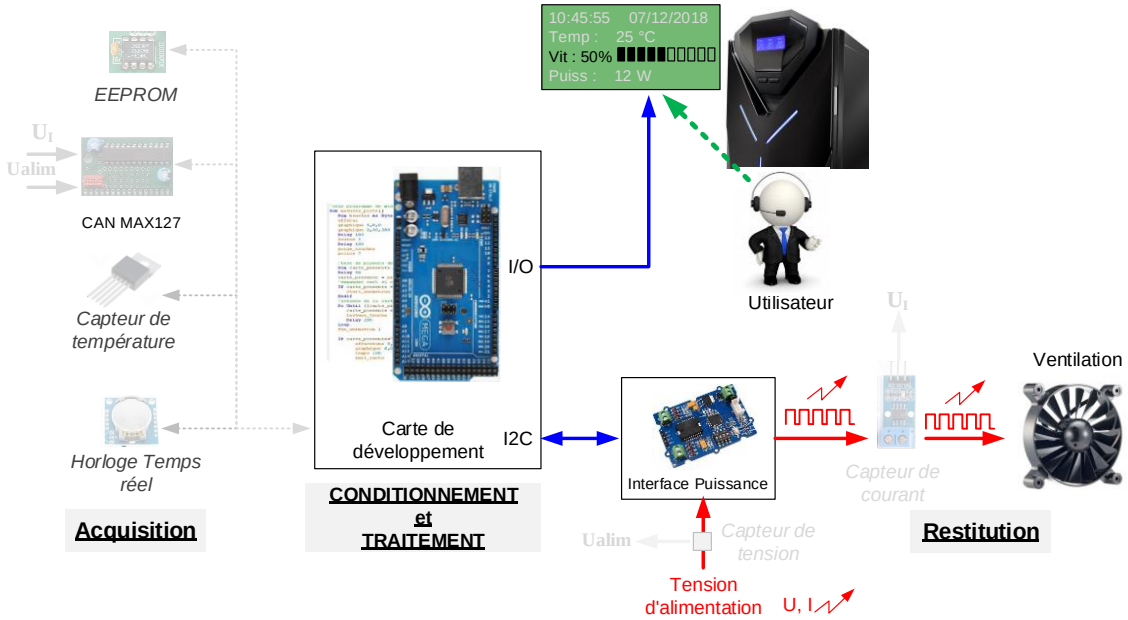
♦ PRESENTATION DU PROJET :

On désire programmer le système afin de répondre au besoin de commande de rotation du moteur de ventilation du boîtier PC en fonction du niveau de température du boitier.

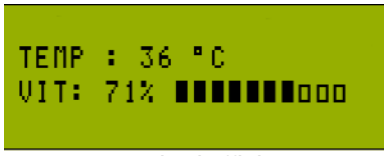
Conditions de fonctionnement :

- Mise en Rotation ou Arrêt du ventilateur,
- Sens de rotation unidirectionnel (Sens horaire),
- Vitesse de Rotation fonction de la température interne du boitier



Problématique posée : *Comment adapter automatiquement la vitesse de rotation d'un ventilateur à courant continu en fonction du niveau de température mesurée sur un système microprogrammé ?*
L'objectif est donc de commander en **temps réel toute les secondes** le niveau de rotation du ventilateur dès la mise sous tension du système :

- Lecture et affichage sur ligne 2 de la température du boitier PC toute les secondes
- Variation de la vitesse de rotation en fonction de la mesure de température et affichage de la vitesse ligne 3 sous forme de barregraphe



Exemple d'affichage

Remarque : la température sera simulée par une tension analogique (0-5V) sur l'entrée **A0** réglable par potentiomètre de façon à générer des températures variables dans le temps.

Cette Activité est évaluée par revue de projet individuelle !

♦ Programmation sous Arduino de l'interface de puissance L298 en I²C

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.

