PROGRAMMATION D'UN ARDUINO

ARDUINO UNO

PROGRAMMATION D'UN ARDUINO

Préambule

Le langage Arduino est très proche du C et du C++.

En particulier, utiliser le langage C pour programmer l'Arduino permet généralement de créer des programmes plus petits et davantage optimisés, avec un contrôle plus fin des tâches effectuées. Le C est adopté dans le monde entier pour la programmation de microprocesseurs, car il offre un bon compromis entre l'effort de développement et l'efficacité du programme, mais aussi, en raison de son histoire, il existe des bibliothèques optimisées, une documentation très complète et des manières de résoudre les problèmes. Donc, si vous trouvez que le langage Arduino crée des programmes trop lourds ou trop lents, et que vous préférez tirer parti au maximum des performances de votre carte, ou si vous voulez une approche plus modulaire, réécrire vos programmes en langage C pourrait être le bon choix à faire.

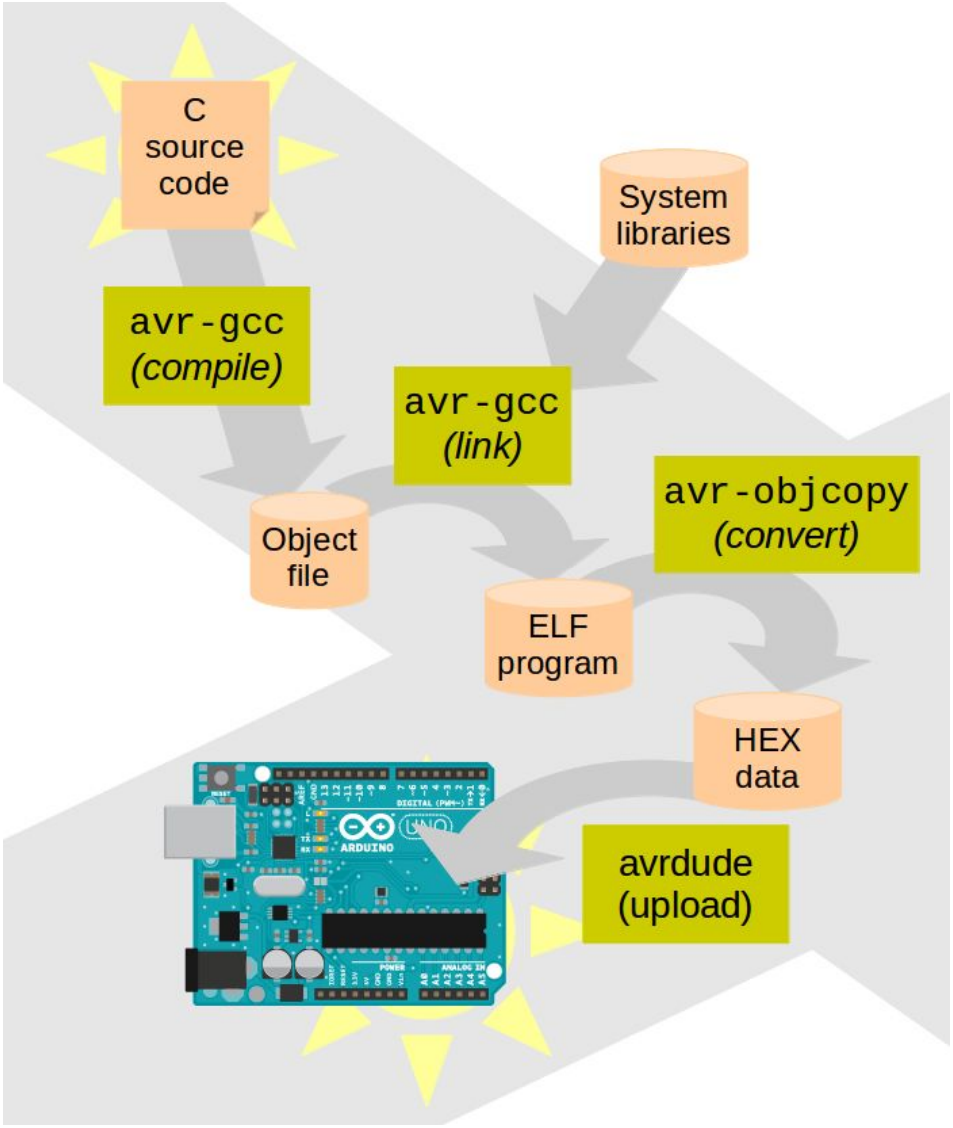
PROGRAMMATION D'UN ARDUINO

Préambule

Heureusement, tous les outils sont déjà présents, dissimulés sous le capot dans l'EDI Arduino. Dans mon cas particulier, comme je développe sous Linux, Arduino utilise le compilateur avr-gcc et avrdude pour téléverser les programmes. Je peux donc utiliser ces outils pour développer un programme en langage C, au lieu du langage Arduino, et téléverser ce programme dans la carte. Dans les préférences de l'EDI Arduino, des options permettent d'augmenter la quantité d'informations et de notifications affichées à l'écran lorsqu'un programme est compilé et téléversé. Voilà qui est très utile pour comprendre l'enchaînement des commandes du processus de compilation C/C++ qui sont lancées en coulisse dans l'interface graphique. On peut reproduire ce processus, compiler notre programme en lançant la commande avr-gcc avec les bonnes options, puis exécuter avr-dude avec les bonnes options pour le téléversement. Le diagramme ci-dessous montre toute la chaîne de compilation, du code source en langage C jusqu'au téléversement dans la carte.

