2017 - 2018

Projet : Fondamentaux scientifiques

GROUPE 5

Table des matières

Partie I 2
Introduction 2
Répartition des rôles 3
Rappel des objectifs 4
Partie II – Module 1 : le cardiofréquencemètre 5
Rappel 5
Schémas 5
Montage électronique 6
Programme C Arduino 7
Partie III – Module 2 : le cœur de LEDs 9
Rappel 9
Schémas 9
Montage électronique 10
Programme C Arduino 10
Application de génération de param.h 12
Lien entre les modules 1 et 2 13
Partie IV – Module 3 : application Processing 14
Rappel 14
Programme 14
Partie V – Module 4 : application de traitement des données 15
Rappel 15
Programme 16
Partie VI – Bilan 22
Analyse des écarts 22
Bilans personnels 22
Montage final 23

Partie I

Introduction

HeXart Care, startup spécialisée dans l'électronique et l'informatique, travaille sur un projet de lecteur portatif de la fréquence cardiaque. Un de leurs ingénieurs étant parti, ils ont perdu tous leurs travaux, et ont donc besoin rapidement d'un prototype fonctionnel.

Le projet se découpe en quatre modules :

- Le premier module consiste en la fabrication d'un cardiofréquencemètre, qui fonctionne selon le principe de la pléthysmographie (qui consiste à détecter les battements cardiaques en mesurant le volume de sang dans les tissus à l'aide d'une source de lumière et d'un détecteur). Le code de ce module assure l'envoie des données recueillies par le montage vers une application (module 3) par voie série.
- Le second module vise à faire un cœur avec des LEDs, qui s'illuminera à chaque battement selon plusieurs schémas possibles. Un fichier définit le schéma à appliquer (chenille, une LED sur deux, etc···), ainsi le code sur l'Arduino s'adapte à ce fichier. Enfin, une application permet à l'utilisateur de choisir le mode qu'il désire, et génère le fichier cité précédemment.
- Le troisième module est une application qui reçoit les données envoyées par l'Arduino, pour les insérer dans un fichier .csv. Il vérifie également l'intégrité de ces informations après le passage par la voie série.
- Le dernier module est également une application. Celle-ci va proposer à l'utilisateur de visualiser les informations enregistrées par le cardiofréquencemètre, et effectuer des tris, recherches, etc...

Répartition des rôles

Avant d'assigner les rôles, nous avons discuté de la structure du projet, et ainsi des choses à réaliser au cours de la semaine.

Le projet étant découpé en quatre parties, nous avons assigné à chaque personne un module.

Ainsi le premier module a été réalisé par Louis, qui a réalisé le montage et codé la récupération du pouls. Il a été, par la fin, aidé par Christophe pour mettre en relation la récupération des pouls avec l'allumage des LEDs (module 2).

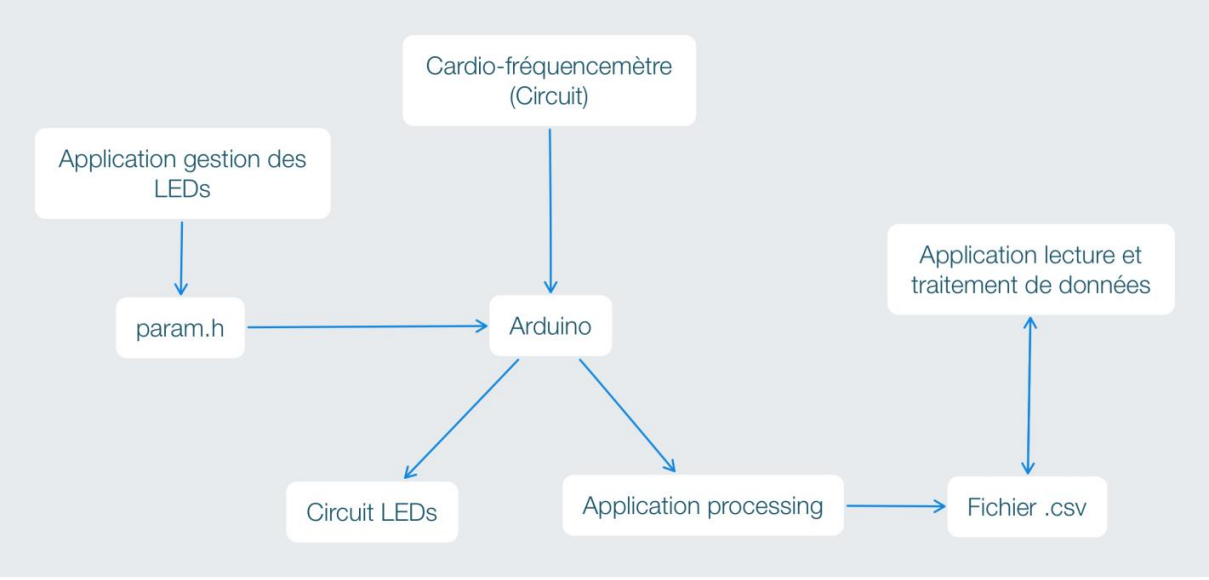
Le second module, l'allumage des LEDs, a été géré par Alexi. Il a tout d'abord réalisé le montage du cœur de LEDs, puis a créé les différents affichages possibles (c'est-à-dire les schémas tels que faire clignoter une LED, ou bien à la suite (en chenille)). Il a été aidé par Louis sur la fin pour peaufiner les différentes fonctions.

