SAÉ-15 – Data Processing (*Traitement de données*)

Connection statistics of a web server

Abstract

The proposed *SAÉ* consists in writing a program in C language allowing to generate statistics on the connections of a web server. The program will have three types of interfaces: by default, an interface for the web and, depending on the call options, a graphical interface and a text interface.

You will have to:

1. Write a program to process log files of an apache server offering three interface modes (two graphical modes allowing to visualize the statistics in the form of histograms and a text mode).
2. Set up a demonstration model to test the program in real conditions of use.

Tools

To realize this *SAÉ* you will work under Linux using exclusively the virtual machine that you will find on the Moodle space RT1SA05-SAE prog, file Debian-SAE15.ova.

If you have not already done so, install VirtualBox on your host machine. The provided virtual machine is small enough to fit on a 4GB USB stick. Download it, copy it to a directory on your host machine where you have write access. Then in Virtual Box, proceed as follow: File menu > Import appliance > select the file Debian-SAE15.ova.

The Apache web server is already installed and configured to accept CGI. The Apache configurations are standard: the root of the site is `/var/www/html/`, the CGI directory is `/usr/lib/cgi-bin/` (to write access to these two directories you will have to use the `sudo` command).

In the same location on Moodle, you will find a file `SAE15-2021.tgz`. From the virtual machine, retrieve this file and place it in `/home/user`. Then, open a command terminal and do:

```
user@debian: ~$ tar xvf SAE15-2021.tgz
user@debian: ~$ cd SAE15/
user@debian: ~/SAE15 $ make install
user@debian: ~/SAE15 $ firefox "http://127.0.0.1/"
```

Thus, you are ready to work. A sample code can be found in the directory `/home/user/SAE15/src/` but the `Makefile` to compile and install the program is not provided.

Example of a realization

The figure below visualizes the execution of the program `/usr/lib/cgi-bin/versiondynamique` (you do not have the source code of this program). Here, it is used in its default version (*ie.* the web interface):

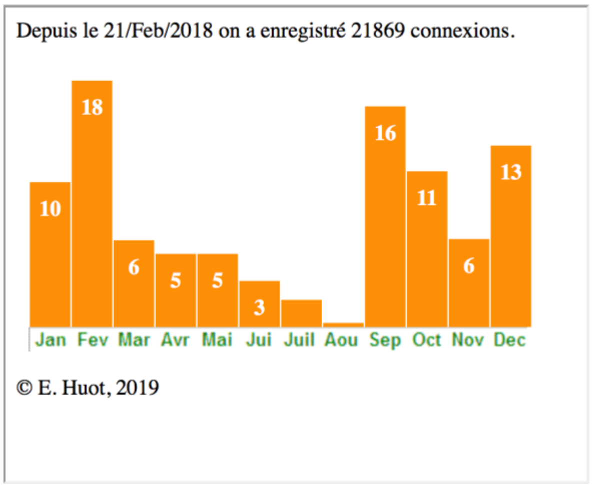


Figure 1: Connection statistics (web display).

The figure below shows a screen shot of the same program, using its graphical interface, while it is launched with its `-gr` option. Note that you are not asked to view this image *via* the web interface:

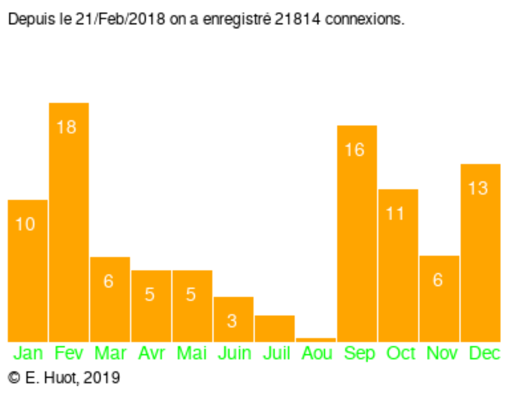
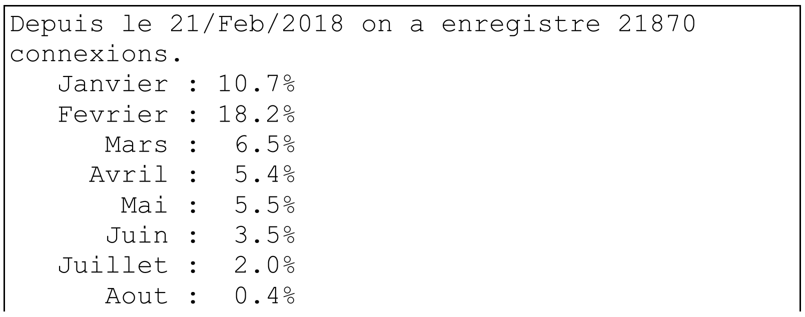


Figure 2: Connection statistics (graphic display).

