System-oriented Programming
Spring 2018

S02

Professor : Philippe Cudré-Mauroux
Assistant : Michael Luggen

Submitted by Sylvain Julmy

Exercice 4

- `./wcount`

  This command would just invoke the program `./wcount`, which would read the standard input (keyboard) in order to obtain something to process. We just have to type something, including <ENTER> char and then we send the EOF char by <CTRL+D>. The ouput is printed on the standard output.

- `./wcount < wcount.c`

  This command invoke the program `./wcount`, but this time the standard input of the program is the file named `wcount`. The ouput is printed on the standard output.

- `./wcount < wcount > test`

  This command invoke the program `./wcount`, but this time the standard input of the program is the file named `wcount.c`. The ouput is redirected to a file named `test`.

- `cat wcount.c | ./wcount`

  The `cat` command will print the content of the file `wcount.c` on the standard output, but using the `|` operator, the standard output is redirected into the standard input of the `./wcount` program.

- `grep { wcount.c`

  The `grep` program would filter and print the content of the file `wcount.c`. This command would print all the line of `wcount.c` that contains a `{` symbol.

- `grep { wcount.c | ./wcount`

  It is the same as before except that the standard output of the `grep` command is redirected to the standard input of the `./wcount` command. So the whole command would count the number of word, letter, ... of all the line of `wcount.c` that contains a `{` symbol.

- `grep -l { * | ./wcount`

  The `-l` option of `grep` is suppressing the normal output of the command and instead print the name of all the file from which the output would normally be printed. The `*` symbol represent all the file of the current directory. So the `grep` command would output all the filename of the current directory that contains the `{` symbol.

Exercice 5

- `printf("%c %i\n", c, c);`

  We don't need any type casting here, `c` is a variable of type `char` which is automaticaly seen as an integer when using the `%i` formatter. 65 is the ascii value of "A".

  Output : `A 65`

- `printf("%c %i\n", i, i);`

  We don't need any type casting here, `i` is a variable of type `int` which is automaticaly downgraded into a `char` by simply "cutting" the extra-part of the `integer` and put it into a `char`.

  Output : `A 65`

- `printf("%f %i\n", pi, (int)pi);`

  This time we need an explicit type casting from a `float` into an `integer` because the representation, in memory, of floating point number and integer number is different and the compiler has to to some additionnal work in order to correctly transform the float value (by rounding the value from 3.14 to 3) into an integer. Without the explicit type casting, the number would have been read directly (the significand, the base and the exponant would'nt have been read separetely).

  Output : `3.140000 3`

Exercice 6

First, the decimal value of the ascii character `@` is 64. The output of the program is the following

```
@ 64 100 40
@ 64 100 40
@ 64 100 40
```

Three times the line printed is the same because :

- the decimal value of `@` is 64,
- the decimal value of `\100` is 64,
- the decimal value of `\x40` is 64.

Exercice 7

The output of the program is the following :

```
1 0
0 1 2
```

In C, each enum fields is represented by an integer value from 0 to $n-1$ where n is the number of fields in the enum. Then, `TRUE` as the value 1 and `FALSE` as the value 0, that's why the first line is `1 0`.
For the second line, C_1 is initialize to `RED`, which is the first field of the `color_tag` enum, so the value of `RED` is 0. C_2 is initialize to $C_1 + 1$ which is $0 + 1 = 1$. Finally C_3 is initialize to `BLUE` which is the third field of the `color_tag` enum, so the value of `BLUE` is 3. Then outputing C_1, C_2 and C_3 would be `0 1 2`.

Exercice 8

We denote by n any integer not equal to 0.

```
p || !q
```

p	q	p	||	!	q
n	n	n	1	0	n
n	0	n	1	1	0
0	n	0	0	0	n
0	0	0	1	1	0

```
p && (p == q)
```

p	q	p	&&	(	p	==	q	)
n	n	n	1		n	1	n	
n	0	n	0		n	0	0	
0	n	0	0		0	0	n	
0	0	0	0		0	1	0	

```
p && (p = q) || (p = !q)
```

First we add parentheses to clearly show the order of evaluation :
`( p && (p = q) ) || (p = !q)`

p	q	(	p	&&	(	p	=	q	))	||	(	p	=	!	q	)
n	n		n	1		n	n	n		1		n	0	0	n	
n	0		n	0		n	0	0		1		0	1	1	0	
0	n		0	0		0	n	n		0		n	0	0	n	
0	0		0	0		0	0	0		1		0	1	1	0	

Exercice 9

Exercice 10

Exercice 11

Exercice 12

Exercice 13