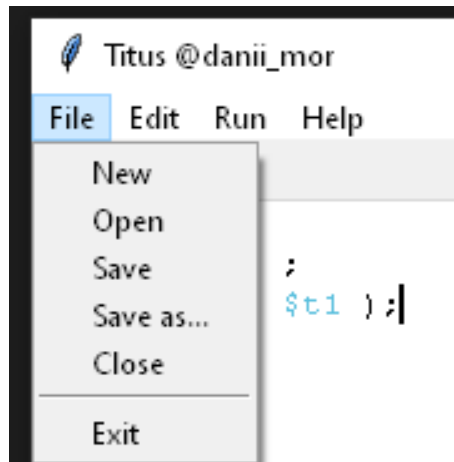


Titus

Developed by @dani_i_mor

USER MANUAL

MENU BAR

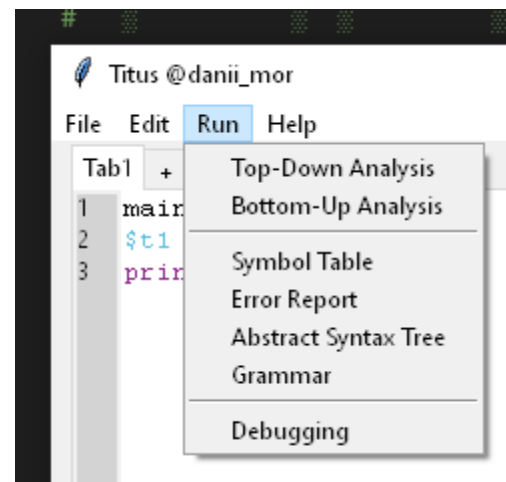


In this section we will be able to appreciate all the functions of our application. We will start with File that is used for:

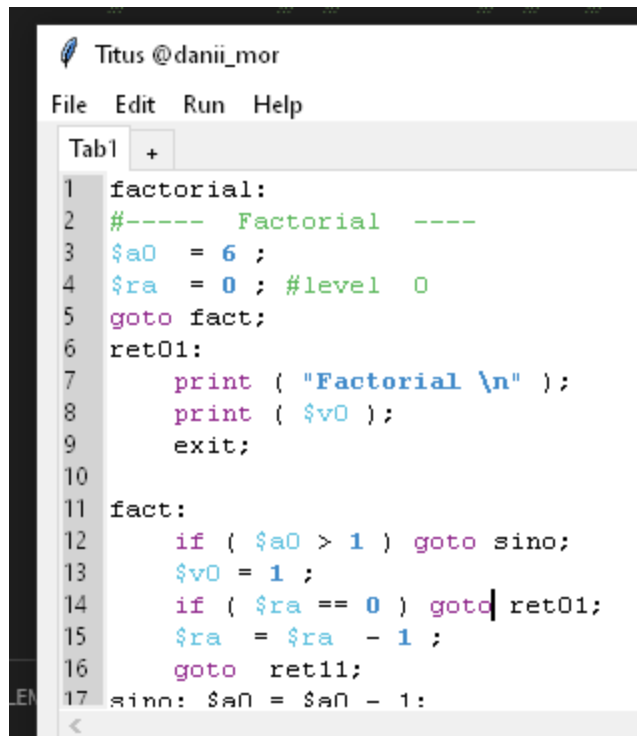
- Create new
- Open file
- Save File
- Save as
- Close file
- Close application

The most important part of our application falls on this section. In Run we can do analysis and verify our options in different ways. In them we can see:

- Top-down analysis
- Bottom-up analysis
- Symbol Table
- Error Report
- AST
- Grammar
- Debugging



EDITOR AREA



```
1 factorial:
2 #---- Factorial ----
3 $a0 = 6 ;
4 $ra = 0 ; #level 0
5 goto fact;
6 ret01:
7     print ( "Factorial \n" );
8     print ( $v0 );
9     exit;
10
11 fact:
12     if ( $a0 > 1 ) goto sino;
13     $v0 = 1 ;
14     if ( $ra == 0 ) goto ret01;
15     $ra = $ra - 1 ;
16     goto ret11;
17 sino: $a0 = $a0 - 1;
```

In the text editor we can have multiple windows. So, you can work with multiple files at the same time.

