CS 3276 Project #5
Semantic Analysis

The purpose of this lab is to perform semantic analysis (type checking) for Tiger programs. The purpose of the type checker is to determine the type for every expression in a program, and to ensure that expressions appear in places that are type consistent. The type checker will be a set of mutually recursive methods that will walk over the AST generated by your parser from Project #4. This lab will also require the creation of a symbol table to keep track of declared symbols in a program. This lab will complete the front end of the compiler which will now produce a type-safe AST which can then be manipulated by the optimization and/or code generation phases of the compiler.

A set of files is available to help you get started. The files are available on Brightspace in a compressed archive file `project5.tar.gz`. Download this file to your unix system and extract the files from the archive using tar (use the command: `tar -xvf project5.tar.gz --gunzip`). These files are meant to be added to your parser project from Project #4. If you were not able to complete Project #4, see me for a `tig_parse.y` file you can use. You should familiarize yourself with the content of these files before attempting changes. Included are the following:

- `absyn.[cpp|h]` -- abstract syntax classes, updated with type checking methods
- `env.[cpp|h]` -- symbol table (environment) files
- `parsetest.cpp` -- for testing the parser and type checker
- `Makefile` -- for compiling and linking all files
- `types.[cpp|h]` -- type declarations and constructors for our type environment
- `typecheck.[cpp|h]` -- the main controlling module for the type checker
- `parsetest-good` -- a fully working executable that you compare your parser against
- `test-all` and `test-cmp` -- scripts to help you test your parsers

For this project, you will only need to make modifications to `env.cpp` and `absyn.cpp`.

Base Assignment

For the base assignment, your compiler should be able to correctly type check all Tiger programs except those containing the following constructs:

- Record (structure) declarations and references
- Array declarations and references
- Mutually recursive type declarations
- Mutually recursive functions

Extensions

If you get your base compiler working, you may want to extend it to handle any of the above four items that were excluded from the base assignment. If you desire to extend your compiler, you are required to implement support for records and arrays before attempting mutually recursive types and functions.

Building and Testing

In order to successfully build your parser, you'll need to set things up properly. You should make a copy of all your files from Project #4 (make a copy of the entire directory) and then add all the files extracted from the tar file to that directory (make sure you do it in that order, so that newer versions of some files overwrite the older

versions). Once you have that, you should be able to go into the directory that contains the files and type "make". It should run bison and the C++ compiler to build everything. If you decide to use your own lexical analyzer from project #2, you will need to copy your `tig_scan.l` file into your working directory and modify the `Makefile` to invoke flex.

At this point, you should be able to run the parser on a test file. To run it on a file test.tig, you would give the command

```
./parsetest test.tig
```

Initially, the compiler will report that type checking failed. I recommend that you add code for a few structures at a time and test. You will have to generate your own short test cases that contain only the structures your type checker handles at that point. When your parser is completed, you can test it on the files Dr. Appel has supplied (also available on Brightspace).

Project Submission

Submit your projects on Brightspace via the assignment page. You should submit updated versions of the files: `env.cpp` and `absyn.cpp`. Also submit a `README` file that describes your project. The `README` file should state which parts and extensions of the project you completed, as well as describe any particularly interesting code or structures.

Grading

All students will start with a score of 80, an average grade (on a scale of 100). If you correctly implement the basic type checker, 80 will be your grade. Your score can move up if you correctly implement any of the extensions. Each extension is worth an additional 5 points. Your score can move down if your implementation is incorrect or has errors. Your score can also move down for poor programming style or the absence of a useful `README` file.

Additional Notes

1. You have been provided with copies of Chapter 5 from Appel's text book. This chapter will provide you with additional information on type checking a Tiger program. However, there are some differences between his description and what we are actually going to implement. The most important difference is that the type checking routines that we will write only return type information. In Appel's description, the functions return an expty object, which is a translated expression and its type information (even though the actual translated expression is produced later). Do not let this different confuse you as you read the Chapter 5 material.
2. The separation of type checking and translation is, in my mind, preferable since we can then add other steps between the two. For example, we can perform optimizing transformations on the AST after type checking but prior to translation. Thus we will make this section of our compiler responsible only for type checking, without including hooks for future translation.
3. Rather than writing our type checker in a functional style (as done by Appel), we will write our type checker in an object-oriented style. Whereas Appel's code would have to test what kind of AST node was being operated on, we will provide each node with the ability to type check itself. Note that in either case, the code to be written is the same – it is only how the code is packaged that is different.
4. I have included a new type: Ty_Error. If you encounter a type error while type checking, return a Ty_Error() type rather than a null pointer. If a Ty_Error type is returned to our driver routine, then it will know that type checking failed and that the compilation process should be halted. Note: in your type checking routines, after you make a function call to type check a subcomponent, you should immediately check whether the return type is Ty_Error.

5. I have created some wrapper functions around the environment processing code to separate the type environment from the value environment. Hopefully it will make using the environment code a bit easier for you.
6. You need to add code to `env.cpp` to seed the environments with the predefined types and library functions. Be sure to include the `printint` and `readint` library functions, which were described in the Project #1 specification.
7. Completing this project is very different than completing prior projects. In prior projects, after you handled a couple constructs the remaining construct went much more quickly and easily. In this project, all constructs are unique and require different handling. Because of that, this project can be much more time consuming than prior projects.

Good luck!

Quote from a prior student's README file (prior to conversion to OOP code):

I only had the opportunity to attempt part of the assignment, and I'm sure that it's not working, though it does compile. The work I did do is located in typecheckExp.cpp. Time was definitely the issue here as I think I was just beginning to work many of the bugs out and get rolling on this thing. I should have headed your warning to start early...

And from another student (who saw the warning above):

As you can probably tell from my program, I woefully underestimated the amount of time the project would take. The program doesn't work in any capacity, and it doesn't even compile. I did do work in env.cpp and typecheckExp.cpp, but the other files have been more or less untouched.

Sorry, this was my mistake; I expected the time requirement for this project to be around that of Project 4, but I was very wrong. I didn't properly manage my time over the last few weeks, and this is unfortunately the result.

And yet another student:

I wish I didn't ignore the warning about starting early.

And yet another student:

Contains only the base assignment, and even that hasn't been completed and debugged. I shouldn't have underestimated how much time and effort this would take.