Décrire le fonctionnement des systèmes

Mise en situation

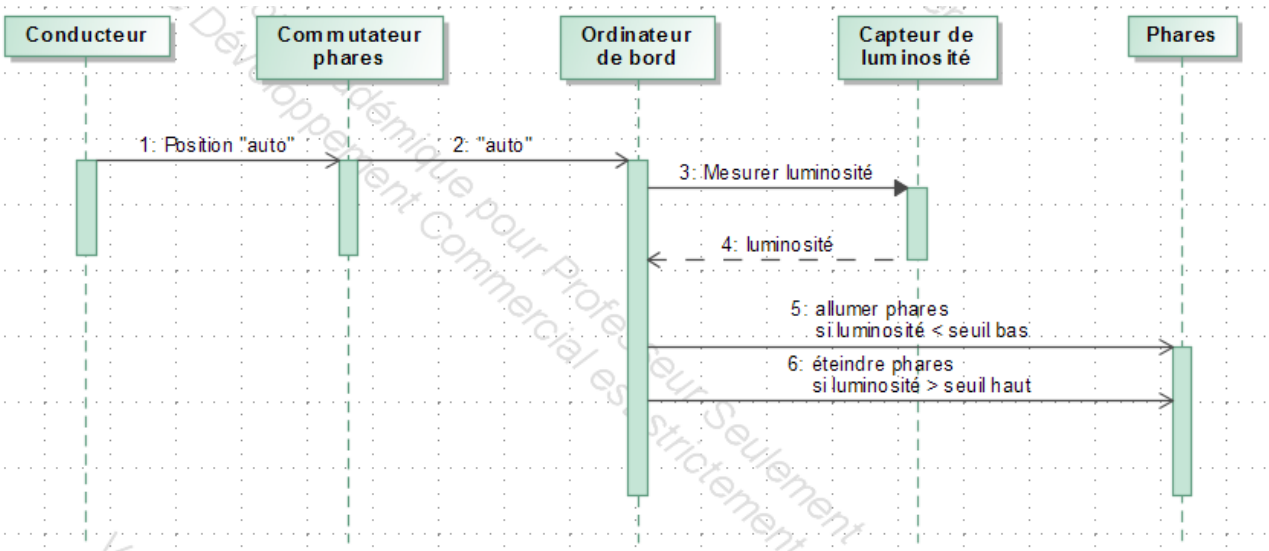
La plupart des voitures du marché sont dotées de contrôles divers et variés.
L'allumage automatique des phares est très répandu sur de nombreux véhicules.
On peut également citer le contrôle de la ceinture de sécurité, la détection de pluie et de nombreux autres automatismes.

Ce cours vous permettra de décrire selon divers formalismes le comportement des systèmes.

Diagramme de séquence (formalisme SysMl)

Il représente les échanges de messages entre les acteurs et le système ou entre des parties durant une séquence temporelle d'actions appelée « scénario » (il y a autant de diagrammes de séquence que de scénarios possibles).

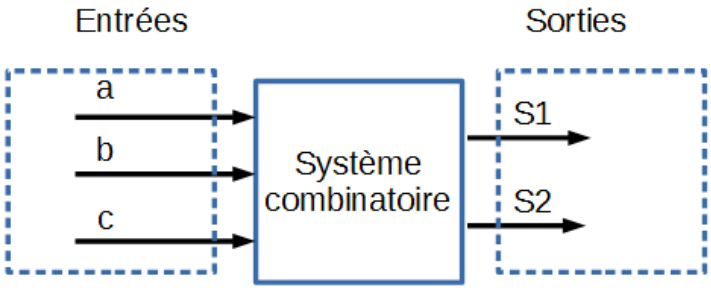
Voici ce que cela donne avec l'exemple de l'allumage automatique des phares...



Fonctionnent combinatoire

Remarque : l'application « soft cours logique » vous permet les manipulation ci-dessous.

Le fonctionnement est dit combinatoire dès lors qu'une action (S1 par exemple) est fonction d'une ou plusieurs combinaisons des variables d'entrées (a, b ,c).



Vidéo Définition d'un fonctionnement combinatoire

Un tel fonctionnement peut être décrit par :

- des équations logiques (encore appelées « équations Booléennes » ;
- des logigrammes (on raccorde des symboles logiques ensemble pour former un logigramme) ;
- des schémas électriques ;
- des algorigrammes ou algorithmes (Le chapitre suivante apportera plus d'information à ce sujet).

La fonction **OUI** :
Vidéo

Équation logique	Symbole logique	Table de Vérité	Schéma électrique	Algorithme
		a | S 0 | 1 |		**SI** a=1 **ALORS** S =1 **SINON** S = 0 **FIN SI**

La fonction **NON** :
Vidéo

Équation logique	Symbole logique	Table de Vérité	Schéma électrique	Algorithme
		a | S 0 | 1 |		**SI** a=1 **ALORS** S =0 **SINON** S = 1 **FIN SI**

La fonction **ET** :
Vidéo

Équation logique	Symbole logique	Table de Vérité	Schéma électrique	Algorithme
		a b S 0 0		**SI** a=1 **ET** b=1 **ALORS** S =1 **SINON** S = 0 **FIN SI**

La fonction **OU** :
Vidéo

Équation logique	Symbole logique	Table de Vérité	Schéma électrique	Algorithme
		a b S 0 0		**SI** a=1 **OU** b=1 **ALORS** S =1 **SINON** S = 0 **FIN SI**

La fonction **OU EXCLUSIF** :
Vidéo

Équation logique	Symbole logique	Table de Vérité	Schéma électrique	Algorithme
		a b S 0 0		**SI** (a=1 **ET** b=0) **OU** (a=0 **ET** b=1) **ALORS** S =1 **SINON** S = 0 **FIN SI**

