HeXart Care

PROJET FONDAMENTAUX SCIENTIFIQUES

Rapport de projet

Hangard Corentin
Brossier Achille
Lupart Gwendal
Semard Charles

Table des matières

Partie I 2
Introduction 2
Répartition des tâches 3
Rappel de la demande 4
Partie 2 5
Planning 5
Partie 3 6
Module Cardio 6
Code du module Cardio 8
Partie 4 12
Montage du Cœur de LEDs 12
Code du cœur du LEDs 14
Partie 5 21
Module Processing et acquisition de données 21
Partie 6 23
Module lecture et traitement de données 23
Partie 7 34
Analyse des écarts 34
Apports personnels 35
Evolutions possibles 37
Partie 8 38
Annexes 38
Outils utilisés 38

Partie I

Introduction

HeXart Care est une startup spécialisée dans l'électronique et l'informatique. Leur dernier projet, un lecteur portatif grand public de la fréquence cardiaque a disparue à la suite d'un sabotage. Pour donner suite à la demande de **HeXart Care** nous avons été sollicités afin de réaliser ce projet.

Pour réaliser ce projet nous devions créer un cardiofréquencemètre se basant sur la photo pléthysmographie, une méthode qui consiste à détecter les battements du cœur en mesurant le volume de sang dans les tissus au moyen d'une source de lumière et d'un capteur. Nous avons donc réparti la réalisation de ce projet en quatre étapes.

Premièrement, la réalisation du module cardio, c'est-à-dire la partie du montage électronique permettant avec l'utilisation d'une LED infrarouge et un phototransistor (sensible aux infrarouge) de détecter les battements du cœur puis à l'aide d'une carte Arduino de calculer le pouls et de donner un fichier CSV contenant le pouls calculé et le nombre de millisecondes depuis le démarrage de la détection du pouls.

La seconde partie consistait à créer un affichage du pouls à travers un cœur de LEDs rouges. Cela comprenait la réalisation d'un montage électronique comportant un cœur formé de LEDs, des résistances, des transistors et une carte Arduino qui par la suite devra être configuré afin d'allumer les LEDs selon un pattern défini par l'utilisateur avec l'utilisation d'un code en C et en C Arduino.

La troisième partie consistait à récupérer toutes les valeurs lues sur la sortie de l'Arduino et de les enregistrer dans un fichier CSV afin de les utiliser par la suite.

La dernière partie fut la création d'un programme en C pouvant charger en mémoire le fichier de la partie précédente et de traiter ces données afin d'exécuter les fonctionnalités suivantes :

- Afficher les données dans l'ordre du fichier .csv
- Afficher les données en ordre croissant/décroissant (selon le temps, selon le pouls)
- Rechercher et afficher les données pour un temps particulier
- Afficher la moyenne de pouls dans une plage de temps donnée
- Afficher le nombre de ligne de données actuellement en mémoire
- Rechercher et afficher les max/min de pouls (avec le temps associé)
- Quitter l'application.

Répartition des tâches

Pour la répartition des tâches, nous avons d'abord regardé la quantité de travail que chaque module impliquait. Pour ce premier projet nous avons décidé de se répartir à deux sur chaque module. De ce fait

Achille et Corentin étaient responsable du module 2 cœur de LED c'est-à-dire la réalisation d'un montage électronique servant à représenter les battements du cœur et à coder la carte Arduino et un programme C afin de gérer cet affichage. Ils étaient aussi responsables du module 3 processing et acquisition de données c'est-à-dire de la liaison permettant de récupérer les valeurs du pouls lues sur la sortie de l'Arduino et de les enregistrer dans un fichier .csv pouvant être lu par le programme du module lecture et traitement de données.

Gwendal et Charles étaient responsable du module 1 cardio, c'est-à-dire la création d'un montage électronique pouvant, à l'aide d'une LED infrarouge et d'un phototransistor de récupérer et calculer le pouls d'une à l'aide d'une carte Arduino. Ils étaient aussi responsables du module 4 lecture et traitement de données, un programme en C pouvant stocker les informations du fichier csv et pouvant proposer une interface basique permettant d'utiliser les informations du fichier csv.

Cependant, au fur et à mesure de la semaine cette répartition des tâches à changer. En effet on a constaté que le module 4 demandait plus d'effort que le module 3 et que pour notre groupe la réalisation du montage du module 2 était plus rapide que celui du module 1.

Rappel de la demande

Pour ce projet, nous devions créer deux montage électronique relié à une carte Arduino afin de calculer et afficher le pouls. Le tout gérer par des programmes en C et en C Arduino.

Le code en C a pour but de proposer un menu pouvant afficher le pouls en fonction de plusieurs variable (le temps, le pouls max et min …).

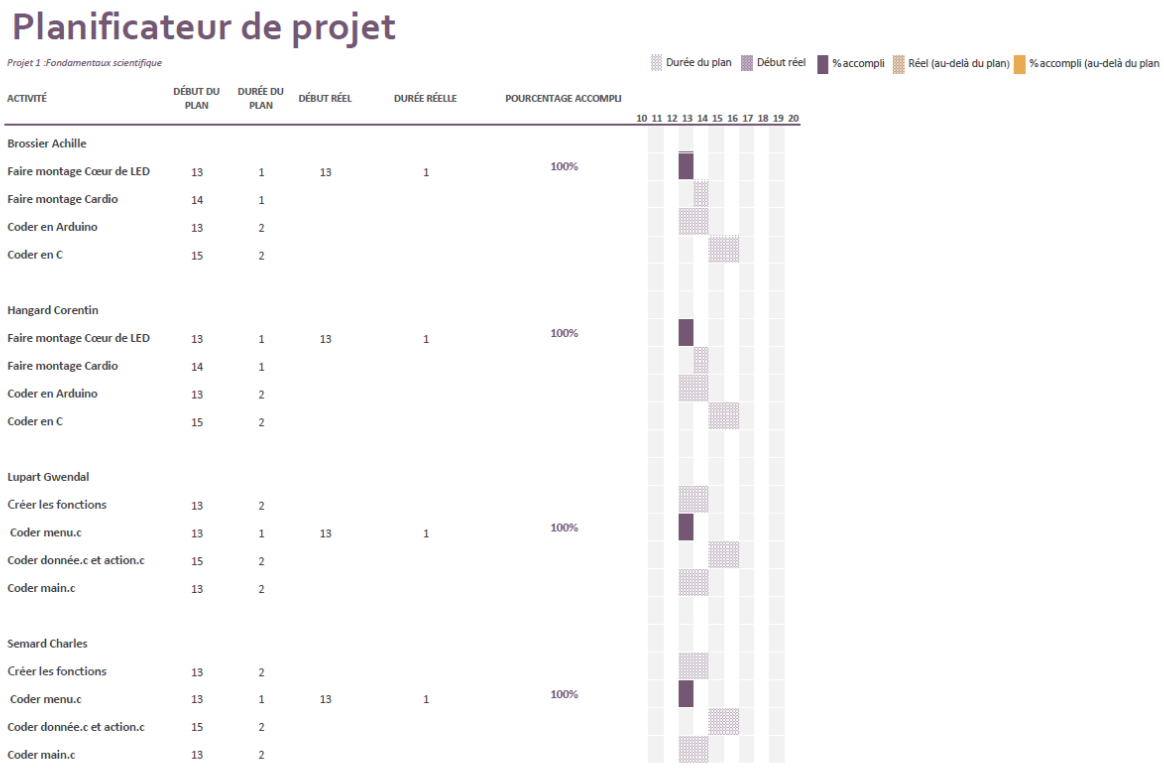
Le code C Arduino a pour but de pouvoir calculer le pouls et de changer l'affichage des LEDs.

Tout ça va nous donner à la fin un cardio fréquencemètre fonctionnel.

