Lexical and syntactical structures in programming language OCaml

Pavel Vraštiak, xvrast00@stud.fit.vutbr.cz
Lukáš Hübner, xhubne00stud.fit.vutbr.cz

OCaml is a programming language that unifies functional, imperative, and object-oriented programming under an ML-like type system. It has a static type system, type inference, parametric polymorphism, tail recursion, pattern matching, first class lexical closures, functors, exception handling, and incremental generational automatic garbage collection.

OCaml is derived from Caml language. Caml is extended by adding object-oriented layer and a set of modules. However, the basic features are common for both Caml and OCaml. Probably the key difference between OCaml and other academic programming languages is its stress on performance. Because of its static typing, OCaml programs can be heavily optimized and also formally proven by the compiler.

Since OCaml supports functional and also imperative programming, it provides structures for both of these programming paradigms. The imperative part of OCaml offers common features such as variables, arrays, records, and several types of loops.

OCaml is designed to use automatic memory management. There is no malloc, delete, or new operator (or function). The automatic memory management provides more safety because it prevents memory leaks and wrong memory allocations.

In our presentation we will try to make readers familiar with basic OCaml expressions as well as with more advanced features of this, for some possibly unknown, language. Apart from simple lexemes, such as whitespace, comments, literals, and keywords, we will focus on the pattern matching, records, arrays and strings.

The last covered topic will be objects and classes, which represents the main difference against Caml language.

After listening to our presentation you should be able to understand basic OCaml lexical elements and their syntax.