CS 577 Homework 1 – Extending a bytecode interpreter – due 4pm, Monday, April 18, 2005

On the course web page, you'll find C code for an interpreter for a small subset of JVM instructions. The Java subset supported includes integer arithmetic, static methods, and a minimal set of output facilities. The interpreter is defined by a set of C files (`interp.c`, `class.[ch]`, `basics.[ch]`, `bytecode.h`) and a `makefile`, which generates an executable `interp`. This can be invoked just like the usual `java` interpreter on a single class file , but fails on programs outside the supported subset.

Your assignment is to extend `interp` to deal with integer arrays. This will involve adding support for about twelve new JVM instructions, and testing your modifications.

Details

The subset handled by the existing interpreter should be sufficient to execute simple integer programs involving `static` methods within a single class. (I may have left out one or two rarely generated instructions; if so, feel free to implement them!) The only way to do output (supported by a nasty special-case hack) is via `System.out.print(`x`)` where x is an integer or a string. Still, this is enough to write test cases that display their results, and will run under the ordinary JVM as well as under this interpreter. The interpreter generally issues an "unimplemented" message about any instruction it can't cope with.

To handle arrays, you'll need to add cases to handle these additional instructions: `ACONST_NULL`, `ALOAD` (and its variants), `ARETURN`, `ARRAYLENGTH`, `ASTORE` (and its variants), `IALOAD`, `IASTORE`, `IF_ACMPEQ`, `IF_ACMPNE`, `IFNONNULL`, `IFNULL`, `NEWARRAY`. Most of the necessary code can be copied – or even reused without copying – from the existing support for integers. The interesting cases are `NEWARRAY`, `ARRAYLENGTH`, `IALOAD`, and `IASTORE`. To see how these instructions are used by real Java programs, write `.java` testcases and use `javap -c` to examine the corresponding bytecode. Make sure you write test cases that exercise all these instructions!

You need to arrange to "raise" the following built-in exceptions when appropriate:

```
NegativeArraySize
NullPointer
ArrayIndexOutOfBounds
```

If one of these exceptional conditions occurs, your generated code should print a line to `stdout` with the message "Uncaught Exception *name*" where *name* is one of the above, and exit immediately. Don't try to implement real exceptions or exception handling! Make sure you write test cases that provoke all these exceptions.

Represent an n-element array by a $(n + 1)$-word heap block, with the length stored in the first word. To allocate a heap block, use the `alloc` function defined in `basics`. Don't worry about deallocation or garbage collection! (If you want, it is easy to link in the Boehm conservative garbage collector.)

How to submit your homework.

Submit the homework *on paper* at the beginning of class on the due date. You should submit:

- Your modified version of `interp.c`.

- Evidence that your revised interpreter works, in the form of a set of test programs that exercise integer arrays (including exception-causing cases) together with the interpreter's run results.
- A brief explanation of how the interpreter passes arguments, and why this approach works. A diagram may be quite helpful!