

# Livrable 3

## Projet : Worldwide Weather Watcher



**Gabriel BAGLIVO, Benoit MICOULAS,  
Corentin PERES, Rayane OULDALI et  
Thaïs VAINES**

# Sommaire

<b>Préambule .....</b>	<b>3</b>
Rappel du contexte .....	3
Objectifs du livrable 3 .....	3
Problématique .....	3
Rappel des attentes du client .....	3
Acteurs .....	4
Modes .....	4
Point de vue global.....	6
Programme finalisé .....	8
Les bibliothèques .....	8
Les déclarations .....	8
Les variables .....	8
Les constantes .....	9
Les structures .....	9
Les Fonctions.....	10
La fonction Interrupt et la Void Setup .....	14
Les Interrupt.....	14
Les entrées sorties.....	15
L'horloge .....	15
La carte SD .....	15
Le capteur .....	15
La Void Loop.....	16
Calcul de la complexité .....	17
Gestion de la mémoire .....	19
Gestion de fichiers et stockage.....	20
Guide d'utilisation .....	22
Utilisation de la station météo Cloduino.....	23
Problématiques courantes.....	24
Conclusion .....	26
Annexe.....	27

## 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 par des navigateurs inexpérimentés, les stations météo devront être suffisamment simples d'utilisation et présenter une documentation technique simple.

Après quelques réunions, notre entreprise est sur la bonne voie, nous avons trouvé une piste de solution pour que le client puisse comprendre notre démarche. Celle-ci ayant été validée par le chef de projet et la structure du code approuvée. Nous allons maintenant finir le code en C Arduino et créer une maquette afin de la présenter à notre client.

### Objectifs du livrable 3

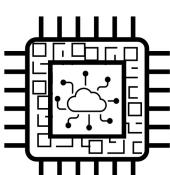
Nous devons présenter un code complet de la station météo. Ce code doit respecter des règles de gestion de fichiers et être le moins complexe possible afin de rentrer dans la mémoire de la carte Arduino de la station.

De plus, nous devons rédiger une documentation technique du projet à destination des techniciens ainsi qu'un guide d'utilisation de la station météo afin que les utilisateurs (qui n'ont pas de formation) puissent s'en servir sans soucis.

### Problématique

Comment pouvons-nous réaliser le code de la station météo en passant outre les difficultés techniques qu'il induit ?

### Rappel des attentes du client



Pour finaliser notre projet, il ne nous manque qu'une seule étape : réaliser le code complet de la station afin de la rendre fonctionnelle.

## Acteurs

- Administrateur / Technicien
- Utilisateur

## Modes

### Standard

- LED verte allumée en continu
- 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

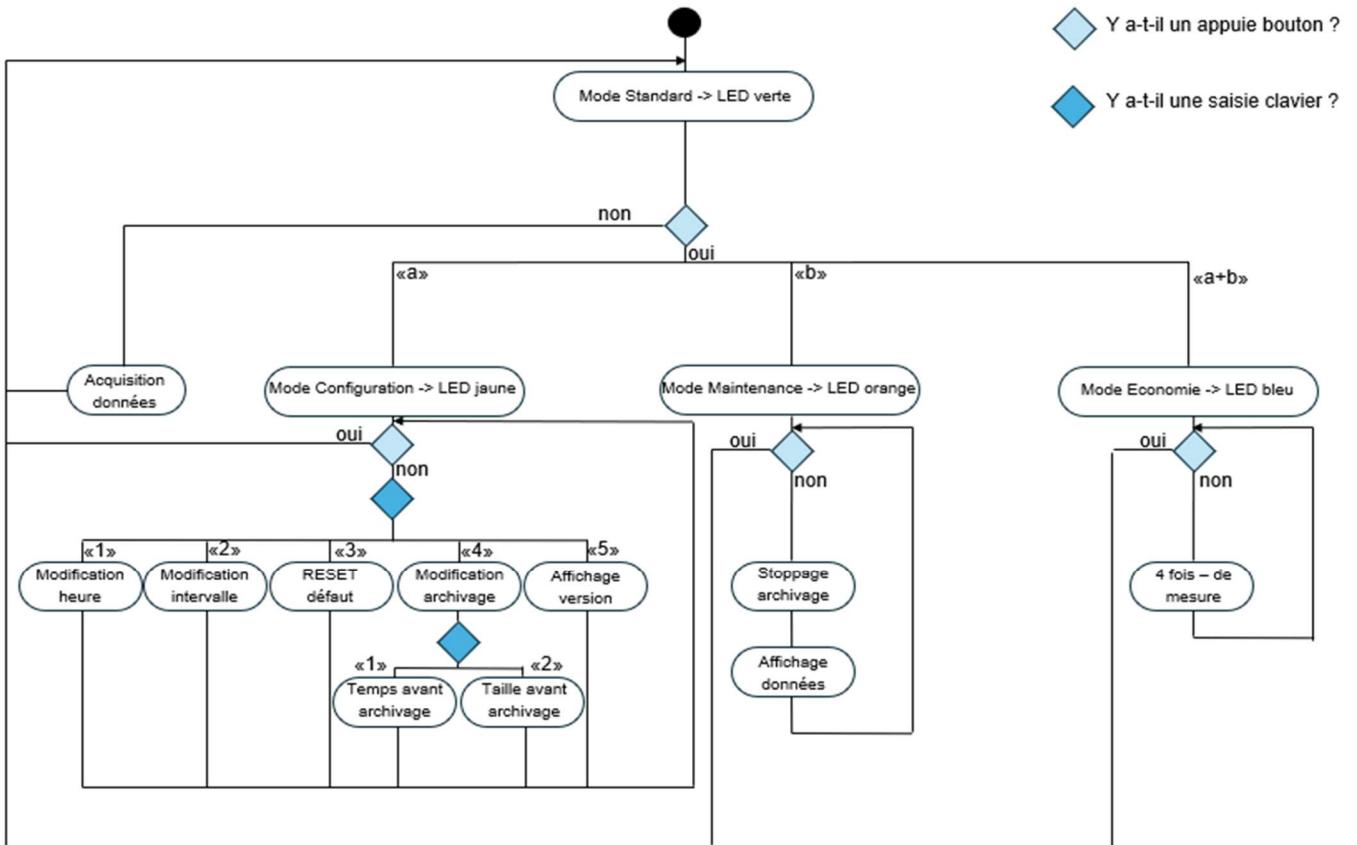
- LED jaune allumée en continu
- Redéfinition de l'intervalle entre deux mesures (cf. standard)
- Taille max fichier avant archivage (carte SD)
- Temps max avant archivage Non Acquis (cf. standard)
- Reset remise des valeurs à défaut des paramètres (ci-dessus)
- Affiche la version du programme et numéro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

### Maintenance

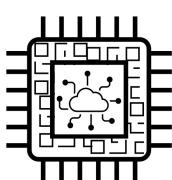
- LED rouge allumée en continu
- Données affichées en direct et non stockées sur la carte SD

### Economie

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



Rappel du fonctionnement de la station décidé lors du premier livrable.



# Point de vue global

Afin d'y voir plus clair dans notre projet, nous avons décidé de commencer pour créer une structure ultra-simplifiée ne contenant que 3 variables globales ainsi que les boucles et noms de fonction.

```
1 declaration des variables :  
2  
3 declaration des fonctions :  
4  
5 void setup :  
6  
7 void loop :  
8  
9 if i=0 LED Vert, (Recup_donnees), (stock_donnees)  
10 if i=1 LED Jaune, (Recup_donnees), (stock_donnees)  
11   if config=1 (Modif_heure)  
12   if config=2 (Modif_inter)  
13   if config=3 (Reset)  
14   if config=4  
15     if archi=1 (temps_AV_archi)  
16     if archi=2 (taille_AV_archi)  
17     if config=5 [version]  
18 if i=2 LED Rouge, (recup_donnees)  
19 if i=3 LED Bleue, (recup_donnees), (stock_donnees)
```

Mini structure du code

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

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

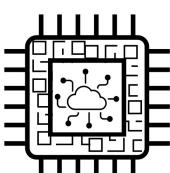
La variable « i » servira tout au long du code d'indicateur. Elle prendra la valeur 0 pour le mode par défaut (standard), la valeur 1 pour le mode qui permet de faire des modifications sur la station (configuration), la valeur 2 pour le mode qui ne stocke rien sur la carte SD (maintenance) et enfin la 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 réfléchi au fonctionnement complet des changements de mode qui est le suivant :

Si  $i = 0$  alors on est dans le mode “Standard” on appelle alors les fonctions “recup\_donnees” et “stock\_donnees” qui vont permettre de récupérer les données sur un intervalle choisi et de pouvoir les stocker sur la carte SD.