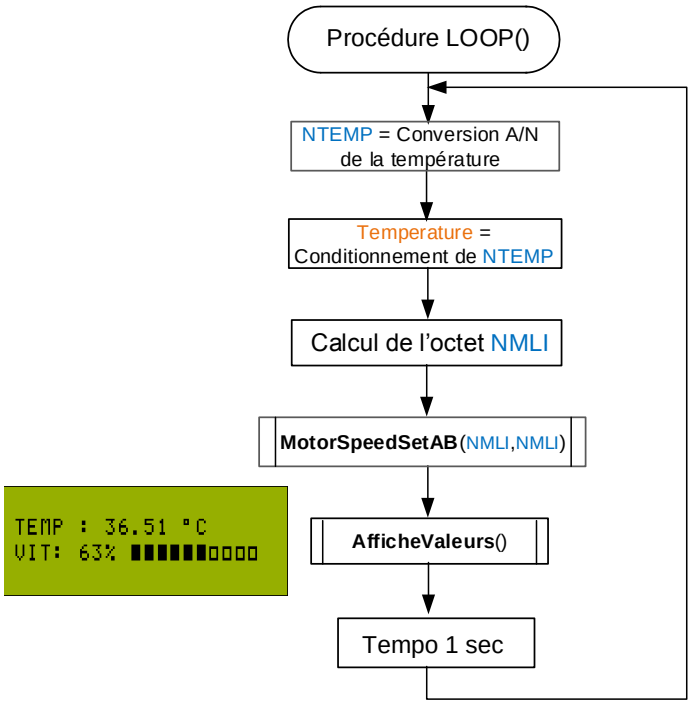
Wire.**read**() : Permet de lire un ou plusieurs octets placés par l'esclave sur le bus. Cette fonction ne peut fonctionner que si un début de transmission est effectué et si les données sont disponibles.

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.

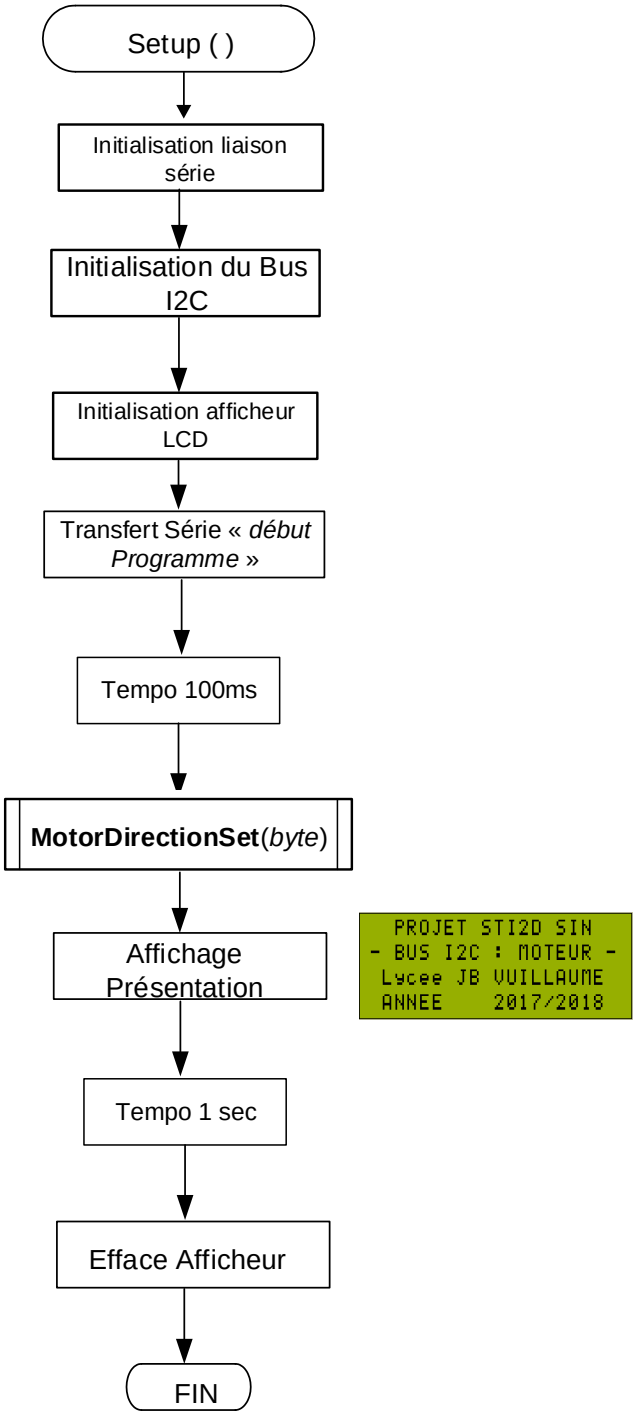
Wire.**write**() : Permet d'écrire une donnée sur le bus. Cette fonction ne peut fonctionner que si un début de transmission est effectué. Le bit R/W est alors automatiquement mis à 0.