The figure below shows a screen shot of the same program, using its textual interface, while it is launched with its `-txt` option. Note that you are not asked to view this image *via* the web interface:



```
Depuis le 21/Feb/2018 on a enregistre 21870
connexions.
   Janvier : 10.7%
   Fevrier : 18.2%
      Mars :  6.5%
     Avril :  5.4%
       Mai :  5.5%
      Juin :  3.5%
   Juillet :  2.0%
      Aout :  0.4%
```

```
 Septembre : 16.3%
   Octobre : 11.5%
  Novembre :  6.5%
  Decembre : 13.5%

(c) E. Huot, 2019
```
Figure 3: Connection statistics (text display).

Subject

Context

You are a web master and system administrator, and you are asked to create a program that allows you to make some simple statistical calculations on the connections to this site. The results of these calculations must be able to be visualized either *via* a display in text mode, or *via* a graphical interface proposing a representation in histogram, or *via* a specialized page of the site. You have to write a program, in C language, performing the calculations based on the file `/var/log/apache2/access.log` summarizing the trace of all the connections to the site and proposing the three user interfaces.

Todo tasks

Your program must necessarily analyze the contents of the file `/var/log/apache2/access.log` and visualize:

- The total amount of requests recorded since the date of the first connection (each line of the log file corresponds to one request).
- A monthly distribution of these requests.

Bonus: you can propose other types of statistics (for example: the three most frequently used browsers).

Your program must necessarily offer three operating modes corresponding to the three types of interfaces listed below:

web (default interface).
    The program behaves like a CGI. Installed in the `/usr/lib/cgi-bin` directory. It generates HTML code and allows to visualize the monthly distribution of requests in the form of a histogram.

graphic (`-gr` option).
    The program opens a graphical window and displays the information. The monthly distribution of requests is visualized in the form of a histogram.

text (`-txt` option).
    The program displays the results as text in the terminal.

Work to be done and submitted

You must provide a Proof of Concept (PoC) of your website, containing of course your program (compilable and executable) as well as all the files necessary for its use. All your files will be compressed in a `.tgz` archive file (see submission).

This archive file will contain:

1. In the `SAE15/compterendu` directory, your report in PDF format (a `.pdf`, no other format accepted). This file will include the algorithms (you will explain why you chose this or that

data structure and present your algorithms in French in a simple pseudo-language), the listings of your source codes (**YES** it is mandatory to put your source codes both in the report and – of course – in the src directory), examples of execution traces (screen shots).

2. The `HTML` and `CSS` files needed to visualize the web interface.
3. All your `.c` and `.h` source files that will be placed in the `SAE15/src/` directory.
4. A `Makefile` allowing to compile (by a call to the `make` command) and to install (`make install`) your program in `/usr/lib/cgi-bin/` and the `HTML/CSS` files in `/var/www/html/`.

Tip list

The subject is not difficult but it is long and has some deliberately technical points. I do advice you to:

1. Start working on a sheet of paper and analyzing the `access.log` file by yourself.
2. Decompose your program into subprograms using top-down analysis.
3. Define the data structure(s) to be used at each step.
4. Test each step independently.

Keep it simple before you make it complex. Do not submit a program where you have thought of everything but that does not work. Prefer to propose a program that works but does not have all the functionalities you have been asked for.

Notation

Everything that is requested is evaluated (the filenames, their location, the respect of the submission guidelines, the presence of a `Makefile` and its behaviour, the functionalities, *etc*).

Approximately the grade will be split into three large and equal parts: theory (mainly algorithmic), program and other.

Failure to comply with an instruction resulting in the impossibility of correcting all or part of your work will result in a zero.

To correct we will start from the virtual machine that you have provided, download your last submission, type the command sequence:

```
user@debian: ~$ tar xvf binome1binome2.tgz
user@debian: ~$ cd SAE15/src
user@debian: ~/M2207/miniprojet/src$ make
user@debian: ~/M2207/miniprojet/src$ make install
user@debian: ~/M2207/miniprojet/src$ ./versiondynamique -txt
user@debian: ~/M2207/miniprojet/src$ ./versiondynamique -gr
user@debian: ~/M2207/miniprojet/src$ cd ..
user@debian: ~/M2207/miniprojet$ evince compterendu/*.pdf
user@debian: ~/M2207/miniprojet$ firefox "http:127.0.0.1"
```

and check that everything is consistent and works properly.