Si  $i=1$  alors nous sommes dans le mode “config”. Notre station sera configurée 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.

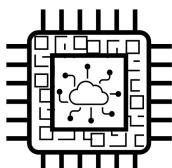


- Si on entre 2, on fait appel à la fonction “modif\_Inter”, on sera donc dans le mode configuration de l'intervalle entre chaque prise de données par les capteurs.
- Si on entre 3, on fait un reset, cela permet de remettre au mode par défaut de la station.
- Si on entre 4, on devra refaire une saisie 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 est réalisée.

Afin de changer la valeur de «  $i$  » des boutons doivent être appuyés selon un ordre précis. L'explication plus détaillée de l'utilisation sera réalisée dans le chapitre « Guide d'utilisation » en page n°20.



# Programme finalisé

## Les bibliothèques

```
3 ////////////// LIBRAIRIES///////////
4 #include <stdio.h>//gestion du programme (Print)
5 #include <string.h>//gestion des variables (Char, Listes...)
6 #include <stdlib.h>//gestion du programme
7 #include <Wire.h>//gestion du programme (Connection filaire avec des capteurs)
8 #include <SD.h>//carte SD
9 #include <SPI.h>//carte SD
10 #include <RTClib.h>//horloge
11 #include <Adafruit_Sensor.h>//capteur de température, humidité, pression
12 #include <Adafruit_BME280.h>//capteur de température, humidité, pression
```

Déclaration des bibliothèques

Afin que notre code fonctionne correctement, il est impératif que certaines bibliothèques soient importées. En effet, celles-ci permettent au code d'accéder à des fonctionnalités qu'il ne pourrait pas utiliser sans cela, ou réaliser des calculs plus complexes comme en témoigne les bibliothèques « Adafruit\_Sensor » ou « Adafruit\_BME280 » qui permettent d'utiliser des fonctionnalités reposant sur le capteur BME\_280 qui réalise la mesure de la pression, de la température ainsi que de l'humidité.

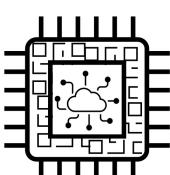
## Les déclarations

### Les variables

```
47 // Intervalle entre les mesures
48 int inter = 3000;
49 // Temps avant archivage
50 int TeAA = 30000;
51 // Taille avant archivage
52 int TaAA = 6;
53 // Déclaration des capteurs
54 float capteur_1;
55 float capteur_2;
56 float capteur_3;
57 float capteur_4;
```

Les variables tel que les intervalles de temps ou les valeurs lues des capteurs sont une partie essentielle de notre code. En effet, déclarées en global, elles sont accessibles depuis plusieurs fonctions et peuvent ainsi être modifiées, ajustées ou utilisées depuis n'importe quelle fonction ou boucle.

Déclaration des variables



## Les constantes

Composant	Port	Composant	Port
LED vert	5	capteur_1 (GPS)	D4
LED jaune	6	capteur_2 (température)	A4 / A5
LED rouge	7	capteur_3 (humidité)	A4 / A5
LED bleu	9	capteur_4 (pression)	A4 / A5
bouton_A	3	horloge	D2
bouton_B	2	Carte SD	4

```
28 // Définition des constantes
29 const int led_verte = 5;
30 const int led_jaune = 6;
31 const int led_rouge = 7;
32 const int led_bleue = 9;
33 const int bouton_A = 3;
34 const int bouton_B = 2;
35 const int carteSD = 4;
```

Déclaration des constantes

Chaque composant du circuit électronique est défini par le pin sur lequel il est branché. Celui-ci va pouvoir envoyer un signal ou récupérer un signal selon le fonctionnement du code. Chaque numéro de pin n'est pas amené à changer au cours du code, pour cette raison nous le déclarons en variables globales mais comme des constantes.

La carte SD est aussi déclarée comme un pin. Même si elle est située dans son port, le code la verra comme branchée sur le pin 4 ou pourra s'en servir comme un composant classique.

## Les structures

```
// Mesures du capteur de température
typedef struct Liste_C2 {
    float temp;
    struct Liste_C2 *suivant2;
} Liste_C2;
```

Les structures elle permettent la création de listes chainées. Ces listes chainées seront utiles pour stocker les données des capteurs (température, pression, humidité etc.). L'avantage des structures, c'est que ces dernières prennent moins de place dans la mémoire. De plus contrairement au tableau en C les structures sont dynamiques. Point important, chaque capteur possède sa propre liste chainée.

## Les Fonctions

**void choix()** La fonction choix est l'atout majeur de notre station. En effet, elle permet les changements de mode en temps réel.

Son fonctionnement est simple. Une autre fonction déclenchée par les interrupts joue le rôle d'activateur. En effet, cette autre fonction plus simple et déclenchée lors de l'appui d'un bouton change la valeur de newMode qui est un booléen de false à true. Si ce booléen vaut true, cela signifie que l'interrupt a été enclenché et donc qu'il faut changer de mode. La fonction choix qui tourne alors dans la void loop se déclenche et analyse les boutons pressés.

```
if (newMode) {  
    bool etatbouton_A = digitalRead(bouton_A) == LOW;  
    bool etatbouton_B = digitalRead(bouton_B) == LOW;
```

La suite de la fonction regarde la valeur du mode actuel. Si le mode actuel est standard et donc « i=0 » la combinaison de boutons pressés déclenche un mode : configuration, maintenance ou économique.

Une fois la valeur de « i » changée et les print concernant le mode affiché, la fonction remet la valeur newMode à false afin qu'elle soit à nouveau activée lors du prochain interrupt.

```

if (i==0){
    // Vérifier d'abord si les deux boutons sont pressés
    if (etatbouton_A && etatbouton_B) {
        i = 3; // Mode éco
        Serial.println(F("Vous êtes en mode économique."));
    }
    // Ensuite, vérifier si seul le bouton A est pressé
    else if (etatbouton_A) {
        i = 1; // Mode config
        Serial.println(F("Vous êtes en mode configuration."));
        Serial.println(F("Entrez 1 pour modifier l'heure de la station météo."));
        Serial.println(F("Entrez 2 pour modifier l'intervalle."));
        Serial.println(F("Entrez 3 pour reset au mode par défaut."));
        Serial.println(F("Entrez 4 pour modifier l'archivage."));
        Serial.println(F("Entrez 5 pour afficher la version."));
    }
    // Ensuite, vérifier si seul le bouton B est pressé
    else if (etatbouton_B) {
        i = 2; // Mode maintenance
        Serial.println(F("Vous êtes en mode maintenance."));
    }
    // Sinon, revenir au mode standard
}
else {
    i = 0; // Mode standard
    Serial.println(F("Vous êtes de retour en mode standard."));
}
newMode = false;
}
}

```

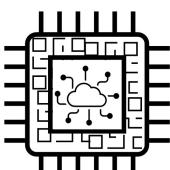
**void recuper\_donnees()** La fonction recuper\_donnees () permet d'ajouter des données dans les listes chainées.

```

capteur_2 = bme.readTemperature();
capteur_3 = bme.readHumidity();
capteur_4 = bme.readPressure();

