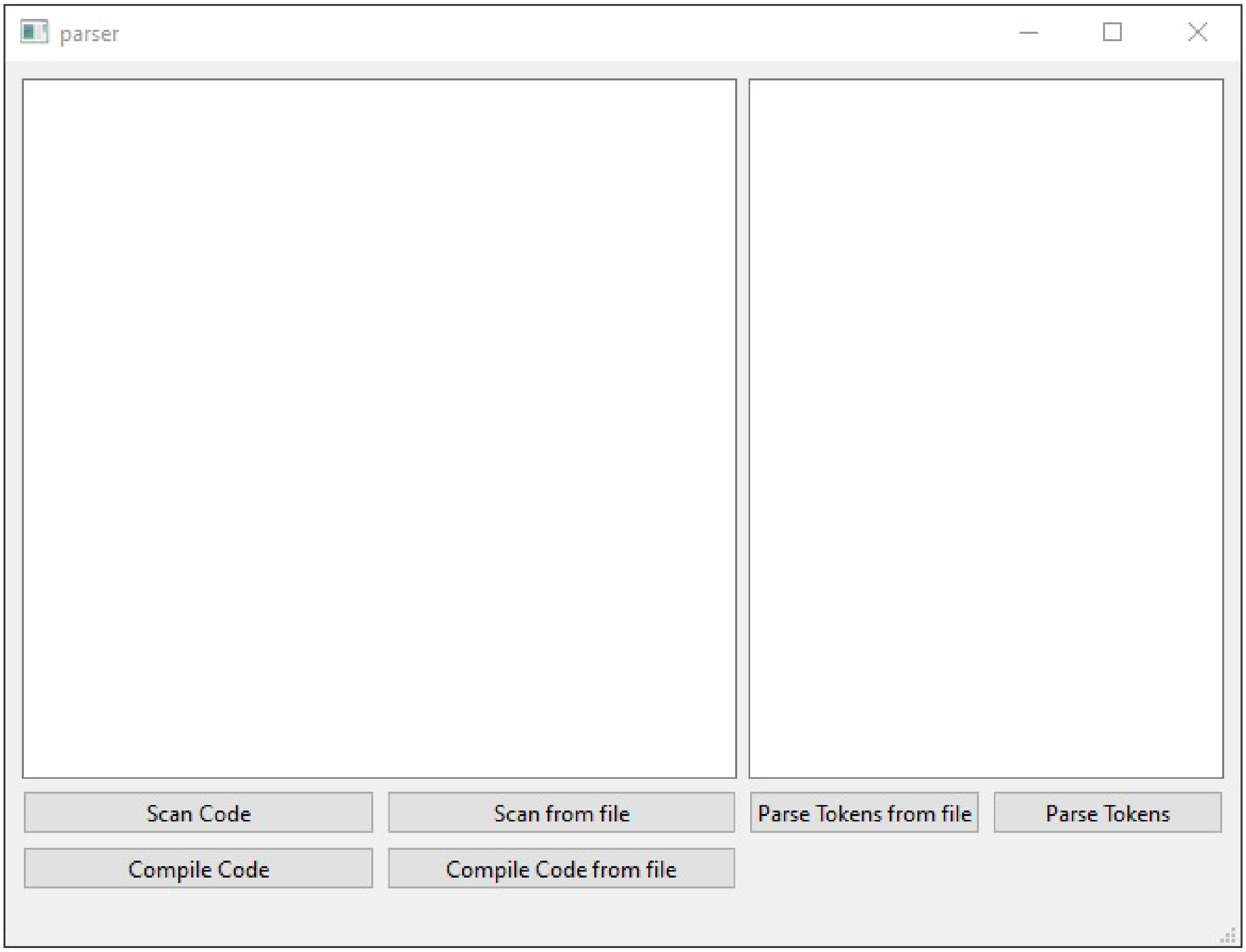
THIS PROGRAM DOESNT FOLLOW TINY LANGUAGE RULES !!
match_repeat no until for the repeat after statment seq

```
read, READ
x, IDENTIFIER
;, SEMICOLON
if, IF
0, NUMBER
<, LESSTHAN
x, IDENTIFIER
then, THEN
fact, IDENTIFIER
:=, ASSIGN
1, NUMBER
;, SEMICOLON
repeat, REPEAT
fact, IDENTIFIER
:=, ASSIGN
fact, IDENTIFIER
*, MULT
x, IDENTIFIER
x, IDENTIFIER
:=, ASSIGN
x, IDENTIFIER
-, MINUS
1, NUMBER
until, UNTIL
x, IDENTIFIER
=, EQUAL
0, NUMBER
;, SEMICOLON
write, WRITE
fact, IDENTIFIER
```

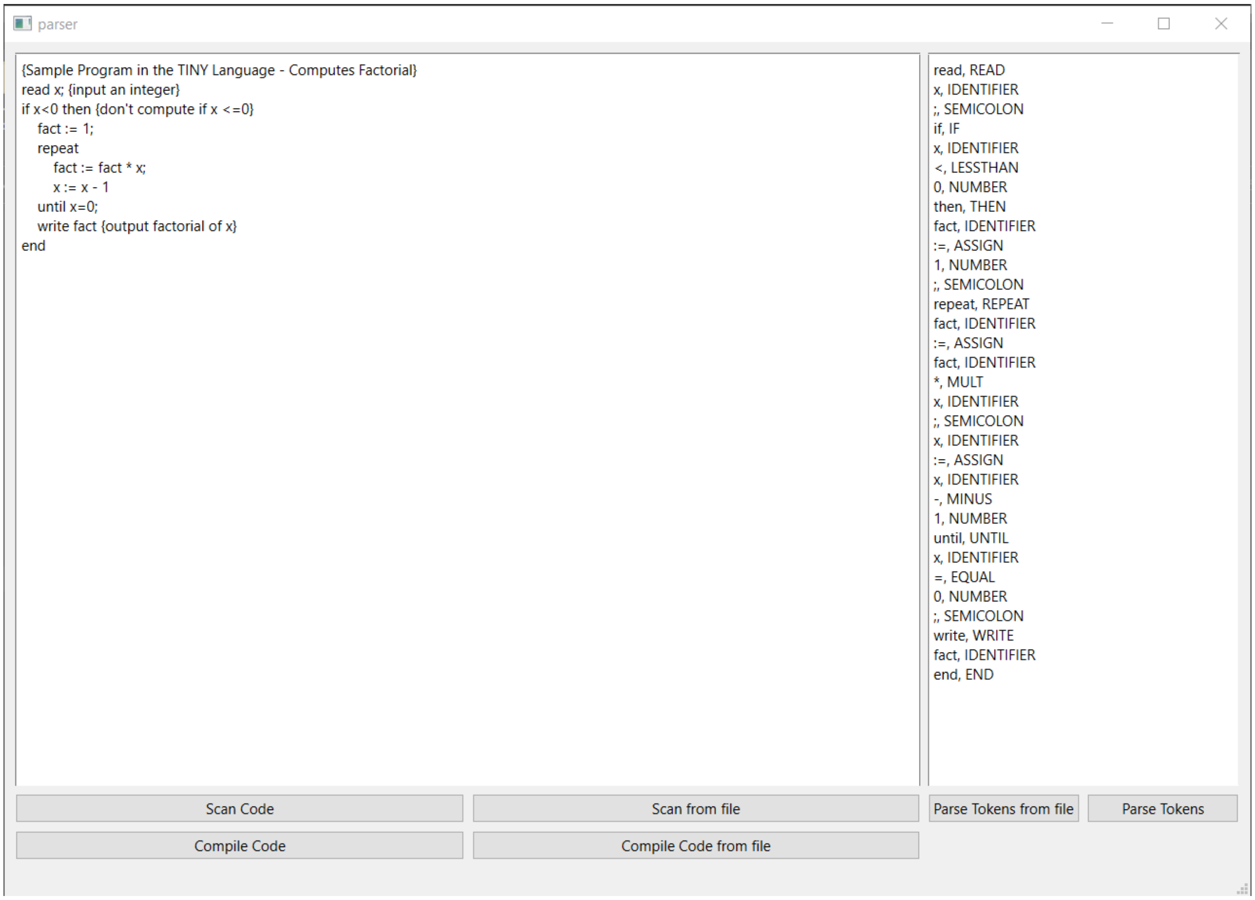
GUI

We built a GUI that takes as input TINY language code as input text in the left textbox or in a “.txt” file.