Logs

Logging

To manage correctly a web server, it is necessary to have feedback about the activity and performance of the server, as well as any problems that may arise. Most web servers allow you to keep track of requests and possibly problems that occur while using them. This is commonly referred to as "log files" or, more commonly, *logs*.

The Apache server is very flexible and allows the administrator to configure the logs.

Common Log Format

The usual configuration will save log entries in a format known as Common Log Format (CLF). Log file entries generated in CLF format look like this:

```
1157.55.39.254 - - [01/Dec/2018:00:42:25 +0100] "GET /index.php HTTP/1.1" 302 495 "-"
"Mozilla/5.0 (iPhone; CPU iPhone OS 7_0 like Mac OS X) AppleWebKit/537.51.1"
```

Each part of this log entry is described in the following:

`1157.55.39.254`
: This is the IP address (or the name) of the client (the remote host) that sent the request to the server.

`-`
: Hyphens (*ie* minus characters) indicate that the corresponding portion of information is not available.

`[01/Dec/2018:00:42:25 +0100]`
: The date and time the request was received. The format is as follow:

```
[day/month/year:hour:minutes:secondes zone]
day = 2 digits
month = 3 letters
year = 4 digits
hour = 2 digits
minutes = 2 digits
secondes = 2 digits
zone = ('+' | '-') 4 digits
```

`"GET /index.php HTTP/1.1"`
: The line of the client's request is enclosed in double quotes. It contains: the method used by the client here it is `GET`, the requested resource `/index.php` and the protocol used `HTTP/1.1`.

`302`
: This is the status code that the server returns to the client.

`495`
: Indicates the size of the object returned to the client, excluding headers. If no content was returned to the client, this part will contain "-".

`"-"`
: Indicates the site from which the client claims to have launched the request. (here - no indication).

`"Mozilla/5.0 (iPhone; CPU iPhone OS 7_0 like Mac OS X) AppleWebKit/537.51.1"`
: The User-Agent header of the HTTP request. This information identifies the client browser.

`access.log` file example

The `access_static.log` file listed, below, corresponds to an anthology of the original `/var/log/apache2/access.log` file. This is the file that was processed to generate the result of the static version (*cf* static section). Here are three lines of this *static* file:

```
151.80.39.160 - - [01/Jan/2018:00:19:14 +0100] "GET
/claroline/document/document.php?cmd=exChDir&file=L0V4ZW1wbGVzL0pBVkEtRUVfV0VCL01WQzItSGliZXJ
uYXRlL2FjdGlvbnM%3D&cidReset=true&cidReq=imp_CLD&sort=date&dir=4 HTTP/1.1" 302 722 "-"
"Mozilla/5.0 (compatible; AhrefsBot/5.2; +http://ahrefs.com/robot/)"
151.80.39.105 - - [01/Jan/2018:00:46:42 +0100] "GET
/claroline/user/userInfo.php?uInfo=60155&cidReset=true&cidReq=M3206 HTTP/1.1" 302 558 "-"
"Mozilla/5.0 (compatible; AhrefsBot/5.2; +http://ahrefs.com/robot/)"
151.80.39.162 - - [01/Jan/2018:00:53:49 +0100] "GET
/claroline/document/document.php?cmd=exChDir&file=L0VKQi9leGVtcGxlcy9wcm9tb3Rpb24yMDEwL3ByZW1
pZXItRUpC&cidReset=true&cidReq=imp_UML_J2EE&sort=date&dir=4 HTTP/1.1" 302 707 "-"
"Mozilla/5.0 (compatible; AhrefsBot/5.2; +http://ahrefs.com/robot/)"
```

CGI-BIN

Common Gateway Interface

The Common Gateway Interface, usually abbreviated to CGI, is an interface used by web servers.

How it works?

Instead of sending the content of a file (HTML file, image), the web server executes a program and returns the generated content. CGI is the standard that indicates how the request is transmitted between the program (CGI-BIN) and the server (HTTP).

One of the characteristics of the CGI interface is that it relies on standard streams (like `stdin`) to communicate with the server. It is therefore independent of any programming language. Thus, it is possible to write a CGI program in C. The program will only need to make some displays (by simple `printf()`) which will be interpreted by the HTTP server.

