

A dark blue vertical bar runs along the left edge of the page. A blue arrow points to the right from this bar, containing the date.

17/11/2017

Projet fondamentaux scientifique

Livrable G2

Table des matières

Rappel du sujet	2
Travail demandé :	2
Contrainte :	3
Analyse du sujet	3
Organisation prévue	4
Module cardio	6
Montages électroniques.....	6
Programmation C Arduino	8
Module cœur de leds.....	9
Montage électronique.....	9
Programmation C Arduino	10
Programmation C	11
Module Processing et Acquisition de données	12
Module Lecture et traitement de données	14
Mise en commun.....	15
Organisation effectuée.....	17
Bilan du groupe et personnel :	17
Pistes d'améliorations	18

Rappel du sujet & organisation

RAPPEL DU SUJET

a. Contexte :

Le sujet que nous avons eu nous place dans la position d'étudiant à l'école **Cesi.exia** en tant que postulant au diplôme d'ingénieur spécialité informatique en première année.



La société HeXart Care ayant été complètement dépouillée par un ingénieur étant le seul à connaître les plans ainsi chapardés, elle ne peut plus que faire appel à nous pour (re)développer son cardiofréquencemètre fonctionnant entre autres par le biais de la photo pléthysmographie.

De ce braquage, il ne reste alors plus que quelques données divisées en 4 modules différents mais qui pourraient cependant permettre d'enfin réaliser ce cardiofréquencemètre.

Travail demandé :

- Construire le montage d'un cardiofréquencemètre
- Gérer le montage à l'aide d'une carte Arduino
- Construire le montage d'un cœur en LEDs
- Gérer le montage du cœur à l'aide de la carte Arduino
- Gérer les paramètres du programme Arduino à l'aide d'un code fait en langage C

- Utiliser Processing afin d'acquérir les données arrivant sur le port série
- Lire et traiter les données acquises dans un fichier .csv à l'aide d'Arduino
- Créer une structure avec plusieurs fonctionnalités dont être capable de trier ou de rechercher un temps.

Contrainte :

- Une semaine pour réaliser un cardiofréquencemètre avec au moins certaines fonctions et un montage fait et fonctionnel.
- Devoir s'adapter au style de programmation demandé dans le sujet (utiliser tant de fichier pour faire une partie d'un module ou utiliser telle méthode par exemple).
- Contraintes matérielles.

ANALYSE DU SUJET

Après avoir lu puis analysé le sujet, nous sommes alors tombés d'accord sur l'interprétation du sujet suivante :

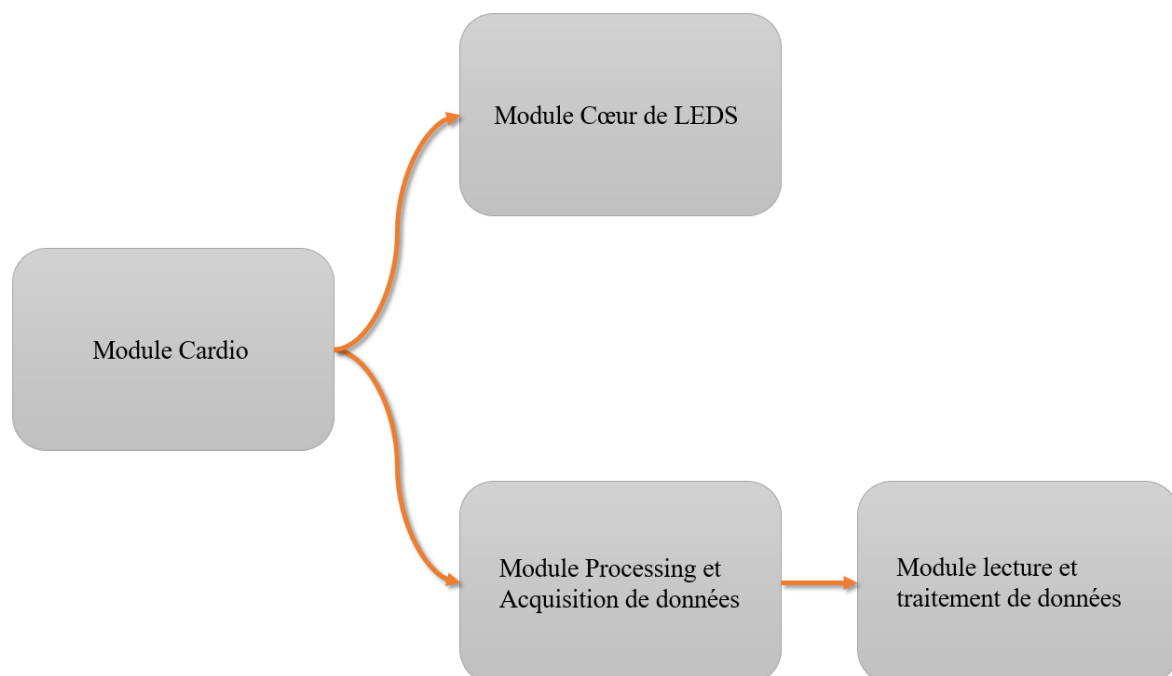


Schéma global du projet

Le projet fonctionnerait donc ainsi :

- On utilise le module **Cardio** pour détecter le pouls de la personne, le module permet de lire une valeur et ensuite de transformer cette valeur en pouls tout en donnant la date ou le temps.
- Les données du module **Cardio** sont transmises aux modules Cœur de LEDS et Processing et Acquisition de données.
 - ⇒ Le module **Cœur de LEDS** prend les valeurs en temps réel et agit sur les LEDS disposées en forme de cœur selon le fichier param.h
 - ⇒ Le module **Processing et acquisition de données** récupère les données et les stocke dans un fichier .csv que lit ensuite le module **Lecture et traitement de donnée**

⇒

En termes de programmation, on peut alors constater le schéma suivant :

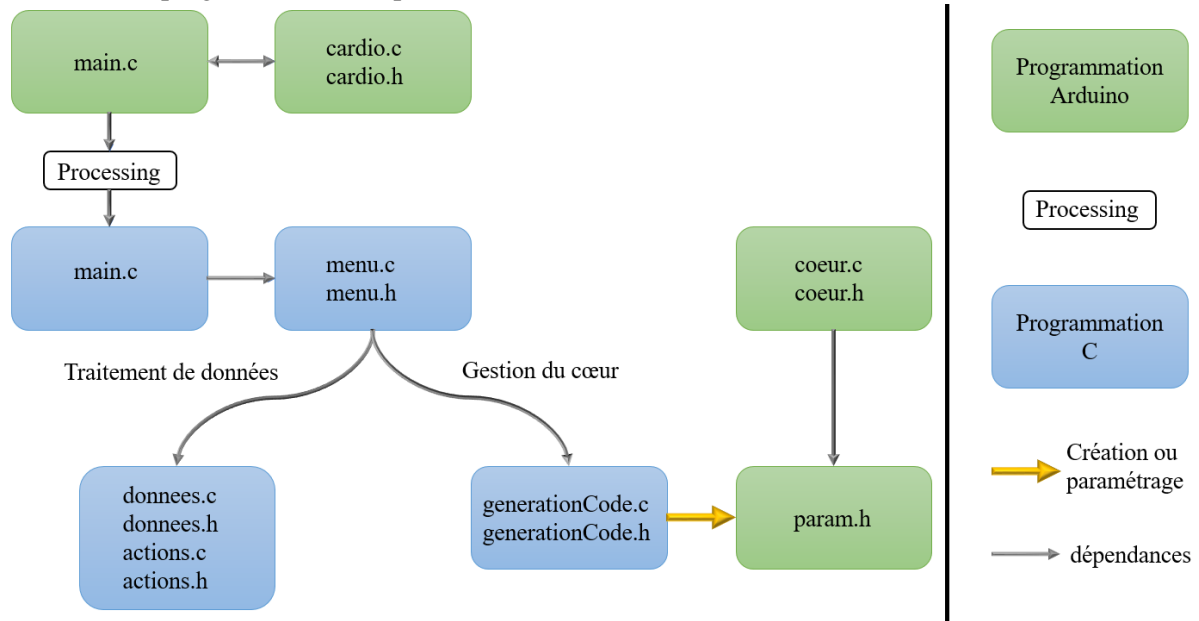


Schéma des dépendances entre les fichiers

C'est-à-dire que chaque fichier est dépendant du fichier duquel part la flèche. En sachant que le main.c, le cardio.c et le cardio.h pourraient être directement relié à l'autre partie du code Arduino en fonctionnant totalement.