Le troisième module a été géré par Christophe. Le code de l'application Processing était déjà rédigé dans les grandes lignes, il l'a juste complété pour vérifier l'intégrité des informations avant la sauvegarde dans le fichier .csv. Christophe a ensuite aidé les autres membres du groupe dans leurs modules respectifs, et a géré la réunion des modules 1 et 2, en assurant la compatibilité du travail d'Alexi avec celui de Louis.

Finalement, Charlotte a travaillé sur le quatrième module tout au long de la semaine. Elle a ainsi codé l'application de lecture et traitement des données stockées dans le fichier .csv créé par l'application Processing (module 3). Elle a été aidée pour quelques fonctions par Christophe pour des bugs et soucis de logique.

Rappel des objectifs

L'objectif principal était de réaliser un dispositif capable de mesurer le pouls d'une personne, lorsqu'elle insère son doigt dans une pince prévue à cet effet. Lorsqu'un battement est détecté, le cœur de LEDs clignote selon un schéma définit préalablement, et une information (contenant le pouls, précédé de l'instant auquel il a été enregistré) vers une application Processing (le tout par voie série), qui enregistrera chaque information dans un fichier .csv. Une autre application lit ce fichier et affiche les données contenues, les trie, etc... Finalement, une dernière application doit être prévue pour indiquer quel schéma d'affichage doit respecter le cœur de LEDs lorsqu'un battement est détecté. Celle-ci génère un fichier compris par le programme stocké dans l'Arduino.



STRUCTURE DU PROJET

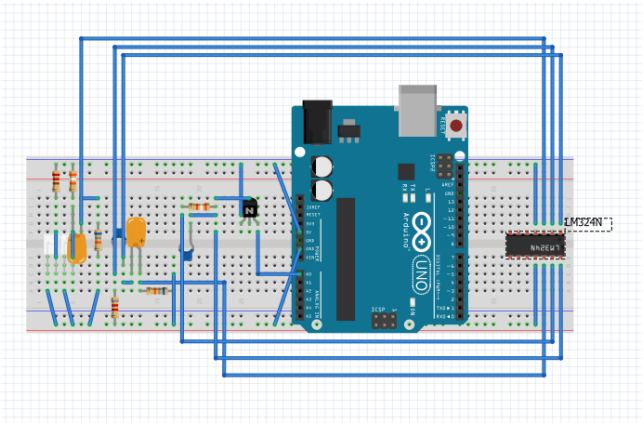
Partie II – Module 1 : le cardiofréquencemètre

Rappel

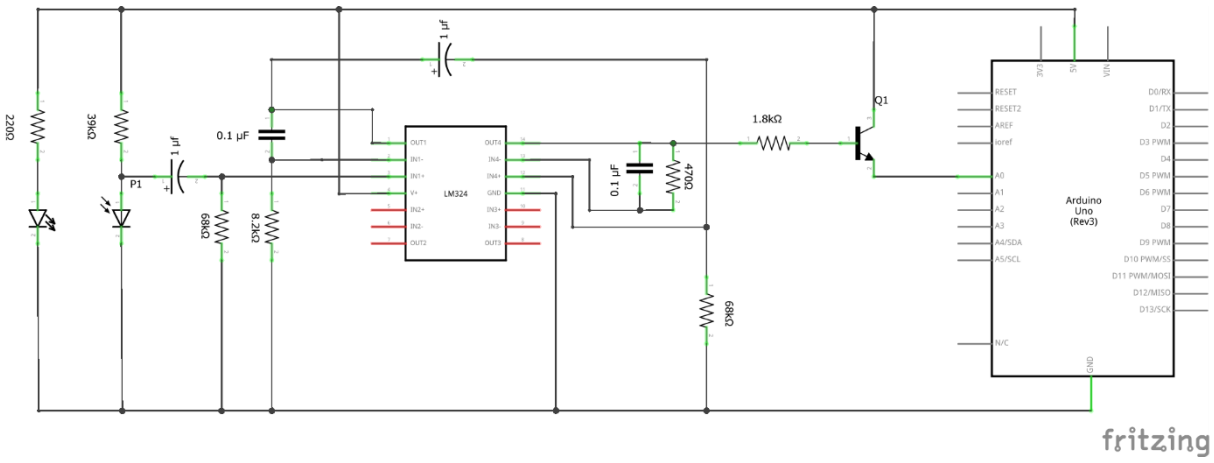
Ici, il s'agit de réaliser le montage électronique qui réalisera la détection des battements cardiaques, ainsi que le code pour calculer le pouls et l'envoyer vers l'application Processing (*voir module 3*).

