K31 Compilers

Spring Semester 2021

May 9, 2021

Course Project

Design and implementation of a compiler for the MiniJava language (a small subset of Java)

To implement the compiler you will use the tools JavaCC and JTB

The implementation for phases 2 and 3 of the project will be done in Java utilizing the visitor pattern

Homework	Description	Deadline
1	Implementation of a LL(1) parser for a simple calculator and a translator to Java for a simple language	18/4/2021
2	Semantic Check (MiniJava)	16/05/2021

Homework 2 – MiniJava Static Checking (Semantic Analysis)

This homework introduces your semester project, which consists of building a compiler for MiniJava, a subset of Java. MiniJava is designed so that its programs can be compiled by a full Java compiler like javac.

Here is a partial, textual description of the language. Much of it **can be safely ignored** (most things are well defined in the grammar or derived from the requirement that each MiniJava program is also a Java program):

- MiniJava is fully object-oriented, like Java. It does not allow global functions, only classes, fields and methods. The basic types are int, boolean, and int [] which is an array of int. You can build classes that contain fields of these basic types or of other classes. Classes contain methods with arguments of basic or class types, etc.
- MiniJava supports single inheritance but not interfaces. It does not support function overloading, which means that each method name must be unique. In addition, all methods are inherently polymorphic (i.e., “virtual” in C++ terminology). This means that foo can be defined in a subclass if it has the same return type and argument types (ordered) as in the parent, but it is an error if it exists with other argument types or return type in the parent. Also all methods must have a return type–there are no void methods. Fields in the base and derived class are allowed to have the same names, and are essentially different fields.

- All MiniJava methods are “public” and all fields “protected”. A class method cannot access fields of another class, with the exception of its superclasses. Methods are visible, however. A class’s own methods can be called via “this”. E.g., this.foo(5) calls the object’s own foo method, a.foo(5) calls the foo method of object a. Local variables are defined only at the beginning of a method. A name cannot be repeated in local variables (of the same method) and cannot be repeated in fields (of the same class). A local variable x shadows a field x of the surrounding class.

- In MiniJava, constructors and destructors are not defined. The new operator calls a default void constructor. In addition, there are no inner classes and there are no static methods or fields. By exception, the pseudo-static method “main” is handled specially in the grammar. A MiniJava program is a file that begins with a special class that contains the main method and specific arguments that are not used. The special class has no fields. After it, other classes are defined that can have fields and methods.
  Notably, an A class can contain a field of type B, where B is defined later in the file. But when we have “class B extends A”, A must be defined before B. As you’ll notice in the grammar, MiniJava offers very simple ways to construct expressions and only allows < comparisons. There are no lists of operations, e.g., 1 + 2 + 3, but a method call on one object may be used as an argument for another method call. In terms of logical operators, MiniJava allows the logical and (“&&”) and the logical not (“!”). For int arrays, the assignment and [] operators are allowed, as well as the a.length expression, which returns the size of array a. We have “while” and “if” code blocks. The latter are always followed by an “else”. Finally, the assignment “A a = new B();” when B extends A is correct, and the same applies when a method expects a parameter of type A and a B instance is given instead.

- You should only accept expressions of type int as the argument of the PrintStatement.

The MiniJava grammar in BNF can be downloaded here. You can make small changes to grammar, but you must accept everything that MiniJava accepts and reject anything that is rejected by the full Java language. Making changes is not recommended because it will make your job harder in subsequent homework assignments. Normally you won’t need to touch the grammar.

The MiniJava grammar in JavaCC form is here. You will use the JTB tool to convert it into a grammar that produces class hierarchies. Then you will write one or more visitors who will take control over the MiniJava input file and will tell whether it is semantically correct, or will print an error message. It isn’t necessary for the compiler to report precisely what error it encountered and compilation can end at the first error. But you should not miss errors or report errors in correct programs.

The visitors you will build should be subclasses of the visitors generated by JTB, but they may also contain methods and fields to hold information during static checking, to transfer information from one visitor to the next, etc. In the end, you will have a Main class that runs the semantic analysis initiating the parser that was produced by

JavaCC and executing the visitors you wrote. You will turn in your grammar file, if you have made changes, otherwise just the code produced by JavaCC and JTB alongside your own classes that implement the visitors, etc. and a Main. The Main should parse and statically check all the MiniJava files that are given as arguments.

Also, for every MiniJava file, your program should store and print some useful data for every class such as the names and the offsets of every field and method this class contains. For MiniJava we have only three types of fields (int, boolean and pointers). Ints are stored in 4 bytes, booleans in 1 byte and pointers in 8 bytes (we consider functions and int arrays as pointers). Corresponding offsets are shown in the example below:

Input:

```
class A{
    int i;
    boolean flag;
    int j;
    public int foo() {}
    public boolean fa() {}
}

class B extends A{
    A type;
    int k;
    public int foo() {}
    public boolean bla() {}
}
```

Output:

```
A.i : 0
A.flag : 4
A.j : 5
A.foo : 0
A.fa: 8
B.type : 9
B.k : 17
B.bla : 16
```

There will be a tutorial for JavaCC and JTB. You can use these files as MiniJava examples and to test your program. Obviously you are free to make up your own files, however the homework will be graded purely on how your compiler performs on all the files we will test it against (both the above sample files and others). You can share ideas and test files, but you are not allowed to share code.

Your program should run as follows:

```
java [MainClassName] [file1] [file2] ... [fileN]
```

That is, your program must perform semantic analysis on all files given as arguments. May the Force be with you!