On perdrait alors cependant la partie stockage de données par le biais de Processing ainsi que son exploitation par le langage C mais aussi la possibilité de paramétrer le cœur de LED via le langage C.

ORGANISATION PREVUE

En analysant le projet, nous nous sommes rendu compte que travailler chacun sur un module n'était pas forcément la bonne méthode étant donné que certains modules demandaient plus de travail que d'autre. Comme par exemple le module 3.4 qui demandait une charge énorme de travail a contrario du module 3.3. Nous avons donc opté pour organisation en fonction des compétences, des affinités et bien entendu du choix de chacun vis-à-vis des environnements qu'on devait manipuler.

PROJET - fondamentaux scientifiques							
Semaine du : novembre 13							
	13/11 LUNDI	14/11 MARDI	15/11 MERCREDI	16/11 JEUDI	17/11 VENDREDI	18/11 SAMEDI	19/11 DIMANCHE
BLIN Clément	Module 1 & 2 partie Arduino	Module 1 & 2 partie Arduino	Résolution des problèmes	MISE EN COMMUN	PREPARATION SOUTENANCE		REVOIR
BOUTIN Pierre	Module 4	Module 4	Module 4	MISE EN COMMUN	PREPARATION SOUTENANCE		REVOIR
BUQUET Dorian	Module 1 & 2 partie électronique	Module 1 & 2 partie électronique	Assemblages et montages	MISE EN COMMUN	PREPARATION SOUTENANCE		REVOIR
SOULAS Thomas	Fiche d'avancement	Module 3	Module 2 partie 3	MISE EN COMMUN	PREPARATION SOUTENANCE		REVOIR

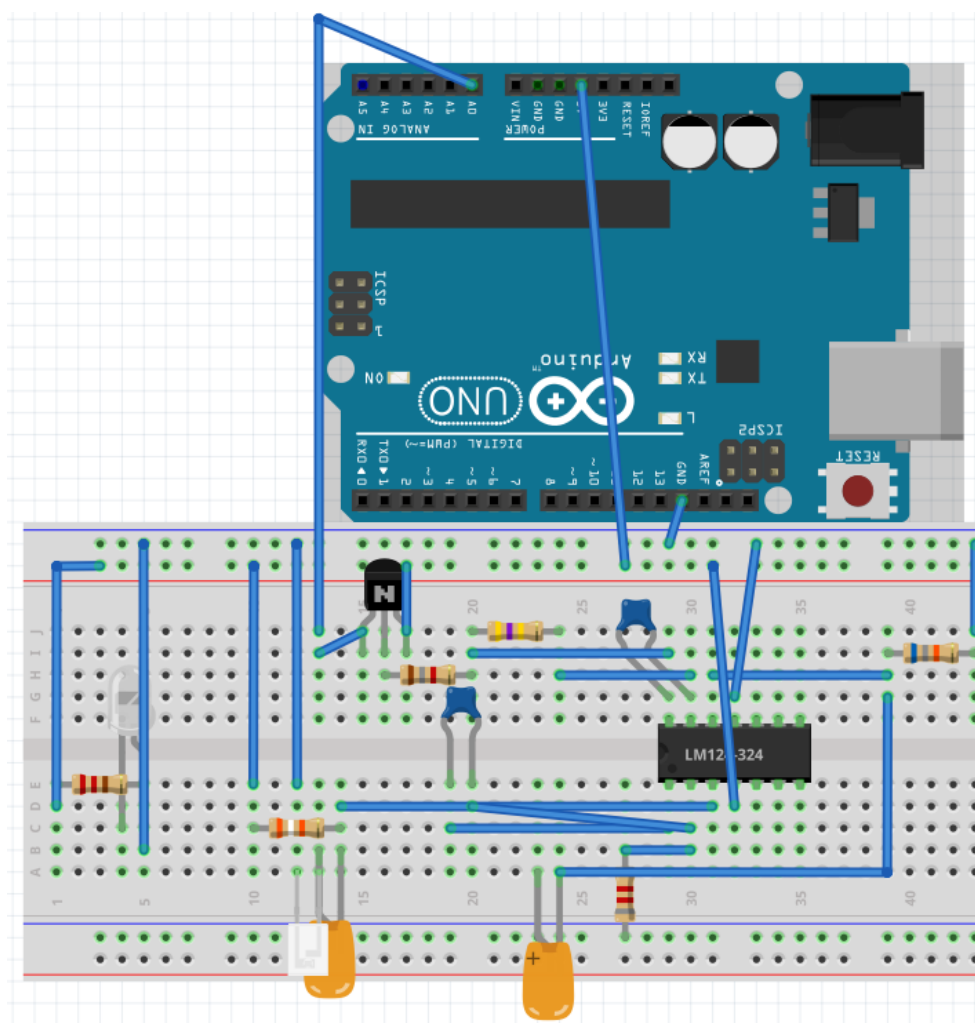
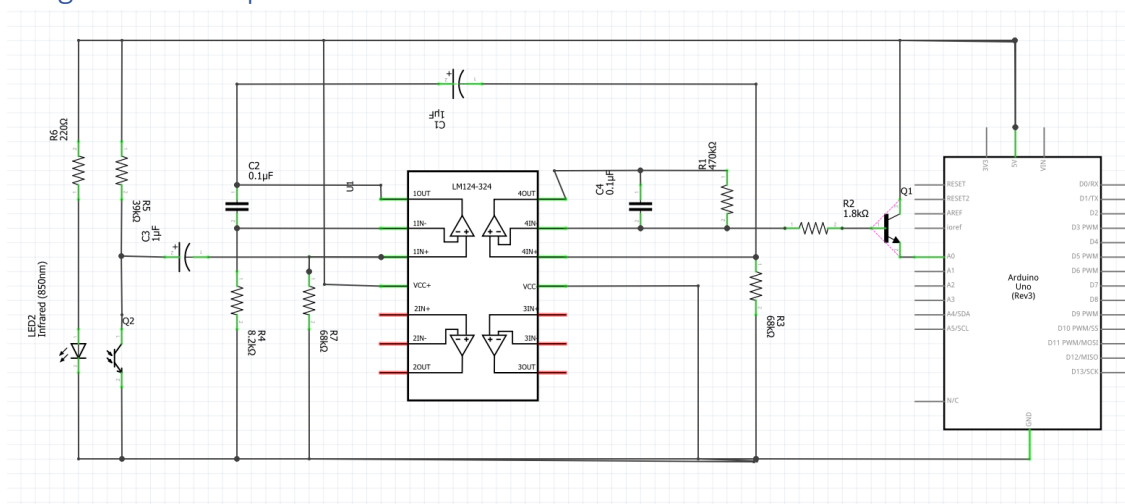
En conséquence, nous avons décidé ensemble que :

- Clément s'occuperait de toute la partie Arduino.
- Pierre, la partie C du module 4.
- Dorian, la partie montage électronique.
- Thomas, la partie C du module 2 et le module 3 en plus de la fiche d'avancement du lundi et le livrable à rendre.

