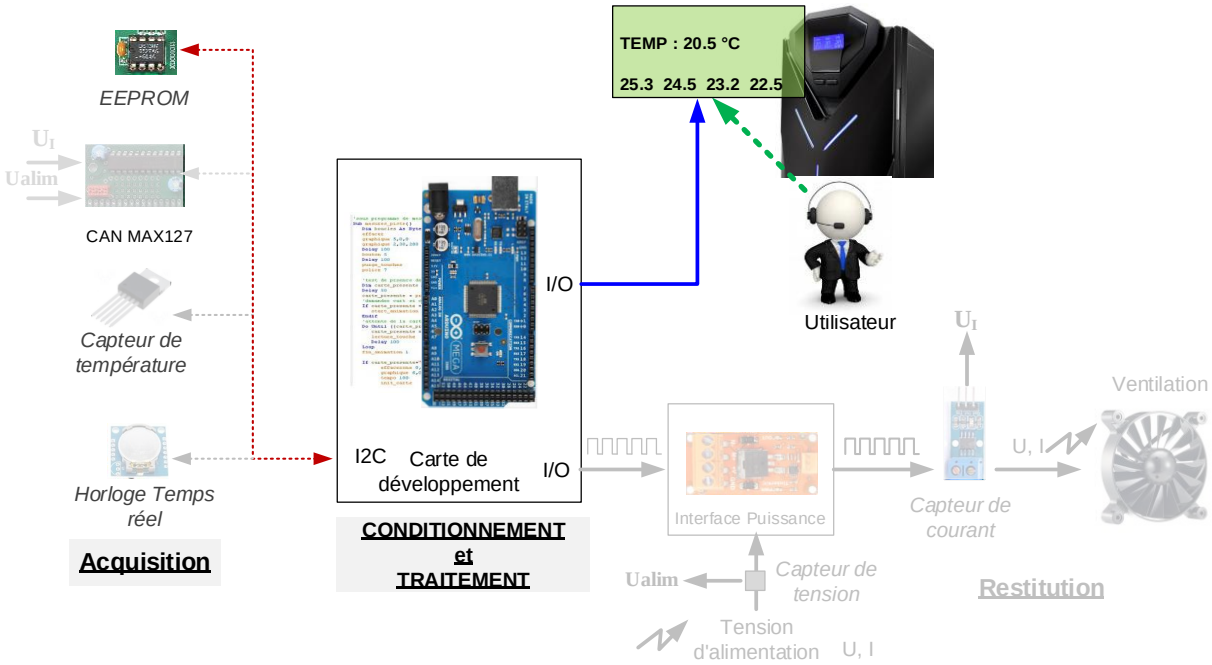
♦ PRESENTATION DU PROJET :

On désire **Mémoriser** les valeurs de Température mesurées sur le système. Pour cela, il vous faudra mettre en place l'enregistrement puis la lecture des données grâce à une mémoire EEPROM sur bus I2C.

Conditions de fonctionnement :

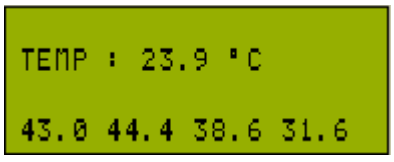
- Enregistrement de la température toute les secondes,
- Affichage des 4 dernières données mesurées



Problématique posée : *Comment mémoriser les mesures de températures pour un post-traitement des données sur un système microprogrammé ?*

L'objectif est donc de mesurer et de mémoriser en **temps réel** la valeur de température mesurée par le capteur.

- Affichage sur la 2ème ligne de la valeur instantanée de la température mesurée
- Affichage sur la 4ème ligne des 4 dernières mesures effectuées



Exemple d'affichage

Remarque : la température pourra être simulée par un signal sinusoïdal (0-5V) sur l'entrée analogique **A0** de l'Arduino Méga, de façon à générer des températures variables dans le temps, ou par valeurs fixes grâce à un potentiomètre.

Cette Activité est évaluée par revue de projet individuelle !

