A Small Example for Literate Programming

Hans-Georg Eßer (*h.g.esser@gmx.de*)
Universität Erlangen-Nürnberg

July 4, 2013

Contents

This is a first paragraph that has nothing to do with the literate program; think of a literature review, remarks about helpful colleagues and similar topics. We do not want it to appear in the slides which is why the BEGIN LITERATE TEACHING comment will follow just after this paragraph.

1 Code Overview

This function is meant to decide, for a given input n, whether n is divisible by 3 or not. So we expect the overall program to look somewhat like this:

⟨*the program*⟩≡

```
/* Test program, (c) 2013 The Author */
⟨header file inclusion⟩
⟨test function⟩

#define THIS_IS_AN_ENORMOUSLY_LONG_SYMBOLIC_CONSTANT_NAME 42

main () {
  int x;
  printf ("Enter number");     // show message
  input (x);                   // read number
  if (testdiv3(x)) {
    printf ("%d is divisible by 3.\n", x);
  } else {
    printf ("%d is not divisible by 3.\n", x);
  }
  return;
}
```

We will certainly need to include some header files because we want to use the `printf` and `input` functions:

⟨*header file inclusion*⟩≡

```
#include <stdio.h>        // for printf()
#include "../myread.h"    // for input()

// This has nothing to do with headers, but
// this way you can see a really long line
#define YET_ANOTHER_SYMBOLIC_CONSTANT_WHICH_HAS_A_LONG_NAME_ONLY_THIS_TIME_IT_IS_SO_LONG_TH

typedef struct {  /* this is also
                     something that */
  int i;  // int
  int j;  // more int
} /* does not belong here! */ useless_t;
```

We could, of course, create a test function that looks like the following one, testing for all possible numbers which are divisible by 3—well, at least up to some given maximum number. Of course, there are infinitely many of them. So, what we're not going to do is the following:

⟨*test function, bad one*⟩≡

```
int testdiv3(int x) {
  return (
    (x ==  3) ||
    (x ==  6) ||
    (x ==  9) ||
    (x == 12) ||
    (x == 15) ||
    (x == 18) ||
    (x == 21) ||
    (x == 24) ||
    (x == 27) ||
    (x == 30) ||
    (x == 33) ||
    (x == 36) ||
    (x == 39) ||
    (x == 42) ||
    (x == 45) ||
    (x == 48) ||
    (x == 51) ||
    (x == 54) ||
    (x == 57) ||
    (x == 60) ||
    (x == 63) ||
    (x == 66) ||
    (x == 69) ||
    (x == 72) ||
    (x == 75) ||
    (x == 78) ||
    (x == 81) ||
    (x == 84) ||
    (x == 87) );
}
```

Now, for some reason (and completely unrelated to the task of this paper) we want to display the output of `ls -li`—and it shall also appear on a slide. We'll try it two times, once with side comments and once just the code.

⟨*directory with comments*⟩≡

```
[esser@macbookpro:literate-teaching]$ LANG=C ls -li
total 288
15653128 drwxr-xr-x  5 esser  staff    170 Apr  1 06:48 Literatur
15656814 drwxr-xr-x  4 esser  staff    136 Apr  1 18:27 Software
15662900 -rwxr-xr-x  1 esser  staff   7698 Apr  2 02:13 combine.py
15668192 -rwxr-xr-x  1 esser  staff   4926 Apr  2 02:18 example.nw
15643259 -rwxr-xr-x  1 esser  staff   6522 Apr  2 01:22 export.py
15658881 drwxr-xr-x  4 esser  staff    136 Apr  1 17:44 fonts
15641951 -rw-r--r--  1 esser  staff    395 Mar 30 20:45 index.html
15661490 drwxr-xr-x  6 esser  staff    204 Apr  1 19:40 logos
15667494 -rwxr-xr-x  1 esser  staff    683 Apr  2 01:40 make.sh
15651837 -rw-r--r--  1 esser  staff  32418 Mar 31 22:35 nicEdit.js
15649904 -rw-r--r--  1 esser  staff   3351 Jun  7  2012 nicEditorIcons.gif
15643349 -rw-r--r--  1 esser  staff  12427 Apr  2 02:21 out.html
15660031 -rw-r--r--  1 esser  staff  17606 Apr  2 02:21 out2.html
15667034 -rw-r--r--  1 esser  staff   1141 Apr  2 02:21 result.txt
15658182 drwxr-xr-x  5 esser  staff    170 Apr  1 17:21 s5-css
15662226 -rwxr-xr-x  1 esser  staff  34312 Apr  2 00:24 sched-rr10.nw
```

As promised, one more time:

⟨*directory without comments*⟩≡

```
[esser@macbookpro:literate-teaching]$ LANG=C ls -li
total 288
15653128 drwxr-xr-x  5 esser  staff    170 Apr  1 06:48 Literatur
15656814 drwxr-xr-x  4 esser  staff    136 Apr  1 18:27 Software
15662900 -rwxr-xr-x  1 esser  staff   7698 Apr  2 02:13 combine.py
15668192 -rwxr-xr-x  1 esser  staff   4926 Apr  2 02:18 example.nw
15643259 -rwxr-xr-x  1 esser  staff   6522 Apr  2 01:22 export.py
15658881 drwxr-xr-x  4 esser  staff    136 Apr  1 17:44 fonts
15641951 -rw-r--r--  1 esser  staff    395 Mar 30 20:45 index.html
15661490 drwxr-xr-x  6 esser  staff    204 Apr  1 19:40 logos
15667494 -rwxr-xr-x  1 esser  staff    683 Apr  2 01:40 make.sh
15651837 -rw-r--r--  1 esser  staff  32418 Mar 31 22:35 nicEdit.js
15649904 -rw-r--r--  1 esser  staff   3351 Jun  7  2012 nicEditorIcons.gif
15643349 -rw-r--r--  1 esser  staff  12427 Apr  2 02:21 out.html
15660031 -rw-r--r--  1 esser  staff  17606 Apr  2 02:21 out2.html
15667034 -rw-r--r--  1 esser  staff   1141 Apr  2 02:21 result.txt
15658182 drwxr-xr-x  5 esser  staff    170 Apr  1 17:21 s5-css
15662226 -rwxr-xr-x  1 esser  staff  34312 Apr  2 00:24 sched-rr10.nw
```

By the way, the tool does know += and = for chunks:

⟨*header file inclusion*⟩+≡

```
  // watch above: this should be +=, not just =
  #include "../somethingelse.h"
```

And as a closing remark we present some very long source code which is all in one chunk—we should not do that in reality of course. It is just to show that the conversion tools can cope with it.

⟨*long code block*⟩≡

```
void syscall_waitpid (struct regs_syscall *r) {
  // ebx: pid of child to wait for
  // ecx: pointer to status
  // edx: options (ignored)
  int chpid = r->ebx;
  int *status = (int*)r->ecx;
  printf ("[%d] in syscall_waitpid, status (1) = 0x%x\n", current_task, status);
  ⟨imagine an even longer code block⟩
  ⟨imagine a code block which is yet longer⟩

  printf ("[%d] waitpid: waiting for pid %d\n", current_task, chpid);
  thread_table[current_task].state = TSTATE_WAITFOR;  thread_table[current_task].waitfor =
  remove_from_ready_queue (current_task);
  add_to_blocked_queue (current_task, &waitpid_queue);

  printf ("[%d] waitpid: calling yield\n", current_task);
  // syscall_yield (r);   // here we yield
  inside_yield = true;
  scheduler (r, SCHED_SRC_WAITFOR);

  /*
  if (thread_table[current_task].state != TSTATE_WAITFOR) {
    printf ("[%d] in syscall_waitpid: wrong return!\n", current_task);
    return;
  };
  */

  // asm ("int $0x81");  // call scheduler

  // PROBLEM
  // We come back here after scheduler(), but with the memory and
  // stack of a different thread. This is bad...

  inside_yield = false;

  printf ("waitpid: returned from yield (pid=%d)\n", current_task);

  // now we’ve returned from syscall_yield, the child must have
  // finished

  // unblocking this process happens in syscall_exit() !
  // remove_from_blocked_queue (current_task, &waitpid_queue);
  // add_to_ready_queue (current_task);

  // return value of waitpid is the process id of the terminated
  // child. We expect that syscall_exit() has updated the waitfor
  // field of the parent’s TCB:
  chpid = thread_table[current_task].waitfor;
  if (chpid>0 && chpid<MAX_THREADS && thread_table[chpid].used) {
    printf ("current_task = %d\n", current_task);
    printf ("chpid = %d\n", chpid);
    r->eax = chpid;

    // the exitcode is in the child’s exitcode field:
    printf ("in syscall_waitpid, status (2) = 0x%x\n", status);

    printf ("A...\n");
    printf ("in syscall_waitpid. exitcode = %d\n", thread_table[chpid].exitcode);
    printf ("B...\n");
    *status = thread_table[chpid].exitcode;
    printf ("in syscall_waitpid. *status = %d\n", *status);

    // now remove child process
  } else {
    // *status = -1;
  }
  printf ("going to return from syscall_waitpid\n");
  return;
}
```

2 Resources

We used

- Presentation software **S5**, `http://meyerweb.com/eric/tools/s5/`
- JavaScript Editor **nicEdit**, `http://nicedit.com/`

⟨*credits*⟩≡

```
// credits:
// - S5, http://meyerweb.com/eric/tools/s5/
// - nicEdit, http://nicedit.com/
```