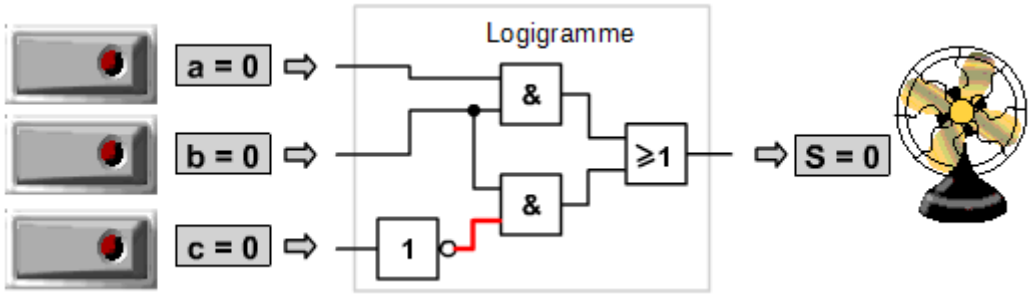
La fonction **NON ET (NAND)** :

Équation logique	Symbole logique	Table de Vérité	Schéma électrique	Algorithme
		a b S 0 0 		**SI** a=1 **ET** b=1 **ALORS** S =0 **SINON** S = 1 **FIN SI**

La fonction **NON OU (NOR)** :

Équation logique	Symbole logique	Table de Vérité	Schéma électrique	Algorithme
		a b S 0 0 		**SI** a=1 OU b=1 **ALORS** S =0 **SINON** S = 1 **FIN SI**

Vidéo exemple de fonctionnement décrit à partir de logigramme



Cliquer sur le schéma pour accéder à la vidéo

Description algorithmique

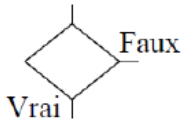
Un algorithme est un modèle universel de description d'un système numérique.
Il existe deux types de représentation :

- l'algorigramme (ou organigramme) ;
- l'algorithmique.

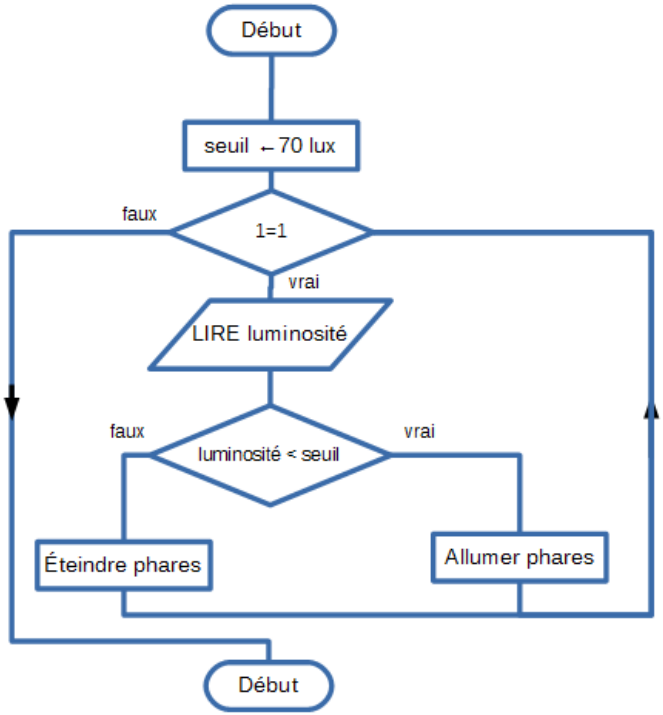
L'algorigramme

L'algorigramme est la représentation graphique d'une suite structurée d'instructions.

Voici les symboles graphiques normalisés :

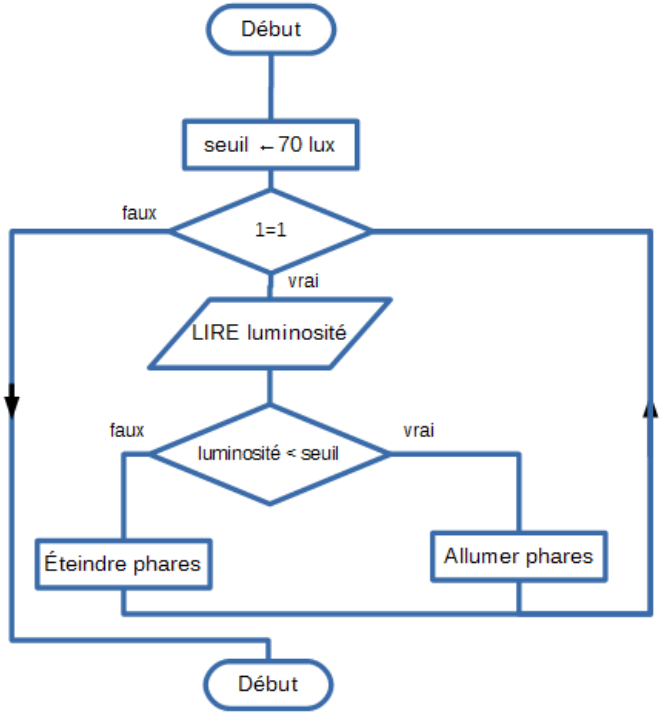
Symbole	Désignation	Symbole	Désignation
	**Début, fin, interruption** Début, fin ou interruption d'un programme		**Lecture ou Ecriture d'une donnée externe** Mise à disposition d'une information (écriture sur un port de sortie) ou enregistrement d'une information (lecture d'un port d'entrée).
	**Traitement interne** Opération ou calcul sur des données dont le résultat est stocké dans le microcontrôleur.		**Branchement** Test, exploitation de conditions variables impliquant le choix d'une parmi deux. Symbole utilisé pour représenter une décision.
	**Sous-programme** Portion de programme considérée comme une simple opération.		