♦ ORGANIGRAMMES :

Programme PRINCIPAL (procédure LOOP) :



Procédure setup() :



♦ ALGORITHME DE LA PROCEDURE *AfficheValeurs*

```
void AfficheValeurs()
```

Positionner le curseur afficheur LCD sur ligne 1 et colonne 0
Afficher de la température en °C sur LCD
Calculer de Taux_MLI à partir de NMLI

POUR i = 0 A <10 AU PAS de 1
    Positionner le curseur afficheur LCD sur ligne 2 et colonne 9 + i
    Effacement (Affiche un espace)
    Positionner le curseur afficheur LCD sur ligne 2 et colonne 9 + i
    Afficher le caractère 219 (□)
FIN

POUR i = 0 A <Taux_MLI/10 AU PAS de 1
    Positionner le curseur afficheur LCD sur ligne 2 et colonne 9 + i
    Afficher le caractère 255 (■)
FIN

Déboguer : transmission en série des valeurs tempareture, NMLI et Taux_MLI

♦ ALGORITHME DE LA PROCEDURE *MotorSpeedSetAB*

```
void MotorSpeedSetAB (byte MotorSpeedA, byte MotorSpeedB )
```

Prise en main du BUS et connexion au circuit L298
Temporisation de 1 ms
Écriture sur le bus de l'octet de commande permettant la modification de la valeur MLI
Écriture sur le bus de l'octet `MotorSpeedA`
Écriture sur le bus de l'octet `MotorSpeedB`
Temporisation de 10 ms
Fin de transmission : Libération du BUS

♦ ALGORITHME DE LA PROCEDURE *MotorDirectionSet*

```
byte MotorDirectionSet (unsigned char Direction)
```

Prise en main du BUS et connexion au circuit L298
Temporisation de 1 ms
Écriture sur le bus de l'octet de commande permettant la modification de sens de rotation
Écriture sur le bus de l'octet `Direction`
Écriture sur le bus de l'octet de commande `Nothing`
Temporisation de 10 ms
Fin de transmission : Libération du BUS

Remarque : Les symboles graphiques □ et ■ sont obtenus sur l'afficheur LCD par l'envoi des caractères **219** et **255**.

Exemple : lcd.print(char(219));

Pour un fonctionnement optimal, il vous faudra tout d'abord envoyer 10 caractères ' □' :

□□□□□□□□□□

Puis afficher les caractères ' ■ ' correspondant au taux de MLI :

VIT : 50% ■■■■■ □□□□□

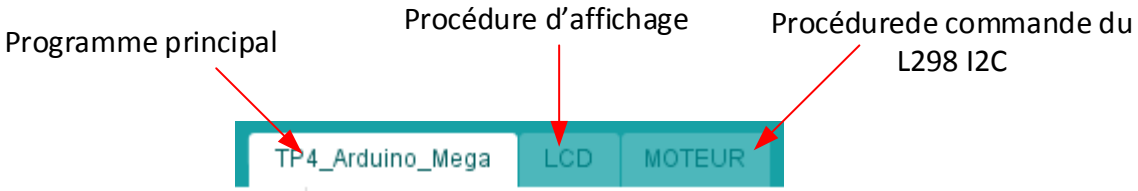
Il vous faudra donc deux boucles pour l'affichage :