From the point of view of the HTTP server, it is necessary to configure it to associate the execution of the CGI program with certain URLs (this configuration has already been done for the apache server that is installed with the virtual machine that is provided to you). Typically, we configure the server so that all the files that are in a certain folder on the server (it is usually called `cgi-bin`) are executed when a request is made to access this file. The result generated by this call can be interpreted by the server.

In a practical way

All types of information that can be viewed by a web browser can be generated (HTML page, GIF image, JPEG, etc). You just have to tell the browser what type of information it is. In the case of an HTML page, the program simply displays (with a `printf()`) the first line (note the two line breaks symbolized by the ↵):

```
Content-type: text/html ↵
↵
```

Then any text displayed (still using `printf()`) will be interpreted as HTML code by the browser. You just have to put the executable program in the `/usr/lib/cgi-bin/` directory of the server to be able to visualize its content like any other web page.

Static version

Sample code

This section gives you the listing of a program in C language which, once compiled and installed, generates the displays corresponding to the analysis of the `access_static.log` file that can be viewed on the logs page.

This is a static version (the execution of the program always generates the same result whatever the content of the log file).

```c
/*
 * versionstatique.c
 *
 * © Copyright 2019 E. Huot <etienne.huot@uvsq.fr>
 *
 * This program is can only be used for education purpose.
 * It is part of a project, so-called SAE, aiming to learn
 * programmation bases using the C language.
 * It can only be used by the students of the RT departement,
 * Technological Institute of Velizy - Université de Versailles - Saint-
 * Quentin-en-Yvelines - Université de Paris-Saclay.
 *
 * The aim of the project is to propose a tool to vizualize a bunch of
 * statistics on a web server logs.
 * The following code propose a static version of the code that have to
 * be developped by the students.
 *
 */


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <libgraphique.h>

void lecture_style()
{
  /* On aurait pu utiliser un fichier css mais il est     */
  /* parfois plus pratique de définir le style dans le    */
  /* code HTML généré par ce CGI-BIN.                     */
  /* On le lit ce style depuis fichier "aecrire.html"     */
  /* puis on l'affiche.                                   */
  /* L'étudiant attentif pourra s'apercevoir aisément que */
  /* cette fonction n'est pas inutile pour ses propres    */
  /* besoins...                                           */

  FILE *fd;
  char str[1024];
  if ((fd = fopen("aecrire.html","r")) == NULL)
    fprintf(stderr,"ERREUR DE CHARGEMENT DE FICHIER\n");
  else {
    while (!feof(fd)){
      fgets(str,1024,fd);
      printf("%s",str);
    }
    fclose(fd);
  }
}

void versionweb()
{
  printf("Content-type: text/html\n\n");

  printf("<!doctype html>\n");
  printf("<html lang=\"fr\">\n");
  printf("<body>\n");

  lecture_style();

/*
 * Notez que pour le web que les caractères accentués ont
 * été transformés en effet les chaînes de caractères C sont nativement
 * codées en ASCII et non en UTF-8.
 * L'étudiant désireux de bien faire devrait normalement prétraiter
 * tous les accents avant de les afficher en HTML.
 */
  printf("Depuis le 01/Jan/2018 on a enregistr&eacute; 998 connexions.<br>\n");

  printf("<div id = \"vertgraph\">\n");
  printf("    <ul>\n");
/*
 * remarquez que chaque barre de l'histogramme est placée tous les 31 pixels,
 * et que le label correspond ici à la hauteur/10, on n'affiche pas le label pour des
 * hauteurs < 25 sinon ça fausse l'affichage.
 */
  printf("        <li style=\"left: 10px;  height: 79px;\">8</li>\n");
  printf("        <li style=\"left: 41px;  height: 81px;\">8</li>\n");
  printf("        <li style=\"left: 72px;  height: 99px;\">10</li>\n");
  printf("        <li style=\"left: 103px;  height: 23px;\"></li>\n");
  printf("        <li style=\"left: 134px;  height: 137px;\">14</li>\n");
  printf("        <li style=\"left: 165px;  height: 113px;\">11</li>\n");
  printf("        <li style=\"left: 196px;  height: 127px;\">13</li>\n");
  printf("        <li style=\"left: 227px;  height: 73px;\">7</li> \n");
  printf("        <li style=\"left: 258px;  height: 134px;\">13</li> \n");
  printf("        <li style=\"left: 289px;  height: 12px;\"></li> \n");
  printf("        <li style=\"left: 320px;  height: 115px;\">11</li>\n");
  printf("        <li style=\"left: 351px;  height: 5px;\"></li> \n");
  printf("    </ul>\n");
  printf("</div>\n");

  printf("</html>\n");
  printf("</body>\n\n");

}

void versiontexte()
{
  printf("Depuis le 01/Jan/2018 on a enregistre 998 connexions.\n");
  printf("   Janvier :  7.9%%\n");
  printf("   Fevrier :  8.1%%\n");
  printf("      Mars :  9.9%%\n");
  printf("     Avril :  2.3%%\n");
  printf("       Mai : 13.7%%\n");
  printf("      Juin : 11.3%%\n");
  printf("   Juillet : 12.7%%\n");
  printf("      Aout :  7.3%%\n");
  printf(" Septembre : 13.4%%\n");
  printf("   Octobre :  1.2%%\n");
  printf("  Novembre : 11.5%%\n");
  printf("  Decembre :  0.5%%\n");
}

void versiongraphique() {
  Point p = {30,50};
  ouvrir_fenetre(430,350);
  dessiner_rectangle((Point){0,0},430,350,blanc);

  afficher_texte("Depuis le 01/Jan/2018 on a enregistré 998 connexions.",12,p,noir);

  p.y=300;
  afficher_texte(" Jan  Fev  Mar  Avr  Mai  Juin  Juil  Aou  Sep  Oct  Nov  Dec",14,p,vert);

  dessiner_rectangle((Point){30,221},30,79,orange);
  dessiner_rectangle((Point){61,219},30,81,orange);
  dessiner_rectangle((Point){92,201},30,99,orange);
  dessiner_rectangle((Point){123,277},30,23,orange);
  dessiner_rectangle((Point){154,163},30,137,orange);
  dessiner_rectangle((Point){185,187},30,113,orange);
  dessiner_rectangle((Point){216,173},30,127,orange);
  dessiner_rectangle((Point){247,227},30,73,orange);
  dessiner_rectangle((Point){278,166},30,134,orange);
  dessiner_rectangle((Point){309,288},30,12,orange);
  dessiner_rectangle((Point){340,185},30,115,orange);
  dessiner_rectangle((Point){371,295},30,5,orange);

  afficher_texte("8",14,(Point){40,231},blanc);
  afficher_texte("8",14,(Point){71,229},blanc);
  afficher_texte("10",14,(Point){97,211},blanc);

  afficher_texte("14",14,(Point){159,173},blanc);
  afficher_texte("11",14,(Point){190,197},blanc);
  afficher_texte("13",14,(Point){221,183},blanc);
  afficher_texte("7",14,(Point){257,237},blanc);
  afficher_texte("13",14,(Point){283,176},blanc);

  afficher_texte("11",14,(Point){345,195},blanc);

  actualiser();
  attendre_clic();
  fermer_fenetre();
}


int main(int argc, char* argv[])
{
    if (argc > 1) {
      if (strcmp(argv[1],"-txt")==0) {
        versiontexte();
      }
      if (strcmp(argv[1],"-gr")==0) {
        versiongraphique();
      }

    } else {
      versionweb();
    }

    return EXIT_SUCCESS;
}
```