```

Ici nous mettons dans les variables capteur\_n (n compris entre 1 et 4) une mesure prise par le capteur, nous adaptons ensuite la commande pour prendre la température ou alors le pourcentage d'humidité.



```
Liste_C2* elt2 = (Liste_C2*)malloc(sizeof(Liste_C2)); // Allouer de la mémoire pour un nouvel élément
elt2->temp = capteur_2;
elt2->suivant2 = NULL;
```

Ce bout de code permet de rajouter un nouvel élément dans notre liste chainée, Il fait en sorte que l'élément suivant pointe vers NULL qui sera la fin de notre liste.

```
if (tete2 == NULL) {
    tete2 = elt2; // Si la liste est vide, le nouvel élément devient la tête
} else {
    Liste_C2* current2 = tete2;
    while (current2->suivant2 != NULL) { // Parcourir la liste jusqu'au dernier élément
        current2 = current2->suivant2;
    }
    current2->suivant2 = elt2; // Ajouter le nouvel élément à la fin
```

A cet instant nous devons placer l'élément en question, On distingue alors deux cas

Premier cas la liste est vide, l'élément ajouté devient la tête de la liste

Le second cas est utilisé lorsque qu'il existe une tête déjà existante. Dans ce cas on parcourt la liste et on rajoute l'élément à la fin de la liste.

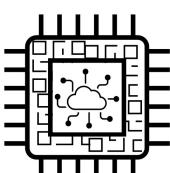
```
unsigned long current2Millis = millis();
int interbis;
// Ajuster l'intervalle en fonction du mode
if (i == 3) {
    interbis = inter * 4;
} else {
    interbis = inter;
}
```

**void stock\_donnees()** Lorsqu'elle est déclenchée, la fonction stock\_donnees, tourne dans la void loop jusqu'à répondre à la condition du premier if de la boucle. Celui-ci donne les conditions requises pour un envoi de données à savoir une taille ou un temps supérieur à la limite fixée par le reset ou par l'utilisateur.

```
unsigned long currentMillis = millis();
// Vérifier si la taille d'une des listes dépasse TaAA ou si TeAA temps est écoulé
if (checkSize(tete1) >= TaAA || checkSize(tete2) >= TaAA || checkSize(tete3) >= TaAA || checkSize(tete4) >= TaAA ||
|| (currentMillis - lastArchiveTime) >= TeAA) {}
```

Si les conditions sont remplies pour l'envoie des données sur la carte SD et la destruction des données en local, alors le code crée un fichier .txt sur la carte SD afin d'y ranger les données horodatées.

```
File dataFile = SD.open("datalog.txt", FILE_WRITE);
DateTime now = rtc.now();
if (dataFile) {
```



La liste chaînée de stockage des données locales est alors parcourue, pour chaque valeur, une date (année, mois, jours) ainsi qu'une horaire (heures, minutes, secondes) est ajoutée de manière à avoir une continuité dans les données stockées.

```
Liste_C2* current2 = tete2;
while (current2 != NULL) {
    dataFile.print("Date/Heure: ");
    dataFile.print(now.year(), DEC);
    dataFile.print('/');
    dataFile.print(now.month(), DEC);
    dataFile.print('/');
    dataFile.print(now.day(), DEC);
    dataFile.print(" - ");
    dataFile.print(now.hour(), DEC);
    dataFile.print(':');
    dataFile.print(now.minute(), DEC);
    dataFile.print(':');
    dataFile.print(now.second(), DEC);
    dataFile.print(" | Température : ");
    dataFile.println(current2->temp);
    current2 = current2->suivant2;
}
```

On peut voir une fois les données envoyées et la carte SD retirée du système que les données archivées sont bien identiques à celles captées au cours du fonctionnement de la station.

SDHC (D:)				Rechercher dans : SDHC (D:)
				Trier ▾ Afficher ▾ Éjecter ... Détails
Nom	Modifié le	Type	Taille	

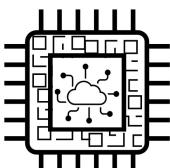
DATALOG.TXT 01/01/2000 00:00 Document texte 9 Ko

Fichier créé dans la carte SD

Coordonnées GPS : Introuvables. Archivage NA. Température actuelle : 22.69°C Humidité actuelle : 69.88% Pression actuelle : 102292.11Pa Données écrites sur la carte SD.	Date/Heure: 2024/10/24 - 19:24:46   GPS lat, long : 0.00 Date/Heure: 2024/10/24 - 19:24:46   GPS lat, long : 0.00 Date/Heure: 2024/10/24 - 19:24:46   Température : 22.69 Date/Heure: 2024/10/24 - 19:24:46   Humidité : 69.88 Date/Heure: 2024/10/24 - 19:24:46   Pression : 102292.11
--	---

Comparaison entre les données stockées dans le fichier et celles captées et retransmises par le moniteur série

**void clearList2(Liste\_C2\*\* head)** Une fois les données envoyées sur la carte SD et afin de libérer de l'espace pour les nouvelles, nous devons supprimer les données qui sont encore dans les listes chaînées. Un pointeur va alors parcourir la liste et vider chaque zone afin de retourner la liste identique à ce à qu'elle ressemblait au début.



```

Liste_C2* current = *head;
while (current != NULL) {
    Liste_C2* next = current->suivant2;
    free(current); // Libérer la mémoire de chaque élément
    current = next;
}
*head = NULL; // Réinitialiser la tête de la liste

```

## La fonction Interrupt et la Void Setup

### Les Interrupt

```

attachInterrupt(digitalPinToInterruption(bouton_A), interrupt, CHANGE);
attachInterrupt(digitalPinToInterruption(bouton_B), interrupt, CHANGE);

```

La fonction interrupt a été légèrement modifiée en comparaison à la structure du code donnée lors du second livrable.

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

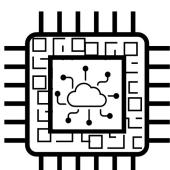
Ces deux lignes situées dans la Void Setup du code montrent le déclenchement de la fonction « interrupt » lorsque le bouton A (ou B en fonction de la ligne) sont appuyés.

La partie « digitalPinToInterruption (bouton\_A) » donne l'élément déclencheur de l'interrupt. La fonction déclenchée lors de l'interrupt est la suivante. Elle a seulement pour but de modifier la valeur de newMode de false à true pour que la fonction « choix » se déclenche. Elle démarre aussi un chronomètre qui, après analyse dans la fonction choix, empêche le ressaut des boutons et donc des bugs d'origine matérielle.

```

void interrupt() {
    if (millis() - lastInterrupt > 200) {
        lastInterrupt = millis();
        newMode = true;
    }
}

```



## 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 système  
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.

## L'horloge

L'initialisation de l'horloge dans la Void Setup permet de prévenir l'utilisateur de toute mal fonction et de vérifier son démarrage correct.

```
if (!rtc.begin()) {  
    Serial.println(F("RTC non trouvée"));  
    while (1);  
}  
  
// Si l'heure n'est pas réglée, la définir à la compilation  
// Code pour l'horloge RTC (commenté)  
if (!rtc.isrunning()) {  
    Serial.println(F("RTC non fonctionnelle. "));  
    // Réglez ici l'heure si nécessaire (année, mois, jour, heure, minute, seconde)  
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));  
}
```

## La carte SD

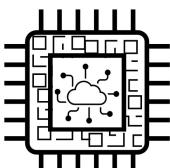
Si aucune carte SD n'est intégrée au programme, les conséquences sont nombreuses : beaucoup de bugs et de perte de données vont survenir. C'est pourquoi nous vérifions sa présence ainsi que son bon fonctionnement dans la Void Setup.

```
if (!SD.begin(carteSD)) { // Vérifiez que carteSD est bien défini  
    Serial.println(F("Erreur d'initialisation de la carte SD !"));  
    while (1); // Boucle infinie en cas d'erreur pour éviter de continuer  
} else {  
    Serial.println(F("Carte SD initialisée avec succès."));  
}
```

## Le capteur

Tout comme l'horloge ainsi que la carte SD, le capteur est testé au cours du Setup afin de prévenir la moindre absence ou mal fonction.

```
if (!bme.begin(0x76)) {  
    // Si le capteur n'est pas détecté à l'adresse 0x76, affiche un message d'erreur.  
    Serial.println(F("Capteur BME280 invalide"));  
    while (1); // Boucle infinie pour bloquer le programme en cas d'erreur.  
}
```



Dans le cas où un de ces 3 trois modules serait absent ou endommagé, le code ne serait pas en mesure de tourner correctement ce qui entraînerait des perturbations voir l'arrêt total du programme ou le bouclage sans fin. L'utilisateur ne serait alors pas en mesure de réparer la station sans faire de reset et donc de perdre les données temporairement stockées dans la partie mémoire DRAM de la station.