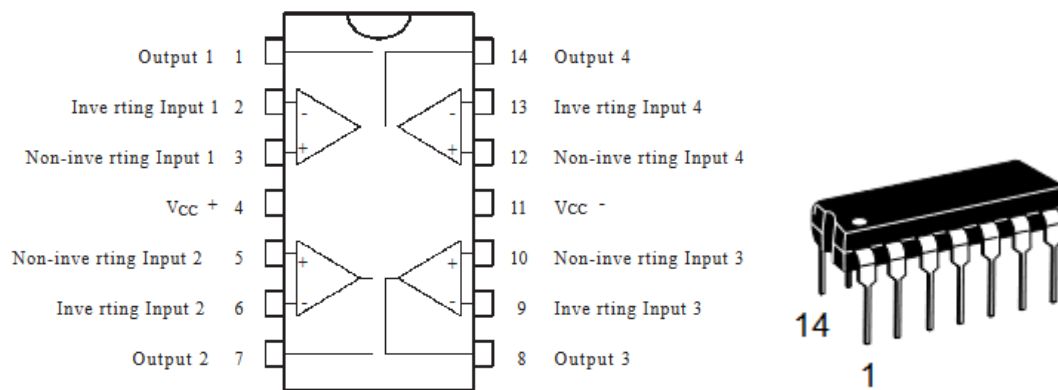
Suite à l'évaluation de la charge de travail et du temps qu'il nous faudrait chacun pour chaque partie, nous avons alors organisé ce planning :

MODULE CARDIO

Montages électroniques



Le module cardio fonctionne grâce à un microcontrôleur LM324 dont le document suivant est la datasheet :



Datasheet du microcontrôleur LM324

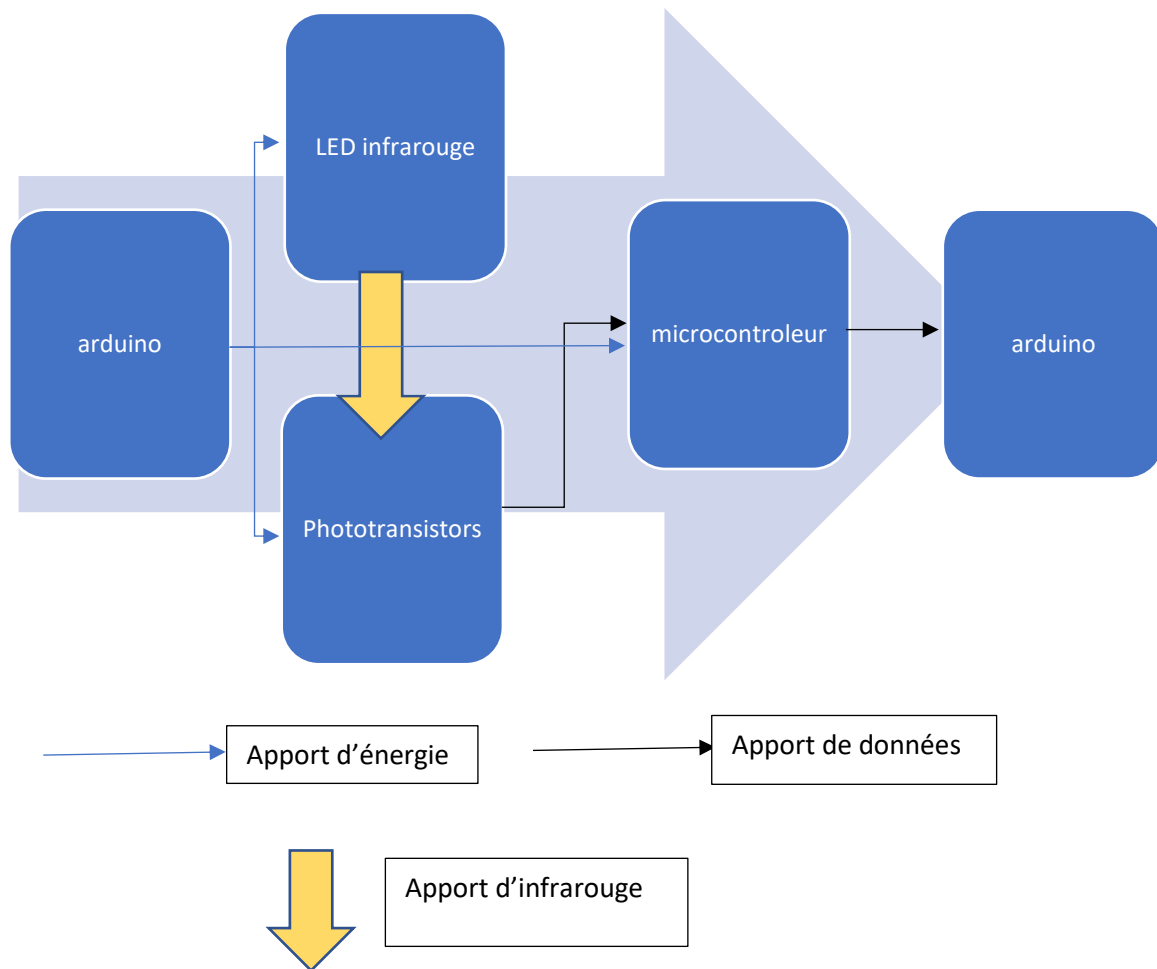
Le rond situé sur le montage permet de trouver où se trouve le port 1 du microcontrôleur puisqu'il est forcément à côté du 1 dans le montage.

Vis-à-vis du montage ci-dessus, le montage du module utilise seulement les ports allant de 1 à 4 et allant de 11 à 14 :

- Les ports 11 et 4 étant à part car le premier est juste branché au – et le second au + sans changement.
- Le port 1 est relié au port 2 par un condensateur conduisant à une résistance de 8.2Kohm puis au grounds et au port 11 (et à une résistance de 68 k Ω) via un condensateur électrique.
- Les ports 14 et 13 sont reliés à un condensateur est à une résistance (470 k Ω) pour accéder à une résistance(1.8Kohm) arrivant à un transistor qui a son entrée est relié au port 5V de l'Arduino et à sa sortie le port A0 de la même Arduino.
- Quant au port 3 il est relié à un condensateur électrique puis à un phototransistor allant vers le grounds. Le condensateur va aussi à une résistance (39 k Ω) prenant l'énergie du port 5V de l'Arduino.

L'Arduino donne le courant par son port 5V à la LED infrarouge, au phototransistor et au microcontrôleur.

Le microcontrôleur amène les données récupérées par le phototransistor à l'Arduino à son port A0.



