CSCI 4160 Project 6

Due: see class calendar

Description:

This is a team project that build on the previous project. In this project, you are required to perform full type checking for Tiger language.

.

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 class introduced in this project is types::Type and its descendants, which are used to define types associated to variables 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 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 converted to the type represented by t1. If t2.coerceTo(t1) is true, it means type represented by t1 can be converted 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 doesn't match), always treat the type of the expression as INT.

Test Report:

In this project, implementation of the project and test case design can be performed simultaneously. Don't wait too long to design test cases. In this project, please provide a tiger program called: test.tig to cover all of the following scenarios:

- Type checking on SubscriptVar like: lvalue[exp]
  - lvalue is not an array type (test case 1)
  - exp is not an integer (test case 2)
- Type checking on OpExp like: exp1 operator exp2
  - If operator is +, -, *, or /
    - exp1 is not an integer (test case 3)
    - exp2 is not an integer (test case 4)
  - If operator is =, or <>
    - Types of exp1 and exp2 doesn't match (test case 5)
    - Exp1 is NIL but exp2 is not an array or record (test case 6)
    - Both exp1 and exp2 are NIL (test case 7)
  - If operator is <, <=, >, or >=
    - One operand is neither INT nor STRING (test case 8)
    - One operand is INT, and the other one is STRING (test case 9)
- Type checking on CallExp like: fname(exp1, exp2, …, expn)
  - fname is not defined (test case 10)
  - fname is defined as a variable (test case 11)
  - too many arguments (test case 12)
  - less arguments than required (test case 13)
  - one expression has wrong data type (test case 14)
- Type checking on AssignExp like: lvalue := exp
  - Types of lvalue and exp don't match (test case 15)
- Type checking on IfExp like: if exp1 then exp2 else exp3
  - exp1 is not an integer (test case 16)
  - if there is no else-clause, exp2 is not VOID type (test case 17)
  - if there is else-clause, types of exp2 and exp3 don't match (test case 18)
- Type checking on WhileExp
  - The body of the loop is not VOID type (test case 19)
- Type checking on ForExp like: for varname := exp1 to exp2 do exp3
  - exp1 is not an integer expression (test case 20)
  - exp2 is not an integer expression (test case 21)
  - exp3 is not VOID type (test case 22)
- Type checking on ArrayExp like: array_typename [exp1] of exp2
  - array_typename is not defined (test case 23)
  - array_typename is defined, but not an array type (test case 24)
  - exp1 is not an integer expression (test case 25)
  - Type of exp2 doesn't match the definition of array_typename (test case 26)
- Type checking on VarDec like: var vname : TypeName := exp (test case 27)
  - Type of exp doesn't match TypeName (test case 28)

- If TypeName is omitted, and exp is NIL. (test case 29)

Please put the test report, and final version of test.tig in the project6/testing folder of team repository. Please put the alpha version of test.tig in the project6/alpha folder.

Important Dates and Deadlines:

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
		7 **Project assigned**	**8**	**9** **1st time log report due**	**10**	**11**
12	**13**	**14** **Test.tig and alpha version due** **2nd time log report due**	**15**	**16** **Test report due**	**17**	**18** **Project Due**
19	**20**	**21**	**22**	**23**		

Notes:

- All reports/files are due at 11:30pm on the due day.
- Each team is required to submit 2 time log reports.

Instructor provided files in the class repository

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
- Testreport.doc: format of testing report for this project

How to submit

1. Once you have finished, submit the following files to D2L dropbox:
   - test case
   - test report
   - time log
   - alpha version of TypeChecking.cpp
   - final version of TypeChecking.cpp