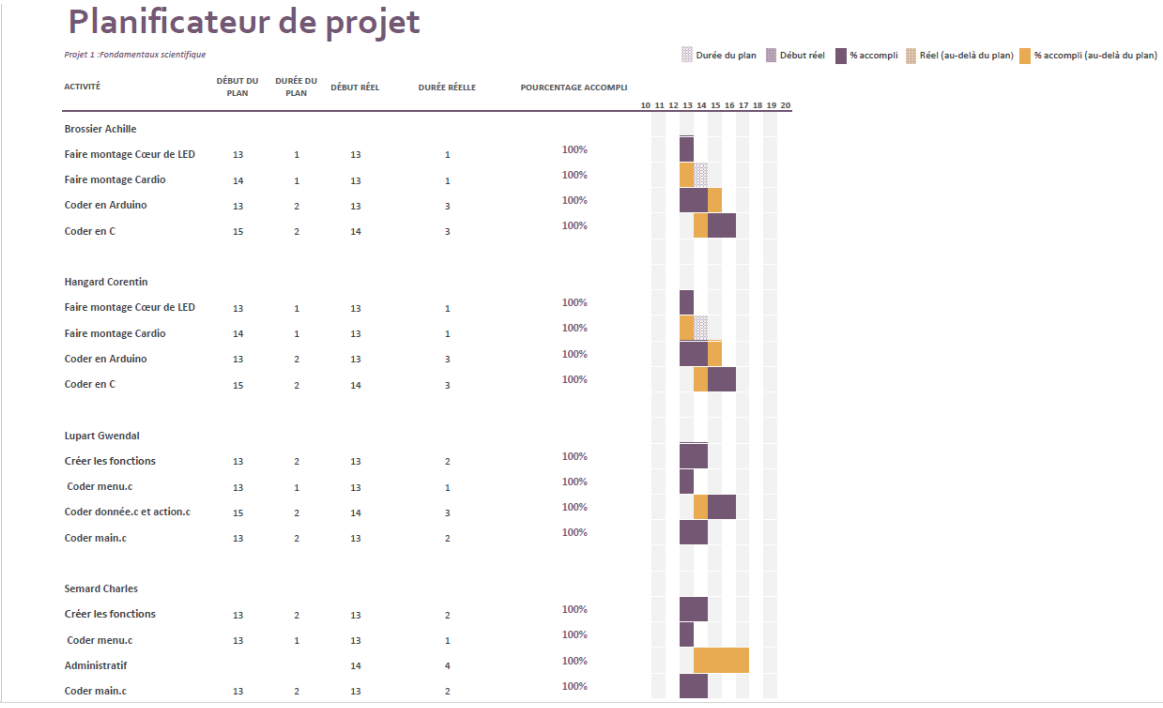
Partie 2

Planning

Voici le planning prévisionnel de la semaine



Cependant cette planification n'a pas été respecter au cours de la semaine



Partie 3

Module Cardio

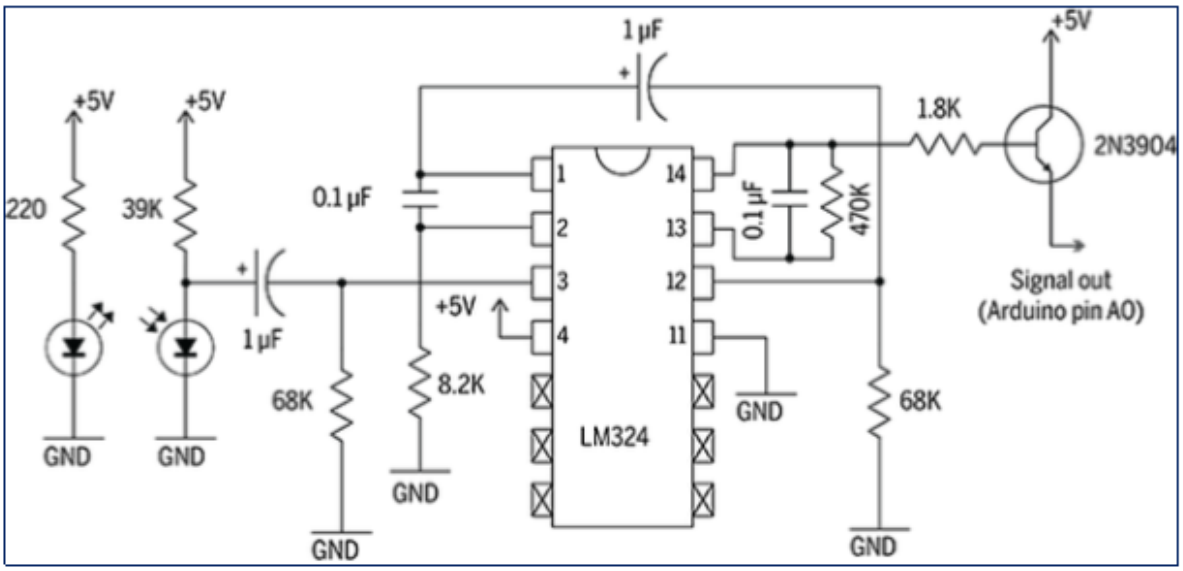


Schéma électronique du circuit "Cardio-fréquencemètre" - Version 2

Sur ce schéma on peut voir plusieurs composants différents :

On a une LED infrarouge envoyant des signaux étant captés par un phototransistor.

Ensuite nous avons de nombreuses résistances de capacités différentes ainsi que des condensateurs. Un transistor NPN et un amplificateur opérationnel.

On peut remarquer deux types de condensateurs ; deux condensateur électronique normaux et deux condensateurs électrolytiques permettant une grande capacité de stockage. Ceux-ci permettent de stocker de l'énergie et la redistribuer dans le circuit cela peut augmenter la tension.

Chacun des composant du circuit est alimenté par l'Arduino il a donc fallu alimenter toute la platine avec la sortie 5V de l'Arduino et donc aussi relier le Ground (GND) de l'Arduino.

On peut remarquer l'association de résistances et condensateurs qui forment des filtres passe haut et passe bas, qui réunis ensemble forment un filtre passe bande permettant de filtrer les fréquences pouvant parasiter les signaux reçus par le phototransistor.

L'amplificateur opérationnel permet lui d'amplifier la différence de potentiel du circuit afin de contrôle les signaux envoyés sur la pin A0 de l'Arduino.

La difficulté de ce schéma consiste à sa compréhension, on a donc réalisé le schéma et testé celui-ci.

Code du module Cardio

Dans la deuxième partie, le but était de renvoyé sur le port série deux valeurs :

- Le nombre de millisecondes depuis le démarrage de l'Arduino,
- Le pouls calculé à l'aide des informations récoltées par le récepteur IR.

Pour arriver à ce résultat, nous avons créé les fichiers main.ino, cardio.ino et cardio.h.

Le fichier main.ino possède une fonction setup (préparation pour la fonction loop) et donc la fonction loop ou se situe l'appelle des fonctions situées dans cardio.ino.

Le fichier cardio.ino possède deux fonctions une fonction **temps** qui permettra de calculer le nombre de millisecondes depuis le démarrage de l'Arduino et la fonction pouls qui permet de calculer le pouls en fonction des informations reçus par le récepteur IR.

Le fichier cardio.h possède les prototypes de fonctions du fichier cardio.ino.

Tout d'abord, nous allons parler du main.ino. Lors de la réalisation du projet, nous avons commencé par le module 3.2, donc nous avons lié le fichier main avec le fichier "cœur.ino" c'est pour cela qu'il n'y a pas de fichier main.ino dans le dossier. On peut cependant expliquer ce que fait normalement le main de cette partie :

```
#include "cardio.h"

int tpsDer;

void setup()
{
  tpsDer = millis();
  Serial.begin(9600);
}

void loop()
{
  while (millis() < 100000)
  {
    if (analogRead(A0) > 640)
    {
      Serial.print("D");
      Serial.print(temps());
      Serial.print(";");
      Serial.print(pouls(tpsDer));
      Serial.println("F");
      tpsDer = millis();

    }
  }
}
```

Dans un premier temps, on inclut la bibliothèque cardio.h dans laquelle sont définie les prototypes des fonctions temps et pouls. Ensuite on créer une variable globale tpsDer qui correspond au dernier temps calculé du pouls.

Après, la fonction setup initialise cette variable avec le nombres de millisecondes depuis le démarrage de l'Arduino mais aussi initialiser le port série ou seront placé les informations à 9600 bits/s.

Et pour finir, la fonction loop, créer une boucle de 100 seconde. Dans cette boucle, si la valeur lue par le récepteur IR est supérieure à 640 (valeur définis grâce au traceur série, c'est-à-dire qu'à chaque fois qu'il y a un pouls, on peut voir un pic dépassent 640) alors on affiche dans le port série "Dvaleurdetemps;valeurdepoulsF" et ensuite on donne à la variable globale tpsDer la valeur du nombres de millisecondes depuis le démarrage de l'Arduino après l'affichage du pouls (On affiche avant et après la lettre D et la lettre F pour Début et Fin, on utilise ses lettres pour définir le début du résultat et la fin).

