BAWK, A BINARY AWK

Kevin M. Graney
kmg2165@columbia.edu

Programming Languages and Translators
COMS W4115, Spring 2013, Professor Stephen Edwards

Contents

Listings

Chapter 1

Introduction

1.1 Overview

The bawk language, who's name is derived from 'binary awk', is intended to be a small, special-purpose language for the parsing of formatted binary files. In the spirit of awk, bawk will match rules in binary files and extract data based on these rules. Bawk will solve the problem of quickly decoding a binary file of a known format to extract information much the way awk solves this same problem for formatted text files.

The bawk interpreter will take two inputs for execution: (1) text in the bawk langauge, and (2) binary data to run the bawk program on. This workflow will be similar to that of a basic awk workflow, and the language itself is inspired by awk, with the same basic program construct. That is, a program will be a series of patterns and statements, with the statements executing when their respective pattern matches.

1.2 Sample program

A representative bawk program is shown below. This program operates on PNG format image files and will print two numbers, newline separated. The first number is the width of the image and the second number is the height.

Listing 1.1: A simple bawk program

```
1 /* print the size of a PNG file */
2
3 /89 "PNG" 0d 0a 1a 0a/ {
4    /length:uint4 "IHDR"/ {
5       /width:uint4 height:uint4/ {
6         print(width);
7         print(height);
8       }
9    }
10 }
```

Chapter 2

Language tutorial

The bawk language is based on pattern matching bytes and extracting values from binary format data files.

2.1 A sample program

The sample program shown in §1.2 illustrates how to print the width and height of a given PNG image file using bawk. This is a perfect example of an application of the bawk language: a simple binary format file for which the extraction of integer data is desired. We will use this sample program in the next section to show how bawk is executed.

2.2 Executing bawk files

To execute a bawk file, call `bawk` with a `-e` flag and specify the file on which to operate. The bawk code is then to be provided via standard input.

```
./bawk -e [binary filename] < [bawk filename]
```

This command will compile and run the bawk program over the given binary file. The `-e` flag is optional and may be ommitted. (The default operation is to execute the program if no flags are given.)

To execute the sample program, named `tests/test-util1.bawk`, we run bawk on a sample file, `tests/lichtenstein.png`, and see that the height and width of this file are both 512 pixels.

```
$ ./bawk -e tests/lichtenstein.png < tests/test-useful1.bawk
512
512
```

To see the bytecode instructions for a given program use instead the `-c` flag. With this flag, the bytecode instructions will be printed to standard output.

```
./bawk -c < [bawk filename]
```

If we compile our sample program for PNG files, we can see the bytecode generated, which is shown below.

```
$ ./bawk -c < tests/test-useful1.bawk
        0: Ldp
        1: Ldp
        2: Beo 124
        3: Lod 0
        4: Bne 10
        5: Rdb 1
        6: Lit 137
        7: Bin Sub
        8: Bne 119
        9: Bra 14
       10: Rdb 1
       11: Lit 137
       12: Bin Sub
       13: Bne 119
       14: Rdb 1
       15: Lit 80
       16: Bin Sub
       17: Bne 119
       18: Rdb 1
       19: Lit 78
       20: Bin Sub
       21: Bne 119
       22: Rdb 1
       23: Lit 71
       24: Bin Sub
       25: Bne 119
       26: Lod 0
       27: Bne 33
       28: Rdb 1
       29: Lit 13
       30: Bin Sub
       31: Bne 119
       32: Bra 37
       33: Rdb 1
       34: Lit 13
       35: Bin Sub
       36: Bne 119
       37: Lod 0
       38: Bne 44
       39: Rdb 1
       40: Lit 10
       41: Bin Sub
       42: Bne 119
       43: Bra 48
       44: Rdb 1
       45: Lit 10
       46: Bin Sub
       47: Bne 119
       48: Lod 0
       49: Bne 55
       50: Rdb 1
       51: Lit 26
       52: Bin Sub
       53: Bne 119
       54: Bra 59
       55: Rdb 1
       56: Lit 26
       57: Bin Sub
       58: Bne 119
       59: Lod 0
       60: Bne 66
       61: Rdb 1
       62: Lit 10
       63: Bin Sub
       64: Bne 119
       65: Bra 70
       66: Rdb 1
       67: Lit 10
       68: Bin Sub
       69: Bne 119
       70: Ldp
       71: Ldp
       72: Beo 116
       73: Rdb 4
       74: Str 2
       75: Rdb 1
       76: Lit 73
       77: Bin Sub
       78: Bne 111
       79: Rdb 1
       80: Lit 72
       81: Bin Sub
       82: Bne 111
       83: Rdb 1
       84: Lit 68
       85: Bin Sub
       86: Bne 111
       87: Rdb 1
       88: Lit 82
```

```
 89: Bin Sub
 90: Bne 111
 91: Ldp
 92: Ldp
 93: Beo 108
 94: Rdb 4
 95: Str 3
 96: Rdb 4
 97: Str 4
 98: Lod 3
 99: Jsr -1
100: Lod 4
101: Jsr -1
102: Bra 108
103: Skp
104: Beo 108
105: Rdb 1
106: Drp
107: Bra 92
108: Skp
109: Skp
110: Bra 116
111: Skp
112: Beo 116
113: Rdb 1
114: Drp
115: Bra 71
116: Skp
117: Skp
118: Bra 124
119: Skp
120: Beo 124
121: Rdb 1
122: Drp
123: Bra 1
124: Skp
125: Skp
126: Hlt
```

2.3 Walking through the example

The first line of the sample program in §1.2 matches the pattern `89 "PNG" 0d0a1a0a`, where the string `"PNG"` is translated into ASCII bytes `50 4e 47`. Bawk will scan through the input file until finding this pattern. On the first occurrence, the block statement following will be executed, and the nested patterns will attempt to match starting at the end of the match from the higher-level pattern. When a block exits, the pattern matching returns to executing at the position where it last began searching.

There is a special variable in bawk, `RP`, that can be used to retrieve or set the *read pointer* over the binary file. The value of `RP` represents the number of bytes from the beginning of the file. If we modify the example program to show us `RP` at various execution points it should become clear how the variable behaves.

Listing 2.1: Showing how RP works

```
print(RP);
/89 "PNG" 0d 0a 1a 0a/ {
    print(RP);
    /length:uint4 "IHDR"/ {
        print(RP);
        /width:uint4 height:uint4/ {
            print(RP);
        }
        print(RP);
    }
    print(RP);
}
print(RP);
RP + 100;
```

```
15 print(RP);
16 /ffee/ { print(RP); }
17 print(RP);
```

Running this code produces the following output on `lichtenstein.png`.

```
0
8
16
24
16
8
0
0
15705
0
```

As you can see, the `RP` pointer resets at the end of each pattern matching block, and can also be manually advanced. Pattern matching always searches forward from `RP`.

In addition to pattern matching bawk supports dynamic variable scoping, functions, conditional statements, and mixed endianness. See the language reference manual in Chapter 3 for details.

Chapter 3

Language reference manual

3.1 Lexical Conventions

A program consists of a single bawk language character file. There are six classes of tokens: identifiers, keywords, constants, string literals, operators, and separators. White space characters are ignored except as they separate tokens or when they appear in string literals.

3.1.1 Comments

Comments follow the ANSI C style, beginning with `/*` and ending with `*/`. They do not nest, and they cannot be present in quoted strings.

3.1.2 Identifiers

An identifier is a sequence of uppercase and lowercase letters A–Z, the numerals 0–9, and the underscore character. The first character of an identifier can not be a numeral.

⟨*identifier*⟩ ::= ⟨*letter*⟩
| ⟨*identifier*⟩ (⟨*letter*⟩ | ⟨*digit*⟩)

3.1.3 Keywords

The following identifiers are reserved for use as keywords, and should not be used otherwise: `if`, `else`, `return`, `def`, `int1`, `int2`, `int4`, `uint1`, `uint2`, `uint4`.

3.1.4 Constants

Bawk only supports a single kind of constant: the integer constant. The semantic meaning of integer constants depends on the context in which they occur. Except for inside of pattern expressions (see §3.5.1), the following is true. An integer constant consisting of a sequence of digits is taken to be in decimal. A sequence of hexadecimal digits prefaced by `0x` is taken to be in hexadecimal (base 16).

3.1.5 String Literals

A string is a sequence of characters enclosed in double quotes. The only valid place for strings is inside of pattern expressions (see §3.5.4).

3.2 Meaning of Identifiers

Identifiers can have several different meanings based on how they are first used. Once an identifier is associated with a particular meaning it cannot be disassociated with that meaning.

3.2.1 Function Name

Previously unused identifiers become function names once a function is declared with a given name.

3.2.2 Variable Name

Previously unused identifiers become variable names once an assignment expression (see §3.4.7) executes with the new identifier on the left hand side. Variables have scope where they are first created in this way. Scoping is dynamic.

3.2.3 Pattern Binding Variable Name

Previously unused identifiers become binded variable names once a pattern expression with a binding pattern token (see §3.5.2) matches.

3.2.4 Special Identifiers

There are certain special identifiers that behave like variables, but whose meaning affects the program execution.

- `RP` – The *file pointer* variable indicates where in the file pattern statements begin matching (see §3.3.2). The value contains the number of bytes from the beginning of the file. At the start of the bawk program this value defaults to 0.
- `LE` – Force *little-endian* interpretation of integers and strings while pattern matching in the file. A value of 0 indicates big-endian interpretation and a value of 1 indicates little-endian interpretation. The default value is 0.

3.3 Statements

The expressions described in §3.4 are a specific form of statement in the bawk language. There are three different types of statements, and a list of statements forms a bawk program. The three different types of statements are: expressions (described in §3.4), block statements, and pattern statements.

⟨*program*⟩ ::= ⟨*statement-list*⟩

⟨*statement-list*⟩ ::= ϵ
| ⟨*statement*⟩
| ⟨*statement-list*⟩ ⟨*statement*⟩

⟨*statement*⟩ ::= ⟨*expression-statement*⟩
| ⟨*block-statement*⟩
| ⟨*pattern-statement*⟩
| ⟨*function-declaration*⟩

3.3.1 Block Statements

Block statements are used to combine a sequence of statements into a single statement. The sequence of statements inside the block is executed in order when the block itself is executed. Variables first used inside the block are locally scoped, but names from outside the block are also available.

⟨*block-statement*⟩ ::= '{' ⟨*statement-list*⟩ '}'

3.3.2 Pattern Statements

Pattern statements allow a type of pattern matching to be performed on the binary file. Pattern statements consist of a pattern expression (see §3.5) and a statement. If the pattern expression matches the data file at the location of `RP` at the time of evaluation of the pattern statement, then the corresponding statement is executed. If the pattern does not match, then the statement is not executed.

⟨*pattern-statement*⟩ ::= '/' ⟨*pattern-expression*⟩ '/' ⟨*statement*⟩

3.3.3 Expression Statements

An expression statement is an expression followed by a semicolon. Expressions are described in §3.4.

⟨*expression-statement*⟩ ::= ⟨*expression*⟩ ';'

3.3.4 Function Declaration

A function declaration is a statement declaring a new function. Function declarations consist of an identifer to serve as the function name, followed by an open parenthesis, an optional comma separated list of identifiers to serve as function parameter names, a closing parenthesis, and a statement to serve as the function body.

⟨*function-declaration*⟩ ::= '`def`' ⟨*identifier*⟩ '(' ⟨*function-decl-params*⟩ ')' ⟨*statement*⟩

⟨*function-decl-params*⟩ ::= ϵ
| ⟨*identifier*⟩
| ⟨*function-decl-params*⟩ ',' ⟨*identifier*⟩

3.3.5 If statement

If the expression provided to the if statement evalutes to a non-zero value then the subsequent statement is executed. If the expression evaluates to a value of zero, then the statement is not executed.

⟨*if-statement*⟩ ::= '`if`' '(' ⟨*expression*⟩ ')' ⟨*statement*⟩

3.3.6 If/else statement

If/else statements work like if statements, with an additional second statement. If the expression evaluates to a non-zero value then the first statement is executed and the second is not. If the expression evaluates to zero then the first statement is not executed and the second is executed.

⟨*if-statement*⟩ ::= '`if`' '(' ⟨*expression*⟩ ')' ⟨*statement*⟩ '`else`' ⟨*statement*⟩

3.4 Expressions

3.4.1 Primary Expression

⟨*primary-expression*⟩ ::= ⟨*constant*⟩
| ⟨*identifier*⟩
| ⟨*string-literal*⟩
| ⟨*function-call*⟩
| '(' ⟨*expression*⟩ ')'

⟨*expression*⟩ ::= ⟨*assignment-expression*⟩
| ⟨*additive-expression*⟩

3.4.2 Function Calls

A function call is an identifier, known as the function name, followed by a pair of parenthesis containing a, possibly empty, comma-separated list of expressions. These expressions constitute arguments to the function. When functions are called a copy of each argument is made and used within the function, that is, all function calls pass arguments by value.

⟨*function-call*⟩ ::= ⟨*identifier*⟩ '(' ⟨*function-call-params*⟩ ')'

⟨*function-call-params*⟩ ::= ϵ
| ⟨*expression*⟩
| ⟨*function-call-params*⟩ ',' ⟨*expression*⟩

3.4.3 Multiplicative Operators

The multiplicative operators, * and /, are left associative. The result of * is multiplication of the two operands, and the result of / is the quotient of the two operands. The quotient with a second operator of zero yields an undefined result.

⟨*multiplicative-expression*⟩ ::= ⟨*primary-expression*⟩
| ⟨*multiplicative-expression*⟩ '*' ⟨*primary-expression*⟩
| ⟨*multiplicative-expression*⟩ '/' ⟨*primary-expression*⟩

3.4.4 Additive Operators

The two additive operators, + and -, are left associative. The expected arithmetic operation is performed on integers. The result of the + operator is the sum of the operands, and the result of the - operator is the difference of the operands.

⟨*additive-expression*⟩ ::= ⟨*multiplicative-expression*⟩
| ⟨*additive-expression*⟩ '+' ⟨*multiplicative-expression*⟩
| ⟨*additive-expression*⟩ '-' ⟨*multiplicative-expresssion*⟩

3.4.5 Relational Operators

The relational operators are all left associative. The operators all produce 0 if the specified relation is false and 1 if the specified relation is true. The operators are < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to).

⟨*relational-expression*⟩ ::= ⟨*additive-expression*⟩
| ⟨*relational-expression*⟩ '<' ⟨*additive-expression*⟩
| ⟨*relational-expression*⟩ '>' ⟨*additive-expression*⟩
| ⟨*relational-expression*⟩ '<=' ⟨*additive-expression*⟩
| ⟨*relational-expression*⟩ '>=' ⟨*additive-expression*⟩

3.4.6 Equality Operators

The == (equal to) and != (not equal to) operators compare operands for equality. Like the relational operators, the equality operators produce 0 if the relationship is false and 1 if the relationship is true. The equality operators have a lower precedence than the relational operators, and are also left associative.

⟨*equality-expression*⟩ ::= ⟨*relational-expression*⟩
| ⟨*equality-expression*⟩ '==' ⟨*relational-expression*⟩
| ⟨*equality-expression*⟩ '!=' ⟨*relational-expression*⟩

3.4.7 Assignment Expressions

All assignment expressions require a modifiable identifier on the left hand side. The only type of modifiable identifier is a variable (see §3.2.2). The assignment operator, =, is right-associative and the return value of assignment is the value being assigned.

⟨*assignment-expression*⟩ ::= ⟨*equality-expression*⟩
| ⟨*identifier*⟩ '=' ⟨*assignment-expression*⟩

3.5 Pattern Expressions

A pattern expression matches content in the binary data file. Pattern expressions are distinct in syntax from the expressions described in §3.4.

⟨*pattern-expression*⟩ ::= ϵ
| ⟨*pattern-token*⟩
| ⟨*pattern-expression*⟩ ⟨*pattern-token*⟩

⟨*pattern-token*⟩ ::= ⟨*pattern-constant*⟩
| ⟨*pattern-binding*⟩
| ⟨*pattern-variable*⟩
| ⟨*string-literal*⟩

Pattern expressions can contain constants or typed variable bindings. Each constant or binding is known as a pattern term, and pattern terms are whitespace delimited. Already bound pattern variables can also be used in pattern statements.

3.5.1 Constants

Inside pattern expressions, constants work differently than those described in §3.1.4. All constants in pattern terms are implicitly expressed in hexadecimal form, and the `0x` prefix required to express hexadecimal constants in §3.1.4 must be ommitted.

Constants are read in hexadecimal as bytes. For this reason leading zeros on a value have semantic meaning. For example `/0000abcd/` matches the four byte pattern `00 00 ab cd` in the binary file, while `/abcd/` matches the two byte pattern `ab cd`.

The byte order expected in the file is dependent on the run-time value of the `LE` special variable (see §3.2.4). If this variable is non-zero, then constants are interpreted in *little endian* order. If the variable is zero, then the interpretation is *big endian*. The default value is zero. As an example, if `LE != 0` the pattern `/aabb/` matches the file with two bytes in the order `bb aa`, and if `LE == 0` then the pattern matches bytes `aa bb` in that order.

Whitespace does make a difference in the interpretation of pattern constants. When `LE != 0`, the patterns `/aabb/` and `/aa bb/` are distinctly different. If `LE == 0`, these two patterns are the same. Whitespace added between byte boundaries also changes the semantic meaning; `/01ab/` is *not* identical to `/0 1 a b/`. If a constant has an odd number of hexadecimal digits, an implied `0` is added to the left-most side of the value. That is `/123/` is the same as `/0123/` (is the same as `/01 23/` if `LE == 0`). For this reason the example of `/0 1 a b/` is identical to `/00 01 0a 0b/` (and `/00010a0b/` if `LE == 0`).

3.5.2 Bindings

A bind pattern term consists of an identifier, a colon (`:`), and a bind type. Valid bind types are: `int1`, `int2`, `int4`, `uint2`, `uint4`. The `int1` type matches a one-byte integer, the `uint4` type matches an unsigned 4-byte integer. Identifiers have the same naming rules as §3.1.2.

⟨*pattern-binding*⟩ ::= ⟨*identifier*⟩ ‘:’ ⟨*bind-type*⟩

⟨*bind-type*⟩ ::= ‘`int1`’
| ‘`int2`’
| ‘`int4`’
| ‘`uint1`’

| ‘uint2’
| ‘uint4’

3.5.3 Pattern Variables

Previously bound pattern variables can be used in a pattern expression to match the previously bound value.

⟨*pattern-variable*⟩ ::= ⟨*identifier*⟩

3.5.4 String Literals

String literals are also supported in pattern expressions. The syntax is exactly the same as strings in §3.1.5. The string literal is converted to and interpreted as a sequence of ASCII-encoded bytes, exactly how constants are (see §3.5.1).

Chapter 4

Project plan

The plan for this project was to execute the following basic steps.

