Livrable 2

Projet : Worldwide Weather Watcher

Gabriel, Benoit, Corentin, Rayane et Thaïs

Sommaire

Préambule :

Rappel du contexte :

Notre société a été contactée pour la création d'un prototype. En effet l'Agence Internationale pour la Vigilance Météorologique (AIVM) souhaite lancer un nouveau projet : embarquer sur leur flotte de navire des stations météo embarquées destinées à surveiller différents paramètres météorologiques parfois responsables de l'apparition de cyclones ou de tempêtes.

Manipulées pas des navigateurs inexpérimentés, les stations météo devront être suffisamment simple d'utilisation et présenté une documentation technique simple.

Après quelques réunions, notre entreprise est sur la bonne voie nous devons maintenant trouver une piste de solution pour que le client puisse comprendre notre démarche. Pour cela il nous demande l'architecture du code qui est intégré à l'Arduino de la station météo ainsi que des explications.

Objectifs du livrable 2 :

Nous devons présenter une architecture de notre code pour cela nous devons créer toute la création des variables, le branchement des capteurs de la carte SD ainsi que les différentes fonctions.

Problématique :

Présenter une architecture simple pour ensuite avoir une vue d'ensemble de notre travaille sur la partie de développement.

Rappel des attentes du client :

Pouvoir embarquer une station météo sur des bateaux. Le but de ce rapport est de fournir une architecture afin de présenter notre piste de réflexion au client.

Acteurs :

-Administrateur / Technicien

-Utilisateur

Modes :

Standard :

- LED verte allumée en continue
- Acquisition des données (hygrométrie, vent, température, pression, GPS) toutes les 10 minutes
- Stockage des données, datées sur la carte SD (mention NA si le capteur ne répond pas sous les 30 secondes après l'appel)

Configuration :

- LED jaune allumée en continue
- Redéfinition de l'intervalle entre 2 deux mesures (cf. standard)
- Taille max fichier avant archivage (carte SD)
- Temps max avant archivage Non Acquis (cf. standard)
- Reset remise des valeurs à défaut des paramètre (ci-dessus)
- Affiche la version du programme et numéro de lot de la station

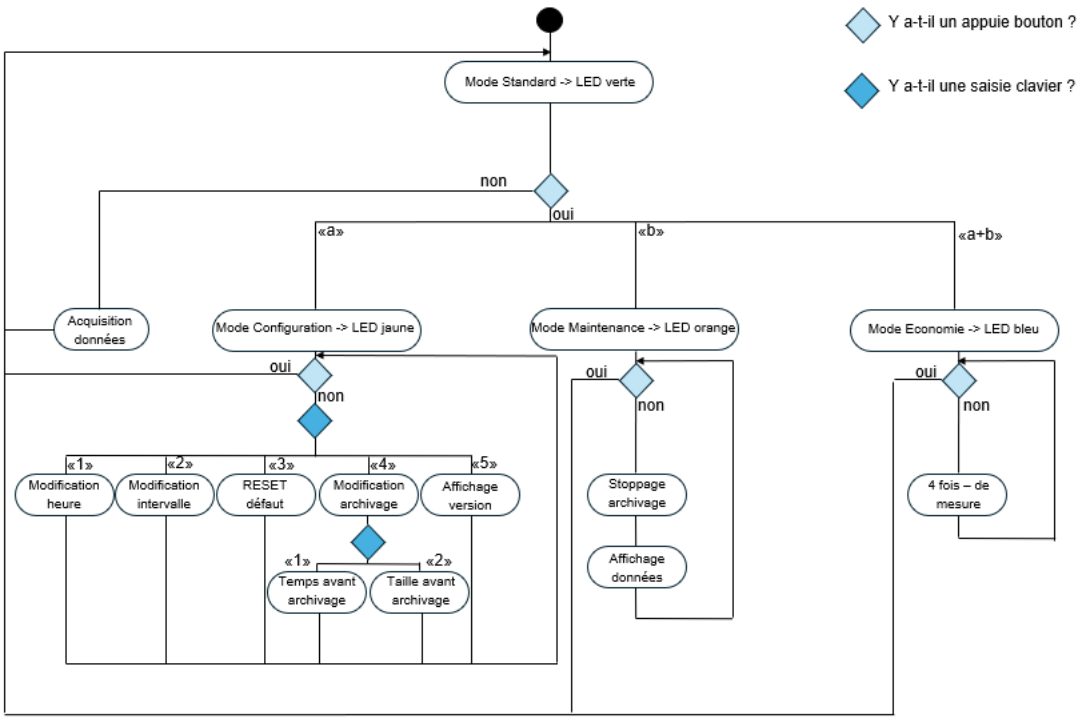
Maintenance :

