



INTERNET DES OBJETS

PROJET - DEVELOPPEMENT D'UNE STATION METEO CONNECTEE

Samuel Mertenat
Internet et Communication - Classe T2f
Kit ARDUINO #134

TABLE DES MATIERES

Historique du document	3
Introduction.....	3
Buts et Objectifs du Projet	3
Analyse	4
Arduino UNO	4
Arduino IDE	5
Le langage « Arduino »	5
Le courant électrique	8
La mémoire	9
Conception.....	10
Blink.....	10
Logging	10
Mémoire libre	11
Schéma de onnexion des capteurs.....	11
Réalisation	12
Blink.....	12
Logging	12
Mémoire libre	13
Tests et validation	14
TP01	14
Problèmes rencontrés & Solutions	16
Acquis	16
Perspectives	16
Conclusion	16
Annexe	16
Références	17

INTERNET DES OBJETS

PROJET - DEVELOPPEMENT D'UNE STATION METEO CONNECTEE

HISTORIQUE DU DOCUMENT

13/03/2015 : Rendu du TP01

- Installation de l'IDE et création d'un premier projet : « IoT »
- Développement d'un programme permettant de faire clignoter une LED et faisant appel au « Serial » afin d'afficher un compteur sur le terminal
- Développement d'un module de « tracing », permettant d'afficher des informations relatives à des erreurs, à du débogage ou simplement des informations
- Analyse du fonctionnement de la mémoire et création d'une méthode permettant d'afficher la quantité de mémoire Ram disponible

INTRODUCTION

Dans le cadre de ce projet, nous allons essayer de mettre en place une station météo connectée, de laquelle nous pourrions consulter les données depuis un Smartphone, via le Bluetooth Low Energy. La station météo sera basée sur un Arduino Uno, auquel nous associerons par la suite, capteurs et actuateurs, tels qu'un capteur de température et d'humidité ou une photorésistance. Le projet se déroulera sur plusieurs sessions de travaux pratiques durant lesquelles nous implémenterons, au fur et à mesure, les différentes fonctionnalités et concepts vus en cours.

BUTS ET OBJECTIFS DU PROJET

Ce projet a pour but de mettre en pratique la matière étudiée durant le cours « Internet des objets » en développant, petit à petit, une station météo.

Plus précisément :

- Mise en œuvre et utilisation d'une plateforme de prototypage Arduino
- Développement d'une station météo connectée
- Assimilation d'éléments particuliers et ciblés
 - Programmation en C/C++
 - Capteurs et actuateurs simples
 - Communication Bluetooth Low Energy

ANALYSE

ARDUINO UNO

L'Arduino UNO est une plateforme de développement très bon marché pour débiter à bricoler avec de l'électronique ou de l'automatisation. Son IDE est multiplateformes et l'Arduino ne nécessite, au minimum, que d'être branché à un port USB pour fonctionner.

Caractéristiques :

- | | |
|--|-----------------------------|
| • Micro contrôleur : | ATmega328 |
| • Tension d'alimentation interne : | 5V |
| • Tension d'alimentation (recommandée) : | 7 à 12V, limites : 6 à 20 V |
| • Entrées / sorties numériques : | 14 dont 6 sorties PWM |
| • Entrées analogiques : | 6 |
| • Courant max par broches E / S : | 40 mA |
| • Courant max sur sortie 3,3V : | 50mA |
| • Mémoire Flash : | 32 KB (bootloader 0.5 KB) |
| • Mémoire SRAM : | 2 KB |
| • Mémoire EEPROM : | 1 KB |
| • Fréquence d'horloge : | 16 MHz |
| • Dimensions : | 68.6mm x 53.3mm |
| • IDE de développement : | Arduino |

L'arduino Uno est donc un bon compris pour réaliser notre station météo. En effet, la plateforme est très répandue et de nombreux capteurs et actionneurs sont disponibles sur le marché. De plus, il ne consomme que peu d'énergie, ce qui en fait un bon candidat pour un système énergétiquement autonome. Cependant, la mémoire embarquée étant assez faible, il faudra être attentif lors de la partie conception et réalisation.

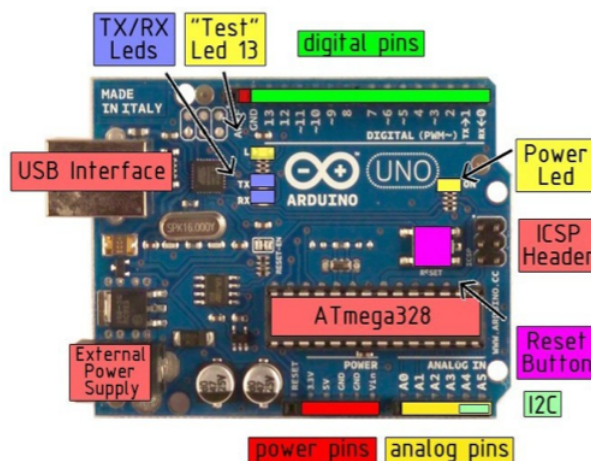


Figure 1: Carte Arduino Uno

ARDUINO IDE

Le logiciel de développement pour l'Arduino est disponible à l'adresse « <http://arduino.cc/en/Main/Software> » (version 1.6, Mac OS X édition).