1. *Determine objectives of the language* – the language was designed to be useful for extracting data quickly from binary files. It was modelled after an awk-style syntax, but was constrained to be simple enough to implement in a single semester.

2. *Design the language syntax* – the language syntax was modelled after awk and the microc language, and the concept was tested in parallel with writing a parser for it.

   (a) *Write the scanner/parser* – This was critical to working through details of the language in parallel with designing it.

   (b) *Design the AST types* – The abstract syntax tree (AST) types were designed to be as simple as possible, so that the translation to bytecode could be done easily.

3. *Write the logic to output the AST to GraphViz* – This was done mostly to help debug the AST structure, but also to make sure the AST was easy enough to traverse before attempting the bytecode translation. Translating the AST to GraphViz code enabled the creation of a visualization that could be generated for different inputs to make sure very early on that things like operator precedence were working correctly.

4. *Design the bytecode* – The plan was to take the bytecode from microc and add some basic instructions specific to the reading of binary files, such as an instruction to read a given number of bytes, branch conditionally on reaching end-of-file, etc. The stack-style design of the microc bytecode made it easier to implement than a register-based bytecode. One change I did intend to make was switching from relative branching to absolute branching.

5. *Implement the translation logic* – The translation logic would hopefully be a straightforward step once the previous were completed. The plan was to implement translation from AST structures into basic blocks, and implement the necessary bytecode instructions in parallel, while testing along the way.

6. *Project report* – The project report would be completed for required intermediate deadlines, with the remainder written after finishing the bulk of the coding.

7. *Test* – Finally, a conclusive round of testing would be done to ensure the language has no glarring bugs. Inefficiency and minor non-fatal flaws would be tolerated.

Development tools used included git and GitHub for version control, Make for generating both the `bawk` executable as well as class documents, and a few decent text editors. GraphViz was used for visualizing the AST, and LaTeX was used to generate all the documents.

4.1 Project log

The project log, listed as git commits, is shown below. The full git repository is included in the project archive file.

Listing 4.1: Git log for project development

```
commit 9b2726855909c940c129ce02e3e81611238e5ed0
Author: Kevin Graney <nanonet@gmail.com>
Date:   Mon Aug 12 21:06:52 2013 -0400

    Report work

    Added the git log as a project log and finished up a few sections.

 Makefile                    |    6 +-
 plt_docs/gitlog.tex         | 1138 ++++++++++++++++++++++++++++++++++++++++++++++
 plt_docs/report-lessons.tex |    8 +-
 plt_docs/report-plan.tex    |    6 +-
 plt_docs/report-test.tex    |    1 -
 5 files changed, 1155 insertions(+), 4 deletions(-)

commit 057a96571e6b99e2cfbf81126564acbeb4095bd8
Author: Kevin Graney <nanonet@gmail.com>
Date:   Mon Aug 12 20:17:55 2013 -0400

    Report work

 plt_docs/report-architecture.tex |    4 +-
 plt_docs/report-introduction.tex |    1 +
 plt_docs/report-lrm.tex          |   16 +++-
 plt_docs/report-plan.tex         |    8 +-
 plt_docs/report-tutorial.tex     |  195 ++++++++++++++++++++++++++++++++++++++++
 plt_docs/report.tex              |    7 +-
 6 files changed, 224 insertions(+), 7 deletions(-)

commit 5d13044f761abc93542db2260c05d2c4087c3950
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Aug 11 17:38:11 2013 -0400

    Remove old test case

 tests/simple.bawk |    6 ------
 1 file changed, 6 deletions(-)

commit 6e9314853eef20ed8d3a856b0f786f1ba68fa1e1
Author: Kevin Graney <nanonet@gmail.com>
```

```
Date:   Sun Aug 11 17:37:50 2013 -0400

    Add test of int vs. uint types

 tests/test-pat11.bawk     |    8 ++++++++
 tests/test-pat11.bawk.out |    2 ++
 2 files changed, 10 insertions(+)

commit 5c58672c9c786b60cc336fe221d89d72318847f1
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Aug 11 17:30:25 2013 -0400

    Report work

 plt_docs/report-lrm.tex  |    5 ++---
 plt_docs/report-plan.tex |   27 ++++++++++++++++++++++++++--------
 2 files changed, 21 insertions(+), 11 deletions(-)

commit 820fc3560c2251d4421636ad13389da4666b1df3
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Aug 11 15:47:49 2013 -0400

    Report work

 plt_docs/report-architecture.tex |    5 ++++-
 plt_docs/report-lessons.tex      |    8 +++++++-
 2 files changed, 11 insertions(+), 2 deletions(-)

commit 9c4ec989cb2718b53c679b21e2521e53b137bff2
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Aug 11 15:47:38 2013 -0400

    Line length enforcement for report

 compile.ml |    3 ++-
 reader.mli |   16 +++++++++-------
 2 files changed, 11 insertions(+), 8 deletions(-)

commit 42d780a9e61394a6d229e5371b498a3c4bed9189
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Aug 11 14:56:07 2013 -0400

    Work on final report

 plt_docs/report-architecture.tex |    3 +++
 plt_docs/report-plan.tex         |   10 ++++++++++
 plt_docs/report-test.tex         |   27 +++++++++++++++++++++++++++
 plt_docs/report.tex              |   28 ++++++++++++++++++++++++++++
 4 files changed, 44 insertions(+), 24 deletions(-)

commit 03f47bf80f889638f51df5aefe62798dc5e715f1
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Aug 11 13:42:53 2013 -0400

    Work on final report

```

```
 Makefile                          |    2 +
 plt_docs/columbia_university.eps | 6907 ++++++++++++++++++++++++++++++++++++++++
 plt_docs/report-architecture.tex |   47 +
 plt_docs/report-lessons.tex      |    9 +
 plt_docs/report-lrm.tex          |   16 +-
 plt_docs/report.tex              |   34 +-
 6 files changed, 7007 insertions(+), 8 deletions(-)

commit b52b971c5ff914e23ea5b32b656fa837265ab03b
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Aug 10 14:46:23 2013 -0400

    Fail on bad use of string literals

    String literals can only be used in pattern expressions.

 compile.ml |    4 ++++
 1 file changed, 4 insertions(+)

commit 7caaed970066afd42fccf9450191928f934db0a0
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Aug 10 14:46:04 2013 -0400

    Add Beq instruction to bytecode interpreter

 bytecode.ml |    3 +++
 1 file changed, 3 insertions(+)

commit bb997604d2c1dacde12fbbd0633bb961e66e5949
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Aug 10 14:22:20 2013 -0400

    Add if/else construct to AST GraphViz output

 ast.ml |    6 ++++++
 1 file changed, 6 insertions(+)

commit 94d1b482f17b1f0bdaffcf0bc9f95b59a8ca47c6
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Aug 10 13:57:04 2013 -0400

    Add test case for LE and a string constant

 tests/test-pat10.bawk     |   13 +++++++++++++
 tests/test-pat10.bawk.out |    4 ++++
 2 files changed, 17 insertions(+)

commit 09902de304da484f90227cbc8c4740f6807f7174
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Aug 10 13:11:19 2013 -0400

    Add test case for LE

 tests/test-pat9.bawk     |   36 ++++++++++++++++++++++++++++++++++++
 tests/test-pat9.bawk.out |   10 ++++++++++
 2 files changed, 46 insertions(+)
```

```
commit eaa5733f9963152162744796 68d5267b01e38eed
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Aug 10 13:02:29 2013 -0400

    Add support for little endian constants

    Pattern constants (of hexadecimal integer form) are now supported
    when LE!=0 to be interpreted as a little endian value.

 compile.ml |   13 ++++++++++++-
 1 file changed, 12 insertions(+), 1 deletion(-)

commit 1f92e256bc1632bc000b2a509f8bab3169808a90
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Aug 10 13:01:38 2013 -0400

    Add support for previously bound pattern variables

    Previously bound pattern variables can be used in future pattern
    matches.

 compile.ml               |   18 ++++++++++++++++++
 parser.mly               |    8 +++++++-
 tests/test-pat8.bawk     |   10 ++++++++++
 tests/test-pat8.bawk.out |    4 ++++
 4 files changed, 39 insertions(+), 1 deletion(-)

commit b03fc3b1ad8c2603175085bfbd8f14b7ec42b6d3
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Aug 10 12:12:40 2013 -0400

    Test writing of RP

 tests/test-pat7.bawk     |   16 ++++++++++++++++
 tests/test-pat7.bawk.out |    5 +++++
 2 files changed, 21 insertions(+)

commit 8e005718abc113fdd2105bdd9ffbc70016482ba5
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Aug 10 10:55:02 2013 -0400

    Add LE as a default variable

 compile.ml |   12 ++++++++----
 1 file changed, 8 insertions(+), 4 deletions(-)

commit d2c07179ed76266c6aea84b8ed313ed31d9cddd4
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Aug 10 10:43:14 2013 -0400

    Support assignment to RP

 bytecode.ml |    3 +++
 1 file changed, 3 insertions(+)
```

```
commit dd0b8d75c79828e90bb557b78738ae31e4b3f487
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Aug 10 10:38:10 2013 -0400

    Add bind type names to reserved words

 plt_docs/report-lrm.tex |    4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)

commit b0f748857c35b232b3c0f7e6f3f1cd0021ba1c45
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Aug 10 10:37:01 2013 -0400

    Change dynamic scoping of assignment

    Changed the scoping of the assignment operator to be dynamic
    and match the scoping of how variables are read.

 compile.ml                 |    8 +++++---
 tests/test-scope1.bawk.out |    2 +-
 tests/test-scope2.bawk     |   16 ++++++++++++++++
 tests/test-scope2.bawk.out |    5 +++++
 4 files changed, 27 insertions(+), 4 deletions(-)

commit 22c9279c64a1d417a2fcee8b6fc8086b65de1e49
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Aug 10 10:36:07 2013 -0400

    Add test case of if/else constructs

 tests/test-cond1.bawk     |   15 +++++++++++++++
 tests/test-cond1.bawk.out |    2 ++
 2 files changed, 17 insertions(+)

commit 49af898a754e376a4856817d9420b1e290345e1f
Author: Kevin Graney <nanonet@gmail.com>
Date:   Thu Aug 8 19:51:14 2013 -0400

    Change sample program to use uint types

 tests/test-useful1.bawk |    8 +++-----
 1 file changed, 3 insertions(+), 5 deletions(-)

commit e6bd2027d36914828584a5d4cca14479d2d1397c
Author: Kevin Graney <nanonet@gmail.com>
Date:   Thu Aug 8 19:47:44 2013 -0400

    Create final report skeleton

    Created a skeleton of the final report based on the syllabus.

 plt_docs/report-introduction.tex |    8 ++
 plt_docs/report-lrm.tex          |  220 ++++++++++++++++++++++++++++++++++++++++
 plt_docs/report.tex              |   52 +++++++++
 3 files changed, 280 insertions(+)
```

```
commit 87e3d252b4ed368753869e6c7ea161b4728ef23d
Author: Kevin Graney <nanonet@gmail.com>
Date:   Thu Aug 8 19:42:39 2013 -0400

    Add if/else statement support

    This will be required for supporting runtime resolution of RP
    easily, so it seems trivial to just implement the if/else construct
    in the language too.

 ast_types.mli |    1 +
 compile.ml    |   18 ++++++++++++------
 parser.mly    |    8 +++++++-
 scanner.mll   |    3 +++
 4 files changed, 23 insertions(+), 7 deletions(-)

commit 3cf57fd37d723fb04bdd7235ee61fa58989cdd3a
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Aug 4 15:57:47 2013 -0400

    Remove unused variable

 scanner.mll |    2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 4eea7d725f07304ab6c4c5f231632b55769f0b72
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Aug 4 15:50:55 2013 -0400

    Add test case for bind types

 tests/test-pat6.bawk     |   23 +++++++++++++++++++++++
 tests/test-pat6.bawk.out |   12 ++++++++++++
 2 files changed, 35 insertions(+)

commit 905b922d5078080bd0575690d148f68b501cb70f
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Aug 4 15:49:14 2013 -0400

    Add uint1 type

 ast.ml        |    3 +++
 ast_types.mli |    1 +
 scanner.mll   |    1 +
 3 files changed, 5 insertions(+)

commit 24dd8c5b1b31fe60fb58003cb96b62ce33ecdc87
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Aug 4 15:27:56 2013 -0400

    Add support for differentiating int/uint types

    Added compiler support for treating integer types as signed.  This
    assumes that the interpretation stored in the file is twos
    complement.
```

```
 compile.ml |    7 +++++--
 1 file changed, 5 insertions(+), 2 deletions(-)

commit 7a5b4836d5aea31cfef8db3d9a522400fc8bd10c
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Aug 4 15:27:04 2013 -0400

    Add instruction to convert to twos complement

    Added a bytecode instruction to reinterpret the top of the stack
    as a twos complement value with the given bit length.

 bytecode.ml        |   10 +++++++++-
 bytecode_types.mli |    1 +
 utile.ml           |    6 ++++++
 utile.mli          |    4 ++++
 4 files changed, 20 insertions(+), 1 deletion(-)

commit 749e0bf50ad1a577adea51496b6466262f8a0274
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Aug 4 15:25:37 2013 -0400

    Add function to map binding types to their sizes

 ast.ml  |    7 +++++++
 ast.mli |    4 ++++
 2 files changed, 11 insertions(+)

commit 22658bec76ebd2d2cf0611d790582fcbf91d49ef
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Aug 4 13:32:05 2013 -0400

    Add pattern strings to AST GraphViz output

 ast.ml |    7 +++++--
 1 file changed, 5 insertions(+), 2 deletions(-)

commit ec915c6e62c8070c8b74f7f29f05ddb96af89d5d
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Aug 4 13:23:56 2013 -0400

    Add LaTeX file for final report

 .gitignore          |    1 +
 Makefile            |    2 +-
 plt_docs/report.tex |  114 ++++++++++++++++++++++++++++++++++++++++++++++++++++
 3 files changed, 116 insertions(+), 1 deletion(-)

commit 37be880f30d6d0e0839f76b2e7d995e310a81821
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Aug 4 12:28:53 2013 -0400

    Add test case for variable scoping

    Show that variable scoping works inside of function calls

