*Projet*

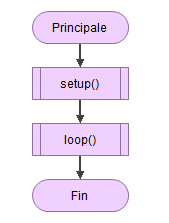
*Fondamentaux Scientifiques*

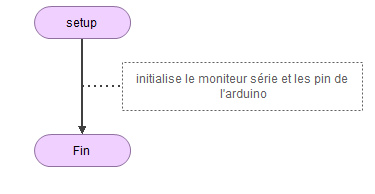
Documentation Technique

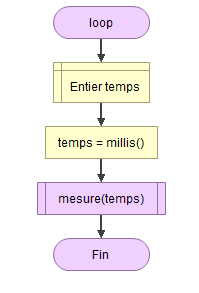
Sommaire

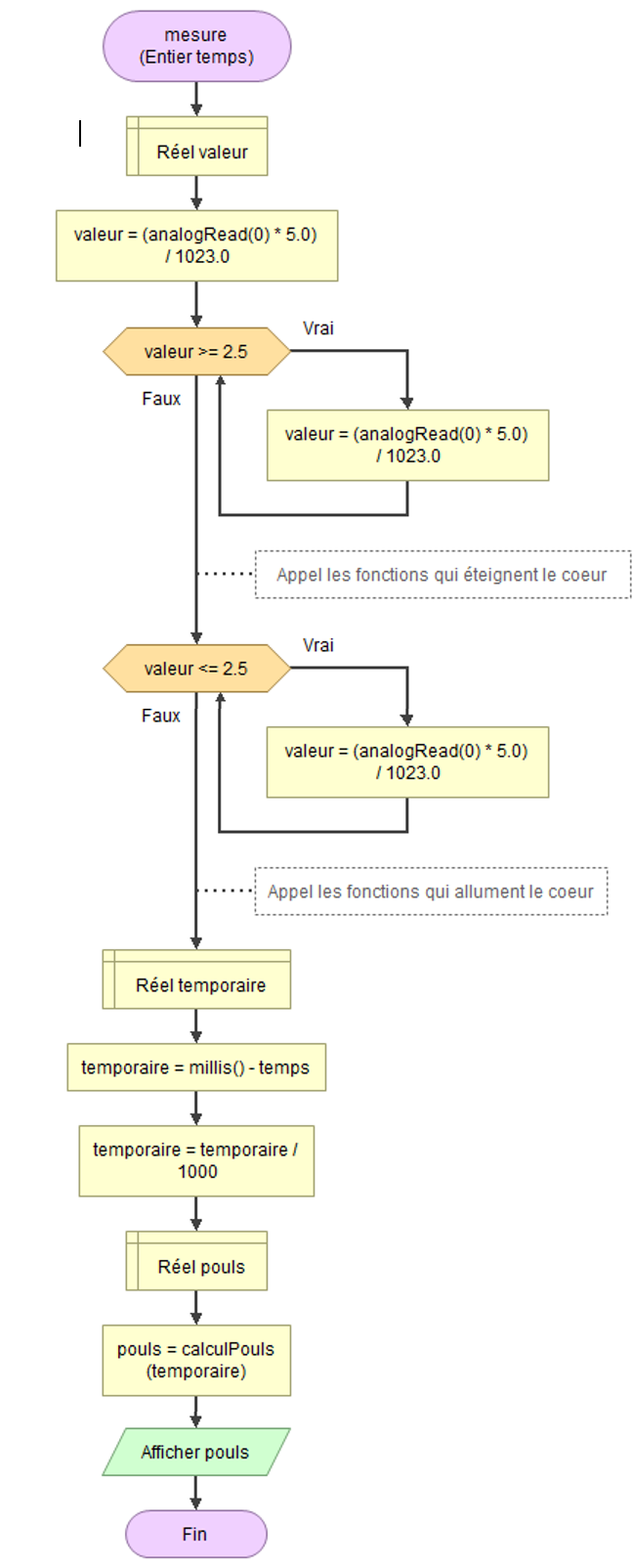
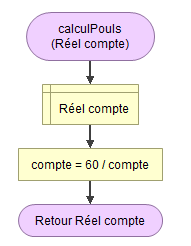
* Module 3.1
* Module 3.2
* Module 3.3
* Module 3.4