L'interface est assez sommaire et permet de prendre rapidement le programme en mains.

Pour créer un nouveau projet, il suffit de cliquer sur « File > New » ; pour y ajouter un fichier : « Flèche vers le bas » (en haut à gauche) puis « New Tab ».

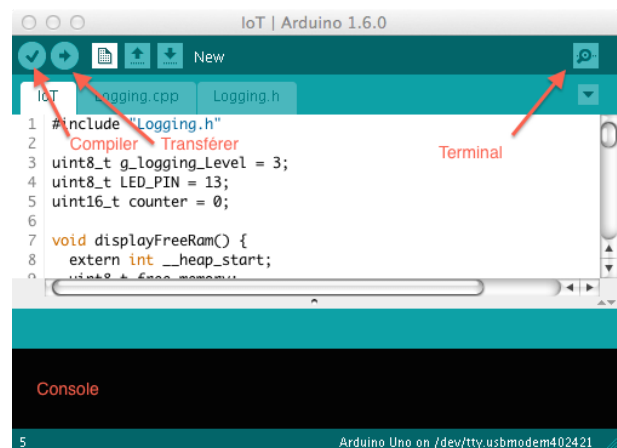


Figure 2: Interface de l'Arduino IDE

La compilation du code et le téléversement de celui-ci sur l'Arduino se font avec les boutons indiqués sur la capture ci-dessus.

Des bibliothèques tierces peuvent être utilisées et doivent être installées dans le dossier « Arduino/librairies »

LE LANGAGE « ARDUINO »

LE CODE MINIMAL

```

void setup()           //fonction d'initialisation de la carte
{
    //contenu de l'initialisation
}

void loop()            //fonction principale, elle se répète (s'exécute) à l'infini
{
    //contenu de votre programme
}
  
```

LES VARIABLES

Voilà les types de variables les plus répandus :

Type	Quel nombre il stocke ?	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
int	entier	-32 768 à +32 767	16 bits	2 octets
long	entier	-2 147 483 648 à +2 147 483 647	32 bits	4 octets
char	entier	-128 à +127	8 bits	1 octets
float	décimale	-3.4 x 10 ^{38} à +3.4 x 10 ^{38}	32 bits	4 octets
double	décimale	-3.4 x 10 ^{38} à +3.4 x 10 ^{38}	32 bits	4 octets

Voici le tableau des types non signés, on repère ces types par le mot unsigned (de l'anglais : non-signé) qui les précède :

Type	Quel nombre il stocke ?	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
unsigned char	entier non négatif	0 à 255	8 bits	1 octets
unsigned int	entier non négatif	0 à 65 535	16 bits	2 octets
unsigned long	entier non négatif	0 à 4 294 967 295	32 bits	4 octets

Une des particularités du langage Arduino est qu'il accepte un nombre plus important de types de variables.

Type	Quel nombre il stocke ?	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
byte	entier non négatif	0 à 255	8 bits	1 octets
word	entier non négatif	0 à 65535	16 bits	2 octets
boolean	entier non négatif	0 à 1	1 bits	1 octets

Exemples :

```
boolean variable = FALSE; // variable est fausse car elle vaut FALSE, du terme anglais "faux"
boolean variable = TRUE; // variable est vraie car elle vaut TRUE, du terme anglais "vrai"

int variable = 0; // variable est fausse car elle vaut 0
int variable = 1; // variable est vraie car elle vaut 1
int variable = 42; // variable est vraie car sa valeur est différente de 0

int variable = LOW; // variable est à l'état logique bas (= traduction de "low"), donc 0
int variable = HIGH; // variable est à l'état logique haut (= traduction de "high"), donc 1
```

LES TABLEAUX

```
float notes[20]; //on créer un tableau dont le contenu est vide, on sait simplement qu'il contiendra 20 nombres
float note[] = {0,0,0,0 /*, etc.* / };

float note[] = {};

void setup()
{
    note[0] = 0;
    note[1] = 0;
    note[2] = 0;
    note[3] = 0;
    //...
}
```

LES OPERATIONS LOGIQUES

IF ... ELSE IF

```
int prix_voiture = 5500;

if(prix_voiture < 5000)
{
    //la condition est vraie, donc j'achète la voiture
}

else if(prix_voiture == 5500)
{
    //la condition est vraie, donc j'achète la voiture
}

else
{
    //la condition est fausse, donc je n'achète pas la voiture
}
```

