Arduino

```
char num = 66;

unsigned int num = 65535;

char nom[9] = {'P', 'o', 'l', 'y', 't', 'e', 'c', 'h','\0'};

if (condition) {   instructions; }
else if (condition) {   instructions; }
else {   instructions; }
```

Pascal MASSON

(pascal.masson@unice.fr)

Version projection
Edition 2019-2020-V11

Variables, constantes, fonctions et autres

Sommaire

Introduction

- L'utilisation des cartes arduino repose sur la compréhension de l'électronique mais aussi sur la programmation en C
- Ce document regroupe une partie du langage C tel que les conditions les fonctions, les boucles, les types de variables …
- Des exemples illustrent toutes les notions

1. Les types de variables/constantes

1.1. int

Définition

- Un entier signé « int » est codé sur 2 octets donc 16 bits et prend ainsi 2^{16} valeurs différentes dans l'intervalle [–32768 ; 32767]
- L'intervalle n'est pas symétrique puisqu'il faut prendre en compte le 0

Exemple

- On définit une variable appelée « num » qui a pour valeur initiale 35565

```
int num = 32767;
```

- C'est une numérotation qui tourne en rond ce qui implique que
  - ✓ 32767 + 1 donne –32768
  - ✓ –32768 – 1 donne 32767

1. Les types de variables/constantes

1.2. unsigned int

- **Définition**
  - Un entier non signé « unsigned int » est codé sur 2 octets donc 16 bits et prend ainsi 2^{16} valeurs différentes dans l'intervalle [0; 65535]
- **Exemple**
  - On définit une variable appelée « num » qui a pour valeur initiale 35565

```
unsigned int num = 65535;
```

  - C'est une numérotation qui tourne en rond ce qui implique que
    - ✓ 65535 + 1 donne 0
    - ✓ 0 – 1 donne 65535

1. Les types de variables/constantes

1.3. long

Définition

- Un entier « long » est codé sur 4 octets donc 32 bits et prend ainsi 2^{32} valeurs différentes dans l'intervalle [–2147483648 ; 2147483647]

Exemple

- On définit une variable appelée « num » qui a pour valeur initiale 2147483647

```
long num = 2147483647;
```

- C'est une numérotation qui tourne en rond ce qui implique que
  - ✓ 2147483647 + 1 donne –2147483648
  - ✓ –2147483648 – 1 donne 2147483647

1.4. unsigned long

- **Définition**
  - Un entier long non signé « unsigned long » est codé sur 4 octets donc 32 bits et prend ainsi 2^{32} valeurs différentes dans l'intervalle [0; 4294967296]
- **Exemple**
  - On définit une variable appelée « num » qui a pour valeur initiale 42147483647

```
unsigned long num = 4294967296;
```

  - C'est une numérotation qui tourne en rond ce qui implique que
    - ✓ 4294967296 + 1 donne 0
    - ✓ 0 – 1 donne 4294967296

1. Les types de variables/constantes

1.5. char

Définition

- Un « char » (caractère) est un entier signé et codé sur 1 octet donc 8 bits qui prend 2^8 valeurs différentes dans l'intervalle [–128; 127]
- C'est un type qui est utilisé pour coder les caractères, comme l'alphabet, regroupés dans un tableau appelé ASCII (American Standard Code for Information Interchange)
- Par exemple le chiffre 65 correspond à « A ».
- Il est possible de faire des opérations sur les « char » avec par exemple 65 + 1 = 66 qui correspond à la lettre « B »

1. Les types de variables/constantes

1.5. char

Code ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

1.5. char

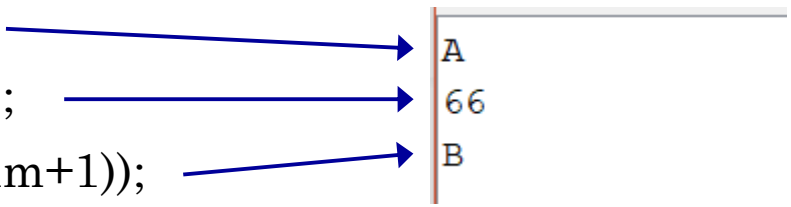
Exemple

- On définit une variable char « num » qui a pour valeur initiale 66 :

```
char num = 66;
```

- Avec le moniteur série on affiche ceci :

```
Serial.println(num);
Serial.println(num+1);
Serial.println(char(num+1));
```



- On remarque que pour l'impression, il est préférable de demander l'affichage d'un caractère
- On peux aussi définir un caractère ASCII :

```
char num = 'A';
```