♦ Programmation sous Arduino de la mémoire I²C EEPROM 24LC256

La bibliothèque **Wire** permet de communiquer sur le bus I2C en Arduino. Il est nécessaire de l'importer dans votre programme :

```
#include <Wire.h>
```

Description des fonctions I²C :

Wire.**begin**() : initialise le mode de communication I2C. A écrire une fois dans la procédure **Setup**( )

Wire.**beginTransmission**(**adress**) : débute une transmission avec un circuit I2C. Cette fonction permet de prendre la main sur le bus, et d'indiquer, par la variable **adress,** avec quel circuit l'on désire communiquer.

Wire.**requestFrom** (***adress***, ***quantity***, ***ACK***) : Requête envoyée à l'esclave (***adress***) lui demandant de placer sur le bus I2C les données de son ou de ses registres. Le bit R/W est alors automatiquement mis à 1 (Lecture).

quantity correspond au nombre **d'octets** à lire.

ACK est un booléen (True/False) indiquant si le maître d'opération doit acquitter chaque lecture de données.

Wire.**available**() : vérifie si les données sont disponibles sur le BUS après le lancement d'une requête. Cette fonction renvoie le nombre d'octets disponibles.

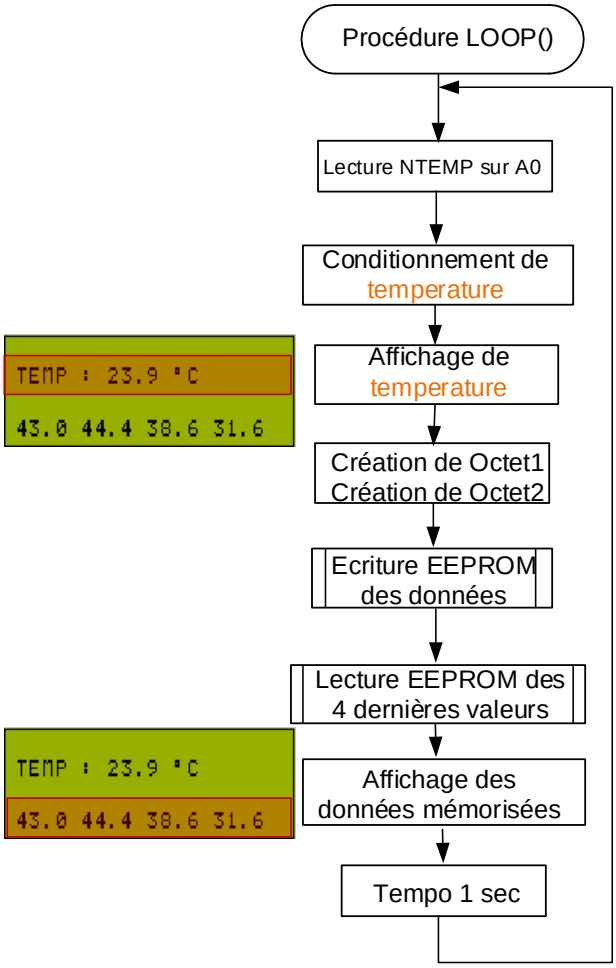
Wire.**read**() : Permet de lire un ou plusieurs octets placés par l'esclave sur le bus. Cette fonction ne peut fonctionner que si un début de transmission est effectué et si les données sont disponibles.

Wire.**endTransmission**( ) : Termine une transmission en cours sur le bus I2C. Cette fonction permet de libérer le bus en plaçant un STOP sur la trame.