PROGRAMMATION D'UN ARDUINO

Préambule



Du code source en langage C jusqu'au téléversement dans la carte Arduino avec la chaîne de compilation avr-gcc.

PROGRAMMATION D'UN ARDUINO

Préambule

Le langage Arduino est très proche du C et du C++.

Nous avons :

1. La syntaxe du langage
2. Les variables
3. Les conditions

PROGRAMMATION D'UN ARDUINO

La syntaxe du langage

La syntaxe d'un langage de programmation est l'ensemble des règles d'écritures liées à ce langage.

Le code minimal avec Arduino: nous devons utiliser un code minimal lorsque l'on crée un programme. Ce code permet de diviser le programme que nous allons créer en deux grosses parties.

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

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

PROGRAMMATION D'UN ARDUINO

La fonction Setup()

Dans ce code se trouvent deux fonctions. Les fonctions sont en fait des portions de code.

```
//fonction d'initialisation de la carte

void setup()
{
   //contenu de l'initialisation
   //on écrit le code à l'intérieur
}
```

Cette fonction **setup()** est appelée une seule fois lorsque le programme commence. C'est pourquoi c'est dans cette fonction que l'on va écrire le code qui n'a besoin d'être exécuté une seule fois. On appelle cette fonction : « **fonction d'initialisation**« .

PROGRAMMATION D'UN ARDUINO

La fonction loop()

C'est donc dans cette fonction **loop()** où l'on va écrire le contenu du programme. Il faut savoir que cette fonction est appelée en permanence, c'est-à-dire qu'elle est exécutée une fois, puis lorsque son exécution est terminée, on la ré-exécute et encore et encore. On parle de **boucle infinie**.

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

A titre informatif, on n'est pas obligé d'écrire quelque chose dans ces deux fonctions. En revanche, il est **obligatoire** de les écrire, même si elles ne contiennent aucun code !

