Reading Assignment. Read the following sections in the textbook:

- Chapter 5: Memory Management

Programming Homework. The goal of this assignment is to perform semantic analysis over the AST generated from your Parser implementation from HW-4. To complete the assignment, finish the following steps. Note that you should get started on this assignment early to give yourself enough time to ask questions (and receive a response) before the due date. If you wait until the last minute and have issues, you will likely have to turn your homework in with the late penalty. If you have questions, please ask them over Piazza or else via office hours with the instructor or graders.

1. *Download the HW-5 Starter Code.* Use the link provided in Piazza to accept the GitHub Classroom assignment for HW-5. When accepting the assignment, please be sure to link your account to your name in the roster (if you haven't done so already). Accepting the assignment will create a copy of the starter code and place it in a hw-4 repository for you. You will then need to clone this repository to your local machine. As always, be sure to frequently add, commit, and push your changes.

2. *Add your previous files HW to your repository.* You will need to copy some of your header and cpp files from HW-4 into your HW-5 repository (and working directory). Note that modified versions of `ast.h` and `mypl_exception.h` are provided in the starter code. A skeleton version of `type_checker.h` is also provided along with a working version of `symbol_table.h`. You must provide all other files needed (beyond those provided) to compile your program, to finish your assignment, and for your submission to be graded.

3. *Implement the TypeChecker class (in* `type_checker.h`*).* Please see the discussion and notes from class for more details. Note that this class implements the visitor interface from HW-4.

4. *Testing your implementation.* A set of basic test files are also provided in the `test` subdirectory. Note that you will need to generate additional tests to ensure your program works correctly. The graders will also be testing your code for error cases (in addition to the cases provided).

5. *Submit your code.* Be sure you have submitted all of your source code to your GitHub repo for the assignment (again, you should get into the habit of frequently adding, committing, and pushing your code). In addition to your source code, you must also submit each of your extended and any additional test files you used for testing. For error cases, submit a file that contains the error cases you tried (or else a script with the error cases themselves). Note that *all necessary files to compile and build* your program must be checked in to your repository. *If your homework doesn't compile, the graders won't be able to test it for correctness or completeness.*

Homework Submission. All homework must be submitted through GitHub Classroom. A link for each assignment will be posted on Piazza when the homework is assigned. Be sure all of your code is pushed by the due date (which you can double check for your repository using the GitHub website). Each programming assignment is worth **35 points**. The points are allocated based on the following.

- **Correct and Complete (25 points)**. Your code must correctly and completely do the requested tasks using the requested techniques. Note that for most assignments you will be provided a *partial* set of test cases to help you determine a *minimal level* of correctness. If your program fails any of the provided test cases you will only receive partial credit. *Note that passing the given test cases does not mean your work is complete nor correct.* Your assignment will also be graded with additional test cases (not provided to you) that will help the graders determine the extent of your solution and your final score. Note that for C++ code, correctness also implies properly handling the creation and deletion of dynamic memory (i.e., the absence of memory leaks).

- **Evidence and Quality of Testing (5 points)**. As part of your homework assignments you must develop additional test cases beyond those given to you to ensure your program is correct and complete. These test cases must be turned in with your assignment. You will be graded on the scope and quality of the additional test cases you provide.

- **Formatting and Comments (5 points)**. Your code must be formatted consistently and appropriately for the language used. For C++, you must follow the provided style guide (see the course webpage). You must also comment your code and test cases, which at a minimum must include a file heading (see examples provided), function comments, and meaningfully selected variable, class, and function names.

Type Inference Rules. The following rules are designed to help clarify type inference in `MyPL`. While the rules are more formal than a text-based description, some of the rules take liberties in terms of notation. We use the notation $e : t$ to say that expression e has type t. We assume `nil` has type `nil` (i.e., `nil` : `nil`). In general, we use e to denote expressions, t to denote types, x to denote variable names, s to denote structured type names, f to denote function names, and *block* to denote statement lists (e.g., the body of a function). Function types are represented as ordered lists of type with one type for each parameter and the last slot reserved for the return type. User-defined types are represented as maps (dictionaries) with keys for variable names and values for corresponding variable types.

$$\frac{\texttt{c}\text{ is a literal with type } t}{\Gamma \vdash \texttt{c} : t} \tag{1}$$

$$\frac{\Gamma \vdash e_1 : t \quad \Gamma \vdash e_2 : t \quad t \in \{\texttt{int}, \texttt{double}\} \quad \texttt{op} \in \{\texttt{+}, \texttt{-}, \texttt{*}, \backslash\}}{\Gamma \vdash e_1 \ \texttt{op} \ e_2 : t} \tag{2}$$

$$\frac{\Gamma \vdash e : t \quad t \in \{\texttt{int}, \texttt{double}\}}{\Gamma \vdash \texttt{neg} \ e : t} \tag{3}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 \ \% \ e_2 : \texttt{int}} \tag{4}$$

$$\frac{\Gamma \vdash e_1 : t_1 \quad \Gamma \vdash e_2 : t_2 \quad t_1, t_2 \in \{\texttt{char}, \texttt{string}\}}{\Gamma \vdash e_1 + e_2 : \texttt{string}} \tag{5}$$

$$\frac{\Gamma \vdash e : t \quad t \neq \texttt{nil}}{\Gamma, \ \texttt{var} \ x = e \ \vdash x : t} \tag{6}$$