Voici ce que cela donne avec l'exemple de l'allumage automatique des phares...avec une petite variante :
nota : il n'y a plus qu'un seuil ! Nous verrons plus en aval comment gérer le cas avec deux seuils.



Les algorigrammes ont l'avantage d'être rapides à lire mais ne sont pas adaptés pour des descriptions de fonctionnement réels. On préfère utiliser le langage algorithmique. Celui utilise une représentation littérale des symboles de l'algorigramme (Début, Fin, Lire, Ecrire, …).

Voici ce que cela donne avec l'algorigramme de l'allumage automatique des phares précédent :

Algorithme	Algorigramme
DÉBUT seuil ← 70 phare ← 0 **TANT QUE** 1 =1 **FAIRE \\création d'un boucle infinie** **LIRE** luminosite **SI** luminosite < seuil **ALORS** Allumer phares" **SINON** **ECRIRE** "Eteindre phares" **FIN SI** **FIN TANT QUE** **FIN**	

Les structures algorithmiques

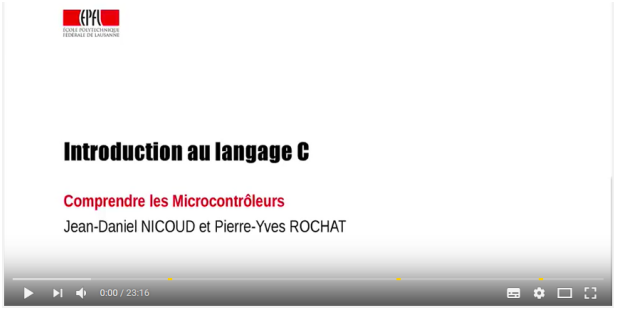
Plusieurs structures élémentaires sont à connaître :

- les boucles conditionnelles (Si … Alors … Sinon, Tant que … Faire, Faire … Tant que, Cas où) ;
- les boucles itératives (Pour).

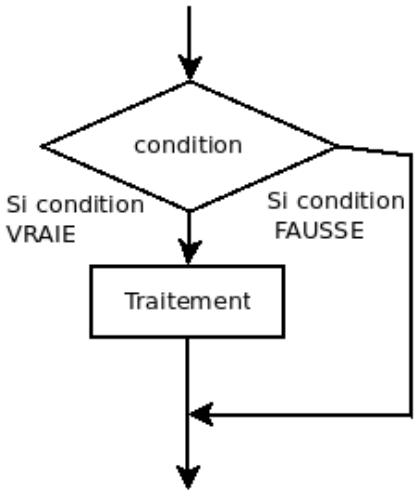
D'une manière générale, les projets sont préalablement décrits selon un algorigramme (dans les cas très simples) ou les algorithmes puis implémentés dans les microcontrôleurs à partir de langage de programmation.

Les langages de programmation les plus connus sont le C, le C++ (C plus plus), le C# (C sharp), le Java, le Python, etc),

Je vous invite à visionner la vidéo ci-contre avant de lire la suite.



Si … Alors (if …)



SI (Condition[1]) ALORS
    Traitement
FIN SI

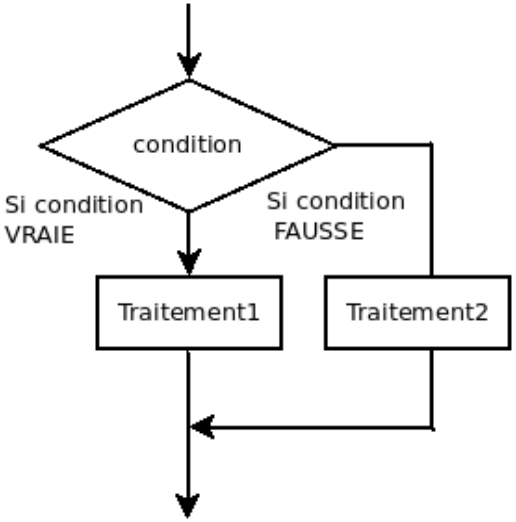
Langage Arduino :

```
if (Condition) {
    Traitement ;
}
```

ex : Allumer une led si le bouton poussoir est actionné.

```
if (etatBP==HIGH) {
    digitalWrite(led,HIGH) ;
}
```

Si … Alors … Sinon (if … else)

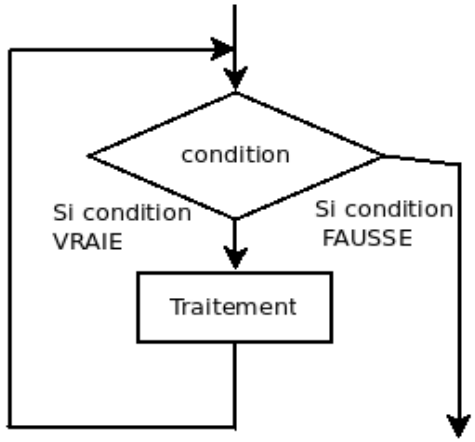


SI (Condition) ALORS
    Traitement 1
SINON
    Traitement 2
FIN SI

Langage Arduino :

```
if (Condition){
    Traitement 1 ;
}
else {
    Traitement 2 ;
}
```

Tant que… faire (While...)