- Il ne faut pas confondre 'A' (guillemets simples) avec "A" (guillemets) qui est un string

1.5. char

- **Exemple**
  - C'est une numérotation qui tourne en rond ce qui implique que
    - ✓ 127 + 1 donne –128
    - ✓ –128 – 1 donne 127

1.6. unsigned char

- **Définition**
  - Un « char » non signé est un entier signé et codé sur 1 octet donc 8 bits qui prend 2^8 valeurs différentes dans l'intervalle [0; 255]
- **Exemple**
  - On définit une variable char non signée « num » qui a pour valeur initiale 230 :

```
unsigned char num = 230;
```

  - C'est une numérotation qui tourne en rond ce qui implique que
    - ✓ 255 + 1 donne 0
    - ✓ 0 – 1 donne 255

1. Les types de variables/constantes

1.7. byte

Définition

- Un « byte » est un synonyme de « unsigned char » donc c'est un entier non signé et codé sur 1 octet donc 8 bits qui prend 2^8 valeurs différentes dans l'intervalle [0; 255]

Exemple

- On définit une variable byte « num » qui a pour valeur initiale 230 :

```
byte num = 230;
```

- C'est une numérotation qui tourne en rond ce qui implique que
  - ✓ 255 + 1 donne 0
  - ✓ 0 – 1 donne 255

1.8. int8_t, uint8_t, int16_t, uint16_t, int32_t et uint32_t

- **Définition**
  - Ces types correspondent à char, unsigned char, byte, int, unsigned int, long et unsigned long
- **Exemple**
  - On définit une variable byte « num » qui a pour valeur initiale 230 :

```
uint8_t num = 255;
```

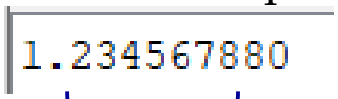
1.9. float

- **Définition**
  - Un « float » est un nombre réel (à virgule et signé) codé sur 4 octets donc 32 bits
  - Sa précision est au maximum de 7 décimales
- **Exemple**
  - On définit une variable char « num » qui a pour valeur initiale 1.23456789

```
float num = 1.23456789
```

  - L'impression sur le moniteur série (en demandant 9 chiffres après la virgule) montre qu'on obtient pas exactement le nombre définit au départ

```
Serial.println(num,9);
```



7 chiffres corrects après la virgule

2 chiffres ajoutés pour arriver aux 9 demandés avec Serial.println

1. Les types de variables/constantes

1.10. array

- **Définition**
  - Les « arrays » ou tableaux servent à stoker et ordonner des valeurs
  - On les utilise aussi pour faire des actions répétitives
- **Déclaration**
  - Soit une variable que l'on appelle « tableau »
    - ✓ X tableau[] avec X pour le type de variable et un nombre de lignes inconnu
    - ✓ X tableau[Y] avec X pour le type de variable et Y pour le nombre de lignes en commençant par 0
    - ✓ X tableau[Y][Z] avec X pour le type de variable, Y pour le nombre de lignes et Z pour le nombre de colonne en commençant par 0

1.10. array

Exemple 1

- Un exemple courant est la définition de plusieurs IO comme OUTPUT par exemple pour commander un driver de moteurs
- On peut écrire dans le setup :

```
pinMode(2, OUTPUT);          digitalWrite(2, LOW);
pinMode(3, OUTPUT);          digitalWrite(3, LOW);
pinMode(4, OUTPUT);          digitalWrite(4, LOW);
pinMode(5, OUTPUT);          digitalWrite(5, LOW);
pinMode(6, OUTPUT);          digitalWrite(6, LOW);
pinMode(7, OUTPUT);          digitalWrite(7, LOW);
```

1.10. array

- **Exemple 1**
  - On peut plus simplement définir un tableau de constantes :

```
const byte IO_Moteurs[6] = {2, 3, 4, 5, 6 ,7};
```

  - et utiliser une boucle « for » pour configurer les IO et les mettre à l'état bas

```
for (int i = 0; i < 7; i++) {
  pinMode(IO_Moteurs[i], OUTPUT);
  digitalWrite(IO_Moteurs [i], LOW);}
```

  - « i » correspond à l'index du tableau et donc à la ligne dans ce cas
  - Dans cet exemple, on passe de 12 à 4 lignes

1.10. array

- **Exemple 2**
  - On peut aussi modifier la valeur des éléments du tableau
  - Soit le tableau d'entiers suivant :

    ```
    int tableau [3] = {100, 46, 700, 1023};
    ```

  - Si on fait l'opération suivante :