Module 3.1







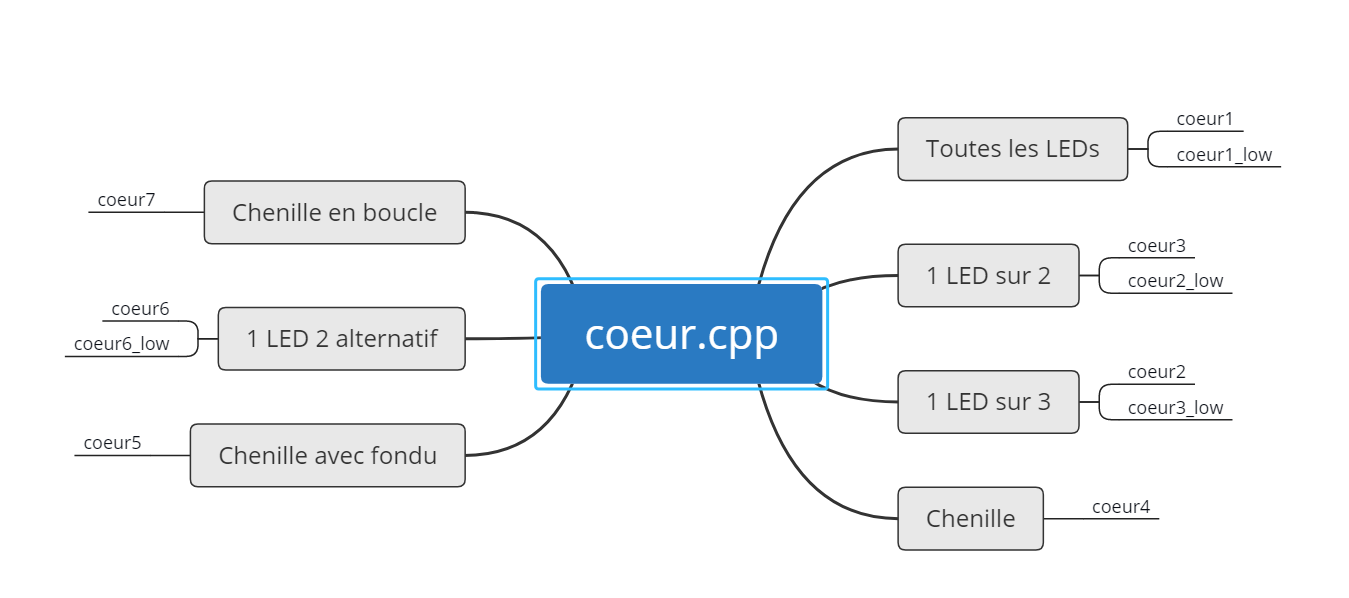
Pour faire un résume, le programme appelle la fonction mesure qui calcule la différence de temps entre deux battements, puis appelle la fonction calcul\_pouls, qui donne le nombre de battements par minute. En même temps de calculer le pouls, la fonction mesure appelle les différentes fonctions d’affichages du cœur pour l’animer (Module 2) (Algorigrammes visibles ci-dessous).

Module 3.2

## Programme cœur

Le programme qui gère le cœur est composé de deux fichier le cœur.cpp, le fichier source, et le cœur.h, le header.

Le fichier source du programme cœur a été écrit en c ++ parce que les bibliothèques Arduino ne sont que codée en c++. Ce fichier cpp est composé de 11 fonctions, certaines de ces fonctions sont liées comme coeur6 et coeur6\_low, voici ci-dessous le schéma de ce fichier source :



Nous avons rajouté deux fonctions, la fonction qui fait une LED sur deux en alternatif et la fonction qui fait le tour du cœur en mode chenille à chaque battement.

Comme expliquée précédemment, certaine fonction sont liées, en effet dans certain cas il faut éteindre les LEDs après le battement du cœur, ce sont les fonctions comprenant le mot low. Lorsqu’on repère un battement, on appelle la fonction normale et dans certain cas lorsqu’on repère la fin du battement on appelle la fonction low si besoins.