Ensuite, nous allons parler du cardio.h. Cette fonction a pour seul but de définir les prototypes des fonctions temps et pouls.

```
int temps();
int pouls(int tpsDer);
```

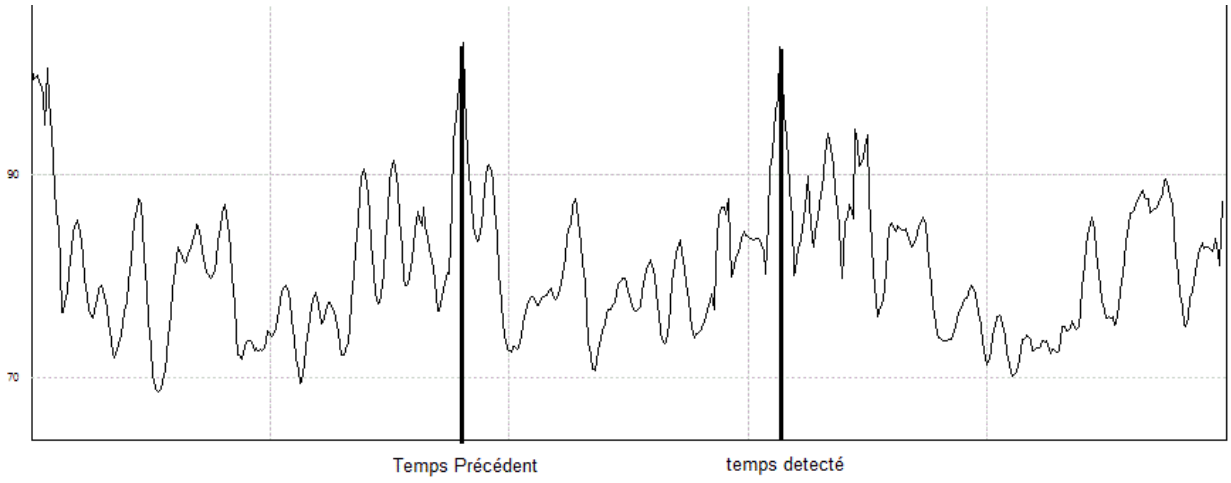
Pour finir, nous allons parler du cardio.ino. Ce fichier possède deux fonctions qui font être appelé dans le main.

La première fonction est la fonction temps. La fonction temps doit renvoyer une valeur correspondant au nombre de millisecondes depuis le démarrage de l'Arduino. Il existe une fonction qui s'appelé millis(), cette fonction a pour but de renvoyé le nombre de millisecondes depuis le démarrage de l'Arduino. On peut donc simplement renvoyer à notre fonction temps la valeur de millis() :

```
int temps()

{
  int tps = millis();

  return tps;

}
```

La deuxième fonction est la fonction pouls. Cette fonction a pour but de calculer le pouls. Pour cela, nous avons déjà vue que la variables tpsDer avait la valeur du dernier temps calculé du pouls. Or pour calculer le pouls en battement par minute, il faut faire l'équation suivante :

Pouls = 1 minute / (Temps Détecté – Temps Précèdent)



Pour faire ce calcul, on initiales une variable pls (pouls) qui possède cette équation.

Cependant il est possible que des pouls proviennent d'un disfonctionnement du phototransistor donc on initialise une boucle Si alors tel que le pouls doit être compris entre 50 et 120 pour que le résultat puisse être affiché.

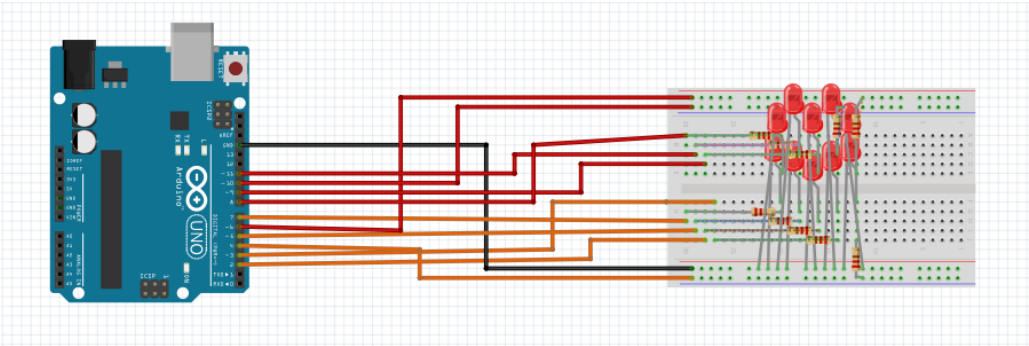
```
int pouls(int tpsDer)
{
  int pls;
  pls = ( (1000.0 * 60.0) / (millis() - tpsDer));
  if (pls > 50 && pls < 120)
  {
  return pls;

  }
}
```

Partie 4

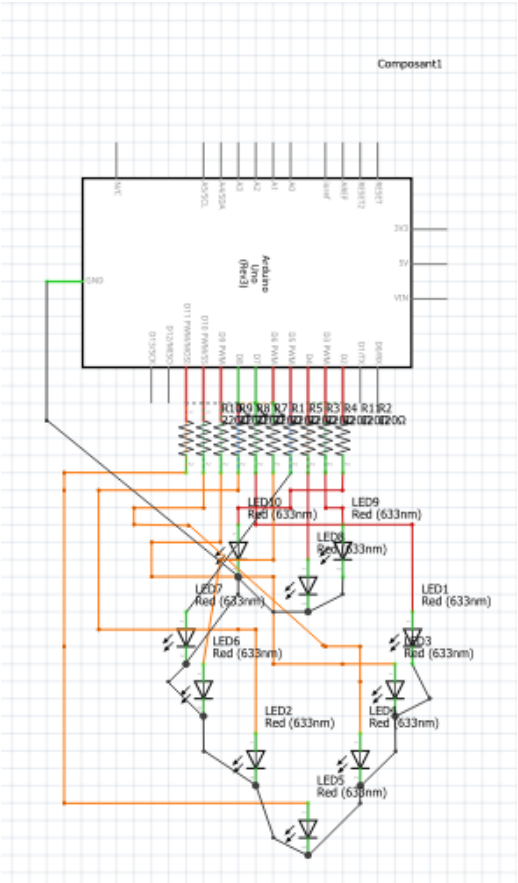
Montage du Cœur de LEDs

L'objectif était de réaliser un cœur de LEDS rouges qui s'allume à chaque battement de cœur détecté. De plus chaque LEDS devait être contrôlé de manière unitaire par l'Arduino. Pour le réaliser on dispose de transistors, de 10 LEDS rouges et d'un jeu de résistance 220 ohm.

Premièrement il a fallu réaliser un montage sur Fritzing. Le voici :



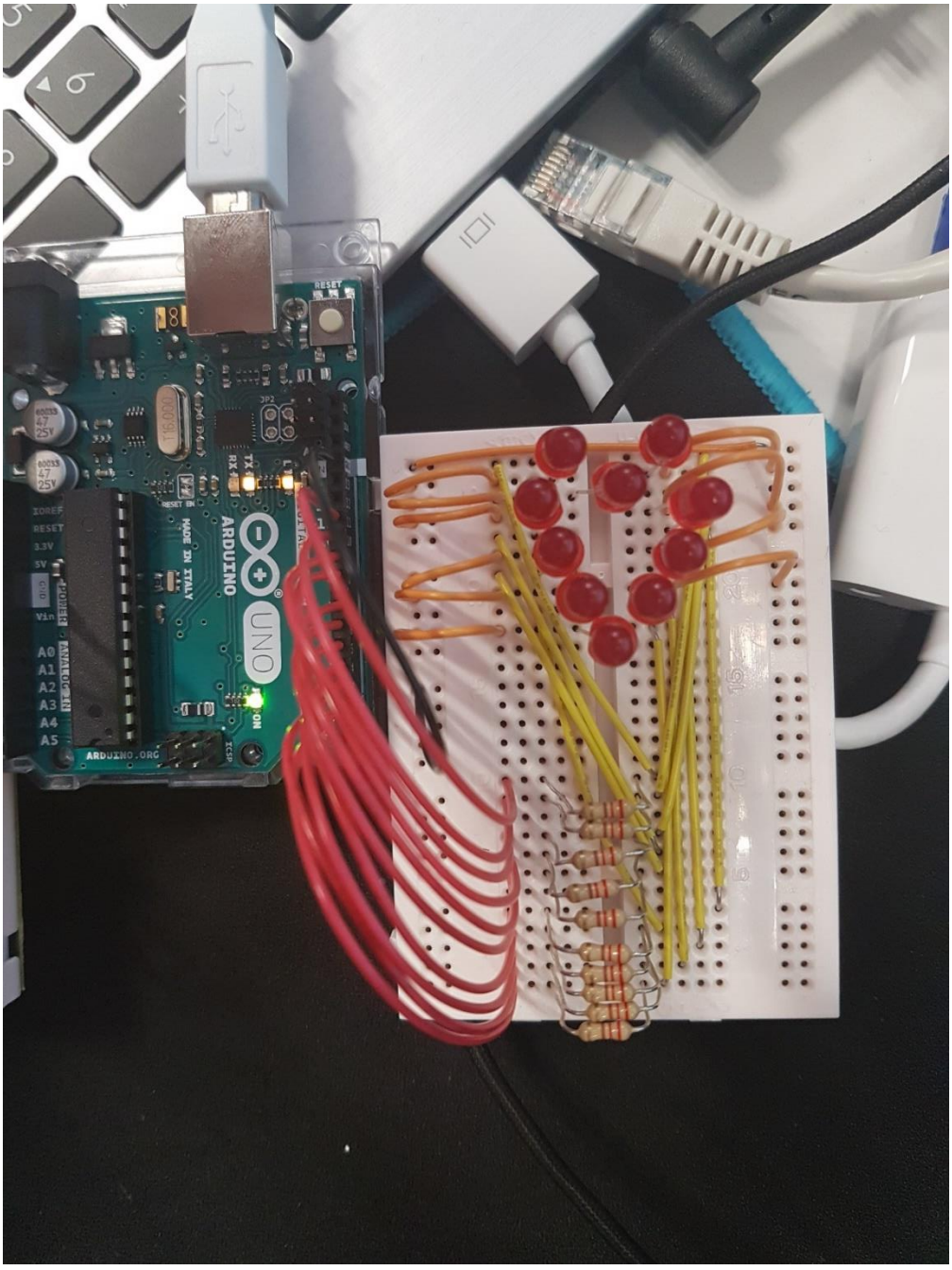
Ensuite la vue schématique :