Schémas

Vue platine



Vue schématique



Montage électronique

Nous avons donc une carte Arduino, ainsi qu'une breadboard. La carte Arduino envoie du courant vers différents composants (5V) (fil le plus haut sur la vue schématique), et le Ground de la carte est également relié à des composants (fil le plus bas sur la vue schématique).

Voyons l'utilité de chaque composant :

- LED Infrarouge : alimentée en continu par l'Arduino, elle émet de la lumière dans le domaine des infrarouges.
- Phototransistor : lui aussi alimenté en continu par la carte, il capte la lumière émise par la LED Infrarouge.
- Amplificateur : il augmente l'intensité du courant qui le traverse, et le contrôle.
- Transistor NPN : en fonction de la lumière reçue par le phototransistor, il laisse passer le courant (ou non) vers le port analogique de la carte Arduino.
- Condensateurs : ils servent à réaliser des filtres passe-haut et passe-bas, afin d'empêcher d'éventuelles interférences.
- Résistances : ici, ils sont utilisés, en plus de leur fonction principale qui est de diminuer l'intensité d'un courant, afin de réaliser des filtres.

Programme C Arduino

Le programme se découpe en deux parties :

- main
- cardio.cpp / .h

Main gère la détection des battements, là où **cardio.c** gère le calcul du pouls et l'envoie sur la voie série.

Main

```
#include "cardio.h"

long last; // Variable contenant l'instant du battement précécdent
// Variables utilisées par les fonctions du coeur de LEDs
int i;
int n;

void setup() {
  Serial.begin(9600); // Démarrage de la communication série
  last = millis(); // Initialisation de la valeur du dernier battement enregistré à l'instant du démarrage du programme
  for(int j = 2; j < 12; j++){ // Initialisation des pins du coeur
    pinMode(j, OUTPUT);
  }
  i = 0;
  n = 2;
}

void loop() {
  if (analogRead(0) >= 650) { // Chaque battement génère un pic qui dépasse 650
    calc_pouls(millis(), millis() - last, i, n); // On calcule le pouls en faisant la différence entre l'instant du battement avec l'instant de l'ancien battement
    last = millis();
  }
}
```

Le principe ici est d'attendre un battement. Dès que la valeur sur le port analogique (ici A0) dépasse 650 (seuil déterminé par lecture graphique), on considère qu'un battement a été détecté. Ainsi on appelle la fonction *calc_pouls*, qui a partir de la durée entre deux battements (*millis() – last* avec millis() la fonction renvoyant le nombre de millisecondes depuis le début de l'exécution du programme et last l'instant du dernier battement en millisecondes).

Cardio.cpp / .h

```
#include <arduino.h>
#include "cardio.h"
#include "coeur.h"
#include "param.h"

int calc_pouls(long time, long pulse, int& i, int& n) {
  Variables var; // Objet permettant d'extraire les paramètres dans param.h
  int resultat = 60000 / pulse; // Calcul du pouls
  if (resultat > 40 && resultat < 150) { // Un résultat valable n'est compris qu'entre 40 et 150 en moyenne
    // On envoie une chaine, par le port série, qui commence par un S et se termine par un E, ce qui permet de contrôler l'intégrité de l'information reçue par la suite (flags) (e.g "S1000;80E")
    Serial.print("S");
    Serial.print(time);
    Serial.print(";");
    Serial.print(resultat);
    Serial.print("E");
    // En fonction du mode renseigné dans param.h, on appelle la fonction qui contrôle les LEDs adéquate
    switch(var.mode){
      case 1:
      chenille(n);
      break;
     case 2:
      everyLeds();
      break;
     case 3:
      led_1_3(i);
      break;
     case 4:
      led_1_2(i);
      break;
     case 5:
      one_led(var.led);
    }
  }
}
```

Ici, la fonction calc_pouls va s'occuper de calculer le pouls, et l'envoyer vers l'application Processing (*voir module 3*).

Voici les différents paramètres :

- **time** : correspond à l'instant (en millisecondes, depuis le démarrage du programme) où le battement a été enregistré
- **pulse** : correspond au temps entre le battement précédent et celui enregistré
- **i** et **n** : deux variables requises par les fonctions contenues dans cœur.c (*voir module 2*)

Résumons la partie de cette fonction liée au module 1.

Elle calcule le pouls à partir d'un produit en croix entre le temps entre les deux derniers battements et une minute.

On fait donc 60 000 *(à savoir 60 secondes)* / temps entre les deux derniers battements. (e.g 60 000 / 500 = 120 bpm).

