

Introduction

- L'informatique est un outil puissant (cf cours du S1)
- L'informatique est omniprésent dans le domaine scientifique
- Apprenez à le maîtriser pour vous faciliter la vie et réussir vos études et vos projets

Info 2 PC

Objectifs de l'UE

- Vous permettre de :
 - manipuler des données structurées
 - lire et stocker les données dans des fichiers
 - comprendre comment l'informatique peut servir en physique et en chimie
 - maîtriser des outils informatiques pour résoudre des problèmes et ou réaliser des expérimentations scientifiques

Nos outils

- Le langage algorithmique
- Le langage C++
- Les ordinateurs des salles de TP
- Une plate-forme Arduino + une carte mémoire



- Quelques composants électroniques



Le projet de l'UE



- Réaliser une plate-forme de mesures de la température qui stockera les données dans un fichier d'une carte mémoire
- Analyser les données collectées et stockées dans le fichier de la carte mémoire
- Suivant l'avancement, on envisagera des manipulations sur le fichier, pour en créer un nouveau intégrant les aspects temporels. L'objectif serait de pouvoir le traiter sous Excel/Calc pour en faire des représentations graphiques

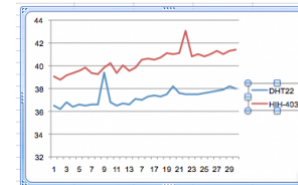
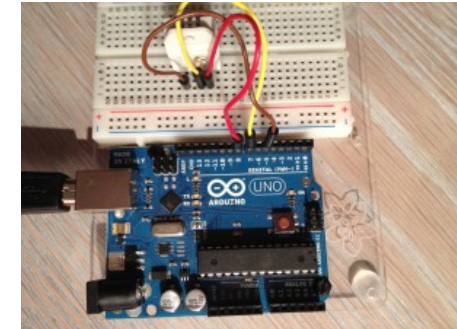
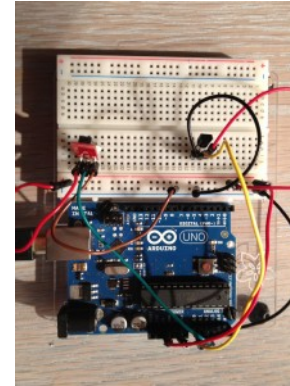
Organisation de l'UE

- Un seul enseignant : moi :-)
- 6 séances de cours
 - Présentation des nouveaux concepts algorithmiques (enregistrements, fichiers, tri) et d'Arduino
- 9 séances de TD
 - Manipulation des nouveaux concepts sur des exercices
 - Début semaine 3
- 4 séances de TP
 - Programmation et manipulation d'Arduino

Les TPs commenceront en semaine 4

MCC :
• 1 écrit !

En résumé on fera



Pour vous motiver

- Quelques idées de ce qu'il est possible de réaliser avec Arduino :
 - <http://hacknmod.com/hack/top-40-arduino-projects-of-the-web/>
 - <http://www.instructables.com/id/Arduino-Projects/>
 - <http://www.theinebriator.com/>
 - <http://blog.makezine.com/2012/05/09/wifi-rc-car-has-camera-and-force-feedback/>
- Et le must : c'est abordable !
 - Les personnes désireuses de faire des projets autour du sujet peuvent venir me voir

... si on (re)venait à l'algorithmique ?

Structure d'un algorithme

Algorithme `Nom_de_l_algorithme`

```
{  
    Bloc de déclarations  
    Séquence d'instructions  
}
```

Le bloc de « Déclarations » contient tous les objets utilisés par l'algorithme, ainsi que leur type (nature).

Tout objet utilisé doit avoir été déclaré.

De nouveaux types :

les types composés (enregistrements)

- Fréquemment, le concepteur d'algorithmes a besoin de types plus adaptés aux données qu'il veut manipuler.
- Par exemple, s'il veut faire des manipulations sur des informations concernant la carte grise d'un véhicule, idéalement, il souhaiterait disposer d'un **type** approprié pour faciliter l'accès aux différentes parties la constituant.
Prenons une carte grise « simplifiée » : elle contient le nom et prénom du propriétaire du véhicule, la marque et le modèle du véhicule, son immatriculation, son année de mise en circulation, sa puissance fiscale.