Programmation C Arduino

```

int valeurPrecedente = 0;
long tempsPrecedent = 0;
int poulis;

int recolte()
{
    int valeurActuelle, valeurSeuil;
    long tempsDetection;

    valeurActuelle = analogRead(0);
    valeurSeuil = 650;

    if (valeurActuelle > valeurSeuil) { // seuil atteint
        if (valeurPrecedente <= valeurSeuil) { // vérification si première fois ou non
            tempsDetection = millis();
            if (tempsDetection > (tempsPrecedent + 200)){ // vérification parasite
                return((1000.0 * 60.0) / (tempsDetection - tempsPrecedent)); // retourne le poulis à la boucle.
                tempsPrecedent = tempsDetection;
            }
        }
    }

    valeurPrecedente = valeurActuelle;
}

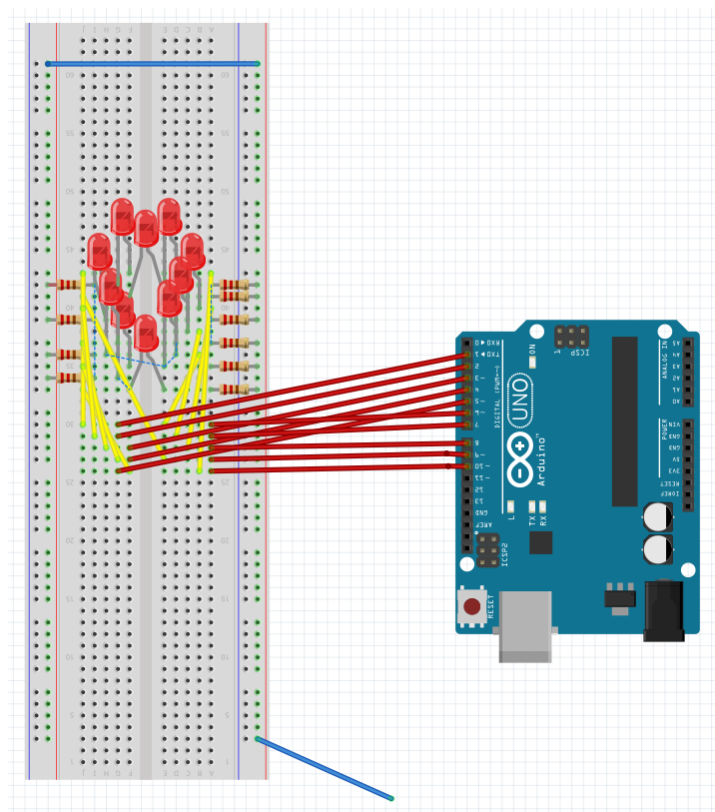
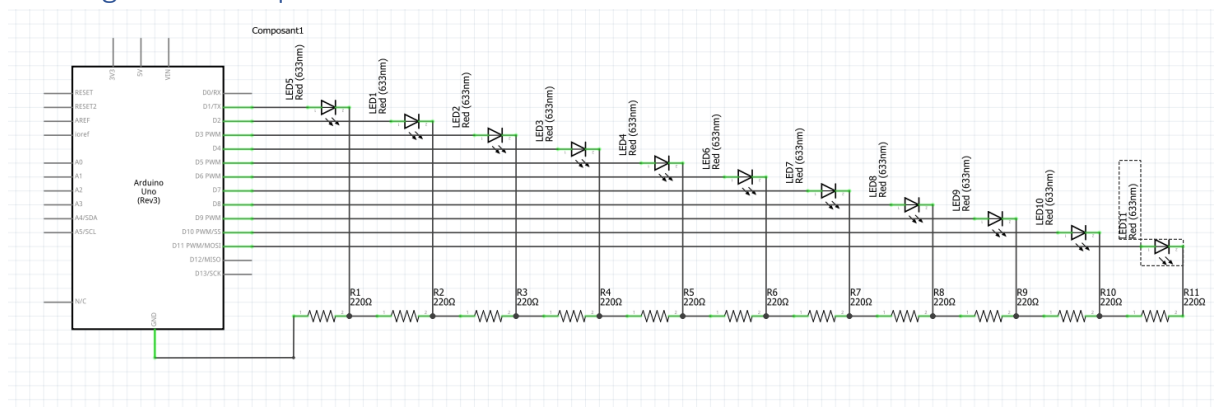
```

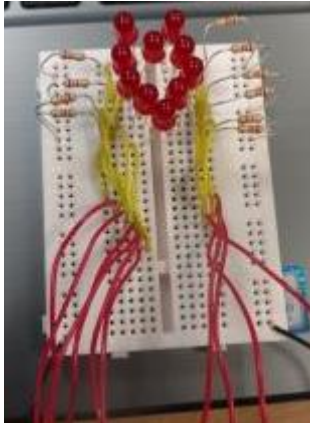
Dans un premier temps, nous avons initialisé un seuil à ne pas dépasser pour avoir une pulsation, si une valeur dépasse ce seuil alors ce n'est pas une pulsation. Ensuite on utilise la fonction Arduino millis()

qui permet de nous dire depuis combien de temps notre Arduino est allumé et on stocke cette valeur dans une variable. Nous avons ensuite fait une boucle **tant que** la fonction millis() moins la variable précédente est inférieure à un grand nombre alors le programme va lire les informations entrantes du port analogique 0 là où le phototransistor est branché. Il a ensuite fallu filtrer les pulsations du cœur et les parasites. Si on détecte une pulsation alors on stocke le temps à laquelle la pulsation a été effectuée dans une variable puis on attend que la pulsation se termine pour enregistrer à nouveau le temps dans une autre variable. Pour terminer, on calcule le pouls en utilisant le calcul $(1000 * 60) / (\text{seconde variable} - \text{première variable})$ et on renvoie le pouls et le temps initial (depuis quand l'Arduino est allumé).

MODULE CŒUR DE LEDS

Montage électronique





Chaque LED du module du cœur est branchée de cette façon : la branche la plus longue sur le « ground » (avec la résistance branchée avant) et la courte sur les ports donnant du courant de façon indépendante. Les câbles jaunes sont présents pour avoir plus de facilité à brancher chaque LED à l'Arduino. La LED se situant en haut à gauche est associée au port 1 et est donc la LED numéro 1 ensuite la LED la plus à gauche est la seconde et ainsi de suite pour former un cœur (le dernière LED étant celle du milieu en haut). Le « ground » correspond aux fils noirs et les fils rouge et jaune correspond au +.

Les LED étant gérées séparément. Il est donc possible de décider de leur ordre d'allumage.

Programmation C Arduino

Dans le module 3.2, je me suis occupé des différents allumages de LED en fonction du pouls.

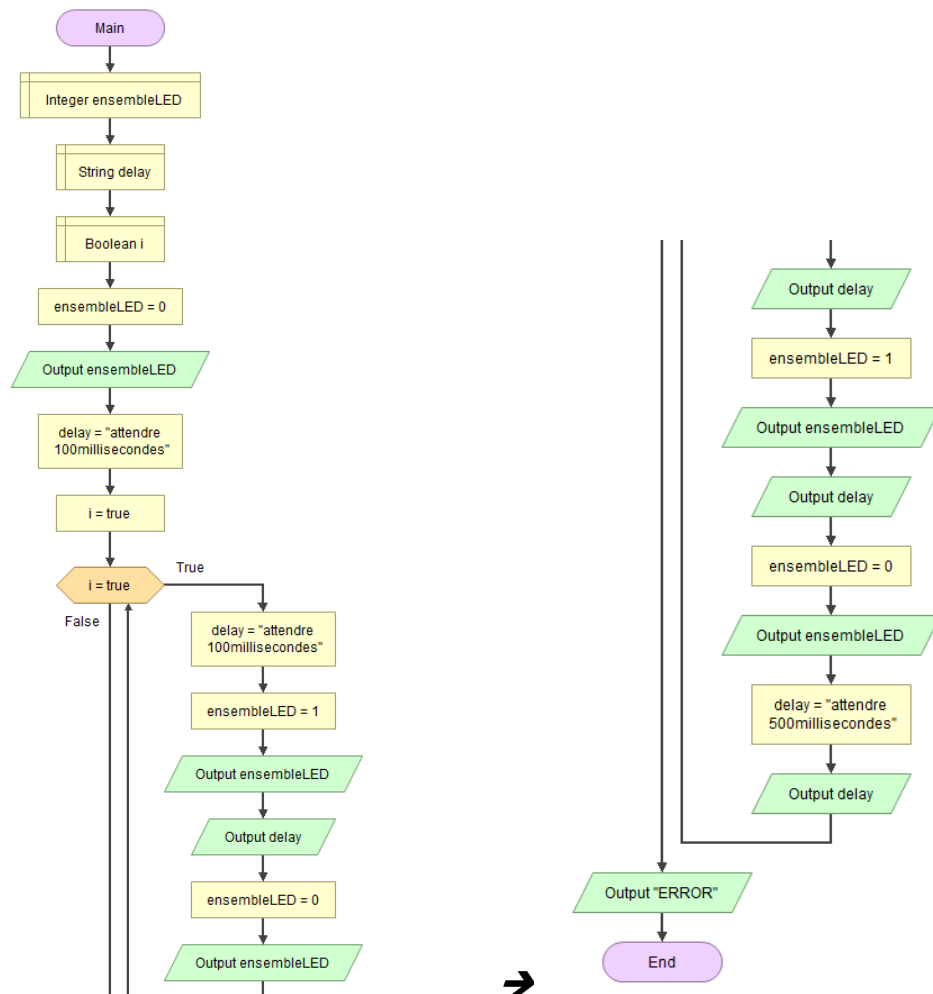
J'ai commencé par faire deux tableaux de constante, une pour les différents ports des LED et l'autre pour le temps d'attente dans les delay(). Ensuite, il a fallu régler 10 ports en mode sortie, donc les ports envoient des signaux aux LED, puis demandé aux LED de s'éteindre pour pouvoir lancer les différents modes d'allumage.

```

1
2 const int ledHeart[10] = {1,2,3,4,5,6,7,8,9,10};
3 const int wait[6] = {10, 50, 100, 250, 500, 1000};
4
5 void setup() {
6 {
7
8   pinMode(ledHeart[0], OUTPUT);
9   pinMode(ledHeart[1], OUTPUT);
10  pinMode(ledHeart[2], OUTPUT);
11  pinMode(ledHeart[3], OUTPUT);
12  pinMode(ledHeart[4], OUTPUT);
13  pinMode(ledHeart[5], OUTPUT);
14  pinMode(ledHeart[6], OUTPUT);
15  pinMode(ledHeart[7], OUTPUT);
16  pinMode(ledHeart[8], OUTPUT);
17  pinMode(ledHeart[9], OUTPUT);
18
19  digitalWrite(ledHeart[0], LOW);
20  digitalWrite(ledHeart[1], LOW);
21  digitalWrite(ledHeart[2], LOW);
22  digitalWrite(ledHeart[3], LOW);
23  digitalWrite(ledHeart[4], LOW);
24  digitalWrite(ledHeart[5], LOW);
25  digitalWrite(ledHeart[6], LOW);
26  digitalWrite(ledHeart[7], LOW);
27  digitalWrite(ledHeart[8], LOW);
28  digitalWrite(ledHeart[9], LOW);
29
30 }
31
/*Allumage 3 en 3*/
digitalWrite(ledHeart[0], HIGH);
delay(wait[2]);
digitalWrite(ledHeart[3], HIGH);
delay(wait[2]);
digitalWrite(ledHeart[6], HIGH);
delay(wait[2]);
digitalWrite(ledHeart[9], HIGH);
delay(wait[2]);
digitalWrite(ledHeart[2], HIGH);
delay(wait[2]);
digitalWrite(ledHeart[5], HIGH);
delay(wait[2]);
digitalWrite(ledHeart[8], HIGH);
delay(wait[2]);
digitalWrite(ledHeart[1], HIGH);
delay(wait[2]);
digitalWrite(ledHeart[4], HIGH);
delay(wait[2]);
digitalWrite(ledHeart[7], HIGH);
delay(wait[2]);
digitalWrite(ledHeart[0], LOW);
delay(wait[2]);
digitalWrite(ledHeart[3], LOW);
delay(wait[2]);
digitalWrite(ledHeart[6], LOW);
delay(wait[2]);
digitalWrite(ledHeart[9], LOW);
delay(wait[2]);
digitalWrite(ledHeart[2], LOW);
delay(wait[2]);
digitalWrite(ledHeart[5], LOW);
delay(wait[2]);
digitalWrite(ledHeart[8], LOW);
delay(wait[2]);
digitalWrite(ledHeart[1], LOW);
delay(wait[2]);
digitalWrite(ledHeart[4], LOW);
delay(wait[2]);
digitalWrite(ledHeart[7], LOW);
delay(wait[2]);
digitalWrite(ledHeart[0], LOW);
delay(wait[2]);