Ensuite elle envoie le résultat du calcul par voie série, en suivant le format suivant : ‘S’ + instant + ‘ ;’ + pouls + ‘E’ (e.g S1000;80E)

(On notera que cardio.h contient le prototype de la fonction calc_pouls())

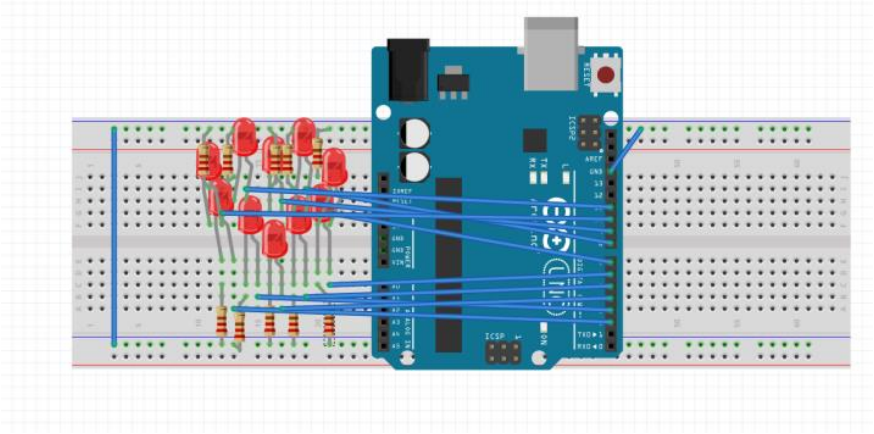
Partie III – Module 2 : le cœur de LEDs

Rappel

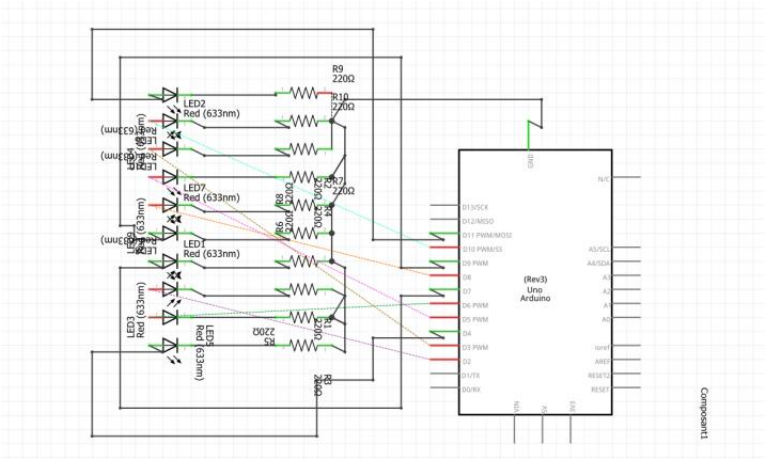
Dans ce module, on cherche à modéliser un cœur avec des LEDs, qui clignotera, selon un schéma définit à l’avance, à chaque battement détecté. Il est également nécessaire de créer une application qui permet de choisir le mode d’affichage désiré.

Schémas

Vue platine



Vue schématique



Montage électronique

Nous disposons de la même Arduino que dans le module 1. Le montage est effectué sur une breadboard différente de la première.

Toutes les LED sont disposées sur la breadboard de manière à former un cœur. Toutes leurs cathodes sont reliées entre elles au Ground de l'Arduino. En revanche, chaque anode est alimentée de manière séparée par les pins de l'Arduino (un par anode). Ainsi, on peut alimenter chaque LED de manière séparée. On notera la présence d'une résistance entre l'anode et le pin de chaque led, afin de ne pas faire surchauffer le composant.

Programme C Arduino

Cœur.c / .h

```
#include "Arduino.h"

void chenille(int& n) { ... }

void everyLeds() { ... }

void led_1_3(int& i) { ... }

void led_1_2(int& i) { ... }

void one_led(int led) { ... }
```

Il réunit toutes les fonctions qui correspondent aux différents affichages possibles.

Chenille(int& n)

Cette première fonction allume les LEDs une par une, c'est-à-dire qu'à chaque battement, une nouvelle LED à la suite s'allume, et l'ancienne s'éteint.

Pour cela, on allume la LED de pin *n*, on éteint la LED de pin *n – 1*, puis on incrémente *n*. Si *n* vaut 2, on éteint la 11ème, si *n* vaut 11, on passe *n* à 2 au lieu de l'incrémenter (car la LED n°12 n'existe pas).

EveryLeds()

Cette fonction allume toutes les LEDs, et les éteint au bout de 0.2 secondes, à l'aide du *delay(200);*

Led_1_3(int& i)

Ce schéma se découpe en trois états possibles :

- Soit les LEDs 2, 5, 8, 11 sont allumées
- Soit les 3, 6, 6
- Soit les 4, 7, 10

En fonction de *i*, qui peut prendre les valeurs 0, 1 et 2, un des trois états se déclenchera. Par la suite, *i* est incrémentée de manière à afficher l'état suivant au prochain battement. On notera que c'est un clignotement, comme dans EveryLeds() qui s'opère, et que si *i* vaut 2, il n'est pas incrémenté mais re-passe à 0.

Led_1_2(int& i)

Même principe que *Led_1_2(int& i)*, avec seulement deux états :

- Soit les LEDs 2, 4, 6, 8, 10 sont allumées
- Soit les 3, 5, 7, 9, 11

One_led(int led)

Ici, une seule LED clignote. Le numéro de la LED est indiqué dans le fichier param.h.

A chaque battement, la LED s'allume, puis s'éteint 0.2 secondes plus tard.