- LED rouge allumée en continue
- Données affichées en direct et non stockées sur la carte SD

Economie :

- LED bleue allumée en continue
- Temps entre les mesures multiplié par 2, et prise d'une mesure sur deux

En se basant sur le diagramme d'activité réalisé au cours du premier rendu, nous allons réaliser l'architecture du code de notre projet :



Point de vue global :

Afin d'y voir plus clair dans notre projet, nous avons décidé de commencer pour créer une architecture ultra-simplifiée ne contenant que 2 variables globales ainsi que les boucles et noms de fonction.

```
declaration des variables :

declaration des fonctions :

-> choix
-> recup_donnees
-> modif_inter
-> reset
-> modif heure
-> temps_AV_archi
-> taille _AV archi

void setup :

void loop :

if i=0 LED Vert, (Recup_donnees), (stock_donnees)
if i=1 LED Jaune, (Recup_donnees), (stock_donnees)
  if config=1 (Modif_heure)
  if config=2 (Modif_inter)
  if config=3 (Reset)
  if config=4
    if archi=1 (temps_AV_archi)
    if archi=2 (taille_AV_archi)
  if config=5 [version]
if i=2 LED Rouge, (recup_donnees)
if i=3 LED Bleue, (recup_donnees), (stock_donnees)
```

On peut voir ci-contre que le code est minimaliste et ne déclare aucunes variables.

En effet cette petite architecture nous permis de simplifier le tout et de modéliser les 4 modes ainsi que les fonctions qu'ils doivent appeler.

La variable « i » servira tout au long du code d'indicateur. Elle prendra la valeur 0 pour le mode par défaut (standard), la valeur 1 pour le mode qui permet de faire des modifications sur la station (configuration), la valeur 2 pour le mode qui ne stacke rien sur la carte SD (maintenance) et enfin la valeur 3 pour le mode qui réduit le nombre de mesure afin de gagner de la place (économique).

Une fois la variable globale choisie, nous avons réfléchis au fonctionnement complet des changements de mode qui est le suivant :

La fonction "modif_inter" permet de modifier les intervalles entre chaque mesure, comme notamment dans le mode "économique" ou l'on double le temps entre chaque mesure.

La fonction "Reset" permet de remettre la station aux valeurs par défaut.

La fonction "modif_heure" permet de modifier l'heure afficher sur la station (par exemple lors d'un changement de fuseau horaire).

La fonction "temps_AV_archi" permet de modifier le temps avant que les données soient archivées dans la carte SD.

La fonction "taille_AV_archi" permet de modifier la taille définie au fichier avant qu'il ne soit archivé.

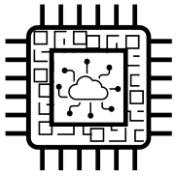
si i = 0 alors on est dans le mode “Standard” on appelle alors les fonctions “recup_donnees” et “stock_donnees” qui va permettre de récupérer les données sur une intervalle choisi et de pouvoir les stocker sur la carte SD .

si i=1 alors on est dans le mode “config” on va pour ici configurer notre station via plusieurs fonctions . (On utilise toujours les fonctions “recup_donnees” et “stock_donnees” pour continuer à récupérer et stocker les informations même si l’on a changé de mode.

- Si on entre 1, on fait appel à la fonction “modif_heure”, on sera donc dans le mode configuration de l’heure de la station.
- Si on entre 2, on fait appel à la fonction “modif_Inter”, on sera donc dans le mode configuration de l’intervalle entre chaque prise de données par les capteurs.
- Si on entre 3, on fait un reset, cela permet de remettre au mode par défaut de la station.
- Si on entre 4, on devra refaire une saisie clavier entre 1 et 2,
    - si 1 est saisi alors on pourra modifier le temps avant l’archivage des données en appelant la fonction “temps_AV_archi”
    - si 2 est saisi alors on pourra modifier la taille maximum du fichier avant l’archivage des données sur la carte SD .
- Si on entre 5, on affiche donc la version du programme et le numéro de lot de la station,

si i=2 alors entre dans le mode maintenance, on fait appel à la fonction “recup_donnees” car on va stopper le stockage des données, on va seulement les afficher en direct .

si i=3 alors on est dans le mode économique (économie d'énergie), on fait appel à la fonction “recup_donnees” et “ stock_donnees” pour pouvoir modifier l’intervalle entre chaque mesure, ici on double le temps entre chaque mesure de données et une mesure sur 2 réalisée .



