Compilers Project 2023: *Mini-PL interpreter*

Implement an *interpreter* for the Mini-PL programming language. The language analyzer must correctly recognize and process all valid (and invalid) Mini-PL programs. It should report syntactic errors. More involved error recovery schemes in parser are not mandatory, but will provide additional points. It must also construct an AST and make necessary passes over this program representation. The semantic analysis part binds names to their declarations, and checks semantic constraints, e.g., expressions types and their correct use. If the given program was found free from errors, the interpreter part will immediately execute it.

Implementation requirements and grading criteria

The assignment is done as individual work. When building your interpreter, you are expected to properly use and apply the compiler techniques taught and discussed in the lectures and exercises. The Mini-PL analyzer is to be written purely in a *general-purpose programming language*. By default C# should be used as the implementation language (if you have questions, please consult the teacher). **Ready-made language-processing tools (language recognizer generators, regex libraries, translator frameworks, etc.) are not allowed**. Note that you can of course use the basic general data structures of the implementation language - such as strings and string builders, lists/arrays, and associative tables (dictionaries, maps). You must yourself make sure that your system can be run on the development tools available at the CS department.

The emphasis on one part of the grading is the quality of the implementation: especially its overall architecture, clarity, and modularity. Pay attention to programming style and commenting. Grading of the code will consider (undocumented) bugs, level of completion, and its overall success (solves the problem correctly). Try to separate the general (and potentially reusable) parts of the system (text handling and buffering, and other utilities) from the source-language dependent issues.

Your interpreter should be usable through terminal and work by passing the filename of the source file as a parameter to the program:

```
“<command for running interpreter> path_source_file”
```

The interpreter should not require any file endings for the source file, but for the sake of clarity, “*.mpl” ending can be used for all source files.

Documentation

Write a report on the assignment, as a document in PDF format. The title page of the document must show appropriate identifications: the name of the student, the name of the course, the name of the project, and the valid date and time of delivery.

Describe the overall architecture of your language processor with, e.g., UML diagrams. Explain your diagrams. Clearly describe yout testing, and the design of test data. Tell about possible shortcomings of your program (if well documented they might be partly forgiven). Give instructions how to build and run your interpreter. The report must include the following parts

1. The Mini-PL token patterns as *regular expressions* or, alternatively, as *regular definitions*.

2. A *modified context-free grammar* suitable for recursive-descent parsing (eliminating any LL(1) violations); modifications must not affect the language that is accepted.
3. Specify *abstract syntax trees* (AST), i.e., the internal representation for Mini-PL programs; you can use UML diagrams or alternatively give a syntax-based definition of the abstract syntax.
4. *Error handling* approach and solutions used in your Mini-PL implementation (in its scanner, parser, semantic analyzer, and interpreter).
5. Include your *work hour log* in the documentation. For each day you are working on the project, the log should include: (1) date, (2) working time (in hours), (3) the description of the work done. And finally, the total hours spent on the project during the project course.

For completeness, include the original project definition and the Mini-PL specification as appendices of your document; you can refer to them when explaining your solutions.

Delivery of the work

The final delivery is due at 12:15 on Mo 13.3.2023.

The work should be returned to the course Moodle page, in a zip form. This zip (included in the e-mail message) should contain all relevant files, within a directory that is named according to your name. The deliverable zip file must contain (at least) the following subfolders.

```
<user>
  ./doc
  ./src
```

When naming your project (.zip) and document (.pdf) files, always include your name and the packaging date. These constitute nice unique names that help to identify the files later. Names would be then something like:

> project zip: `user_proj_2023_03_12.zip`
> document: `user_doc_2023_03_12.pdf`

More detailed instructions and the requirements for the assignment are given in the lectures. If you have questions about the folder structure and the ways of delivery, or in case you have questions about the whole project or its requirements, please contact me.