```

Exemple fonctionnement Cœur de LEDs :



Programmation C

La programmation C de ce module n'est pas complexe, le but est de gérer le fichier param.h par le langage C, il suffit donc d'accéder au fichier param.h et de changer ce qui est écrit dedans.

Pour ce faire, on utilise la fonction « *fichier = fopen("main/param.h", "w");* » afin de réinitialiser l'intérieur du fichier (pour ne pas réécrire par-dessus ce qui est déjà écrit mais tout effacer).

Soit le code suivant :

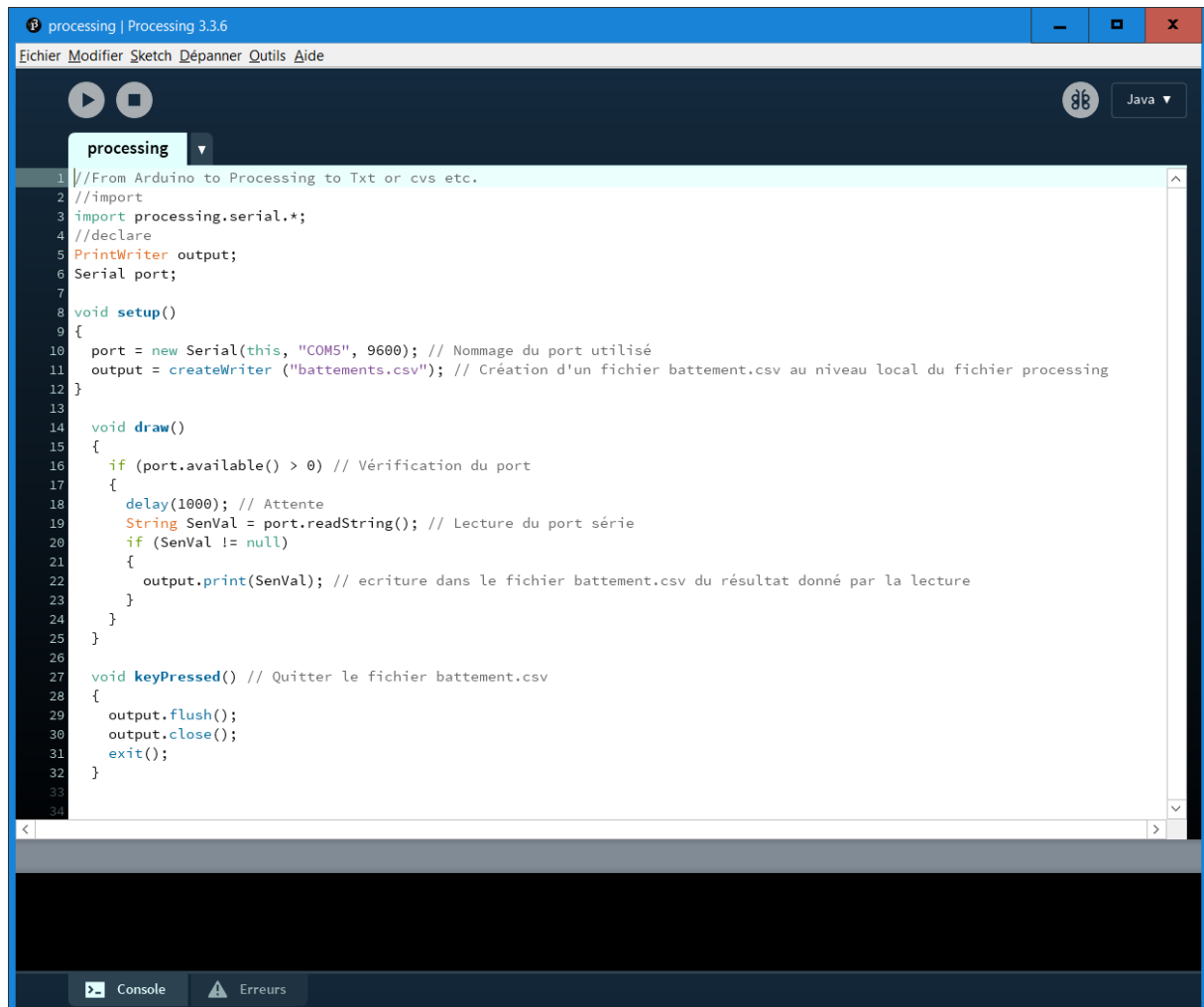
```

void Lecture(int choix)
{
    int d_Choix = 0;
    int nb_Leds;
    FILE* fichier = NULL;
    fichier = fopen("main/param.h", "w");
    printf("Votre choix est %d.", choix);
    switch(choix) // switch choix de l'utilisateur
    {
        case 1:
            LEDS_HeartBeat(fichier); // Appel de fonction generationCode.c
            break;
        case 2:
            printf("Il y a deux modes :\n 1) Les LEDs s'eteignent des apres qu'on passe à la suivant\n 2) Les LEDs s'eteignent a la fin de la boucle\n");
            while(d_Choix < 1 || d_Choix > 3)
            {
                scanf("%d", &d_Choix);
            }
            switch(d_Choix)
            {
                case 1:
                    LEDS_Chenille(fichier); // Appel de fonction generationCode.c
                    break;
                case 2:
                    LEDS_ChenilleAutre(fichier); // Appel de fonction generationCode.c
                    break;
            }
            break;
        case 3:
            printf("Une LED doit etre allumee tous les tant de LEDs, mais combien ?\n");
            scanf("%d", &nb_Leds); // nombre de LEDs
    }
}
  
```

MODULE PROCESSING ET ACQUISITION DE DONNEES

Le module 3 nommé Processing et acquisition de données permet de faire la passerelle entre les données que reçoit en direct le port Série et les données à traiter dans un fichier texte ou un tableur (fichier .csv) par le langage C.