Architecture du code :

Les bibliothèques :

Utilisation des différentes bibliothèques utile pour ce projet.

```
////////////////PROJET WORLDWIDE WEATHER WATCHER////////////////////
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <Wire.h>
#include <RTClib.h>
#include <DHT.h>
RTC_DS3231 rtc;
```

Les déclarations :

Les variables :

Ici nous prenons le temps de déclarer toutes les variables que nous allons utiliser. Dans ces déclarations nous devons aussi créer les variables/constantes de nos différentes LED, des boutons et de la carte SD dans notre Arduino. Ensuite nous pouvons déclarer les variables globales tel que le « i » ou le TeAA (Temps Avant Archivage).

Pour finis, nous déclarons à la carte Arduino les variables qui sont affiliées aux valeurs des capteurs. Nous utiliserons des floats afin d'avoir une bonne précision des valeurs.

```
////////////// DECLARATION DES VARIABLES, ENTREES ET SORTIES//////////////
//Définition des constantes
const int led_verte = 2;
const int led_jaune = 3;
const int led_rouge = 4;
const int led_bleu = 5;
const int bouton_A = 6;
const int bouton_B = 7;
const int carteSD;
//Définition des variables
bool etatbouton_A;
bool etatbouton_B;
//variables des boucles
int i = 0;
//version du systeme
char version = "Clouduino";
//intervalle entre les mesures
int inter=10
//Temps avant archivage
int TeAA=10
//Temps avant archivage
int TaAA=20000 //en octets
//declaration des capteurs
float capteur_1;
float capteur_2;
float capteur_3;
float capteur_4;
```

Ici nous avons un tableau qui nous montre sur quel port chaque capteur est branché.

Pour les LEDs comme il s'agit de port digitaux nous déclarons les éléments comme des entiers et affilions chaque LED et bouton au port sur lequel il est branché. Pour les capteurs, comme il s'agit de ports analogiques, nous déclarons seulement les floats sans leur donner de valeurs puisque nous ferons les appels plus tard dans le code avec des fonctions spécifiques.

LED vert	2	capteur_1 (GPS)	D4
LED jaune	3	capteur_2 (température)	D3
LED rouge	4	capteur_3 (humidité)	D8
LED bleu	5	capteur_4 (pression)	A4 A5
bouton_A	6	horloge	D2
bouton_B	7		

Les structures :

Ici nous avons la définition d'une structure. On aura alors une liste chainée qui va récupérer les valeurs du capteur GPS. Detaille important chaque capteur a sa liste chainée qui contient ses valeurs.

```
typedef structure Liste_C1{
    float gps;
    struct liste *suivant1;
}Liste_C1;
```

Les Fonctions :

Nous avons ensuite déclaré toutes les fonctions de notre code. Comme seul une architecture était demandée le contenue des fonction à été commenté afin de donner leur fonctionnalité.

Ici la fonction récup donnée récupère les données de chaque capteur et les affichent . Selon la valeur de « i » et donc du mode, recup_donnees ajuste le temps entre chaque mesure et multiplie pas 4 le temps si le mode économique est activé.

```
78  void recup_donnees(){
79      //
80      if i=3{
81          int interbis = inter * 4;
82      }
83      if i != 3{
84          int interbis = inter;
85      }
86      //attendre interbis minutes (intervalle entre chaque mesure);
87      //recuperer et afficher donnees capteur_1;
88      //recuperer et afficher donnees capteur_2;
89      //recuperer et afficher  donnees capteur_3;
90      //recuperer et afficher donnees capteur_4;
91      //stocker données capteur_1 dans liste_c1;
92      //stocker données capteur_2 dans liste_C2;
93      //stocker données capteur_3 dans liste_C3;
94      //stocker données capteur_4 dans liste_C4;
95  }
```

Nous avons de manière identique définit chaque fonction qui est appelée au cours du programme :

```
void stock_donnes (){
    //stocker liste_C1 dans la carte sd;
    //stocker liste_C2 dans la carte sd;
    //stocker liste_C3 dans la carte sd;
    //stocker liste_C4 dans la carte sd;
}
```