Running the static web version

Once compiled, the code listed in the previous paragraph, the program must be placed in `/usr/lib/cgi-bin/`.

The figure below shows the execution of the program `/usr/lib/cgi-bin/versionstatic`. It is used here with its default interface (web).

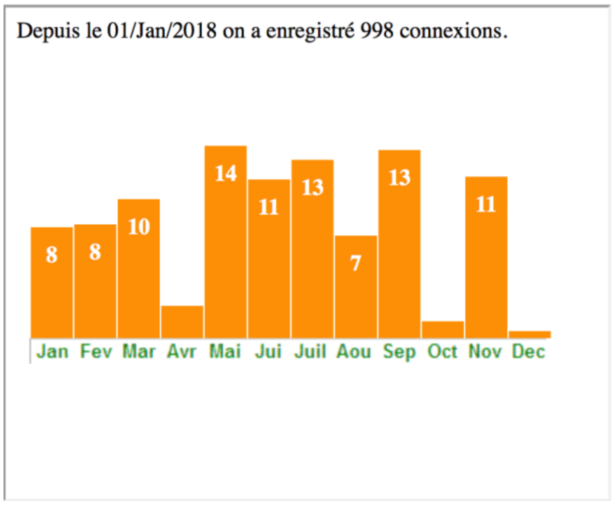

Depuis le 01/Jan/2018 on a enregistré 998 connexions.