Param.h

```
class Variables{
public:
  int mode = 4;
  int led = 5;
};
```

Ce fichier contient une classe Variables, qui possède en membres *mode*, qui définit la fonction à appeler à chaque battement (chenille, etc…), et *led* qui sert dans le cas où on appelle la fonction *one_led(int led)*.

Pour accéder à ces variables, il suffit de déclarer un objet de type *Variables*.

Application de génération de param.h

Cette application se découpe en trois parties :

- Main.c
- Menu.c / .h
- GenerationCode.c / .h

(La fonction *main* du fichier **Main.c** ne fait qu'appeler la fonction *menu* de **Menu.c**.)

Menu.c

Ce fichier contient uniquement une fonction *menu*, qui affiche un menu proposant à l'utilisateur de sélectionner l'affichage de son choix. En fonction de sa réponse, une des fonctions de **GenerationCode.c** sera appelée.

GenerationCode.c

```
void chenille()
{
    FILE* fichier = NULL;
    fichier = fopen("param.h","w+");
    fprintf(fichier,"class Variables{\npublic:\n\tint mode = 1;\n\tint led = 5;\n};");
    fclose(fichier);
}
```

Voici une fonction type, trouvable dans **GenerationCode.c**.

La fonction crée un fichier **param.h**, puis écrit dedans un code suivant ce schéma :

```
class Variables{
public:
  int mode = 4;
  int led = 5;
};
```

En fonction du mode choisi, la valeur de *mode* (et *led* si la fonction One_Led a été sélectionnée) va être modifiée par celle du mode désiré.

Enfin, le fichier est refermé et l'application se ferme.

Lien entre les modules 1 et 2

Le module 1 et 2 interagissent ensemble. Lorsqu'un battement est détecté par le module 1, les LEDs du module 2 clignotent.

L'interaction se fait dans le fichier **Cardio.h**, et plus précisément dans la fonction *calc_pouls()*.

```
#include <arduino.h>
#include "cardio.h"
#include "coeur.h"
#include "param.h"

int calc_pouls(long time, long pulse, int& i, int& n) {
 Variables var; // Objet permettant d'extraire les paramètres dans param.h
 int resultat = 60000 / pulse; // Calcul du pouls
 if (resultat > 40 && resultat < 150) { // Un résultat valable n'est compris qu'entre 40 et 150 en moyenne
   // On envoie une chaine, par le port série, qui commence par un S et se termine par un E, ce qui permet de contrôler l'intégrité de l'information reçue par la suite (flags) (e.g "S1000;80E")
   Serial.print("S");
   Serial.print(time);
   Serial.print(";");
   Serial.print(resultat);
   Serial.print("E");
   // En fonction du mode renseigné dans param.h, on appelle la fonction qui contrôle les LEDs adéquate
   switch(var.mode){
     case 1:
     chenille(n);
     break;
    case 2:
     everyLeds();
     break;
    case 3:
     led_1_3(i);
     break;
    case 4:
     led_1_2(i);
     break;
    case 5:
     one_led(var.led);
   }
 }
}
```

Un objet de type Variables est instancié en début de fonction. Il suffit ensuite de faire *var.mode* pour récupérer le mode d'affichage (et *var.led* dans le cas de la fonction *one_led()*). Ainsi, le switch sur *var.mode* permet de déterminer quelle fonction appeler dans **Cœur.c**.

On notera les variables *i* et *n*, présentes dans Main.ino, qui gardent en mémoire l'état du mode d'affichage (e.g *i = 5* car la prochaine LED à faire clignoter est la 5ème), d'où le passage par référence dans les paramètres.

Partie IV – Module 3 : application Processing

Rappel

L'application Processing est celle qui va récupérer les données envoyées par l'Arduino sur la voie série.

Programme

Dans la logique, il suffit d'ouvrir un fichier .csv, de lire en continu sur un port série, et d'écrire dans le fichier chaque information qui passe par la voie série.

Le problème est que nous désirons avoir sur chaque ligne de notre fichier .csv une information au format : temps;pouls

Or la voie série n'est pas fiable complètement, et en envoyant une chaine de caractères, la réception peut se faire en plusieurs temps. Il est donc nécessaire de vérifier l'intégrité des informations.

Pour cela, il a fallu instaurer un système de **flags**. Ainsi, en ajoutant un flag au début de l'information et à la fin de l'information, on peut savoir à la réception, par la présence (ou non) des deux flags, si la chaine de caractères est complète.