La fonction stock_donnees () stock les données dans la carte SD. En effet la mémoire de l'Arduino est très petite. Alors si on veut conserver les listes déjà présente pour les analyser plus tard on doit les enregistrer dans la carte SD qui possède 2Go de stockage.

```
int modif_inter() {
    //modification de la variable globale inter qui définit l'espacement
temporel entre les mesures
}
```

La fonction modif_inter() touche uniquement une variable global elle permet de changer le temps avant deux mesure.

```
int reset(){
    // taille de l'intervalle remise de la valeur par defaut inter =10
    //Temps avant archivage remise de la valeur par defaut TeAA= 10
    //Taille avant archivage remise de la valeur par defaut TaAA= 20000
}
```

Le technicien a la capacité de modifier certain paramètre du système. La fonction reset() lui permet de repasser sur les valeurs par défaut.

```
int modifi_heure(int annee,int mois,int jour,int heure, int min,int sec){
    //modifie l'heure du système selon une demande de l'utilisateur
    // modifie avec les paramètre Année, Mois, Jour, Heure, Minute, Seconde
}
```

La fonction modif_heure (int annee, int mois, int jour, int heure, int min, int sec) modifie avec précision la notion de temps.

```
int temps_AV_archi(){
    //modifie le temps maximal avant l'archivage selon une demande de
l'utilisateur
}
int taille_AV_archi(){
    //modifie la taille maximale avant l'archivage selon une demande de
l'utilisateur
}
```

Les fonction taille_AV_archi() et temps_AV_archi() modifie des paramètre d'archivage sur la carte SD

La fonction Interrupt et la Void Setup :

Les Interrupt :

Ain de pouvoir changer de mode quel que soit le moment dans le code et utiliser des boucles sans fon sans tomber dedans, il est nécessaire de créer un « Interrupt ». Cette fonction ainsi que son appel dans le Setup du code garanti la possibilité de changer de mode quelque soit le moment.

```
144   //setup de l interrupt
145   attachInterrupt(digitalPinToInterrupt(bouton_A), Choix, FALLING);
146   attachInterrupt(digitalPinToInterrupt(bouton_B), Choix, FALLING);
```

Ces deux ligues situées dans la Void Setup du code montrent le déclanchement de la fonction choix lorsque le bouton A (ou B en fonction de la ligne) sont appuyés. Un seul appuie est nécessaire même lorsque le code est dans la boucle configuration ou en train de proposer un menu à l'écran.

La partie « digitalPinToInterrupt (bouton_A) » donne l'élément déclencheur de l'interrupt. Le nom de la fonction qui suit est celle qui est déclenchée lorsque de m'interruption du code ici il s'agit de la fonction choix.

```
void Choix (){
    if (i == 0)
    {
      digitalread(Bouton_A);
      digitalread(Bouton_B);
      if (Bouton_A == “HIGH” and Bouton_B == “LOW”);
        i = 1;
      if (Bouton_A == “LOW” and Bouton_B == “HIGH”);
        i = 2;
      if (Bouton_A == “HIGH” and Bouton_B == “HIGH”);
        i = 3;
    if i!=0
      i = 0;
    }
}
```

Dans notre code nous avons fait le choix lorsqu’il y a un appuie bouton et donc, par conséquent une interruption, d’analyser la nature du ou des boutons pressés afin de passer d’un mode à un autre. Lorsque le système se trouve dans le mode Standard et donc « i » vaut 0 il est possible d’appuyer sur le bouton A afin de se connecter au mode Configuration, le bouton B au mode Maintenance et enfin appuyer sur les 2 en même temps pour accéder au mode Economique. Si un ou des boutons sont pressés alors que le système est déjà dans un mode et donc « i » est différents de 0 alors le système réouvre automatiquement le mode standard. Il est donc nécessaire de revenir au mode standard pour passer d’un mode Configuration à Maintenance par exemple.

Les entrées sorties :

```
127     //OUTPUT : sortie d'informations
128     pinMode(led_verte, OUTPUT);
129     pinMode(led_jaune, OUTPUT);
130     pinMode(led_rouge, OUTPUT);
131     pinMode(led_bleue, OUTPUT);
132     // INPUT : entrée dans le systeme
133     pinMode(bouton_A, INPUT);
134     pinMode(bouton_B, INPUT);
```

Il est aussi nécessaire de donner au code les différents moyens de communication avec l’utilisateur. Les output /input, ou entrée/sortie sont indispensable au bon fonctionnement de notre code.