    ```
    tableau [0] = 14;
    ```

  - on obtient pour le tableau :

100	→	14
46		46
700		700
1023		1023

1.11. Chaine de caractères (string, « s » minuscule)

Définition

- Une chaine de caractères ou « string » en Anglais est représentée sous forme de tableau de variables de type char et se termine par le caractère « nul ». On remarquera que « string » s'écrit avec un « s » minuscule pour ne pas confondre avec « String » qui est une variable de type Objet
- Le caractère « nul » correspond au code ASCII 0 qui est différent du '0' qui correspond au le code ASCII 48 :

Dec	Hx	Oct	Char
0	0	000	NUL (null)
48	30	060	0

- Le caractère « nul » permet aux fonctions comme Serial.print() de savoir où se termine la chaîne de caractères. Sans cela, les fonctions utiliseront les octets qui suivent dans la mémoire
- L'omission de ce caractère peut expliquer le mauvais fonctionnement de certains programmes

1.11. Chaine de caractères (string, « s » minuscule)

- **Exemples**
  - Pour déclarer une chaine de caractère appelée « nom » à laquelle on affecte la valeur « Polytech », il faut écrire :

```
char nom[9] = {'P', 'o', 'l', 'y', 't', 'e', 'c', 'h','\0'};
```

  - On peut aussi omettre d'ajouter \0 et c'est le compilateur qui s'en chargera :

```
char nom[9] = {'P', 'o', 'l', 'y', 't', 'e', 'c', 'h'};
```

  - On peut écrire plus simplement le mot « Polytech » dans le programme :

```
char nom[9] = "Polytech";
```

  - Le compilateur peut se charger de dimensionner le tableau et d'ajouter \0 :

```
char nom[] = "Polytech";
```

1.11. Chaine de caractères (string, « s » minuscule)

- **Exemples**

  - On peut choisir une dimension plus grande pour le tableau afin de le compléter plus tard dans le programme :

```
char nom[20] = "Polytech";
```

  - Si texte est trop long, il peut être écrit sur plusieurs lignes :

```
char nom[] = "Je suis etudiant"
"en deuxieme annee du PeiP"
"de Polytech Nice";
```

1.11. Chaine de caractères (string, « s » minuscule)

- **Exemples**

  - Le tableau « nom » peut être constitué de plusieurs chaines de caractères. C'est un cas de tableau à 2 dimensions et il faut pointer vers l'adresse mémoire du début de chaque chaine. La variable « char » est alors agrémentée d'un Astérix « * » :

```
char* nom[] = {"Je suis etudiant", "en deuxieme annee du PeiP", "de Polytech Nice"};
```

  - La commande suivant envoie le texte « en deuxieme annee du PeiP » sur le moniteur série :

```
Serial.print(nom[1]);
```

1.12. String (« S » majuscule)

- **Définition**
  - L'objet String (avec un S majuscule) contient à la fois des chaines de caractères et des fonctions. Il est donc beaucoup plus complexe qu'un tableau de caractères et prends beaucoup plus de place en mémoire.
  - Il est possible de faire des opérations sur les Strings comme rechercher/remplacer un mot, connaitre leurs longueurs
- **Exemple**
  - Voici comment définir un String :

```
String texte = "Hello je suis en Peip a Polytech Nice";
```

1.12. String (« S » majuscule)

- **Exemple de fonctions**
  - La fonction « X.toUpperCase() » passe le texte X en majuscule :

    texte.toUpperCase();

    donne « HELLO JE SUIS EN PEIP A POLYTECH NICE »
  - La fonction « X.replace("Y", "Z") » remplace le texte Y par le texte Z

    texte.replace("Peip", "Peip2");

    donne « Hello je suis en Peip2 a Polytech Nice »
  - La fonction « X.lenght() » donne le nombre de caractères du texte X

    texte.lenght();

    donne 38

1.12. String (« S » majuscule)

- **Exemple de fonctions**

  - La fonction « X.string.toCharArray(buf, len) » transforme le String X en chaine de caractères. « buf » est le nom de la chaine de caractères et « len » la longueur du texte X

```
char texte_Char[texte.length()+1];
texte.toCharArray(texte_Char,texte.length()+1)
```

  - La fonction « X. toFloat() » transforme le String X en nombre flottant
  - La fonction « X. toInt() » transforme le String X en nombre entier
  - Pour trouver les autres fonctions, vous pouvez aller sur le site arduino : www.arduino.cc/reference/en/language/variables/data-types/stringobject/