Le module nous fournit un code, il suffit juste de le réadapter et de changer le port pour que ce soit bien celui utilisé par Arduino. Soit le code total suivant :

A screenshot of the Processing IDE (version 3.3.6) with a dark theme. The code editor shows a Java program for serial communication. The code includes comments in French explaining the steps: connecting to COM5 at 9600 bauds, creating a CSV file named 'battements.csv', and reading data from the serial port. The code is as follows:

```
1 //From Arduino to Processing to Txt or cvs etc.
2 //import
3 import processing.serial.*;
4 //declare
5 PrintWriter output;
6 Serial port;
7
8 void setup()
9 {
10  port = new Serial(this, "COM5", 9600); // Nomme du port utilisé
11  output = createWriter ("battements.csv"); // Création d'un fichier battement.csv au niveau local du fichier processing
12 }
13
14 void draw()
15 {
16  if (port.available() > 0) // Vérification du port
17  {
18    delay(1000); // Attente
19    String SenVal = port.readString(); // Lecture du port série
20    if (SenVal != null)
21    {
22      output.print(SenVal); // ecriture dans le fichier battement.csv du résultat donné par la lecture
23    }
24  }
25 }
26
27 void keyPressed() // Quitter le fichier battement.csv
28 {
29  output.flush();
30  output.close();
31  exit();
32 }
33
34
```

Explications :

```
void setup()
{
  port = new Serial(this, "COM5", 9600);
  output = createWriter ("battements.csv");
}
```

La fonction « *void setup()* » suit le même principe que dans l'IDE Arduino (même si Processing utilise le langage java et Arduino le langage C en majorité). Cette fonction permet donc le paramétrage.

Dans celle-ci on retrouve 2 fonctions :

➔ La première permet de désigner le port utilisé par l'Arduino, et donc le port duquel vont arriver les infos envoyées par la commande « **Serial.print** ».

➔ La deuxième nomme une variable qui permettra ensuite d'écrire dans un fichier battements.csv tout en créant ce fichier.

```
void draw()
{
  if (port.available() > 0)
  {
    delay(1000);
    String SenVal = port.readString();
    if (SenVal != null)
    {
      output.print(SenVal);
    }
  }
}
```

La fonction « *void draw()* » est l'équivalent de la fonction « *void loop()* » en C Arduino, elle exécute les lignes entre accolades de façon continue jusqu'à ce qu'on cesse le programme.

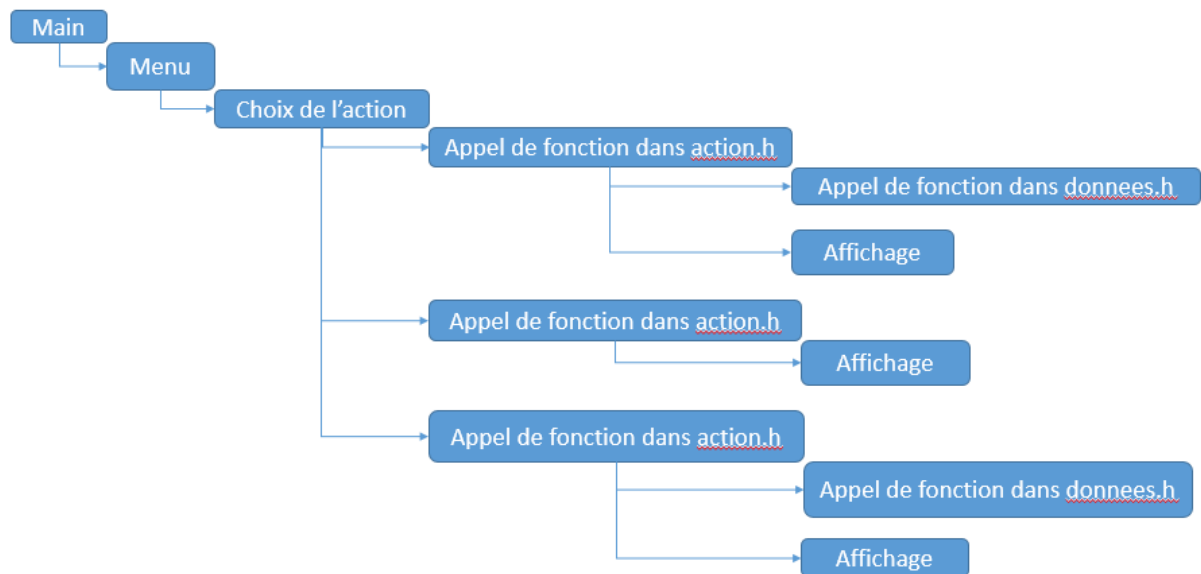
En l'occurrence, les instructions à l'intérieur de cette fonction font, dans l'ordre :

Vérification du port ➔ délais 1s ➔ Récupération valeur en port série ➔ Vérification valeur non « **null** » ➔ Si non « **null** », écriture de la valeur dans le fichier battements.csv.

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

La fonction « *void keyPressed()* » est appelée dès que l'on presse une « clef », en l'occurrence une clef est un caractère tapé sur le clavier. Dès que l'on tape un caractère (ou sur la touche entrée), la fonction va alors faire en sorte que toutes les données soient envoyées au fichier battements.csv puis fermer le programme.

MODULE LECTURE ET TRAITEMENT DE DONNEES



Nous devons réaliser un menu qui nous donnait accès à différentes fonctions pour traiter les données et en tirer des informations, comme par exemple trouver le pouls le plus haut dans toutes les données.

Pour ce faire nous avons des fonctions explicites (c'est-à-dire pour reprendre l'exemple plus haut une fonction : *AfficherPoulsMax* qui nous indique que cette fonction va afficher le pouls max. Mais pour que la fonction affiche le pouls max la fonction appelle une autre fonction qui va chercher le pouls max.

Le principe est donc simple :

