# Concordia University
# Department of Computer Science
# and Software Engineering

# Compiler Design (COMP 442/6421)
# Winter 2017

# Assignment 3, Semantic Analysis

| | |
|---|---|
| **Deadline:** | Monday March 20th, 2017 |
| **Evaluation:** | 10% of final grade |
| **Late submission:** | penalty of 50% for each late working day |

In this assignment, you are to implement a symbol table support for the language described in assignment 2. Its aim is to resolve the typing and cross-referencing of your identifiers, taking into consideration the scoping of your identifiers. As you can see from the syntax, this language resembles C++:

- A program consists of a main function (the **`program`** function) that uses a set of free functions. Functions used must be declared before being called. If not, an error message is issued that notices that the function called is undeclared. In order to avoid that, you may want to implement two passes: a first pass that constructs the symbol tables and does not report such semantic errors, and a second pass that uses the information generated in the first pass.
- Classes represent the encapsulation of a user defined data type and its functions. They are defined before the **`program`** function. Each class must be defined before it is used either as a data member in another class, or in any function definition. If not, an error message is issued noticing that the data type is undefined. Two passes (as described in the above bullet) can be used to resolve this issue. In any case, circular class dependencies should be reported as a semantic error.
- In functions, (including member functions in classes) variables are defined before the statements. Variables used in a function must be declared before use. If not, an "undeclared variable" error message is issued.
- An identifier cannot be declared twice in the same scope. In such a case, a "multiply declared identifier" message should be issued. However, note that it is allowed to have two members with the same name in two different classes, as they are not defined in the same scope.
- The variables declared inside the functions or classes (data members) are considered local and thus can only be used in the current function or class scope. Data members can be used in all member functions of their respective class. This raises the need for a nested symbol table structure:

    1. A symbol table contains an entry for all identifiers (*variables, functions, classes)* defined in its own scope. There are scopes for each class definition, free or member function definition, and a global scope for the whole program.
    2. The global symbol table, representing all the symbols defined in the global scope, exists until the end of the compilation process.
    3. All local symbol tables are representing sub-scopes and should be bound to their respective elements in the current symbol table.
    4. A local symbol table is created at the beginning of the compilation of any function, and can cease to exist when the compilation process is over for this function.
    5. A local symbol table is created at the beginning of the compilation of any class, and is only deleted at the end of the compilation process, as their symbol tables will be required as functions will refer to class' data members.

Therefore, you have to associate with each variable and function identifier a record that contains its properties in the appropriate symbol table. You have to keep in mind that you might have to change the structure of these records later in the design of the compiler. Make sure you can change the record structure with minimal changes to your symbol table manipulating functions.

## Functions to manipulate symbol tables

| | |
|---|---|
| **`create(Tn)`** | Creates a new, empty table |
| **`search(Tn,i,ptr,found)`** | Searches the table **`Tn`** for a given identifier **`i`**, and recursively upwards in the symbol table nest if not found locally. If the identifier is found, parameter found is true and **`ptr`** gives the pointer to the record associated with that identifier. Otherwise found is false. |
| **`insert(Tn,i,ptr)`** | Inserts identifier **`i`** in table **`Tn`**, and **`ptr`** points to the newly inserted record. |
| **`delete(Tn)`** | Deletes the symbol table **`Tn.`** |
| **`print(Tn)`** | Prints all identifiers in **`Tn`** and their properties. This procedure is needed only for debugging purposes. |

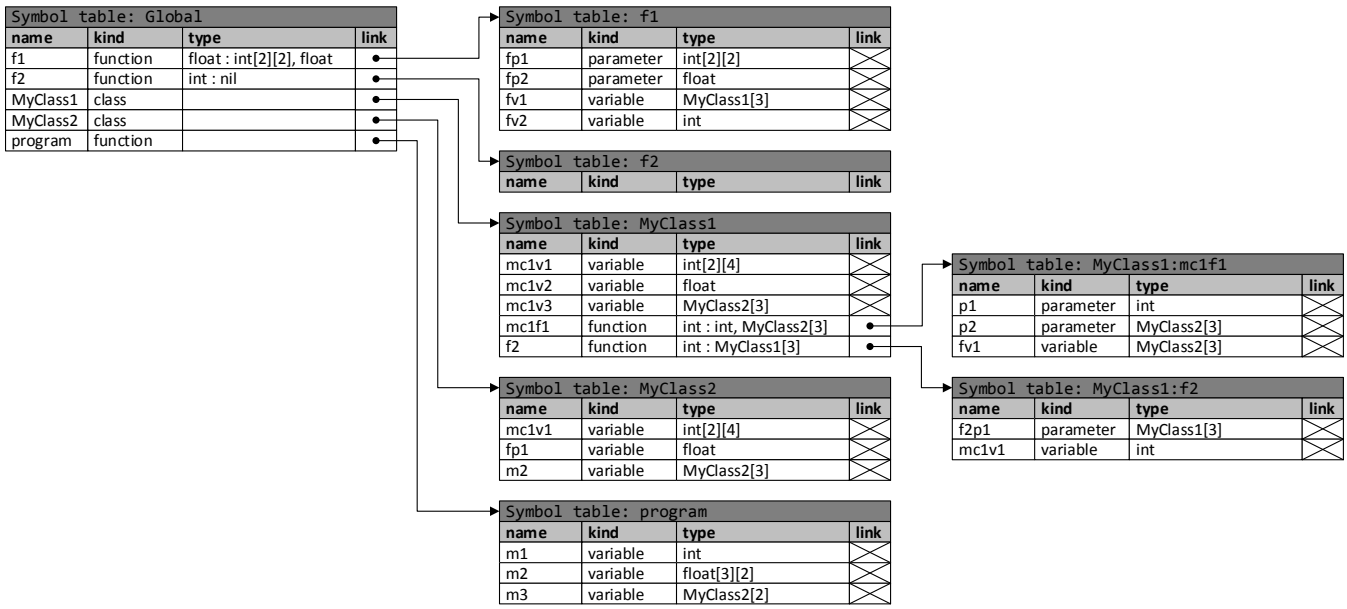
## Properties stored in the symbol table