TANT QUE (*Condition*) FAIRE
    Traitement
FIN TANT QUE

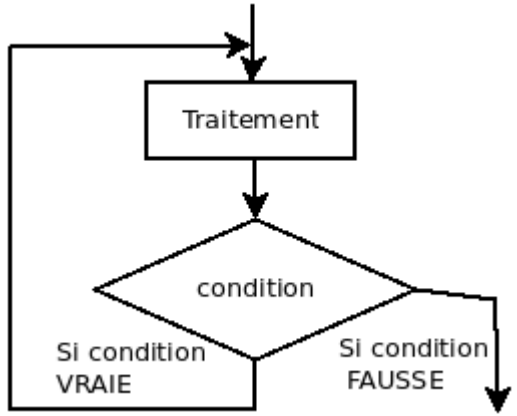
Langage Arduino :

```
while (Condition){
    Traitement ;
}
```

ex : attendre l'appui sur BP

```
while(etatBp!=HIGH){
    digitalWrite(led,HIGH) ;
}
```

Répéter …. Tant que (Do …. While) ou Faire …. Tant que



REPETER
    Traitement
TANT QUE (*Condition*)

Langage Arduino :

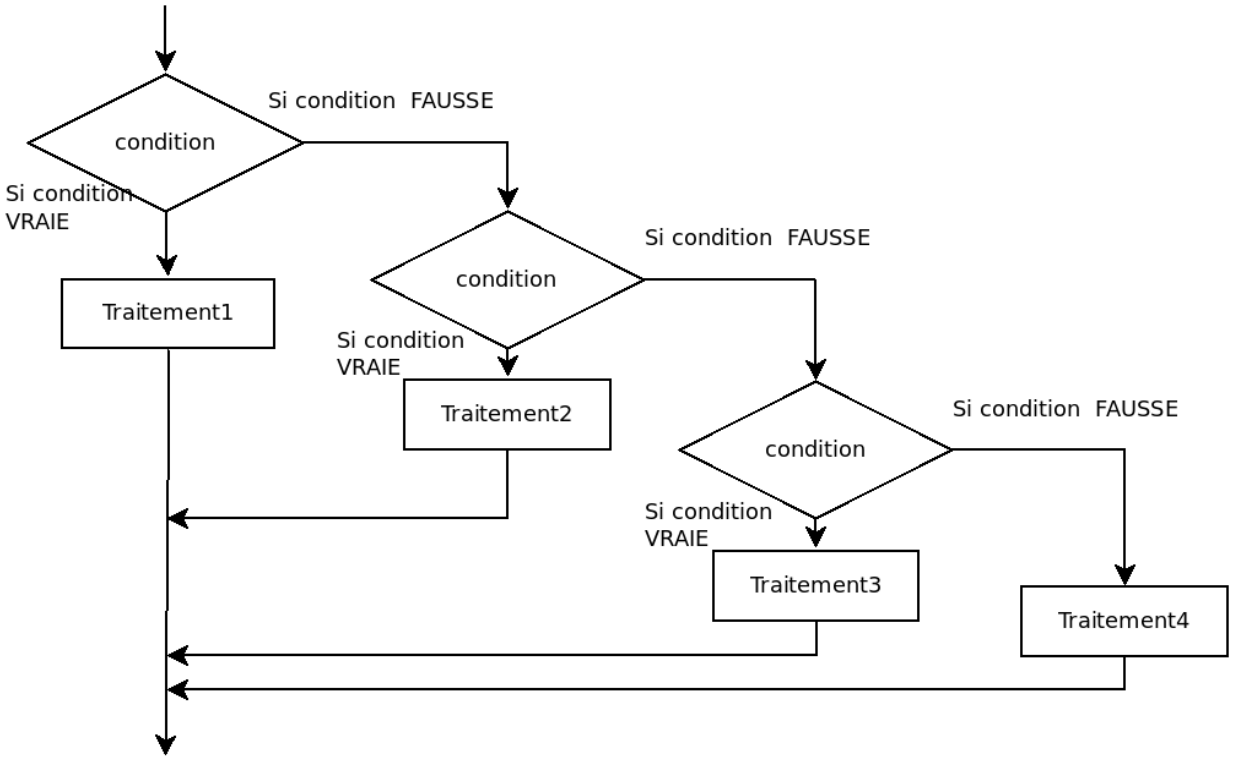
```
do {
    Traitement ;
} while (Condition)
```

ex : attendre l'appui sur BP

```
do {
    digitalWrite(led,HIGH) ;
} while (etatBp!=HIGH)
```

1 Exemple de condition à vérifer : luminosité > 70 lux, vitesse ≤ 1500tr/min, age = 18 ans, etc.

Si … Alors … Sinon Si …. Alors ...Sinon




```
SI (Condition1) ALORS
    Traitement 1
    SINON SI (Condition2) ALORS
        Traitement 2
        SINON SI (Condition3) ALORS
            Traitement 3
            SINON SI (Condition4) ALORS
                Traitement 4
            FIN SINON SI
        FIN SINON SI
    FIN SINON SI
FIN SI
```