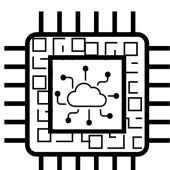
## La Void Loop

La Void Loop est la partie du code ayant subie le moins de changements par rapport au précédent rapport contenant la structure de notre code.

La variable globale « i » change de valeur entre 0 et 3 en fonction des modes données au cours de l'interruption et de la fonction choix. Chaque valeur de « i » déclenche alors une boucle « if » qui lance des appels aux fonctions nécessaires à chaque mode.

Nous avons ajouté une fonction qui vide les listes chainées à chaque tour de la Loop dans le mode maintenance car à défaut d'appeler la fonction stock\_donnees, le mode laissait les données dans les listes et saturait donc la mémoire. La destruction des données après leur affichage répond au critère du mode qui est d'afficher les données sans les stocker tout en ménageant l'espace mémoire de la carte.

```
if(i == 2){ // Modo mantenimiento
    digitalWrite(led_rouge, HIGH); // Allumer la LED rouge
    digitalWrite(led_bleue, LOW);
    digitalWrite(led_verte, LOW);
    digitalWrite(led_jaune, LOW);
    recuperation(); // Appel de la fonction
    clearList1(&tete1);
    clearList2(&tete2);
    clearList3(&tete3);
    clearList4(&tete4);
```



# Calcul de la complexité

## **Explication de la complexité du code :**

La complexité algorithmique est essentielle pour évaluer la performance d'un programme. Elle se divise principalement en  $O(1)$  (temps constant) et  $O(n)$  (linéaire, dépendant du nombre d'éléments).

### **O(1) Temps constant :**

Les opérations en  $O(1)$  sont exécutées en temps constant, peu importe la taille des données.

#### **Exemples d' $O(1)$ dans notre code :**

- Déclaration de variables : Créer une variable, comme un capteur, ne dépend pas de la taille des données, c'est instantané.
- Lecture de capteurs : Interroger un capteur pour obtenir une valeur, par exemple lire la température, est une opération instantanée.
- Conditions simples (if/else) : Vérifier une condition, comme l'état d'un capteur, ne nécessite pas de parcours de données.

Ces opérations ne varient pas avec la taille des données et n'impactent pas beaucoup la performance globale.

### **2. $O(n)$ Complexité linéaire :**

Les opérations en  $O(n)$  dépendent du nombre d'éléments à traiter et augmentent proportionnellement à la taille des données.

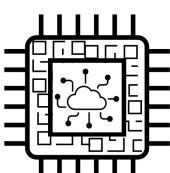
#### **Exemples d' $O(n)$ dans le code :**

- Ajout d'un élément à une liste chaînée : Parcourir la liste pour ajouter un élément à la fin prend un temps proportionnel à sa taille.
- Parcours d'une liste : Compter ou supprimer des éléments d'une liste nécessite de parcourir chaque élément, ce qui prend du temps linéaire selon le nombre d'éléments.

Plus la liste est grande, plus ces opérations prendront de temps, d'où l'importance de limiter la taille des listes pour éviter une dégradation des performances.

### **3. Impact sur la performance :**

Le code contient des listes chaînées, où la complexité linéaire ( $O(n)$ ) peut ralentir les opérations de manipulation des données si la taille des listes augmente. Par exemple, les

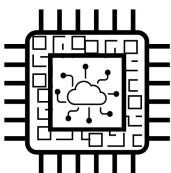


fonctions qui ajoutent ou suppriment des éléments dans ces listes deviennent de plus en plus lentes avec le temps.

Pour éviter cela, il est crucial de bien gérer la taille des listes, en limitant leur croissance ou en les purgeant régulièrement pour garder des performances stables.

**Pour conclure ces explications :**

- O(1) : Opérations rapides comme la lecture des capteurs ou la gestion de variables.
- O(n) : Opérations plus coûteuses comme la manipulation de listes chaînées, qui deviennent plus lentes avec la taille des données.



## Gestion de la mémoire

Afin de téléverser notre code sur la carte Arduino, il était nécessaire que celui-ci ne dépasse pas une taille maximum de 32256 octets en taille de programme et 2048 octets de mémoire dynamique (déclaration de variables, print, etc). Le code initial étant trop volumineux, il a été impératif pour nous de le réduire. Pour cela, nous avons utilisé les techniques suivantes :

- Inclusion de bibliothèques légères. En effet, les seules bibliothèques que nous avons utilisées sont obligatoires au bon fonctionnement du code. Celles-ci sont aussi spécifiques à nos capteurs et ne sont pas générales à plusieurs sortes de capteurs ce qui aurait entraîné une surcharge pour le programme déjà très lourd.
- L'utilisation de « Serial.println(F("Vous êtes en mode économique.")); la différence entre cette manière d'affichage et une autre (ne contenant pas le F dans les parenthèses) est que cette version ne stocke pas le message qui doit être affiché mais le lit et l'affiche uniquement au moment opportun sans utiliser de place superflue.
- L'utilisation de listes chainées dynamiques. Contrairement aux listes classiques, les listes chainées peuvent être déclarées comme étant vides et se remplir au fur et à mesure de l'avancement de notre code. Une liste doit forcément être définie par une taille qu'il est impossible de modifier contrairement aux listes chainées qui peuvent rentrer vide jusqu'à ce que le programme ait besoin de les remplir ce qui entraîne un gain de place considérable ainsi qu'une plus grande facilité d'utilisation dans notre cas.
- La déclaration des nombreuses constantes, variables et leur type est aussi très important dans la qualité et la taille du code que nous avons produit. Il est possible de voir dans le début du code de nombreuses déclarations de variables et de constantes, celles-ci sont globales et ont besoin d'être utilisées à l'intérieur de plusieurs fonctions du code. Néanmoins, avec plus d'attention, on repère aussi des déclarations locales : comme à l'intérieur des fonctions ou du Setup. Leur utilisation à des endroits précis nous a permis de les déclarer localement, leur présence est moins lourde dans le code, elles n'impactent pas le fonctionnement du code mais ces valeurs ne sont tout de même accessibles uniquement dans la fonction dans laquelle elles sont déclarées.

## Gestion de fichiers et stockage

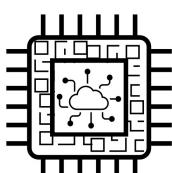
Dans notre station, nous allons utiliser un système de gestion de fichier FAT (Files allocation table ou table d'allocation de fichier), cette gestion de fichier permet de stocker les informations et les adresses mémoires des informations contenues dans notre carte SD.

Lors de l'envoi des fichiers dans une carte SD, il est indispensable de choisir la gestion de fichier que nous allons utiliser : Fat 16 ou 32.

Dans le contexte de notre projet, utiliser FAT32 semble l'option la plus adaptée.

En effet, notre carte SD fait 16 Go, or le FAT 16 n'est en mesure d'envoyer des données uniquement lorsque la taille de la mémoire est inférieure à 2 Go. De plus, le FAT 16 est une bonne manière de gérer des fichiers lorsque ceux-ci sont peu nombreux et légers. Dans notre cas, le FAT 32 va être nécessaire car le fichier sur lequel les données vont être stockées est lourd et contient beaucoup de données (elles sont envoyées 20 par 20 sur la carte SD au type 'float').