SWITCH

```
int options_voiture = 0;

switch (options_voiture)
{
    case 0:
        //il n'y a pas d'options dans la voiture
        break;
    default:
        //retente ta chance ;- )
        break;
}
```

CONDITION TERNAIRE

```
int prix_voiture = 5000;
int achat_voiture = FALSE;

achat_voiture= (prix_voiture == 5000) ? TRUE : FALSE;
```

WHILE

```
while(/* condition à tester */)
{
    //les instructions entre ces accolades sont répétées tant que la condition est vraie
}
```

DO WHILE

```
do
{
    //les instructions entre ces accolades sont répétées tant que la condition est vraie
}while(/* condition à tester */);
```

FOR

```
for(int compteur = 0; compteur < 5; compteur++)
{
    //code à exécuter
}
```

LES FONCTIONS

SANS PARAMETRE

```
void fonction()
{
    int var = 24;
    return var; //ne fonctionnera pas car la fonction est de type void
}
```

AVEC PARAMETRE(S)

```
int x = 64;
int y = 192;

void loop()
{
    maFonction(x, y);
}

int maFonction(int param1, int param2)
{
    int somme = 0;
    somme = param1 + param2;
    //somme = 64 + 192 = 255

    return somme;
}
```

L'UTILISATION DU « SERIAL »

Le « Serial » permet d'écrire et de recevoir des données via le terminal. Pour ce faire, il faut initialiser la « ligne » avec un « Serial.begin(9600) », 9600 étant la vitesse en bits par seconde.

Les données peuvent être ensuite lues et écrites avec les méthodes suivantes (cf. doc) : print(), write(), read(), etc.

```
uint8_t LED_PIN = 13;
uint16_t counter = 0;

// the setup function runs once when you press reset or power the board
void setup() {
    Serial.begin(9600);           // initiates the serial communication
    pinMode(LED_PIN, OUTPUT);    // sets the pin as output
}

// the loop function runs over and over again forever
void loop() {
    digitalWrite(LED_PIN, HIGH); // switches on the led
    Serial.print("The led is switched on; Number of times: ");
    Serial.println(++counter, DEC);
    delay(1000);                 // waits for 1000ms -> 1s
    digitalWrite(LED_PIN, LOW);  // switches off the led
    delay(1000);
}
```

Source : <http://arduino.cc/en/reference/serial>

DIVERS

Récupérer le temps : millis()

LE COURANT ELECTRIQUE

SENS DU COURANT

Le courant électrique se déplace selon un sens de circulation. Un générateur électrique, par exemple une pile, produit un courant. Et bien ce courant va circuler du pôle positif vers le pôle négatif de la pile, si et seulement si ces deux pôles sont reliés entre eux par un fil métallique ou un autre conducteur. Ceci, c'est le sens conventionnel du courant.

INTENSITE DU COURANT

On mesure la vitesse du courant, appelée intensité, en Ampères avec un Ampèremètre. En général, en électronique de faible puissance, on utilise principalement le milliampère (mA) et le micro-Ampère (μA), mais jamais bien au-delà.

TENSION

Autant le courant se déplace, ou du moins est un déplacement de charges électriques, autant la tension est quelque chose de statique. Pour bien définir ce qu'est la tension, sachez qu'on la compare à la pression d'un fluide.

LOI D'OHM

$$U = R * I$$

Dans le cas d'une LED, on considère, en général, que l'intensité la traversant doit-être de 20 mA. Si on veut être rigoureux, il faut aller chercher cette valeur dans le datasheet.

On a donc $I = 20 \text{ mA}$.

Ensuite, on prendra pour l'exemple une tension d'alimentation de 5V (en sortie de l'Arduino, par exemple) et une tension aux bornes de la LED de 1,2V en fonctionnement normal. On peut donc calculer la tension qui sera aux bornes de la résistance :

$$U_r = 5 - 1,2 = 3,8 \text{ V}$$

Enfin, on peut calculer la valeur de la résistance à utiliser :

$$\text{Soit : } R = U / I$$

$$R = 3,8 / 0,02$$

$$R = 190 \text{ Ohms}$$

LA MEMOIRE

L'Arduino ne disposant que de très peu de mémoire, il est important de l'utiliser avec parcimonie. Comme nous pouvons l'observer sur la figure ci-dessous, la mémoire RAM est divisée en 4 parties distinctes: « .data variables », « .bss variables », « heap » et « stack ». La mémoire à notre disposition, pour instancier des variables, débute à la partie « heap » (« __heap_start ») et se termine à la fin de la « stack » (« __brkval »).