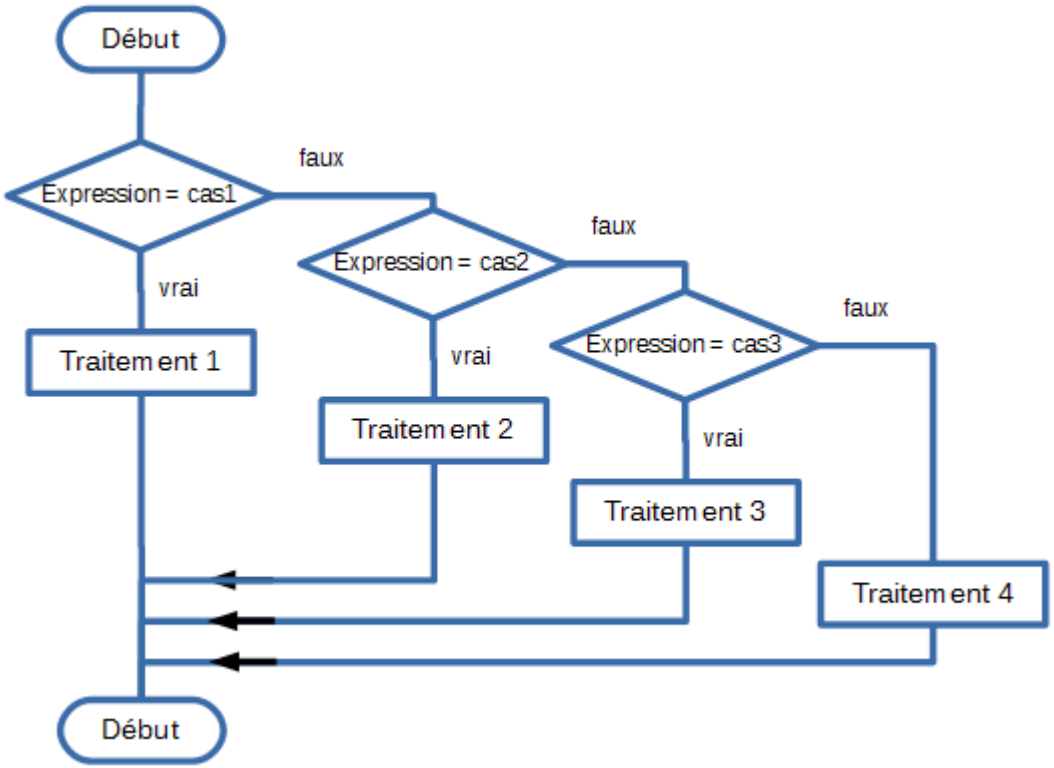
Langage Arduino :

```
if (Condition1){
    Traitement 1 ;
}
else if (Condition2){
    Traitement 2 ;
}
else if (Condition3 vérifiée){
    Traitement 3 ;
}
else if (Condition4){
    Traitement 4 ;
}
```

Il ne faut pas abuser de ce genre de structure ! L'algorithme ci-avant est déjà presque trop !!

On préfère utiliser dans ce cas la structure SELECTIONNER

Cas Où (Switch Case)



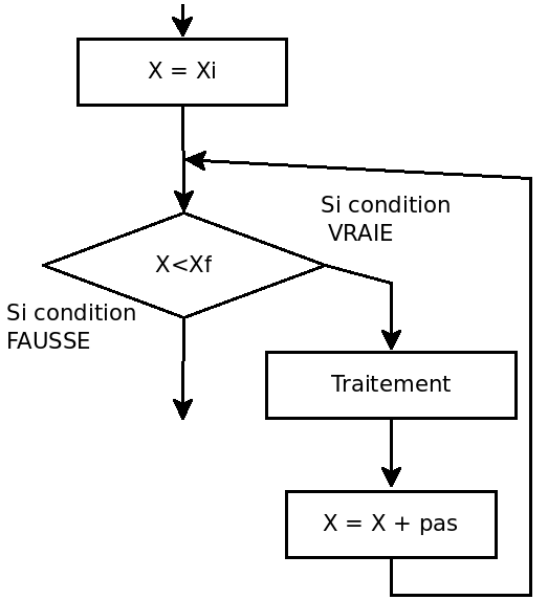

```
SELECTIONNER ( expression )
    Expression = cas1 : Traitement1
    Expression = cas2 : Traitement2
    Expression = cas3 : Traitement3
  SINON
    Traitement4
FIN SELECTIONNER
```

Langage Arduino :

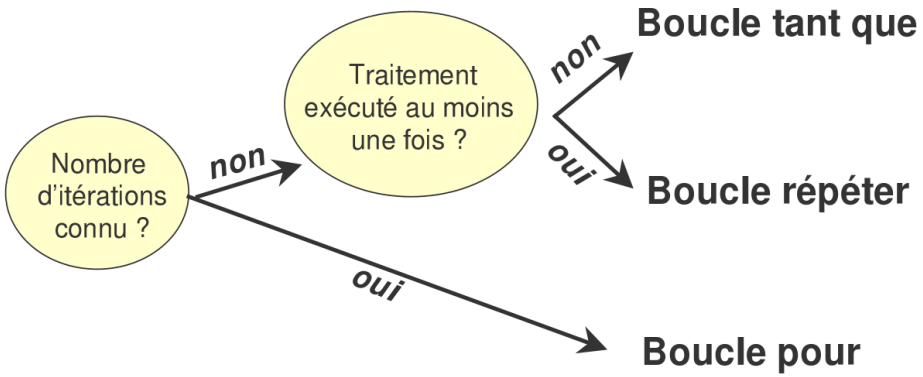
```
switch (var) {
  case 1 : Traitement1 ;
    break ;
  case 2 : Traitement2 ;
    break ;
  case 3 : Traitement3 ;
    break ;
  default :
      Traitement4
}
```

Exemple Langage Arduino :

```
enum {VERT, JAUNE, BLEU}
switch (var) {
  case VERT : Println(« la couleur est VERTE »);
    break ;
  case JAUNE : Println(« la couleur est JAUNE»);
    break ;
  case BLEU : Println(« la couleur est BLEUE»);
    break ;
  default :
      Traitement4
}
```

POUR (for)	
 Xi : valeur initiale Xf : valeur finale **pas** : pas d'incrémentation ou de décrémentation	POUR X = Xi JUSQU'A Xf Traitement FIN POUR *Langage Arduino (exemple de 10 itérations) :* *for (i=0 ; i<10 ; i=i+1) {* *Traitement ;* *}* *ex : faire clignoter une led 10 fois* *for (i=0 ; i<10 ; i = i + 1) {* *digitalWrite(ledRouge, HIGH) ; // allume la led rouge* *delay (500) ; // pause de 500 ms* *digitalWrite(ledRouge, LOW) ; // éteind la led rouge* *}*

