# COMP 520 - Milestone 1

Alexandre St-Louis Fortier (260720127)
Stefan Knudsen (260678259)
Cheuk Chuen Siow (260660584)

February 26, 2016

## Design Decisions

### Language choice

We chose OCaml as our implementation language for compiler design because the recursive nature of the language allows for easy manipulation of abstract syntax trees. In addition, the pattern matching feature that OCaml provides is particularly helpful in constructing the compiler and has been applied to most parts of the compiler, including scanner, parser, and pretty printer. The tools that we used for building the scanner and parser are `ocamllex` and `Menhir` and by referring to [1, 2, 3]. The maturity of the parsing and scanning tools were another reason for the choice of OCaml. What's more, the lead TA had pointed out that OCaml is a good language for such a project.
We initially had trees for Sum of two expressions, Add of two expressions, etc. We changed this so that there was only one binary expression Node which included the kind of binary operation to be done, and the same thing for unary operations.

### Semicolon

For inserting semicolon token according to rule 1, we added a variable `insert_semic` for each of the tokens that define the insertion condition. Upon reaching a newline or end of file tokens, our program checks whether the previous token that satisfy rule 1 has updated `insert_semic`, and if it evaluates to `true` it will trigger the insertion of the semicolon token.

### Type

We made a decision to handle types as identifiers and not include the primitive types as tokens. This way our parser will be able to accept both primitive types and type aliases when specifying types. We also decided to defer the check for invalid use of primitive types in variable identifiers to the weeding phase.

### Tokens

We took the token definitions from `parser.mly` and placed them into a separate file `tokens.mly`. Along with the flags defined in `myocamlbuild.mly`, this enables the main program to display a list of tokens from the tokenizer.

### Pretty printing

Initially, we used semicolons to separate each kind of print statement. For better readability, we now use Printf.fprintf so that the string component of a statement/expression can be written one line with the parts to be expanded following. followed by the "string" body and functions that output each of the required output.

## Team Work

### Scanner

Alexandre laid the ground work for defining the tokens. Cheuk Chuen and Alexandre defined the regular expressions for the literals.

### Parser

Cheuk Chuen laid the ground work for defining the grammar and AST. Alexandre made important changes and thorough bug checks.

### Pretty printer

Stefan worked extensively on the pretty printer.

### Testing

Alexandre wrote shell scripts to automate testing of the existing valid and invalid programs.

## References


1. http://caml.inria.fr/pub/docs/manual-ocaml-4.00/manual026.html
2. https://realworldocaml.org/v1/en/html/parsing-with-ocamllex-and-menhir.html
3. http://pauillac.inria.fr/ fpottier/menhir/manual.pdf