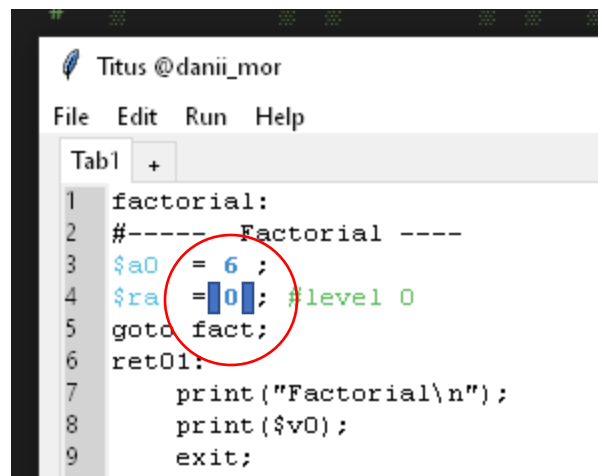
It can also be seen that the line number is being indicated while we are advancing in the writing of our editor.

Seeing in more detail you can see the highlighting of colors. This to be effective we must apply spaces around each keyword. The part of applying highlighting to pasted code in the editor is still in alpha.

Therefore, the space between the keywords must be manually applied for the editor to detect the colors.

In this example we can detail how this small air should be created between the keywords. This for the highlighting to take effect.

Also keep in mind that when there is text strings they generate a bug so you must have a space within the text to be able to detect where the highlighting of the text ends.



```
1 factorial:
2 #---- Factorial ----
3 $a0 = 6 ;
4 $ra = 0 ; #level 0
5 goto fact;
6 ret01:
7     print ( "Factorial \n" );
8     print ( $v0 );
9     exit;
```

Symbol Table

The screenshot shows a debugger window titled "Titus @dani_mor". The main window displays assembly code for a factorial function. A symbol table window is overlaid on the code, showing the following data:

IDENTIFIER	TYPE	SIZE	VALUE	SCOPE	REFERENCE
MAIN	FUNCTION	0	None	GLOBAL	LABEL
factorial	FUNCTION	0	None	GLOBAL	LABEL
\$a0	INT	0	6	factorial	PARAMETER
\$ra	INT	0	0	factorial	RETURNPOINT
fact	FUNCTION	0	None	GLOBAL	LABEL
ret01	FUNCTION	0	None	GLOBAL	LABEL
\$v0	INT	0	720	ret01	RETURNEDVALUE
sino	FUNCTION	0	None	GLOBAL	LABEL
ret11	FUNCTION	0	None	GLOBAL	LABEL

The assembly code in the background includes labels like main, factorial, fact, and ret01, with instructions such as \$a0 = 6, \$ra = 0, goto fact, and print statements.

Error Report

The screenshot shows a debugger window titled "Titus @dani_mor". The main window displays assembly code. An error report window is overlaid on the code, showing the following message:

Lexical error <@> at
line:2, column:1
Syntax error at line:22,
column:11

The error report window is titled "Titus @dani_mor". The assembly code in the background includes labels like sino, ret11, and instructions such as \$a0 = \$a0 - 1, \$ra = \$ra + 1, goto fact, \$t100 = - , and \$ra = \$ra - 1.

Grammar

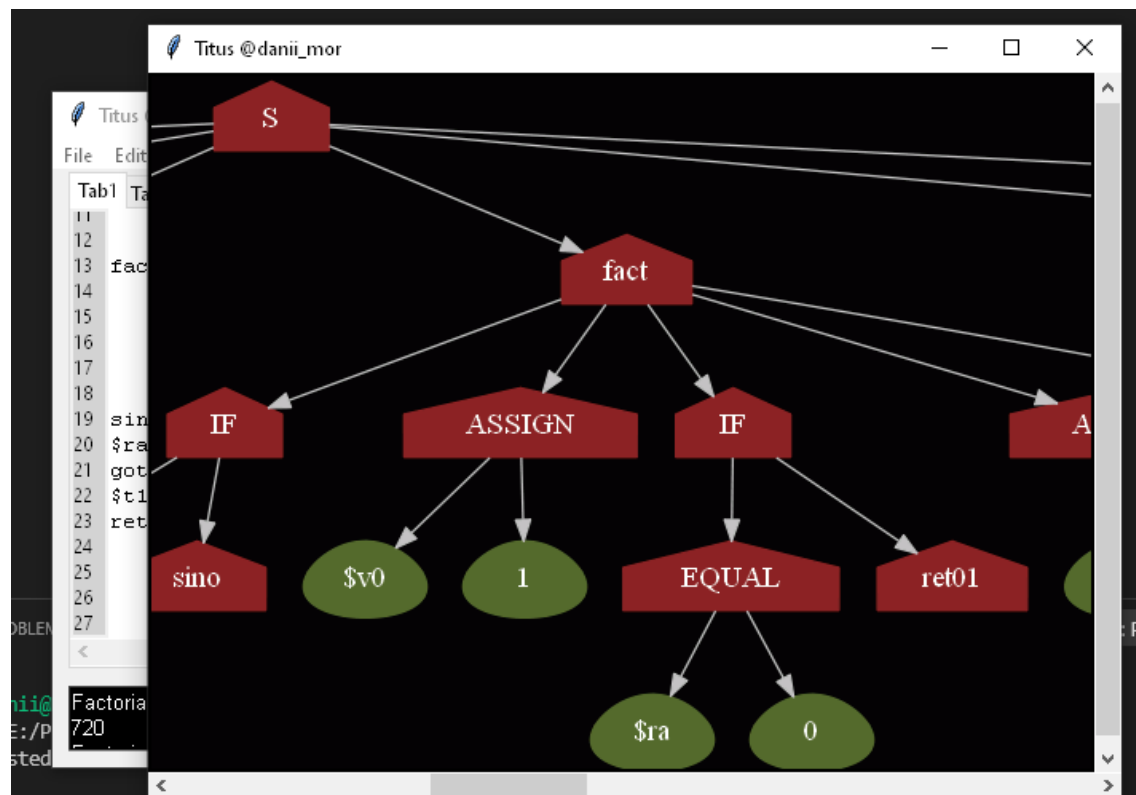
The image shows a code editor window titled 'Titus @dani_mor' with a file menu (File, Edit, Run, Help). The editor contains assembly-like code for a factorial function. A separate window titled 'Titus @dani...' displays a grammar specification.

```
11 exit;
12
13 fact:
14     if ($a0>1) goto sino;
15     $v0 = 1;
16     if ($ra==0) goto ret01;
17     $ra = $ra - 1;
18     goto ret11;
19 sino: $a0 = $a0 - 1;
20 $ra = $ra + 1; #level ++
21 goto fact;
22 $t100 = - ;
23 ret11:
24     $a0 = $a0 + 1;
25     $v0 = $a0 * $v0;
26     if ($ra==0) goto ret01;
27     $ra = $ra - 1;
```