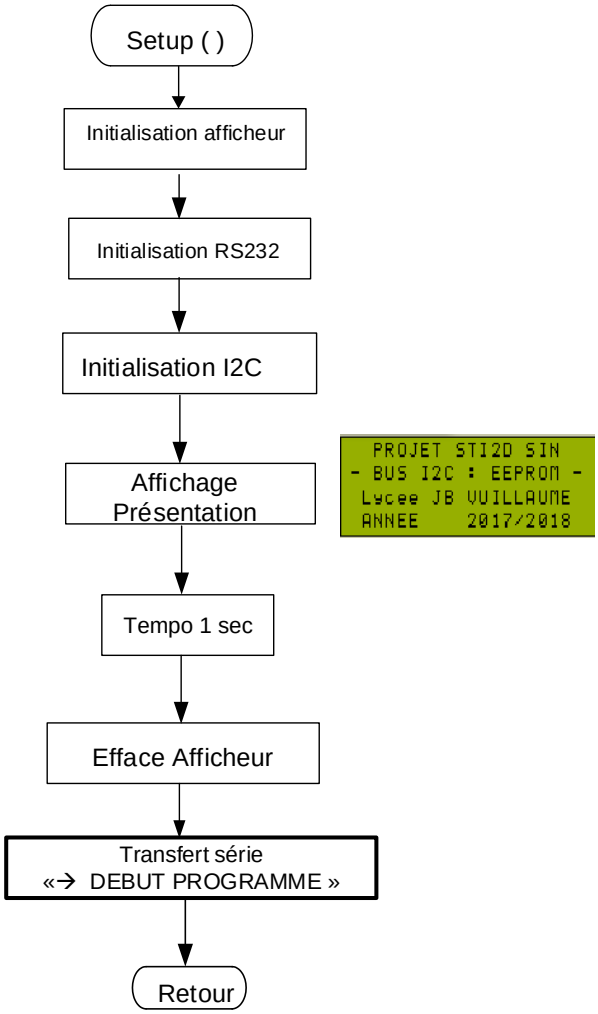
Wire.**write**() :Permet d'écrire une donnée sur le bus. Cette fonction ne peut fonctionner que si un début de transmission est effectué. Le bit R/W est alors automatiquement mis à 0.

♦ ORGANIGRAMMES :

Programme PRINCIPAL (procédure LOOP) :



SP Initialisation () :



♦ ALGORITHME DE LA PROCÉDURE *writeEEPROM*

```
void writeEEPROM(int ADDR_I2C, unsigned int eeaddress, byte data )
```

Prise en main du BUS et connexion au circuit EEPROM
Temporisation de 1 ms
Écriture sur le bus de l'octet MSB de l'adresse 16 bits eeadress
Écriture sur le bus de l'octet LSB de l'adresse 16 bits eeadress
Écriture de l'octet data à mémoriser sur le BUS
Temporisation de 1 ms
Fin de transmission : Libération du BUS is

♦ ALGORITHME DE LA FONCTION *readEEPROM*

```
byte readEEPROM(int ADDR_I2C, unsigned int eeaddress )
```

Prise en main du BUS et connexion au circuit EEPROM
Temporisation de 1 ms
Écriture sur le bus de l'octet MSB de l'adresse 16 bits eeadress
Écriture sur le bus de l'octet LSB de l'adresse 16 bits eeadress
Écriture de l'octet data à mémoriser sur le BUS
Temporisation de 1 ms
Fin de transmission : Libération du BUS

Prise en main du BUS et connexion au circuit EEPROM
Temporisation de 1 ms
Requête de lecture d'1 Octet sur le circuit EEPROM sans ACK
SI la donnée est disponible ALORS
    Octet ← lecture de la donnée
FIN
Temporisation de 1 ms
Fin de transmission : Libération du BUS
Retourne la variable Octet

Remarques :

⇨ Le **ET logique** (&) avec le **OU logique** (|) permet de reconstruire une donnée codée sur 16 bits à partir de deux éléments codés sur 8 bits. Le ET élimine les bits inutilisés alors que le OU associe les bits.

Data1 = 0b0000000000110011 Data2 = 0b1100110000000000
Data = Data1 **|** Data2 -> 0b1100110000110011

⇨ L'opérateur de décalage en Arduino est **>>** pour un décalage à droite et **<<** pour un décalage à gauche :

```
int val  = 0b1111000010010000;
val = val << 4;       \\val = %0000 1001 0000 0000 : décalage de 4 bits à gauche
```

