CS 380C Compilers — Spring 2014	Programming Assignment #2
Local Optimizations	Due January 31, 2014

In this assignment, you'll learn how LLVM can be used to make your programs run faster. In particular, you will use LLVM to learn interesting properties about your program and to perform local optimizations.

1 Logistics

Any clarifications and corrections to this assignment will be posted on the class discussion board on Piazza.

2 Meet The Functions

A compiler pass is the standard mechanism for analyzing and optimizing programs, and your first task is to implement an analysis pass. In general, analysis passes are used to ensure that an optimization pass is safe, ie, it does not alter the meaning of the program, but for this assignment, you will implement a simple `FunctionInfo` pass to learn interesting properties about the functions in a program. Please name your pass `FunctionInfo` and make sure that the `opt` command-line option for it is `-function-info`. Your pass should report the following information about all functions that are used in a program:

1. Name.
2. Number of arguments.
3. Number of call sites (i.e. locations where this function is called). All call sites in the module should be counted.
4. Number of basic blocks.
5. Number of instructions.

For each function, print a line (to standard error) in the following format:

```
functionName: arguments=2, call sites=1, basic blocks=5, instructions=50
```

We recommend that you debug your pass with complex source files, as you can imagine grading will be done with complex programs. Feel free to turn in your additional testing source files in a separate directory, along with your source code.

3 Optimize The Block

Now that you are an expert in creating LLVM passes, it's time to write a local optimization pass that improves program execution time. There are many types of local optimizations, but we will keep things simple and focus on algebraic optimizations. Specifically, you will implement the following local optimizations:

1. Algebraic identities on : e.g, $x + 0 = 0 + x = x$ and $x/x = 1$
2. Constant folding: e.g, $2 * 4 = 8$
3. Strength reductions: e.g, $2 * x = (x + x)$ or $(x << 1)$

You should support all forms of multiplication, division, addition, and subtraction over both 32 and 64 bit integers (i32 and i64 in LLVM). Make sure to support both signed and unsigned operations correctly. Use the provided test files for inspiration to figure out what algebraic and strength reduction optimizations might be useful. (*hint*: If you make your optimization framework general it should be very easy to support all the various required cases without too much duplication of code.)

4 Implementation Details

You should create a new LLVM pass named LocalOpts.cpp (with name `my-local-opts` inside LLVM for `opt` commandline parameters). This should be in a directory `assignment2`. Since the `clang` optimizer performs local optimizations, your LocalOpts pass should be run on unoptimized LLVM bitcode. Unoptimized bitcode of loop.c can be prepared as follows:

```
clang -O0 -emit-llvm -c loop.c
```

You can then run your pass as follows:

```
opt -load llvm/Debug/lib/LocalOpts.so -my-local-opts loop.bc -o loop-opt.bc
```

And you can view the resulting LL assembly using the following:

```
llvm-dis loop-opt.bc
```

In addition to transforming the bitcode, your pass should also print a summary of the optimizations that it performed: e.g., how many constants were folded. To help you debug your code, we will provide toy source files that have unrealistic amounts of local optimization opportunities. Of course, we recommend that you also test your pass on more realistic programs.

5 Submission

Use the `turnin` program to submit a single `tar.gz` or `tar.bz2` file that contains your source code in a directory with a `Makefile` that will build it. For this project, please name the directory `assignment2`. The turn-in command will be: `turnin --submit amp cs380c_assgn2 assignment2.tar.gz`. Make sure that your code builds correctly on the provided virtual machine and does not depend on any files outside the code that you submit.

As with the first assignment, the build system is not the subject of this class so feel free to help each other with it or post useful variants of the `Makefile`.

This assignment is due on Friday January 31st at 5:00pm.

Acknowledgments. This assignment was originally created by Todd Mowry and then modified by Arthur Peters.