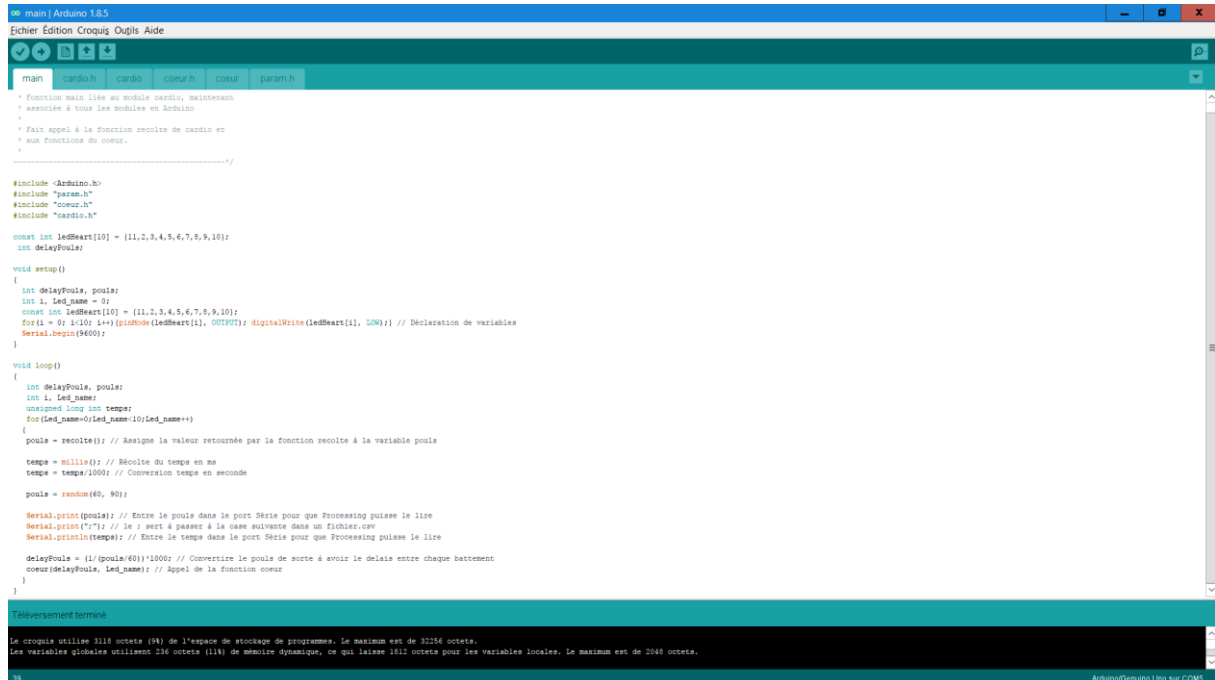


Pour réaliser cette partie nous avons donc créé une structure de type liste chaîné pour stocker toutes les données. Nous pouvons ainsi facilement la parcourir pour chercher telle ou telle information. Nous avons aussi mis en place un système de tri du temps et des pouls dans l'ordre croissant et décroissant.

Mise en commun

La mise en commun consiste à associer tous les fichiers ensembles, et, en l'occurrence à aussi lier la programmation Arduino et la programmation C.

Programmation Arduino



```
main | Arduino 1.8.5
Echier Edition Croquis Outils Aide

main | cardio.h | cardio | coeur.h | coeur | param.h

/* Fonction main liée au module cardio, maintenant
 * associée à tous les modules en Arduino
 *
 * Fait appel à la fonction recolte de cardio et
 * aux fonctions du coeur.
 *
 * =====*/

#include <Arduino.h>
#include "param.h"
#include "coeur.h"
#include "cardio.h"

const int ledHeart[10] = {11,2,3,4,5,6,7,8,9,10};
int delayPouls;

void setup()
{
  int delayPouls, pouls;
  int i, led_name = 0;
  const int ledHeart[10] = {11,2,3,4,5,6,7,8,9,10};
  for(i = 0; i<10; i++){pinMode(ledHeart[i], OUTPUT); digitalWrite(ledHeart[i], LOW); // Déclaration de variables
  Serial.begin(9600);
}

void loop()
{
  int delayPouls, pouls;
  int i, led_name;
  unsigned long int temps;
  for(led_name=0;led_name<10;led_name++)
  {
    pouls = recolte(); // Assigne la valeur retournée par la fonction recolte à la variable pouls
    temps = millis(); // Récupère du temps en ms
    temps = temps/1000; // Conversion temps en seconde
    pouls = random(60, 80);

    Serial.print(pouls); // Entre le pouls dans le port Série pour que Processing puisse le lire
    Serial.print("\n"); // le : sert à passer à la case suivante dans un fichier.csv
    Serial.println(temps); // Entre le temps dans le port Série pour que Processing puisse le lire

    delayPouls = (1/(pouls/60))*1000; // Convertir le pouls de sorte à avoir le délai entre chaque battement
    coeur(delayPouls, led_name); // Appel de la fonction coeur
  }
}

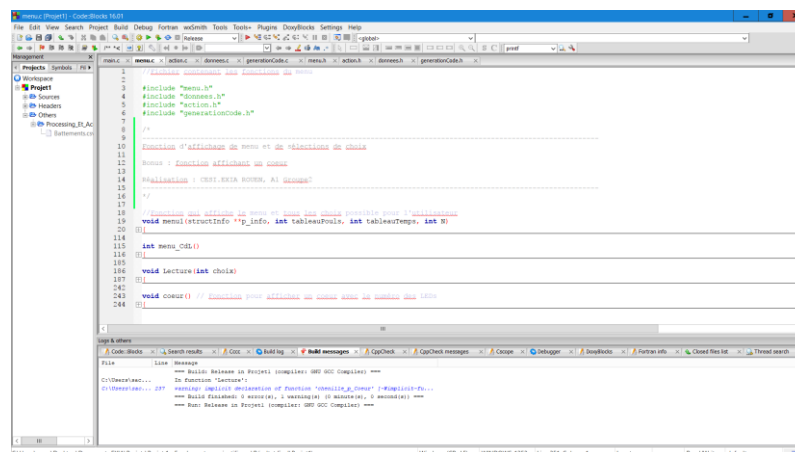
Taille versé : 1000 octets.
Le croquis utilise 1118 octets (94) de l'espace de stockage de programmes. Le maximum est de 32768 octets.
Les variables globales utilisent 236 octets (114) de mémoire dynamique, ce qui laisse 1812 octets pour les variables locales. Le maximum est de 2048 octets.
39
Arduino/GNUmakefile Linux Sur C/C++
```

En programmation C Arduino, nous avons alors liés ensemble les modules **cardio** et **coeur**. En choisissant par logique le fichier **main.c** comme fichier principal. La fonction « *loop()* » est donc dans ce fichier et appelle les différentes fonctions. La fonction « *recolte()* » étant donc une fonction du fichier **cardio.c** et la fonction « *cœur()* », une fonction du fichier **coeur.c**

Comme en C, les fichiers **.h** sont des headers et permettent de nommer la fonction.

Il est à préciser que le montage électronique et le code sont fonctionnels, cependant les LEDS ne semblent pas bien détecter. Afin de simuler un pouls, nous utilisons donc une fonction « *random* ». Le but de celle-ci est de constater l'efficacité des autres modules dépendants du pouls.

Programmation C



```
main | param.h | coeur.h | cardio.h | coeur | param.h

/* Fonction main liée au module cardio, maintenant
 * associée à tous les modules en Arduino
 *
 * Fait appel à la fonction recolte de cardio et
 * aux fonctions du coeur.
 *
 * =====*/

#include <Arduino.h>
#include "param.h"
#include "coeur.h"
#include "cardio.h"

const int ledHeart[10] = {11,2,3,4,5,6,7,8,9,10};
int delayPouls;

void setup()
{
  int delayPouls, pouls;
  int i, led_name = 0;
  const int ledHeart[10] = {11,2,3,4,5,6,7,8,9,10};
  for(i = 0; i<10; i++){pinMode(ledHeart[i], OUTPUT); digitalWrite(ledHeart[i], LOW); // Déclaration de variables
  Serial.begin(9600);
}

void loop()
{
  int delayPouls, pouls;
  int i, led_name;
  unsigned long int temps;
  for(led_name=0;led_name<10;led_name++)
  {
    pouls = recolte(); // Assigne la valeur retournée par la fonction recolte à la variable pouls
    temps = millis(); // Récupère du temps en ms
    temps = temps/1000; // Conversion temps en seconde
    pouls = random(60, 80);

    Serial.print(pouls); // Entre le pouls dans le port Série pour que Processing puisse le lire
    Serial.print("\n"); // le : sert à passer à la case suivante dans un fichier.csv
    Serial.println(temps); // Entre le temps dans le port Série pour que Processing puisse le lire

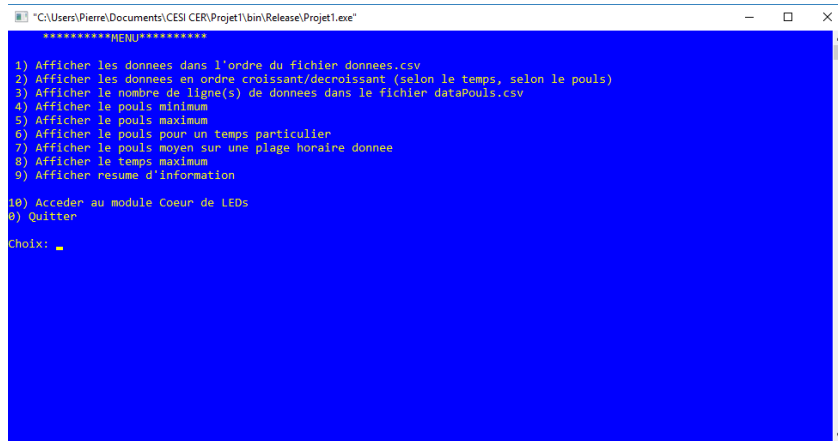
    delayPouls = (1/(pouls/60))*1000; // Convertir le pouls de sorte à avoir le délai entre chaque battement
    coeur(delayPouls, led_name); // Appel de la fonction coeur
  }
}

Taille versé : 1000 octets.
Le croquis utilise 1118 octets (94) de l'espace de stockage de programmes. Le maximum est de 32768 octets.
Les variables globales utilisent 236 octets (114) de mémoire dynamique, ce qui laisse 1812 octets pour les variables locales. Le maximum est de 2048 octets.
39
Arduino/GNUmakefile Linux Sur C/C++
```


Les modules sont aussi liés en programmation C, le **main.c** et le **menu.c** (ainsi que le **menu.h**) accueillent donc les modules « cœur » et « lecture et traitement de données ».