Quelles structure choisir entre POUR (for), REPETER … TANT QUE (do … while) et TANT QUE … FAIRE (while)

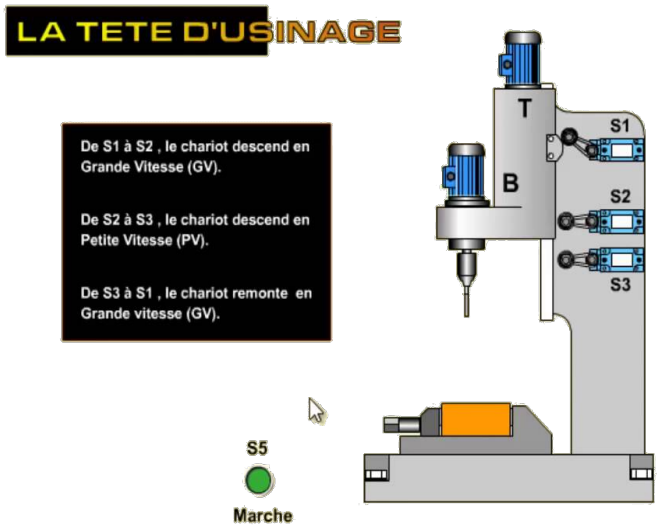


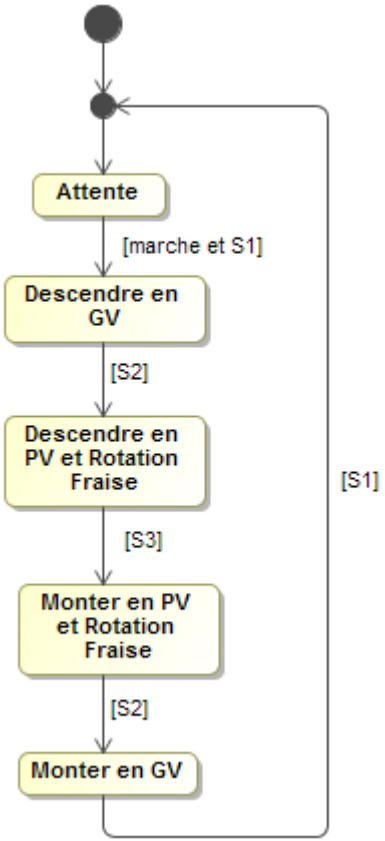
Accéder au travail dirigé « Décrire le fonctionnement des systèmes - TD1 »

Diagramme d'états

De nombreux système ont un fonctionnement séquentiel et ne peuvent être décrits simplement par les outils de description évoqués jusque présent.

Application à une fraiseuse automatique :





Pour accéder à la vidéo explicative, cliquer ici.

Application aux phares automatiques :

Utiliser un diagramme d'état est peu pertinent pour cet exemple car le fonctionnement des phares tel que décrit jusque présent est exclusivement combinatoire (i.e ne dépend que des conditions et pas des états antérieurs).

L'équation logique est :

Phares = a
avec a = Luminosité < seuil

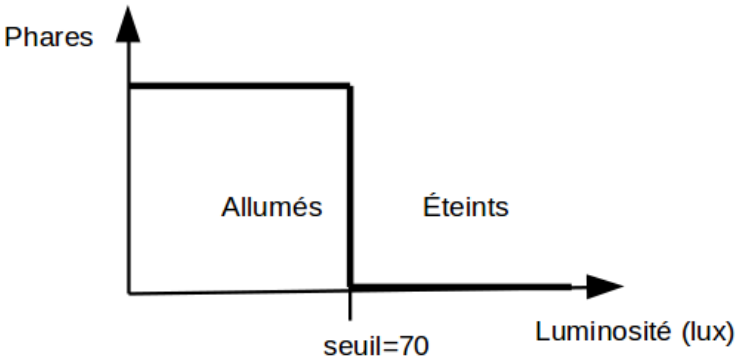
Une simple algorithme est suffisant…

SI Luminosité > seuil ALORS
    Allumer les phares
SINON Éteindre les phares
FIN SI

Cependant le fonctionnement des phares ainsi décrit n'est pas sans poser de problème que l'on implémente le programme selon l'algorithme précédent.

Page suivante, je vous propose de mettre en évidence la problématique.

Pour mieux comprendre, représentons le fonctionnement des phares à partir d'un graphe.

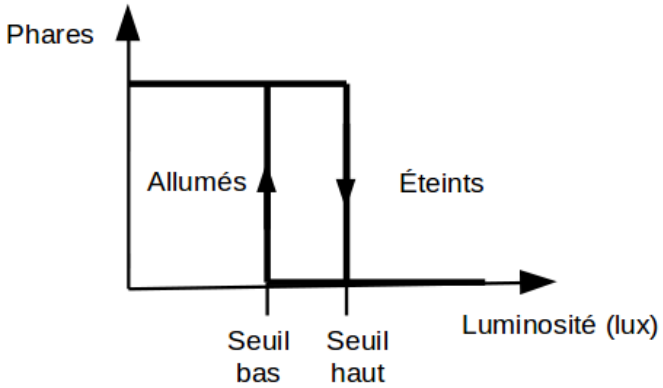


Le basculement de « phares allumés » vers « phares éteints » se faisant sur un seul seuil, que se passe t-il lorsque la luminosité est juste de 70 lux ?

Réponse :
Ils vont s'allumer et s'éteindre de manière anarchique en raison des variations légères autour de 70 Lux et des parasites captés sur le capteur de luminosité!! Pas cool !

Il faut donc trouver une autre solution pour gérer le fonctionnement des phares (la solution est celle évoquée dans le diagramme de séquence vu en début de ce cours en commutant sur deux seuils).