PROGRAMMATION D'UN ARDUINO

Soit un exemple d'un programme avec les deux fonctions setup() et loop()

Tout ce qui commence par // est un commentaire qui ne s'exécute pas

```
void setup() {
  // initialize la pin 13 (digital pin) comme SORTIE.
  pinMode(13, OUTPUT);
}

// La fonction loop() va être exécutée en continue
void loop() {
  digitalWrite(13, HIGH);   // La sortie 13 se met en niveau HAUT soit 5V
  delay(1000);                 // On attend 1s ou bien 1000ms
  digitalWrite(13, LOW);    // La sortie 13 se met en niveau BAS soit 0V
  delay(1000);                 // wait for a second
}
```

PROGRAMMATION D'UN ARDUINO

Les instructions

Dans ces fonctions, on écrit quoi ?

Les instructions sont des lignes de code qui disent au programme : « fait ceci, fait cela, … » C'est tout bête mais très puissant car c'est ce qui va orchestrer votre programme.

Les points virgules

Les points virgules terminent les instructions.

Attention: Les points virgules (**;**) sont synonymes d'erreurs car il arrive très souvent de les oublier à la fin des instructions. Par conséquent le code ne marche pas et la recherche de l'erreur peut nous prendre un temps conséquent ! Donc faites bien attention.

PROGRAMMATION D'UN ARDUINO

Les accolades

Les accolades sont les « conteneurs » du code du programme. Elles sont propres aux fonctions, aux conditions et aux boucles. Les instructions du programme sont écrites à l'intérieur de ces accolades. Parfois elles ne sont pas obligatoires dans les *conditions*, mais il est recommandable de les **mettre tout le temps** ! Cela rendra plus lisible votre programme.

Les commentaires

Pour finir, on va voir ce qu'est un commentaire. J'en ai déjà mis dans les exemples de codes. Ce sont des lignes de codes qui seront ignorées par le programme. Elles ne servent en rien lors de l'exécution du programme.

Les accents

Il est formellement interdit de mettre des accents en programmation. Sauf dans les commentaires.

PROGRAMMATION D'UN ARDUINO

Les variables

Il y a plusieurs types de mémoire (RAM, NVRAM, RAMCMOS, ROM, PROM, EPROM, EEPROM …etc) .

Nous nous occuperons seulement de la mémoire « vive » (RAM) et de la mémoire « morte » effaçable électriquement (EEPROM).

Exemple: Imaginons que vous avez connecté un bouton poussoir sur une broche de votre carte Arduino. Comment allez-vous stocker l'état du bouton (appuyé ou éteint) ?

Une variable, qu'est ce que c'est ?

Une **variable est un nombre**. Ce nombre est stocké dans un espace de la mémoire vive (RAM) du microcontrôleur.
Ce nombre a la particularité de changer de valeur. Une variable est en fait le **conteneur** du nombre en question. Et ce conteneur va être stocké dans une case de la mémoire.
Si on matérialise cette explication par un schéma, cela donnerait :
nombre => variable => mémoire
le symbole « => » signifiant : « est contenu dans… »

PROGRAMMATION D'UN ARDUINO

Le nom d'une variable

Le nom de variable accepte quasiment tous les caractères sauf :
. (le point)
, (la virgule)
é,à,ç,è (les accents)

En fait, il n'accepte que l'alphabet alphanumérique ([a-z], [A-Z], [0-9]) et _ (underscore)

Définir une variable

Si on donne un nombre à notre programme, il ne sait pas si c'est une variable ou pas. Il faut le lui indiquer. Pour cela, on donne un **type** aux variables. Oui, car il existe plusieurs types de variables ! Par exemple la variable « x » vaut 4 :

```
x = 4;
```

PROGRAMMATION D'UN ARDUINO

Définir une variable

Et bien ce code ne fonctionnerait pas car il ne suffit pas ! En effet, il existe une multitude de nombres : les nombres entiers, les nombres décimaux, … C'est pour cela qu'il faut assigner une variable à un type. 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

PROGRAMMATION D'UN ARDUINO

Définir une variable

Par exemple, si notre variable « x » ne prend que des valeurs entières, on utilisera les types **int**, **long**, ou **char**. Si maintenant la variable « x » ne dépasse pas la valeur 64 ou 87, alors on utilisera le type **char**

char x = 0;