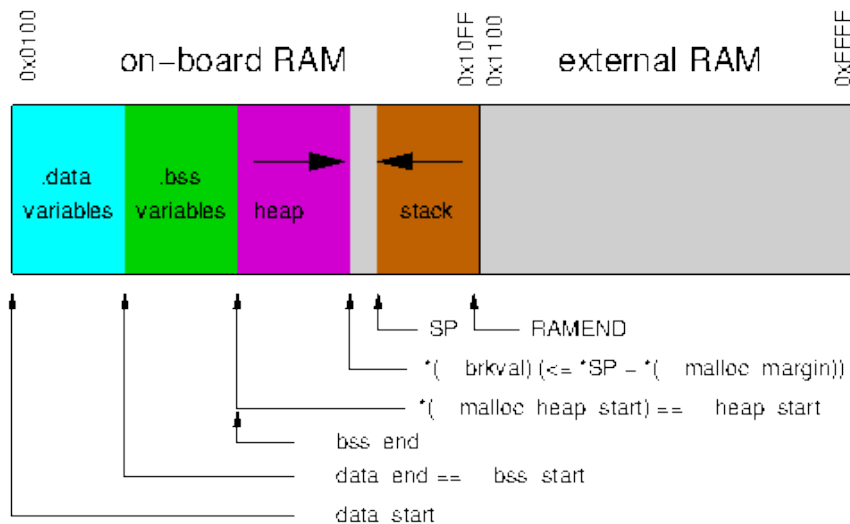


Figure 3: La mémoire

CONCEPTION

BLINK

A l'étape 5 du premier TP, nous devons nous inspirer du code de l'exemple « Blink » (Files > Examples > 01.Basics > Blink), faisant clignoter une la led 13 de l'Arduino toutes les deux secondes afin d'afficher à chaque allumage, un message sur la console, ainsi que le nombre de récurrence de cet événement. Pour réaliser ceci, il suffit de déclarer une nouvelle variable qui nous servira de compteur (« counter »), que l'on incrémente à chaque tour de la « loop ». L'affichage se fait ensuite en utilisant le « Serial », en débutant par l'initialiser dans la fonction « setup() » (« Serial.begin(9600) ») et en affichant ensuite un message avec l'instruction « Serial.println() ».

LOGGING

A l'étape 6 du premier TP, nous devons réaliser un système nous permettant de « logger » notre code, car la plateforme Arduino n'en propose pas par défaut. Le principe consiste à afficher les logs sur la console, selon leur type (ERROR, INFO, DEBUG) et après quel temps d'exécution ils ont eu lieu. D'autre part, le niveau de « logging », qui peut prendre une valeur allant de 1 à 3, permet de spécifier la quantité de logs, et donc d'économiser de la mémoire ($1 < 3$).

Deux fichiers sont nécessaires à l'implémentation d'un tel système :

- **Logging.cpp** : Module contenant les méthodes « PrintHeader() » permettant d'afficher le moment où le message de log est affiché (hh-mm-ss), ainsi que le type de log (ERROR, INFO, DEBUG) et la méthode « PrintSerial », permettant d'utiliser une macro à deux paramètres et donc, de formater sur une seule ligne, toutes ces infos.
- **Logging.h** : Entête du fichier Logging.cpp contenant des instructions pour le préprocesseur afin de définir des macros pour les 3 types de logs. Le niveau de log

définit ce que sera ou non compilé ; si le niveau de log est inférieur au niveau de log requis par les macros, celles-ci ne seront donc pas compilées.

MEMOIRE LIBRE

A l'étape 7 du premier TP, nous devons écrire une fonction permettant de calculer la quantité de mémoire libre. Bien que cette donnée soit affichée dans la console, il est intéressant de comprendre comment celle-ci est calculée.

Pour ce faire, il suffit d'instancier une variable (partie de la mémoire « stack »), d'en récupérer l'adresse et d'en soustraire l'adresse de la variable externe « `__heap_start` ».

