CSE304 : Compiler Design : Semantic Analysis and Intermediate Code Generation

Phase IV : Semantic Analysis and Intermediate Code Generation

Due: 28-May-2019

Overview

The fourth phase of developing a compiler for ***rascl*** is writing a semantic analyzer and generating intermediate code. How you implement this depends on the type of parser you wrote. For a recursive decent parser, it involves mainly adding new code to the current functions. The functions also must accept parameters for 'inherited attributes' and they must return values that are 'synthesized attrbutes' for the associated non-terminal.

Semantic Analysis

The semantic analysis must include type checking and the generation of conversion operators (type coercion) in order to generate operations the produce the correct results. Type coercion follows rules similar to C. That is, integer operations are used between integer operands, float operations are used for float operands. In the case of a mathematical operation between an integer and a float, the integer must be 'promoted' (that is converted) to a float and a floating point operation must be used.

Intermediate code format

The Intermediate code you generate will be three address code in the form of quadruples. They will have an operator, two source operands and a result field. Operands for arithmetic operations are always registers. There are an infinite number of registers. Temporary register names for integer values will be T# where '#' is an integer. For floating point numbers, temporary names should be of the form FT# where # is an integer. RIF is described completely in The RIF_Developer_Guide provided.

Final Results

The semantic analysis and intermediate code generation code should be integrated with your other components (symbol table manager, lexical analyzer, and parser). The resulting

compiler should be able to generate intermediate code correctly for any of the Rascl programs in the test suite I give you.
I will provide examples of code generated for the test suite. My compiler generates some extra comment statements that indicate where different code starts like *if* statements, *while* statements, *assignment* statements, etc. Your code need not generate these comments (although you may want to try to write these into your code as a help to debug your code generation!)

Testing

I will provide a moderate set of test code for you to compile. I may use additional tests when I grade your intermediate code generation. Here are a couple of short examples of what RASCL code looks like:

=====
T00_rascl_test_exprs1.rsc
=====

```
int a, b;
float c, d;
{
  a = 5;
  b = a + 2 * (-b + -a);
}
```

=====
T41_rascl_test_io.rsc
=====

```
float a;
float b;

{
   a = 3.7;
   b = a * 2.0;
   print b;
}
```

=====
T62_rascl_test_all_features3.rsc
=====

```
int b, c;
int d;
float e;
float f;

{
  b = 1;
```

```
 c = 10;
 e = 5.0;
 f = e * c;
 if (c > b)
 {
  d = 5;
  c = c + -b;
 }
 else
 {
  while (b < 5) {
     c = -d * b;
        b = b + 1;
  };
 };
 print c;
}
```

The minimal set of test cases is attached to the dropbox as the file *basic_rascl_tests.zip.*

Test Output

The compiler will accept a file with an extension of .rsc and produce the intermediate code in a file with an extension of .rso. The output file will have code compliant with the *RIF Developer Guide.*

Submission Process:

The deliverables for this part of the project are:

1. Source code to the parser
2. A script that builds and links the code (if it is in a compiled language like C).
3. A script that takes a file name of a file to compile from its command line and runs the parser giving it that source (.rsc) file.
4. An output file (.rso) for each test compiled.

Zip (or tar/gzip) the source code, the scripts, and the log files for each .rsc file in the test suite provided.

Upload the ZIP or .tgz file to the *'Phase IV: Semantic Analysis and Intermediate Code Generation'* dropbox.

***** Late Assignments will NOT be accepted, and any assignments emailed to the professor will also NOT be accepted. Start working early and plan to submit working code on time. *****