Si en revanche x = 260, alors on utilisera le type supérieur (qui accepte une plus grande quantité de nombre) à **char**, autrement dit **int** ou **long**

PROGRAMMATION D'UN ARDUINO

Définir une variable

On peut par exemple penser, pour éviter les dépassements de valeur ont met tout dans des double ou long !
Oui, mais un microcontrôleur, ce n'est pas un ordinateur 2GHz multicore, 4Go de RAM ! Ici on parle d'un système qui fonctionne avec un CPU à 16MHz (soit 0,016 GHz) et 2 Ko de SRAM pour la mémoire vive. Donc deux raisons font qu'il faut choisir ses variables de manière judicieuse :
– La RAM n'est pas extensible, quand il y en a plus, y en a plus !
– Le processeur est de type 8 bits (sur Arduino UNO), donc il est optimisé pour faire des traitements sur des variables de taille 8 bits, un traitement sur une variable 32 bits prendra donc (beaucoup) plus de temps !
Si à présent notre variable « x » ne prend jamais une valeur négative (-20, -78, …), alors on utilisera un type **non-signé**. C'est à dire, dans notre cas, un **char** dont la valeur n'est plus de -128 à +127, mais de 0 à 255. Voici le tableau des types non signés, on repère ces types par le mot **unsigned** (de l'anglais : non-signé) qui les précède :

PROGRAMMATION D'UN ARDUINO

Définir une variable

Si à présent notre variable « x » ne prend jamais une valeur négative (-20, -78, …), alors on utilisera un type **non-signé**. C'est à dire, dans notre cas, un **char** dont la valeur n'est plus de -128 à +127, mais de 0 à 255. 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				

PROGRAMMATION D'UN ARDUINO

Définir une variable

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. Je vous les liste dans ce tableau :

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

PROGRAMMATION D'UN ARDUINO

Les variables booléennes

Les variables **booléennes** sont des variables qui ne peuvent prendre que deux valeurs : ou VRAI ou FAUX. Elles sont utilisées notamment dans les boucles et les conditions. Nous verrons pourquoi. Une variable booléenne peut être définie de plusieurs manières :

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

PROGRAMMATION D'UN ARDUINO

Les variables booléennes

Quand une variable vaut « 0 », on peut considérer cette variable comme une variable booléenne, elle est donc fausse. En revanche, lorsqu'elle vaut « 1 » ou n'importe quelle valeurs différente de zéro, on peut aussi la considérer comme une variable booléenne, elle est donc vraie. Voilà un exemple :

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

PROGRAMMATION D'UN ARDUINO

Les variables booléennes

Le langage Arduino accepte aussi une troisième forme d'écriture (qui lui sert pour utiliser les broches de sorties du microcontrôleur) :

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

Par exemple: Nous nous servirons de cette troisième écriture pour allumer et éteindre des lumières…

PROGRAMMATION D'UN ARDUINO

Les opérations « simples »

On va voir à présent les opérations qui sont possibles avec le langage Arduino (addition, multiplication, …). Je vous vois tout de suite dire : « Mais pourquoi on fait ça, on l'a fait en primaire ! 🤦 » Et bien parce que c'est quelque chose d'essentiel, car on pourra ensuite faire des opérations avec des variables. Vous verrez, vous changerez d'avis après avoir lu la suite !