This GUI uses our scanner and parser projects to output the Syntax Tree directly from the code or from the tokens as input text in the right textbox.



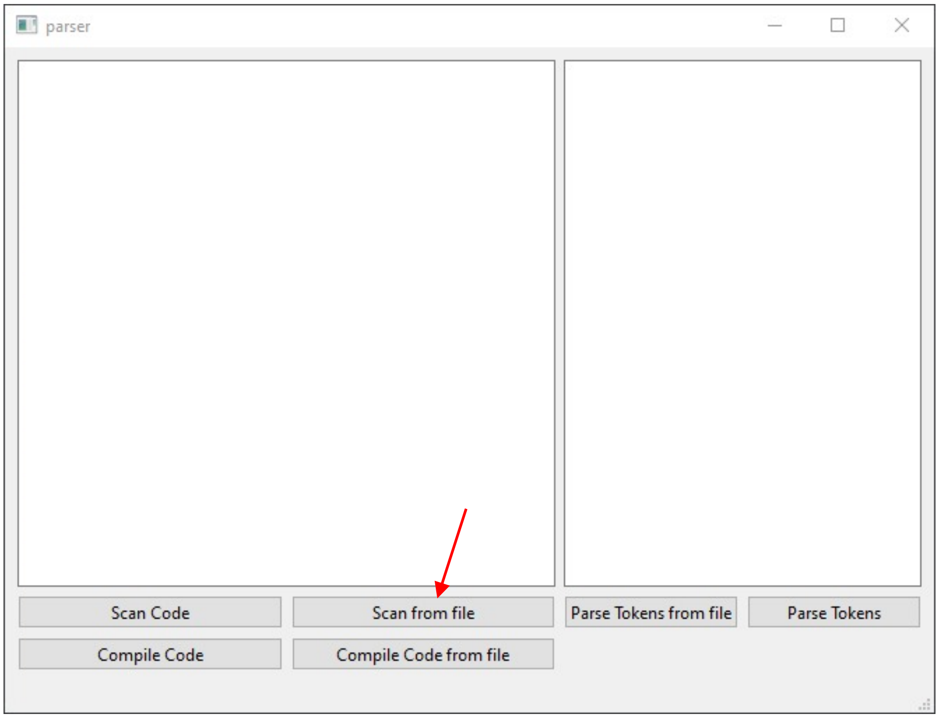
Scan from textbox:

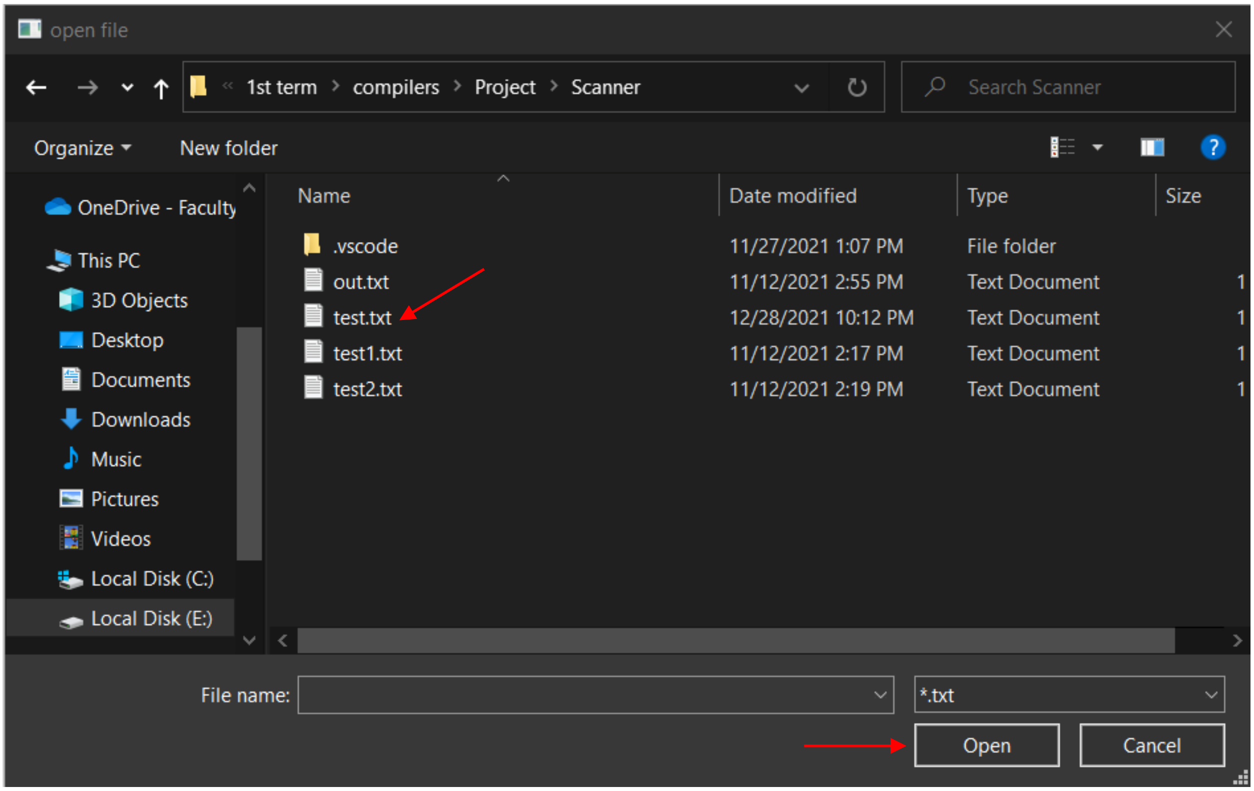


```
{Sample Program in the TINY Language - Computes Factorial}
read x; {input an integer}
if x<0 then {don't compute if x <=0}
   fact := 1;
   repeat
      fact := fact * x;
      x := x - 1
   until x=0;
   write fact {output factorial of x}
end
```

```
read, READ
x, IDENTIFIER
;, SEMICOLON
if, IF
x, IDENTIFIER
<, LESSTHAN
0, NUMBER
then, THEN
fact, IDENTIFIER
:=, ASSIGN
1, NUMBER
;, SEMICOLON
repeat, REPEAT
fact, IDENTIFIER
:=, ASSIGN
fact, IDENTIFIER
*, MULT
x, IDENTIFIER
;, SEMICOLON
x, IDENTIFIER
:=, ASSIGN
x, IDENTIFIER
-, MINUS
1, NUMBER
until, UNTIL
x, IDENTIFIER
=, EQUAL
0, NUMBER
;, SEMICOLON
write, WRITE
fact, IDENTIFIER
end, END
```