Le fichier header de ce programme est constitué des prototypes de toutes les 11 fonctions en public, ce qui permet de les utiliser dans tous le programme HeXart Care. Puis il y a aussi l’initialisation des variables en private, ce qui permet d’utiliser ces variables que dans le fichier source du programme cœur, ce qui permet d’éviter des erreurs si on réutilise le même nom de variable dans un autre fichier source.

## param.h

Le fichier param.h contient seulement la définition d’une constante, choisie par l’utilisateur dans menu.c, et la structure des header (ifndef, define…).

## Programme generationCode

Dans ce programme on génère un fichier param.h, et on initialise dans ce même fichier une constante.

Le fichier header de ce programme contient seulement le prototype de la fonction generationCode.

Le fichier source utilise la bibliothèque stdlib pour générer le fichier et écrire dedans. Voici un schéma du fonctionnement de cette algorithme :



Module 3.3

Pour ce module, nous avons pris le fichier d’extension .pde (Processing Development Environment) fourni. Son but était de récupérer les informations du port série correspondant aux sorties de l’Arduino et d’insérer ces valeurs dans un fichier .csv (Comma Separated Values) qui par défaut est ouvert sur Excel. Il insère deux valeurs par ligne (pouls puis temps en heures), ils sont séparés par un point-virgule à chaque fois.

