lab session 5: compiler construction

January 11, 2016

This lab session of the course *Compiler Construction* focuses on implementating some optimizations on *intermediate representation (IR)* code. A scanner and parser are already available, as well as some helper code. In fact, probbably the only file that you will change is the file `main.c`, and you are not likely to change any of the other files (although allowed).

On Nestor you can find the file (`iroptimizer.tar`. Download it, extract it, and compile the code using the commands:

```
tar xvf iroptimizer
cd iroptimizer
make
```

Exercise 1: Copy propagation and Constant propagation

The first exercise consists of the implementation of copy/constant propagation. The input and output are files consisting of *quadruples*, i.e. statements of the form:

- `lhs=var;` where `lhs` and `rhs` are variables.
- `lhs=const;` where `lhs` is a variable, and `const` a positive integer constant.
- `lhs=opnd1 op opnd2;` where `lhs` is a variable, `op` an operator (+,-,*,/), and `opnd1` and `opnd2` are variables or positive integer constants.

For example, the following input (left hand side, available in the file `copyprop.in`) should produce the following output (right hand side):

```
b=a;                          b=a;
c=b+1;                        c=a+1;
d=b;                          d=a;
b=d+c;                        b=a+c;
b=d;                          b=a;
a=21;                         a=21;
b=2;                          b=2;
c=a*b;                        c=42;
d=a+a;                        d=42
e=d+e;                        e=42+e;
```

Note that some statements of the form `lhs = opnd1 op opnd2` can change in the form `lhs=const` as a result of constant propagation and arithmetic.

Study the code in the file `main.c` and extend the code in the routine `processQuadruple(quadruple quad)` such that it performs copy/constant propagation. The pseudo algorithms can be found in the lecture slides.

Exercise 2: common subexpression elimination

In this exercise you will implement common subexpression elimination (see lecture slides). We will introduce temporary variables with a special naming convention (the parser/scanner combination already accepts them): an underscore followed by a number. Note that it is perfeclty fine to replace `a+b` by `b+a`, but this is clearly not the case for the operations subtraction and division.
An example input/output session is the following:

```
c = a+b;          _1=a+b; c=_1;
d = m*n;          _2=m*n; d=_2;
e = b+d;          _3=b+d; e=_3;
f = a+b;          f=_1;
h = b+a;          h=_1;
a = j+a;          a=j+a;
k = m*n;          k=_2;
j = b+d;          j=_3;
p = q+r;          _4=q+r; p=_4;
```

Exercise 3: dead code elimination

The last exercise is about dead code elimination. This process removes assignments to variables that are redundant. For example, the two assignments `a=b; a=42;` can clearly be reduced to `a=42;`. Moreover, as the result of common subexpression elimination, many redundant variables are introduced. For example, in the above example (last statement in exercise 2), the temporary variable _4 was introduced, but it was not re-used in subsequent subexpressions. So, this introduction should be undone as well. In other words, the statements `_4=q+r; p=_4;` should be merged again to `p=q+r;`.
For example, the following input (left) should produce the following output (right):

```
_0=y+2;
a=_0;          a=y+2;
z=x+w;
x=a;           x=a;
_1=
z=b+c;         z=b+c;
b=a;           b=a;
_4=q+r;
p=_4;          p=q+r;
```

Implement this optimization. After you implemented all three exercises, experiment a bit with placing (iteratively) these three optimizations in cascade. Also experiment a bit with different orderings. What is your conclusion?

Deadline: Thursday January 28th (after exam).
Submit your code to Justitia (at `http://justitia.housing.rug.nl`). Note that justitia will always accept your code, even if it is incorrect! In this course, we use Justitia just as a dropbox for your solutions which can be tested manually by the teaching assistants. You do not have to write a report for this lab session.

Grading: The grading of your implementation will be based on:

- Correctness, Programming quality, efficiency, etc.: 50%
- Tests performed by the teaching assistants: 50%

- Penalty for missing deadlines: -1 grade point per day.