Rappels sur les types et les déclarations

- Types numériques : `entier` ou `réel`
- Types alphanumériques : `caractère` ou `chaîne de caractères`
- Type `logique` (booléen) : `VRAI` ou `FAUX`
- Déclaration :

```
variable Nom_var : type
```

exemple :

```
variable Nom_etudiant : chaîne de caractères  
    An_naissance, Annee_inscription : entier  
    Inscrit : logique
```

Les types composés (enregistrements)

- Malheureusement, un tel type n'existe pas de base dans le langage algorithmique (ni dans le langage C++). D'ailleurs on constate que les informations qu'il définit sont complexes car de natures différentes : certaines textuelles (nom, prénom, marque, modèle, immatriculation), certaines numériques (année de mise en service, puissance fiscale)
- Afin de permettre au développeur de manipuler ces données complexes, le langage algorithmique introduit la possibilité de définir des enregistrements.

Les types composés (enregistrements)

- Un enregistrement est une structure qui consiste en un nombre fixe de champs (éléments) de différents types.
- Le nom de la structure servira à nommer le nouveau type.
- Dans la structure, chaque champ est défini par :
 - un type (simple ou composé)
 - un identificateur

Définition et utilisation d'enregistrements

- Une fois, la structure d'un enregistrement (un type composé) ayant été définie, il est possible de déclarer des variables de ce type et de les utiliser. L'accès aux champs se fait au travers de l'opérateur « . » comme nous l'illustrons ci-dessous.

Algorithme Exemple_utilisation_enregistrement

```
{
  variable CG_voit1 : carte_grise
  CG_voit1.nom ← "DUPONT"
  CG_voit1.prenom ← "Louis"
  CG_voit1.marque ← "Renault"
  CG_voit1.modele ← "CLIO 2"
  CG_voit1.annee_mise_en_service ← 1910
  CG_voit1.puissance_fiscale ← 4
}
```

CG_voit1 est une variable dont le nom définit un enregistrement de type carte_grise

Les types composés (enregistrements)

- La syntaxe est :

```
structure nom_du_type_composé
{
  nom_du_champ1 : type1
  nom_du_champ2 : type2
  ...
  nom_du_champn : typen
}
```

- Exemple :

```
structure carte_grise
{
  nom : chaîne de caractères
  prenom : chaîne de caractères
  marque : chaîne de caractères
  modele : chaîne de caractères
  immatriculation : chaîne de caractères
  annee_mise_en_service : entier
  puissance_fiscale : entier
}
```

Définition et utilisation d'enregistrements

```
structure triangle
{
  cote1 : reel
  cote2 : reel
  cote3 : reel
}
```

Algorithme Testons_des_triangles

```
{
  variable t : triangle

  Afficher("Saisir les valeurs des 3 cotés du triangle")
  Saisir(t.cote1,t.cote2,t.cote3)
  Si ((t.cote1 = t.cote2) ET (t.cote2 = t.cote3)) alors
  {
    Afficher("Ce triangle est équilatéral.\n")
  }
  sinon
  {
    // faire ici les autres tests
  }
}
```

Définition et utilisation d'enregistrements

- On peut affecter un enregistrement à un autre de même type.

Algorithme Exemple1

```
{
    variable t1, t2 : triangle

    Afficher("Saisir les valeurs des 3 cotés du triangle")
    Saisir(t1.cote1,t1.cote2,t1.cote3)

    t2 ← t1
    Afficher("Les valeurs des cotés du premier triangle sont : ",
t1.cote1,t1.cote2,t1.cote3,"\n")
    Afficher("Les valeurs des cotés du second triangle sont : ",
t2.cote1,t2.cote2,t2.cote3,"\n")
    Afficher("Ces cotés sont identiques pour les deux triangles.\n")
}
```

Traduction en C++

- Définition d'un type composé :

```
struct nom_structure
{
    Définition des éléments...
};
```