SCHEMA DE CONNEXION DES CAPTEURS

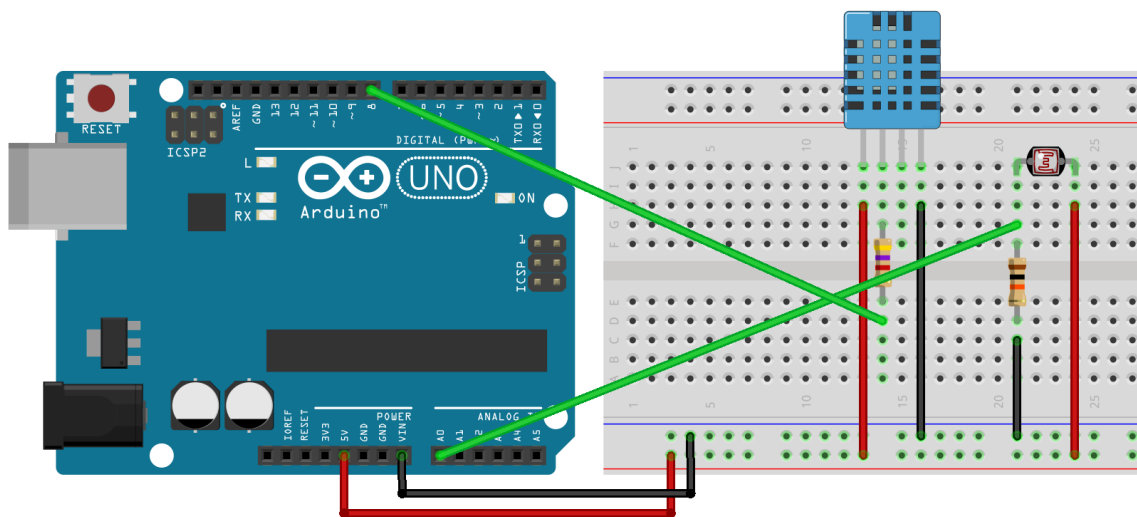


Figure 4: Schéma de connexion des capteurs

DHT11 :

- Pin 1 : 5 V
- Pin 2 : pin 8 (digitale) + résistance de 4.7 Ohms
- Pin 3 : terre

Photorésistance :

- Pin 1 : pin A0 (analogique) + résistance de 10k Ohms + terre
- Pin 2 : 5 V

REALISATION

BLINK

Afin de faire clignoter une led, nous devons tout d'abord définir le numéro du pin sur lequel elle est connectée.

```
uint8_t LED_PIN = 13;
```

Pour nous définissons le pin en tant que sortie, il nous permet ainsi d'y faire « sortir » du courant. Dans le cas d'un bouton, par exemple, on l'aurait défini en tant qu'entrée (INPUT).

```
pinMode(LED_PIN, OUTPUT); // sets the pin as output
```

Nous pouvons dès à présent utiliser la led. Pour cela, nous pouvons utiliser la méthode « digitalWrite() », qui permet d'allumer une led (niveau logique haut) ou d'éteindre une led (niveau logique bas). Dans le cas d'un « HIGH », la tension de sortie du pin sera donc de 5 V, ce qui permet d'alimenter leds, capteurs, etc.

```
digitalWrite(LED_PIN, HIGH); // switches on the led
delay(1000); // waits for 1000ms -> 1s
digitalWrite(LED_PIN, LOW); // switches off the led
delay(1000);
```

Quant à la fonction « delay() », elle permet de mettre en pause le programme pendant un temps t [ms] donné.

LOGGING

LOGGING.CPP

Les méthodes « PrintHeader() » et PrintSerial » sont appelées par les différentes macros de logging et permettent d'afficher convenablement les messages et arguments passés à celles-ci.

La méthode « PrintHeader() » permet d'afficher le temps d'exécution jusqu'à l'appel de l'une des macros de logging. Nous utilisons la méthode « millis() » pour récupérer le temps d'exécution [ms], duquel nous pouvons en tirer les heures / minutes / secondes à l'aide de divisions et de modulus.

```
void PrintHeader(const char* szHeaderType) {
    // returns the number of ms since the Arduino began running
    unsigned long time = millis();
    Serial.print(szHeaderType);
    Serial.print("\tat time : ");
    Serial.print(time / 3600000); // hours
    time = time % 3600000;
    Serial.print("h ");
    Serial.print(time / 60000); // minutes
    time = time % 60000;
    Serial.print("m ");
    Serial.print((time / 1000) % 60); // seconds
    Serial.print("s : \t");
}
```

La méthode « PrintSerial() » permet simplement d'afficher deux paramètres sur la même ligne, ce qui n'est pas possible avec un seul « Serial.print() ».

```
void PrintSerial(const char* format, int arg1) {
    Serial.print(format);
    Serial.println(arg1);
}
```

LOGGING.H

Le fichier « Logging.h » contient les différentes macros de logging. Nous débutons par déclarer la constante de préprocesseur « LOGGING_H », importer les librairies nécessaires au bon fonctionnement du programme, à « récupérer » la variable « g_logging_level », instanciée dans le fichier « Logging.cpp » et à déclarer une constante « TR_LOGLEVEL » à destination du préprocesseur, afin de définir le niveau de logging et donc, le niveau de compilation des différentes macros ci-dessous. Par exemple, en assignant la valeur 0 à « TR_LOGLEVEL », nous obtiendrons le programme le plus léger ; cependant, nous disposerons d'aucune information de logging dans la console.

```
#pragma once
#ifndef LOGGING_H
#define LOGGING_H

#include <stdio.h>
#include "Arduino.h"

extern uint8_t g_logging_Level;

//logging macros
#ifndef TR_LOGLEVEL
#define TR_LOGLEVEL 3
#endif
```

On « importe » ensuite les deux méthodes écrites dans le fichier « Logging.cpp », afin de pouvoir les appeler depuis les macros.

```
void PrintHeader(const char* szHeaderType);
void PrintSerial(const char* format, int arg1);
```