D'une manière générale, nous allons réaliser l'envoi de fichier sur la carte SD cela la gestion FAT 32 car celle-ci présente beaucoup plus d'avantages et une compatibilité avec notre projet.



## Adaptations sur la structure du code

Lors du passage entre l'architecture et la structure du code, certaines adaptations ont dû être réalisées afin que le code puisse tourner correctement. En effet, lorsque nous avons écrit puis réalisé de nombreux tests sur notre code, nous nous sommes rendu compte que certaines adaptations étaient à prévoir afin que le code ne tourne pas sans fin ou ne sature sa mémoire.

Certaines fonctions ont, par exemple, dû être ajoutées afin de vider les listes chainées une fois l'envoie des données réalisées sur la carte SD ou après une mesure dans le mode maintenance où la mesure une fois affichée est détruite.

Une fonction de lecture de la taille de la liste a aussi été créée afin que la fonctionnalité d'envoi de la liste sur la carte SD à partir d'une taille définie puisse se réaliser. Cette fonction parcourt la liste chainée et retourne le nombre d'éléments que celle-ci contient.

architecture	code final
void recuperation_donnees	void recuperation_donnees
void stockage_donnees	void stockage_donnees
int modification_inter	int modification_inter
int reset	int reset
int modification_heure	int modification_heure
int temps_AV_archi	int temps_AV_archi
int taille_AV_archi	int taille_AV_archi
	int checksize
	void clearlist1
	void clearlist2w
	void clearlist3
	void clearlist4

Une autre des modifications a aussi été réalisée sur la place des branchements de la carte Arduino des différentes entrées et sorties. En effet les boutons par exemple ne pouvaient pas se trouver sur les broches 6 et 7 comme déclaré dans la structure car les seules broches qui peuvent gérer les interrupt de boutons sur l'Arduino sont la 2 et la 3, il a donc fallu les déplacer.

Ancienne place des entrées sorties :

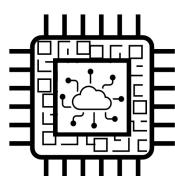
LED vert	2	capteur_1 (GPS)	D4
LED jaune	3	capteur_2 (température)	D3
LED rouge	4	capteur_3 (humidité)	D8
LED bleu	4	capteur_4 (pression)	A4 A5

bouton_A	6	horloge	D2
bouton_B	7		

Nouvelle place des entrées sorties :

LED vert	5	capteur_1 (GPS)	UART
LED jaune	6	capteur_2 (température)	A4 A5
LED rouge	7	capteur_3 (humidité)	A4 A5
LED bleu	9	capteur_4 (pression)	A4 A5
bouton_A	3	horloge	I2C
bouton_B	2	carte SD	4

Avec l'aide de ces modifications, le code est fonctionnel à 100 %.



# Guide d'utilisation

## Utilisation de la station météo Cloduino

### Nous avons 4 modes différents sur notre station météo

Le mode standard qui est le mode par défaut qui se met en route si aucun bouton n'est pressé. Si le mode par défaut est activé, une led verte s'allume, si cette led ne s'allume pas, vérifier les branchements de la station. Différentes données vont être acquise toutes les 10 minutes (hygrométries, températures, pression, position GPS)

Les données sont stockées et datés dans la carte SD, si pas de réponse des capteurs au bout de 30 secondes lors du stockage de données « NA » s'affiche.

**Si le bouton 1 est pressé, le mode configuration se lance**, une led jaune s'allume, si cette led ne s'allume pas, vérifier les branchements de la station.

Si vous voulez revenir au mode par défaut, il vous suffit de presser n'importe quel bouton.

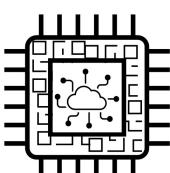
Dans ce mode configuration vous avez le choix entre plusieurs actions :

- Si vous entrez « 1 » vous allez pouvoir modifier la date et l'heure qui s'affiche sur la station.
- Si vous entrez « 2 » vous allez pouvoir modifier l'intervalle
- Si vous entrez « 3 » vous allez pouvoir reset au mode par défaut (mode standard)
- Si vous entrez « 4 » vous allez pouvoir modifier l'archivage, en entrant à nouveau soit «1» soit « 2 » vous allez soit :
  - En entrant « 1 » vous allez modifier le temps avant l'archivage des données
  - En entrant « 2 » vous allez modifier la taille avant l'archivage des données
- Si vous entrez « 5 » vous allez afficher la version du programme et numéro de lot de la station

**Si le bouton 2 est pressé, cela active le mode maintenance**, une led orange s'allume, si cette led ne s'allume pas, vérifier les branchements de la station.

Si vous voulez revenir au mode par défaut, il vous suffit de presser n'importe quel bouton.

Ce mode va affichées en direct les données et stopper l'archivage des données.



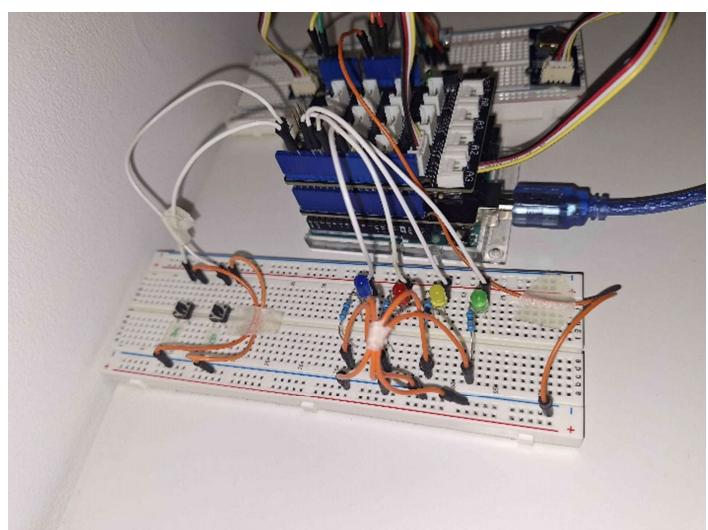
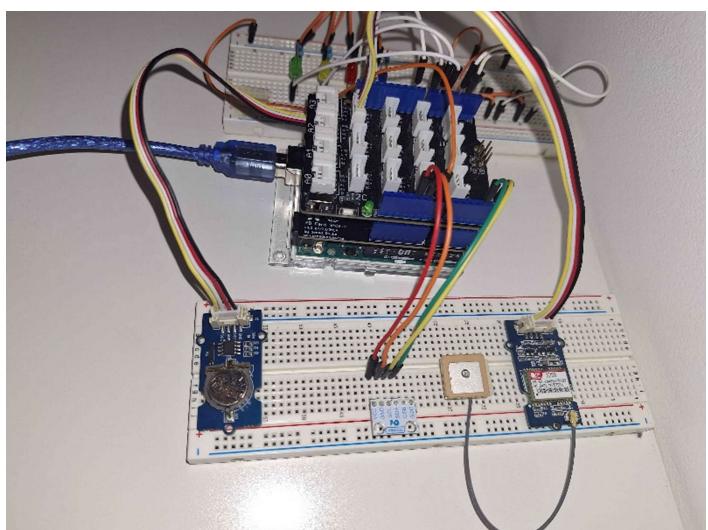
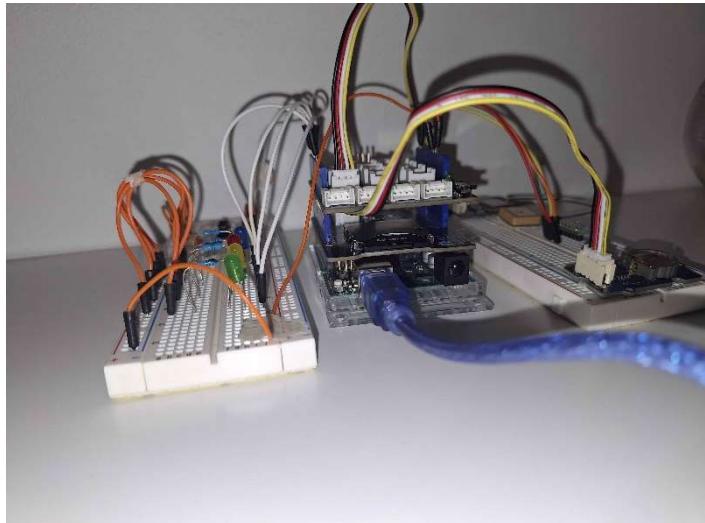
## **Si le bouton 1 et 2 est pressé, cela active le mode économie d'énergie.**

Dans ce mode le temps entre 2 mesures est multiplié par deux, et une mesure sur deux est retenue, il y a donc 4 fois moins de mesures. Ce mode est utile lorsque la batterie de la station est faible.

