EPITA S3 promo 2018
Practical Programming - Machine Test

Marwan Burelle*

Friday, November 14, 2013

Instructions:

You must read the whole subject and all these instructions. Every explicit instructions in the subject are mandatory. Points lost for ignoring subject rules are not open to arguments, including compiling issues or usage of directory hierarchy.

Your home directory during the test is temporary, in this directory you'll find a directory `subject` *and a directory* `rendu`*. In the* `subject` *directory, you'll find a sub-directory called* `Skel`*, you have to copy the* ***content*** *of this directory in your* `rendu` *directory.*

You must make regular uploads to be sure not to lose your work. In order to upload your work, simply call the command `rendu`.

Here is given as example, commands to perfom the needed copy from `subject/Skel` *directory to* `rendu` *directory:*

```
> cd
> cd exam/subject
> cp Skel/* ~/exam/rendu/
```

In this directory you'll find: a `Makefile` *offering targets to compile your code, some annex files, a file for each questions named* `questionXX.c`*. Only question files can be modified, all other files wil be replaced by the original one during the automatic correction.*

The `Makefile` *offers a target building a test program for each question. This test program will perform all the interraction part (input and output) and call your (or yours) function(s) with the correct expected parameters. In order to target the build of these test programs, you need to issue the command (for question number XX):* `make questionXX`*.*

During automatic correction, this `Makefile` *will be used and thus the question XX will be evaluated only if* `make questionXX` *succeed. Of course,* ***the grade will depends on the correctness of your answer****.*

Scale information are only indicative and may be changed later.

At the end of the document, you'l find the extra sections providing:

- *Advices about test programs;*
- *List of all files that* **must** *be in your* `rendu` *directory.*

There are 25 points and 12 questions in this test.

*marwan.burelle@lse.epita.fr

C: Bases

Question 1 (1)

Write the following function(s):

```
unsigned long power(unsigned long a, unsigned long b);
```

`power(a,b)` compute the integer a^b.

Exemple 1.1:

```
shell> ./question01 0 5
Fixed tests:
  power( 2,  0) =                    1 (expected 1)
  power( 2,  1) =                    2 (expected 2)
  power( 2,  2) =                    4 (expected 4)
  power( 2,  3) =                    8 (expected 8)
  power( 2,  4) =                   16 (expected 16)
Random tests:
  power( 9,  6) =               531441
  power( 3, 15) =             14348907
  power( 3, 15) =             14348907
  power( 4, 12) =             16777216
  power( 3,  1) =                    3
```

Question 2 (1)

Write the following function(s):

```
unsigned long fibo(unsigned long n);
```

`fibo(n)` returns the $n-$th terms of the Fibonacci sequence.

Here is a reminder of the Fibonacci sequence:

$$\texttt{fibo}(n) = \begin{cases} 0 & \text{when } n = 0 \\ 1 & \text{when } n = 1 \\ \texttt{fibo}(n-1) + \texttt{fibo}(n-2) & \end{cases}$$

Exemple 2.1:

```
shell> ./question02 10
fibo(0) = 0
fibo(1) = 1
fibo(2) = 1
fibo(3) = 2
fibo(4) = 3
```

```
fibo(5) = 5
fibo(6) = 8
fibo(7) = 13
fibo(8) = 21
fibo(9) = 34
```

Question 3 (1)

Write the following function(s):

```
unsigned long fact(unsigned long n);
```

`fact(n)` returns the integer $!n$.

$$\texttt{fact}(n) = \begin{cases} 1 & \text{when } n = 0 \\ n * \texttt{fact}(n-1) \end{cases}$$

Exemple 3.1:

```
shell> ./question03 0 5
fact(13) = 6227020800
fact(01) = 1
fact(12) = 479001600
fact(10) = 3628800
fact(08) = 40320
```

Question 4 (2)

Write the following function(s):

```
unsigned hamming(unsigned a, unsigned b);
```

`hamming(a, b)` compute the Hamming distance between a and b. The Hamming distance is defined as the number of different bits between a and b. For example, between the integer 4 (100 in binary) and the integer 5 (101 in binary) the Hamming distance is 1.

As a reminder: we can test the last bit (least significant) of an integer, using the modulus (`a % 2`). Also, dividing by 2 deletes the last bit.

Finally, it can be useful to consider the bit-by-bit *exclusive-or* operation between two integer: `a ^ b`. In the result of this operation, all bits that was different between a and b are set to 1 while the others are set to 0. thus, the Hamming distance can be computed by counting the number of bits set to 1 in `a ^ b`.