Le diagramme de fonctionnement satisfaisant au diagramme de séquence est le suivant :



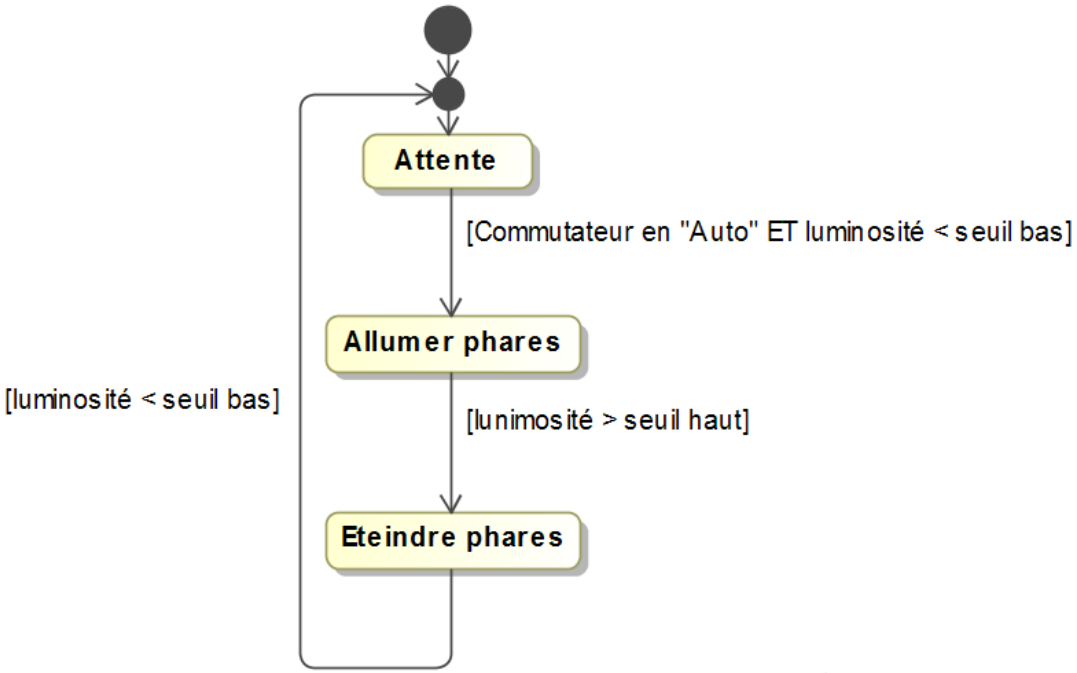
Explications :

Combinatoire !

- luminosité < seuil bas : les phares sont allumés ;
- luminosité > seuil haut : les phares sont éteints ;
- seuil bas < luminosité < seuil haut : ça dépend de l'état précédent !
  - Si les phares étaient allumés, alors ils restent allumés ;
  - Si les phares étaient éteints, alors ils restent éteints.

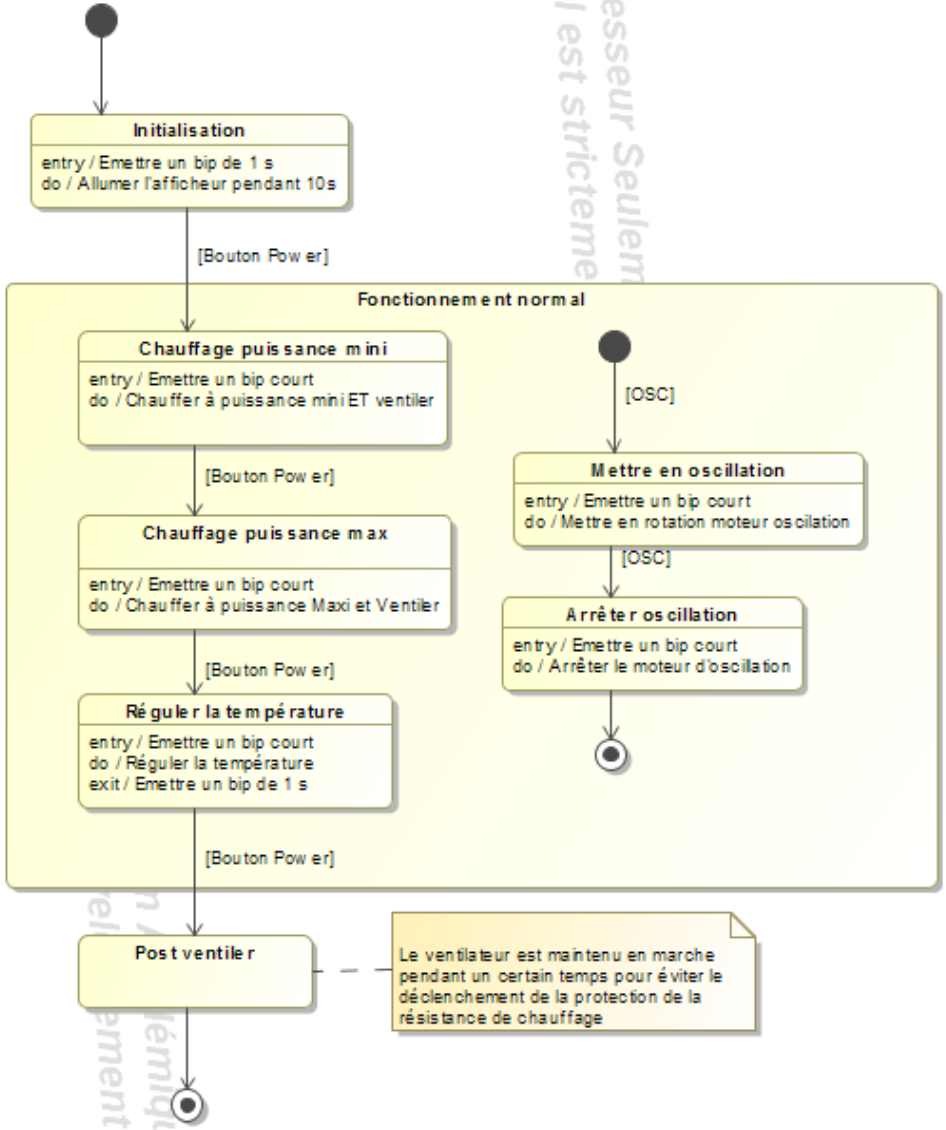
Séquentiel!