```

```
 tests/test-func3.bawk     |   20 ++++++++++++++++++++
 tests/test-func3.bawk.out |    7 +++++++
 2 files changed, 27 insertions(+)

commit bad13c86818b3be0eb077773a5326e0c67da40e0
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Aug 3 17:29:40 2013 -0400

    Add support for function calls with arguments

    Added support for function call arguments.  Values are placed on
    the stack at an offset from the frame pointer.  The symbol table
    contains values <-100 to represent these values differently from
    global variables.

 bytecode.ml        |   31 +++++++++++++++++++++++++++-------
 bytecode_types.mli |    8 ++++----
 compile.ml         |   19 +++++++++++++++---
 3 files changed, 44 insertions(+), 14 deletions(-)

commit 96ec3a1b3a402498188f1ff289c6aee524d058ce
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Aug 3 17:28:20 2013 -0400

    Add support for starting enumeration at a value

    Added an optional argument to start the enumeration at a specific
    value.  The default remains 0.

 utile.ml  |    4 ++--
 utile.mli |    3 ++-
 2 files changed, 4 insertions(+), 3 deletions(-)

commit 5f3fa692057c26791269550c5f8f11da956dcb20
Author: Kevin Graney <nanonet@gmail.com>
Date:   Fri Aug 2 20:55:34 2013 -0400

    Make pattern matching work across entire file

    Pattern matching now searches forward over the the entire file
    instead of expecting a match at the current location of RP.  This
    is implemented in an incredibly inefficient way, but this should
    be good enough for class.

 bytecode.ml          |    6 ++++--
 compile.ml           |   30 +++++++++++++++++++++++-------
 tests/test-pat3.bawk |    2 +-
 3 files changed, 28 insertions(+), 10 deletions(-)

commit 76e451ee8c908db8d8f1755cc239a4ac47230052
Author: Kevin Graney <nanonet@gmail.com>
Date:   Fri Aug 2 20:53:27 2013 -0400

    Add support for reading the RP variable

    The variable is added as a special case in the bytecode interpreter.
```

```
    Writing to this variable is not yet supported.

 bytecode.ml |    3 +++
 compile.ml  |    2 +-
 2 files changed, 4 insertions(+), 1 deletion(-)

commit b6edc2975e4ee271ba7d8e0622f830c61ffefd59
Author: Kevin Graney <nanonet@gmail.com>
Date:   Fri Aug 2 20:51:32 2013 -0400

    Added the Beo and Drp instructions to bytecode

    Beo branches if the next byte reaches EOF and Drp drops a value
    from the stack.

 bytecode.ml        |    7 +++++++
 bytecode_types.mli |    1 +
 compile.ml         |    1 +
 reader.ml          |   17 +++++++----------
 reader.mli         |    4 +++-
 5 files changed, 19 insertions(+), 11 deletions(-)

commit e794ccf368d43475081411e6b7b063aaa6274f1a
Author: Kevin Graney <nanonet@gmail.com>
Date:   Fri Aug 2 19:41:42 2013 -0400

    Add test for scoping of variables

 tests/test-scope1.bawk     |   15 +++++++++++++++
 tests/test-scope1.bawk.out |    5 +++++
 2 files changed, 20 insertions(+)

commit f5652c7d9dff21400ccb243a805d0148308afb87
Author: Kevin Graney <nanonet@gmail.com>
Date:   Fri Aug 2 19:37:59 2013 -0400

    Merge symbol map into a single StringMap

    Removed the function_map and variable_map objects and created a
    single symbol_map StringMap to hold all the symbols.  There still
    exists a bindings map to contain the size of the binding variables.

 bawk.ml     |    4 ++--
 compile.ml  |   60 ++++++++++++++++++++++++++++--------------------------------
 compile.mli |   13 ++++++-------
 3 files changed, 36 insertions(+), 41 deletions(-)

commit d238b3e53cb35d6285251e98d202ff6fafa6450b
Author: Kevin Graney <nanonet@gmail.com>
Date:   Fri Aug 2 18:34:12 2013 -0400

    Add parent pointers to env

    Added parent pointers to the environment records to support linked
    lists.  Create a simple resolution function for function calls so
    things like print() (a built-in function) work from inside functions.
```

```
 compile.ml  |   22 ++++++++++++++++++++--
 compile.mli |    1 +
 2 files changed, 21 insertions(+), 2 deletions(-)

commit 2fa4bd4f81bbe1e9f4980029be7f7d03bd3a5ae0
Author: Kevin Graney <nanonet@gmail.com>
Date:   Fri Aug 2 18:57:03 2013 -0400

    Add Beq to label resolution

 compile.ml |    1 +
 1 file changed, 1 insertion(+)

commit dd63a8c9d9a101245eadef753d7812bf3fcf67c7
Author: Kevin Graney <nanonet@gmail.com>
Date:   Fri Aug 2 13:48:32 2013 -0400

    Added support for strings in pattern expressions

    Added support for string literals inside of pattern expressions.
    This allows the first useful test case to be added for finding
    the width and height of a PNG file.

 ast_types.mli               |    1 +
 compile.ml                  |    6 +++++-
 parser.mly                  |    1 +
 tests/test-useful1.bawk     |   12 ++++++++++++
 tests/test-useful1.bawk.out |    2 ++
 utile.ml                    |    3 +++
 utile.mli                   |    4 ++++
 7 files changed, 28 insertions(+), 1 deletion(-)

commit fdad37b643f96444bb64620717901e649dd514d1
Author: Kevin Graney <nanonet@gmail.com>
Date:   Fri Aug 2 13:30:19 2013 -0400

    Added explode and implode functions to Utile

    These are functions defined in the OCaml docs as inefficient ways
    to convert a string to a char list or a char list to a string
    respectively.

 utile.ml  |   13 ++++++++++++
 utile.mli |    6 ++++++
 2 files changed, 19 insertions(+)

commit 34e7d8295fcaf5f18103305609d5f583f95d3b38
Author: Kevin Graney <nanonet@gmail.com>
Date:   Fri Aug 2 13:29:14 2013 -0400

    Add boolean operators to bytecode interpreter.

    These may not be of limited use since we don't have conditionals,
    but they should work.
```

```
 bytecode.ml |    7 +++++++
 1 file changed, 7 insertions(+)

commit 4d96f677c8b0ce4c67b202f3b0fe886d0442eedf
Author: Kevin Graney <nanonet@gmail.com>
Date:   Fri Aug 2 13:00:15 2013 -0400

    Remove error-prone branch

    No need to have Bne instruction for binding pattern match, since
    this will cause it to reject cases where the variable binds
    to a value of zero.

 compile.ml |    1 -
 1 file changed, 1 deletion(-)

commit 79603df70433b8ed4b836236cc53f8d3b49c3531
Author: Kevin Graney <nanonet@gmail.com>
Date:   Fri Aug 2 13:00:03 2013 -0400

    Add test case for int4 bind type

 tests/test-pat5.bawk     |    5 +++++
 tests/test-pat5.bawk.out |    1 +
 2 files changed, 6 insertions(+)

commit 85c646aaf977d13568ecb04c4f741ef603f43937
Author: Kevin Graney <nanonet@gmail.com>
Date:   Fri Aug 2 12:13:38 2013 -0400

    Add test case for int1 pattern binding

 tests/test-pat4.bawk     |   12 ++++++++++
 tests/test-pat4.bawk.out |    3 +++
 2 files changed, 15 insertions(+)

commit 12cf98e1a573a16d463f25e2669b0f4642a7984a
Author: Kevin Graney <nanonet@gmail.com>
Date:   Wed Jul 31 20:25:40 2013 -0400

    Initial cut at pattern variables

    Added pattern variables.  The implementation is limited.  Only int1
    type works at the moment, the namespace for variables is shared
    with the globals, and the storage of the data is still messy.

 bytecode.ml |    3 +++
 compile.ml  |   31 +++++++++++++++++++++++++++---
 compile.mli |    2 +-
 3 files changed, 32 insertions(+), 4 deletions(-)

commit c1efac5a1334654490cfbe2472498d633c8d31d8
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Jul 28 22:13:01 2013 -0400

    Add simple global variables
```

```
601
602     Added simple global variables.  These are implemented as an array
603     in the bytecode interpreter.  The compiler assigns each name a
604     unique entry in the array.  Currently all variables are considered
605     global.  Variables are not being set at an offset of FP on the stack
606     because they are dynamically created and not required to be declared.
607
608  bytecode.ml              |    9 +++++++++
609  compile.ml               |   33 +++++++++++++++++++++++++++++++++
610  compile.mli              |    7 +++++++
611  tests/test-var1.bawk     |    5 +++++
612  tests/test-var1.bawk.out |    2 ++
613  5 files changed, 56 insertions(+)
614
615 commit 2ee86a40da8b5aed8e0e7844c1b4ccb7a3ef9d39
616 Author: Kevin Graney <nanonet@gmail.com>
617 Date:   Sat Jul 27 21:02:36 2013 -0400
618
619     Add basic function call test cases
620
621     Added test cases for function calls with no arguments.
622
623  tests/test-func1.bawk     |   10 ++++++++++
624  tests/test-func1.bawk.out |    5 +++++
625  tests/test-func2.bawk     |    7 +++++++
626  tests/test-func2.bawk.out |    2 ++
627  4 files changed, 24 insertions(+)
628
629 commit d79330e9a40633c38a61a7760254a9a3756689f0
630 Author: Kevin Graney <nanonet@gmail.com>
631 Date:   Sat Jul 27 20:51:53 2013 -0400
632
633     Support function calls by name
634
635     Added support for function calls by name.  Still need to decide
636     how exactly the 'env' record is going to work.
637
638  bytecode.ml        |    6 ++++++
639  bytecode_types.mli |    2 +-
640  compile.ml         |   15 +++++++++++----
641  3 files changed, 18 insertions(+), 5 deletions(-)
642
643 commit bd94bb11eb0546ff2633e152df2e61759144d8ab
644 Author: Kevin Graney <nanonet@gmail.com>
645 Date:   Sat Jul 27 19:12:03 2013 -0400
646
647     Implement compilation of function declarations
648
649     Each function declaration is preceded by a Bra call to skip over
650     it in case the PC approaches it from something prior falling
651     through it (e.g. at the start of the program).  This is a quicker
652     hack than actually reloacting all of them after the Hlt instruction
653     or something.
654
655  bytecode.ml        |    2 ++
656  bytecode_types.mli |    9 +++++----
```

```
 compile.ml         |    8 +++++++-
 3 files changed, 14 insertions(+), 5 deletions(-)

commit 32a42f8b1cf9abc0f90ab10ea98aaee9e83aa4f5
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jul 27 17:35:04 2013 -0400

    Don't compile code when printing out the AST

    The AST is useful for debugging, often of code that doesn't compile.
    Therefore, it is silly to compile the code (and abort) before
    printing the AST.

 bawk.ml |    7 ++++---
 1 file changed, 4 insertions(+), 3 deletions(-)

commit 2b25b94e38e0de1f6dbe991c31460cffe3f04186
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jul 27 17:33:46 2013 -0400

    Adding function declarations to parser

    To declare functions the syntax is changed so a "def" keyword must
    precede the function identifier.  This eliminates reduce/reduce
    conflicts between expressions and the function declarations.

 ast.ml        |   19 ++++++++++++++++++-
 ast_types.mli |    8 +++++++-
 parser.mly    |   16 ++++++++++++++++-
 scanner.mll   |    1 +
 4 files changed, 41 insertions(+), 3 deletions(-)

commit 821a574a5238d228136cb885a78e5bae6606f010
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jul 27 16:06:49 2013 -0400

    Fix bug in pattern match compilation

    The file wasn't seeking back to the pre-pattern match location when
    the pattern did not match.

 compile.ml |    3 +--
 1 file changed, 1 insertion(+), 2 deletions(-)

commit a36cc2c7e1545108511263a18654f9dfcce6643a
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jul 27 16:06:12 2013 -0400

    Add test cases for pattern matching

 run_tests.sh             |    2 +-
 tests/lichtenstein.png   |  Bin 0 -> 365198 bytes
 tests/test-pat1.bawk     |    9 +++++++++
 tests/test-pat1.bawk.out |    2 ++
 tests/test-pat2.bawk     |    9 +++++++++
 tests/test-pat2.bawk.out |    3 +++
```

```
 tests/test-pat3.bawk     |   13 ++++++++++++
 tests/test-pat3.bawk.out |    3 +++
 8 files changed, 40 insertions(+), 1 deletion(-)

commit 8314d555d1c85be1bc9113a7d9fc6bd53ec5357a
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jul 27 16:04:21 2013 -0400

    Fix interpreter bug in Bne

    Top of stack (compare-to value) was not being popped.

 bytecode.ml |    2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit abfecc6c182861783762343b579e344382912429
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jul 27 15:50:13 2013 -0400

    Add instructions to store file position on stack

    Added instructions to store the file position on the stack and
    read a file position to seek to from the stack.  This enables the
    bytecode interpreter to return to the position in the file at
    the conclusion of a pattern statement.

 bytecode.ml        |   12 ++++++++++-
 bytecode_types.mli |    2 ++
 compile.ml         |    2 ++
 reader.ml          |    6 ++++++
 reader.mli         |    6 ++++++
 5 files changed, 27 insertions(+), 1 deletion(-)

commit 3753438ca40393e69f64184a9f1795224c8d2d18
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jul 27 15:49:06 2013 -0400

    Fix error in built-in print function

    The built-in print function didn't pop it's argument from the stack.

 bytecode.ml |    2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 0e87833401824d6d87596a7f107e53d9a837a785
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jul 27 15:28:56 2013 -0400

    Add branch rewriting on second pass

 bytecode.ml  |    9 +++++++++
 bytecode.mli |    7 +++++++
 compile.ml   |   27 +++++++++++++++++++++++++++-
 3 files changed, 42 insertions(+), 1 deletion(-)

commit 2bf1ec02de8c3fe35d3b154b68d56eb7b127ae45
```

```
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jul 27 15:19:07 2013 -0400

    Add Reader module and support for branching

    Added the Reader module and support for resolving branch addresses
    correctly using the 'Label' pseudo instruction and two passes.

 Makefile    |    1 +
 bytecode.ml |    4 ++++
 reader.ml   |   50 ++++++++++++++++++++++++++++++++++++++++++++++++++
 reader.mli  |   26 ++++++++++++++++++++++++++
 4 files changed, 81 insertions(+)

commit df48722be79dd90290f9ff2bf6e394c62dac47cc
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jul 27 15:16:20 2013 -0400

    Don't compile inside Bytecode module

    This removes the dependency of the Bytecode module on the
    Compile module.  OCaml appears somewhat picky about enforcing
    non-cyclic dependencies.

 bawk.ml      |    5 +++--
 bytecode.ml  |    8 +++++---
 bytecode.mli |    4 ++--
 3 files changed, 8 insertions(+), 9 deletions(-)

commit 6092ae879e5ed00e52f74100f2435643a21f66ae
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jul 27 14:56:06 2013 -0400

    Use ocamldep to find linking order

    Use 'ocamldep -sort' to determine the order the .cmo files get
    linked.

 Makefile |    2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit ad4836c8a8e890289a5d77a2998242ff560f05e0
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jul 27 14:45:42 2013 -0400

    Allow enumerate to do some primitive introspection

    Changed the step function for enumerate to allow it to behave
    differently based on the value of the head of the content list.
    This is useful for enumerating bytecode with pseudo instructions,
    where we want to no give the pseudo instructions their own address.

 compile.ml |    2 +-
 utile.ml   |    4 ++--
 utile.mli  |    2 +-
 3 files changed, 4 insertions(+), 4 deletions(-)
```

```
commit 3e153a4adf5f5be93ffa7a127487573a38b51a3b
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jul 27 14:19:55 2013 -0400

    Add bytecode support for branches

    Added the Label pseudo-instruction to the bytecode to allow the
    branching addresses to be resolved.  Basically labels are entered
    by the one stage of the compilation and then resolved to actual
    addresses by a later stage.

 bytecode.ml        |    5 +++++
 bytecode_types.mli |    4 +++-
 compile.ml         |   15 ++++++++++----
 3 files changed, 19 insertions(+), 5 deletions(-)

commit 41b603c93a75d06b673989533ed4541735910d63
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jul 27 13:45:43 2013 -0400

    Support passing a binary filename in argv

    Added support for reading argv[2] as the filename of a binary file
    to process.  Argument parsing is still very primitive, but this
    is enough to build something that works.

 bawk.ml      |   10 +++++++++-
 bytecode.ml  |    6 ++++--
 bytecode.mli |    4 ++--
 3 files changed, 14 insertions(+), 6 deletions(-)

commit 747889f4963cf7c90a2d1f6219c5b8ccf59b7663
Author: Kevin Graney <nanonet@gmail.com>
Date:   Thu Jul 25 22:00:04 2013 -0400

    Begin implementing pattern matching compilation

    A start on the implementation of compiling pattern expressions into
    bytecode.

 compile.ml |   19 +++++++++++++++++++-
 1 file changed, 18 insertions(+), 1 deletion(-)

commit a74c6bd36d904a37206f7e2b5dc6bfff9c74ca91
Author: Kevin Graney <nanonet@gmail.com>
Date:   Thu Jul 25 21:59:37 2013 -0400

    Add more instructions to string_of_instruction

 bytecode.ml |    4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)

commit 368e05b6506f7ea33dfc16c5fb91694195c0ca92
Author: Kevin Graney <nanonet@gmail.com>
Date:   Thu Jul 25 21:56:25 2013 -0400
```

```
881
882     Add size_of_bind_type function
883
884     Added a function to return the size in bytes of a given bind_type.
885
886  ast.ml  |    7 +++++++
887  ast.mli |    4 ++++
888  2 files changed, 11 insertions(+)
889
890 commit 3057ec54d1eedc46d4fcf6a549c92c05fb0350a0
891 Author: Kevin Graney <nanonet@gmail.com>
892 Date:   Thu Jul 25 21:33:15 2013 -0400
893
894     Reverse parsing order of pattern tokens
895
896     Pattern tokens were stored in the AST in the reverse of the desired
897     order.  This switches the order.
898
899  parser.mly |    5 ++++-
900  1 file changed, 4 insertions(+), 1 deletion(-)
901
902 commit bfc73199e14bf37ca708d6c2e47258bab1eaf02d
903 Author: Kevin Graney <nanonet@gmail.com>
904 Date:   Thu Jul 25 21:29:51 2013 -0400
905
906     Add PC values to bytecode output
907
908     Added value of program counter address to instructions when printing
909     the bytecode output.
910
911  bytecode.ml |    4 +++-
912  1 file changed, 3 insertions(+), 1 deletion(-)
913
914 commit ede71a47f4d8fa8e1f2b40893044358bd9046007
915 Author: Kevin Graney <nanonet@gmail.com>
916 Date:   Thu Jul 25 21:18:37 2013 -0400
917
918     Add parsing of hex constants in pattern expression
919
920     Added parsing of hexadecimal constants in the pattern expression,
921     as defined in the LRM.  Big/little endian conversion should
922     probably be handled by the bytecode at runtime.
923
924  parser_help.ml |   18 ++++++++++++++++-
925  1 file changed, 17 insertions(+), 1 deletion(-)
926
927 commit f8dbc43c280dc1264938e567664ee38994e95368
928 Author: Kevin Graney <nanonet@gmail.com>
929 Date:   Wed Jul 24 20:03:51 2013 -0400
930
931     Add regression tests
932
933     Started the regression test suite with a very simple test case.
934
935  run_tests.sh              |  123 ++++++++++++++++++++++++++++++++++++++++++++++++
936  tests/test-arith.bawk     |    1 +
```

```
 tests/test-arith.bawk.out |    1 +
 3 files changed, 125 insertions(+)

commit 35794eb2cc8055dafff965986bc070b5868e1142
Author: Kevin Graney <nanonet@gmail.com>
Date:   Wed Jul 24 19:40:07 2013 -0400

    Halt at conclusion of program

    Added the Hlt instruction to the end of compiled programs, and
    implemented halt functionality in the interpreter.  Previously
    we just kept reading through to the next instruction in memory.

 bytecode.ml |    2 ++
 compile.ml  |    2 +-
 2 files changed, 3 insertions(+), 1 deletion(-)

commit 1b3bb14eb3a13beede68a3c66d9e389f6ca2d63e
Author: Kevin Graney <nanonet@gmail.com>
Date:   Tue Jul 23 20:30:03 2013 -0400

    Add basic arithmetic operators to interpreter

    Added basic arithmetic instructions to bytecode interpreter.  It's
    still missing a lot of things, but this is a start.

 bytecode.ml  |   23 ++++++++++++++++++++++-
 bytecode.mli |    6 +++++-
 2 files changed, 27 insertions(+), 2 deletions(-)

commit f9aaa15acb4fee676f7ba5bc4468168991d6d5bc
Author: Kevin Graney <nanonet@gmail.com>
Date:   Tue Jul 23 19:31:00 2013 -0400

    Add stub for bytecode execution flag

    Added the -e flag to compile a program into bytecode and then
    execute that bytecode.

 bawk.ml      |    7 ++++---
 bytecode.ml  |    4 ++++
 bytecode.mli |    3 +++
 3 files changed, 11 insertions(+), 3 deletions(-)

commit 878dd0aa65efeea16f1adece43e9b8b1683b1c80
Author: Kevin Graney <nanonet@gmail.com>
Date:   Mon Jul 22 23:03:40 2013 -0400

    Fix bug where built-in function addresses increase

    Fixed a bug where built-in function addresses increased as the
    name list is read instead of decreased.  (Negative addresses are
    used to identify built-in functions.)

 compile.ml |    3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

```
commit 51109abddc9345b4147b6771c0f2c3c0eacff4b0
Author: Kevin Graney <nanonet@gmail.com>
Date:   Mon Jul 22 23:02:41 2013 -0400

    Add step function to enumerate

    Added an optional argument to enumerate to allow a step function
    to be specified.

 utile.ml  |    4 ++--
 utile.mli |    2 +-
 2 files changed, 3 insertions(+), 3 deletions(-)

commit c5f9a558bbec35da21b91546ebc287afc0b4b510
Author: Kevin Graney <nanonet@gmail.com>
Date:   Mon Jul 22 22:53:57 2013 -0400

    Populate starting environment with built-ins

    Populated the default environment with built-in functions.  They
    are read from an array, enumerated.

 bawk.ml     |   13 +++++++++++
 compile.ml  |   14 +++++++++-----
 compile.mli |   38 ++++++++++++++++++++++++++++++++++++++
 3 files changed, 60 insertions(+), 5 deletions(-)

commit b970704fb337247656f959c79fc90415f68566f5
Author: Kevin Graney <nanonet@gmail.com>
Date:   Mon Jul 22 22:53:13 2013 -0400

    Add enumerate function

    Added enumerate function to Utile library.

 utile.ml  |    5 +++++
 utile.mli |    6 +++++-
 2 files changed, 10 insertions(+), 1 deletion(-)

commit 1bbdd68335ca43bafa27186611dc21a3edca399f
Author: Kevin Graney <nanonet@gmail.com>
Date:   Mon Jul 22 21:42:09 2013 -0400

    Add function call stubs

    Added preliminary logic for generating bytecode for function
    calls.  Still stub code.  Also added basic environment to support
    things like function names (and eventually scoping).

 bytecode.ml |    3 ++-
 compile.ml  |   35 +++++++++++++++++++++++++++++------
 compile.mli |    4 ++--
 3 files changed, 33 insertions(+), 9 deletions(-)

commit 19dc1c220040c645ab88bcbc946062360e2a53e0
```

```
Author: Kevin Graney <nanonet@gmail.com>
Date:   Fri Jul 19 09:50:16 2013 -0400

    Ignore autogenerated makefiles

 .gitignore |    1 +
 1 file changed, 1 insertion(+)

commit 5fed5f231c68a46dc7208fbb4ae8e267d0ebf436
Author: Kevin Graney <nanonet@gmail.com>
Date:   Fri Jul 19 09:49:50 2013 -0400

    Add more stub code to the translator

 bytecode.ml        |    1 +
 bytecode_types.mli |    3 ++-
 compile.ml         |    6 +++++-
 3 files changed, 8 insertions(+), 2 deletions(-)

commit 74d7f39a9d0a6f80c5e207947fc9f1e009a9a440
Author: Kevin Graney <nanonet@gmail.com>
Date:   Wed Jul 17 22:17:26 2013 -0400

    Remove stub value and replace with actual.

 compile.ml |    2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 8468b5b5886bdf32c26d618185ca71ab7de4bfbf
Author: Kevin Graney <nanonet@gmail.com>
Date:   Wed Jul 17 22:14:36 2013 -0400

    Add bytecode creation support for binary operators.