Exemple 4.1:

```
shell> ./question04 0 5
Fixed Tests:
a = 00000000000000000000000000000000
b = 11111111111111111111111111111111
hamming(a, b) = 32
Random Tests:
a = 01101011100010110100010101100111
b = 00110010011110110010001111000110
hamming(a, b) = 15
a = 01100100001111001001100001101001
b = 01100110001100110100100001110011
hamming(a, b) = 11
a = 01110100101100001101110001010001
b = 00011001010010010101110011111111
hamming(a, b) = 17
a = 00101010111010001001010001001010
b = 01100010010101010110001110110 0
hamming(a, b) = 16
a = 00100011100011100001111100101001
b = 01000110111010000111110011001101
hamming(a, b) = 16
```

Question 5 (3)

Write the following function(s):

```
unsigned int_sqrt(unsigned x);
```

`int_sqrt(x)` computes the integer part of the square root of x. The integer square root (noted $\lfloor \sqrt{x} \rfloor$) of an integer is the integer solution to the equation:

$$\lfloor \sqrt{x} \rfloor^2 \leq x < (\lfloor \sqrt{x} \rfloor + 1)^2$$

We can compute this equation by using the Héron method (a specific case of the Newton method): starting from a partial solution y, we compute the next approximation y' using the mean:

$$y' = (y + x/y)/2$$

The algorithm stops when $y \leq y'$. We choose to start with an initial approximation greater than the root (in this case, x itself can be a good start.)

Exemple 5.1:

```
shell> ./question05 0 5
int_sqrt(1804289385) = 42476 (OK)
int_sqrt(846930888) = 29102 (OK)
int_sqrt(1681692779) = 41008 (OK)
int_sqrt(1714636917) = 41408 (OK)
int_sqrt(1957747795) = 44246 (OK)
```

C: Strings

Question 6 (2)

Write the following function(s):

```
void to_lower(char *s);
```

`to_lower(s)` transform into lower-case every letters from string *s*. The pointer *s* is not `NULL`. Of course, your function must leave unchanged character that are not letters (or letters already in lower-case.) You may take a look at the `ascii(7)` manual page.

Notes: the test program verifies that you haven't override the end of the input string (marked by the character of ASCII code 0.)

Exemple 6.1:

```
shell> ./question06 1 10
FIXED TESTS:
Original:
  "A FULL UPPER-CASE STRING."
After to lower:
  "a full upper-case string."
Off-bound check: OK OK
RANDOM TESTS:
Original:
  "\VA/vK~|%r"
After to lower:
  "\va/vk~|%r"
Off-bound check: OK OK
```

Question 7 (1)

Write the following function(s):

```
size_t mystrlen(char *s);
```

`mystrlen(s)` function calculates the length of the string *s*, excluding the terminating null byte (`'\0'`). The input pointer is not `NULL`. You must respect the expected behavior of the function `strlen(3)`.

Exemple 7.1:

```
shell> ./question07 0 5
s = "n{6\P"
mystrlen(s) = 5 -- check: OK
shell> ./question07 0 0
s = ""
mystrlen(s) = 0 -- check: OK
```

Question 8 (2)

Write the following function(s):

```
char *mystrncpy(char *dst, char *src, size_t len);
```

`mystrncpy(dst,src,len)` : function copies at most `len` bytes of the string pointed to by `src` to the buffer pointed to by `dst`. The strings may not overlap, and the destination string `dst` must be large enough to receive the copy. If there is no null byte among the first `len` bytes of `src`, the string placed in `dst` will not be null-terminated. If the length of `src` is less than `len`, `mystrncpy()` writes additional null bytes to `dst` to ensure that a total of `len` bytes are written.

Note: `mystrncpy()` **always** writes exactly `len` bytes, whatever is the length of `src`.

You should read carefully the manual page of `strncpy(3)` which provides a complete description of the expected function.

Exemple 8.1:

```
shell> ./question08 0 5
src = "n{6\P"

test: mystrncpy(dst,src,6)
dst = "n{6\P"
-- check:
  first char: OK
  last char: OK
  0 fill: OK
  overflow: OK

test: mystrncpy(dst,src,2)
dst = "n{"
```

```
-- check:
  first char: OK
  last char: OK
  overflow: OK

test: mystrncpy(dst,src,10)
dst = "n{6\P"
-- check:
  first char: OK
  last char: OK
  0 fill: OK
  overflow: OK

test: mystrncpy(dst,src,0)
-- check:
  overflow: OK
```

C: Linked Lists

Question 9 (2)

Write the following function(s):

```
struct s_list *duplicate_list(struct s_list *l);
```

`duplicate_list(l)` returns a **newly allocated** list containing every element of `l`, in the same order.