CESI exia
ÉCOLE D'INGÉNIEURS INFORMATIQUE

On a branché toutes les LEDS a des sorties DIGITALES de l'Arduino pour pouvoir contrôler les LEDS de manière unitaire. On a donc utilisé les résistances de 220 ohm pour régulariser la tension aux bornes des LEDS. Ces résistances sont branchées aux sorties digitales de l'Arduino qui alimentent le circuit. Enfin on relie ces mêmes résistances aux LEDS. La difficulté de ce circuit de situe dans la réalisation de la forme de cœur avec les LEDS et dès les relier aux bonnes bornes de l'Arduino.

Voici le montage final :



Code du cœur du LEDs

La deuxième partie consiste un code en C Arduino permettant de gérer les affichages de LEDs. Pour cela il nous a fallu créer trois fichiers :

- Le fichier cœur.ino (le main),
- Le fichier cœur.h,
- Et le fichier param.h.

Tout d'abord le fichier cœur.ino représente le main. Dans ce fichier, il y'a la fonction setup qui permet d'initialiser nos éléments et la fonction loop qui effectue notre programme. Tout d'abord il nous faut initialiser toutes nos sorties de l'Arduino en mode sortie et n'envoyé aucun courant. Pour cela on créer une boucle permettant de faire ceci.

```
void setup()
{
  for (int i = 0; i < 10; i++)
  {
    pinMode(pinLed[i], OUTPUT);
    digitalWrite(pinLed[i], LOW);
  }
}
```

Ensuite dans la fonction loop permet de faire un boucle infini. Il nous suffit donc de rentrer nos fonction situé dans param.h pour faire l'affichage des LEDs :

```
void loop()
{
      Complet();
      chenille();
      UnDeux();
      UnTrois();
      delay(450);
}
```

Le délai à la fin du loop permet de bloquer la boucle pendant un temps déterminé qui est ici 450ms ce qui peut ressembler à un intervalle entre chaque battement de cœur.

Ensuite le fichier cœur.h possède seulement l'initialisations des LEDs en fonction du port digital sur lesquelles elles sont relier :

```
int  pinLed[10] = {9, 10, 6, 4, 3, 2, 5, 7, 8, 11};
```

On a créé un tableau contenant ses information pour par la suis pouvoir créer des boucles et non devoirs faire appel à chaque LEDs ce qui rallongerai le programme pour rien.

Pour finir, le fichier param.h possède les fonctions permettant d'afficher les LEDs avec différents motifs.

1/Le motif complet :

Le motif complet permet à chaque battement de cœur détecté d'allumé toutes les LEDs pendant un court instant puis toutes les éteindre.

```
void Complet()
{
     int  pinLed[10] = {9, 10, 6, 4, 3, 2, 5, 7, 8, 11};

        digitalWrite(pinLed[1], HIGH);
        digitalWrite(pinLed[2], HIGH);
        digitalWrite(pinLed[3], HIGH);
        digitalWrite(pinLed[4], HIGH);
        digitalWrite(pinLed[5], HIGH);
        digitalWrite(pinLed[6], HIGH);
        digitalWrite(pinLed[7], HIGH);
        digitalWrite(pinLed[8], HIGH);
        digitalWrite(pinLed[9], HIGH);
        digitalWrite(pinLed[0], HIGH);
        delay(100);
        digitalWrite(pinLed[1], LOW);
        digitalWrite(pinLed[2], LOW);
        digitalWrite(pinLed[3], LOW);
        digitalWrite(pinLed[4], LOW);
        digitalWrite(pinLed[5], LOW);
        digitalWrite(pinLed[6], LOW);
        digitalWrite(pinLed[7], LOW);
        digitalWrite(pinLed[8], LOW);
        digitalWrite(pinLed[9], LOW);
        digitalWrite(pinLed[0], LOW);
        delay(100);
}
```

La fonction digitalWrite permet de changer l'états d'une pine. C'est pour cela que nous passons toutes les pines a HIGH ce qui veut dire que l'on envoie du courant dans toutes les pines puis au bout de 100 ms on refait le même processus mais en mettant LOW ce qui enlève le courant des pines et donc étaient toutes les LEDs.

2/Le motif Un sur Deux :

Ce motif permet d'allumer une LED sur deux à chaque battement de cœur.

```
void UnDeux()
{
     int  pinLed[10] = {9, 10, 6, 4, 3, 2, 5, 7, 8, 11};
        for (int i = 0; i < 10; i=i+2)
        {
            digitalWrite(pinLed[i], HIGH);
            delay(50);
            digitalWrite(pinLed[i], LOW);
        }
}
```

Dans cette fonction on utilise une boucle pour. Cette boucle a pour paramètre i allant de 0 à 9 et de passer de i a i+2 à chaque tour de la boucle (c'est ce qui donne l'allumage d'une LED une sur deux). Lors de cette boucle, on envoie du courant dans la LED i puis on l'éteint au bout de 100ms. Ensuite on fait le même processus pour i+2, … jusqu'à la fin de la boucle.

3/Le motif Un sur Trois :

Ce motif est exactement le même que le motif Un sur Deux seulement on change le paramètre de la boucle Pour avec i=i+3 pour sauter de trois en trois.

```
void UnTrois()
{
     int  pinLed[10] = {9, 10, 6, 4, 3, 2, 5, 7, 8, 11};
        for (int i = 0; i < 10; i=i+3)
        {
            digitalWrite(pinLed[i], HIGH);
            delay(50);
            digitalWrite(pinLed[i], LOW);
        }
}
```

4/Le motif en chenille :

Ce motif permet d'allumer les LEDs en formes de chenille. C'est-à-dire une LED qui s'allume puis la précédente d'éteint, …

```
void chenille()
{
    int i = 0;
    int  pinLed[10] = {9, 10, 6, 4, 3, 2, 5, 7, 8, 11};

     for (int i = 1; i < 10; i++)
        {
            digitalWrite(pinLed[i-1], HIGH);
            delay(20);
            digitalWrite(pinLed[i], HIGH);
            delay(20);
            digitalWrite(pinLed[i-1], LOW);
         }
           digitalWrite(pinLed[9], LOW);
}
```

On utilise aussi dans cette fonction une boucle Pour avec comme paramètre i allant de 0 à 9 et à chaque tour, i prend +1. Dans cette boucle, on allume la LED précédente, au bout de 20ms, on allume la LED en question, au bout de 20ms, on éteint la LED précédente et ce jusqu'à la fin de la boucle cependant lorsque la boucle est finie, il reste une LED allumée donc on éteint cette LED à la fin de la boucle et non dans la boucle car cela ne donnera pas l'effet de chenille.