Ci-dessous, les deux macros permettant d'afficher des logs à titre informatif, qui nécessitent un (« TraceInfo() ») ou deux arguments (« TraceInfoFormat »). Le « résultat » de ces deux macros sera visible dans la console uniquement si la variable destinée au préprocesseur « TR_LOGLEVEL » est plus grande ou égale à 2 et que le programmeur ait décidé d'afficher ces logs (« g_logging_level » >= 2).

```
#if(TR_LOGLEVEL >= 2)
#define TraceInfo(format) if (g_logging_Level>=2) {PrintHeader("INFO"); Serial.println(format);}
#define TraceInfoFormat(format, arg1) if (g_logging_Level >= 2) { PrintHeader("INFO"); PrintSerial(format, arg1);}
#else
#define TraceInfo(format)
#define TraceInfoFormat(format, arg1)
#endif
```

Le « # » permet de définir des instructions pour le préprocesseur.

MEMOIRE LIBRE

Comme précédemment expliqué dans la partie conception, la mémoire libre peut être déterminée en soustrayant l'adresse d'une variable fraîchement instanciée par la première adresse disponible (« __heap_start »).

```
int displayFreeRam() {
    extern int __heap_start;
    uint8_t free_memory;
    return ((int)&free_memory) - ((int)&__heap_start);
}
```

TESTS ET VALIDATION

TP01

Afin d'attester le bon fonctionnement des macros de logging, nous pouvons incorporer des appels vers celles-ci à l'intérieur du code réalisé pour les étapes de ce présent TP. Nous utilisons donc les méthodes « `TraceErrorFormat()` », « `TraceInfoFormat()` » et `TraceDebugFormat()` » et les constantes « `TR_LOGLEVEL` » et « `g_logging_level` » afin de faire varier la taille du programme compilé et l'affichage en sortie sur la console.

En définissant un « `TR_LOGLEVEL` » à 0, nous obtenons un code compilé d'une taille de 184 bytes, ce qu'illustre le fait qu'aucune des macros n'a été compilée et donc, nous obtenons aucun message sur la console. Mais en définissant des niveaux de logging plus élevés, nous pouvons constater, comme le montrent très bien les figures ci-dessous, que la taille des programmes augmente, ainsi que la quantité d'information affichée sur la console.

LOGGING DE NIVEAU 1

Niveau de logging à 1 : affichage sur la console que des messages d'erreur ; taille du programme de 242 bytes.