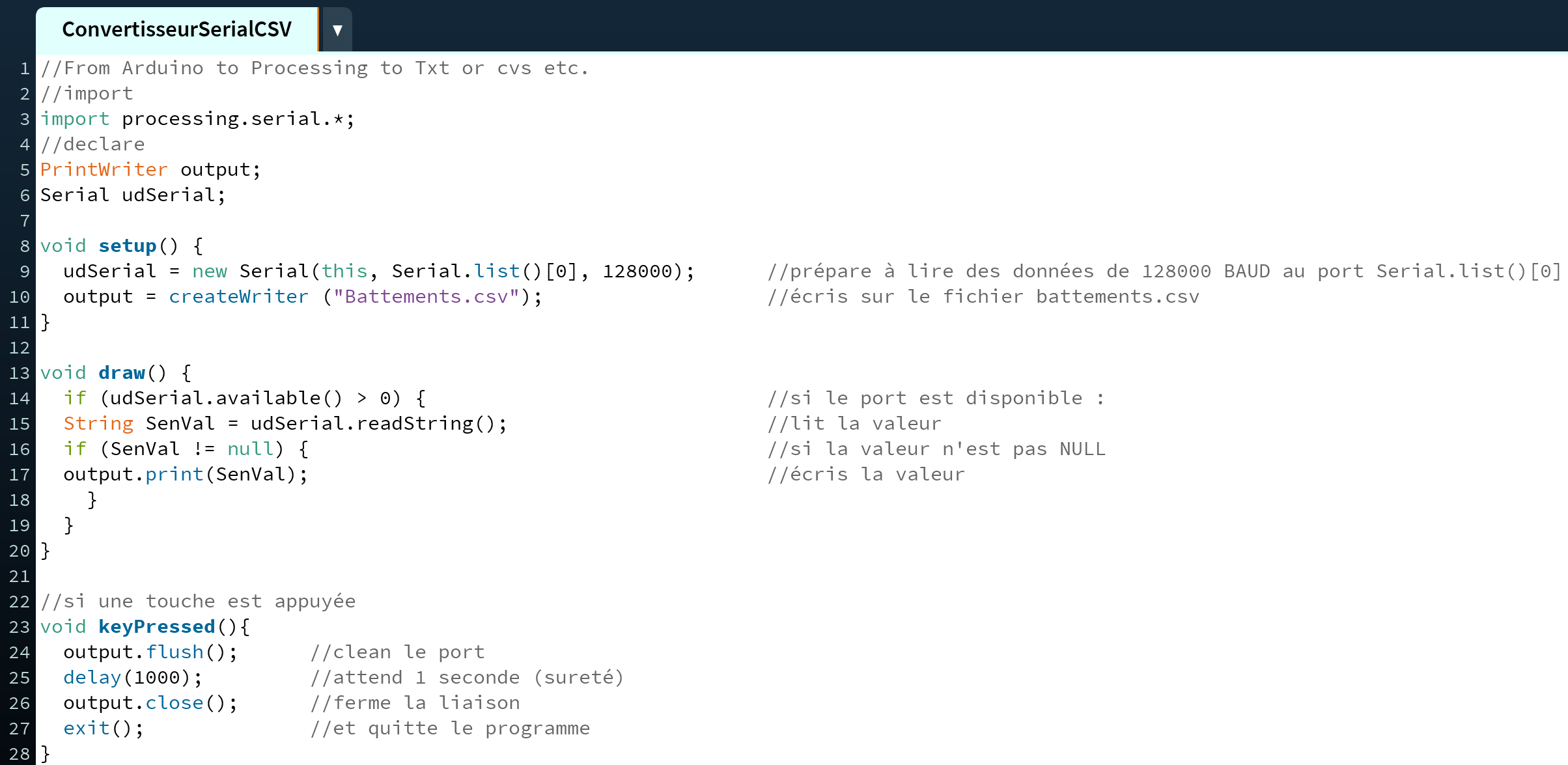


Figure – Capture d’écran du code Processing

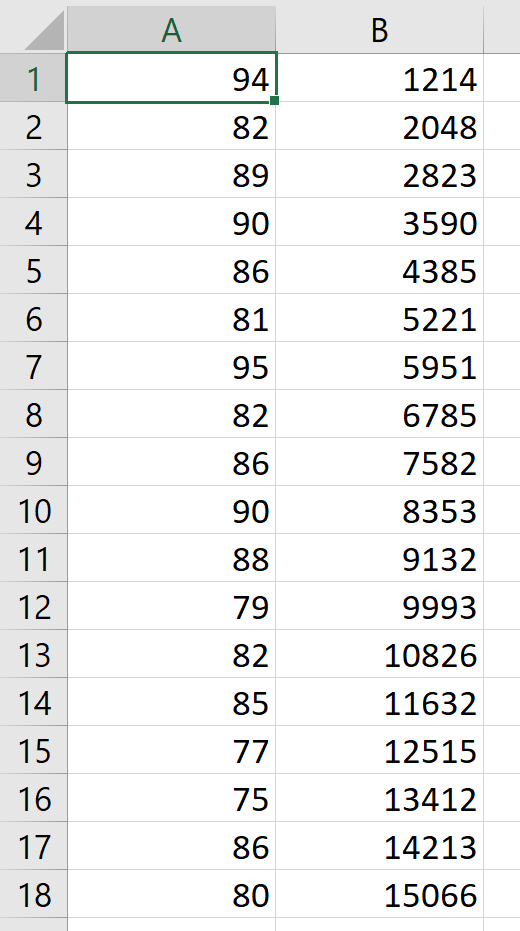