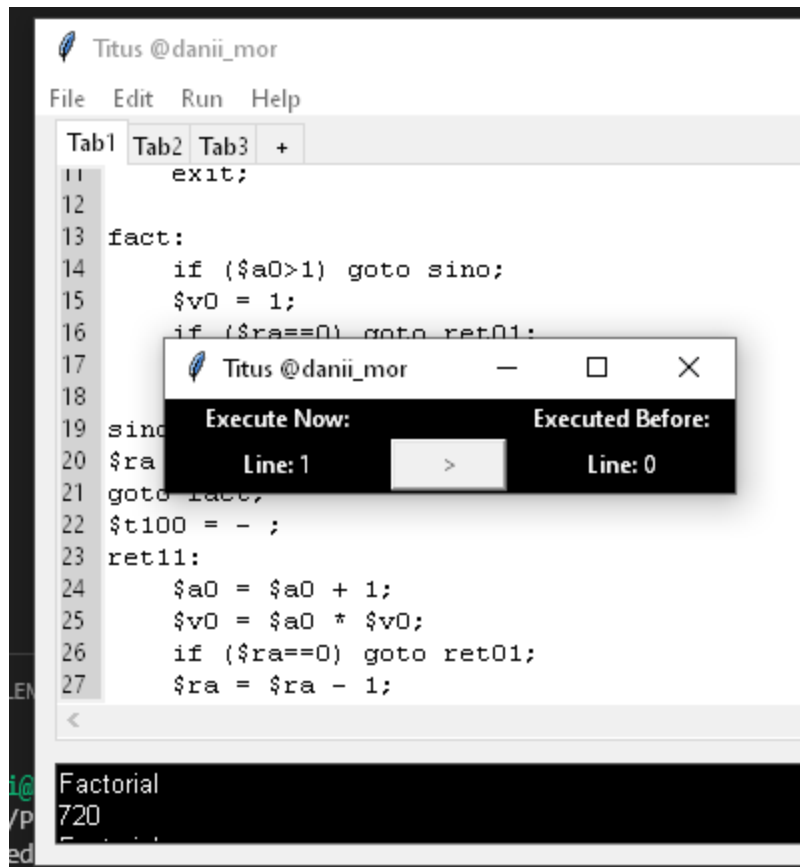
Factorial
720

s -> code
code -> MAIN :
code -> code LABEL : list
list -> statement ;
list -> list statement ;
statement -> is_array_term = expression
statement -> PRINT (term)
statement -> IF (expression) GOTO LABEL
statement -> GOTO LABEL
statement -> EXIT
expression -> term + term
expression -> term - term
expression -> term * term
expression -> term > term
expression -> term == term
expression -> term
factor -> NUMBER
factor -> STRING
is_array_term -> VAR
factor -> is_array_term
term -> factor

AST



Debugg



Info