5/Le motif Random :

Ce motif permet d'allumer une LED aléatoire à chaque pouls détecté.

```
void Random()
{
    int  pinLed[10] = {9, 10, 6, 4, 3, 2, 5, 7, 8, 11};
    int i = random(11);
    digitalWrite(pinLed[i], HIGH);
    delay(100);
    digitalWrite(pinLed[i], LOW);
}
```

On initialise un nombre random allant de 0 à 9 pour ensuite allumer et éteindre au bout de 100ms cette LED.

6/Le motif Choix :

Ce motif consiste à faire choisir l'utilisateur, sur la LED dont il veut l'allumage lors de la détection du pouls :

```
void Choix()
{
    int  pinLed[10] = {9, 10, 6, 4, 3, 2, 5, 7, 8, 11};
    int choix = 0;
    digitalWrite(pinLed[choix], HIGH);
    delay(100);
    digitalWrite(pinLed[choix], LOW);
}
```

L'initialisation de la variable choix dépend de l'utilisateur. Cette variable est changée lors de la généralisation du fichier param.h en fonction du choix de l'utilisateur.

La troisième et dernière partie consiste à créer un programme en C pouvant gérer les différents param.h. Pour cela on a créé un fichier main, un fichier menu, un fichier générationcode et les fichiers d'entête des fichiers menu et générationcode.

Tout d'abord le main ne contient que l'appel de la fonction menu.

```
#include <stdio.h>
#include <stdlib.h>
#include "menu.h"

int main()
{
    menu();
    return 0;
}
```

Ensuite, le fichier menu.h et generationcode.h ne possède que les prototypes de fonctions dans les fichiers menu.c et generationcode.c.

Pour le fichier menu.h :

```
#include <stdio.h>
#include <stdlib.h>

void menu();
```

Et pour le fichier generationcode.h :

```c
#include <stdio.h>
#include <stdlib.h>

void Complet();
void chenille();
void UnDeux();
void UnTrois();
```

Après, le fichier menu.c possède la fonction menu qui permet d'afficher dans la console quand on lance le programme un menu et choisir quel type d'affichage on souhaite :

```c
#include "menu.h"

void menu()
{
    int Aff;

    printf("===========MENU===========\n\n");
    printf("Quelle mode d'affichage voulez vous utiliser ?\n");
    printf("1 - Toutes les LEDs\n");
    printf("2 - Une LED sur Deux\n");
    printf("3 - Une LED sur trois\n");
    printf("4 - La chenille\n");
    printf("0 - Quitter\n");
    scanf("%d", &Aff);

    switch (Aff)
    {
    case 0:
        printf("Je quitte\n");
        break;

    case 1:
        Complet();
        system("start C:\\Users\\coren\\Documents\\CESI\\Fondamentauscientifique\\Projet\\coeur\\coeur.ino");
        break;

    case 2:
        UnDeux();
        system("start C:\\Users\\coren\\Documents\\CESI\\Fondamentauscientifique\\Projet\\coeur\\coeur.ino");
        break;

    case 3:
        UnTrois();
        system("start C:\\Users\\coren\\Documents\\CESI\\Fondamentauscientifique\\Projet\\coeur\\coeur.ino");
        break;

    case 4:
        chenille();
        system("start C:\\Users\\coren\\Documents\\CESI\\Fondamentauscientifique\\Projet\\coeur\\coeur.ino");
        break;
    }
}
```

On affiche d'abord dans la console le menu. Ensuite on demande quelle option on veut choisir et cette réponse sera stocker dans la variables Aff préalablement définis. Ensuite on utilise un switch qui permet de passer d'une case à l'autre en fonction du résultat marqué dans la console. Si l'utilisateur marque le chiffre 1 qui correspond à Toutes les LEDs allumé alors le switch pointe vers la case 1. La Case 1 a pour but d'exécuter la fonction Complet situé dans le fichier generationcode.c puis d'ouvrir le programme Arduino. Et ce pour toute les cases.

Pour finir, le fichier generationcode.c possède 4 fonctions (Complet, UnDeux, UnTrois, chenille). Dans ses 4 fonctions le principe est le même, on ouvre où créer le fichier pharm. s'il n'est pas créer et on met à l'intérieur le code correspondant au bon param.h vue dans la partie 2. C'est-à-dire que l'on fait un copié collé des codes.

```
#include "generationCode.h"

void Complet()
{
    FILE*f = NULL;

    f = fopen("C:\\Users\\coren\\Documents\\CESI\\Fondamentauscientifique\\Projet\\coeur\\param.h","w");

    fprintf(f,"void Complet()                                   //Fonction d'affichage avec toutes les LEDs allumé\n{\n");
    fprintf(f,"     int  pinLed[10] = {9, 10, 6, 4, 3, 2, 5, 7, 8, 11};    //Initialisation des LEDs en fonction des ports digitals\n\n");
    fprintf(f,"        digitalWrite(pinLed[1], HIGH);                  //Allumage de toutes les LEDS\n        digitalWrite(pinLed[2], HIGH);\n");
    fprintf(f,"        digitalWrite(pinLed[3], HIGH);\n        digitalWrite(pinLed[4], HIGH);\n        digitalWrite(pinLed[5], HIGH);\n        digitalWrite(pinLed[6], HIGH);\n");
    fprintf(f,"        digitalWrite(pinLed[7], HIGH);\n        digitalWrite(pinLed[8], HIGH);\n        digitalWrite(pinLed[9], HIGH);\n        digitalWrite(pinLed[0], HIGH);\n");
    fprintf(f,"        delay(100);                                 //Au bout de 100 ms,\n        digitalWrite(pinLed[1], LOW);                       //Eteindre toutes les LEDs\n");
    fprintf(f,"        digitalWrite(pinLed[2], LOW);\n        digitalWrite(pinLed[3], LOW);\n        digitalWrite(pinLed[4], LOW);\n");
    fprintf(f,"        digitalWrite(pinLed[5], LOW);\n        digitalWrite(pinLed[6], LOW);\n        digitalWrite(pinLed[7], LOW);\n        digitalWrite(pinLed[8], LOW);\n");
    fprintf(f,"        digitalWrite(pinLed[9], LOW);\n        digitalWrite(pinLed[0], LOW);\n        delay(100);\n}\nvoid chenille(){}\nvoid UnDeux(){}\nvoid UnTrois(){}");
}


void chenille()
{
    FILE*f = NULL;

    f = fopen("C:\\Users\\coren\\Documents\\CESI\\Fondamentauscientifique\\Projet\\coeur\\param.h","w");

    fprintf(f,"void chenille()                                  //Fonction chenille qui permet\n{\n    int i = 0;                                      //On initialise une variable pour la boucle for\n");
    fprintf(f,"    int  pinLed[10] = {9, 10, 6, 4, 3, 2, 5, 7, 8, 11};   //Initialisation des LEDs en fonction des ports digitals\n\n");
    fprintf(f,"     for (int i = 1; i < 10; i++)                //Boucle allant de 1 à 10\n        {\n          digitalWrite(pinLed[i-1], HIGH);            //Allumage de la LED -1\n");
    fprintf(f,"            delay(20);                           //Au bout de 20 ms\n          digitalWrite(pinLed[i], HIGH);              //Allumage de la LED 1\n");
    fprintf(f,"            delay(20);                           //Au bout de 20 ms\n");
    fprintf(f,"            digitalWrite(pinLed[i-1], LOW);          //On eteint la LED -1\n        }\n          digitalWrite(pinLed[9], LOW);               //On eteint la derniere LED pas eteinte\n)");
    fprintf(f,"\nvoid Complet(){}\nvoid UnDeux(){}\nvoid UnTrois(){}");
}


void UnDeux()
{
    FILE*f = NULL;

    f = fopen("C:\\Users\\coren\\Documents\\CESI\\Fondamentauscientifique\\Projet\\coeur\\param.h","w");

    fprintf(f,"void UnDeux()                                    //Fonction UnDeux, affiche une LED sur deux\n{\n     int  pinLed[10] = {9, 10, 6, 4, 3, 2, 5, 7, 8, 11};");
    fprintf(f,"    //Initialisation des LEDs en fonction des ports digitals\n");
    fprintf(f,"        for (int i = 0; i < 10; i=i+2)          //Boucal allant de 1 a 10 mais allant de 2 en 2\n        {\n            digitalWrite(pinLed[i], HIGH);");
    fprintf(f,"                //On allume la LED pointé par i\n");
    fprintf(f,"            delay(50);                          //Au bout de 50ms\n");
    fprintf(f,"            digitalWrite(pinLed[i], LOW);           //On eteint la LED\n        }\n}\nvoid chenille(){}\nvoid Complet(){}\nvoid UnTrois(){}");
}


void UnTrois()
{
    FILE*f = NULL;

    f = fopen("C:\\Users\\coren\\Documents\\CESI\\Fondamentauscientifique\\Projet\\coeur\\param.h","w");

    fprintf(f,"void UnTrois()                                    //Fonction UnDeux, affiche une LED sur deux\n{\n     int  pinLed[10] = {9, 10, 6, 4, 3, 2, 5, 7, 8, 11};");
    fprintf(f,"    //Initialisation des LEDs en fonction des ports digitals\n");
    fprintf(f,"        for (int i = 0; i < 10; i=i+3)          //Boucal allant de 1 a 10 mais allant de 2 en 2\n        {\n            digitalWrite(pinLed[i], HIGH);");
    fprintf(f,"                //On allume la LED pointé par i\n");
    fprintf(f,"            delay(50);                          //Au bout de 50ms\n");
    fprintf(f,"            digitalWrite(pinLed[i], LOW);           //On eteint la LED\n        }\n}\nvoid chenille(){}\nvoid Complet(){}\nvoid UnTrois(){}");

}
```