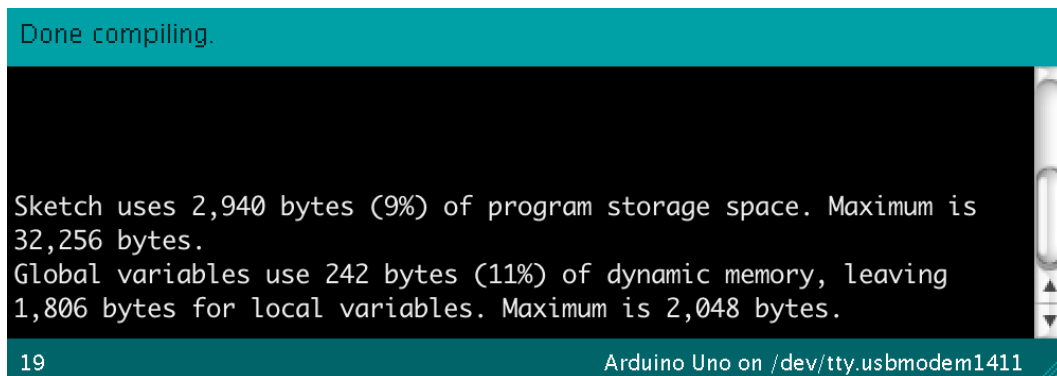


Figure 5: Compilation du programme avec un niveau de logging de 1

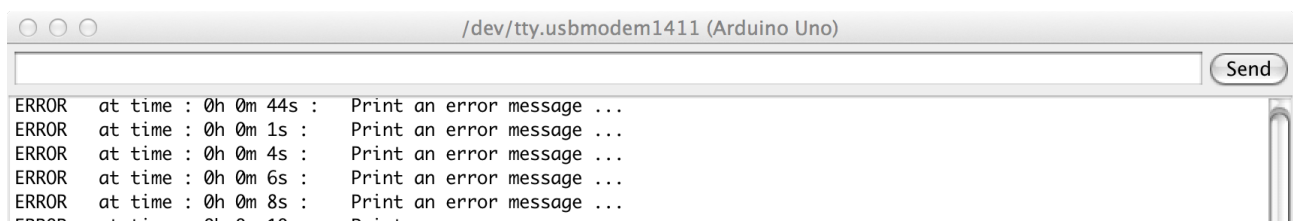
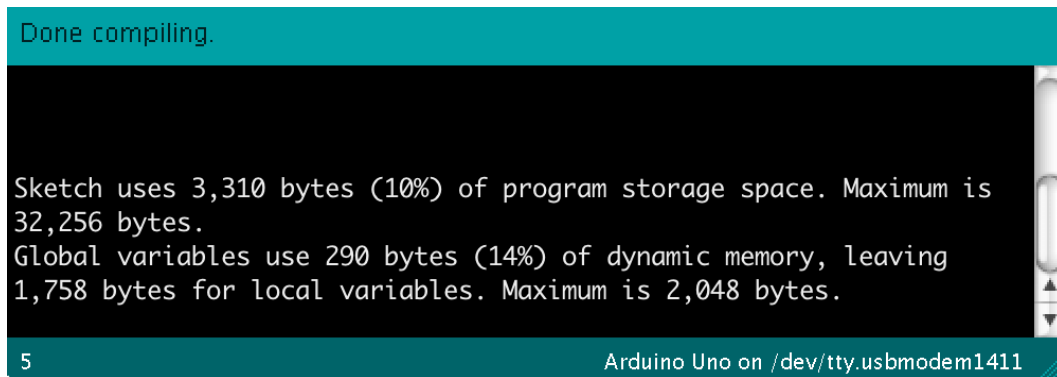


Figure 6: Sortie sur la console

LOGGING DE NIVEAU 2

Niveau de logging à 2 : affichage sur la console des messages d'erreur et d'information; taille du programme de 290 bytes.

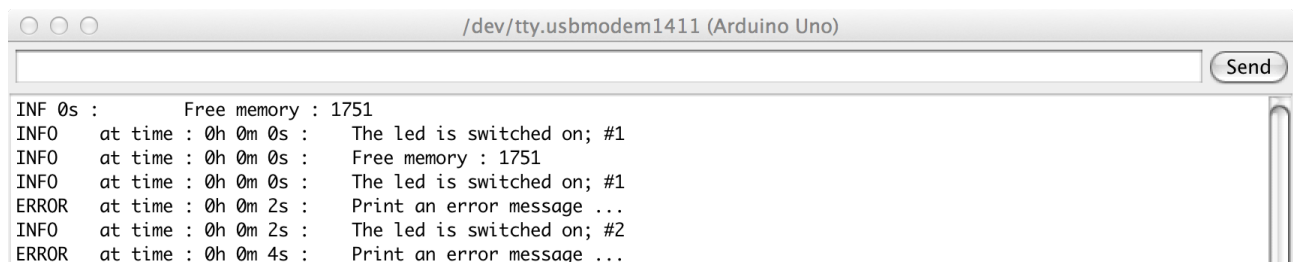


```
Done compiling.

Sketch uses 3,310 bytes (10%) of program storage space. Maximum is
32,256 bytes.
Global variables use 290 bytes (14%) of dynamic memory, leaving
1,758 bytes for local variables. Maximum is 2,048 bytes.

5 Arduino Uno on /dev/tty.usbmodem1411
```

Figure 7: Compilation du programme avec un niveau de logging de 2



```
/dev/tty.usbmodem1411 (Arduino Uno) Send

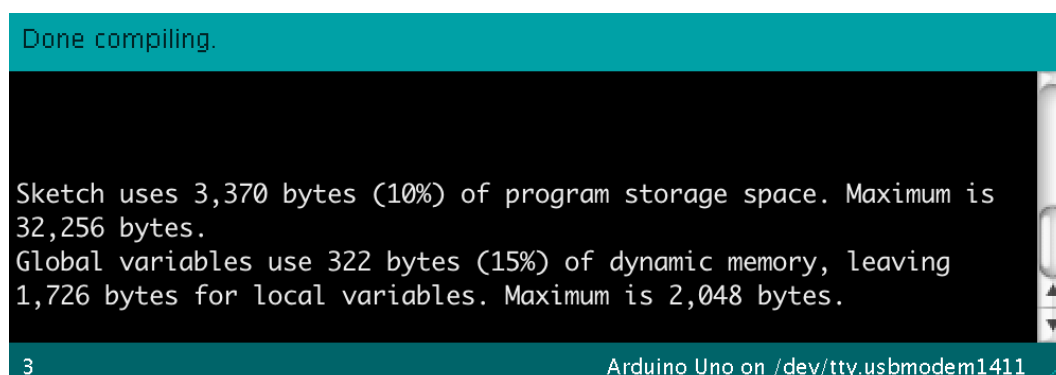
INF 0s :      Free memory : 1751
INFO  at time : 0h 0m 0s :    The led is switched on; #1
INFO  at time : 0h 0m 0s :    Free memory : 1751
INFO  at time : 0h 0m 0s :    The led is switched on; #1
ERROR at time : 0h 0m 2s :    Print an error message ...
INFO  at time : 0h 0m 2s :    The led is switched on; #2
ERROR at time : 0h 0m 4s :    Print an error message ...
```

Figure 8: Sortie sur la console

LOGGING DE NIVEAU 3

Niveau de logging à 3 : affichage sur la console de tous les types de logging (ERROR, INFO & DEBUG) ; taille du programme de 242 bytes.