Les variables des LEDs comme led_verte contenant le numéro du port sur lequel elle est branchée, il est simple de définir la sortie. On sait que led_verte est un entier qui vaut 2 donc le code prend en compte 2 comme un port de sorti d’information mais l’écriture du mot « led_verte » à la place dans la code rend la compréhension plus facile.

L’horloge :

```
139     if (!rtc.begin()) { //test si l’horloge est correctement fonctionnelle
140         Serial.println("Erreur de connexion au RTC");
141         while (1);{
142             if (rtc.lostPower()) { // Si l'heure n'est pas réglée, la définir à la compilation
143                 Serial.println("RTC a perdu l'alimentation, réglage de l'heure !");
144     rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));//Ajustement de l’heure
145             }
146         }
147     }
```

Dans la Void Setup on retrouve aussi la mode à 0 et la configuration de l'horloge. On fait un rapide test ligne 139 pour savoir si l'horloge communique bien avec le système.

Ensuite on doit définir l'heure cette étape doit être fait lors de la compilation.

La Void Loop :

Notre Void Loop est un élément essentiel de notre code. Enfin c'est cette partie qui régit tous les lancements de mode à l'aide de la variable « i » ainsi que tous les appels de fonction. Comme répété plus haut, « i » contient l'information sur quel mode est en cours de fonctionnement ainsi chaque valeur de « i » déclenche des appels et des affichages différents :

Exemple avec le mode configuration :

```
159     if i = 1{
160         serial.println ("Vous etes en mode configuration.")
161         //allumage de la led selon la couleur du mode ici jaune pour le mode configuration
162         recup_donnees();//appel de la fonction
163         stock_donnes();//appel de la fonction
164         // affichages des fonctionnalités du mode
165         serial.println ("Que voulez vous faire ?")
166         serial.println ("Entrez 1 pour modifier l'heure de la station météo.")
167         serial.println ("Entrez 2 pour modifier l'intervalle")
168         serial.println ("Entrez 3 pour reset au mode par défaut")
169         serial.println ("Entrez 4 pour modifier l'archivage")
170         serial.println ("Entrez 5 pour afficher version")
```

Lors du démarrage du mode les fonctions de récupération et stockage des données sont appelées car le mode continue les mesures sans modification par rapport au mode standard. On peut aussi constater qu'un menu est affiché. Celui-ci informe l'utilisateur des fonctionnalités qu'offre le mode.

```
171         int confi =Serial.read();
172         if confi = 1{
173             modif_heure();//appel de la fonction
174         }
175         if confi = 2{
176             modif_inter();//appel de la fonction
177         }
178         if confi = 3{
179             reset();//appel de la fonction
180         }
181         if confi = 4{
182             serial.println ("Entrez 1 pour modifier le temps avant larchivage des données sur la carte SD.")
183             serial.println ("Entrez 2 pour modifier la taille avant larchivage des données sur la carte SD.")
184             int archi =Serial.read();
185             if archi = 1{
186                 temps_AV_archi();//appel de la fonction
187             }
188             if archi = 2{
189                 taille_AV_archi();//appel de la fonction
190             }
191         }
192         if confi = 5{
193             serial.println(version);//affiche la version de la station
194         }
195     }
```

Une réponse est attendue de l'utilisateur après l'affichage du menu. L'entrée qu'il va alors faire appellera une fonction ou fera un affichage adapter au besoin de l'utilisateur. En cas d'erreur de frappe il faudra revenir au mode standard et recommencer la manipulation dans le mode configuration.

On fait la même chose pour les 3 modes restants :

Le mode standard :

```
153     if i = 0{
154         serial.println ("Vous etes en mode standard.")
155         //allumage de la led selon la couleur du mode ici vert pour le mode defaut
156         recup_donnees();//appel de la fonction
157         stock_donnes();//appel de la fonction
158     }
```

Ainsi que les 2 modes Maintenance et configuration :

```
196     if i = 2{
197         serial.println ("Vous etes en mode maintenance.")
198         //allumage de la led selon la couleur du mode ici rouge pour le mode maintenance
199         recup_donnees();//appel de la fonction
200     }
201     if i = 3{
202         serial.println ("Vous etes en mode economique.")
203         //allumage de la led selon la couleur du mode ici bleu pour le mode economique
204         recup_donnees();//appel de la fonction
205         stock_donnes();//appel de la fonction
206     }
```

Conclusion :

L’architecture de notre code se décompose en 4 zones distinctes :

- La déclaration des variables :

Dans cette section, on définit toutes les variables globales donc le code va se servir. On y retrouve des variables du code mais aussi les boutons, LEDs, ainsi que les structures qui sont utilisés à plusieurs endroits du code.