$$\frac{\Gamma \vdash e : t \quad t \neq \texttt{nil}}{\Gamma, \ \texttt{var} \ x \texttt{:} \ t = e \ \vdash x : t} \tag{7}$$

$$\frac{\Gamma \vdash e : \texttt{nil}}{\Gamma, \ \texttt{var} \ x \texttt{:} \ t = e \ \vdash x : t} \tag{8}$$

$$\frac{\Gamma, \ \texttt{var} \ x_1 \ \tau_1 = e_1 \ \vdash x_1 : t_1 \quad \ldots \quad \Gamma, \ \texttt{var} \ x_n \ \tau_n = e_n \ \vdash x_n : t_n}{\Gamma, \ \texttt{type} \ s \ \texttt{var} \ x_1 \ \tau_1 = e_1 \ \ldots \ \texttt{var} \ x_n \ \tau_n = e_n \ \texttt{end} \ \vdash s : \{x_1 \to t_1, \ldots, x_n \to t_n\}} \ ^{\dagger} \tag{9}$$

$$\frac{\Gamma, \ \mathit{block} \ \vdash e : t}{\Gamma, \ \texttt{return} \ e \in \mathit{block} \ \vdash \mathit{block} : t} \tag{10}$$

$$\frac{}{\Gamma, \ \texttt{return} \ e \notin \mathit{block} \ \vdash \mathit{block} : \texttt{nil}} \tag{11}$$

$$\frac{\Gamma \vdash \mathit{block} : t' \quad t' \in \{t, \texttt{nil}\}}{\Gamma, \ \texttt{fun} \ t \ f(p_1 : t_1, \ \ldots, \ p_n : t_n) \ \mathit{block} \ \texttt{end} \ \vdash f : \{t_1, \ldots, t_n, t\}} \tag{12}$$

† where τ_i denotes an optional type expression "$\texttt{:} \ t_i$"

$$\frac{}{\Gamma \vdash \texttt{new}\ s : s} \tag{13}$$

$$\frac{\Gamma \vdash e : s \quad \Gamma \vdash s : \{\ldots, x_i \to t_i, \ldots\}}{\Gamma \vdash e.x_i : t_i} \tag{14}$$

$$\frac{\Gamma \vdash e_1 : t_1 \quad \ldots \quad \Gamma \vdash e_n : t_n \quad \Gamma \vdash f : \{t_1, \ldots, t_n, t\}}{\Gamma \vdash f(e_1, \ldots, e_n) : t} \tag{15}$$

$$\frac{\Gamma \vdash x : t}{\Gamma,\ x = e\ \vdash e : t\ \vee\ e : \texttt{nil}} \tag{16}$$

$$\frac{\Gamma \vdash e_1 : t \quad \Gamma \vdash e_2 : t \quad t \in \{\texttt{int}, \texttt{double}, \texttt{bool}, \texttt{string}, \texttt{char}, \texttt{nil}, s\}}{\Gamma \vdash : e_1\ \{\texttt{=}, \texttt{!=}\}\ e_2 : \texttt{bool}} \tag{17}$$

$$\frac{\Gamma \vdash e_1 : t \quad \Gamma \vdash e_2 : \texttt{nil} \quad t \in \{\texttt{int}, \texttt{double}, \texttt{bool}, \texttt{string}, \texttt{char}, s\}}{\Gamma \vdash : e_1\ \{\texttt{=}, \texttt{!=}\}\ e_2 : \texttt{bool}} \tag{18}$$

$$\frac{\Gamma \vdash e_1 : \texttt{nil} \quad \Gamma \vdash e_2 : t \quad t \in \{\texttt{int}, \texttt{double}, \texttt{bool}, \texttt{string}, \texttt{char}, s\}}{\Gamma \vdash : e_1\ \{\texttt{=}, \texttt{!=}\}\ e_2 : \texttt{bool}} \tag{19}$$

$$\frac{\Gamma \vdash e_1 : t \quad \Gamma \vdash e_2 : t \quad t \in \{\texttt{int}, \texttt{double}, \texttt{char}, \texttt{string}\}}{\Gamma \vdash : e_1\ \{\texttt{<}, \texttt{>}, \texttt{<=}, \texttt{>=}\}\ e_2 : \texttt{bool}} \tag{20}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \texttt{bool}}{\Gamma \vdash e_1\ \texttt{and}\ e_2 : \texttt{bool}} \tag{21}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \texttt{bool}}{\Gamma \vdash e_1\ \texttt{or}\ e_2 : \texttt{bool}} \tag{22}$$

$$\frac{\Gamma \vdash e : \texttt{bool}}{\Gamma \vdash \texttt{not}\ e : \texttt{bool}} \tag{23}$$

$$\frac{}{\Gamma, \texttt{while}\ e\ \texttt{do}\ \ldots\ \texttt{end} \vdash e : \texttt{bool}} \tag{24}$$

$$\frac{}{\Gamma, \texttt{if}\ e\ \texttt{then}\ \ldots\ \texttt{end} \vdash e : \texttt{bool}} \tag{25}$$

$$\frac{}{\Gamma, \texttt{if}\ \ldots\ \texttt{elif}\ e\ \texttt{then}\ \ldots\ \texttt{end} \vdash e : \texttt{bool}} \tag{26}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma, \texttt{for}\ x = e_1\ \texttt{to}\ e_2\ \texttt{do}\ \ldots\ \texttt{end} \vdash x : \texttt{int}} \tag{27}$$