L'addition

Vous savez ce que c'est, pas besoin d'explications. Voyons comment on fait cette opération avec le langage Arduino. Prenons la même variable que tout à l'heure :

```
//définition de la variable x
int x = 0;

//on change la valeur de x par une opération simple
x = 12 + 3;
// x vaut maintenant 12 + 3 = 15
```

PROGRAMMATION D'UN ARDUINO

Les opérations « simples »

L'addition

Faisons maintenant une addition de variables :

```
//définition de la variable x et assignation à la valeur 38
int x = 38;
int y = 10;
int z = 0;
//faisons une addition
// on a donc z = 38 + 10 = 48
z = x + y;
```

PROGRAMMATION D'UN ARDUINO

Les opérations « simples »

La soustraction

On peut reprendre les exemples précédents, en faisant une soustraction :

```
 /définition de la variable x
int x = 0;

//on change la valeur de x par une opération simple
x = 12 - 3;
// x vaut maintenant 12 - 3 = 9
```

Soustraction de variables :

```
int x = 38;   //définition de la variable x et assignation à la valeur 38
int y = 10;
int z = 0;

z = x - y;    // on a donc z = 38 - 10 = 28
```

PROGRAMMATION D'UN ARDUINO

Les opérations « simples »

La Multiplication

```
int x = 0;
int y = 10;
int z = 0;
x = 12 * 3; // x vaut maintenant 12 * 3 = 36
z = x * y; // on a donc z = 36 * 10 = 360
// on peut aussi multiplier (ou toute autre opération) un nombre et une variable :
z = z * ( 1 / 10 ); //soit z = 360 * 0.1 = 36
```

PROGRAMMATION D'UN ARDUINO

Les opérations « simples »

La Division

```
float x = 0;
float y = 15;
float z = 0;

x = 12 / 2; // x vaut maintenant 12 / 2 = 6

z = y / x; // on a donc z = 15 / 6 = 2.5
```

PROGRAMMATION D'UN ARDUINO

Les opérations « simples »

La Division

Attention cependant, si vous essayer de stocker le résultat d'une division dans une variable de type char, int ou long, le résultat sera stocké sous la forme d'un entier arrondi au nombre inférieur. Par exemple dans le code précédent si on met z dans un int on aura :

```
float x = 0;
float y = 15;
int z = 0;

x = 12 / 2; // x vaut maintenant 12 / 2 = 6

z = y / x; // on a donc z = 15 / 6 = 2 !
```

PROGRAMMATION D'UN ARDUINO

Les opérations « de base »

Le Modulo

Le modulo est une opération de base, certes moins connue que les autres. Cette opération permet d'obtenir le reste d'une division.

*18 % 6 // le reste de l'opération est 0, car il y a 3*6 dans 18 donc 18 - 18 = 0*
*18 % 5 // le reste de l'opération est 3, car il y a 3*5 dans 18 donc 18 - 15 = 3*

Le modulo est utilisé grâce au symbole %. C'est tout ce qu'il faut retenir. Autre exemple

```
int x = 24;
int y = 6;
int z = 0;

z = x % y; // on a donc z = 24 % 6 = 0 (car 6 * 4 = 24)
```

Le modulo ne peut-être fait que sur des nombres entiers

PROGRAMMATION D'UN ARDUINO

Les opérations « de base »

Quelques opérations pratiques

Voyons un peu d'autres opérations qui facilitent parfois l'écriture du code.

L'incrémentation

```
var = 0;
var++; //c'est cette ligne de code qui nous intéresse
```

»var++; » revient à écrire : « var = var + 1; » En fait, on ajoute le chiffre 1 à la valeur de *var*. Et si on répète le code un certain nombre de fois, par exemple 30, et bien on aura *var* = 30.

La décrémentation

C'est l'inverse de l'incrémentation. Autrement dit, on enlève le chiffre 1 à la valeur de var.

```
var = 30;
var--; //décrémentation de var
```

PROGRAMMATION D'UN ARDUINO

Les opérations composées

Il existe des raccourcis lorsque l'on veut effectuer une opération sur une même variable :

```
int x, y;
x += y; // correspond à x = x + y;
x -= y; // correspond à x = x - y;
x *= y; // correspond à x = x * y;
x /= y; // correspond à x = x / y;
```