    Added bytecode generation for binary operators.

 bytecode.ml       |    1 +
 compile.ml        |    4 +++-
 tests/simple.bawk |    2 +-
 3 files changed, 5 insertions(+), 2 deletions(-)

commit f9939d8c691397880f4a5e47ede0eed287601714
Author: Kevin Graney <nanonet@gmail.com>
Date:   Wed Jul 17 22:12:20 2013 -0400

    Change string_of_operator to produce bytecode

    Changed string_of_operator from producing human readable strings
    to bytecode operator strings.  This benefits the bytecode generation
    and makes the AST graph more concise.

 ast.ml  |   20 ++++++++++----------
 ast.mli |    4 ++++
 2 files changed, 14 insertions(+), 10 deletions(-)

commit d8f9b1658ca62d6920c2a69e9d7535f420b1c62f
```

```
Author: Kevin Graney <nanonet@gmail.com>
Date:   Wed Jul 17 21:52:47 2013 -0400

    Add primitive bytecode creation

    Added Bytecode_types module to mimic pattern used for Ast_types.
    Also got simple translation working, so the -c flag now produces
    the bytecode output.  Currently only works for primitive programs.

 Makefile           |    2 +-
 bytecode.ml        |   24 ++++++------------------
 bytecode.mli       |   10 +++++++++-
 bytecode_types.mli |   16 ++++++++++++++++
 compile.ml         |   18 ++++++++++++------
 compile.mli        |    5 +++++
 tests/simple.bawk  |    6 ++++++
 7 files changed, 55 insertions(+), 26 deletions(-)

commit 3277f84748d4b697c71bc01b8fee79b6a9a91200
Author: Kevin Graney <nanonet@gmail.com>
Date:   Wed Jul 17 21:50:48 2013 -0400

    Produce ocamldoc by default

    Changed Makefile to produce design_docs target by default.  (This
    is the ocamldoc output.)

 Makefile |    2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit c269dc76c098f6f880e395abf334a6262a7cae20
Author: Kevin Graney <nanonet@gmail.com>
Date:   Wed Jul 17 19:34:53 2013 -0400

    Automatic dependency generation

    Makefile now generates dependencies from ocamldep automatically.  It's
    a little ugly in that it makes the dependencies regardless of what the
    target is (including clean) but this is still better than manually
    updating the Makefile all the time.

 Makefile |   47 +++++++++++++++++++----------------------------------
 1 file changed, 19 insertions(+), 28 deletions(-)

commit 21e268ef44da666feaeabcd77926a3884f25f87f
Author: Kevin Graney <nanonet@gmail.com>
Date:   Wed Jul 17 19:15:14 2013 -0400

    Add bytecode and compile modules

    Add primitive bytecode and compile modules.

 Makefile     |   11 ++++++++---
 bawk.ml      |    2 +-
 bytecode.ml  |   21 +++++++++++++++++++++
 bytecode.mli |    2 ++
```

```
 compile.ml   |   10 ++++++++++
 5 files changed, 42 insertions(+), 4 deletions(-)

commit 75444530c89124b5ba77072bacbfd4d0a4be50db
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Jul 14 14:38:29 2013 -0400

    Add support for command line arguments

    Added primitive support for command line arguments to distinguish
    between actions such as compile, output the AST, output the Bytecode,
    etc.

 bawk.ml |   35 +++++++++++++++++++++++------------
 1 file changed, 23 insertions(+), 12 deletions(-)

commit 7c73eafb8a0d38620081c90e417c3110561c7302
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Jul 14 14:01:17 2013 -0400

    Expand definition of bind patterns

    Work on LRM to expand the definition of bind patterns.

 plt_docs/lrm.tex |   20 ++++++++++++++++++--
 1 file changed, 18 insertions(+), 2 deletions(-)

commit 244a3faaee0a96b850562e183fc94103800e6598
Author: Kevin Graney <nanonet@gmail.com>
Date:   Thu Jun 27 14:39:18 2013 -0400

    LRM work

 plt_docs/lrm.tex |  149 ++++++++++++++++++++++++++++++++++++++----------------
 1 file changed, 108 insertions(+), 41 deletions(-)

commit c52dd031082c4562d9030a5784bc19ec1994fa94
Author: Kevin Graney <nanonet@gmail.com>
Date:   Thu Jun 27 10:43:05 2013 -0400

    Add assignment operator

    Added right-associative assignment operator.

 ast.ml               |    4 ++++
 ast_types.mli        |    1 +
 parser.mly           |    2 ++
 tests/prototype.bawk |    6 ++++++
 4 files changed, 13 insertions(+)

commit 0dc571931abd863381854157cd1a7c67d17efa98
Author: Kevin Graney <nanonet@gmail.com>
Date:   Thu Jun 27 10:08:00 2013 -0400

    Add preliminary pattern expr constant parsing

```

```
    Added preliminary code to differentiate between integer constants
    in pattern expressions and those in plain expressions so they can
    be treated differently.  Pattern expression constants will always
    be interpreted in hexadecimal as a series of bytes, and plain
    expression constants will be more C-like in their interpretation.

 Makefile        |   16 ++++++++-------
 ast.ml          |   11 +++++++----
 ast_types.mli   |    3 ++-
 parser.mly      |   10 ++++++----
 parser_help.ml  |    5 +++++
 parser_help.mli |    6 ++++++
 scanner.mll     |    6 +++---
 7 files changed, 38 insertions(+), 19 deletions(-)

commit 03c361378ae507fae887147f1b22f85e51776cff
Author: Kevin Graney <nanonet@gmail.com>
Date:   Thu Jun 27 09:31:13 2013 -0400

    Add Sublime Text 2 project file

    Also update .gitignore to ignore other Sublime Text files.

 .gitignore           |    2 ++
 bawk.sublime-project |   21 +++++++++++++++++++++
 2 files changed, 23 insertions(+)

commit a345dd50a587d0b70d2709cf64a0908521ad2c4a
Author: Kevin Graney <nanonet@gmail.com>
Date:   Thu Jun 27 09:16:11 2013 -0400

    Remove recursive makefiles

    Changing to a flat structure for simplicity.

 Makefile          |    9 ++++++++-
 plt_docs/Makefile |    9 ---------
 2 files changed, 8 insertions(+), 10 deletions(-)

commit b950f353c6c05e0c8400cf63c0114da0b9d0142a
Author: Kevin Graney <nanonet@gmail.com>
Date:   Tue Jun 25 22:34:34 2013 -0400

    Work on LRM

 plt_docs/lrm.tex |   16 ++++++++++++---
 1 file changed, 13 insertions(+), 3 deletions(-)

commit 0f3edccfc005cc678f3e12d4a201f2f926a86ead
Author: Kevin Graney <nanonet@gmail.com>
Date:   Tue Jun 25 20:17:15 2013 -0400

    Work on LRM

 plt_docs/lrm.tex |   70 +++++++++++++++++++++++++++++++++----------------------
 1 file changed, 42 insertions(+), 28 deletions(-)
```

```
commit a05a96b731bd0492b2efe2cbfe709b7fcd72786e
Author: Kevin Graney <nanonet@gmail.com>
Date:   Mon Jun 24 22:46:32 2013 -0400

    Added to prototype.bawk test cases

 tests/prototype.bawk |    6 ++++++
 1 file changed, 6 insertions(+)

commit 9f52f5177a5b24b421b5e63fab59532a772e5278
Author: Kevin Graney <nanonet@gmail.com>
Date:   Mon Jun 24 22:46:09 2013 -0400

    LRM work

 plt_docs/lrm.tex |   45 +++++++++++++++++++++++++++++++++++++++------
 1 file changed, 39 insertions(+), 6 deletions(-)

commit 01cbe8ea67897d83343bc32614160a99d8abe291
Author: Kevin Graney <nanonet@gmail.com>
Date:   Mon Jun 24 22:45:16 2013 -0400

    Add string literals

    Added lexer/parser support for string literals. Parser support
    may need to be firmed up still (assignment to a string literal
    should not be allowed, so we need some kind of lvalue/rvalue
    distinction).

 ast.ml        |    3 +++
 ast_types.mli |    1 +
 parser.mly    |    2 ++
 scanner.mll   |   72 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++-
 4 files changed, 77 insertions(+), 1 deletion(-)

commit 840aa273da4bf579668a4cfdfae7ef5c40e0d825
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Jun 23 09:47:36 2013 -0400

    Fix warning about non-unit type

    Changed to use ignore in the print out of the GraphViz DOT file
    where a function with a side effect is called that returns a value
    we don't care about.

 ast.ml |    2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 16b81c4478d499ff5d68e9baf0fb2a79762df0af
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Jun 23 09:46:17 2013 -0400

    Add precedence and associativity for comparators

    Resolves a bunch of shift/reduce conflicts in the parser generator.
```

```
    The precedence ordering is taken from the microc example.

 parser.mly |    2 ++
 1 file changed, 2 insertions(+)

commit f8eb3945c676eb2bdd89424824854d6ec753173c
Author: Kevin Graney <nanonet@gmail.com>
Date:   Wed Jun 19 21:13:49 2013 -0400

    Initial test case

    A simple test case to test several different parts of the parser.
    Useful for looking at the AST GraphViz DOT output of.

 tests/prototype.bawk |   23 +++++++++++++++++++++++
 1 file changed, 23 insertions(+)

commit 9a51cf106b5cfa45da43624a578ae26b9454d415
Author: Kevin Graney <nanonet@gmail.com>
Date:   Wed Jun 19 21:13:30 2013 -0400

    Work on language reference manual

 plt_docs/lrm.tex |   92 ++++++++++++++++++++++++++++++++++++++++++++++++----
 1 file changed, 86 insertions(+), 6 deletions(-)

commit a173295ba72fd0e3b8561a1f144af84f28ec5616
Author: Kevin Graney <nanonet@gmail.com>
Date:   Wed Jun 19 21:11:59 2013 -0400

    Let identifiers start with underscore

 scanner.mll |    2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 7270f30448c43458653804ee80195b9145eecb92
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Jun 16 19:58:26 2013 -0400

    Add comparator operators

    Added equality comparison along with >, <, >=, <=, !=.

 ast.ml        |    6 ++++++
 ast_types.mli |    2 ++
 parser.mly    |    7 +++++++
 3 files changed, 15 insertions(+)

commit 93514726fb63d927360c2b0682a9648b276ad526
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jun 15 23:34:11 2013 -0400

    Allow function calls with no arguments

 parser.mly |    1 +
 1 file changed, 1 insertion(+)
```

```
commit 42d1ff15e1ed4a5223c6bcf03c95d51077b5c1a5
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jun 15 23:27:02 2013 -0400

    Make edge insertion consistent

    Some of the edge insertion for the GraphViz DOT generation of the
    AST was inconsistent.  Some functions assumed the caller made the
    edge and some assumed the callee, resulting in a few double or
    missing edges.  Now the caller is always required to create a
    node and a link to its parent.

 ast.ml |   16 +++++++----------
 1 file changed, 7 insertions(+), 9 deletions(-)

commit 5ce7f0fc576ff7b9a272c2363237ae9d49a4c0c3
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jun 15 23:11:21 2013 -0400

    Add function calls to AST

    Added parsing of function calls to the AST.  Identified an
    inconsistency in how parent edges are drawn in the GraphViz output,
    so that bug needs to be fixed in the future.

 ast.ml        |    7 +++++++
 ast_types.mli |    1 +
 parser.mly    |   10 +++++++++-
 scanner.mll   |    1 +
 4 files changed, 18 insertions(+), 1 deletion(-)

commit 386c4d95c38f1975579fce7e6152e568960da326
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jun 15 21:14:55 2013 -0400

    Fix bug with reverse ordered statement_list

    Fixed a bug in the parser where the statement_list was in reverse
    order, and the GraphViz DOT generating function was reversing it
    to hide the problem.  Should be correct now, with a Yacc pattern
    that reverses it in the parser.

 ast.ml     |    5 ++---
 parser.mly |    5 ++++-
 2 files changed, 6 insertions(+), 4 deletions(-)

commit 0f627538861edd0c9ecaf3f42fdae3426a521d27
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jun 15 20:16:03 2013 -0400

    Add support for decimal literals

    Added support for base 10 decimal literals.

 scanner.mll |    2 ++
```

```
 1 file changed, 2 insertions(+)

commit 8b7ed5b0dd1cd37eb6888f807ecd3c1c22c763ea
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jun 15 20:08:01 2013 -0400

    Add arithmetic binary operators

    Added addition, subtraction, multiplication, and division.  Had to
    define division in terms of the existing FSLASH token.  This makes
    the code a little unclear, and might be changed for clarity
    reasons in the future.

 ast.ml        |   11 ++++++++++
 ast_types.mli |    3 +++
 parser.mly    |   10 +++++++++-
 scanner.mll   |    2 +-
 4 files changed, 24 insertions(+), 2 deletions(-)

commit f5ed9c83e92888500ae3c5ca99f942cb67f3f9ed
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jun 15 19:36:03 2013 -0400

    Make output graph read left-to-right with input

    The ouput GraphViz DOT tree for the AST now reads left-to-right in
    order with the input lines of code.  Before it read right-to-left
    because the lists were not reversed before being passed to
    List.fold_left.

 ast.ml |    9 +++++----
 1 file changed, 5 insertions(+), 4 deletions(-)

commit 3fd147d56da971257ce3df971ba2c136ce3b555c
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jun 15 19:14:54 2013 -0400

    Add interfaces for ast.ml

    Added interface definitions and documentation for some of the
    functions in ast.ml.

 ast.mli |   26 ++++++++++++++++++++++++++
 1 file changed, 26 insertions(+)

commit 901689776b1286fdc994959ddf8c32c6f4e838cd
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jun 15 18:59:58 2013 -0400

    Add design_docs target to top-level Makefile

 Makefile |    6 +++++-
 1 file changed, 5 insertions(+), 1 deletion(-)

commit c49ed3e989557b9d5b32638972495963074f9122
Author: Kevin Graney <nanonet@gmail.com>
```

```
Date:   Sat Jun 15 18:59:31 2013 -0400

    Add pattern expressions to DOT output.

    Added pattern expression to the GraphViz DOT output of the AST.

 ast.ml |   28 ++++++++++++++++++++++++++-
 1 file changed, 27 insertions(+), 1 deletion(-)

commit 16fe0e54bba34ac41e52c74382a6f8ed32fd4ff3
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jun 15 18:58:24 2013 -0400

    Add expressions to DOT output

    Added the expressions to the GraphViz DOT output of the AST.

 ast.ml |   13 +++++++++++
 1 file changed, 13 insertions(+)

commit 569b897a0d9b42356f72fc51603f2bd1d86c1d3f
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jun 15 18:23:39 2013 -0400

    Add support for non-terminals to DOT output

    Added support for non-terminal nodes to the GraphViz dot output.

 ast.ml |   11 +++++++++--
 1 file changed, 9 insertions(+), 2 deletions(-)

commit b4b6df9eff32b4111f25569960620560b82487c0
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jun 15 17:10:09 2013 -0400

    Add additional punctuation to lexer

    Added additional punctuation symbols to lexer for binary operators
    as well as parenthesis.  Changed strings of length 1 to characters
    in pattern matching.

 parser.mly  |    9 ++++++++-
 scanner.mll |   23 +++++++++++++++++++-----
 2 files changed, 26 insertions(+), 6 deletions(-)

commit 466ff9a815d9c8b79df2216b9eae1345664259cc
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jun 15 16:18:24 2013 -0400

    Allow underscore in literal names

    Allow an underscore as the second and subsequent character in
    literal names.  In the future it may also be allowed as the first
    character.

 scanner.mll |    2 +-
```

```
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 4434b3157a6093affd36ffed17364181749eff84
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jun 15 16:18:03 2013 -0400

    Add Literals to the AST

 ast_types.mli |    1 +
 parser.mly    |    1 +
 2 files changed, 2 insertions(+)

commit 30fe72378962e036fbff55887fec066fd286b21a
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jun 15 16:14:31 2013 -0400

    Remove extraneous make targets in plt_docs

    Removed the lrm and proposal targets since it's just as easy
    to type "make lrm.pdf" and "make proposal.pdf".

 plt_docs/Makefile |    5 +----
 1 file changed, 1 insertion(+), 4 deletions(-)

commit 704d83a3797db42f08b46b00dd6cb2a3c31a8907
Author: Kevin Graney <nanonet@gmail.com>
Date:   Wed Jun 12 21:15:26 2013 -0400

    Add PLT documents

    Added documents for PLT (COMS W4115) to the repository.

 .gitignore            |    7 ++++
 plt_docs/Makefile     |   12 ++++++
 plt_docs/lrm.tex      |   27 ++++++++++++
 plt_docs/proposal.tex |  109 ++++++++++++++++++++++++++++++++++++++++++++++++++
 4 files changed, 155 insertions(+)

commit e500cbd1e47b791227ff13233cb0c38bf9940674
Author: Kevin Graney <nanonet@gmail.com>
Date:   Wed Jun 12 21:11:38 2013 -0400

    Renamed docs folder

    Renamed the 'docs' folder since it's a little too generic, and for
    this project several types of documentation are expected.  The
    ocamldoc documentation is now in a folder named 'design_docs'.

 .gitignore           |    4 ++--
 design_docs/Makefile |    2 ++
 docs/Makefile        |    2 --
 3 files changed, 4 insertions(+), 4 deletions(-)

commit 0158e187d22268255cca67ab6683321535cae3cb
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Jun 9 23:15:40 2013 -0400
```

```

    Clean up the parse tree dot code generation

    Cleaned up the print_tree function to abstract away some of the
    dot code generation into a couple routines.  Still need to expand
    the function to more types in the AST.

 ast.ml |   47 ++++++++++++++++++++++++++++-------------------
 1 file changed, 28 insertions(+), 19 deletions(-)

commit 691c33954f98ba711216915f3833f686eb98e852
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Jun 9 21:28:57 2013 -0400

    Change print_tree to print AST in dot form

    Changed the print_tree method to print out graphviz dot code to
    draw the tree.  This is still very crude, but it's much easier to
    visualize how the tree is being formed.