Partie 5

Module Processing et acquisition de données

Dans ce module, le but est de récupéré les valeurs situées dans le port série déjà indiqué grâce à l'Arduino puis de les afficher dans un fichier .csv à l'aide du logiciel processing.

Tout d'abord nous allons expliquer le code :

```
import processing.serial.*;
```

Cette ligne de code permet d'initialiser le port série.

```
PrintWriter output;
Serial udSerial;
String SenVal ;
```

Ensuite, la fonction PrintWriter permet d'initialiser une variable, ici output, qui permettra d'écrire des caractères. La deuxième ligne de code permet de placer le port série dans la variable udSerial. Et pour finir, on initialise une variable SenVal de type String.

Ensuite, dans la fonction setup, on initialise le port série et on ouvre/créer le fichier battements.csv :

```
void setup() {
  udSerial = new Serial(this, Serial.list()[0], 9600);
  output = createWriter ("Battements.csv");
}
```

Après, la fonction draw permet de lire dans le port série la valeur de D à F et à l'intérieur de ce D et F se trouve les informations temps ; pouls que l'on affiche ensuite dans le fichier .csv.

```
void draw() {
  if (udSerial.available() > 0) {
    SenVal += udSerial.readString();
    if (SenVal.indexOf('D') != -1 && SenVal.indexOf('F') != -1 && SenVal.substring(SenVal.indexOf('D') + 1, SenVal.indexOf('F')).indexOf('D') == -1) {
      output.println(SenVal.substring(SenVal.indexOf('D') + 1, SenVal.indexOf('F')));
      SenVal = SenVal.substring(SenVal.indexOf('F') + 1);
    }
  }
}
```

On utilise une lettre pour le Début et la Fin car si le programme rencontre un problème dans la lecture des informations, le programme sous processing va lire seulement les valeurs entre D et F et non toutes les valeurs qui peuvent venir d'un parasite.

Pour finir, la fonction keyPressed permet de fermer tous les fichiers et de sortir du programme en appuyant sur une touche du clavier :

```
void keyPressed(){
  output.flush();
  output.close();
  exit();
}
```

Partie 6

Module lecture et traitement de données

Ce module s'articule en un projet contenant 7 fichiers. Nous avons respectivement donnees.c, donnees.h, menu.c, menu.h, actions.c, actions.h et enfin le main.c. Dans le fichier donnees.c, nous ouvrons le fichier .csv sortant du module 3.

```c
int lireFichier(struct stockerInformation *tab){
FILE *f = NULL;
char buffer[TAILLE_MAX_STRING]; //initialisation du buffer
int i=0;
int pls = 1; // variable concernant le pouls
int tmps = 2; // variable concernant le temps

f = fopen("./Randgen.csv", "r"); //on ouvre le fichier

if (f == NULL){
    printf("Le fichier n'a pas pu etre ouvert.");
    exit(1); //on ferme le programme si on ne peut pas ouvrir le fichier
}
else {
    printf("Le fichier est ouvert.\n\n");
}

while (fgets(buffer, TAILLE_MAX_STRING, f) != NULL){
    sscanf(buffer, "%d;%d", &pls, &tmps);
    tab[i].pouls = pls; // on stock la valeur pls dans le tableau
    tab[i].temps = tmps; // on stock la valeur tmps dans le tableau
    i++;

}
fclose(f);
//on ferme le fichier et on retourne le compteur i
return (i);
}
```

Notre fonction lireFichier est de type int car nous comptons renvoyer le compteur (i) à la fin. Nous avons en paramètre la structure stockerInformation qui servira tout au long du programme. Nous initialisons notre buffer, qui contiendra les valeurs du tableau.

Ensuite, nous pouvons passer à l'initialisation des variables que nous allons manipuler ici. Nous avons dans un premier temps i, qui sera un compteur, pls, qui représentera le pouls et tmps qui représentera le temps (nous passons par ces variables pour récupérer les valeurs du buffer). Nous ouvrons ensuite le fichier de test Randgen.csv, avec fopen, en lecture seule (« r » = read). Une fois ouvert, nous vérifions si le fichier est bel et bien ouvert avec la condition

si f est nul, alors il n'y a pas eu de changement (car nous l'avons initialisé à NULL au début) et donc le fichier n'a pas pu être ouvert. Nous pouvons donc refermer le programme avec l'instruction exit(1). Si ce fichier est ouvert, nous l'affichons, et nous pouvons ainsi passer à la suite.

Nous allons à présent récupérer les valeurs du buffer à l'aide de la fonction fgets. Tant que nous aurons encore des valeurs à récupérer dans le buffer (tant que la récupération de donnée ne sera pas nulle), nous incrémenterons le compteur de 1. L'instruction « tab[i].pouls = pls » injecte les valeurs récupérées auparavant dans le tableau tab, de même pour l'instruction « tab[i].temps = tmps » mais cette fois-ci pour le temps.

Une fois toutes les données récupérées, nous pouvons fermer le fichier (« fclose(f) ») et nous retournons le compteur en fin de fonction.

Toujours sur ce fichier donnees.c, nous avons une fonction servant à initialiser les structures :

```
void initialisationStructures(struct stockerInformation *tab, int tailleTab){
    int i;
    for (i = 0; i <= tailleTab; i++){
        tab[i].pouls = 0;
        tab[i].temps = 0;
    }
}
```

Dans cette fonction nous avons en paramètre la structure stockerInformation *tab, ainsi que l'entier tailleTab représentant la taille du tableau de données. Nous initialisons l'entier i (compteur) et tout de suite après nous entamons notre boucle pour, nous permettant d'initialiser la valeur du tableau à 0 pour le pouls ainsi que pour le temps.

Le fichier donnees.c étant expliqué, nous pouvons passer à son header, donnees.h.

Dans le fichier donnees.h, nous avons les bibliothèques utilisées :

```
#include <stdio.h>
#include <stdlib.h>

#define TAILLE_MAX_STRING 15

#define TAILLE_TAB 100
```

Nous utiliserons la bibliothèque stdio.h ainsi que stdlib afin d'avoir les fonctions de base. Nous pouvons constater que nous avons deux define, qui sont respectivement define TAILLE_MAX_STRING (taille max des chaînes de caractères) initialisé à 15 (la chaîne de caractère ne pourra jamais dépasser 15 caractères), et TAILLE_TAB qui est la taille maximum

du tableau de données. Ensuite, nous avons les prototypes des fonctions/structures utilisées dans le fichier donnees.c.

```c
struct stockerInformation{
    int pouls;
    int temps;
};

struct stockerInformation *tab;

int lireFichier(struct stockerInformation *donnees);

void initialisationStructures(struct stockerInformation *tab, int tailleTab);
```

Nous pouvons reprendre ici stockerInformation, structure composée du pouls et du temps, après nous avons la référence au tableau, toujours avec stockerInformation, la fonction lireFichier avec ses paramètres respectifs et en dernier notre fonction servant à initialiser les structures afin d'éviter les éventuelles erreurs possibles.

Les fichiers donnees.c et .h étant épuisés, nous pouvons passer à l'explication des fichiers de menu.

Le menu.c est composé de deux éléments importants, le menu, affichant les possibilités d'actions et le switch/case permettant d'amorcer la réalisation de l'action.

