Exercise 1. Types construction

Download the file dcc_types.tgz and decompress it.

- **Inspect the files** Type.h/.cc. In the main, write the code to create and print-out the type char[8].
- **Open the file** parser.ypp. What is the attribute of the non-terminal type? Complete the rules of type to build correctly the types.
- **Inspect the files** SymbolTable.h/.cc. In parser.ypp, what is the purpose of add_type($3,$2) (after line 215 - rule of type_def)? Add print_symbols(cout) to print-out the registered types in the symbols table (Note: Instead of doing it in the main, do it inside the rule prog before normalizing the types). Test on tests/test.c.

Exercise 2. Normalization and well-funded types

- Type owns a method print_dot() which prints out the dotty representation (graph) of the current type (to have a "ps": dot -Tps test.dot > test.ps). Experiment.
- We still have identifiers inside the types, and we need to replace them by their definitions. This step is called normalization and happens after the last reduction of type_def_list (last line of parser.ypp).
- **Inspect the code of** normalize_types (SymbolTable.cc). Print-out the graph (print_dot) of the normalized list_t.
- **Inspect the code of** is_well_formed (Type.cc). After this step, we are sure that all the types are well-formed.
- What does reset_functions() do? (parser.ypp, last line)

Exercise 3. Type equivalence

Before checking the functions, we need an equivalence between types. Open Type.cc (line 116), and implement the equivalence of types.

Exercise 4. Type control

Each time a function is declared (parser.ypp, line 478), its signature is added to the symbol table. add_function() creates a new (signature of the) current function. Then, add_argument_type() adds the types of the arguments. add_argument() declares an argument and add_local_var declares a new local variable. Then, these informations are used to type the expressions inside the function body.

- **Inspect** the rules of function, declare_args, declare_local_vars.
- **It is time to control the types...** Inspect the rules of stmt. How do we manage the affectation polymorphism?
- **Inspect** the rule for the return.
- **Inspect** the procedure call (function that returns void). What does arg_type correspond to? What is type_check() doing?
- **Complete** the rules of the non-terminal expr to control the types.