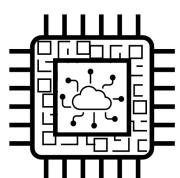
Si vous voulez revenir au mode par défaut, il vous suffit de presser n'importe quel bouton.

## **Problématiques courantes**

- Lors de l'allumage, il est important d'attendre que la led verte s'allume en mode statique pendant plus de 3 secondes (témoignant du bon démarrage du code) avant de changer de mode, sans ce délai d'attente, le changement de mode ne marchera pas.
- Lors de chaque changement de mode, il est impératif de repasser par le mode standard (led verte allumée).
- En cas de message d'erreur, vérifiez que tous les composants sont bien branchés et que la carte SD est présente dans le lecteur, placez la y le cas échéant.
- En cas de problème durant le fonctionnement (écran qui devient statique, boutons inopérants) supprimez le moniteur série et ouvrez en un nouveau, la station redémarrera mais les données précédentes enregistrées sur la carte SD ne seront pas perdues.
- Lors de la tentative du passage dans le mode économie, si un autre mode se lance à la place (maintenance ou configuration) revenez au mode standard puis recommencez.
- Si la carte SD se déconnecte subitement et ne stocke plus les données, il y a de fortes chances pour que celle-ci soit pleine, veillez donc à la retirer et à la vider régulièrement lors des maintenances.
- Si l'un des branchements venait à se déconnecter, rebranchez-le en vous basant sur les images de la station ci-dessous.



Station Météo Cloduino



# Conclusion

La structure 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, LED, 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 abritent une fonctionnalité indispensable au bon fonctionnement du code.

- La Void Setup :

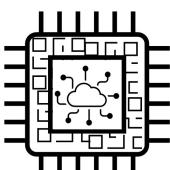
Cette boucle initialise certains mécanismes du code comme les Interrupts, l'horloge ou les entrées/sorties. Cette boucle réalise une dernière initialisation avant que la boucle principale du code ne se lance : la Void Loop.

- La Void Loop

Cette partie s'exécute en continue. Elle vérifie en continu 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.

Dans ce rendu, nous avons aussi analyser différents enjeux du codage sur système embarqué. Parmi ceux-ci, la gestion de la mémoire et la complexité du programme sont d'une importance capitale en raison du peu de mémoire DRAM présent sur les systèmes comme l'Arduino Uno.

Nous avons aussi joint à ce livrable, à destination des utilisateurs, un guide d'utilisation de la station ainsi que le guide de résolution des problèmes majeurs qui peuvent survenir durant le fonctionnement de la station météo.