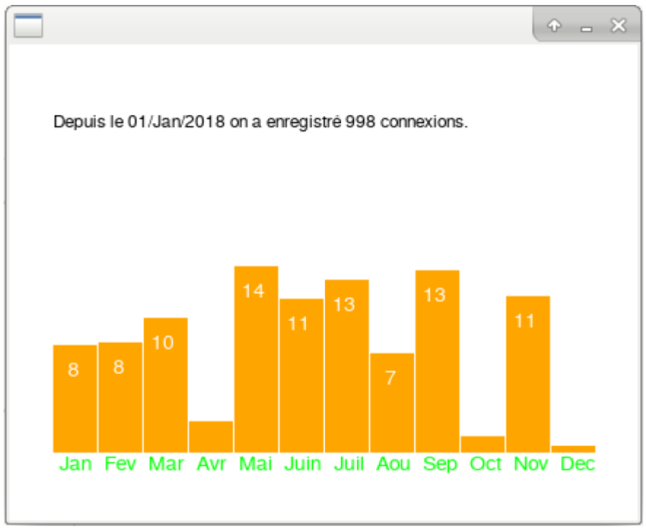
Figure 4: Execution of the static version (web display)

Running the static graphical version

When executed with the `-gr` option as in the following command line:

```
user@debian: ~$ /usr/lib/cgi-bin/versionstatique -gr
```

one obtain a visualization with the graphical interface which relies on the use of the graphical library (libgraphique) that you will use in further TP. Note that this version of the library has been slightly modified to use a font installed in a system directory of this Linux distribution and not the font `verdana.ttf` installed in a local directory.



Depuis le 01/Jan/2018 on a enregistré 998 connexions.

Figure 5: Execution of the static version (graphical display)

Running the static text version

When executed with the `-txt` option as in the following command line:

```
user@debian: ~$ /usr/lib/cgi-bin/versionstatique -txt
```

one obtains the following result:

```
Depuis le 01/Jan/2018 on a enregistre 998
connexions.
   Janvier :  7.9%
   Fevrier :  8.1%
      Mars :  9.9%
     Avril :  2.3%
       Mai : 13.7%
      Juin : 11.3%
   Juillet : 12.7%
      Aout :  7.3%
 Septembre : 13.4%
   Octobre :  1.2%
  Novembre : 11.5%
  Decembre :  0.5%
```

Figure 6: Execution of the static version (text display)

Submission

Deadline

For this project you will work in binomial constituted within the same TP group (possibly in trinomial if odd TP group, **nobody is allowed to work alone**).

You have until January 9th 2021 23:59 to submit your work on Moodle. You can submit your work as many times as you want, only the last submission will be corrected. Do not wait until the last minute. The server may crash, especially if it is saturated with your submissions.

At each submission, check that the upload has been done correctly by downloading your file and checking its integrity.

Archive

Your archive file in `.tgz` format (containing all your files to be submitted - see above) must imperatively bear the family names (unaccented and without spaces) of the two students of the binomial in alphabetical order.

For example, for the binomial composed of Laurent Marsan and Etienne Huot, the file will be called: `HuotMarsan.tgz`

Only the `tar` command (not a graphical program like TheUnarchiver, Winzip, xarchiver, *etc*) should be used to create it.

If your files are correctly placed in the directory `~user/SAE15` you just will have to do:

```
user@debian: ~$ cd
user@debian: ~$ tar cfz HuotMarsan.tgz SAE15
```

Upload your file on Moodle

The submission will be done on Moodle in the RT1SA05-SAE prog lecture space in the *ad hoc* section.

Only the first student, in alphabetical order, should submit the work of the binomial, i.e. the `.tgz` archive file.

Bibliography

[1] E. Huot Bases de la programmation. Langage C. Polycopié de la ressource R107. IUT de Vélizy, Université de Versailles - Saint-Quentin-en-Yvelines, Université Paris-Saclay.
[2] Wikipedia make.
[3] Wikipedia Log file.
[4] Apache HTTP Server Documentation.
[5] Wikipedia Common Log Format.
[6] Wikipedia Common Gateway Interface.
[7] SuperSonique studio Introduction au CGI (Common Gateway Interface).
[8] I. Cohen CGI/Perl et JavaScript, la création de pages HTML interactives. Eyrolles, pp.294, 1996.
[9] Wikilivre Programmation C.
[10] B. Kernighan, D. Ritchie Le langage C - 2e éd - Norme ANSI. Dunod, 2014.
[11] man tar - la version GNU de l'utilitaire tar de gestion d'archives.