- La déclaration des fonctions :

Les différentes fonctions avant d’être appelées doivent être déclarées. Certaines sont typées ou non mais toutes abriteront dans le rendu final une fonctionnalité indispensable au bon fonctionnement du code.

- La Void Setup :

Cette boucle initialise certains mécanismes du code comme les Interrupts, l’horloge ou les entrées/sorties. Cette boucle réalise une dernière initialisation avant que la boucle principale du code ne se lance : la Void Loop.

- La Void Loop

Cette partie s’exécute en continue. Elle vérifie en continue l’état de la variable globale i. De cette façon on peut selon l’état de i , savoir quel action notre code doit mettre en place. Exemple si on appuie sur un bouton la fonction choix() change l’état de i. ensuite le Loop analyse en continue i .si i vaut 1 alors nous rentrons dans le mode configuration.

Dans ce rendu, nous mettons en place les bases de notre futur code complet. En effet, l’architecture est comme les fondations sur lequel notre code se base. La déclaration de nos fonctions et la mise en place de nos boucles étant fait nous allons pouvoir compléter le contenue des différentes fonctions qui vont rendre notre code fonctionnel.

Annexe :

Architecture complète du projet :

```
////////////////PROJET WORLDWIDE WEATHER WATCHER////////////////////
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <Wire.h>
#include <RTClib.h>
#include <DHT.h>
RTC_DS3231 rtc;


////////////// DECLARATION DES VARIABLES, ENTREES ET SORTIES//////////////
//Définition des constantes
const int led_verte = 2;
const int led_jaune = 3;
const int led_rouge = 4;
const int led_bleu = 5;
const int bouton_A = 6;
const int bouton_B = 7;
const int carteSD;
//Définition des variables
bool etatbouton_A;
bool etatbouton_B;
//variables des boucles
int i = 0;
//version du systeme
char version = "Clouduino";
//intervalle entre les mesures
int inter=10
//Temps avant archivage
int TeAA=10
//Temps avant archivage
int TaAA=20000//nombre d'octets
//declaration des capteurs
int capteur_1;
int capteur_2;
int capteur_3;
int capteur_4;



/////tableau de stockage des mesures sous forme de listes chainees/////
//mesures du capteur gps
typedef structure Liste_C1{
    float gps;
    struct liste *suivant1;
}Liste_C1;
// mesures du capteur de température
typedef structure Liste_C2{
    float temp;
    struct liste *suivant2;
}Liste_C2;
// mesures du capteur d'humidité
typedef structure Liste_C3{
    float humi;
    struct liste *suivant3;
}Liste_C3;
// mesures du capteur de pression
typedef structure Liste_C4{
    float pression;
    struct liste *suivant4;
}Liste_C4;


////////////////////////////fonction/////////////////////////////
void Choix (){
```

```
    if (i == 0)
    {
      digitalread(Bouton_A);
      digitalread(Bouton_B);
      if (Bouton_A == “HIGH” and Bouton_B == “LOW”);
        i = 1;
      if (Bouton_A == “LOW” and Bouton_B == “HIGH”);
        i = 2;
      if (Bouton_A == “HIGH” and Bouton_B == “HIGH”);
        i = 3;
    if i!=0
      i = 0;
    }
}
void recup_donnees(){
    //
    if i=3{
        int interbis = inter * 4;
    }
    if i != 3{
        int interbis = inter;
    }
    //attendre interbis minutes (intervalle entre chaque mesure);
    //recuperer et afficher donnees capteur_1;
    //recuperer et afficher donnees capteur_2;
    //recuperer et afficher  donnees capteur_3;
    //recuperer et afficher donnees capteur_4;
    //stocker données capteur_1 dans liste_c1;
    //stocker données capteur_2 dans liste_C2;
    //stocker données capteur_3 dans liste_C3;
    //stocker données capteur_4 dans liste_C4;
}
void stock_donnes (){
    //stocker tab_c1 dans la carte sd;
    //stocker tab_c2 dans la carte sd;
    //stocker tab_c3 dans la carte sd;
    //stocker tab_c4 dans la carte sd;
}
int modif_inter() {
    //modification de la variable globale inter qui définit l’espacement temporel entre les mesures
}
int reset(){
    // taille de l'intervalle remise de la valeur par defaut inter =10
