# CSCI 4160 Project 6

Due: see class calendar

**Description:**

In this project, you are required to detect all static semantic errors for a Tiger program.

.

**What to do in this project?**

Your major task in this project is to provide implementation for most member functions of class TypeChecking to detect all static semantic errors. The member functions to be implemented are specified by the comments. No other files need to be modified.

When you implement each member function, please refer to the slides at
http://www.cs.mtsu.edu/~zdong/4160/public/slides/TypeChecking.pdf

**Classes in this project**

The new classes introduced in this project are types::Type and its descendants, which are used to define types associated to expressions and type definitions in Tiger language

- Class Type: the abstract base class. It defines two virtual member functions
  - types::Type* actual(): It returns the real data type. See NAME type.
  - Bool coerceTo(const Type *t): it returns true if t can be converted to current type, otherwise returns false. This function is needed in order to check type compatibility.
- Class ARRAY: represent array data type.
- Class RECORD: represent record data type.
- Class FUNCTION: represent function signature.
- Class INT: represent integer data type
- Class STRING: represent string data type
- Class NIL: represent NIL constant
- Class VOID: represent void type
- Class NAME: represent the alias name of an existing data type. For example, the statement below defines a new data type in Tiger language:
  *type money = int*
  Then, a NAME object can be created to represent the money type.
  NAME *n = new NAME("money);
  n->bind( new types::INT() );
  In this case, the statement n->actual() will return a pointer to types::INT object, which is the real data type of money.

**Tips for the project**

- When should I use the member function Type::actual()?
  ANSWER: Everytime when you return a variable (say t) of (type::Type *) in TypeChecking::visit function, you should return t->actual() instead of t. For example, see TypeChecking::visit(const VarExp *) in TypeChecking.cpp file provided by the instructor .

- Assume we have the declaration: type::Type *t; after some assignment statement to variable t; how can we check if it actually points to an object of some type, say FUNCTION?
  ANSWER: Use dynamic_cast. For example, if the Boolean expression below is true, it means t points to a FUNCTION object.

```
dynamic_cast<const types::FUNCTION *>(t) != NULL
```

- Assume there is variable t which is declared as type::Type *. If you know t points to a specific type of object such as FUNCTION, how to convert t to type::FUNCTION* so that methods of FUNCTION can be used?
  ANSWER: Use dynamic_cast.
  Types::FUNCTION* fp = `dynamic_cast<types::FUNCTION *>(t);`

- Assume we have declarations: type::Type *t1, *t2; after some assignment statement to both variables, how to check if these two types are compatible?
  ANSWER: use coerceTo function. If t1->coerceTo(t2) is true, it means type represented by t2 can be coerced to the type represented by t1. If t2->coerceTo(t1) is true, it means type represented by t1 can be coerced to the type represented by t2.

- Each TypeChecking::visit function returns a pointer to types::Type. So if an expression contains a semantic error, what should be returned? For example: expression 1 + "a string" is semantically wrong, and if it is passed to TypeChecking::visit(const OpExp * e), what is supposed to return?
  ANSWER: If an expression contains a semantic error (undefined variable used, type dones't match), always treat the type of the expression as INT.

The following files are provided by the instructor:

- TypeCheckingProject folder. This contains a sample Visual Studio 2010 project.
- Description6.pdf: this file
- Rubric6.doc: the rubric used to grade this assignment.
- example.txt. sample output

**How to submit**

Once you have finished, submit the project in the following way:

- Copy the file projects/project5/rubric5.doc from the class repository to the project5 folder in your local repository of project5. Edit the file to put your name.
- Commit the whole project5 folder to your local repository.
- Push all the changes to master repository on ranger.
- **Any commit of the project after the deadline is considered as cheating. If this happens, the latest version before the deadline will be graded, and you may receive up to 50 points deduction.**

You can check your overall grade by pull rubric3.doc from the master repository after the notice from the instructor.