List are defined using the following structures:

```
struct s_list {
  struct s_list        *next;
  int                   val;
};
```

Note: the original list is deleted by the test program before printing the result of your function.

Exemple 9.1:

```
shell> ./question09 5
l =
  00 -> 01 -> 02 -> 03 -> 04
Cloning list ... rl = duplicate_list(l)
Delete (free) l ...
rl =
  00 -> 01 -> 02 -> 03 -> 04
```

Question 10 (4)

Write the following function(s):

```
struct list* merge(struct list *l1, struct list *l2);
```

`merge(l1, l2)` returns a list composed of the elements of the two lists `l1` and `l2` in increasing order. The original lists are themsleves sorted in increasing order.

Your function **must not** create a new list but must reuse cells from the original lists, modifying only the `next` pointer. Thus, you'll **never** need `malloc`, nor `free`.

Exemple 10.1:

```
shell> ./question10 5
l1 =
  01 -> 03 -> 05 -> 07 -> 09
l2 =
  00 -> 02 -> 04 -> 06 -> 08
l = merge(l1, l2)
l =
  00 -> 01 -> 02 -> 03 -> 04 -> 05 -> 06 -> 07 -> 08 -> 09
```

C: Tableaux

Question 11 (3)

Write the following function(s):

```
void select_sort(int tab[], size_t len);
```

`select_sort(tab, len)` sort the array `tab` of length `len` using a selection sort. The selection sort algorithm is the following one:

```
select_sort(tab, len):
  for i <- 0 to len - 1 do
    min <- i
    for j <- i + 1 to len - 1 do
      if tab[j] < tab[min] then min <- j
    done
    tab[i] <-> tab[min]
  done
```

Exemple 11.1:

```
shell> ./question11 0 5
Before sort:
tab = | 83 | 86 | 77 | 15 | 93 |
After sort:
tab = | 15 | 77 | 83 | 86 | 93 |
```

Question 12 (3)

Write the following function(s):

```
void hist(char *str, size_t len, size_t counts[]);
```

`hist(str,len,counts)` counts the number of different occurrences of every characters in the buffer `str` and stores the result in the array `counts`. The array `counts` is already allocated with 256 cells, but theses cells have not been initialized to 0. The buffer `str` contains exactly `len` characters and is not a string: you should read exactly `len` characters, even if you encounter the character `'\0'`.

For practical reasons, only characters using the lower 7bits will be tested (code from 0 to 127) but the `counts` array has 256 cells, and all unused cells **must** be initialized to 0.