PROGRAMMATION D'UN ARDUINO

Les opérations composées

Avec un exemple, cela donnerait :

```
int var = 10;
//opération 1
var = var + 6;
var += 6; //var = 16
//opération 2
var = var - 6;
var -= 6; //var = 4
//opération 3
var = var * 6;
var *= 6; //var = 60
//opération 4
var = var / 5;
var /= 5; //var = 2
```

PROGRAMMATION D'UN ARDUINO

L'opération de bascule (ou « inversion d'état »)

Nous l'utiliserons notamment lorsque l'on voudra faire clignoter une lumière. Sans plus attendre, voilà cette astuce :

```
//on définit une variable x qui ne peut prendre que la valeur 0 ou 1 (vraie ou fausse)
boolean x = 0;

x = 1 - x;  //!
```

PROGRAMMATION D'UN ARDUINO

L'opération de bascule (ou « inversion d'état »)

Analysons cette instruction. A chaque exécution du programme (oui, il se répète jusqu'à l'infini), la variable x va changer de valeur :
1er temps : x = 1 – x soit x = 1 – 0 donc x = 1
2e temps : x = 1 – x or x vaut maintenant 1 donc x = 1 – 1 soit x = 0
3e temps : x vaut 0 donc x = 1 – 0 soit x = 1
Ce code se répète donc et à chaque répétition, la variable x change de valeur et passe de 0 à 1, de 1 à 0, de 0 à 1, etc. Il agit bien comme une bascule qui change la valeur d'une variable booléenne. En mode console cela donnerait quelque chose du genre (n'essayez pas cela ne marchera pas, c'est un exemple) :

x = 0
x = 1
x = 0
x = 1
x = 0
…

PROGRAMMATION D'UN ARDUINO

L'opération de bascule (ou « inversion d'état »)

Mais il existe d'autres moyens d'arriver au même résultat. Par exemple, en utilisant l'opérateur '!' qui signifie « not » (« non »). Ainsi, avec le code suivant on aura le même fonctionnement :

```
x = !x;
```

Puisqu'à chaque passage x devient « pas x » donc si x vaut 1 son contraire sera 0 et s'il vaut 0, il deviendra 1.

PROGRAMMATION D'UN ARDUINO

Les conditions

Qu'est-ce qu'une condition

C'est un choix que l'on fait entre plusieurs propositions. En informatique, les conditions servent à tester des variables.

Quelques symboles

Pour tester des variables, il faut connaître quelques symboles. Ci-joint le tableau suivant:

PROGRAMMATION D'UN ARDUINO

Les conditions

Quelques symboles

Ci-dessous le tableau en question

Symbole	A quoi il sert	Signification
==	Ce symbole, composé de deux égales, permet de tester l'égalité entre deux variables	… est égale à …
<	Celui-ci teste l'infériorité d'une variable par rapport à une autre	…est inférieur à…
>	Là c'est la supériorité d'une variable par rapport à une autre	…est supérieur à…
<=	teste l'infériorité ou l'égalité d'une variable par rapport à une autre	…est inférieur ou égale à…
>=	teste la supériorité ou l'égalité d'une variable par rapport à une autre	…est supérieur ou égal à…
!=	teste la différence entre deux variables	…est différent de…

PROGRAMMATION D'UN ARDUINO

Les conditions

En informatique, on parle de **condition**. « si la condition est vraie », on fait une action.
En revanche « si la condition est fausse », on exécute une autre action.

If…else
La première condition que nous verrons est la condition if…else. Voyons un peu le

fonctionnement. *If*

On veut tester la valeur d'une variable.
int prix_voiture = 480000; //variable : prix de la voiture définit à 480000DA

PROGRAMMATION D'UN ARDUINO

Les conditions

D'abord on définit la variable « prix_voiture ». Sa valeur est de 480000DA. Ensuite, on doit tester cette valeur. Pour tester une condition, on emploie le terme *if* (de l'anglais « si »). Ce terme doit être suivi de parenthèses dans lesquelles se trouveront les variables à tester. Donc entre ces parenthèses, nous devons tester la variable prix_voiture afin de savoir si elle est inférieure à 500000DA.

```
if(prix_voiture < 500000)
{
   //la condition est vraie, donc j'achète la voiture
}
```

