A Self-Guided Tour of NESL

NESL is a parallel language developed here at Carnegie Mellon. It is loosely based on the programming language ML. This document will help you get started with NESL, which we will use in this class.

1 Loading NESL

NESL has an interactive interface that runs on Andrew Linux machines. To start NESL, type

```
/afs/cs.cmu.edu/project/scandal/nesl/runnesl
```

at a shell prompt. You should have the following screen, ending with a NESL prompt `<Nesl>`.

```
  i i i i i i i       ooooo    o        ooooooo   ooooo   ooooo
  I I I I I I I      8     8   8           8     8     o  8    8
  I  \ '+' /  I      8         8           8     8        8    8
   \  '-+-'  /       8         8           8      ooooo   8oooo
    '-__|__-'        8         8           8           8  8
        |            8     o   8           8     o     8  8
  ------+------       ooooo    8ooooooo  ooo8ooo   ooooo   8

Copyright (c) Bruno Haible, Michael Stoll 1992, 1993
Copyright (c) Bruno Haible, Marcus Daniels 1994-1997
Copyright (c) Bruno Haible, Pierpaolo Bernardi, Sam Steingold 1998
Copyright (c) Bruno Haible, Sam Steingold 1999-2000
Copyright (c) Sam Steingold, Bruno Haible 2001-2006


% NESL version 3.1 (November 1, 1995) %

% Setting machine configuration LOCAL.... %

% Use (nesl) when an error aborts you out of the interpreter. %
% Type help; for a list of the top level commands. %

<Nesl>
```

Now that you can start NESL, an important command to learn is `exit;`. This will bring you back to the shell. You can also exit NESL with Ctrl-D.

In the remaining of the document, you will learn to operate NESL by examples. For a quick-reference guide, visit

`http://www.cs.cmu.edu/~scandal/nesl/quickref.html`

2 Scalar Operations

```
<Nesl> 14;

it = 14 : int
<Nesl> (2.2 + 1.1) / 5.0;

it = 0.66 : float
<Nesl> t or f;

it = T : bool
<Nesl> 1.6 + 7;

Error at top level.
For function + in expression
  1.6 + 7
inferred argument types don't match function specification.
  Argument types: float, int
  Function types: a, a :: (a in number)

<Nesl> 1.6 + float(7);

it = 8.6 : float
<Nesl> sin(.6);

it = 0.564642473395035 : float
<Nesl> a = 3;

a = 3 : int
<Nesl> a + 5;

it = 8 : int
<Nesl> if (4 < 5) then "joe" else "susan";

it = "joe" : [char]
<Nesl> let a = 3 * 4;
           b = 4 * 5;
       in a + b;

it = 32 : int
<Nesl> function fact(i) =
       if (i == 1) then 1
       else i * fact(i-1);

fact = fn : int -> int
<Nesl> fact(5);

it = 120 : int
<Nesl> (2, `a);

it = (2, `a) : int, char
```

```
<Nesl> function div_rem(a,b) = (a / b, rem(a,b));

div_rem = fn : (int, int) -> (int, int)
<Nesl> div_rem(10,3);

it = (3, 1) : int, int
```

3 Vector Operations

```
<Nesl> [2, 5, 1, 3];

it = [2, 5, 1, 3] : [int]
<Nesl> "this is a vector";

it = "this is a vector" : [char]
<Nesl> [(2, "joe"), (3, "peter")];

it = [(2, "joe"), (3, "peter")] : [(int, [char])]
<Nesl> [2, 3.0, 5];

Error at top level.
For function make_sequence in expression
  [2, 3.0]
inferred argument types don't match function specification.
  Argument types: [int], float
  Function types: [a], a :: (a in any)

<Nesl> {a + 1 : a in [2, 3, 4]};

it = [3, 4, 5] : [int]
<Nesl> let a = [2, 3, 4] in {a + 1 : a};

it = [3, 4, 5] : [int]
<Nesl> {a + b : a in [2,3,4]; b in [4,5,6]};

it = [6, 8, 10] : [int]
<Nesl> {fact(a) : a in [1, 2, 3, 4, 5]};

it = [1, 2, 6, 24, 120] : [int]
<Nesl> sum([2,3,4]);

it = 9 : int
<Nesl> dist(5,10);

it = [5, 5, 5, 5, 5, 5, 5, 5, 5, 5] : [int]
<Nesl> [1:10];

it = [1, 2, 3, 4, 5, 6, 7, 8, 9] : [int]
<Nesl> "big" ++ "boy";

it = "bigboy" : [char]
```

```
<Nesl> {x in "wombat" | x <= `m};

it = "mba" : [char]
<Nesl> {sum(a) : a in [[2,3,4], [1], [7,8,9]]};

it = [9, 1, 24] : [int]
<Nesl> bottop("testing");

it = ["test", "ing"] : [[char]]
<Nesl> partition("break into words", [5, 5, 6]);

it = ["break", " into", " words"] : [[char]]
<Nesl> function qsort(a) =
       if #a < 2 then a
       else let
         pivot = a[rand(#a)];
         le = {a | a < pivot};
         gr = {a | a >= pivot};
       in flatten({qsort(r) : r in [le,gr]}) ;

qsort = fn : [a] -> [a] :: (a in ordinal)
<Nesl> qsort([7, 2, 1, 9, 11, 3, 4]);

it = [1, 2, 3, 4, 7, 9, 11] : [int]
```

4 Loading Program Files

```
<Nesl> load("/afs/cs/academic/class/15492-f07/nesl/examples/median.nesl");

% Loading /afs/cs/academic/class/15492-f07/nesl/examples/median.nesl. %

<Nesl> my_median([8, 1, 2, 9, 5, 3, 4]);

it = 4 : int
<Nesl> my_median("this is a test");

it = `i : char
```

5 Getting Help

```
<Nesl> describe ++;

INTERFACE:

v1 ++ v2

TYPE:

 ([a], [a]) -> [a] :: (a in any)
```

```
DOCUMENTATION:

 Given two sequences, {\tt ++} appends them.

CODE:

  join(v1,
       iseq-l(0, 1, v1), v2, iseq-l(length(v1), 1, v2))


<Nesl> help();

NESL top-level forms:
  function <name> <pattern> [: <typedef>] = <exp>;  -- Function Definition
  datatype <name> <typedef>;                        -- Record Definition
  <pattern> = <exp>;                                -- Top level Assignment
  <exp>;                                            -- Any NESL expression

 .
 .
 .
```