The Extended-Kaleidoscope Language Specification

Version 0.2 - 2018-01-12
Hal Finkel and Kavon Farvardin

General Notes

- This language is an extended version of the Kaleidoscope language from LLVM's tutorial: https://llvm.org/docs/tutorial/

Grammar

- Whitespace (space, newline, etc) is allowed between any two tokens in the grammar below.
- Comments begin with the “#” character and continue until the end of the line.

```
<prog>    ::= <extern>* <func>+

<extern>  ::=  extern <type> <globid> "(" <tdecls>? ")" ";"

<func>    ::= def <type> <globid> "(" <vdecls>? ")" <blk>
<blk>     ::= "{" <stmts>? "}"
<stmts>   ::= <stmt>+

<stmt>    ::= <blk>
            | return <exp>? ";"
            | <vdecl> "=" <exp> ";"
            | <exp> ";"
            | while "(" <exp> ")" <stmt>
            | if "(" <exp> ")" <stmt> (else <stmt>)?
            | print <exp> ";"
            | print <slit> ";"

<exps>    ::= <exp> | <exp> "," <exps>
<exp>     ::= "(" <exp> ")"
            | <binop>
            | <uop>
            | <lit>
            | <var>
            | <globid> "(" <exps>? ")"

<binop>   ::= <exp> * <exp>
            | <exp> / <exp>
            | <exp> + <exp>
            | <exp> - <exp>
            | <var> = <exp>     # assignment
            | <exp> == <exp>    # equality
            | <exp> < <exp>
            | <exp> > <exp>
            | <exp> && <exp>    # logical or
            | <exp> || <exp>    # logical and

<uop>     ::= ! <exp>          # logical negation
            | - <exp>          # signed negation

<lit>     ::= [0-9]+(\.[0-9]+)?
<slit>    ::= "[^"]*"

<ident>   ::= [a-zA-Z_]+[a-zA-Z0-9_]*
<var>     ::= "$" <ident>
<globid>  ::= <ident>
<type>    ::= int | cint | float | sfloat | void | (noalias)? ref <type>
<vdecls>  ::= <vdecl> | <vdecl> "," <vdecls>
<tdecls>  ::= <type> | <type> "," <tdecls>
<vdecl>   ::= <type> <var>
```

Typing Rules and Constraints *(Informally)*

1. In `<vdecl>, the type may not be void.`
2. In `ref <type>`, the type may not be void or itself a reference type.
3. All functions must be declared and/or defined before they are used.
4. A function may not return a ref type.
5. `print` prints to stdout followed by a new line.
6. Values of reference type are bound to their initialization's right-hand-side expression (or, for function arguments, the provided function parameter), which must be a variable itself. For example:

```
int $y = 0;
int $w = 1;

ref int $x = $y;  # $x is bound to $y.
ref int $z = $x;  # $z is also bound to $y.
ref int $a = 11;  # illegal, RHS must be a var.

def void foo (ref int $f, int $g) { ... }

foo ($w, $y)  # $w is passed by reference, and $y is passed by value.
```

7. Uses of the reference variable evaluate to the then-current value of the bound variable. Assignments to the reference variable set the value of the bound variable.
8. If the types of a binary operator don't match, apply the following rules in order:
   a. If either type is float, convert the other value to float and the result is float.
   b. If either type is sfloat, convert the other value to sfloat and the result is sfloat.
   c. If either type is int, convert the other value to int and the result is int.
   d. If either type is cint, convert the other value to cint and the result is cint.

Operational Semantics *(Informally)*

- Implicit Booleans: The arguments to `if` and `while,` !, &&, || are treated as Boolean values implicitly. The value is considered false if it is equal to zero (either integer or floating point) or NaN. Otherwise, the value is true.
- All comparison operators (==, <, >) produce an integer value: 0 for false and 1 for true. For comparisons of two sfloat values, the comparison is ordered (i.e., the result is false if either operand is NaN).
- The integer types are signed and:
  a. For int, the behavior of the program is undefined if the value overflows.
  b. For cint, if the value overflows the program must print an error message to stderr and exit (the exit status of the program must indicate failure).
- Arguments to a function are evaluated left-to-right. The value of an assignment is its left-hand side.
- All programs must define exactly one function named "run" which returns an integer (the program exit status) and takes no arguments. This is the program entry point.
- If a reference variable, r, is declared noalias, then the programmer is promising that, within the scope of the reference variable, the bound variable, b, is accessed only through r or a reference derived from r. When a reference variable is bound using a reference variable on the right-hand side, both refer to the same underlying variable (and this reference variable is considered to be derived from the one of the right-hand side).
- If control-flow reaches the end of the function without returning, and the function has a void return type, then a void return is implicitly assumed.

- Associativity and precedence, from highest to lowest:
    a. Right-to-left: - (unary) ! (logical not)
    b. Left-to-right: * /
    c. Left-to-right: + - (binary)
    d. Left-to-right: < >
    e. Left-to-right: ==
    f. Left-to-right: &&
    g. Left-to-right: ||
    h. Right-to-left: =

External Functions

- Get the specified command-line argument as an integer: `extern int arg(int);`
- Get the specified command-line argument as a float: `extern float argf(int);`

Examples

```
def int fib (int $n) {
    if ($n < 2)
        if ($n == 0)
            return 0;
        else
            return 1;

    int $a = fib ($n - 1);
    int $b = fib ($n - 2);
    return $a + $b;
}

def void inc (ref int $n) {
  $n = $n + 1;
}

def int run () {
    print "fib(5):";
    int $val = fib(5);
    print $val;

    print "fib(5)+1:";
    inc($val);
    print $val;

    return 0;

}
```