PROGRAMMATION D'UN ARDUINO

Les conditions

fonctionnement. *If*

On peut lire cette ligne de code comme ceci : « **si** la variable *prix_voiture* est inférieure à 500000, on exécute le code qui se trouve entre les accolades.
Les instructions qui sont entre les accolades ne seront exécutées que si la condition testée est vraie !
Le « schéma » à suivre pour tester une condition est donc le suivant :

```
if(/* contenu de la condition à tester */)
{
   //instructions à exécuter si la condition est vraie
}
```

PROGRAMMATION D'UN ARDUINO

Les conditions

Fonctionnement. else

On a pour l'instant testé que si la condition est vraie. Maintenant, nous allons voir comment faire pour que d'autres instructions soient exécutées si la condition est fausse. Le terme *else* de l'anglais « sinon » implique notre deuxième choix si la condition est fausse. *Par exemple, si le prix de la voiture est inférieur à 500000DA, alors je l'achète. Sinon, je ne l'achète pas.* Pour traduire cette phrase en ligne de code, c'est plus simple qu'avec un if, il n'y a pas de parenthèses à remplir :

```
int prix_voiture = 5500;
if(prix_voiture < 5000)
{
   //la condition est vraie, donc j'achète la voiture
}
else
{
   //la condition est fausse, donc je n'achète pas la voiture
}
```

PROGRAMMATION D'UN ARDUINO

Les conditions

Fonctionnement. else

Le *else* est généralement utilisé pour les **conditions dites de défaut**. C'est lui qui à le pouvoir sur toutes les conditions, c'est-à-dire que si aucune condition n'est vraie, on exécute les instructions qu'il contient.

Le *else* n'est pas obligatoire, on peut très bien mettre plusieurs *if* à la suite.

Le « schéma » de principe à retenir est le suivant :

```
else  // si toutes les conditions précédentes sont fausses...
{
   //...on exécute les instructions entre ces accolades
}
```

PROGRAMMATION D'UN ARDUINO

Les conditions

Fonctionnement. Else if

A ce que je vois, on a pas trop le choix : soit la condition est vraie, soit elle est fausse. Il n'y a pas d'autres possibilités ? o_O
Bien sur que l'on peut tester d'autres conditions ! Pour cela, on emploie le terme *else if* qui signifie « sinon si… » *Par exemple, SI le prix de la voiture est inférieur à 500000DA je l'achète; SINON SI elle est égale à 550000DA mais qu'elle a l'option GPS en plus, alors je l'achète ; SINON je ne l'achète pas.* Le sinon si s'emploie comme le *if* :

PROGRAMMATION D'UN ARDUINO

Les conditions

Fonctionnement. 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
}
```

PROGRAMMATION D'UN ARDUINO

Les conditions

Fonctionnement. Else if

A retenir donc, si la première condition est fausse, on teste la deuxième, si la deuxième est fausse, on teste la troisième, etc. « Schéma » de principe du *else*, idem au *if* :

```
else if(/* test de la condition */)  //si elle est vraie...
{
   //...on exécute les instructions entre ces accolades
}
```

Le « else if » ne peut pas être utilisée toute seule, il faut obligatoirement qu'il y ait un « if » avant !

PROGRAMMATION D'UN ARDUINO

Les opérateurs logiques

Et si je vous posais un autre problème ? Comment faire pour savoir si la voiture est inférieure à 500000 DA ET si elle est grise ?
C'est vrai ça, si je veux que la voiture soit grise en plus d'être inférieure à 500000DA, comment je fais ?
Il existe des opérateurs qui vont nous permettre de tester cette condition ! Voyons quels sont ses opérateurs puis testons-les !

Opérateur	Signification
&&	… ET …
||	… OU …
!	NON

PROGRAMMATION D'UN ARDUINO

Les opérateurs logiques

Opérateur logique ET

Reprenons ce que nous avons testé dans le *else if* : *SI la voiture vaut 550000DA ET qu'elle a l'option GPS en plus, ALORS je l'achète.* On va utiliser un *if* et un opérateur logique qui sera le *ET* :