## Annexe

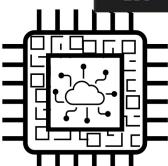
Code complet de la station :

```
1 //////////////// PROJET WORLDWIDE WEATHER WATCHER ///////////////////////
2
3 ////////////////// LIBRAIRIES///////////////////////////
4
5 #include <stdio.h>//gestion du programme (Print)
6 #include <string.h>//gestion des variables (Char, Listes...)
7 #include <stdlib.h>//gestion du programme
8 #include <Wire.h>//gestion du programme (Connection filaire avec des capteurs)
9 #include <SD.h>//carte SD
10 #include <SPI.h>//carte SD
11 #include <RTClib.h>//horloge
12 #include <Adafruit_Sensor.h>//capteur de température, humidité, pression
13 #include <Adafruit_BME280.h>//capteur de température, humidité, pression
14 #include <SoftwareSerial.h>//capteur gps
15 #include <TinyGPS++.h>//capteur gps
16
17 //////////// DECLARATION DES Capteurs /////////////
18
19 // Définir la broche où le capteur est connecté
20 RTC_DS1307 rtc;
21 #define SEALEVELPRESSURE_HPA (1013.25)
22 Adafruit_BME280 bme; // I2C pour le capteur BME280
23 TinyGPSPlus gps;
24 SoftwareSerial ss(4, 3);
25
26 /////DECLARATION DES VARIABLES, ENTREES ET SORTIES, CONSTANTES //////////
27
28 // Définition des constantes
29 const int led_verte = 5;
30 const int led_jaune = 6;
31 const int led_rouge = 7;
32 const int led_bleue = 9;
33 const int bouton_A = 3;
34 const int bouton_B = 2;
35 const int carteSD = 4;
36 // Définition des variables
37 bool etatbouton_A = false;
38 bool etatbouton_B = false;
39 // Variables des boucles
40 int i = 0;
41 bool newMode = false;
42 unsigned long lastInterrupt = 0; //Temps de la dernière Interruption
43 unsigned long lastArchiveTime = 0; // Temps de la dernière archive sur la carte SD
44 unsigned long previousMillis = 0; // Temps de la dernière mesure de capteur
45 // Version du système
46 const char* version = "Clouduino";
47 // Intervalle entre les mesures
48 int inter = 3000;
49 // Temps avant archivage
50 int TeAA = 30000;
51 // Taille avant archivage (en octets)
52 int TaAA = 6;
53 // Déclaration des capteurs
54 float capteur_1;
55 float capteur_2;
56 float capteur_3;
57 float capteur_4;
58
59 ///// Tableau de stockage des mesures sous forme de listes chainées /////
60
61 // Mesures du capteur GPS
62 typedef struct Liste_C1 {
63     float gps;
64     struct Liste_C1 *suivant1;
65 } Liste_C1;
```

```

66 // Mesures du capteur de température
67 typedef struct Liste_C2 {
68     float temp;
69     struct Liste_C2 *suivant;
140     capteur_2 = bme.readTemperature();
141     capteur_3 = bme.readHumidity();
142     capteur_4 = bme.readPressure();
143     while (ss.available() > 0) {
144         gps.encode(ss.read()); // Décoder les données GPS
145         // Si la position GPS a été mise à jour, afficher la latitude et la longitude
146         if (gps.location.isUpdated()) {
147             Serial.print(F("Latitude: "));
148             capteur_1 = gps.location.lat();
149             Serial.println(capteur_1, 6); // Affiche la latitude avec une précision de 6 décimales
150             // Ajouter la nouvelle latitude dans la liste chaînée
151             Liste_C1* elt1 = (Liste_C1*)malloc(sizeof(Liste_C1)); // Allouer de la mémoire pour un nouvel élément
152             elt1->gps = capteur_1;
153             elt1->suivant = NULL;
154
155             if (tete1 == NULL) {
156                 tete1 = elt1; // Si la liste est vide, le nouvel élément devient la tête
157             } else {
158                 Liste_C1* current1 = tete1;
159                 while (current1->suivant != NULL) { // Parcourir la liste jusqu'au dernier élément
160                     current1 = current1->suivant;
161                 }
162                 current1->suivant = elt1; // Ajouter le nouvel élément à la fin
163             }
164             Serial.print(F("Longitude: "));
165             capteur_1 = gps.location.lng();
166             Serial.println(capteur_1, 6); // Affiche la longitude avec une précision de 6 décimales
167         }
168     }
169     // Optionnel: Si tu veux afficher un message lorsque la position GPS n'est pas encore disponible
170     if (!gps.location.isValid()) {
171         capteur_1 = 0;
172         Serial.println(F("Coordonnées GPS : Inaccessibles. Archivage NA."));
173     }
174     // Afficher la température actuelle sur le moniteur série
175     Serial.print(F("Température actuelle : "));
176     Serial.print(capteur_2);
177     Serial.println(F("°C"));
178     Serial.print(F("Humidité actuelle : "));
179     Serial.print(capteur_3);
180     Serial.println(F("%"));
181     Serial.print(F("Pression actuelle : "));
182     Serial.print(capteur_4);
183     Serial.println(F("Pa"));
184     // Ajouter la nouvelle longitude dans la liste chaînée
185     Liste_C1* elt1 = (Liste_C1*)malloc(sizeof(Liste_C1)); // Allouer de la mémoire pour un nouvel élément
186     elt1->gps = capteur_1;
187     elt1->suivant = NULL;
188
189     if (tete1 == NULL) {
190         tete1 = elt1; // Si la liste est vide, le nouvel élément devient la tête
191     } else {
192         Liste_C1* current1 = tete1;
193         while (current1->suivant != NULL) { // Parcourir la liste jusqu'au dernier élément
194             current1 = current1->suivant;
195         }
196         current1->suivant = elt1; // Ajouter le nouvel élément à la fin
197     }
198     // Ajouter la nouvelle température dans la liste chaînée
199     Liste_C2* elt2 = (Liste_C2*)malloc(sizeof(Liste_C2)); // Allouer de la mémoire pour un nouvel élément
200     elt2->temp = capteur_2;
201     elt2->suivant2 = NULL;
202     if (tete2 == NULL) {
203         tete2 = elt2; // Si la liste est vide, le nouvel élément devient la tête
204     } else {
205         Liste_C2* current2 = tete2;
206     }
207     // Si l'intervalle est atteint, effectuer la mesure du capteur
208     if (current2Millis - previousMillis >= interbis) {
209         previousMillis = current2Millis; // Mettre à jour l'heure de la dernière mesure

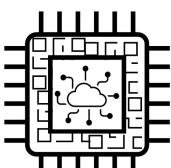
```



```

205     Liste_C2* current2 = tete2;
206     while (current2->suivant2 != NULL) { // Parcourir la liste jusqu'au dernier élément
207         current2 = current2->suivant2;
208     }
209     current2->suivant2 = elt2; // Ajouter le nouvel élément à la fin
210 }
211 // Ajouter la nouvelle humidité dans la liste chaînée
212 Liste_C3* elt3 = (Liste_C3*)malloc(sizeof(Liste_C3)); // Allouer de la mémoire pour un nouvel élément
213 elt3->humi = capteur_3;
214 elt3->suivant3 = NULL;
215
216 if (tete3 == NULL) {
217     tete3 = elt3; // Si la liste est vide, le nouvel élément devient la tête
218 } else {
219     Liste_C3* current3 = tete3;
220     while (current3->suivant3 != NULL) { // Parcourir la liste jusqu'au dernier élément
221         current3 = current3->suivant3;
222     }
223     current3->suivant3 = elt3; // Ajouter le nouvel élément à la fin
224 }
225 // Ajouter la nouvelle pression dans la liste chaînée
226 Liste_C4* elt4 = (Liste_C4*)malloc(sizeof(Liste_C4)); // Allouer de la mémoire pour un nouvel élément
227 elt4->pres = capteur_4;
228 elt4->suivant4 = NULL;
229
230 if (tete4 == NULL) {
231     tete4 = elt4; // Si la liste est vide, le nouvel élément devient la tête
232 } else {
233     Liste_C4* current4 = tete4;
234     while (current4->suivant4 != NULL) { // Parcourir la liste jusqu'au dernier élément
235         current4 = current4->suivant4;
236     }
237     current4->suivant4 = elt4; // Ajouter le nouvel élément à la fin
238 }
239 }
240 }
241 void stock_donnees() {
242     unsigned long currentMillis = millis();
243     // Vérifier si la taille d'une des listes dépasse TaAA ou si TeAA temps est écoulé
244     if (checkSize(tete2) >= TaAA || checkSize(tete3) >= TaAA || checkSize(tete4) >= TaAA ||
245         (currentMillis - lastArchiveTime) >= TeAA) {
246         // Ouvrir le fichier sur la carte SD pour y écrire
247         File dataFile = SD.open("datalog.txt", FILE_WRITE);
248         DateTime now = rtc.now();
249         if (dataFile) {
250             // Écrire les données gps
251             Liste_C1* current1 = tete1;
252             while (current1 != NULL) {
253                 dataFile.print("Date/Heure: ");
254                 dataFile.print(now.year(), DEC);
255                 dataFile.print('/');
256                 dataFile.print(now.month(), DEC);
257                 dataFile.print('/');
258                 dataFile.print(now.day(), DEC);
259                 dataFile.print(" - ");
260                 dataFile.print(now.hour(), DEC);
261                 dataFile.print(":");
262                 dataFile.print(now.minute(), DEC);
263                 dataFile.print(":");
264                 dataFile.print(now.second(), DEC);
265                 dataFile.print(" | GPS lat, long : ");
266                 dataFile.println(current1->gps);
267                 current1 = current1->suivant1;
268             }
269             // Écrire les données de température
270             Liste_C2* current2 = tete2;
271             while (current2 != NULL) {
272                 dataFile.print("Date/Heure: ");
273                 dataFile.print(now.year(), DEC);
274                 dataFile.print('/');
275                 dataFile.print(now.month(), DEC);

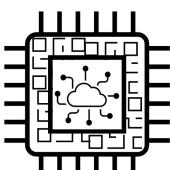
```



```

275     dataFile.print(now.month(), DEC);
276     dataFile.print('/');
277     dataFile.print(now.day(), DEC);
278     dataFile.print(" - ");
279     dataFile.print(now.hour(), DEC);
280     dataFile.print(':');
281     dataFile.print(now.minute(), DEC);
282     dataFile.print(':');
283     dataFile.print(now.second(), DEC);
284     dataFile.print(" | Température : ");
285     dataFile.println(current2->temp);
286     current2 = current2->suivant2;
287   }
288   // Écrire les données d'humidité
289   Liste_C3* current3 = tete3;
290   while (current3 != NULL) {
291     dataFile.print("Date/Heure: ");
292     dataFile.print(now.year(), DEC);
293     dataFile.print('/');
294     dataFile.print(now.month(), DEC);
295     dataFile.print('/');
296     dataFile.print(now.day(), DEC);
297     dataFile.print(" - ");
298     dataFile.print(now.hour(), DEC);
299     dataFile.print(':');
300     dataFile.print(now.minute(), DEC);
301     dataFile.print(':');
302     dataFile.print(now.second(), DEC);
303     dataFile.print(" | Humidité : ");
304     dataFile.println(current3->humi);
305     current3 = current3->suivant3;
306   }
307   // Écrire les données de pression
308   Liste_C4* current4 = tete4;
309   while (current4 != NULL) {
310     dataFile.print("Date/Heure: ");
311     dataFile.print(now.year(), DEC);
312     dataFile.print('/');
313     dataFile.print(now.month(), DEC);
314     dataFile.print('/');
315     dataFile.print(now.day(), DEC);
316     dataFile.print(" - ");
317     dataFile.print(now.hour(), DEC);
318     dataFile.print(':');
319     dataFile.print(now.minute(), DEC);
320     dataFile.print(':');
321     dataFile.print(now.second(), DEC);
322     dataFile.print(" | Pression : ");
323     dataFile.println(current4->pres);
324     current4 = current4->suivant4;
325   }
326   dataFile.close(); // Fermer le fichier après l'écriture
327   Serial.println(F("Données écrites sur la carte SD."));
328 } else {
329   Serial.println(F("Erreur d'ouverture du fichier sur la carte SD."));
330 }
331 // Vider les listes chaînées après archivage
332 clearList2(&tete2);
333 clearList3(&tete3);
334 clearList4(&tete4);
335 // Mettre à jour le temps de la dernière archive
336 lastArchiveTime = currentMillis;
337 }
338 }
339 int checkSize(void* head) { // Fonction pour calculer la taille d'une liste chaînée
340   int size = 0;

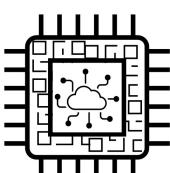
```



```

341     Liste_C1* current = (Liste_C1*)head;
342     while (current != NULL) {
343         size += sizeof(Liste_C1); // Ajouter la taille de chaque élément
344         current = current->suivant1;
345     }
346     return size;
347 }
348 void clearList4(Liste_C1** head) {// Fonction pour vider une liste chaînée
349     Liste_C1* current = *head;
350     while (current != NULL) {
351         Liste_C1* next = current->suivant1;
352         free(current); // Libérer la mémoire de chaque élément
353         current = next;
354     }
355     *head = NULL; // Réinitialiser la tête de la liste
356 }
357 void clearList2(Liste_C2** head) {// Fonction pour vider une liste chaînée
358     Liste_C2* current = *head;
359     while (current != NULL) {
360         Liste_C2* next = current->suivant2;
361         free(current); // Libérer la mémoire de chaque élément
362         current = next;
363     }
364     *head = NULL; // Réinitialiser la tête de la liste
365 }
366 void clearList3(Liste_C3** head) {// Fonction pour vider une liste chaînée
367     Liste_C3* current = *head;
368     while (current != NULL) {
369         Liste_C3* next = current->suivant3;
370         free(current); // Libérer la mémoire de chaque élément
371         current = next;
372     }
373     *head = NULL; // Réinitialiser la tête de la liste
374 }
375 void clearList4(Liste_C4** head) {// Fonction pour vider une liste chaînée
376     Liste_C4* current = *head;
377     while (current != NULL) {
378         Liste_C4* next = current->suivant4;
379         free(current); // Libérer la mémoire de chaque élément
380         current = next;
381     }
382     *head = NULL; // Réinitialiser la tête de la liste
383 }
384 int modif_inter() {
385     Serial.println(F("Quelle intervalle de temps entre laquisition des données souhaitez vous ?"));
386     inter = Serial.read() - '0';
387 }
388 int reset() {
389     // Réinitialisation des paramètres par défaut
390     inter = 10;
391     TeAA = 10;
392     TaAA = 20000;
393     return 0;
394 }
395 int modifi_heure(int annee,int mois,int jour,int heure, int min,int sec){
396     //modifie l'heure du système selon une demande de l'utilisateur
397     rtc.adjust(DateTime(annee, mois, jour, heure, min, sec));
398 }
399 int temps_AV_archi() {
400     Serial.println(F("Quelle intervalle de temps entre l'archivage des données souhaitez vous ?"));
401     TeAA = Serial.read() - '0';
402 }
403 int taille_AV_archi() {
404     Serial.println(F("Quelle intervalle de taille entre l'archivage des données souhaitez vous ?"));
405     TaAA = Serial.read() - '0';

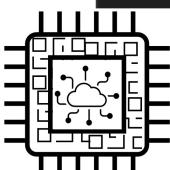
```



```

407 }
408
409 //////////////// VOID SETUP INITIALISATION DU SYSTÈME ///////////////
410
411 void setup() {
412     Serial.begin(9600);
413     ss.begin(9600);
414     pinMode(led_verte, OUTPUT);
415     pinMode(led_jaune, OUTPUT);
416     pinMode(led_rouge, OUTPUT);
417     pinMode(led_bleue, OUTPUT);
418     pinMode(bouton_A, INPUT_PULLUP);
419     pinMode(bouton_B, INPUT_PULLUP);
420     if (!rtc.begin()) {
421         Serial.println(F("RTC non trouvée"));
422         while (1);
423     }
424     // Si l'heure n'est pas réglée, la définir à la compilation
425     // Code pour l'horloge RTC (commenté)
426     if (!rtc.isrunning()) {
427         Serial.println(F("RTC non fonctionnelle."));
428         // Réglez ici l'heure si nécessaire (année, mois, jour, heure, minute, seconde)
429         rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
430     }
431     if (!SD.begin(carteSD)) { // Vérifiez que carteSD est bien défini
432         Serial.println(F("Erreur d'initialisation de la carte SD !"));
433         while (1); // Boucle infinie en cas d'erreur pour éviter de continuer
434     } else {
435         Serial.println(F("Carte SD initialisée avec succès."));
436     }
437     if (!bme.begin(0x76)) {
438         // Si le capteur n'est pas détecté à l'adresse 0x76, affiche un message d'erreur.
439         Serial.println(F("Capteur BME280 invalide"));
440         while (1); // Boucle infinie pour bloquer le programme en cas d'erreur.
441     }
442     Serial.println(F("Vous avez été log au mode Standard"));
443     // Setup des interruptions
444     attachInterrupt(digitalPinToInterrupt(bouton_A), interrupt, CHANGE);
445     attachInterrupt(digitalPinToInterrupt(bouton_B), interrupt, CHANGE);
446 }
447 //////////////// VOID LOOP, CODE PRINCIPAL ///////////////
448 void loop() {
449     choix();
450     if(i == 0){
451         digitalWrite(led_verte, HIGH); // Allumer la LED verte
452         digitalWrite(led_jaune, LOW);
453         digitalWrite(led_rouge, LOW);
454         digitalWrite(led_bleue, LOW);
455         recuperer_donnees(); // Appel de la fonction
456         stocker_donnees(); // Appel de la fonction
457     }
458     if(i==1){
459         digitalWrite(led_jaune, HIGH); // Allumer la LED jaune
460         digitalWrite(led_rouge, LOW);
461         digitalWrite(led_bleue, LOW);
462         digitalWrite(led_verte, LOW);
463         recuperer_donnees(); // Appel de la fonction
464         stocker_donnees(); // Appel de la fonction
465         Serial.println("Que voulez-vous faire ?");
466     }
}