Scan from file:

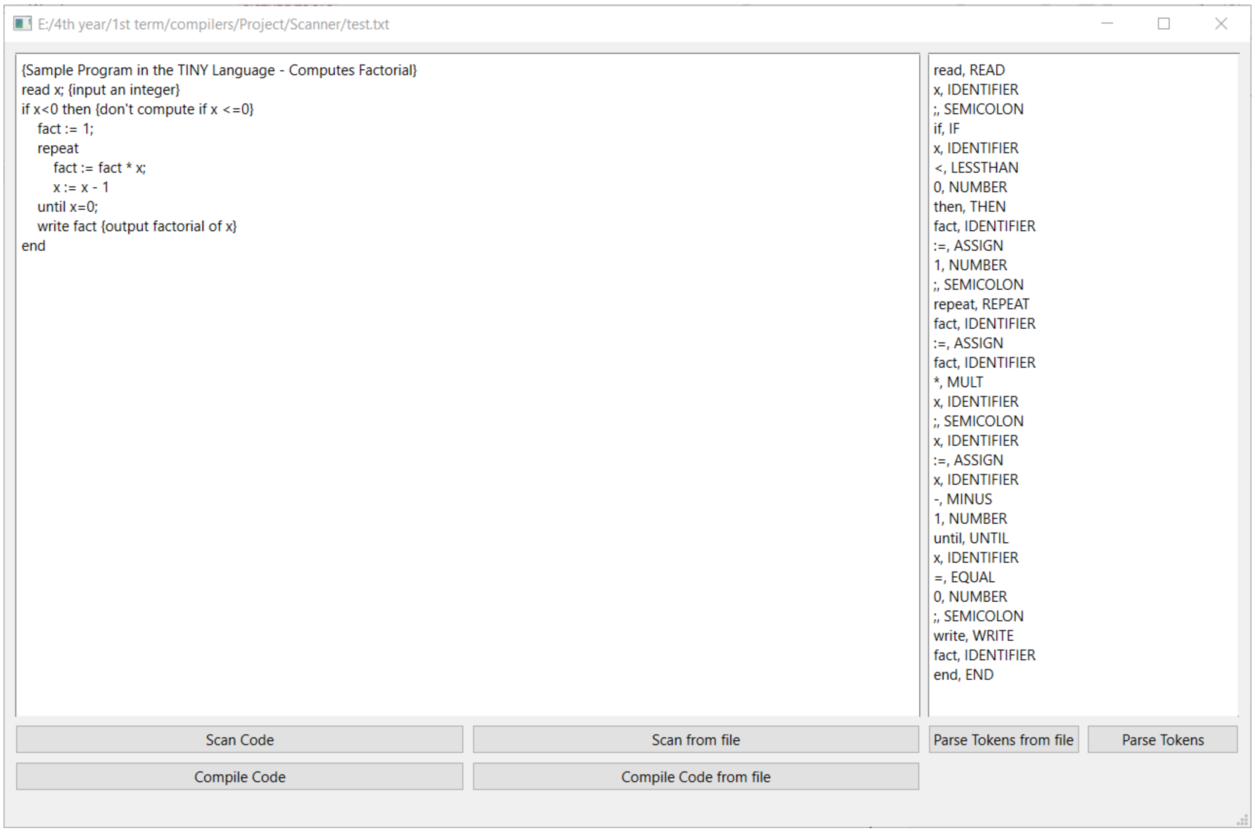




Name	Date modified	Type	Size
.vscode	11/27/2021 1:07 PM	File folder	
out.txt	11/12/2021 2:55 PM	Text Document	1
test.txt	12/28/2021 10:12 PM	Text Document	1
test1.txt	11/12/2021 2:17 PM	Text Document	1
test2.txt	11/12/2021 2:19 PM	Text Document	1

File name:

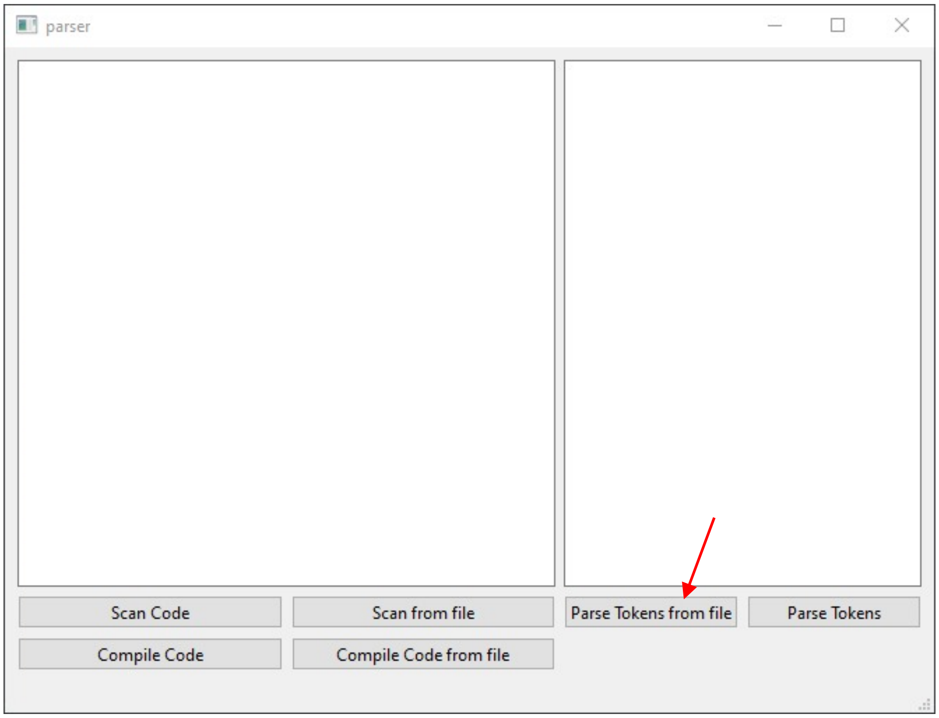
*.txt

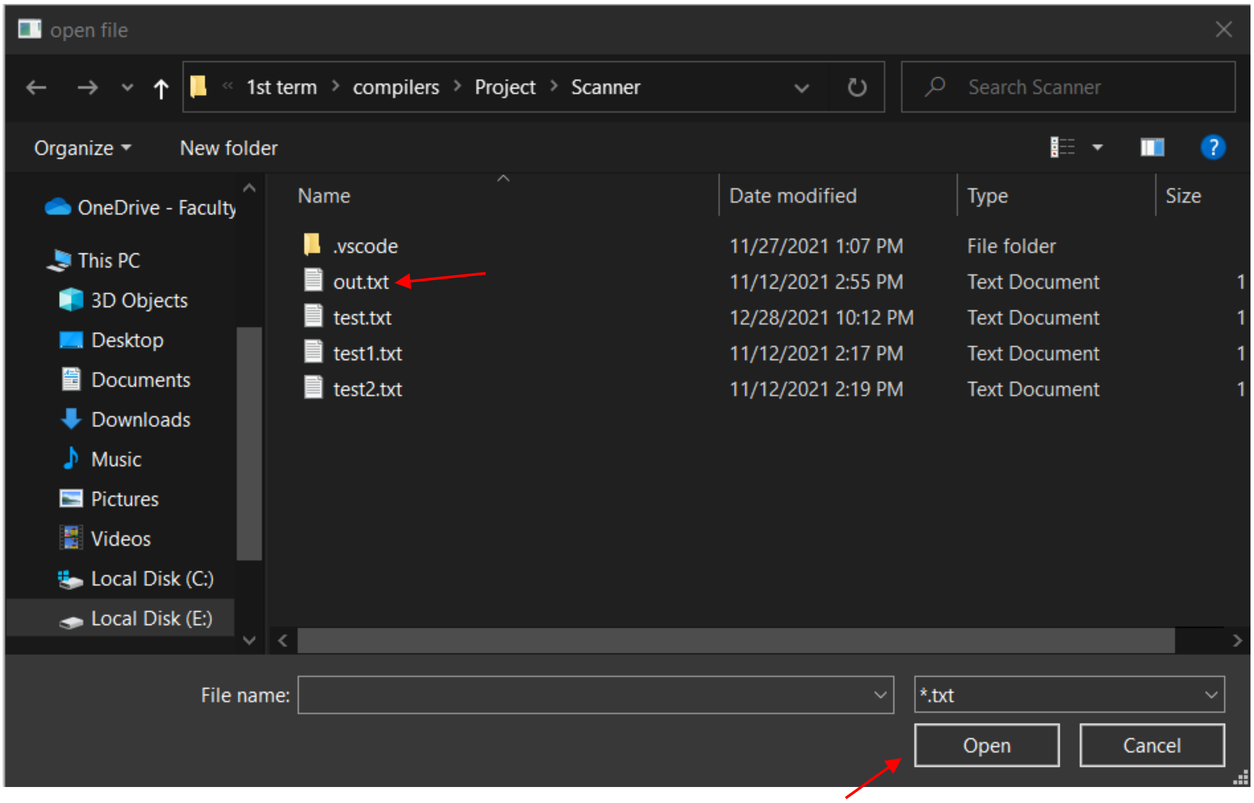


```
{Sample Program in the TINY Language - Computes Factorial}
read x; {input an integer}
if x<0 then {don't compute if x <=0}
   fact := 1;
   repeat
      fact := fact * x;
      x := x - 1
   until x=0;
   write fact {output factorial of x}
end
```

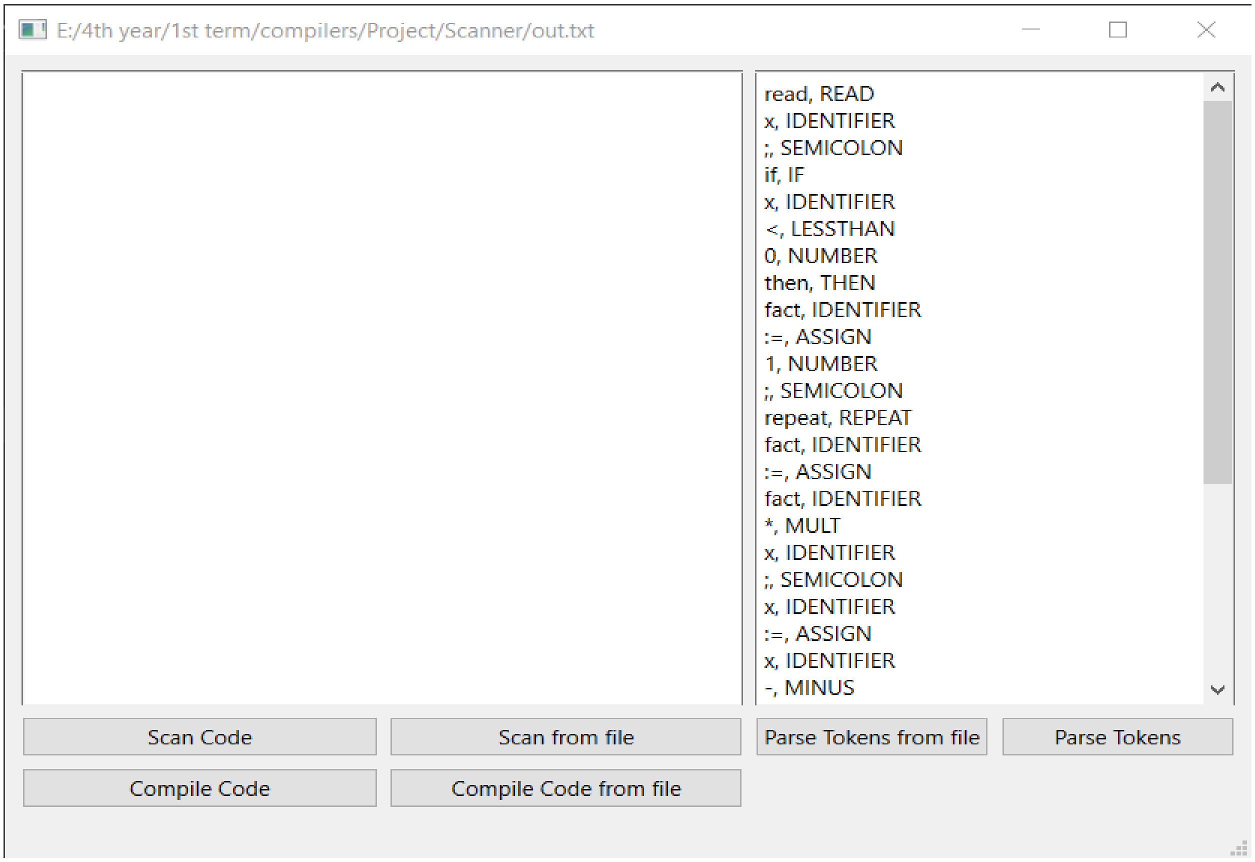
```
read, READ
x, IDENTIFIER
;, SEMICOLON
if, IF
x, IDENTIFIER
<, LESSTHAN
0, NUMBER
then, THEN
fact, IDENTIFIER
:=, ASSIGN
1, NUMBER
;, SEMICOLON
repeat, REPEAT
fact, IDENTIFIER
:=, ASSIGN
fact, IDENTIFIER
*, MULT
x, IDENTIFIER
;, SEMICOLON
x, IDENTIFIER
:=, ASSIGN
x, IDENTIFIER
-, MINUS
1, NUMBER
until, UNTIL
x, IDENTIFIER
=, EQUAL
0, NUMBER
;, SEMICOLON
write, WRITE
fact, IDENTIFIER
end, END
```

Parse tokens from file:

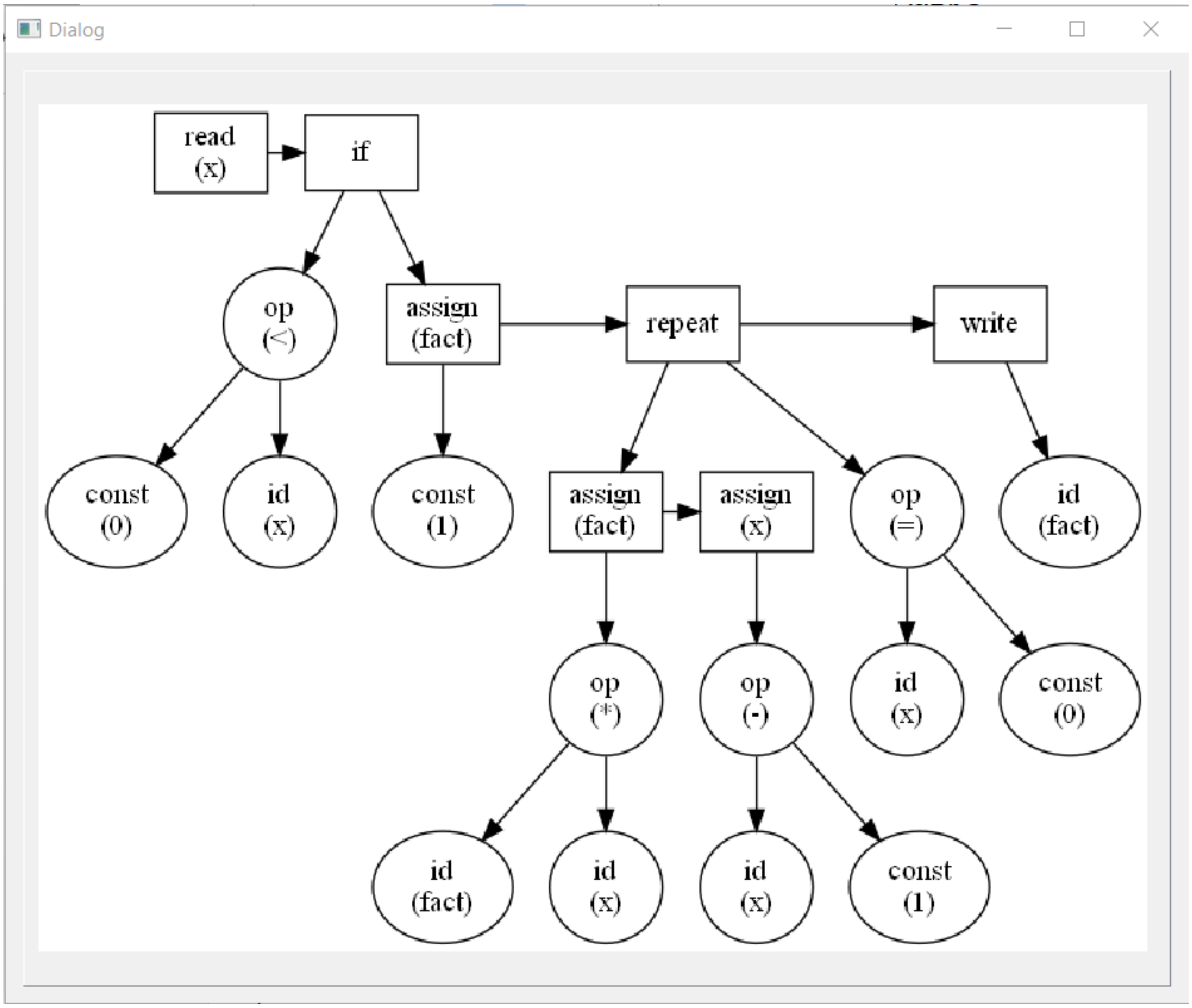




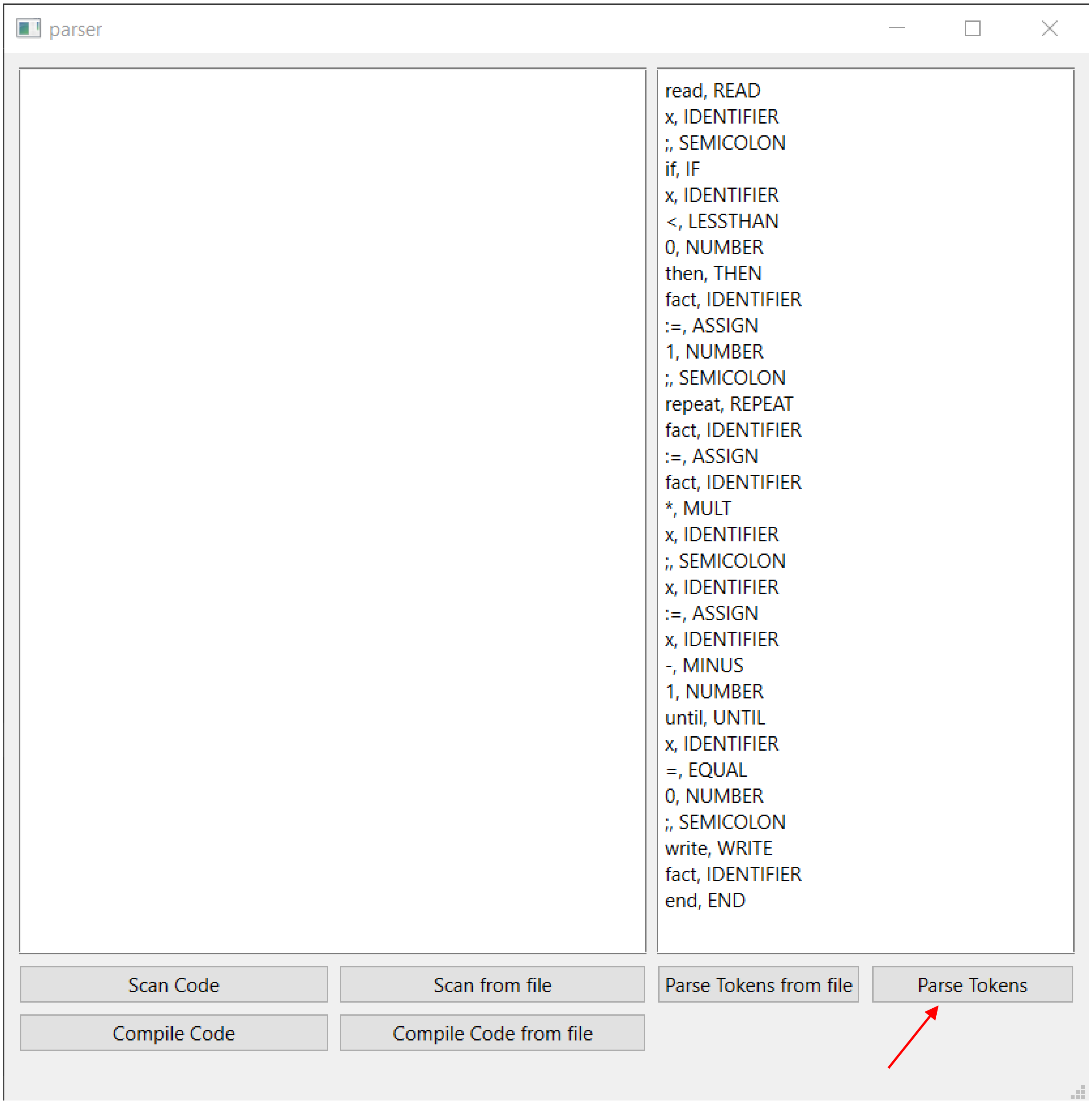
Name	Date modified	Type	Size
.vscode	11/27/2021 1:07 PM	File folder	
out.txt	11/12/2021 2:55 PM	Text Document	1
test.txt	12/28/2021 10:12 PM	Text Document	1
test1.txt	11/12/2021 2:17 PM	Text Document	1
test2.txt	11/12/2021 2:19 PM	Text Document	1