```
int prix_voiture = 550000;
int option_GPS = TRUE;

/* l'opérateur && lie les deux conditions qui doivent être
vraies ensemble pour que la condition soit remplie */
if(prix_voiture == 550000 && option_GPS)
{
   //j'achète la voiture si la condition précédente est vraie
}
```

PROGRAMMATION D'UN ARDUINO

Les opérateurs logiques

Opérateur logique OU

On peut reprendre la condition précédente et la première en les assemblant pour rendre le code beaucoup moins long.

```
int prix_voiture = 550000;
int option_GPS = TRUE;
if(prix_voiture < 500000)
{
   //la condition est vraie, donc j'achète la voiture
}
else if(prix_voiture == 550000 && option_GPS)
{
   //la condition est vraie, donc j'achète la voiture
}
else
{
   //la condition est fausse, donc je n'achète pas la voiture
}
```

PROGRAMMATION D'UN ARDUINO

Les opérateurs logiques + condition

Vous voyez bien que l'instruction dans le *if* et le *else if* est la même. Avec un opérateur logique, qui est le OU, on peut rassembler ces conditions :

```
int prix_voiture = 550000;
int option_GPS = TRUE;
if((prix_voiture < 500000) || (prix_voiture == 550000 && option_GPS))
{
   //la condition est vraie, donc j'achète la voiture
}
else
{
   //la condition est fausse, donc je n'achète pas la voiture
}
```

Lisons la condition testée dans le if : « SI le prix de la voiture est inférieur à 500000DA OU SI le prix de la voiture est égal à 5500€ ET la voiture à l'option GPS en plus, ALORS j'achète la voiture ».

Attention aux parenthèses qui sont à bien placer dans les conditions, ici elles n'étaient pas nécessaires, mais elles aident à mieux lire le code.

PROGRAMMATION D'UN ARDUINO

Switch

Il existe un dernier test conditionnel que nous n'avons pas encore abordé, c'est le *switch*. Voilà un exemple :

```
int options_voiture = 0;
if(options_voiture == 0)
{
   //il n'y a pas d'options dans la voiture
}
if(options_voiture == 1)
{
   //la voiture a l'option GPS
}
if(options_voiture == 2)
{
   //la voiture a l'option climatisation
}
if(options_voiture == 3)
{
```

PROGRAMMATION D'UN ARDUINO

Switch

```
//la voiture a l'option vitre automatique
}
if(options_voiture == 4)
{
   //la voiture a l'option barres de toit
}
if(options_voiture == 5)
{
   //la voiture a l'option  siège éjectable
}
else
{
//retente ta chance ;-)
}
```

PROGRAMMATION D'UN ARDUINO

Switch

Le code précédent est trop farfelue et non intéressant.
La solution pour le remplacer et le rendre plus simple, existe, c'est le *switch*.

Le *switch*, comme son nom l'indique, va tester la variable jusqu'à la fin des valeurs qu'on lui aura données. Voici comment cela se présente :

```
int options_voiture = 0;
switch (options_voiture)
{
   case 0:
      //il n'y a pas d'options dans la voiture
      break;
   case 1:
      //la voiture a l'option GPS
      break;
   case 2:
      //la voiture a l'option climatisation
      break;
```

PROGRAMMATION D'UN ARDUINO

Switch

```
case 3:
      //la voiture a l'option vitre automatique
      break;
   case 4:
      //la voiture a l'option barres de toit
      break;
   case 5:
      //la voiture a l'option siège éjectable
      break;
   default:
      //retente ta chance ;-)
      break;
}
```