(e.g : prenons comme flag de départ **S** et flag de fin **E** :

- Si je reçois **S1000;80E**, alors j'en déduis que l'information est complète

- En revanche si je n'ai que **S1000;**, je ne détecte pas le **E** de fin, j'en déduis que l'information n'est pas entière, je dois donc attendre pour la suite de l'information)

Ce procédé se traduit par ce code :

```
//From Arduino to Processing to Txt or cvs etc.
//import
import processing.serial.*;
//declare
PrintWriter output; // Fichier de sortie
Serial udSerial; // Objet permettant la communication série
String SenVal; // Texte lu sur le port série

void setup() {
  udSerial = new Serial(this, Serial.list()[0], 9600); // Ouverture du premier port de la liste des ports série
  output = createWriter ("Battements.csv"); // Création du fichier Battements.csv
  SenVal = ""; // Initialisation du texte
}

void draw() {
  if (udSerial.available() > 0) { // Si le port série est ouvert
    SenVal += udSerial.readString(); // On ajoute au texte lu le texte à lire sur le port série
    if (SenVal.length() >= 5 && SenVal.indexOf('S') != -1 && SenVal.indexOf('E') != -1 && SenVal.substring(SenVal.indexOf('S') + 1, SenVal.indexOf('E')).indexOf('S') == -1) { // Si un S et un E sont détectés
      output.println(SenVal.substring(SenVal.indexOf('S') + 1, SenVal.indexOf('E'))); // Ecriture dans le fichier du bloc contenu entre les deux flags
      SenVal = SenVal.substring(SenVal.indexOf('E') + 1); // Suppression du bloc lu
    }
  }
}

void keyPressed(){ // Fonction qui ferme le fichier ainsi que l'application quand une touche est pressée
  output.flush();
  output.close();
  exit();
}
```

A la fin de l'exécution du programme, l'utilisateur sera en possession d'un fichier **Battements.csv**, présent dans le dossier de l'application Processing.

Partie V – Module 4 : application de traitement des données

Rappel

Cette dernière application se charge d'ouvrir le fichier .csv généré par l'application Processing (*voir module 3*), et de lire toutes les données inscrites, les trier dans l'ordre croissant / décroissant en fonction du temps / pouls, de chercher un pouls en fonction d'un temps, de calculer la moyenne des pouls dans une plage de temps, d'afficher le nombre de battements enregistrés

Programme

Le programme est structuré en trois parties :

- Main.c
- Menu.c / .h
- Actions.c / .h
- Donnees.c / .h (et Variables.h)

Main.c

La fonction *main()* ne fait qu'appeler la fonction *showMenu()* dans **menu.c**.

Menu.c

Ce fichier contient une fonction *showMenu()* , qui appelle *ReadFile()* dans **donnees.c**, puis affiche un menu proposant différentes actions à l'utilisateur, telles qu'afficher toutes les données, les trier, etc...

```c
void ShowMenu(){
    ReadFile(); // Lit le fichier et stocke les différents pouls dans un tableau
    printf("Choisissez une commande parmi :\n- liste\n- croissant [temps/pouls]\n- decroissant [
    char commande[50];
    while(strcmp(commande, "exit") != 0){
        printf("> ");
        gets(commande);
        char **com = split(commande, ' '); // Sépare la commande en plusieurs mots
        // Appel de la bonne fonction en regard du premier mot écrit
        if(strcmp(com[0], "liste") == 0){ // Si la commande est "liste"
            Show(); // On affiche la liste des données
        }
        else if(strcmp(com[0], "croissant") == 0){ // Si la commande est "croissant"
            if(strcmp(com[1], "temps") == 0) // Si le deuxième paramètre est temps
                ShowIncreasing(0); // On trie la liste en fonction du temps
            else if(strcmp(com[1], "pouls") == 0) // Si le deuxième paramètre est pouls
                ShowIncreasing(1); // On trie la liste en fonction du pouls
        }
        else if(strcmp(com[0], "decroissant") == 0){ // Si la commande est "décroissant"
            if(strcmp(com[1], "temps") == 0) // Si le deuxième paramètre est temps
                ShowDecreasing(0); // On trie la liste en fonction du temps
            else if(strcmp(com[1], "pouls") == 0) // Si le deuxième paramètre est pouls
                ShowDecreasing(1); // On trie la liste en fonction du pouls
        }
        else if(strcmp(com[0], "recherche_temps") == 0){ // Si la commande est "recherche_temps"
            printf("%d\n", LookAt(atoi(com[1]))); // On affiche le pouls à l'instant com[1]
        }
        else if(strcmp(com[0], "moyenne_pouls") == 0){ // Si la commande est "moyenne_pouls"
            printf("%d\n", Average(atoi(com[1]), atoi(com[2]))); // On affiche la moyenne entre
        }
        else if(strcmp(com[0], "nombre_lignes") == 0){ // Si la commande est "nombre_lignes"
            printf("%d\n", Count()); // On affiche le nombre de lignes
        }
        else if(strcmp(com[0], "recherche_max") == 0){ // Si la commande est "recherche_max"
            printf("%d\n", Max()); // On affiche le maximum
        }
        else if(strcmp(com[0], "recherche_min") == 0){ // Si la commande est "recherche_min"
            printf("%d\n", Min()); // On affiche le minimum
        }

    }
}
```

Tout d'abord, la fonction *ReadFile()* de **donnees.c** est appelée de sorte à lire le fichier .csv et ajouter toutes les entrées du fichier dans un tableau.

Ensuite, une boucle s'exécute tant que l'utilisateur n'entre pas la commande « exit ».