Avec cette figure, nous pouvons aussi attester du bon fonctionnement de la méthode de calcul de la mémoire libre, qui retourne une valeur de 1719. Ce n'est, certes, pas identique à la valeur affichée dans la console, mais celle-ci en est très proche (+ 7 bytes).

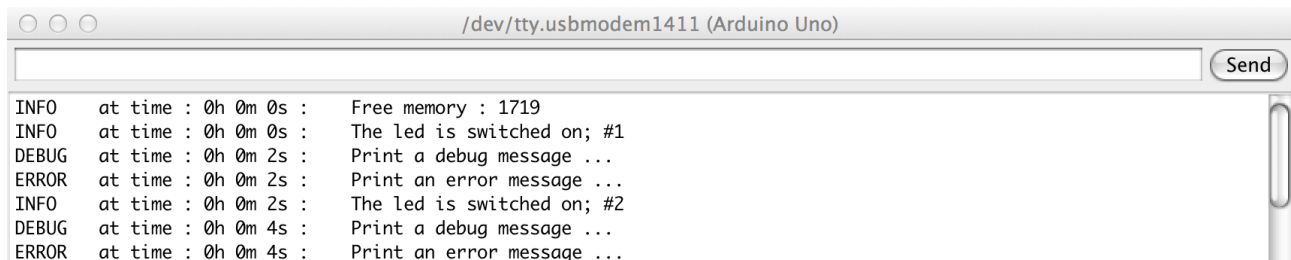


```
Done compiling.

Sketch uses 3,370 bytes (10%) of program storage space. Maximum is
32,256 bytes.
Global variables use 322 bytes (15%) of dynamic memory, leaving
1,726 bytes for local variables. Maximum is 2,048 bytes.

3 Arduino Uno on /dev/tty.usbmodem1411
```

Figure 9: Compilation du programme avec un niveau de logging à 3

The image shows a screenshot of a serial monitor window titled "/dev/tty.usbmodem1411 (Arduino Uno)". The window has a text input field at the top and a "Send" button. Below the input field, there is a scrollable area containing the following log output:

```
INFO    at time : 0h 0m 0s :   Free memory : 1719
INFO    at time : 0h 0m 0s :   The led is switched on; #1
DEBUG   at time : 0h 0m 2s :   Print a debug message ...
ERROR   at time : 0h 0m 2s :   Print an error message ...
INFO    at time : 0h 0m 2s :   The led is switched on; #2
DEBUG   at time : 0h 0m 4s :   Print a debug message ...
ERROR   at time : 0h 0m 4s :   Print an error message ...
```

Figure 10: Sortie sur la console

Synthèse des tests réalisés :

- ✓ Affichage de la valeur du compteur à chaque itération
- ✓ Affichage des logs de niveau 1
- ✓ Affichage des logs de niveaux 1-2
- ✓ Affichage des logs de niveaux 1-3
- ✓ Variation de la taille du programme compilé en fonction du niveau de logging
- ✓ Affichage de la quantité de mémoire libre au démarrage du programme

PROBLEMES RENCONTRES & SOLUTIONS

13/03/2015 : Rendu du TP01

- J'ai rencontré quelques difficultés à comprendre le fonctionnement des macros de logging, mais après plusieurs explications, tout est rentré dans l'ordre.

ACQUIS

TP01 :

- Ce premier travail pratique a été l'occasion d'appivoiser la plateforme Arduino et de prendre en mains son IDE. D'autre part, il nous a permis de comprendre le fonctionnement de macros de logging et d'appréhender le fonctionnement de la mémoire sur l'Arduino.

PERSPECTIVES

Cette première approche avec l'Arduino laisse entrevoir de belles opportunités, en y branchant, par exemple, des capteurs afin de démarrer notre projet de station météo connectée.

CONCLUSION

ANNEXE

Le code source du projet se trouve sur Git, à l'adresse : <https://forge.tic.eia-fr.ch/git/samuel.mertenat/iot-project>.

- TP01 → <https://forge.tic.eia-fr.ch/git/samuel.mertenat/iot-project/tree/master/tp01>

REFERENCES

- <http://www.louisreynier.com/fichiers/KesacoArduino.pdf>
- <http://openclassrooms.com> (cours sur l'Arduino, plus disponible)
- <http://arduino.cc/en/Reference/Serial>
- <http://playground.arduino.cc/Code/AvailableMemory>
- <http://www.nongnu.org/avr-libc/user-manual/malloc.html>