Dans ce cas précis (en raison du fonctionnement séquentiel) il est intéressant de représenter le fonctionnement par un diagramme d'états.



Pour ne pas alourdir la page, seul la gestion des modes principaux a été représentée dans le diagramme d’état.

Cliquer sur l'image pour accéder à la vidéo de fonctionnement

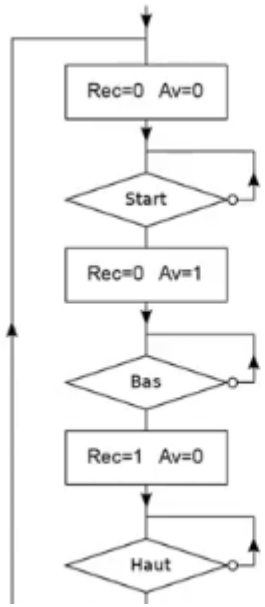


Traduire un fonctionnement séquentiel en langage algorithmique

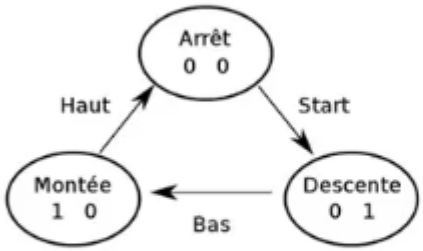
Le langage algorithmique est le préalable à toute programmation quel que soit le langage de programmation utilisé (C,C++, Java, Arduino, Python, etc).

Comment faire pour réaliser un algorithme à partir d'un diagramme d'états en vue d'une implémentation dans un langage de programmation.

Plusieurs méthodes existent.

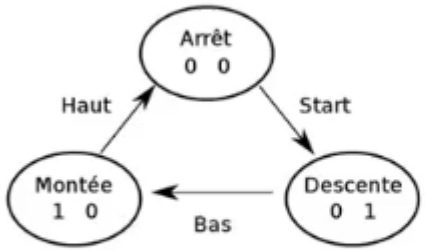



```
while (1) {
    Av = 0; Rec = 0;
    while  (Start) {
    }
    Av = 1; Rec = 0;
    while  (Bas) {
    }
    Av = 0; Rec = 1;
    while  (Haut) {
    }
}
```



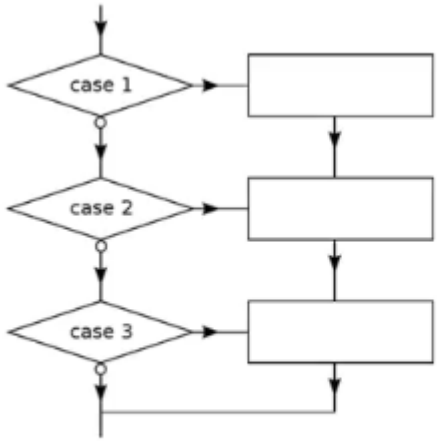
Graphe d'état

```
enum {arret, descente, montee};
...
int main() {
  ...
  int etat = arret;
  while (1) {
    if (etat == arret) ...

    if (etat == descente) ...

    if (etat == montee) ...

  }
}
```



Graphe d'état

```
enum {arret, descente, montee};
...
int main() {
  ...
  int etat = arret;
  while (1) {
   ...
    if (etat == descente) {
      Av(0) ; Rec(1) ;
      if (Bas()) {
        etat = montee ;
      } ...
  }
}
```




```
while (1) {
  switch (etat) {
    case arret : Av(0); Rec(0);
      if (Start()) { etat=descente; }
      break;
    case descente : Av(1); Rec(0);
      if (Bas()) { etat=montee; }
      break;
    case montee : Av(0); Rec(1);
      if (Haut()) { etat=arret; }
      break;
  }
```

`break` permet de sortir d'une structure

```
\\ Module principal
DÉBUT
   \\Déclaration et initialisation des variables
   etat="attente"
   seuilBas=65
   seuilHaut=75
   commutateur = "manu"
   luminosite =400

   \\On cherche à connaitre dans quel état est le commutateur auto/manu
   ECRIRE "Dans quel état est le commutateur 'auto' ou 'manu'?"
   LIRE commutateur

   SI commutateur ="auto" ALORS
      TANTQUE 1=1 FAIRE    \\Création d'un boucle infinie
         ECRIRE "saisisser la valeur de la luminosité' "
         LIRE luminosite
         SÉLECTIONNER etat
            "attente" : ECRIRE "Etat d'attente"
               SI luminosite < seuilBas ET commutateur ="auto" ALORS
                  etat = "phareOn"
               FINSI
            "phareOn" : ECRIRE "Phares allumés"
               SI luminosite>seuilHaut ALORS
                  etat = "phareOff"
               FINSI
            "phareOff" : ECRIRE "Phares éteints"
               SI luminosite < seuilBas ALORS
                  etat = "attente"
               FINSI
            SINON
               ECRIRE "Vous n'avez pas repecté les mots"
         FINSÉLECTIONNER
      FINTANTQUE
   SINON
      ECRIRE "Le fonctionnement est manuel"
   FINSI
FIN
```