A l'intérieur de cette boucle, l'utilisateur entre la commande son choix à l'aide de *gets(commande)*, et la variable *commande* est séparée en différents paramètres, à l'aide de la fonction *split()*, qui va séparer la commande à chaque espace et ajouter chaque bloc à un tableau. Par ailleurs, la commande est du format : *commande [paramètre] [paramètre]*

```c
char** split(char* str, char split){
    int count = 1;
    int i = 0;

    while(str[i] != '\0'){ // Compte le nombre de blocs
        if(str[i] == split)
            count++;
        i++;
    }

    int *sizes = (int*)malloc(count * sizeof(int)); // Tableau des tailles des différents blocs

    for(i = 0; i < count; i++){ // Initialisation du tableau
        sizes[i] = 0;
    }

    i = 0;
    int j = 0;

    while(str[i] != '\0'){ // Calcul des différentes tailles des blocs
        if(str[i] != split)
            sizes[j]++;
        else
            j++;
        i++;
    }

    int totalSize = strlen(str) - count; // Taille du texte sans les délimiteurs

    char** strings = (char**)malloc(totalSize * sizeof(char)); // On alloue la mémoire requise pour le tabl

    i = 0;
    j = 0;

    for(i = 0; i < count; i++){ // Initialise le tableau avec des textes vides
        strings[i] = "";
    }

    j = 0;

    for(i = 0; i < totalSize + count; i++){ // Ajout des différents blocs dans le tableau des textes
        if(str[i] != split){
            strings[j] = append(strings[j], str[i]);
        }
        else
        {
            j++;
        }
    }

    return strings; // Retourne le tableau des blocs
}
```

(Pour résumer, cette fonction calcule et stocke dans un tableau les tailles des différents blocs, puis alloue de la mémoire pour stocker les différents blocs, et enfin parcourt la chaine pour ajouter chaque bloc à ce tableau, en changeant d'index à chaque fois que le délimiteur est trouvé. La fonction *append* permet d'assembler deux chaines de caractères ensemble.)

Ensuite, en fonction de la commande (qui est enregistrée dans *com[0]*), la fonction correspondante est appelée (ces fonctions sont contenues dans actions.c).

Donnees.c

```c
struct Pulse{
    int pulse;
    int time;
};
```

(Donnees.h) Voici la structure qui représentera chaque donnée enregistrée dans le fichier .csv. L'instant du battement est renseigné dans *time*, et le pouls à cet instant dans *pulse*.

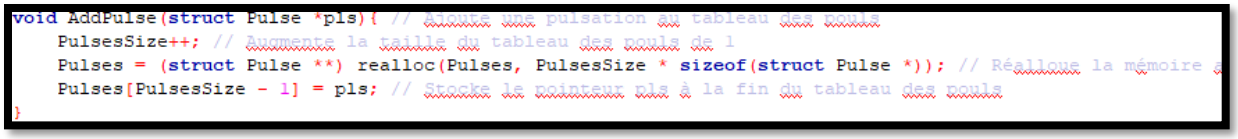
Ce fichier permet la manipulation des données :

```c
void ReadFile(){ // Lit le fichier Battements.csv et stocke les données dans un tableau de structures
    FILE *f = NULL;
    char buffer[100];

    f = fopen("Battements.csv", "r"); // Ouvre le fichier

    if (!f) // Vérifie si le fichier a été ouvert et affiche un message d'erreur sinon
    {
       printf("Le fichier Battements.csv n'a pas pu etre lu.");
       fclose(f);
       exit(1);
    }

    while (fgets(buffer, 100, f) != NULL){ // Parcourt toutes les lignes du fichier et stocke les données d
        struct Pulse *pls = NULL; // Initialise un pointeur vers une structure Pulse
        pls = (struct Pulse *)malloc(sizeof(struct Pulse));
        pls->time = atoi(strtok(buffer, ";")); // Transforme la chaîne de caractères située avant le ";" en
        pls->pulse = atoi(strtok(NULL, ";"));
        AddPulse(pls);
    }

    fclose(f);

}
```

Premièrement, la fonction *ReadFile()* sert à lire le fichier .csv et stocker chaque ligne dans une structure, dont le pointeur va être ajouté dans un tableau. Elle commence par ouvrir le fichier, puis elle va le lire ligne par ligne en stockant la ligne lue dans la variable *buffer*, à l'aide d'une boucle. A chaque tour de boucle, un nouveau pointeur de structure est créé, et de la mémoire est allouée pour stocker la nouvelle structure, pointée par le pointeur créé précédemment. La ligne est ensuite séparée par la fonction *strtok()*, la première partie est stockée dans le champ *temps* de la nouvelle structure, et la seconde dans *pouls*. Finalement la fonction *AddPulse()* est appelée pour ajouter le pointeur de cette nouvelle structure dans un tableau prévu à cet effet.

Enfin, le fichier est fermé.

Voici la fonction *AddPulse()* :



Elle commence par augmenter la taille du tableau de 1.

Puis elle réalloue la mémoire dédiée au tableau avec la nouvelle taille, de façon à ajouter un espace.

Finalement, le pointeur est ajouté dans ce dernier espace alloué.