Exemple 12.1:

```
shell> ./question12 0 40
s = "n{6\Pavw[:mO4=ZvMD^bvU;e'Rrs^4LJ1_$^A{S|"
hist(s, 40, counts)
| 0x00   :      | 0x20   :      | 0x40 @ :      | 0x60 ‘ :      |
| 0x01   :      | 0x21 ! :      | 0x41 A :    1 | 0x61 a :    1 |
| 0x02   :      | 0x22 " :      | 0x42 B :      | 0x62 b :    1 |
| 0x03   :      | 0x23 # :      | 0x43 C :      | 0x63 c :      |
| 0x04   :      | 0x24 $ :    1 | 0x44 D :    1 | 0x64 d :      |
| 0x05   :      | 0x25 % :      | 0x45 E :      | 0x65 e :    1 |
| 0x06   :      | 0x26 & :      | 0x46 F :      | 0x66 f :      |
| 0x07   :      | 0x27 ' :    1 | 0x47 G :      | 0x67 g :      |
| 0x08   :      | 0x28 ( :      | 0x48 H :      | 0x68 h :      |
| 0x09   :      | 0x29 ) :      | 0x49 I :      | 0x69 i :      |
| 0x0a   :      | 0x2a * :      | 0x4a J :    1 | 0x6a j :      |
| 0x0b   :      | 0x2b + :      | 0x4b K :      | 0x6b k :      |
| 0x0c   :      | 0x2c , :      | 0x4c L :    1 | 0x6c l :      |
| 0x0d   :      | 0x2d - :      | 0x4d M :    1 | 0x6d m :    1 |
| 0x0e   :      | 0x2e . :      | 0x4e N :      | 0x6e n :    1 |
| 0x0f   :      | 0x2f / :      | 0x4f O :    1 | 0x6f o :      |
| 0x10   :      | 0x30 0 :      | 0x50 P :    1 | 0x70 p :      |
| 0x11   :      | 0x31 1 :    1 | 0x51 Q :      | 0x71 q :      |
| 0x12   :      | 0x32 2 :      | 0x52 R :    1 | 0x72 r :    1 |
| 0x13   :      | 0x33 3 :      | 0x53 S :    1 | 0x73 s :    1 |
| 0x14   :      | 0x34 4 :    2 | 0x54 T :      | 0x74 t :      |
| 0x15   :      | 0x35 5 :      | 0x55 U :    1 | 0x75 u :      |
| 0x16   :      | 0x36 6 :    1 | 0x56 V :      | 0x76 v :    3 |
| 0x17   :      | 0x37 7 :      | 0x57 W :      | 0x77 w :    1 |
| 0x18   :      | 0x38 8 :      | 0x58 X :      | 0x78 x :      |
| 0x19   :      | 0x39 9 :      | 0x59 Y :      | 0x79 y :      |
| 0x1a   :      | 0x3a : :    1 | 0x5a Z :    1 | 0x7a z :      |
| 0x1b   :      | 0x3b ; :    1 | 0x5b [ :    1 | 0x7b { :    2 |
```

```
| 0x1c   :      | 0x3c < :      | 0x5c \ :    1 | 0x7c | :   1 |
| 0x1d   :      | 0x3d = :    1 | 0x5d ] :      | 0x7d } :     |
| 0x1e   :      | 0x3e > :      | 0x5e ^ :    3 | 0x7e ~ :     |
| 0x1f   :      | 0x3f ? :      | 0x5f _ :    1 | 0x7f   :     |
Total count: 40
```

About The Test Session

Once the test is over, you must leave your session by closing the clock (that's the only way.) Note that when the test is over, your session will close directly.

When the session closed, you'll be prompted for your password (the one used to login.) This will end the test (your *rendu* directory will be archived and sent to the collecting server.) **You must not shutdown the computer before the completion of this final step, otherwise your work will be lost.**

You can send intermediary versions of your test by using the shell command `rendu`. It is strongly advised that you do so to prevent data lost before the end of the test.

Even after the end of the test (in the few minutes following the test, of course), you can restart your computer to eventually re-send your work (this may be required sometimes if something goes wrong during the final step.)

About Questions Skel

For every question, a skeleton of code is provided. This code is the minimal requirement for the compilation of the file *w.r.t.* the test program. The content of the skeleton will also induce a failure at execution time and thus you must remove the the body of the function(s). In C, be sure to remove (or comment) the `REMOVE_ME()` line of code: if it's still in the file, it will probably be executed anyway.

Exemple 1:
For example, if you're asked for the following function:

```
int identity(int x);
```

`identity x` returns `x`.
You'll find the following skeleton:

```
int identity(int x) {
  /* FIX ME */
  REMOVE_ME(x);
}
```

Your answer will look like:

```
int identity(int x) {
  return x;
}
```

About test programs

When invoked with `make questionXX`, `make` will build a binary program named `questionXX`. This program can be used to test your answer to the question X in this subject. All binary wait

for parameters et display a small help when run with `-help`.

Exemple 2:
For example, the program for question 1 (this is an example and may not corresponds to the actual question 1) will display:

```
Question 1:
./question01 graine taille
  -help   Display this list of options
  --help  Display this list of options
```

The two parameters are thus: `graine` (seed) and `taille` (size). These parameters are present in most question: the seed is used to initilize the random number generator (for a given seed, the generator will produce the same sequence of number) and the size can be either the size of generated data (for list or strings ...) or the number of tested ...

If you need more detail, read the `test_qXX.c` files.

Lists of Files

Immutable Files

Makefile
base_test.c
base_test.h
cheaters.h
question01.h
question02.h
question03.h
question04.h
question05.h
question06.h
question07.h
question08.h
question09.h
question10.h
question11.h
question12.h
question13.h
skel.h
test_q01.c
test_q02.c
test_q03.c
test_q04.c
test_q05.c
test_q06.c
test_q07.c
test_q08.c
test_q09.c
test_q10.c
test_q11.c
test_q12.c

Answer Files

question01.c
question02.c
question03.c
question04.c
question05.c
question06.c
question07.c
question08.c
question09.c
question10.c
question11.c
question12.c