1.12. String (« S » majuscule)

- **Tailles String et chaine de caractères**
  - On compare ici l'espace mémoire utilisé pour un texte définit comme un String ou comme une chaine de caractères (string)
  - La compilation d'un programme vide indique :

```
void setup() {}
void loop() {}
```

indique :

```
Le croquis utilise 444 octets (1%) de l'espace de stockage de programmes. Le maximum est de 32256 octets.
Les variables globales utilisent 9 octets (0%) de mémoire dynamique, ce qui laisse 2039 octets pour les variables locales. Le maximum est de 2048 octets.
```

1.12. String (« S » majuscule)

Tailles String et chaine de caractères

- On définit alors une chaine de caractères :

```
char texte[] = "Hello je suis en Peip a Polytech Nice";
void setup() {}
void loop() {}
```

- La compilation donne :

```
Le croquis utilise 444 octets (1%) de l'espace de stockage de programmes. Le maximum est de 32256 octets.
Les variables globales utilisent 9 octets (0%) de mémoire dynamique, ce qui laisse 2039 octets pour les variables locales. Le maximum est de 2048 octets.
```

- Il n'y a pas de différence avec un programme vide pour cet exemple

1.12. String (« S » majuscule)

Tailles String et chaine de caractères

- On définit le même texte mais comme un String :

```
String texte = "Hello je suis en Peip a Polytech Nice";
void setup() {}
void loop() {}
```

- La compilation donne :

```
Le croquis utilise 1846 octets (5%) de l'espace de stockage de programmes. Le maximum est de 32256 octets.
Les variables globales utilisent 63 octets (3%) de mémoire dynamique, ce qui laisse 1985 octets pour les variables locales. Le maximum est de 2048 octets.
```

- Cette fois la différence est très grande puisque le texte consomme 1402 octets de l'espace de stockage de programme et 54 octets de mémoire dynamique

1. Les types de variables/constantes

1.13. enum

Définition

- « enum » permet de définir des constantes qui seront utilisées pour leurs noms et non pour leurs valeurs
- Le com

```
enum {nom1, nom2, nom3 …}
```

1.14. Les constantes

Pourquoi des constantes ?

- Il est possible de déclarer par défaut toutes les constantes comme des variables (sous-chapitres précédents) et de ne pas y toucher dans le programme mais cela pose 2 problèmes :

  - ✓ On est pas à l'abris que finalement le programme modifie la valeur des constantes.
  - ✓ Une variable prend beaucoup plus de place en mémoire qu'une constante.

1.14. Les constantes

- **Déclaration des constantes ?**
  - Il suffit d'ajouter « const » devant le type :

```
const int val = 10;
```

  - On peut aussi les déclarer de cette façon sans « ; » à la fin de l'instruction :

```
#define val 10
```

  - Dans ce cas, le préprocesseur remplacera tous les « val » par 10 avant de donner la main au compilateur.

2. Les boucles

2.1. La boucle For

- **Forme générale**
  - Une boucle for sert à exécuter une série d'instructions un nombre définit de fois.
  - Ce nombre est contrôlé par un compteur que l'on incrémente ou décrémente

```
for (initialisation; condition; incrémentation) {
  instructions; }
```

- **Exemple**
  - On fait clignoter 5 fois la LED de l'I/O n°13
  - Le compteur est un entier noté « i »

```
for (int i=1, i<=10, i++){
  digitalWrite(13, !digitalRead(13));
  delay(500);}
```

2.2. La boucle while

- **Forme générale**
  - Une boucle while sert à exécuter une série d'instructions tant qu'une condition est vérifiée

```
while (condition) {
  instructions; }
```

- **Exemple**
  - On fait clignoter la LED de l'I/O n°13 tant que l'entrée n°7 détecte le niveau logique « 0 »

```
while (digitalRead(7)==LOW){
  digitalWrite(13, !digitalRead(13));
  delay(500);}
```

2.3. La boucle do while

Forme générale

- Une boucle do while sert à exécuter une série d'instructions une fois puis tant qu'une condition est vérifiée :

```
do {
   instructions;
} while (condition);
```

2.3. La boucle do while

- **Exemple**
  - On fait clignoter la LED de l'I/O n°13 rapidement tant que l'entrée n°7 détecte le niveau logique « 0 »
  - Sinon la LED emet un cout flash lumineux de 200 ms avec une période de 2.2 s

```
digitalWrite(13, LOW);
delay(2000);
do {
  digitalWrite(13, !digitalRead(13));
  delay(200);
} while (digitalRead(7)==LOW);
```