Exemples : Transformation Adresse EEPROM et reconstruction donnée NTEMP

L'adresse EEPROM est $0**19B** — *eeaddress* = % **0000 0001 1001 1011**

8 décalages droite de *eeaddress*: addHIGH = % 0000 0000 **0000 0001**

ET logique entre *eeaddress* et 0x**00FF** : addLOW = % 0000 0000 **1001 1011**

La donnée à lire correspond à $**1FE** (510) : % **0000 0001 1111 1110**

Lecture de l'Octet MSB : Data1 = % 0000 0000 **0000 0001**

Lecture de l'Octet LSB : Data2 = % 0000 0000 **1111 1110**

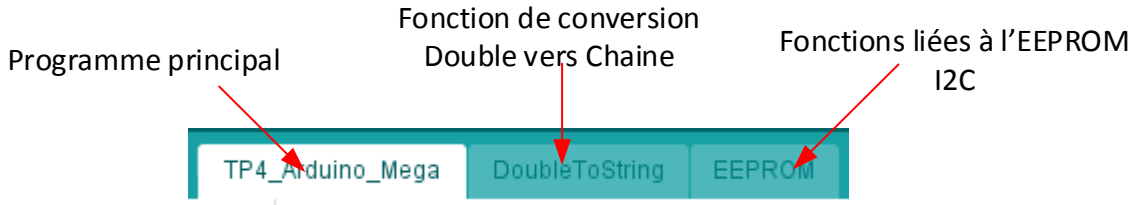
8 décalage gauche de Data1 : Data1 = % **0000 0001** 0000 0000

OU logique entre Data1 et Data2 : NTEMP = % **0000 0001 1111 1110**

♦ REALISATION du PROGRAMME :

☞ Ouvrez le programme « *TP1_Arduino_Mega.ino* » contenu dans le dossier de ressources « *\programme* »

Celui-ci est organisé en Onglets :

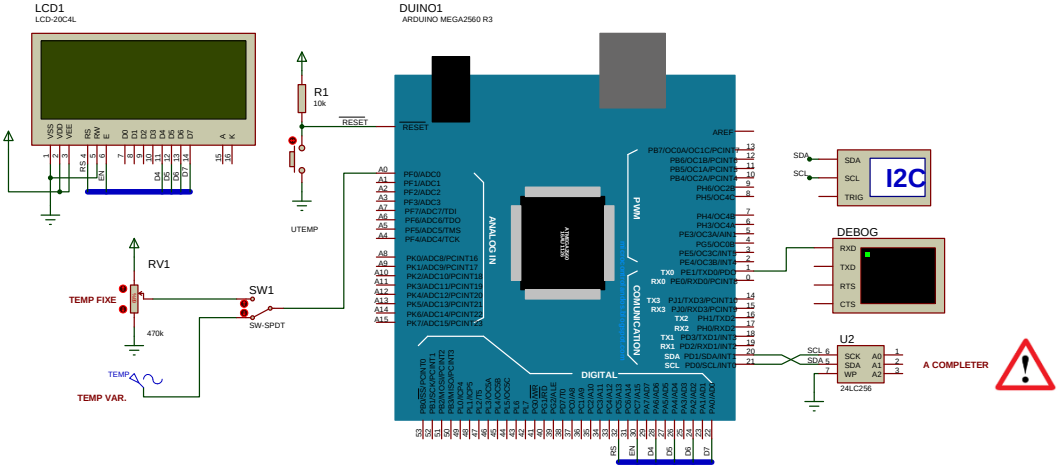


☞ Ouvrez le fichier Arduino puis complétez les différentes parties du programme en vous aidant du cahier des charges, à savoir :

- Procédure d'initialisation `setup()`
- Programme principal `loop()`
- Procédure `writeEEPROM(int deviceaddress, unsigned int eeaddress, byte data )`
- Fonction `byte readEEPROM(int deviceaddress, unsigned int eeaddress )`

♦ SIMULATION et MISE AU POINT du PROGRAMME :

☞ Ouvrez le fichier de simulation **Proteus ISIS** « *TP1-HORLOGE-I2C.DSN* » contenu dans le dossier de ressources « \simulation »



☞ Placez le programme ***TP4_Arduino_Mega.ino.HEX*** dans le modèle de simulation Arduino, et simulez le montage. Mettre au point le programme afin de répondre au cahier des charges.

TRAVAILLEZ PAR ÉTAPE !!! Utilisez les opérateurs /* et */ pour mettre en commentaire les fonctions ou parties de programme en attentes d'écriture.

Utilisez le débogueur I2C pour analyser le contenu des trames !