Arizona State University
Computer Science and Engineering
CSE 440/598
Compiler Construction I
Fall 2012

Project 1: Semantic Checking

Table of Contents

1. Introduction
2. Programming Language: Mini Object Pascal
3. Requirements
4. Submission Guidelines

1. Introduction

The goal of this project is to get you introduced to semantic checking using scanner and parse generator tools such as lex and yacc. You will be doing type checking as well as other semantic checking of a small objected-oriented language derived from Pascal. The next section introduces the language and the section after that gives the details of the requirements. Finally, the last section gives the submission guidelines.

1. Programming Language: Mini Object Pascal

The programming language we will be considering is a simplified version of Pascal augmented with classes and inheritance. What follows is the description of the language, especially the parts that are not found in Pascal. For Pascal semantics, you are referred to the ANSI Pascal Language manual except as they are superseded in what follows. You are also provided with a detailed test suite that includes the various features for which your program will be tested. In practice, this can be used instead of the Pascal specifications.

Classes

The language has classes that can be declared using the "class" reserved word. Single inheritance is supported using the "extends" reserved word.

Fields of a Class.

A class cannot share any field names with any of its ancestors. It is not enough for the set of the names of fields to be disjoint from that of the parent class. It has to be disjoint from the sets of names of all the ancestor classes.

Private vs. Public.

All fields of a class are assumed to be public.

Methods.

A class can have a number of methods. Different methods have different names.

Constructor

Every class can have a constructor method whose name is the same as the name of the class.

Main class.

The program has a main class whose name is identical to the program name. The constructor of the main class gets invoked first when a progran is executed. The main class constructor takes no argument and the language does not support command line arguments.

Recursive Constructs.

The language allows recursive constructs. So, a field of a class A can be of type A. This allows the definition of recursive structures but witout the use of expicit pointers.

new.

Instances of a class can be created using the new construct which returns a "pointer" to a object instance of the class. The memory for the new object is allocated on the heap.

this.

"this" is used to disembiguate reference between a method's local variable and a class field. It is used to the fielf of the class in which it appears. It is not supported in the grammar and should be implemented by the semantic rules.

Types

In addition to the basic "integer", "real", and "boolean" types, the language supports unnamed single dimensional array types (multidimensional arrays can be declared as arrays of arrays ...) as well as class types. For the purposes of type checking, class types can be treated as named record types in Pascal.

For the basic types, type assignment compatibility rules **are as specified in the ANSI reference**. For other types, here are the rules:

- Arrays are compatible if their ranges are compatible (same number of elements) and their elements have compatible types.
- If O1 is of type A and O2 is of type B, and A extends B (or is a descendent of B), then O1 can be assigned to O2.
- A and B are compatible classes if their fields, in the order they are listed, are compatible or are the same type (this defines structural equivalence). Objects whose types are compatible can be assigned to each other and can be passed to functions whose parameters have compatible types.

```
Note: the grammar does not allow passing arrays directly to functions.
Passing arrays to functions would require them to be encapsulated inside
an object. Also, the grammar does not allow functions without parameters.
```

2. Requirements

In the project1 directory, you will find links for a grammar for the simplified and augmented pascal. Also, you will find the lex file for pascal as well as stubs to functions that you might want to write and corresponding header files. There is an extensive set of test cases as well as a script to run the test cases. Passing all the test cases is required, but different test cases will also be used to test your programs.

You are asked to write a translation scheme to check input programs for semantic errors. You can assume that input programs have no syntax errors. That is, you are asked to write the code that will do semantic checking. I will spend next week going over the specifications and what needs to be done.

With your submission, you should list all errors that you detect and those errors that you do not detect.

Errors to be detected include, but are not limited to:

1. valid declaration
2. type checking (this includes all basic types, arrays, classes,..)
3. array index checking
4. function declaration/definition/call
5. undeclared variables
6. classes declarations/inheritance/usage

The test suite contains an extensive list of errors for which your programs will be tested. Do not be intimidated by the number of the test cases. Most of these files are very small files that illustrate simple features to be tested.

If you have time:

- You should try evaluation of constant expressions and range checking for array indices that are constant expressions.
- You should handle syntax errors

There is no expectation that you will do the last two, but it would be nice to do that.

Graduate Students: graduate students are expected to do all the semantic checking. In addition, you should do constant expression evaluation and handling some syntax errors.
Undergraduate Students: You can handle syntax errors for bonus points, but you should finish the semantic checking first.

This project is somewhat involved. You should start early and work on it regularly. You should familiarize yourself with the grammar and then have a good design for your symbol table. If you have a good symbol table and basic functions to manipulate it, your work will be greatly simplified.

Submission Guidelines

Here is what and how you should submit your project:

1. You should have a large comment explaining what cases you handle and what cases you do not handle. Also, as always, I expect that the code is well documented. In particular, you should pay attention to the documentation of your data structures.
2. You should provide compilation instructions for your project. The project will be compiled and tested on general (`general.asu.edu`) or the stats (`stats.asu.edu`) machine. Probably stats will work best. It is your responsibility to make sure that it runs and compile correctly on one of the two platforms.
3. Bring a printout of the md5 of your files to class on the day the project is due. This is absolutely necessary. Otherwise, your submission is late.

4. Include the names of the group members on your submission and in all files you submit (as a comment). You should state clearly which group member did which parts.
5. Make sure that you test your project on general or stats.
6. Upload your files in one tar file to blackboard