3. Les conditions

3.1. Les conditions if/else

Forme générale

- Les instructions contenues dans le « if » sont exécutées lorsqu'une (ou plusieurs) condition est vrai, si ce n'est pas le cas, ce sont les instructions contenues dans le « else » qui sont exécutées
- Il n'est pas nécessaire de mettre un « else » après un « if »

```
if (condition) {
   instructions; }
else {
   instructions; }
```

3.1. Les conditions if/else

- **Exemple**
  - On fait clignoter la LED de l'I/O n°13 plus ou moins rapidement en fonction du niveau logique détecté sur l'entrée n° 7

```
void loop() {
digitalWrite(13, !digitalRead(13));
delay(temps);
if (digitalRead(7)==LOW){
  temps=500;}
else {
  temps=100;}
}
```

3.2. Les conditions if/else if/else

- **Forme générale**
  - « else if » permet d'ajouter d'autres cas au couple « if/else »
  - Il n'est pas nécessaire de mettre un « else » après un « else if »

```
if (condition) {
   instructions; }
else if (condition) {
   instructions; }
else if (condition) {
   instructions; }
else {
   instructions; }
```

3.2. Les conditions if/else if/else

- **Exemple**

  - On fait clignoter la LED de l'I/O n°13 plus ou moins rapidement en fonction de la valeur de la variable « val » donnée par le moniteur série :

```
if (Serial.available()){
  val=Serial.parseInt(); }
 if (val <3){  temps=100;}
 else if ((val>=3) && (val<5)){ temps=300;}
 else if ((val>=5<) && (val<8)){ temps=500;}
 else { temps=1000;}
digitalWrite(13, !digitalRead(13));
delay(temps);
```

3.3. Les conditions switch case

Forme générale

- Une alternative au trio « if / else if / else est d'utiliser l'instruction conditionnelle à choix multiples « switch case »
- Cela permet de tester des valeurs particulières (de types int et char) pour la variable qui est testée.
- « break » permet de sortie du « switch »
- « default » permet de prendre en compte les cas qui ne sont pas explicités dans les différents « case »

```
switch (variable) {
  case valeur1 :
     instructions;
     break;
  case valeur2 :
     instructions;
     break;
  default valeur3 :
    instructions; }
    break; }
```

3.3. Les conditions switch case

- **Exemple**

  - On fait clignoter la LED de l'I/O n°13 plus ou moins rapidement en fonction de la valeur de la variable « val » donnée par le moniteur série :

```
if (Serial.available()){
  val=Serial.parseInt(); }
 switch (val) {
   case 1 :
      temps=100;
      break;
   case 5:
      temps=300;
      break;
   case 10:
      temps=500;
      break;
   default :
      temps=1000;
      break;}
digitalWrite(13, !digitalRead(13));
delay(temps);
```

4. Les fonctions

4.1. Pourquoi créer des fonctions

- L'utilisation de fonctions permets d'alléger l'écriture et la lecture d'un programme
- On peut aussi plus simplement répéter une tache tout au long du programme

4.2. Fonction sans entrée ni sortie

Forme générale

- Elle est très simple et similaire à « void setup » et « void loop » :

```
void NOM(){
  instructions;}
```

- A noter que « void » signifie « vide » donc que la fonction ne renvoie rien.

Exemple

- La fonction « led » est appelée à chaque fois qu'il faut allumer ou éteindre la led

```
void loop() {
  led();
  delay(200);}
void led(){
  digitalWrite(13, !digitalRead(13));}
```

4.3. Fonction avec paramètres d'entrée

Forme générale

- Les paramètres sont passés entre les parenthèses et il peuvent être des constantes ou des variables qui seront donc modifiées dans la fonction

```
void NOM(param 1, param2 …){
  instructions;}
```

Exemple

- La fonction « led » est appelée à chaque fois qu'il faut allumer une des 3 leds

```
void led(byte num){
  if (num==1) {digitalWrite(LED1, LOW);}
  if (num==2) {digitalWrite(LED2, LOW);}
  if (num==3) {digitalWrite(LED3, LOW);}
}
```

4.4. Fonction avec paramètres d'entrée et de sortie

Forme générale

- En plus des paramètres, il faut indiquer le type de la valeur qui sera retourné par la fonction.

```
type NOM(param 1, param2 …){
  instructions;
  return resultat;}
```

Exemple

- On additionne 2 entiers :

```
int addition(int a, int b){
  int resultat ;
  resultat = a + b
  return resultat;}
```