    To support this change the Ast_types.program type was removed, and
    now the root program is represented as a Block statement instead of
    a list of statements.  This should be a nicer overall
    representation anyway.  The parser handles the type conversion.

 ast.ml        |   37 +++++++++++++++++++++++++++++++++----
 ast.mli       |    2 +-
 ast_types.mli |    4 ++--
 parser.mly    |    4 ++--
 4 files changed, 38 insertions(+), 9 deletions(-)

commit d86ec95db986452cdc51e728b06e1e25a1a2c858
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Jun 9 19:32:45 2013 -0400

    Add variable bindings to pattern

    Added suppport for variable bindings to the pattern matching.

 ast_types.mli |   14 +++++++++++++-
 parser.mly    |   10 +++++++---
 scanner.mll   |   13 ++++++++-----
 3 files changed, 28 insertions(+), 9 deletions(-)

commit 39e981dd9607dd6354adec46ceef31885dfd7024
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Jun 9 19:28:19 2013 -0400

    Change hex literals to require 0x prefix

    Changed hex literals to require 0x prefix.  This avoids ambiguity
    with variable name literals, which are required to start with a
    letter.  (e.g. is 'ff' a variable or 255?)

 parser.mly  |    4 ++--
 scanner.mll |    4 ++--
```

```
 2 files changed, 4 insertions(+), 4 deletions(-)

commit ee673b1bcb29a7986af13e8050e096ed697b067f
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Jun 9 18:19:43 2013 -0400

    Add pattern syntax to parser

    First cut at getting the patten matching syntax into the parser.

 parser.mly |    4 +---
 1 file changed, 1 insertion(+), 3 deletions(-)

commit 04ff7d64bb1eae6be83abe054d3a5dbe0596d710
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Jun 9 18:09:17 2013 -0400

    Ignore auto-generated documentation

 .gitignore |    2 ++
 1 file changed, 2 insertions(+)

commit 128bd7649ab921477918371202fe9900f124ef38
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Jun 9 12:58:16 2013 -0400

    Update ocamldep dependencies

    Updated ocamldep dependencies.  These remain static for now.

 Makefile |   26 ++++++++++++++++----------
 1 file changed, 16 insertions(+), 10 deletions(-)

commit 49ac872a829372682f68b1a1ca7bbb57ebde926c
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Jun 9 12:56:52 2013 -0400

    Add block statements to grammar

    Added support for block statements to the grammar, and began
    adding some scanner ability for the patterns.  The bawk main loop
    is just dummy logic that prints 'item' for each statement it
    encounters.

 ast.ml        |    7 +++++++
 ast.mli       |    4 +---
 ast_types.mli |   12 ++++++++++++
 bawk.ml       |   16 ++++++++++++----
 parser.mly    |   25 ++++++++++++++++++++++----
 scanner.mll   |    2 +-
 6 files changed, 54 insertions(+), 12 deletions(-)

commit 916197c3490919ccef12d77e9701a88a0b87c3b8
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sun Jun 9 12:09:11 2013 -0400
```

```
    Add .gitignore

    Ignore OCaml build files

 .gitignore |   11 ++++++++++
 1 file changed, 11 insertions(+)

commit 56c8597a3730aeb0a4d92c21379ce09974b9f418
Author: Kevin Graney <nanonet@gmail.com>
Date:   Sat Jun 8 22:45:30 2013 -0400

    Initial commit

    Basic parser for reading in a hexadecimal string and printing the
    value in hex.

 Makefile      |   32 ++++++++++++++++++++++++++++++++
 ast.mli       |    4 ++++
 bawk.ml       |    9 +++++++++
 docs/Makefile |    2 ++
 parser.mly    |   18 ++++++++++++++++++
 scanner.mll   |   30 ++++++++++++++++++++++++++++++
 utile.ml      |    5 +++++
 utile.mli     |    5 +++++
 8 files changed, 105 insertions(+)
```

Chapter 5

Architectual design

5.1 Design decisions

The design decisions for this project were not made with considerations towards performance or scalability. Design decisions were based around ease of implementation and demonstration of compiler concepts. For this reason some features of a practical language were excluded because they cost significant effort for minimal benefit to the experience of writing a compiler.

5.2 Execution phases

The bawk interpreter consists of a bytecode compiler and a bytecode runtime environment. The two are seamlessly integrated into a single executable program, `bawk`. By calling this program with a `-c` option the results of the bytecode compiler are output, and by calling it with no option (or an explicit `-e`) the the given program is compiled internally to bytecode and then immediately executed in the bytecode interpreter.

5.2.1 Bytecode compiler

The bytecode compiler consists of a scanner (generated by OCamllex), a parser (generated by OCamlyacc), and the compiler itself. The scanner parses a sequence of input characters that form a given bawk language program and returns a list of tokens. The token types include punctuation that has meaning in the language, the C-style comments (`/*...*/`), identifiers, keywords, constants, and string literals. Whitespace is consumed by the scanner, but serves no purpose other than to separate tokens.

There is at least one instance of ambiguity in the scanner. Identifier names, as defined in §3.1.2, consist of a letter or underscore followed by letters, underscores, and/or numerals. However, constants in pattern expressions, as defined in §3.5.1, consist of a sequence of hexadecimal digits, possibly including the letters 'a'-'f' in uppercase or lowercase. This leads to possible confusion in the code chunk listed below. On line 2 it is not known if `cafe123` is a variable name or a hexadecimal number. This ambiguity is actually resolved by the parser, which treats ambiguous constants as constants if they are valid hexadecimal.

Listing 5.1: Possibly ambiguous code

```
/cafe123:int1/ {
    /cafe123/ {
        /* do_something */
    }
}
```

The parser converts the sequence of tokens into an abstract syntax tree (AST). The types used in the AST are defined in `ast_types.mli`, and rely on the OCaml tagged union types to represent the tree. The OCamlyacc specification for the parser is defined in the `parser.mly` file. The parser definition is fairly straight forward. There is some conversion done at this stage from strings to integers, but primarily it simply builds the AST using the tagged untion types defined in the `Ast_types` module. The root of the generated AST is a `statement`, specifically a `Ast_types.Block` type, which contains a list of `statements`.

The `bawk` compiler can output the AST in a GraphViz format, which is useful for visualizing and debugging the AST. The `-ast` flag to the compiler will output GraphViz code. This can be turned into a PDF by piping it to GraphViz using the command

```
bawk -ast [binary filename] < [source filename] | dot -Tpdf -o out-file.pdf.
```

The logic to generate the GraphViz code is the majority of the `ast.ml` file. Other functions in that file serve to provide a few helper functions used when parsing the AST.

In the translation phase of the compiler, most of the heavy lifting is done by translating AST structures into basic blocks. In addition, the compiler keeps track of a a symbol table in the `env` type in `compile.ml`. The structure forms a linked list, with a new node added each time a new, nested scope, is entered. Each node has a pointer to the parent scope, and a `StringMap` from symbols to either (1) the jump address for a subroutine, or (2) the index into globals. The environment also contains additional information for binding variables, which includes the size of the binding.

5.2.2 Bytecode operations

The compiler takes the AST created by the parser portion and translates it to a custom stack-based bytecode. The instructions for the bytecode are defined in `bytecode_types.mli`. Most of the instructions are related directly to the stack, but several are designed specifically for the application of this language: reading binary files.

The translation phase from the AST to bytecode is really a two phase translation process. The first does the basic translation from the AST to basic blocks of bytecode. In this initial translation the various branch instructions are given a unique value referencing a `Label` instruction. The `Label` instruction is a special pseudo instruction that is removed in the second phase of the translation. When the `Label` instructions are removed, their address, or position in the instruction list, is noted. These addresses then replace the values in all the branch instructions, converting them from the value referencing a label to the address of that label.

All the branch instructions in the bytecode operate on absolute addresses. There is no relative addressing in the bytecode by design, simply for simplicity. After the second phase of the translation is complete, the `Label` instructions removed and branch instructions pointing to addresses, the bytecode is ready for the runtime interpreter.

5.2.3 Bytecode runtime

The runtime interpreter is fairly simple, and defined primarily in `bytecode.ml`. The runtime executes the stack-based bytecode in a virtual machine environment. For simplicity the bytecode instructions composing the program, the execution stack, and global variable values are stored in three different arrays: `instructions`, `stack`, and `globals` respectively. Execution is performed by the recursive `exec` function, which takes a frame pointer, a stack pointer, and a program counter.

The program counter is simply an index into the `instructions` array for the instruction being executed in the current cycle. When the recursive call to `exec` is made, to execute the next instruction, the program counter is either incremented by one to run the next sequential instruction, or, in the case of a branch or function return instruction, called with some absolute address from the stack or an instruction argument. Since all branching is to absolute addresses the bytecode interpreter never computes relative offsets from the current program counter.

The stack grows from an index of zero in the `stack` array upwards, with items pushed and popped as individual instructions are executed. The argument to `exec` is a pointer to where the next value can be pushed on the stack. The stack only holds integer values, which is the primary motivation for not supporting functions, such as a `printf`-like function, that take string arguments.

Function arguments are placed on the stack, and the frame pointer references the location of these arguments. The previous frame pointer is also placed onto the stack and reset when the function returns, allowing nested function calls with arguments to work properly.

Global variables are stored in the `globals` array, with each symbol, other than function arguments, assigned an offset into this array. The name 'globals' is a bit of a misnomer, since variables local to blocks or functions are also stored in this same array. The slots in the `globals` array are not re-used; they last the entire duration of the program and are statically assigned to symbols by the compiler.

The bytecode interpreter `stack` is limited to 1024 values, and the `globals` array is limited to this same size of 1024 values. These limits are *not* enforced by the compiler, but the bytecode interpreter will crash if they are exceeded. The length of the `instructions` array is not explicitly limited.

5.3 Limitations

One major limitation of the language design is the lack of any type of `return` statement or return value from a function. This prevents the use of many types of useful recursion, and really limits the utility of the language. See §7.3.1 for more details on this limitation.

Chapter 6

Test plan

The testing for this project was performed using a variety of very tiny bawk language programs designed to exercise various features of the language. These programs are provided with the source code in the `tests` directory. Each program is named `test-<name>.bawk` and its output is expected to be identical to the `test-<name>.bawk.out` file. Both the source and expected output files are listed in §8.2.

The tests are all designed to operate on the binary file `tests/lichtenstein.png`, which serves as a modestly sized sample file. The beginning of the `lichtenstein.png` file is shown below for convenience, and the file itself can be found packaged with the bawk source.

```
0000000: 8950 4e47 0d0a 1a0a 0000 000d 4948 4452  .PNG........IHDR
0000010: 0000 0200 0000 0200 0802 0000 007b 1a43  .............{.C
0000020: ad00 0000 0970 4859 7300 000a f000 000a  .....pHYs.......
0000030: f001 42ac 3498 0000 0007 7449 4d45 07d7  ..B.4.....tIME..
0000040: 0511 0921 0919 38af 7500 0020 0049 4441  ...!..8.u.. .IDA
0000050: 5478 dab4 bd5b ac6d 5b76 1dd4 5aef 63ae  Tx...[.m[v..Z.c.
0000060: bdcf bdb7 1ed7 a9b2 2b71 d995 38b1 63f2  ........+q..8.c.
0000070: b021 4fe7 0181 8404 a280 1421 10e2 2522  .!O........!..%"
0000080: 7ef8 46fc 874f 7ef9 e40f 81f8 43e1 2bca  ~.F..O~.....C.+.
0000090: 4f82 2148 0ea0 5809 2838 ca8b d871 1c95  O.!H..X.(8...q..
00000a0: 53b6 eb75 5fe7 ec35 476f 7cf4 3ec6 1c73  S..u_..5Go|.>..s
00000b0: edb5 cfbd 65c8 aeab 5be7 eeb3 f75a 73cd  ....e...[....Zs.
00000c0: 3946 1fbd b7d6 7aeb fcbd fff6 7f2b 7a17  9F....z......+z.
00000d0: 4220 626b 6c06 2a48 9a39 e980 0114 0420  B bkl.*H.9.....
00000e0: 22f6 d09b c075 8fab 60f4 2638 f317 ed72  "....u..`.&8...r
00000f0: d1e6 7cd8 daa5 b953 4e38 e194 2481 8289  ..|....SN8..$...
...
0059270: 3122 4a04 2150 a6c8 ff1f 00c1 c91a 2379  1"J.!P........#y
0059280: 1a27 0000 0000 4945 4e44 ae42 6082       .'....IEND.B`.
```

The test programs are executed by a shell script, `run_tests.sh`, which runs each program and compares its output to the expected values.

Chapter 7

Lessons learned

This project provided a number of valuable software engineering and language design lessons. The language design lessons were invaluable, as this was my first attempt at writing a compiler from scratch that had any level of sophistication beyond regular expression matching on text files.

7.1 Language design

The design of the language was critical to being successful with this project. It was crucial to really think through the specifics of the language before even thinking about writing the compiler's translation logic. It was critical to write the scanner and parser in parallel with hashing out the language details. Writing these helped force the resolution of many shift/reduce ambiguities and couple reduce/reduce ambiguities in the original, pre-parser, design.

One visible addition to the language that came about from writing the parser is the `def` keyword before function definitions. This was added to resolve a conflict in the parser. Similarly, when adding if statements I realized that parenthesis were required around the conditional expression when a parser conflict was resolved after adding them.

7.2 Scanner and parser generators

As described in §5.2.1, there is some ambiguity in the scanner distinguishing between hexadecimal values and identifier names. I didn't realize this ambiguity in the initial language design until testing the compiler with a hexadecimal constant beginning with a letter. This ambiguity was obviously not detected as a shift/reduce or reduce/reduce conflict by the parser generator because it was a collision of definitions in the scanner. Luckily it was possible to change the language definition slightly and add some logic in the parser to resolve the ambiguity.

7.2.1 GraphViz output

The GraphViz output was very useful in the early stages of writing the scanner and parser, but as it became clear that the basic design was solid the utility of the visualization decreased. Despite the decreasing returns on invested time, it wasn't a whole lot more difficult than implementing an ASCII format of AST output.

7.3 Compiler and bytecode design

7.3.1 Return statement problem

Late in the project I realized that I did not implement any type of `return` statement in the langauge. This was an oversight, but unfortunately the design of the bytecode did not make it easy to add at the last minute. The bytecode instruction `Rts` returns from a subroutine by popping the arguments to the function off the stack. Unfortunately this instruction requires a hard-coded constant for the number of instructions to pop. Without changing the bytecode and its interpreter to support reading this value from the stack it is difficult to return from an arbitrary point in the function. It could likely still be accomplished without a bytecode interpreter change, but it would require some redesign of the `translate` function in `compile.ml` to pass around additional information, namely a `Label` number for the end of the function. This change was just too large to risk making in the last week of the semester, so the language stands without a `return` statement.

7.3.2 Lack of full support for string types

One way to improve the bawk language would be to add support for strings inside of expressions. This would facilitate adding support for functions such as `printf` and also the reading of a fixed-length or null-terminated string from a file. Support from strings in this way was one of the initial conceptual ideas behind the language, but late in the semester it became apparent that the bytecode interpreter would have to be overhauled to support both integers and strings on the stack and in the globals array.

Chapter 8

Source code listing

8.1 Compiler and bytecode interpreter

8.1.1 Scanner & parser definitions

scanner.mll

The `scanner.mll` file contains Ocamllex specifications for the bawk scanner.

Listing 8.1: scanner.mll

```
{ open Parser

let initial_string_buffer = String.create 256
let string_buff = ref initial_string_buffer
let string_index = ref 0

let is_in_string = ref false
let in_string () = !is_in_string

let reset_string_buffer () =
    string_buff := initial_string_buffer;
    string_index := 0

let store_string_char c =
    if !string_index >= String.length (!string_buff) then
    begin
        let new_buff = String.create (String.length (!string_buff) * 2) in
            String.blit (!string_buff) 0 new_buff 0 (String.length (!string_buff));
            string_buff := new_buff
    end;
    String.unsafe_set (!string_buff) (!string_index) c;
    incr string_index

let store_lexeme lexbuf =
    let s = Lexing.lexeme lexbuf in
    for i = 0 to String.length s - 1 do
        store_string_char s.[i];
    done

let get_stored_string () =
```

```
31      let s = String.sub (!string_buff) 0 (!string_index) in
32      string_buff := initial_string_buffer;
33      s
34
35  let char_for_backslash = function
36      | 'n' -> '\010'
37      | 'r' -> '\013'
38      | 'b' -> '\008'
39      | 't' -> '\009'
40      | c -> c
41
42  }
43
44  let hex_character = ['0'-'9' 'A'-'F' 'a'-'f']
45  let hex_sequence = "0x" hex_character+
46  let hexdigit_sequence = hex_character+
47  let literal = ['A'-'Z' 'a'-'z' '_']['0'-'9' 'A'-'Z' 'a'-'z' '_']*
48
49  let newline = ('\010' | "\013\010")
50
51  rule token = parse
52      | [' ' '\t' '\r' '\n'] { token lexbuf }
53      | "/*" { comment lexbuf }
54
55      (* language semantics *)
56      | ':'  { COLON }
57      | ';'  { SEMICOLON }
58      | ','  { COMMA }
59      | '/'  { FSLASH }
60      | '{'  { LBRACE }
61      | '}'  { RBRACE }
62      | '('  { LPAREN }
63      | ')'  { RPAREN }
64      | '+'  { PLUS }
65      | '-'  { MINUS }
66      | '*'  { TIMES }
67        (* DIVIDE is the same as FSLASH *)
68      | '='  { ASSIGN }
69      | "==" { EQ }
70      | "!=" { NEQ }
71      | '<'  { LT }
72      | "<=" { LEQ }
73      | '>'  { GT }
74      | ">=" { GEQ }
75
76      (* type definitions *)
77      | "int1" { BIND_TYPE(Ast_types.Int_1_byte) }
78      | "int2" { BIND_TYPE(Ast_types.Int_2_bytes) }
79      | "int4" { BIND_TYPE(Ast_types.Int_4_bytes) }
80      | "uint1" { BIND_TYPE(Ast_types.UInt_1_byte) }
81      | "uint2" { BIND_TYPE(Ast_types.UInt_2_bytes) }
82      | "uint4" { BIND_TYPE(Ast_types.UInt_4_bytes) }
83
84      | "if" { IF }
85      | "else" { ELSE }
86
```