- whether the identifier has been properly *declared.*
- its *type* (integer, floating point number, function, class).
- if it is a function, then the *number of parameters* and the *type* of each parameter (probably implemented as a linked list) is stored, along with a pointer to a symbol table structure describing the symbols local to this function.
- if it is a class, the record is actually a pointer to a local symbol table structure describing the symbols local to this class. Note that each member function entry in this local table will also point to their respective local tables.
- if it is a variable then store the *kind* of variable (normal variable, or parameter).
- *structure* (simple, array, class).
- if the variable is an array then store its *dimension*
- the *address* of the corresponding element in memory, or the assembly code alias of this variable (to be used for code generation)

## Work to be done

- Implement the data structures and functions for the symbol tables.
- Add into your parser, in appropriate places, calls for symbol table handling so that:
  - A new table is created at the beginning of the program for the global scope.
  - A new entry is created in the global table for each class definition. These entries should be links to local tables for these classes.
  - An entry in the appropriate table is created for each variable defined, i.e. data members or function's local variables.
  - An entry in the appropriate table is created for each function definition. These entries should be links to local tables for these functions.
  - The content of the symbol tables should be printed in order to demonstrate their correctness/completeness.
- Provide with your program a documentation that describes the overall organization of the symbol tables, the data structure used to implement the tables, and the locations in your grammar of the function calls to create and destroy tables, and the function calls to create entries in the tables.

## Example




```
class MyClass1 {
  int mc1v1[2][4];
  float mc1v2;
  MyClass2 mc1v3[3];
  int mc1f1(int p1, MyClass2 p2[3]) {
    MyClass2 fv1[3];
    …
  }
  int f2(MyClass1 f2p1[3]) {
    int mc1v1;
    …
  }
}
class MyClass2 {
  int mc1v1[2][4];
  float fp1;
  MyClass2 m2[3];
  …
}
program {
  int m1;
  float[3][2] m2;
  MyClass2[2] m3;
  ...
}
float f1(int fp1[2][2], float fp2) {
  MyClass1[3] fv1;
  int fv2;
  …
}
int f2() {
...
}
```

## Assignment submission requirements and procedure

You have to submit your assignment before midnight on the due date using the ENCS Electronic Assignment Submission system under the category "*programming assignment 3*". The file submitted must be a **`.zip`** file containing:

- all your code
- a set of input files to be used for testing purpose, as well as a printout of the resulting output of the program for each input file (symbol table output and error reporting, as described above)
- a simple document containing the information requested above

You are also responsible to give proper compilation and execution instructions to the marker in a README file. If the marker cannot compile and execute your programs, you might have to have a meeting with the marker for a demonstration.

## Evaluation criteria and grading scheme

| | | |
|---|---|---|
| **Analysis:** | | |
| Description of the semantic rules used in the implementation | ind 2.1 | 2 pts |
| Grammar augmented with the placing of the symbol table actions | ind 2.2 | 3 pts |
| **Design/implementation:** | | |
| Description/rationale of the overall structure of the solution and the roles of the individual components used in the applied solution. | ind 4.3 | 4 pts |
| Correct implementation according to original assignment statement. | ind 4.4 | 15 pts |
| Output of clear error messages (error description and location). | ind 4.4 | 3 pts |
| Output of symbol tables in separate stream or file. | ind 4.4 | 3 pts |
| Completeness of test cases. | ind 4.4 | 15 pts |
| **Use of tools:** | | |
| Description/justification of tools/libraries/techniques used in the analysis/implementation. | ind 5.2 | 2 pts |
| Successful/correct use of tools/libraries/techniques used in the analysis/implementation. | ind 5.1 | 3 pts |
| **Total** | | **50 pts** |