On notera que **donnees.c** comporte deux autres fonctions, *GetPulses()* qui retourne le tableau des pointeurs de structures, contenu dans un fichier **variables.h**, et *GetPulsesSize()*, qui retourne la taille de ce dernier, contenu dans le fichier **variable.h** également.

Actions.c

Ce fichier contient toutes les fonctions qui effectuent différentes actions sur le tableau des pointeurs de structures.

On retrouve :

- Show() : affiche la liste des données enregistrées. Il s'agit juste d'une boucle qui parcourt tout le tableau et affiche les données à la volée.
- ShowIncreasing(int parameter) : tri et affiche la liste des données enregistrées dans l'ordre croissant en fonction du temps ou du pouls (*parameter*). Si c'est en fonction du temps, alors il suffit d'appeler *Show()*, car de base la liste est triée par ordre croissant du temps. Si c'est en fonction du pouls, alors ici on crée un nouveau tableau de la taille de celui contenant toutes les informations, de manière à trier toutes les données sans altérer le tableau de base.
- ShowDecreasing(int parameter) : tri et affiche la liste des données enregistrées dans l'ordre décroissant en fonction du temps ou du pouls (*parameter*). Si c'est en fonction du temps, alors il suffit de lire le

tableau de données dans le sens inverse. Sinon, c'est le même principe que dans *ShowIncreasing()*.

- LookAt(int time) : cherche et retourne la valeur du pouls à un instant donnée (*time*), en parcourant le tableau des informations.
- Average(int borne_inf, int borne_sup) : retourne la moyenne des valeurs des pouls entre deux bornes. Pour cela, elle cherche l'index de la borne inférieure, puis celui de la borne supérieure, additionne toutes les valeurs contenues entre ces index et divise le tout par le nombre de données.
- Min() : cherche et retourne la valeur minimale du pouls
- Max() : cherche et retourne la valeur maximale du pouls
- Count() : retourne le nombre d'informations enregistrées

Partie VI – Bilan

Analyse des écarts

Le premier problème auquel nous avons fait face est qu'il fallait saisir la structure du projet. En schématisant la situation, nous avons pu avoir une vue globale du projet.

L'autre difficulté majeure était que le montage du cardiofréquencemètre ne fonctionnait pas, ou pas comme espéré, c'est-à-dire que les battements n'étaient pas détectés. Après avoir refait le montage plusieurs fois, il n'a jamais réellement fonctionné à la perfection.

Bilans personnels

Alexi Al Khoury

Ce premier projet est un projet assez simple que ce soit au niveau de la programmation ou des montages, je me suis chargé du module 2 ce module contient une partie sur arduino, une partie sur du code en C et le montage du cœur de LEDs. Lorsque j'avais des difficultés je demandais à mon groupe.

Les différentes parties du projet ont été assemblées en modifiant certaines lignes de code pour la compatibilité.

Louis CHOCHOY

J'ai trouvé le projet simple malgré la perte de temps sur le module 1 avec le problème de détection de pouls (j'ai recommencé le montage 7 fois mais sans résultat probant) car le graphique est composé de plein de parasites et j'ai l'impression que l'infrarouge ne passe pas à travers mes doigts. C'était le seul problème rencontré, à part ça le code était simple. Sinon il n'y a pas eu de problème pour le travail. Tout le monde a bien travaillé. Le contexte du projet m'a bien plu, cela m'a permis de découvrir ce que l'on pouvait faire avec l'électronique.

Charlotte BENARD

J'ai aussi trouvé que le projet était assez facile étant donné que le schéma du montage électronique donné dans le guide du projet fonctionnait, et j'ai trouvé le sujet intéressant. Je me suis occupée du module 4, c'est-à-dire le programme en C qui permet de manipuler les données contenues dans le fichier .csv. Les fonctionnalités demandées n'étaient globalement pas trop

compliquées, puisque la réalisation d'algorithmes de tri et de recherche compliqués n'était pas exigée, par exemple. J'ai eu un peu de mal à faire certaines fonctions du programme, mais j'ai fini par réussir. Le principal problème que j'ai rencontré dans mon programme est qu'il y avait une variable définie dans plusieurs fichiers, mais j'ai résolu le problème avec l'aide de Christophe. Concernant la composition du groupe, j'ai trouvé que nous nous entendions bien et tout le monde s'aidait en cas de difficultés.

Christophe CHICHMANIAN

Comme les membres de mon groupe, j'ai trouvé ce projet assez simple, étant donné que certaines parties étaient déjà faites, ou du moins avancées, dans l'énoncé. J'ai géré le module Processing, et ai vite constaté le problème de la voie série, que j'ai corrigé avec le système de flags. J'ai ensuite pu aider les autres. Le sujet n'était pas spécialement celui que je préférais, mais j'ai quand même aimé mener à bien ce projet. Etre chef d'équipe n'était pas trop compliqué, sachant que le projet était séparé en quatre parties distinctes, soit autant que le nombre de membres. La seule difficulté était d'arriver à aider tout le monde en même temps, tout en me concentrant sur ma partie également.

Montage final