- 10 itérations pour afficher le caractère 219 (□) à partir de la colonne 9
- un nombre '**n**' d'itérations pour afficher le caractère 255 (■) à partir de la colonne 9. '**n**' est déterminé à partir du taux de MLI et du nombre de caractères graphiques maximal affichable : 10

♦ REALISATION du PROGRAMME :

☞ Ouvrez le programme « *TP4_Arduino_Mega.ino* » contenu dans le dossier de ressources « *programme* »

Celui-ci est organisé en Onglets :

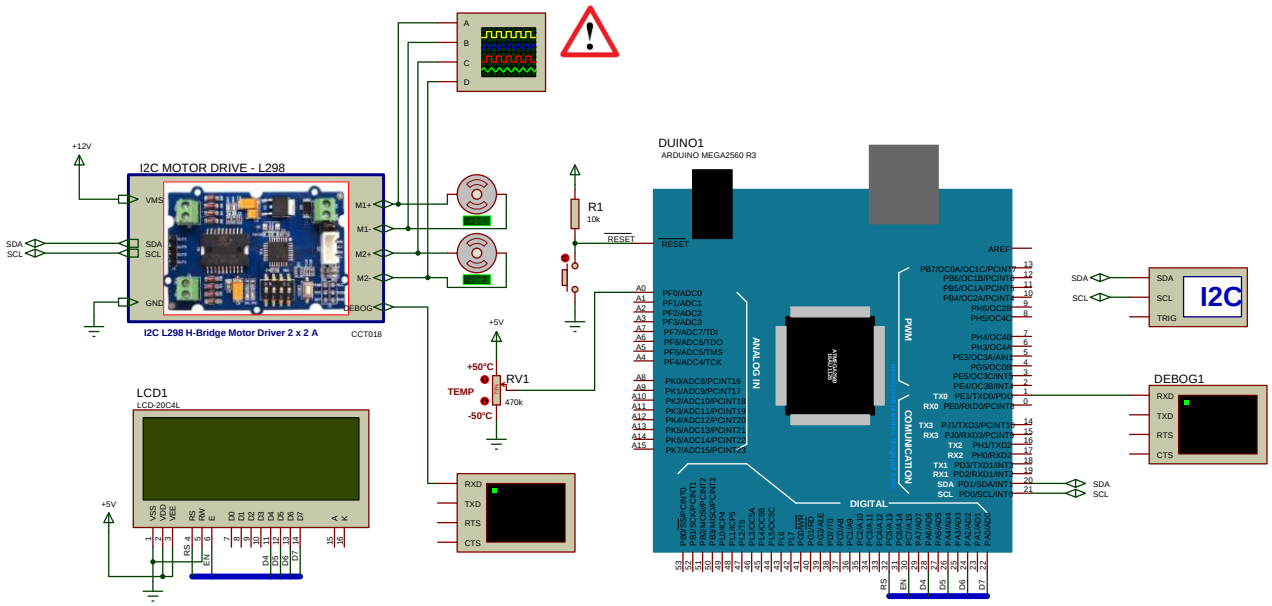


☞ Ouvrez le fichier Arduino puis complétez les différentes parties du programme en vous aidant du cahier des charges, à savoir :

- Procédure d'initialisation `setup()`
- Programme principal `loop()`
- Procédure **MotorSpeedSetAB**
- Procédure **MotorDirectionSet**
- Procédure **AfficheValeurs**

♦ **SIMULATION et MISE AU POINT du PROGRAMME** :

☞ Ouvrez le fichier de simulation **Proteus ISIS** « *TP4-COMMANDE-FAN.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. Utilisez le potentiomètre pour simuler différentes valeurs de température de 15°C à 50°C.

Vérifier que le taux de MLI varie bien comme définie dans l'étude préliminaire. 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 !