```
87      | "def" { DEF }
88      | literal as lit { LITERAL(lit) }
89
90      | hexdigit_sequence as lit { INT_LITERAL(lit)}
91      | hex_sequence as lit { INT_LITERAL(lit) }
92
93      (* string literals *)
94      | "\"" { reset_string_buffer();
95               is_in_string := true;
96               let string_start = lexbuf.Lexing.lex_start_p in
97               string lexbuf;
98               is_in_string := false;
99               lexbuf.Lexing.lex_start_p <- string_start;
100              STRING_LITERAL(get_stored_string()) }
101
102     | eof { EOF }
103
104
105 and comment = parse
106     | "*/" { token lexbuf }
107     | _ { comment lexbuf }
108
109 and string = parse
110     '"'
111       { () }
112   | '\\' newline ([' ' '\t'])
113       { string lexbuf }
114   | '\\' ['\\' '\'' '"' 'n' 't' 'b' 'r' ' ']
115       { store_string_char(char_for_backslash(Lexing.lexeme_char lexbuf 1));
116         string lexbuf }
117   | eof
118       { is_in_string := false;
119         assert false
120         (*raise (Error (Unterminated_string, 0))*) }
121   | _
122       { store_string_char(Lexing.lexeme_char lexbuf 0);
123         string lexbuf }
```

parser.mly

The `parser.mly` file contains Ocamlyacc specifications for the bawk parser.

Listing 8.2: parser.mly

```
1  %{
2  open Parser_help
3  %}
4
5  %token <Ast_types.bind_type> BIND_TYPE
6
7  /* punctuation */
8  %token COLON SEMICOLON COMMA FSLASH RBRACE LBRACE RPAREN LPAREN DEF IF ELSE
9
10 /* operators */
11 %token PLUS MINUS TIMES ASSIGN
```

```
12
13 /* comparators */
14 %token EQ NEQ LT LEQ GT GEQ
15
16 %token <string> STRING_LITERAL
17 %token <string> LITERAL
18 %token <string> INT_LITERAL
19 %token EOF
20
21 /* associativity and precedence */
22 %nonassoc NOELSE
23 %nonassoc ELSE
24 %right ASSIGN
25 %left EQ NEQ
26 %left LT GT LEQ GEQ
27 %left PLUS MINUS
28 %left TIMES FSLASH
29
30 %start program
31 %type <Ast_types.statement> program
32
33 %%
34
35 pat_token:
36     | INT_LITERAL { parse_pattern_const $1 }
37     | LITERAL COLON BIND_TYPE { Ast_types.Binding($1, $3) }
38     | LITERAL {
39         (* Resolve a little ambiguity from the scanning phase, basically
40            binding variables can't be valid hex numbers or we don't know if it's
41            a hex number or a variable.  Here it's resolved to be a number. *)
42         try parse_pattern_const $1
43         with Failure(_) -> Ast_types.Literal($1)
44         }
45     | STRING_LITERAL { Ast_types.PatString($1) }
46
47 pat_expr:
48     rev_pat_expr { List.rev $1 }
49
50 rev_pat_expr:
51     | { [] }
52     | rev_pat_expr pat_token { $2 :: $1 }
53
54 statement:
55     | expr SEMICOLON { Ast_types.Expr($1) }
56     | LBRACE statement_list RBRACE { Ast_types.Block($2) }
57     | FSLASH pat_expr FSLASH statement { Ast_types.Pattern($2,$4) }
58     | DEF LITERAL LPAREN lit_list RPAREN LBRACE statement_list RBRACE
59         { Ast_types.FunctionDecl({
60             Ast_types.fname = $2;
61             Ast_types.arguments = $4;
62             Ast_types.body = Ast_types.Block($7)})
63         }
64     | IF LPAREN expr RPAREN statement %prec NOELSE
65         { Ast_types.If($3, $5, Ast_types.Block([])) }
66     | IF LPAREN expr RPAREN statement ELSE statement
67         { Ast_types.If($3, $5, $7) }
```

```
68
69 expr:
70     | INT_LITERAL { Ast_types.LitInt(int_of_string $1) }
71     | expr PLUS   expr { Ast_types.Binopt($1, Ast_types.Add, $3) }
72     | expr MINUS  expr { Ast_types.Binopt($1, Ast_types.Subtract, $3) }
73     | expr TIMES  expr { Ast_types.Binopt($1, Ast_types.Multiply, $3) }
74     | expr FSLASH expr { Ast_types.Binopt($1, Ast_types.Divide, $3) }
75     | expr EQ     expr { Ast_types.Binopt($1, Ast_types.Equal, $3) }
76     | expr NEQ    expr { Ast_types.Binopt($1, Ast_types.Neq, $3) }
77     | expr LT     expr { Ast_types.Binopt($1, Ast_types.Less, $3) }
78     | expr LEQ    expr { Ast_types.Binopt($1, Ast_types.Leq, $3) }
79     | expr GT     expr { Ast_types.Binopt($1, Ast_types.Greater, $3) }
80     | expr GEQ    expr { Ast_types.Binopt($1, Ast_types.Geq, $3) }
81     | LITERAL ASSIGN expr { Ast_types.Assign($1, $3) }
82     | LITERAL { Ast_types.ExprLiteral($1) }
83     | LITERAL LPAREN expr_list RPAREN { Ast_types.Call($1, $3) }
84     | STRING_LITERAL { Ast_types.LitString ($1) }
85
86 expr_list:
87     | rev_expr_list { List.rev $1 }
88
89 rev_expr_list:
90     | { [] }
91     | expr { [$1] }
92     | rev_expr_list COMMA expr { $3 :: $1 }
93
94 statement_list:
95     | rev_statement_list { List.rev $1 }
96
97 rev_statement_list:
98     | { [] }
99     | rev_statement_list statement { $2 :: $1 }
100
101 lit_list:
102     | rev_lit_list { List.rev $1 }
103
104 rev_lit_list:
105     | { [] }
106     | LITERAL { [$1] }
107     | rev_lit_list COMMA LITERAL { $3 :: $1 }
108
109 program:
110     statement_list { Ast_types.Block($1) }
```

8.1.2 Interfaces

Listing 8.3: parser_help.mli

```
1
2
3 (** [parse_pattern_const] converts a pattern constant into a [PatternBytes]
4     list of integers for the AST.  This method interprets leading zeros as
5     significant among other things. *)
6 val parse_pattern_const: string -> Ast_types.pat_token
```

Listing 8.4: ast_types.mli

```
(** Types used by the AST *)

type operators = Add | Subtract | Multiply | Divide
    | Equal | Neq | Less | Leq | Greater | Geq

type bind_type =
    | Int_1_byte
    | Int_2_bytes
    | Int_4_bytes
    | UInt_1_byte
    | UInt_2_bytes
    | UInt_4_bytes

type literal = string

type expr =
    | LitInt of int
    | ExprLiteral of string
    | Binopt of expr * operators * expr
    | Call of string * expr list
    | LitString of string
    | Assign of string * expr

type pat_token =
    | PatternByte of int
    | PatternBytes of pat_token list
    | Binding of literal * bind_type
    | Literal of literal
    | PatString of string

type pat_expr = pat_token list

type func_decl = {
    fname: string;
    arguments: string list;
    body: statement;
}
and statement =
    | Pattern of pat_expr * statement
    | Block of statement list
    | Expr of expr
    | FunctionDecl of func_decl
    | If of expr * statement * statement
```

Listing 8.5: bytecode_types.mli

```

type instruction =
      Lit of int    (* Push a literal *)
    | Drp           (* Discard a value *)
    | Bin of Ast_types.operators (* Perform arithmetic on top of stack *)
    | Two of int    (* Convert top of stack to two's complement *)
    | Lod of int    (* Fetch global variable *)
    | Str of int    (* Store global variable *)
    | Lfp           (* Load frame pointer from the stack *)
```

```
10      | Sfp           (* Store frame pointer to the stack *)
11      | Jsr of int    (* Call function by absolute address *)
12      | Ent           (* Push FP, FP -> SP, SP += i *)
13      | Rts of int    (* Restore FP, SP, consume formals, push result *)
14      | Beq of int    (* Branch absolute if top-of-stack is zero *)
15      | Bne of int    (* Branch absolute if top-of-stack is non-zero *)
16      | Bra of int    (* Branch absolute *)
17      | Beo of int    (* Branch if end of file *)
18      | Hlt           (* Terminate *)
19      | Rdb of int    (* Read a number of bytes from the file *)
20      | Ldp           (* Load the position in the file onto the stack *)
21      | Skp           (* Seek the file to the position on the stack *)
22
23      (* Pseudo bytecode instructions used by compiler *)
24      | Label of int  (* Used for branching jumps *)
```

Listing 8.6: ast.mli

```
1
2
3  (** [print_tree] outputs the AST in GraphViz DOT format.  This is useful for
4      visualizing how the parser is constructing the tree *)
5  val print_tree: Ast_types.statement -> unit
6
7  (** [string_of_bind_type] converts an [Ast_types.bind_type] to a [string] *)
8  val string_of_bind_type: Ast_types.bind_type -> string
9
10 (** [size_of_bind_type] returns the size, in bytes, of an [Ast_types.bind_type]
11     *)
12 val size_of_bind_type: Ast_types.bind_type -> int
13
14 (** [is_signed_type] returns true if the bind type is signed and false
15     otherwise *)
16 val is_signed_type: Ast_types.bind_type -> bool
17
18 (** [make_terminal] creates a terminal node for the node with an id of [this],
19     and labels it with the string argument.  If the optional [parent] argument
20     is provided a link is also created between this node and the parent. *)
21 val make_terminal: this:int -> ?parent:int -> string -> unit
22
23 (** [make_nonterminal] creates a non-terminal node for the node with an id of
24     [this], and labels it with the string argument.  If the optional [parent]
25     argument is provided a link is also created between this node and the
26     parent. *)
27 val make_nonterminal: this:int -> ?parent:int -> string -> unit
28
29 (** [make_link] creates a link between two of the GraphViz nodes *)
30 val make_link: int -> int -> unit
31
32 (** [folded_printer] returns a function that can be used with [List.fold_left]
33     to traverse a list of objects that should be children of a common parent.
34     The parent's ID is the second argument and the print function is the
35     first argument. *)
36 val folded_printer: ('a -> int -> int -> 'b) -> int -> int -> 'a -> 'b
37
38 (** [string_of_operator] returns a string of the bytecode instruction for
```

```
39      the given operator. *)
40  val string_of_operator: Ast_types.operators -> string
```

Listing 8.7: compile.mli

```
1   (** Compilation of bawk code *)
2
3   module StringMap : (* TODO: figure out how to get rid of this monstrosity! *)
4     sig
5       type key = String.t
6       type 'a t = 'a Map.Make(String).t
7       val empty : 'a t
8       val is_empty : 'a t -> bool
9       val mem : key -> 'a t -> bool
10      val add : key -> 'a -> 'a t -> 'a t
11      val singleton : key -> 'a -> 'a t
12      val remove : key -> 'a t -> 'a t
13      val merge :
14        (key -> 'a option -> 'b option -> 'c option) -> 'a t -> 'b t -> 'c t
15      val compare : ('a -> 'a -> int) -> 'a t -> 'a t -> int
16      val equal : ('a -> 'a -> bool) -> 'a t -> 'a t -> bool
17      val iter : (key -> 'a -> unit) -> 'a t -> unit
18      val fold : (key -> 'a -> 'b -> 'b) -> 'a t -> 'b -> 'b
19      val for_all : (key -> 'a -> bool) -> 'a t -> bool
20      val exists : (key -> 'a -> bool) -> 'a t -> bool
21      val filter : (key -> 'a -> bool) -> 'a t -> 'a t
22      val partition : (key -> 'a -> bool) -> 'a t -> 'a t * 'a t
23      val cardinal : 'a t -> int
24      val bindings : 'a t -> (key * 'a) list
25      val min_binding : 'a t -> key * 'a
26      val max_binding : 'a t -> key * 'a
27      val choose : 'a t -> key * 'a
28      val split : key -> 'a t -> 'a t * 'a option * 'a t
29      val find : key -> 'a t -> 'a
30      val map : ('a -> 'b) -> 'a t -> 'b t
31      val mapi : (key -> 'a -> 'b) -> 'a t -> 'b t
32    end
33
34
35  type pattern_binding = {
36      loc: int;
37      size: int;
38  }
39
40  type env = {
41      symbol_map: int StringMap.t;
42      bindings: pattern_binding StringMap.t;
43      parent: env ref option;
44  }
45
46  (** [translate_program] takes an AST statement and returns a list of bytecode
47      instructions implementing the statement *)
48  val translate_program: Ast_types.statement -> Bytecode_types.instruction list
49
50  val clean_environment: env
```

Listing 8.8: bytecode.mli

```
(** Bytecode generation *)

(** [execute_bytecode] is a function called from the main argument parser.  It
    generates and then executes the bytecode instructions. *)
val execute_bytecode: Bytecode_types.instruction list -> in_channel -> unit

(** [print_bytecode] is a function called from the main argument parser.  It
    generates and then prints out the bytecode instructions. *)
val print_bytecode: Bytecode_types.instruction list -> unit

(** [string_of_instruction] converts a bytecode instruction into a string
    representation *)
val string_of_instruction: Bytecode_types.instruction -> string

(** [execute_instructions] interprets an array of bytecode instructions
    as a program, executing said instructions. *)
val execute_instructions: Bytecode_types.instruction array -> in_channel -> unit

(** [is_pseudo] returns true if the given bytecode instruction is a pseudo
    instruction and false otherwise *)
val is_pseudo: Bytecode_types.instruction -> bool

val enumerate_instructions: Bytecode_types.instruction list ->
    (int * Bytecode_types.instruction) list
```

Listing 8.9: reader.mli

```
(** The Reader module contains functions for extracting and decoding bits of
    information from an [in_channel] for a given binary file. *)


(** [read_byte] reads a single byte from the input channel and returns an
    integer representing that byte *)
val read_byte: in_channel -> int

(** [read_bytes] reads the next [n] bytes from the input channel into a list of
    integers *)
val read_bytes: in_channel -> int -> int list

(** [read_string_null] reads a null-terminated string from the input channel *)
val read_string_null: in_channel -> string

(** [read_string_fixed] reads a fixed-length string from the input channel *)
val read_string_fixed: in_channel -> int -> string

(** [read_unsigned] reads the next [n] bytes as an unsigned integer

    The algorithm is: read bytes into a list with [read_bytes], traverse the
    list in reverse order, and accumulate a sum of every byte shifted by it's
    position in the list. *)
val read_unsigned: in_channel -> int -> int

(** [get_pos] returns the position in the file *)
val get_pos: in_channel -> int

```

```
29 (** [set_pos] set the position in the file *)
30 val set_pos: in_channel -> int -> unit
31
32 val advance: in_channel -> int -> unit
33
34 (** [is_eof] returns true if the next byte of the file reaches EOF and false
35     otherwise *)
36 val is_eof: in_channel -> bool
```

Listing 8.10: utile.mli

```
1  (** General purpose utility functions *)
2
3
4  (** [int_of_hex] converts a string of hexadecimal digits into an integer *)
5  val int_of_hex: string -> int
6
7  (** [enumerate] converts a list of values in a list of tuples of an integer
8      followed by that value.  Integers begin at zero and count upwards *)
9  val enumerate: ?step:('a -> int -> int) -> ?start:(int) -> 'a list
10     -> (int * 'a) list
11
12 (** [explode] converts a string into a list of characters *)
13 val explode: string -> char list
14
15 (** [implode] converts a list of characters into a string *)
16 val implode: char list -> string
17
18 (** [bytes_of_string] converts a string into a list of integers representing
19     the byte values for each character *)
20 val bytes_of_string: string -> int list
21
22 (** [signed_of_unsigned] converts an integer into its twos' complement
23     interpretation *)
24 val signed_of_unsigned: int -> int -> int
```

8.1.3 Source Files

Listing 8.11: bawk.ml

```
1  (** The main function of the bawk compiler *)
2  open Compile
3
4  module StringMap = Map.Make(String) (* TODO: can we use the other def? *)
5
6  (** Possible actions for the compiler to take *)
7  type action = Ast | Compile | Execute
8      | D_print_clean_env
9
10 (** [decode_action] reads the [argv] array of command line arguments and returns
11     the action that the compiler is being asked to take.  This is fairly
12     primitive, and only the first argument is used to make the decision. *)
13 let decode_action argv =
14     if Array.length argv > 1 then
```

```
15          List.assoc argv.(1) [
16              ("-ast", Ast);
17              ("-c", Compile);
18              ("-e", Execute);
19              ("-D-print-clean-env", D_print_clean_env)
20          ]
21      else Execute;;
22
23  let decode_in_file argv =
24      if Array.length argv > 2 then
25          open_in_bin argv.(2)
26      else
27          open_in_bin "/dev/random";;
28
29
30  (** [parse_channel] runs an input channel through the lexer and parser phases
31      of the compiler returning an AST. *)
32  let parse_channel channel =
33      let lexbuf = Lexing.from_channel channel in
34      let program = Parser.program Scanner.token lexbuf in
35      program;;
36
37  let _ =
38      let action = decode_action Sys.argv in
39      let in_file = decode_in_file Sys.argv in
40      match action with
41            Ast -> Ast.print_tree (parse_channel stdin)
42          | Compile -> Bytecode.print_bytecode
43              (Compile.translate_program (parse_channel stdin))
44          | Execute -> Bytecode.execute_bytecode
45              (Compile.translate_program (parse_channel stdin)) in_file
46
47          (* Debug actions *)
48          | D_print_clean_env ->
49              let env = Compile.clean_environment in
50              Printf.printf "Starting symbol table:\n";
51              StringMap.iter (fun s i -> Printf.printf "\t%d %s\n" i s)
52                      env.symbol_map;;
```

Listing 8.12: parser_help.ml

```
1
2   let split_bytes s =
3       let split s =
4           let num_bytes = (String.length s) / 2 in
5           let arr = Array.make num_bytes (String.create 2) in
6           for i = 0 to num_bytes - 1 do
7               arr.(i) <- String.create 2;
8               String.blit s (2*i) arr.(i) 0 2
9           done;
10          Array.to_list arr
11      in
12      let pieces =
13          if (String.length s) mod 2 == 0 then split s
14          else split (Printf.sprintf "0%s" s)
15      in
```

```
    List.map (fun x -> int_of_string ("0x" ^ x)) pieces;;

let parse_pattern_const s =
    let lst = split_bytes s in
    let bytes = List.map (fun x -> Ast_types.PatternByte x) lst in
    Ast_types.PatternBytes bytes
```

Listing 8.13: ast.ml