Le menu se présente comme ceci :

```c
void menu(struct stockerInformation *tab, int tailleTab){

    int choixMenu;
    printf("-=-= MENU =-=-\n\n");
    printf("1. Afficher les donnees dans l ordre du fichier .csv\n");
    printf("2. Afficher les donnees en ordre croissant selon le pouls\n");
    printf("3. Afficher les donnees en ordre croissant selon le temps\n");
    printf("4. Afficher les donnees en ordre decroissant selon le temps\n");
    printf("5. Afficher les donnees en ordre decroissant selon le pouls\n");
    printf("6. Rechercher et afficher les donnees pour un temps particulier\n");
    printf("7. Afficher la moyenne de pouls dans une plage de temps donnee\n");
    printf("8. Afficher le nombre de lignes de donnees actuellement en memoire\n");
    printf("9. Rechercher et afficher les max/min de pouls (avec le temps associe)\n");
    printf("10. Quitter l application\n");

scanf("%d", &choixMenu); //on récupère le choix de l'utilisateur
```

Une fois toutes les propositions énoncées, nous pouvons récupérer l'information à l'aide d'un scanf. Une fois l'information récupérée, nous traitons cette information à l'aide d'un switch avec un « case » pour chaque action à réaliser.

```
switch (choixMenu){ // chaque choix appel une fonction

case 1: //si l'utilisateur tape 1
    afficherDonnees(tab, tailleTab);
    break;

case 2: //si l'utilisateur tape 2
    triCroissantPouls(tab, tailleTab);
    break;

case 3: //si l'utilisateur tape 3
    triCroissantTemps(tab, tailleTab);
    break;

case 4: //si l'utilisateur tape 4
    triDecroissantTemps(tab, tailleTab);
    break;

case 5: //si l'utilisateur tape 5
    triDecroissantPouls(tab, tailleTab);
    break;

case 6: //si l'utilisateur tape 6
    rechercheDonnees(tab, tailleTab);
    break;

case 7: //si l'utilisateur tape 7
    moyenneDePouls(tab, tailleTab);
    break;

case 8: //si l'utilisateur tape 8
    afficherMemoire(tab, tailleTab);
    break;

case 9: //si l'utilisateur tape 9
    afficherPoulsMinMax(tab, tailleTab);
    break;


default : //sinon
    quitterApplication();
    break;
```

Si l'utilisateur tape 10 ou quelque chose d'autre, le programme se refermera. Une fois ce fichier examiné, on peut passer au menu.h.

Dans le menu.h, nous avons la déclaration des bibliothèques et du fichier actions.h car les actions à réaliser se trouvent dans le fichier actions.c, donc sont déclarées dans le fichier actions.h.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "actions.h"
```

Nous utilisons les fonctions de bases, et nous incluons comme dit précédemment « actions.h ». Ensuite, nous avons un seul prototype car le menu.c n'est composé que d'une fonction, la fonction menu.

```c
void menu(struct stockerInformation *tab, int tailleTab);
```

Les fichiers menu.c et .h étant éclairés, nous pouvons passer au main.c, fichier principal servant à l'exécution du programme. De plus, nous avons le lien avec le lien vers le menu.h.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "menu.h"
```

Le fichier main.c est composé de ses bibliothèques ainsi que de l'int main().

```c
int main()
{
    struct stockerInformation tab[TAILLE_TAB];
    initialisationStructures(tab , TAILLE_TAB);
    int nbrDeDonnees = 0;

    nbrDeDonnees = lireFichier(tab); //on lit le fichier .csv

    menu(tab, nbrDeDonnees); // on affiche le menu afin de choisir l'action
    return 0;
}
```

Le main est composé de la structure stockerInformation, de l'initialisation des structures afin d'éviter les éventuelles erreurs possibles. Comme la fonction lireFichier renvoi une donnée, nous devons le stocker dans une autre variable, d'où la fonction nbrDeDonnées. Bien évidemment nous l'avons initialisé juste avant. Nous pouvons ensuite inclure le menu avec en paramètre le tableau et nbrDeDonnees donc la fonction lireFichier. Ensuite on retourne 0 afin de clôturer le main.

Nous pouvons à présent rentrer dans le cœur, dans le vif, dans la partie dure du module 4 : actions.c (et actions.h). Ce fichier est le fichier comportant toutes les fonctions exécutables dans le menu.

Elles sont au nombre de neuf, et nous pouvons les citer dans l'ordre :

- Afficher les données dans l'ordre ;
- Trier par ordre croissant du pouls ;
- Trier par ordre croissant du temps ;
- Trier par ordre décroissant du temps ;
- Trier par ordre décroissant du pouls ;
- La recherche de donnée ;
- La moyenne de pouls sur une plage de temps donnée ;
- Afficher le nombre de ligne de donnée ;
- Afficher le pouls min et le pouls max.

Dans le fichier actions.c, nous devons d'ores-et-déjà inclure le fichier actions.h :

```
#include "actions.h"
```

Puis nous pouvons passer à la première fonction :

```
//Afficher les données dans l'ordre du fichier .csv

void afficherDonnees(struct stockerInformation *tab, int tailleTab){
    int i=0;
    while (i < tailleTab){
        printf("Pouls = %d.\nTemps = %d ms.\n\n", tab[i].pouls, tab[i].temps);
        i++;
    }
}
```

Cette fonction est de type void car elle ne renverra aucune valeur, puis en paramètre on inclut la structure stockerInformation ainsi que la taille du tableau tailleTab. Nous verrons à la suite que ces paramètres sont ceux d'approximativement toutes les fonctions. Ensuite, après avoir initialisé le compteur, nous créons la boucle qui fait que tant que i est inférieur à la taille du tableau, on affiche le pouls et le temps, puis on incrémente afin de parcourir toutes les lignes du tableau.

En seconde position, nous avons la fonction de tri croissant en fonction du pouls, elle se présente comme ceci :

```
//Afficher les données en ordre croissant/décroissant (selon le temps, selon le pouls)

void triCroissantPouls(struct stockerInformation *tab, int tailleTab){
    int i, j, memory, memory2 = 0; // i et j seront les cases du tableau
    for (i = 0; i<tailleTab; i++){ //on parcourt toutes les cases du tableau
        for (j = i; j<tailleTab; j++){ //données croissantes
            if (tab[j].pouls < tab[i].pouls){
                memory = tab[i].pouls; //on met les données en mémoire
                tab[i].pouls = tab[j].pouls;
                tab[j].pouls = memory;
                memory2 = tab[i].temps;//on met les données en mémoire
                tab[i].temps = tab[j].temps;
                tab[j].temps = memory2;
            }
        }
    }
    afficherDonnees(tab, tailleTab);
}
```

Les fonctions de tris seront toutes de types void, car elles ne renverront rien et afficheront le tableau trié grâce à la fonction afficherDonnees en dernière position.

Tout d'abord, nous initialisons les cases du tableau i et j qui seront deux compteurs, puis nous initialisons des variables memory qui serviront de tampon. On parcourt toutes les données à l'aide de la première boucle for, puis nous agissons sur l'action demandée (croissant/décroissant) avec la 2nd boucle for. Ensuite nous passons au tri avec la boucle if. Une fois les données en mémoire, on inverse ces données s'il y a nécessité. A la fin on affiche le tableau trié avec la fonction afficherDonnees.

Une fois cette fonction terminée, on peut passer à la troisième : tri croissant en fonction du temps.

```
void triCroissantTemps(struct stockerInformation *tab, int tailleTab){
        int i, j, memory, memory2 = 0;
    for (i = 0; i<tailleTab; i++){
        for (j = i; j<tailleTab; j++){
            if (tab[j].temps < tab[i].temps){
                memory = tab[i].pouls;
                tab[i].pouls = tab[j].pouls;
                tab[j].pouls = memory;
                memory2 = tab[i].temps;
                tab[i].temps = tab[j].temps;
                tab[j].temps = memory2;
            }
        }
    }
    afficherDonnees(tab, tailleTab);
}
```

Même forme que la précédente, le seul détail changeant est la boucle pour, nous avons remplacé pouls en .temps. Puis nous avons affiché le tableau.

Passons maintenant au tri décroissant en fonction du pouls.

```
void triDecroissantPouls(struct stockerInformation *tab, int tailleTab){
        int i, j, memory, memory2 = 0;
    for (i = 0; i<tailleTab; i++){
        for (j = i; j<tailleTab; j++){
            if (tab[j].pouls > tab[i].pouls){
                memory = tab[i].pouls;
                tab[i].pouls = tab[j].pouls;
                tab[j].pouls = memory;
                memory2 = tab[i].temps;
                tab[i].temps = tab[j].temps;
                tab[j].temps = memory2;
            }
        }
    }
    afficherDonnees(tab, tailleTab);
}
```

Toujours la même forme que les précédents, mais dans la boucle pour nous avons les données du pouls, puis le signe est « > », car cette fois-ci nous voulons un tri décroissant. Nous affichons à la fin le tableau trié en fonction du pouls de manière décroissante.