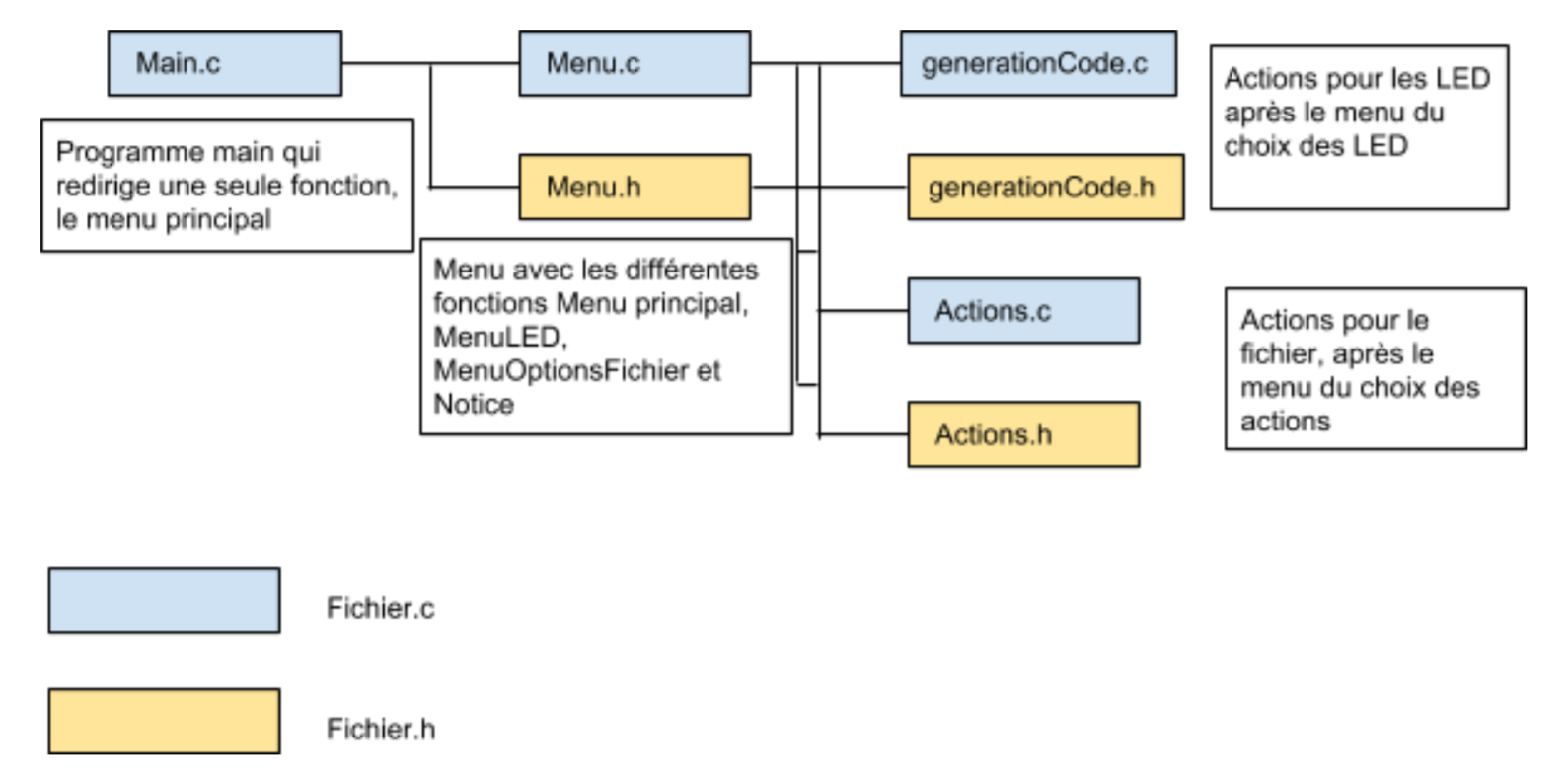


Figure – Capture d’écran du fichier .csv crée par le programme avec le ***pouls à gauche*** et le ***temps à droite.***