```
open Ast_types
open Printf

let make_link n1 n2 =
    printf "%d -> %d\n" n1 n2;;

let conditional_link_with_parent this_id parent_id =
    if parent_id != -1 then make_link parent_id this_id
    else ();;

let make_terminal ~this:this_id ?parent:(parent_id=(-1)) label =
    printf "%d [label=\"%s\" shape=hexagon style=filled]\n" this_id label;
    conditional_link_with_parent this_id parent_id;;

let make_nonterminal ~this:this_id ?parent:(parent_id=(-1)) label =
    printf "%d [label=\"%s\" shape=plaintext]\n" this_id label;
    conditional_link_with_parent this_id parent_id;;

let folded_printer func parent_id =
    (fun id x -> let this_id = id + 1 in
        func x this_id parent_id );;

let print_plain str id parent =
    make_nonterminal str ~this:id ~parent:parent;
    id + 1

let size_of_bind_type = function
    | Int_1_byte -> 1
    | Int_2_bytes -> 2
    | Int_4_bytes -> 4
    | UInt_1_byte -> 1
    | UInt_2_bytes -> 2
    | UInt_4_bytes -> 4

let string_of_bind_type = function
    | Int_1_byte -> "int1"
    | Int_2_bytes -> "int2"
    | Int_4_bytes -> "int4"
    | UInt_1_byte -> "uint1"
    | UInt_2_bytes -> "uint2"
    | UInt_4_bytes -> "uint4"

let is_signed_type = function
    | Int_1_byte -> true
    | Int_2_bytes -> true
    | Int_4_bytes -> true
    | UInt_1_byte -> false
```

```
48      | UInt_2_bytes -> false
49      | UInt_4_bytes -> false
50
51  let string_of_operator = function
52      | Add -> "Add"
53      | Subtract -> "Sub"
54      | Multiply -> "Mul"
55      | Divide -> "Div"
56      | Equal -> "Eq"
57      | Neq -> "Neq"
58      | Less -> "Lt"
59      | Leq -> "Leq"
60      | Greater -> "Gt"
61      | Geq -> "Geq"
62
63  (* each *_print function returns the next id available for use *)
64  let print_tree prog =
65      let rec stmt_print root id parent =
66          match root with
67          | Pattern(pat_expr, stmt) ->
68              let pattern_id = id + 1 in
69              let stmt_id = pat_expr_print pat_expr pattern_id id in
70              let consumed_ids = stmt_print stmt (stmt_id + 1) stmt_id in
71              make_nonterminal "pattern" ~this:id ~parent:parent;
72              make_nonterminal "statement" ~this:stmt_id ~parent:id;
73              consumed_ids
74
75          | Block(statement_lst) ->
76              make_nonterminal "block" ~this:id ~parent:parent;
77              List.fold_left (folded_printer stmt_print id) id statement_lst
78
79          | Expr(expr) ->
80              make_nonterminal "expression" ~this:id ~parent:parent;
81              expr_print expr (id + 1) id
82
83          | FunctionDecl(decl) ->
84              let stmt_id = stmt_print decl.body (id + 1) id in
85              let consumed_ids = List.fold_left
86                      (folded_printer print_plain stmt_id) (stmt_id + 1)
87                      decl.arguments
88              in
89              make_nonterminal "arguments" ~this:stmt_id ~parent:id;
90              make_nonterminal (sprintf "fdef:%s" decl.fname) ~this:id
91                      ~parent:parent;
92              consumed_ids
93          | If(cond, stmt1, stmt2) ->
94              let if_id = expr_print cond (id + 1) id in
95              let ifstmt = stmt_print stmt1 if_id id in
96              let consumed_ids = stmt_print stmt2 ifstmt id in
97              make_nonterminal "if" ~this:id ~parent:parent;
98              consumed_ids
99
100         (* | _ ->
101             make_nonterminal "other_stmt" ~this:id ~parent:parent;
102             id + 1 *)
103
```

```
104     and expr_print expr id parent =
105         match expr with
106         | LitInt(value) ->
107             make_terminal (string_of_int value) ~this:id ~parent:parent;
108             id + 1
109         | Binopt(e1, op, e2) ->
110             let e2_id = expr_print e1 (id + 1) id in
111             let consumed_ids = expr_print e2 e2_id id in
112             make_nonterminal (string_of_operator op) ~this:id ~parent:parent;
113             consumed_ids
114         | Call(name, arg_list) ->
115             let name_id = id + 1 in
116             let arg_id = name_id + 1 in
117             make_nonterminal "call" ~this:id ~parent:parent;
118             make_terminal name ~this:name_id ~parent:id;
119             make_nonterminal "arguments" ~this:arg_id ~parent:id;
120             List.fold_left (folded_printer expr_print arg_id) arg_id arg_list
121         | LitString(str) ->
122             make_terminal str ~this:id ~parent:parent;
123             id + 1;
124         | Assign(name, expr) ->
125             make_nonterminal "assign" ~this:id ~parent:parent;
126             make_nonterminal name ~this:(id + 1) ~parent:id;
127             expr_print expr (id + 2) id
128         | _ ->
129             make_nonterminal "other_expr" ~this:id ~parent:parent;
130             id + 1
131
132     and pat_expr_print lst id parent =
133         make_nonterminal "pat_expr" ~this:id ~parent:parent;
134         List.fold_left (folded_printer pat_token_print id) id lst
135
136     and pat_token_print token id parent =
137         match token with
138         | Binding(literal, bind_type) ->
139             let literal_id = id + 1 in
140             let type_id = id + 2 in
141             make_nonterminal "binding" ~this:id ~parent:parent;
142             make_terminal literal ~this:literal_id ~parent:id;
143             make_terminal (string_of_bind_type bind_type)
144                     ~this:type_id ~parent:id;
145             id + 3
146         | Literal(literal) ->
147             make_nonterminal "literal" ~this:id ~parent:parent;
148             make_terminal literal ~this:(id + 1) ~parent:id;
149             id + 2
150         | PatternBytes(values) ->
151             make_nonterminal "bytes" ~this:id ~parent:parent;
152             List.fold_left (folded_printer pat_token_print id) id values
153         | PatternByte(value) ->
154             make_terminal (string_of_int value) ~this:id ~parent:parent;
155             id + 1
156         | PatString(str) ->
157             make_terminal (sprintf "'%s'" str) ~this:id ~parent:parent;
158             id + 1
159         (*| _ ->
```

```
160             make_nonterminal "foo" ~this:(id+1) ~parent:id;
161             id + 2*)
162     in
163     print_string "digraph AST {\n";
164     print_string "ordering = out;\n";
165     ignore (stmt_print prog 0 0);
166     print_string "}";;
```

Listing 8.14: compile.ml

```
1  open Ast_types
2  open Bytecode_types
3
4  module StringMap = Map.Make(String)
5  exception Compile_error of string;;
6
7  type pattern_binding = {
8      loc: int;
9      size: int;
10 }
11
12 type env = {
13     symbol_map: int StringMap.t;
14     bindings: pattern_binding StringMap.t;
15     parent: env ref option;
16 }
17
18 let built_in_functions = ["print"; "RP"];;
19 let special_vars = ["LE"];;
20
21 let clean_environment =
22     let preset_binding_list =
23             (Utile.enumerate ~step:(fun x y -> y - 1) ~start:(-1)
24                 built_in_functions)
25             @ Utile.enumerate special_vars in
26     let built_ins =
27         List.fold_left (fun map item ->
28             let (value, key) = item in
29             StringMap.add key (value) map
30         ) StringMap.empty preset_binding_list in
31     {
32         symbol_map = built_ins;
33         bindings = StringMap.empty;
34         parent = None;
35     }
36
37 let new_environment parent_env =
38     ref {
39         symbol_map = StringMap.empty;
40         bindings = StringMap.empty;
41         parent = Some parent_env;
42     };;
43
44 let rec resolve_symbol env name =
45     try StringMap.find name !env.symbol_map
46     with Not_found ->
```

```
47          match !env.parent with
48            None -> let error_msg = Printf.sprintf "No such symbol: %s" name in
49              raise (Compile_error error_msg)
50          | Some(env) -> resolve_symbol env name;;
51
52  let rec resolve_binding env name =
53      try StringMap.find name !env.bindings
54      with Not_found ->
55          match !env.parent with
56            None -> let error_msg = Printf.sprintf "No such symbol: %s" name in
57              raise (Compile_error error_msg)
58          | Some(env) -> resolve_binding env name;;
59
60  let add_function env fname addr =
61      let symbol_map_new = StringMap.add fname addr env.symbol_map in
62      {env with symbol_map = symbol_map_new}
63
64  let global_counter = ref (List.length special_vars);;
65  let get_next_global () =
66      global_counter := !global_counter + 1;
67      !global_counter;;
68
69  let add_variable_force env vname id =
70      let symbol_map_new = StringMap.add vname id
71              env.symbol_map in
72      {env with symbol_map = symbol_map_new}
73
74  let add_variable env vname =
75      let symbol_map_new = StringMap.add vname (get_next_global ())
76              env.symbol_map in
77      {env with symbol_map = symbol_map_new}
78
79  let rec get_var_address env vname =
80      try StringMap.find vname !env.symbol_map;
81      with Not_found ->
82          match !env.parent with
83                None -> env := add_variable !env vname;
84                  StringMap.find vname !env.symbol_map
85              | Some(env) -> get_var_address env vname;;
86
87  let add_binding env bname size =
88      let vaddr = get_var_address env bname in
89      (* let symbol_map_new = StringMap.add bname vaddr env.symbol_map in *)
90      let bindings_new = StringMap.add bname { loc = vaddr; size = size}
91              !env.bindings in
92      {!env with bindings = bindings_new}
93
94  exception Bad_binding of string;;
95
96  let get_binding_address env bname size =
97      let bind_info =
98      try StringMap.find bname !env.bindings;
99      with Not_found ->
100         env := add_binding env bname size;
101         StringMap.find bname !env.bindings
102     in
```

```
103     if bind_info.size != size then
104         raise (Bad_binding "size mismatch")
105     else bind_info.loc;;
106 
107 let label_counter = ref 0;;
108 let get_new_label () =
109     label_counter := !label_counter + 1;
110     !label_counter;;
111 
112 let form_label_map instructions =
113     let arr = Array.make (!label_counter + 1) 0 in
114     let enumerated = Bytecode.enumerate_instructions instructions in
115     List.iter (fun x ->
116         let (addr, instr) = x in
117         match instr with
118               Label (id) -> arr.(id) <- addr; ()
119             | _ -> ()
120         ) enumerated;
121     arr;;
122 
123 let resolve_labels instructions =
124     let label_map = form_label_map instructions in
125     let rec emit_resolution = function
126           [] -> []
127         (* rewrite branches *)
128         | Bne(id)::t -> Bne label_map.(id) :: emit_resolution t
129         | Bra(id)::t -> Bra label_map.(id) :: emit_resolution t
130         | Beq(id)::t -> Beq label_map.(id) :: emit_resolution t
131         | Beo(id)::t -> Beo label_map.(id) :: emit_resolution t
132         | Jsr(id)::t when id >= 0 -> Jsr label_map.(id) :: emit_resolution t
133 
134         (* remove the pseudo instructions *)
135         | Label(id)::t -> emit_resolution t
136 
137         (* all other instructions *)
138         | h::t -> h :: emit_resolution t
139     in emit_resolution instructions;;
140 
141 let rec translate_expr env expr =
142     let recurse = translate_expr env in
143     match expr with
144       LitInt(integer) ->
145         [Bytecode_types.Lit(integer)]
146     | ExprLiteral(var_name) ->
147         let vaddr = resolve_symbol env var_name in
148         [ Lod vaddr ]
149     | Binopt(e1, op, e2) ->
150         recurse e1 @ recurse e2 @ [Bin op]
151     | Call(func_name, args) ->
152         let function_addr = resolve_symbol env func_name in
153         (List.concat (List.map recurse args)) @ [Jsr function_addr]
154     | Assign(var_name, expr) ->
155         let vaddr = get_var_address env var_name in
156         translate_expr env expr @ [
157             Str vaddr
158         ]
```

```
159     | LitString(str) ->
160         let error_msg = Printf.sprintf "Invalid use of string literal: %s" str
161         in raise (Compile_error error_msg)
162         []
163
164 and translated_pattern env expr fail_label =
165     let rec check_item = function
166           PatternByte(value) -> [
167                 Rdb 1;
168                 Lit value;
169                 Bin Subtract;
170                 Bne fail_label (* branch to failed match *)
171             ]
172         | PatternBytes(bytes) ->
173             let big_endian = List.flatten (List.map check_item bytes) in
174             let little_endian = List.flatten
175                     (List.map check_item (List.rev bytes)) in
176             let le_label = get_new_label () in
177             let end_label = get_new_label () in
178             [
179                 Lod 0; (* Load LE *)
180                 Bne le_label;
181             ]
182             @ big_endian
183             @ [ Bra end_label; Label le_label; ]
184             @ little_endian
185             @ [ Label end_label; ]
186         | Binding(literal, bind_type) ->
187             let num_bytes = Ast.size_of_bind_type bind_type in
188             let vaddr = get_binding_address env literal num_bytes in
189             [
190                 (* TODO: handle EOF here? *)
191                 Rdb num_bytes;
192             ]
193             @ (
194                 if Ast.is_signed_type bind_type then [Two (num_bytes * 8);]
195                 else []
196             )
197             @ [ Str vaddr; ]
198         | PatString(str) ->
199             let bytes = Utile.bytes_of_string str in
200             let ast_bytes = List.map (fun x -> PatternByte x) bytes in
201             translated_pattern env ast_bytes fail_label @ [];
202         | Literal(symbol) ->
203             let register = resolve_symbol env symbol in
204             let binding = resolve_binding env symbol in
205             let size = binding.size in
206             [
207                 Lod register;
208                 Rdb size;
209                 Bin Subtract;
210                 Bne fail_label
211             ]
212     in
213     List.flatten (List.map check_item expr)
214
```

```
and translate env stmt =
    let recurse = translate env in
    match stmt with
      Block(stmt_list) ->
        List.flatten (List.map recurse stmt_list)
    | Expr(expr) ->
        translate_expr env expr
    | Pattern(pat_expr, stmt) ->
        let start_label = get_new_label () in
        let fail_label = get_new_label () in
        let end_label = get_new_label () in

        let new_env = new_environment env in

        let pattern_code = translated_pattern new_env pat_expr fail_label in
        [ Ldp; Label start_label; Ldp; Beo end_label; ] @ pattern_code @
        translate new_env stmt @
        [
            Bra end_label;
            Label fail_label;
            Skp;
            Beo end_label;
            Rdb 1; Drp;
            Bra start_label;
            Label end_label;
            Skp;
            Skp;
        ]
    | FunctionDecl(decl) ->
        let start_label = get_new_label () in
        let end_label = get_new_label () in
        let num_args = List.length decl.arguments in
        let new_env = new_environment env in
        List.iter (fun x ->
            let (id,name) = x in
            new_env := add_variable_force !new_env name id;
            ();
        ) (Utile.enumerate ~step:(fun x y -> y - 1) ~start:(-100)
                (List.rev decl.arguments));
        env := add_function !env decl.fname start_label;
        [ Bra end_label; Label start_label; Ent; ] @
        translate new_env decl.body @
        [ Rts num_args; Label end_label ]
    | If(condition, ifstmt, elsestmt) ->
        let if_label = get_new_label () in
        let end_label = get_new_label () in
        translate_expr env condition
        @ [ Bne if_label; ] @ translate env elsestmt
        @ [ Bra end_label; Label if_label; ]
        @ translate env ifstmt @ [ Label end_label; ];;

let translate_program stmt =
    let env = ref clean_environment in
    resolve_labels (translate env stmt @ [Hlt]);;
```

Listing 8.15: bytecode.ml

```
open Printf
open Bytecode_types
open Ast_types

let string_of_instruction = function
      Lit(integer) -> sprintf "Lit %d" integer
    | Bin(operator) -> "Bin " ^ Ast.string_of_operator operator
    | Two(num) -> sprintf "Two %d" num
    | Rdb(num) -> sprintf "Rdb %d" num
    | Jsr(num) -> sprintf "Jsr %d" num
    | Rts(num) -> sprintf "Rts %d" num
    | Lod(num) -> sprintf "Lod %d" num
    | Str(num) -> sprintf "Str %d" num
    | Bra(addr) -> sprintf "Bra %d" addr
    | Beq(addr) -> sprintf "Beq %d" addr
    | Bne(addr) -> sprintf "Bne %d" addr
    | Beo(addr) -> sprintf "Beo %d" addr
    | Drp -> "Drp"
    | Ldp -> "Ldp"
    | Skp -> "Skp"
    | Lfp -> "Lfp"
    | Sfp -> "Sfp"
    | Ent -> "Ent"
    | Hlt -> "Hlt"

    | Label(id) -> sprintf "Label %d (PSEUDO)" id

let is_pseudo = function
      Label(id) -> true
    | _ -> false

let enumerate_instructions lst =
    Utile.enumerate ~step:(fun x y ->
        if is_pseudo x then y
        else y + 1) lst;;

let print_bytecode instructions =
    List.iter (fun x -> let (i, ins) = x in
            printf "%9d: %s\n" i (string_of_instruction ins))
        (enumerate_instructions instructions);;

let execute_instructions instructions on_file =
    let stack = Array.make 1024 0 in
    let globals = Array.make 1024 0 in
    let rec exec fp sp pc =
        (* Printf.printf "(fp=%d sp=%d pc=%d) " fp sp pc;
                Array.iter (fun d -> Printf.printf "%d " d) stack;
        Printf.printf "\n"; *)
        match instructions.(pc) with
              Lit i -> stack.(sp) <- i;
                exec fp (sp + 1) (pc + 1)
            | Bin op ->
                let op1 = stack.(sp - 2) and op2 = stack.(sp - 1) in
                stack.(sp - 2) <- (
                    let boolean i = if i then 1 else 0 in
```