    //Temps avant archivage remise de la valeur par defaut TeAA= 10
    //Taille avant archivage remise de la valeur par defaut TaAA=   20000
}
int modifi_heure(int annee,int mois,int jour,int heure, int min,int sec){
    //modifie l’heure du système selon une demande de l’utilisateur
    // modifie avec les parametre Année, Mois, Jour, Heure, Minute, Seconde + verif la bibi pour la fct adjust
}
int temps_AV_archi(){
    //modifie le temps maximal avant l’archivage selon une demande de l’utilisateur
}
int taille_AV_archi(){
    //modifie la taille maximale avant l’archivage selon une demande de l’utilisateur
}


/////////////////VOID SETUP INITIALISATION DU SYSTEME/////////////////
void setup()
{
    //OUTPUT : sortie d'informations
    pinMode(led_verte, OUTPUT);
    pinMode(led_jaune, OUTPUT);
    pinMode(led_rouge, OUTPUT);
    pinMode(led_bleue, OUTPUT);
    // INPUT : entrée dans le systeme
    pinMode(bouton_A, INPUT);
    pinMode(bouton_B, INPUT);
    // Si l'heure n'est pas réglée, la définir à la compilation
    //declare l’horloge rtc dans le setup
    if (!rtc.begin()) { //test si l’horloge est correctement fonctionnelle
        Serial.println("Erreur de connexion au RTC");
        while (1);{
            if (rtc.lostPower()) { // Si l'heure n'est pas réglée, la définir à la compilation
```

```
                    Serial.println("RTC a perdu l'alimentation, réglage de l'heure !");
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));//Ajustement de l'heure
                }
            }
        }
  //setup des interrupts
  attachInterrupt(digitalPinToInterrupt(bouton_A), Choix, FALLING);
  attachInterrupt(digitalPinToInterrupt(bouton_B), Choix, FALLING);
}

//////////////////////VOID LOOP, CODE PRINCIPAL//////////////////////
void loop(){
    if i = 0{
        serial.println ("Vous etes en mode standard.")
        //allumage de la led selon la couleur du mode ici vert pour le mode defaut
        recup_donnees();//appel de la fonction
        stock_donnes();//appel de la fonction
    }
    if i = 1{
        serial.println ("Vous etes en mode configuration.")
        //allumage de la led selon la couleur du mode ici jaune pour le mode configuration
        recup_donnees();//appel de la fonction
        stock_donnes();//appel de la fonction
        // affichages des fonctionnalités du mode
        serial.println ("Que voulez vous faire ?")
        serial.println ("Entrez 1 pour modifier l'heure de la station météo.")
        serial.println ("Entrez 2 pour modifier l'intervalle")
        serial.println ("Entrez 3 pour reset au mode par défaut")
        serial.println ("Entrez 4 pour modifier l'archivage")
        serial.println ("Entrez 5 pour afficher version")
        int confi =Serial.read();
        if confi = 1{
            modif_heure();//appel de la fonction
        }
        if confi = 2{
            modif_inter();//appel de la fonction
        }
        if confi = 3{
            reset();//appel de la fonction
        }
        if confi = 4{
            serial.println ("Entrez 1 pour modifier le temps avant larchivage des données sur la carte SD.")
            serial.println ("Entrez 2 pour modifier la taille avant larchivage des données sur la carte SD.")
            int archi =Serial.read();
            if archi = 1{
                temps_AV_archi();//appel de la fonction
            }
            if archi = 2{
                taille_AV_archi();//appel de la fonction
            }
        }
        if confi = 5{
            serial.println(version);//affiche la version de la station
        }
    }
    if i = 2{
        serial.println ("Vous etes en mode maintenance.")
        //allumage de la led selon la couleur du mode ici rouge pour le mode maintenance
        recup_donnees();//appel de la fonction
    }
    if i = 3{
        serial.println ("Vous etes en mode economique.")
        //allumage de la led selon la couleur du mode ici bleu pour le mode economique
        recup_donnees();//appel de la fonction
        stock_donnes();//appel de la fonction
    }
}
```