E:/4th year/1st term/compilers/Project/Scanner/out.txt

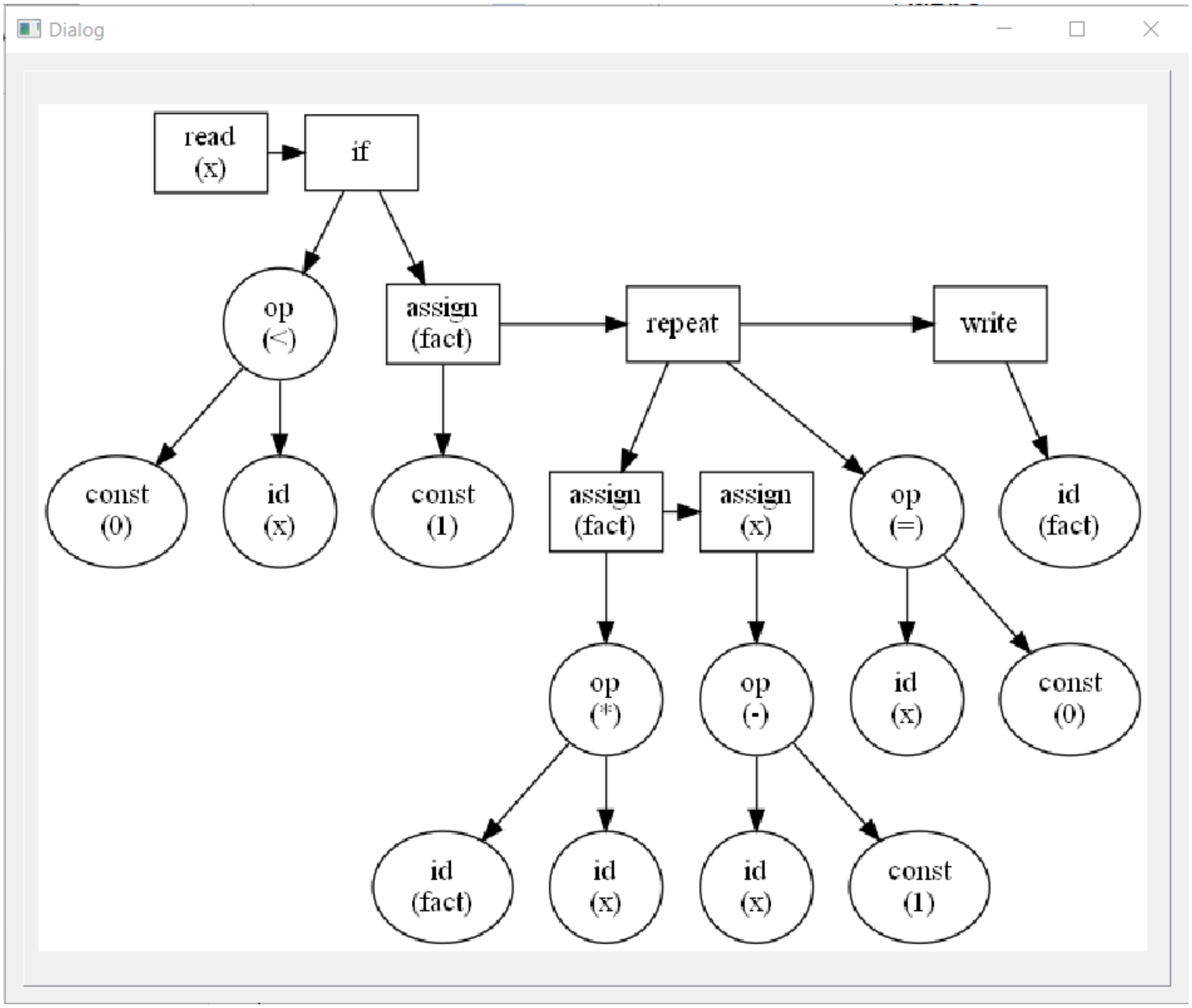
```
read, READ
x, IDENTIFIER
;, SEMICOLON
if, IF
x, IDENTIFIER
<, LESSTHAN
0, NUMBER
then, THEN
fact, IDENTIFIER
:=, ASSIGN
1, NUMBER
;, SEMICOLON
repeat, REPEAT
fact, IDENTIFIER
:=, ASSIGN
fact, IDENTIFIER
*, MULT
x, IDENTIFIER
;, SEMICOLON
x, IDENTIFIER
:=, ASSIGN
x, IDENTIFIER
-, MINUS
```


Dialog
read
(x)
if
op
(<)
assign
(fact)
repeat
write
const
(0)
id
(x)
const
(1)
assign
(fact)
assign
(x)
op
(=)
id
(fact)
op
(*)
op
(-)
id
(x)
const
(0)
id
(fact)
id
(x)
id
(x)
const
(1)

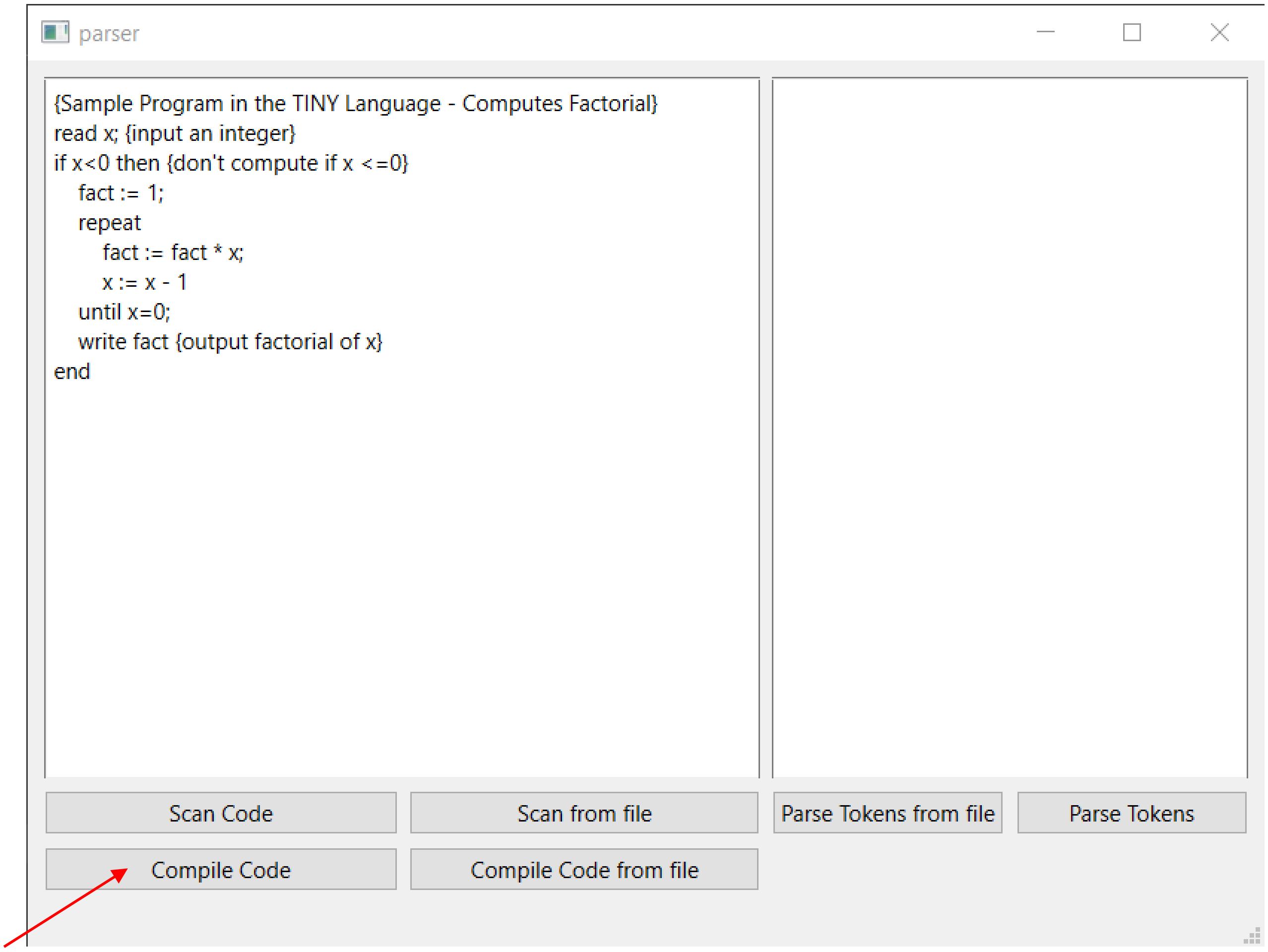
Parse tokens from textbox:



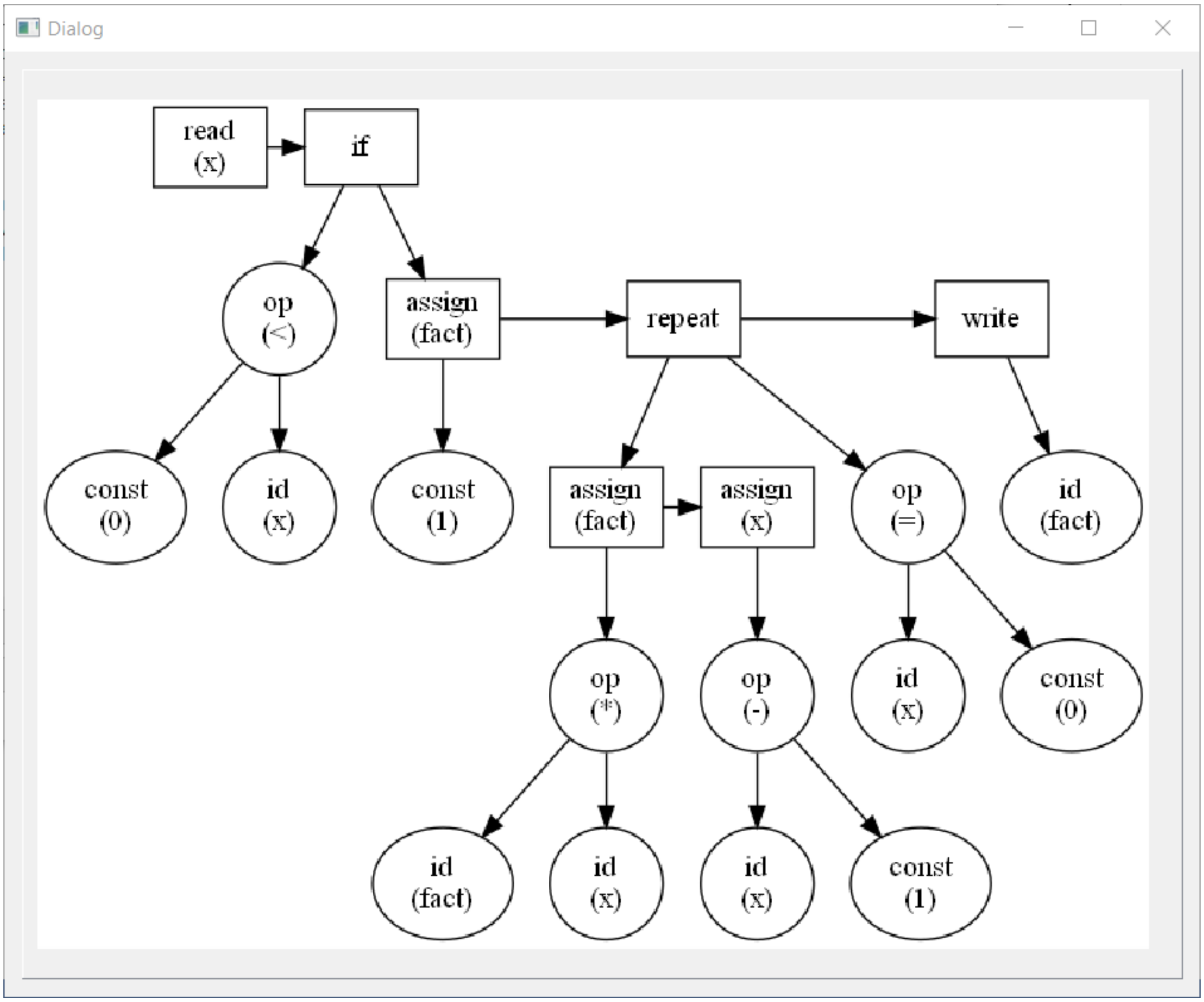
```
read, READ
x, IDENTIFIER
;, SEMICOLON
if, IF
x, IDENTIFIER
<, LESSTHAN
0, NUMBER
then, THEN
fact, IDENTIFIER
:=, ASSIGN
1, NUMBER
;, SEMICOLON
repeat, REPEAT
fact, IDENTIFIER
:=, ASSIGN
fact, IDENTIFIER
*, MULT
x, IDENTIFIER
;, SEMICOLON
x, IDENTIFIER
:=, ASSIGN
x, IDENTIFIER
-, MINUS
1, NUMBER
until, UNTIL
x, IDENTIFIER
=, EQUAL
0, NUMBER
;, SEMICOLON
write, WRITE
fact, IDENTIFIER
end, END
```


Dialog
read
(x)
if
op
(<)
assign
(fact)
repeat
write
const
(0)
id
(x)
const
(1)
assign
(fact)
assign
(x)
op
(=)
id
(fact)
op
(*)
op
(-)
id
(x)
const
(0)
id
(fact)
id
(x)
id
(x)
const
(1)

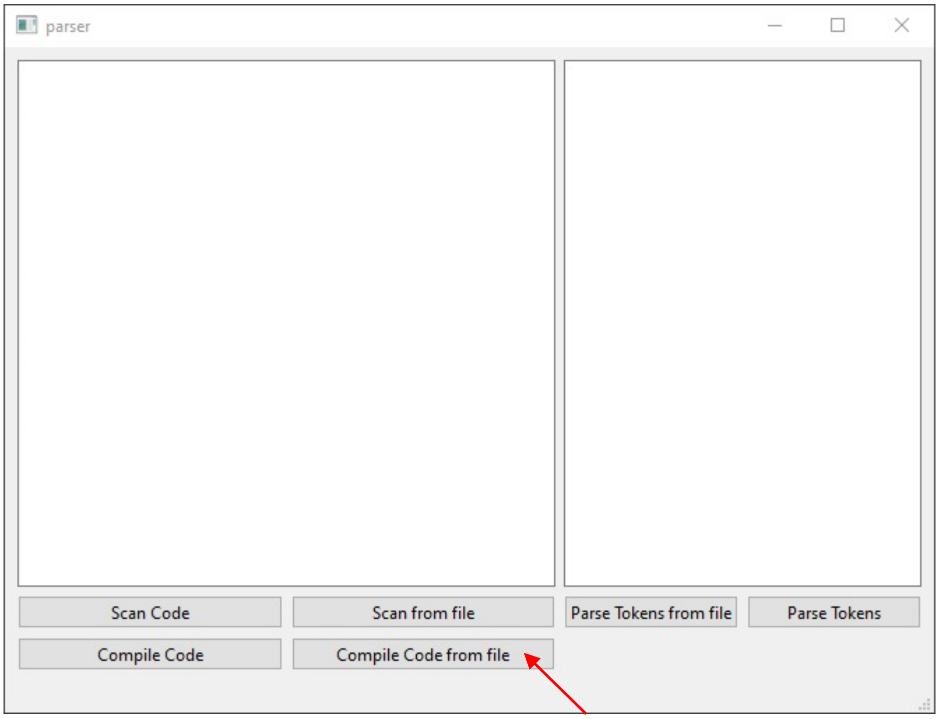
Compile code:

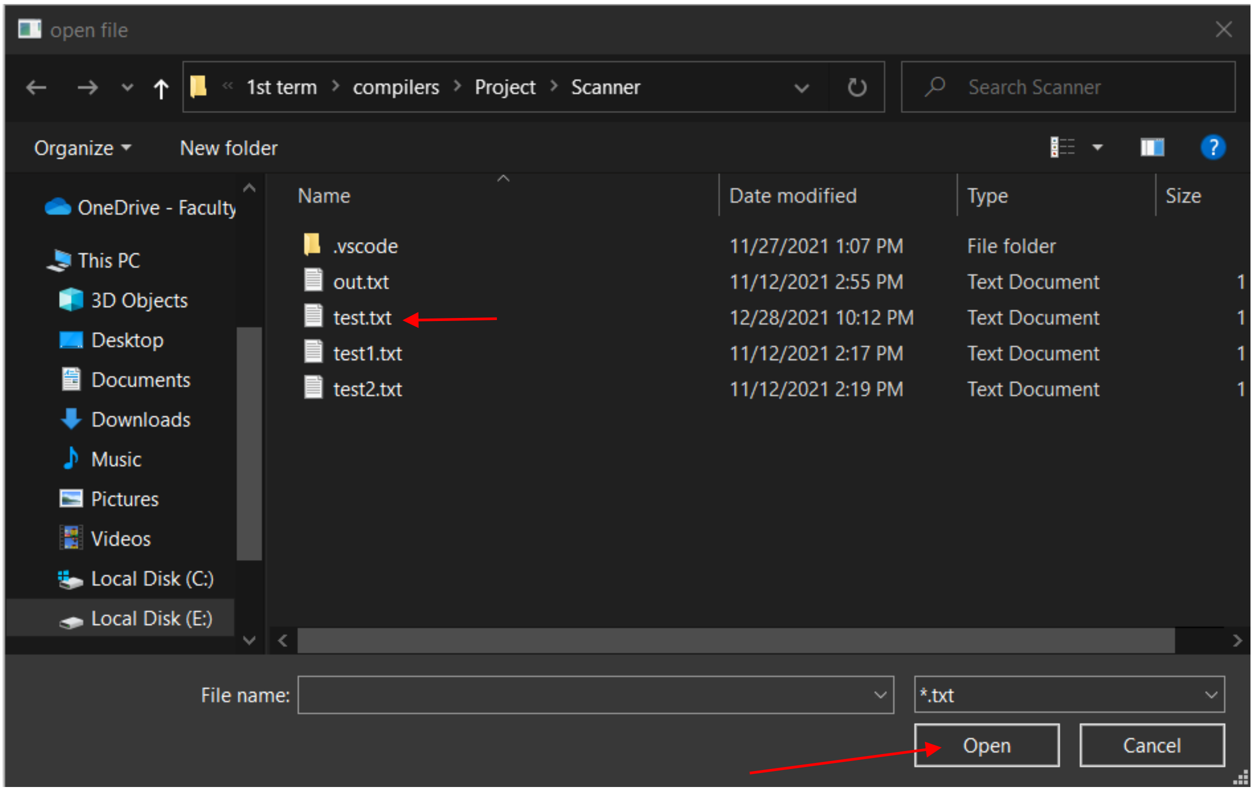


```
{Sample Program in the TINY Language - Computes Factorial}
read x; {input an integer}
if x<0 then {don't compute if x <=0}
   fact := 1;
   repeat
      fact := fact * x;
      x := x - 1
   until x=0;
   write fact {output factorial of x}
end
```


Dialog
read
(x)
if
op
(<)
assign
(fact)
repeat
write
const
(0)
id
(x)
const
(1)
assign
(fact)
assign
(x)
op
(=)
id
(fact)
op
(*)
op
(-)
id
(x)
const
(0)
id
(fact)
id
(x)
id
(x)
const
(1)

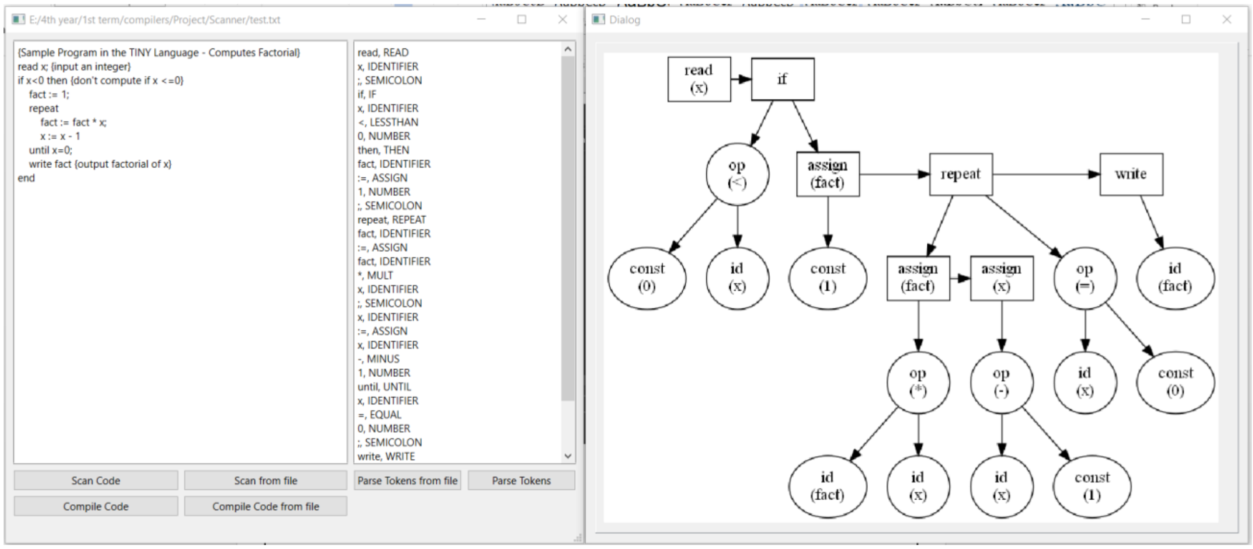
Compile code from file:





Name	Date modified	Type	Size
.vscode	11/27/2021 1:07 PM	File folder	
out.txt	11/12/2021 2:55 PM	Text Document	1
test.txt	12/28/2021 10:12 PM	Text Document	1
test1.txt	11/12/2021 2:17 PM	Text Document	1
test2.txt	11/12/2021 2:19 PM	Text Document	1

File name: *.txt



```
{Sample Program in the TINY Language - Computes Factorial}
read x; {input an integer}
if x<0 then {don't compute if x <=0}
   fact := 1;
   repeat
      fact := fact * x;
      x := x - 1
   until x=0;
   write fact {output factorial of x}
end
```

```
read, READ
x, IDENTIFIER
;, SEMICOLON
if, IF
x, IDENTIFIER
<, LESSTHAN
0, NUMBER
then, THEN
fact, IDENTIFIER
:=, ASSIGN
1, NUMBER
;, SEMICOLON
repeat, REPEAT
fact, IDENTIFIER
:=, ASSIGN
fact, IDENTIFIER
*, MULT
x, IDENTIFIER
;, SEMICOLON
x, IDENTIFIER
:=, ASSIGN
x, IDENTIFIER
-, MINUS
1, NUMBER
until, UNTIL
x, IDENTIFIER
=, EQUAL
0, NUMBER
;, SEMICOLON
write, WRITE
```