```



```

466 // while (Serial.available() == 0) {} // Attente de l'entrée utilisateur
467 int confi = Serial.read() - '0';
468
469 if (confi == 1) {
470     // modif_heure(); // Appel de la fonction
471 }
472 if (confi == 2) {
473     modif_inter(); // Appel de la fonction
474 }
475 if (confi == 3) {
476     reset(); // Appel de la fonction
477 }
478 if (confi == 4) {
479     Serial.println(F("Entrez 1 pour modifier le temps avant l'archivage."));
480     Serial.println(F("Entrez 2 pour modifier la taille avant l'archivage."));
481
482     //while (Serial.available() == 0) {} // Attente de l'entrée utilisateur
483     int archi = Serial.read() - '0';
484
485     if (archi == 1) {
486         temps_AV_archi(); // Appel de la fonction
487     }
488     if (archi == 2) {
489         taille_AV_archi(); // Appel de la fonction
490     }
491 }
492 if (confi == 5) {
493     Serial.println(version); // Affiche la version
494 }
495 }
496 if(i == 2){ // Modo mantenimiento
497     digitalWrite(led_rouge, HIGH); // Allumer la LED rouge
498     digitalWrite(led_bleue, LOW);
499     digitalWrite(led_verte, LOW);
500     digitalWrite(led_jaune, LOW);
501     recuper_donnees(); // Appel de la fonction
502     clearList1(&tete1);
503     clearList2(&tete2);
504     clearList3(&tete3);
505     clearList4(&tete4);
506 }
507 if(i==3){
508     digitalWrite(led_bleue, HIGH); // Allumer la LED bleue
509     digitalWrite(led_verte, LOW);
510     digitalWrite(led_jaune, LOW);
511     digitalWrite(led_rouge, LOW);
512     recuper_donnees(); // Appel de la fonction
513     stock_donnees(); // Appel de la fonction
514 }
515 }
516 }
```