Tandis que les autres fichiers sont dépendants soit du module cœur, soit du module lecture et traitement de données.

Résultat console :

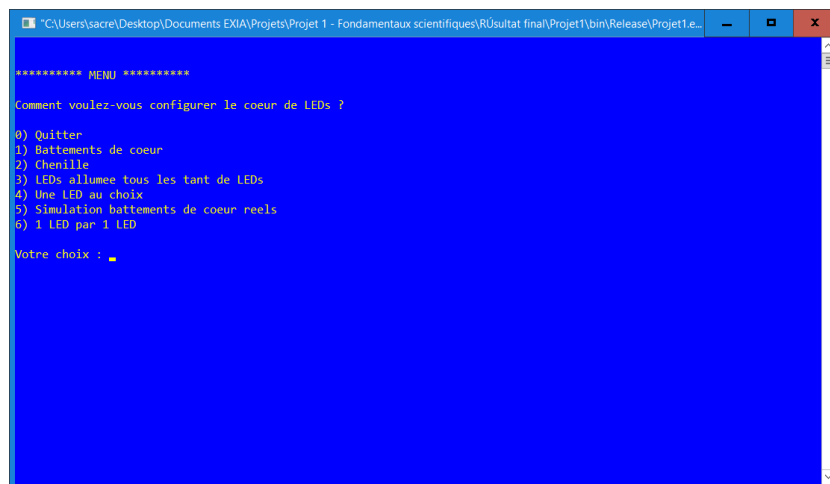


```
"C:\Users\Pierre\Documents\CESI CER\Projet1\bin\Release\Projet1.exe"
*****MENU*****
1) Afficher les donnees dans l'ordre du fichier donnees.csv
2) Afficher les donnees en ordre croissant/decroissant (selon le temps, selon le pouls)
3) Afficher le nombre de ligne(s) de donnees dans le fichier dataPouls.csv
4) Afficher le pouls minimum
5) Afficher le pouls maximum
6) Afficher le pouls pour un temps particulier
7) Afficher le pouls moyen sur une plage horaire donnee
8) Afficher le temps maximum
9) Afficher resume d'information
10) Acceder au module Cœur de LEDs
11) Quitter
Choix: █
```

Sur la console, on peut voir 10 points, 11 si l'on compte le point 0.

- ⇒ Les 9 premiers points sont ceux liés au module de traitement de données, en sachant qu'on a rajouté certaines fonctionnalités vis-à-vis de ce que demandait le projet.
- ⇒ Le dernier point permet d'accéder à un sous-menu gérant le Cœur-de-LED (affiché ci-dessous)

On accède donc à ce menu :



```
"C:\Users\sacre\Desktop\Documents EXIA\Projets\Projet 1 - Fondamentaux scientifiques\RUsultat final\Projet1\bin\Release\Projet1.e...
***** MENU *****
Comment voulez-vous configurer le coeur de LEDs ?
0) Quitter
1) Battements de coeur
2) Chenille
3) LEDs allumee tous les tant de LEDs
4) Une LED au choix
5) Simulation battements de coeur reels
6) 1 LED par 1 LED
Votre choix : █
```

Une fois le choix fait et s'il n'est pas incorrect, la console va alors demander si vous voulez ouvrir Processing et Arduino. Où Processing permet de récupérer les infos et Arduino permet de téléverser les changements.

Il suffira de revenir sur la console et d'appuyer sur la touche « entrée » pour que la console éteigne les programmes.

ORGANISATION EFFECTUEE

Le planning que nous avions n'a pas été totalement bien réalisé, c'est-à-dire que finalement on a dû revoir certains éléments et s'associer pour certaines parties. Au final, le planning réellement réalisé est celui-ci :

PROJET - fondamentaux scientifiques							
Semaine du : novembre 13							
	13/11 LUNDI	14/11 MARDI	15/11 MERCREDI	16/11 JEUDI	17/11 VENDREDI	18/11 SAMEDI	19/11 DIMANCHE
BLIN Clément	Module 1 & 2 partie Arduino	Module 1 & 2 partie Arduino	Module 1 (fin)	Résolution des problèmes MISE EN COMMUN	PREPARATION SOUTENANCE		REVOIR
BOUTIN Pierre	Module 4	Module 4	Module 4	Résolution des problèmes MISE EN COMMUN	PREPARATION SOUTENANCE		REVOIR
BUQUET Dorian	Module 1 & 2 partie électronique	Module 1 & 2 partie électronique	LIVRABLE	Résolution des problèmes MISE EN COMMUN	PREPARATION SOUTENANCE		REVOIR
SOULAS Thomas	Fiche d'avancement	Module 3 & Module 2 partie 3	Module 2 partie 2 (fin et synthétisation du code)	Résolution des problèmes MISE EN COMMUN	PREPARATION SOUTENANCE		REVOIR

C'est-à-dire que Clément s'est occupé de la résolution de certains problèmes mercredi étant donné qu'il avait terminé les autres parties avant. Le planning n'a cependant pas changé vis-à-vis de celui de Dorian et de Thomas.

BILAN DU GROUPE ET PERSONNEL :

Bilan du groupe : Un groupe soudé qui a avancé et qui a essayé de bien s'organiser même si c'était un peu compliqué au début.

Clément : Nous avons rencontré quelques difficultés mais l'équipe est restée soudée. Il y avait une bonne ambiance dans l'équipe. Tout le monde a fait le travail demandé.

Pierre : Je trouve que le groupe a bien fonctionné, la motivation était là. Malgré quelques moments de baisse de moral lors d'échec nous avons su surmonter ces moments. D'un point de vue personnel, réalisant la partie 3.4, je n'ai rencontré des difficultés quand dans la gestion des tris et dans la gestion des « erreurs » pouvant arriver. « Erreur » de type : rentrer une valeur inexistante.

Tout ceci se gère avec la structure de donnée et ici c'était une liste chaînée. Notion que j'ai découverte cette année. Je ne la maîtrise donc pas totalement ceci explique donc les difficultés rencontrées. J'ai d'ailleurs utilisé des tableaux pour les tris après avoir essayé pendant deux jours de faire des tris sur les listes chaînées.

Dorian : L'ambiance de travail du groupe est bonne :

On travaille avec une bonne coordination, travailler sur un même module ensemble avec quelqu'un qui s'occupe d'une partie et l'autre d'une autre partie c'est assez intéressant. Malgré les difficultés, le groupe

a su s'entraider dans certaines parties plus dures. On arrive à se recentrer rapidement lorsqu'on se désintéresse du projet.

Thomas : Un bon groupe, j'ai travaillé sur pas mal de plateformes en même temps, que ce soit l'IDE Arduino, l'IDE Code::Block ou encore Processing et c'était assez sympathique et intéressant.

PISTES D'AMELIORATIONS

- Utiliser des LEDS plus fiables et améliorer le système pour éviter les parasites.
- Faire un montage SolidWorks.
- Mesurer la saturation en oxygène dans le sang.
- Faire une interface graphique pour le programme en C