Università degli studi di Pisa

DIPARTIMENTO DI INFORMATICA

Anno Accademico 2014/2015

Funw@p Project

Documentation on the project

Federico Conte

1 How to use the software

The software was built with the goal to simulate the behaviour of an **IDE** (Integrated development environment). The Figure 1 below enumerates all the components of the IDE in order to simplify their descriptions.

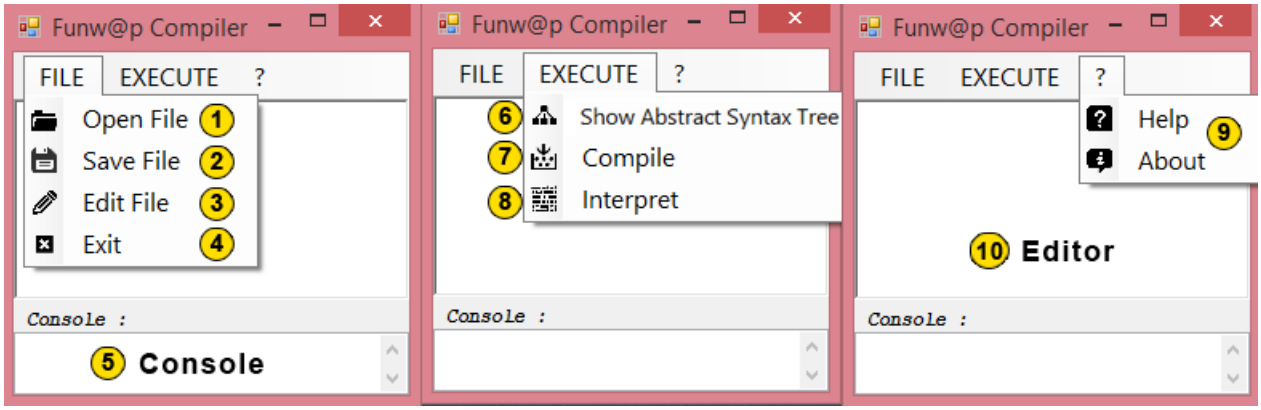

Figure 1: *Three screenshot of the IDE showing all possible menu buttons.*

1. **Open File** → The text of the file is written into the Editor; the **Scanner phase** is performed automatically in order to recognize all the **Tokens** of the **Funw@p** language and to properly colorize the code, like it is shown it the Figure 2.
   In the folder *Examples*, located inside the folder of the project, the user can find a list of working examples and looking them, it is easy to see the potential of language.

2. **Save File** → When the user is satisfied with the written code, it can save it into a *txt* file.

3. **Edit File** → Every time the user performs an operation on the written code (e.g save the file, compile or interpret the code), the Editor become uneditable. In order to switch into the edit mode, the user has to click this button.

4. **Exit** → This button allows to exit from the program.

5. **Console** → All the action performed and all the errors are shown in this *TextArea*.

6. **Show Abstract Syntax Tree** → Generate a new windows that shows the graphical AST obtained after the **Parser phase** (as it is possible to notice looking the Console).
   The Figure 2 shows an example of AST obtained from the the code of the factorial program.

7. **Compile** → In order to obtain the compiled code, we need the AST; so, first of all, we have to perform the **Parser phase**. Then we use the generated AST to do the **type and enviroment checking**, so we will be sure that the generated code is correct. At the end, we compile the code generating a *C#* file (*.cs*) and an executable file (*.exe*).

8. **Interpret** → As above, in order to perform the interpretation of the code, we need to perform the **Parser phase**, generate the AST and do the **type and enviroment checking**. Then, the code is interpret using the AST, and a special black Windows is used to visualize the *println* and *readln* operations.

9. Help and About→ These buttons shows some information about the software.

10. Editor At the beginning, this *TextArea* is editable and the user can type the code like in a normal text editor. When an operation is performed, the code is colored and the Editor becomes uneditable.

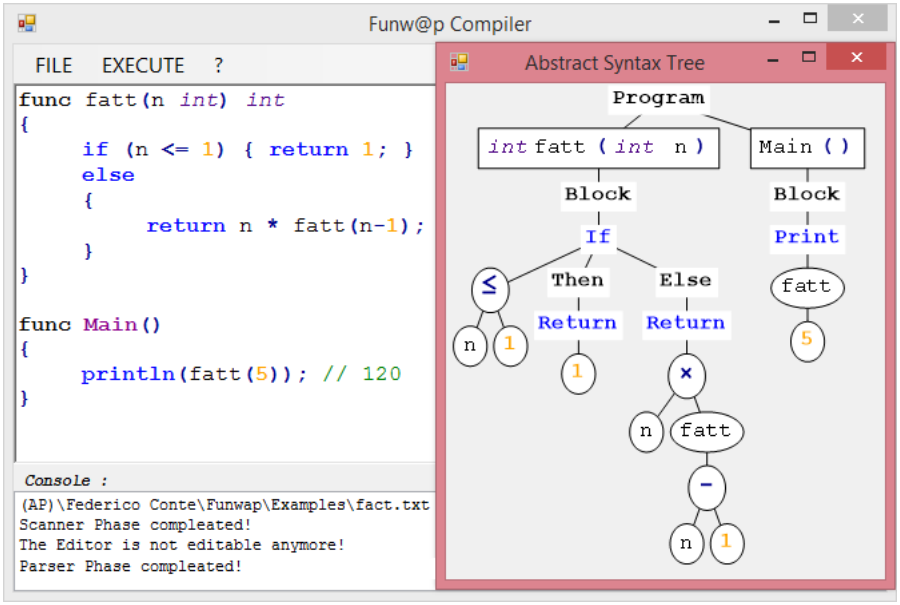

Figure 2: *An example of colorized code and the Abstract Syntax Tree generated from it.*

2 Implementation choices

The file ***Funw@p - Grammar.pdf*** shows the grammar of the **Funw@p** language.

Respect with the specifications:

- We used the Token **func**, instead of **fun**, in order to identify immediately the function declaration (also useful during the code's coloration).
- We force the variable of type function to specify the type of the parameters the function is expecting and the expected return type (that is the same format used for the function return type during the function declaration).
  This choice was made in order to simplify the type checking and the compilation of higher order functions.

The project supports the **Sandcastle Documentation**, thanks to which is possible to go into the implementation deteils.

3 Issues in the project

When the **dasync** command is compiled, the program creates a server file containing the remote function. The user source code, now refers to this file in order to perform the remote call. Therefore, in order to compile the user souce code, we need to add this reference among the options of *C#* compiler, that is called by the program via command line.
Despite the references included, the **compilation fails**.

We are trying to solve the problem, asking help in some forum, like:
`http://stackoverflow.com/questions/27788231/`
`how-to-properly-use-lib-and-reference-options-when-compiling-c-sharp-code-via-co`