```
56                      match op with
57                            Add -> op1 + op2
58                          | Subtract -> op1 - op2
59                          | Multiply -> op1 * op2
60                          | Divide -> op1 / op2
61                          | Equal -> boolean (op1 == op2)
62                          | Neq -> boolean (op1 != op2)
63                          | Less -> boolean (op1 < op2)
64                          | Leq -> boolean (op1 <= op2)
65                          | Greater -> boolean (op1 > op2)
66                          | Geq -> boolean (op1 >= op2)
67                  );
68                  exec fp (sp - 1) (pc + 1)
69              | Two size ->
70                  let operand = stack.(sp - 1) in
71                  stack.(sp - 1) <- (
72                      Utile.signed_of_unsigned operand size
73                  );
74                  exec fp sp (pc + 1)
75              | Jsr (-1) -> (* print function *)
76                  print_endline (string_of_int stack.(sp - 1));
77                  exec fp (sp - 1) (pc + 1)
78              | Rdb (1) ->
79                  stack.(sp) <- Reader.read_byte on_file;
80                  (* Printf.printf "Read byte: %x\n" stack.(sp); *)
81                  exec fp (sp + 1) (pc + 1)
82              | Rdb (n) ->
83                  stack.(sp) <- Reader.read_unsigned on_file n;
84                  exec fp (sp + 1) (pc + 1)
85              | Ldp ->
86                  stack.(sp) <- Reader.get_pos on_file;
87                  (* Printf.printf "Store position: %d at %d\n" stack.(sp) sp; *)
88                  exec fp (sp + 1) (pc + 1)
89              | Skp ->
90                  Reader.set_pos on_file stack.(sp - 1);
91                  (* Printf.printf "Set position: %d\n" stack.(sp - 1); *)
92                  exec fp (sp - 1) (pc + 1)
93              | Bne addr -> if stack.(sp - 1) == 0 then
94                  exec fp (sp - 1) (pc + 1) else
95                  exec fp (sp - 1) addr
96              | Beq addr -> if stack.(sp - 1) != 0 then
97                  exec fp (sp - 1) (pc + 1) else
98                  exec fp (sp - 1) addr
99              | Bra addr -> exec fp sp addr
100             | Beo addr ->
101                 if Reader.is_eof on_file then exec fp sp addr
102                 else exec fp sp (pc + 1)
103             | Jsr addr ->
104                 stack.(sp) <- pc + 1;
105                 exec fp (sp + 1) addr
106             | Rts(num) ->
107                 (* Printf.printf "\nReturn to %d from %d (fp=%d, num=%d, sp=%d)\n"
                        stack.(fp - 1) (fp - 1) stack.(sp - 1) num sp; *)
108                 exec stack.(fp) (fp - num) stack.(fp - 1)
109             | Lod(-2) -> (* RP variable *)
110                 stack.(sp) <- Reader.get_pos on_file;
```

```
111                 exec fp (sp + 1) (pc + 1)
112             | Lod index ->
113                 (
114                     if (index > 0) then stack.(sp) <- globals.(index)
115                     else stack.(sp) <- stack.(fp + index + 100 - 2)
116                 );
117                 exec fp (sp + 1) (pc + 1)
118
119             | Str(-2) -> (* RP variable *)
120                 Reader.set_pos on_file stack.(sp);
121                 exec fp (sp - 1) (pc + 1)
122             | Str index ->
123                 (
124                     if (index > 0) then globals.(index) <- stack.(sp - 1)
125                     else stack.(fp + index + 100 - 2) <- stack.(sp - 1)
126                 );
127                 exec fp (sp - 1) (pc + 1)
128             | Ent ->
129                 stack.(sp) <- fp;
130                 exec sp (sp + 1) (pc + 1)
131             | Lfp ->
132                 exec stack.(sp) (sp - 1) (pc + 1)
133             | Drp ->
134                 exec fp (sp - 1) (pc + 1)
135             | Hlt -> ()
136     in exec 0 0 0
137
138 let execute_bytecode instructions on_file =
139     execute_instructions (Array.of_list instructions) on_file;;
```

Listing 8.16: reader.ml

```
1
2  let read_byte ic =
3      let byte = input_char ic in
4      int_of_char byte;;
5
6  let read_bytes ic n =
7      let rec read c lst =
8          if c == 0 then List.rev lst
9          else read (c - 1) (read_byte ic :: lst)
10     in read n [];;
11
12 let read_string_null ic =
13     let rec scan ic str =
14         let c = input_char ic in
15         match c with
16             | '\000' -> Utile.implode (List.rev str)
17             | _ -> scan ic (c :: str)
18     in scan ic [];;
19
20 let read_string_fixed ic n =
21     let rec scan ic str n =
22         if n == 0 then Utile.implode (List.rev str)
23         else
24             let c = input_char ic in
```

```
            scan ic (c :: str) (n - 1)
    in scan ic [] n;;

let read_unsigned ic n =
    let bytes = read_bytes ic n in
    let rbytes = List.rev bytes in
    let rec form_int n shift bytes =
        match bytes with
            | [] -> n
            | h :: t ->
                let next_n = n + (h lsl shift) in
                form_int next_n (shift + 8) t
    in form_int 0 0 rbytes;;

let advance ic n =
    let pos = pos_in ic in
    let new_pos = pos + n in
    seek_in ic new_pos;;

let get_pos ic =
    pos_in ic;;

let set_pos ic n =
    seek_in ic n;;

let is_eof ic =
    let pos = get_pos ic in
    let result = try input_char ic; false with End_of_file -> true in
    set_pos ic pos; result;;
```

Listing 8.17: utile.ml

```


let int_of_hex str =
    int_of_string ("0x" ^ str);;

let enumerate ?step:(step=(fun x y -> y + 1)) ?start:(start=0) lst =
    let rec enum count = function
          [] -> []
        | h :: t -> (count, h) :: (enum (step h count) t)
    in enum start lst;;

let explode s =
    let rec exp i l =
        if i < 0 then l else exp (i - 1) (s.[i] :: l) in
    exp (String.length s - 1) [];;

let implode l =
    let res = String.create (List.length l) in
    let rec imp i = function
    | [] -> res
    | c :: l -> res.[i] <- c; imp (i + 1) l in
    imp 0 l;;

let bytes_of_string str =
```

```
25      let chars = explode str in
26      List.map int_of_char chars;;
27
28  let signed_of_unsigned value num_bits =
29      let shift = num_bits - 1 in
30      let mask = 1 lsl shift in
31      if mask land value > 0 then (value land (lnot mask)) - mask
32      else value;;
```

8.2 Test files

All test files are designed to be run on the input file `lichtenstein.png` unless specified otherwise.

8.2.1 test-arith.bawk

Listing 8.18: Test program (test-arith.bawk)

```
1 print(0 + 10 - 9 + 3);
```

Listing 8.19: Expected output (test-arith.bawk.out)

```
1 4
```

8.2.2 test-cond1.bawk

Listing 8.20: Test program (test-cond1.bawk)

```
1
2 if (100 > 100*2) {
3     print(1);
4 } else {
5     print(0);
6 }
7
8
9 if (100) {
10    print (100);
11 }
12
13 if (0) {
14    print (0);
15 }
```

Listing 8.21: Expected output (test-cond1.bawk.out)

```
1 0
2 100
```

8.2.3 test-func1.bawk

Listing 8.22: Test program (test-func1.bawk)

```
1
2 def food () {
3     print(100);
4 }
5
6 print(1);
7 food();
8 print(2);
9 food();
10 print(3);
```

Listing 8.23: Expected output (test-func1.bawk.out)

```
1 1
2 100
3 2
4 100
5 3
```

8.2.4 test-func2.bawk

Listing 8.24: Test program (test-func2.bawk)

```
1 print(1);
2
3 def food () {
4     print(100);
5 }
6
7 print(2);
```

Listing 8.25: Expected output (test-func2.bawk.out)

```
1 1
2 2
```

8.2.5 test-func3.bawk

Listing 8.26: Test program (test-func3.bawk)

```
1
2 a = 100;
3
4 def bar(c) {
5     print(c);
6 }
7
8 def foo(a, b, c) {
```

```
 9      print (a);
10      print (b);
11      print (c);
12      a = 10;
13      bar(c * 2);
14      print(a);
15      print(c);
16      c = c * 2;
17      print(c);
18  }
19
20  foo(50+20, 10, a);
```

Listing 8.27: Expected output (test-func3.bawk.out)

```
1  70
2  10
3  100
4  200
5  10
6  100
7  200
```

8.2.6 test-pat1.bawk

Listing 8.28: Test program (test-pat1.bawk)

```
1  /89504e47/ { /* should match */
2      print(10);
3  }
4
5  /89504347/ { /* should not match */
6      print(20);
7  }
8
9  print(30);
```

Listing 8.29: Expected output (test-pat1.bawk.out)

```
1  10
2  30
```

8.2.7 test-pat2.bawk

Listing 8.30: Test program (test-pat2.bawk)

```
1  /89504e47/ { /* should match */
2      print(10);
3  }
4
5  /89504e47/ { /* should match */
6      print(20);
7  }
```

```
8
9 print(30);
```

Listing 8.31: Expected output (test-pat2.bawk.out)

```
1 10
2 20
3 30
```

8.2.8 test-pat3.bawk

Listing 8.32: Test program (test-pat3.bawk)

```
1 /8950/ { /* should match */
2     print(10);
3 }
4
5 /8951123456789/ { /* should not match */
6     print(20);
7 }
8
9 /8950/ { /* should match */
10    print(40);
11 }
12
13 print(30);
```

Listing 8.33: Expected output (test-pat3.bawk.out)

```
1 10
2 40
3 30
```

8.2.9 test-pat4.bawk

Listing 8.34: Test program (test-pat4.bawk)

```
1 a = 100;
2
3 /89 foo:int1 4e47/ { /* should match */
4     print(foo);
5 }
6
7 /89504e47/ { /* should match */
8     print(20);
9 }
10
11
12 print(30);
```

Listing 8.35: Expected output (test-pat4.bawk.out)

```
80
20
30
```

8.2.10 test-pat5.bawk

Listing 8.36: Test program (test-pat5.bawk)

```

/89 foo2:int4/ { /* should match */
    print(foo2); /* foo = 0x504e470d */
}
```

Listing 8.37: Expected output (test-pat5.bawk.out)

```
1347307277
```

8.2.11 test-pat6.bawk

Listing 8.38: Test program (test-pat6.bawk)

```

/* these values should be different */

/000a value:int4/  { print(value); }
/000a value:uint4/ { print(value); }

/000a value:int2/ { print (value); }
/000a value:uint2/ { print (value); }

/000a value:int1/ { print (value); }
/000a value:uint1/ { print (value); }


/* these values should be the same */

/000d value:int4/  { print(value); }
/000d value:uint4/ { print(value); }

/000d value:int2/  { print(value); }
/000d value:uint2/ { print(value); }

/000d value:int1/  { print(value); }
/000d value:uint1/ { print(value); }
```

Listing 8.39: Expected output (test-pat6.bawk.out)

```
-268435446
4026531850
-4096
61440
```

```
-16
240
1229472850
1229472850
18760
18760
73
73
```

8.2.12 test-pat7.bawk

Listing 8.40: Test program (test-pat7.bawk)

```
1
2 my_RP = RP;
3 print(my_RP);
4
5 / foo:int1 19 38af / {
6     print(foo);
7     my_RP = RP;
8     print(my_RP);
9 }
10
11 print(my_RP);
12 RP = my_RP;
13
14 / foo:int1 00 / {
15     print(foo);
16 }
```

Listing 8.41: Expected output (test-pat7.bawk.out)

```
1 0
2 9
3 72
4 72
5 117
```

8.2.13 test-pat8.bawk

Listing 8.42: Test program (test-pat8.bawk)

```
1
2 /4e47 value:int1/ {
3     print(value);
4     print(RP);
5     /value value2:int2/ {
6         print(value2);
7         print(RP);
8     }
9 }
```

Listing 8.43: Expected output (test-pat8.bawk.out)

```
13
5
18760
14
```

8.2.14 test-pat9.bawk

Listing 8.44: Test program (test-pat9.bawk)

```
/* LE defaults to false (0) */

/ 091938af / {
    print(100);
    print(RP);
}

LE = 1;

/ af381909 / {
    print(200);
    print(RP);
}

/ 09 19 38 af / {
    print(300);
    print(RP);
}

/ af 38 19 09 / {
    print(401);
    print(RP);
}

LE = 0;

/ 091938af / {
    print(500);
    print(RP);
}


/ 09 19 38 af / {
    print(600);
    print(RP);
}
```

Listing 8.45: Expected output (test-pat9.bawk.out)

```
100
72
200
72
300
```

```
72
500
72
600
72
```

8.2.15 test-pat10.bawk

Listing 8.46: Test program (test-pat10.bawk)

```
LE = 0;
/ "IHDR" / {
    print(100);
    print(RP);
}

LE = 1;
/ "IHDR" / {
    print(200);
    print(RP);
}
```

Listing 8.47: Expected output (test-pat10.bawk.out)

```
100
16
200
16
```

8.2.16 test-pat11.bawk

Listing 8.48: Test program (test-pat11.bawk)

```

/"IEND" end:uint2/ {
    print(end);
}

/"IEND" end:int2/ {
    print(end);
}
```

Listing 8.49: Expected output (test-pat11.bawk.out)

```
44610
-20926
```

8.2.17 test-scope1.bawk

Listing 8.50: Test program (test-scope1.bawk)

```
1
2 a = 1000;
3 b = 200;
4
5 def foo() {
6     a = 10 + 100;
7     print(a);
8     print(b);
9 }
10
11 print(a);
12 foo();
13 print(a);
14 print(b);
```

Listing 8.51: Expected output (test-scope1.bawk.out)

```
1 1000
2 110
3 200
4 110
5 200
```

8.2.18 test-scope2.bawk

Listing 8.52: Test program (test-scope2.bawk)

```
1
2 b = 200;
3
4 def foo() {
5     a = 10 + 100;
6     print(a);
7     print(b);
8 }
9
10 a = 1000;
11
12 print(a);
13 foo();
14 print(a);
15 print(b);
```

Listing 8.53: Expected output (test-scope2.bawk.out)

```
1 1000
2 110
3 200
4 110
5 200
```

8.2.19 test-useful1.bawk

Listing 8.54: Test program (test-useful1.bawk)

```
/* print the size of a PNG file */

/89 "PNG" 0d 0a 1a 0a/ {
   /length:uint4 "IHDR"/ {
      /width:uint4 height:uint4/ {
        print(width);
        print(height);
      }
   }
}
```

Listing 8.55: Expected output (test-useful1.bawk.out)

```
512
512
```

8.2.20 test-var1.bawk

Listing 8.56: Test program (test-var1.bawk)

```

a = 10 + 20 * 20;
b = 20 * 20;
print(a);
print(b);
```

Listing 8.57: Expected output (test-var1.bawk.out)

```
410
400
```

8.3 Misc. Files

Listing 8.58: Makefile

```
DEP_FILENAME=.ocamldeps.mk

OBJS = utile.cmo \
    reader.cmo \
    scanner.cmo \
    parser_help.cmo \
    parser.cmo \
    ast.cmo \
    compile.cmo \
    bytecode.cmo \
    bawk.cmo

default: bawk plt_docs/lrm.pdf plt_docs/proposal.pdf design_docs
```

```
14
15 plt_docs/report.pdf: $(wildcard plt_docs/report-*.tex) plt_docs/gitlog.tex
16
17 plt_docs/gitlog.tex:
18     git log --stat > plt_docs/gitlog.tex
19
20 %.pdf: %.tex
21     cd $(shell dirname $@); pdflatex $(shell basename $<)
22     cd $(shell dirname $@); pdflatex $(shell basename $<)
23
24 bawk: $(OBJS)
25     ocamlc -o bawk $(shell ocamldep -sort *.ml | sed 's/\.ml/\.cmo/g')
26
27 scanner.ml: scanner.mll
28     ocamllex scanner.mll
29
30 parser.ml parser.mli: parser.mly
31     ocamlyacc parser.mly
32
33 %.cmo: %.ml | $(DEP_FILENAME)
34     ocamlc -c $<
35
36 %.cmi: %.mli | $(DEP_FILENAME)
37     ocamlc -c $<
38
39 clean:
40     rm -f bawk *.cmi *.cmo scanner.ml parser.ml parser.mli
41     rm -f design_docs/*.html design_docs/*.css
42     rm -f plt_docs/*.pdf plt_docs/*.toc plt_docs/*.aux plt_docs/*.log plt_docs/*.
          lol
43     rm -f $(DEP_FILENAME)
44     rm -f plt_docs/gitlog.tex
45
46 design_docs: bawk
47     $(MAKE) -C ./design_docs
48
49 include $(DEP_FILENAME)
50
51 $(DEP_FILENAME): parser.ml parser.mli scanner.ml
52     ocamldep *.ml *.mli > $(DEP_FILENAME)
53
54 .PHONY: clean default design_docs
```

Listing 8.59: run_tests.sh

```
1  #!/bin/sh
2
3  BAWK="./bawk"
4
5  # Set time limit for all operations
6  ulimit -t 30
7
8  globallog=testall.log
9  rm -f $globallog
10 error=0
11 globalerror=0
```

```
12
13 keep=0
14
15 Usage() {
16     echo "Usage: testall.sh [options] [.mc files]"
17     echo "-k    Keep intermediate files"
18     echo "-h    Print this help"
19     exit 1
20 }
21
22 SignalError() {
23     if [ $error -eq 0 ] ; then
24     echo "FAILED"
25     error=1
26     fi
27     echo "  $1"
28 }
29
30 # Compare <outfile> <reffile> <difffile>
31 # Compares the outfile with reffile.  Differences, if any, written to difffile
32 Compare() {
33     generatedfiles="$generatedfiles $3"
34     echo diff -b $1 $2 ">" $3 1>&2
35     diff -b "$1" "$2" > "$3" 2>&1 || {
36     SignalError "$1 differs"
37     echo "FAILED $1 differs from $2" 1>&2
38     }
39 }
40
41 # Run <args>
42 # Report the command, run it, and report any errors
43 Run() {
44     echo $* 1>&2
45     eval $* || {
46     SignalError "$1 failed on $*"
47     return 1
48     }
49 }
50
51 Check() {
52     error=0
53     basename=`echo $1 | sed 's/.*\\///
54                               s/.mc//'`
55     reffile=`echo $1 | sed 's/.mc$//'`
56     basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."
57
58     echo -n "$basename..."
59
60     echo 1>&2
61     echo "###### Testing $basename" 1>&2
62
63     generatedfiles=""
64
65     generatedfiles="$generatedfiles ${basename}.i.out" &&
66     Run "$BAWK" "-e tests/lichtenstein.png" "<" $1 ">" ${basename}.i.out &&
67     Compare ${basename}.i.out ${reffile}.out ${basename}.i.diff
```

```
68
69      #generatedfiles="$generatedfiles ${basename}.c.out" &&
70      #Run "$BAWK" "-c" "<" $1 ">" ${basename}.c.out &&
71      #Compare ${basename}.c.out ${reffile}.out ${basename}.c.diff
72
73      # Report the status and clean up the generated files
74
75      if [ $error -eq 0 ] ; then
76      if [ $keep -eq 0 ] ; then
77          rm -f $generatedfiles
78      fi
79      echo "OK"
80      echo "###### SUCCESS" 1>&2
81      else
82      echo "###### FAILED" 1>&2
83      globalerror=$error
84      fi
85  }
86
87  while getopts kdpsh c; do
88      case $c in
89      k) # Keep intermediate files
90          keep=1
91          ;;
92      h) # Help
93          Usage
94          ;;
95      esac
96  done
97
98  shift `expr $OPTIND - 1`
99
100 if [ $# -ge 1 ]
101 then
102     files=$@
103 else
104     files="tests/fail-*.bawk tests/test-*.bawk"
105 fi
106
107 for file in $files
108 do
109     case $file in
110     *test-*)
111         Check $file 2>> $globallog
112         ;;
113     *fail-*)
114         CheckFail $file 2>> $globallog
115         ;;
116     *)
117         echo "unknown file type $file"
118         globalerror=1
119         ;;
120     esac
121 done
122
123 exit $globalerror
```