Module 3.4

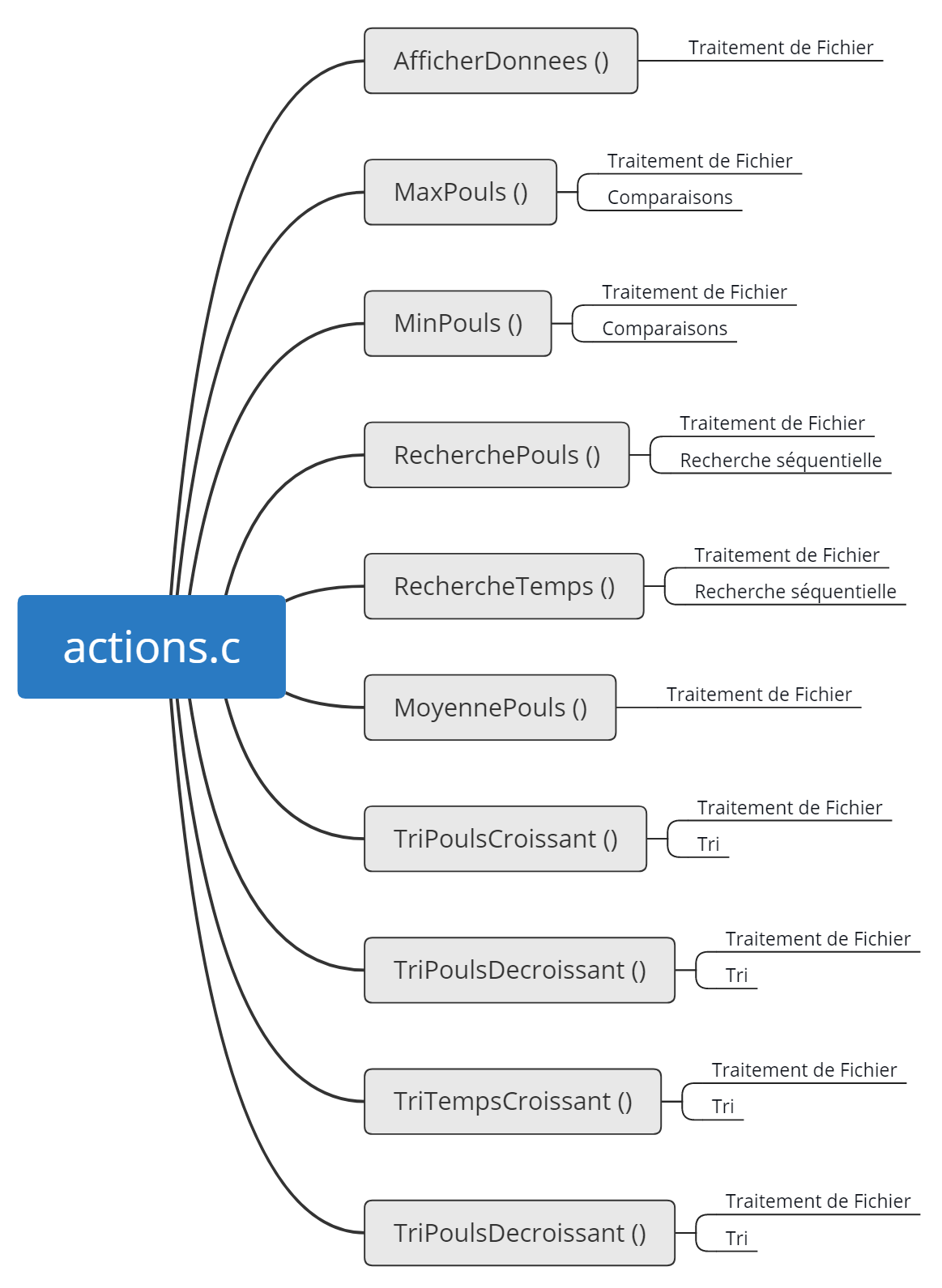
Le Module 3.4 est composé de plusieurs fichiers qui permettent de réaliser toutes les fonctions possibles sur les LED et le fichier CSV, Il comporte ainsi les programmes suivants :



Le main.c ne contient qu’une seule et unique fonction : MenuPrincipal() qui se trouve dans menu.c, cette fonction permet d’atteindre les sous menus LED et Actions qui se trouvent également dans menu.c. Une fois un sous menu sélectionné, nous avons deux choix possibles :

- LED nous redirige vers generationCode.c qui va créer le fichier param.h qui va gérer l’allumage des LED

- Actions qui nous redirige vers actions.c qui contient toutes les fonctions que l’on peut faire par rapport au fichier, vous les trouverez ci-dessous :



Pour mettre en place ces fonctions, nous avons dû lire le fichier :

* fopen → ouvre le fichier
* fgets → permet de prendre les informations d’un fichier
* fclose → ferme le fichier lorsqu’on a finit de faire nos actions dessus

Mais nous devons récupérer les deux données d’une ligne séparément, pour cela on utilise :

* string.h → bibliothèque qui nous permet d’utiliser strtok
* atoi → permet de convertir un string en int
* strtok → permet de diviser notre string en tokens avec un délimiteur (ici le « ; » )