Le dernier tri est le tri décroissant en fonction du temps.

```
void triDecroissantTemps(struct stockerInformation *tab, int tailleTab){
        int i, j, memory, memory2 = 0;
    for (i = 0; i<tailleTab; i++){
        for (j = i; j<tailleTab; j++){
            if (tab[j].temps > tab[i].temps){
                memory = tab[i].pouls;
                tab[i].pouls = tab[j].pouls;
                tab[j].pouls = memory;
                memory2 = tab[i].temps;
                tab[i].temps = tab[j].temps;
                tab[j].temps = memory2;
            }
        }
    }
    afficherDonnees(tab, tailleTab);
}
```

Cette fois-ci nous mettons le temps dans la boucle if, et nous mettons le signe « > » de manière à avoir un tri décroissant. Comme toujours, nous affichons le tableau à la fin.

Une fois les tris réalisés, nous pouvons passer à la 6e fonction : la fonction de recherche. Tout d'abord, nous demandons une plage de temps, l'utilisateur entre son temps en milliseconde et le programme affiche toutes les données dans sa plage de temps.

```
//Rechercher et afficher les données pour un temps particulier

void rechercheDonnees(struct stockerInformation *tab, int tailleTab){
    int plageDeTemps,i=0; //on initialise le temps souhaité
    printf("Donnez une plage de temps en milliseconde.\n\n");
    scanf("%d", &plageDeTemps); // on récupère la plage de temps
    while (tab[i].temps < plageDeTemps){
        printf("Pouls = %d.\nTemps = %d ms.\n\n", tab[i].pouls, tab[i].temps);
        i++;
    }
}
```

Cette fonction est de type void car elle ne renvoie aucune valeur, puis on récupère une plage de temps, une boucle tant que permet ensuite d'afficher toutes les données comprises entre t = 0 et t = plageDeTemps. On incrémente le compteur afin de parcourir tout le tableau.

La 7e fonction est la moyenne de pouls dans une plage de temps donnée.

```
//Afficher la moyenne de pouls dans une plage de temps donnée


void moyenneDePouls(struct stockerInformation *tab, int tailleTab){

    float sommePouls = 0.0, moyenne = 0.0;
    int plageDeTemps, i =0;


    printf("Donnez une plage de temps en ms :\n");
    scanf("%d", &plageDeTemps);

    while (tab[i].temps <= plageDeTemps){
        sommePouls += tab[i].pouls; // sommePouls = sommePouls + tab[i].pouls
        i++;
    }

    moyenne = sommePouls/i;
    printf("\nVotre moyenne de pouls est de : %f \n", moyenne);
}
```

Tout d'abord on initialise nos variables, puis on récupère la plage de temps, pour enfin la traiter avec la boucle tant que. On compare les valeurs du temps du tableau avec la plage de temps afin de ne pas dépasser le temps indiqué. Puis on calcul la moyenne afin de l'afficher à l'utilisateur.

La 8e fonction est la fonction d'affichage de mémoire.

```c
void afficherMemoire(struct stockerInformation *tab, int tailleTab){
    printf("\nIl y a actuellement %d lignes de donnees en memoire.\n", tailleTab);
}
```

Fonction courte car il s'agit simplement d'afficher la mémoire (et donc tailleTab).

L'avant-dernière fonction est l'affichage du pouls min et du pouls max

```c
void afficherPoulsMinMax(struct stockerInformation *tab, int tailleTab){
    int min, max, i = 0;
    min = tab[i].pouls;
    max = tab[i].pouls;

    for (i=1; i < tailleTab; i++){

        if (min > tab[i].pouls){
            min = tab[i].pouls;
        }

        if (max < tab[i].pouls){
            max = tab[i].pouls;
        }
    }
    printf("Le pouls minimum est : %d\nLe pouls maximum est : %d\n\n", min, max);
}
```

Tout d'abord, nous initialisons les variables, puis nous stockons le pouls du tableau dans min ET dans max, afin de traiter les données après dans une boucle pour. On affiche le min quand le pouls du tableau est au minimum et inversement pour le maximum. Enfin on affiche les valeurs du minimum et du maximum.

La dernière fonction (et la plus simple) est la fonction quitterApplication car elle n'est composé que de exit(1).

```
//Quitter l'application
int quitterApplication(){
    exit(1);
}
```

Et dernière position nous avons composé du fichier actions.h qui est composé des prototypes des fonctions du actions.c.

```
#include <stdio.h>
#include <stdlib.h>
#include "donnees.h"


void afficherDonnees(struct stockerInformation *tab, int tailleTab);

void triABulle(struct stockerInformation *tab);

void triCroissantPouls(struct stockerInformation *tab, int tailleTab);

void triCroissantTemps(struct stockerInformation *tab, int tailleTab);

void triDecroissantPouls(struct stockerInformation *tab, int tailleTab);

void triDecroissantTemps(struct stockerInformation *tab, int tailleTab);

void rechercheDonnees(struct stockerInformation *tab, int tailleTab);

void moyenneDePouls(struct stockerInformation *tab, int tailleTab);

void afficherMemoire(struct stockerInformation *tab, int tailleTab);

void afficherPoulsMinMax(struct stockerInformation *tab, int tailleTab);

int quitterApplication();
```

Partie 7

Analyse des écarts

Nous avons eu plusieurs écarts par rapport à la planification. En effet la réalisation du montage du module 2 par Achille et Corentin a été plus rapide que prévue. De ce fait ils ont voulu continuer sur leur lancé et faire le montage électronique du module 1.

Un autre problème important rencontrer est la réalisation de certaine fonction du code C du module lecture et traitement de données. On a aussi eu des problèmes par rapport à la gestion du tableau en C.

Au niveau des montages électroniques nous avons eu quelque problème pour la polarité des condensateurs et le code couleur des résistances.

Apports personnels

Bilan personnel :

Brossier Achille

Ce projet m'as permis de découvrir la programmation en Arduino et de travailler sur un montage moyennement complexe. J'ai développé mes compétences à travailler en groupe. J'ai beaucoup aimé cette méthode de travail, se partager les tâches, mettre en commun nos travaux accomplis et se réunir pour réussir à résoudre certains problèmes techniques. J'ai réussi à créer des liens professionnels avec mes collègues de projet afin de réussir à remplir nos objectifs.

Hangard Corentin

Ce projet m'a apporté la compréhension du lien entre le codage C et C Arduino mais aussi le lien entre un code et un montage électronique. Ce projet m'a permis de de consolider le travail en équipe, donc l'entraide, la répartition du travailles.

Lupart Gwendal

Ce projet m'a beaucoup apporté, et principalement, une meilleure vision de la pression possible du monde du travail. J'ai appris à respecter une plage de temps donnée pour réaliser un travail ample et complexe. Le travail de groupe ne fut que bénéfique et la bonne dynamique implantée par un excellent chef de projet améliora ma vision du travail de groupe. J'ai appris que le rôle de chef de projet était plus qu'important et ce rôle peut m'intéresser pour les projets à venir. J'ai amélioré ma précision et ma vitesse de programmation : j'analyse plus rapidement mes erreurs et les corrige plus proprement. Maintenant, je sais que j'aime la pression qu'on peut avoir lors du travail de groupe, c'est un objectif à accomplir et c'est une épreuve fascinante.

Semard Charles (chef de projet)

Pour ce premier projet, je n'ai pas eu de problème d'intégration avec mon groupe, il y a eu une bonne ambiance dès le départ. Je ne suis pas forcément bon en montage électronique mais ce projet m'as permis de m'améliorer.

De plus cela m'a permis de me rendre compte de l'importance d'un chef de projet car c'est lui qui vas pousser les gens à travailler, qui va donner le rythme de travail et établir une bonne ambiance, qui vas répartir le travail et essayer d'apporter des solutions aux membres du groupes. Cependant je pense que j'aurais dû plus m'appliquer dès le départ car au début je n'avais pas une vision du rôle de chef de projet.

Evolutions possibles

Sur la fin du projet nous avons constaté que nous pouvions créer d'autre type d'affichage et évidement améliorer la qualité du montage de la pince

Partie 8

Annexes

Les différents codes ainsi qu'une vidéo montrant le montage sont dans le dossier.

Outils utilisés

Pour les algorigramme : draw.io

Pour le planificateur : Excel

Pour le traitement de texte : Word

Pour la programmation

- En C : CodeBlock
- Arduino : Arduino et Processing