- Accès à un élément de la structure :

```
variable.nom_élément
```

Définition et utilisation d'enregistrements

- Il est également possible de procéder à la copie champ par champ

Algorithme Exemple2

```
{
    variable t1, t2 : triangle

    Afficher("Saisir les valeurs des 3 cotés du triangle")
    Saisir(t1.cote1,t1.cote2,t1.cote3)

    t2.cote1 ← t1.cote1
    t2.cote2 ← t1.cote2
    t2.cote3 ← t1.cote3
    Afficher("Les valeurs des cotés du premier triangle sont : ",
t1.cote1,t1.cote2,t1.cote3,"\n")
    Afficher("Les valeurs des cotés du second triangle sont : ",
t2.cote1,t2.cote2,t2.cote3,"\n")
    Afficher("Ces cotés sont identiques pour les deux triangles.\n")
}
```

Initialisation en C++

- Exemples :

```
struct date
{
    int jour;
    int mois;
    int annee;
} d1 = {1,1,1996}; // d1 : variable de type structure

date d2 = {12,4,1996}; //d2 : variable de type structure

date d3;
d3.jour = 23;

cin >> d3.mois;
```

Exemple de tableau d'enregistrements

- Écrire un algorithme qui demande à l'utilisateur de saisir les coordonnées planaires de points définissant 10 segments et affiche les coordonnées points formant le plus grand des segments ainsi que sa taille.

```
structure Point
{
  X,Y : réel
}
structure Segment
{
  P1, P2 : Point
}
```

Exemple de tableau d'enregistrements

```
fonction longueurSegment(E S: Segment) : réel
{
  retourne  $\sqrt{(S.P2.X-S.P1.X)^2 + (S.P2.Y-S.P1.Y)^2}$ 
}
```

Et si on voyait comment utiliser ces nouveaux types pour des tableaux et avec des sous-programmes ?

Exemple de tableau d'enregistrements

```
fonction SaisiePoint(E i : entier) : Point
{
  variable P : Point
  Afficher("Saisissez les coordonnées du point ", i)
  Saisir(P.X,P.Y)
  retourne P
}
fonction SaisieSegment(E i : entier) : Segment
{
  variable S : Segment
  Afficher("Segment n°", i)
  S.P1 ← SaisiePoint(1)
  S.P2 ← SaisiePoint(2)
  retourne S
}
```

Exemple de tableau d'enregistrements

```

Algorithme Exo_Sur_Segments
{
    constante entier TAILLE ← 10
    variable v : tableau de TAILLE Segment
        i, iPlusLong : entier
        PlusLong : réel
    Afficher("Nous allons vous demander de saisir ", TAILLE, " segments.")
    Pour i de 0 à TAILLE-1
        v[i] ← SaisieSegment(i+1)
    iPlusLong ← 0
    PlusLong ← longueurSegment(v[iPlusLong])
    Pour i de 1 à TAILLE-1
        Si (PlusLong < longueurSegment(v[i])) alors
            {
                iPlusLong ← i
                PlusLong ← longueurSegment(v[iPlusLong])
            }
    Afficher("Le segment [(,v[iPlusLong].P1.X,",",v[iPlusLong].P1.Y,"),
    (,v[iPlusLong].P2.X,",",v[iPlusLong].P2.Y,")] est de tous les segments saisis celui
    ayant la plus grande longueur. Elle est de ", PlusLong,".\n")
}

```

Exemple de vecteur d'enregistrements en C++

```

#include <iostream>

using namespace std;

struct athlete
{
    int num;
    float rec;
};

int main()
{
    const int nb_part=15;
    athlete tmp,part[nb_part]; // part est un tableau du type athlete
    int i,j,indicemin,rang;
    float min;

    // saisie du tableau
    for (i = 0; i < nb_part; i++)
    {
        cout << "N° et record ?" << endl;
        cin >> part[i].num >> part[i].rec;
    }
}

```

Exemple de tableau d'enregistrements en C++

- Saisir le record et le N° des athlètes en utilisant un tableau d'enregistrements, puis le trier dans l'ordre croissant des records.
- Afficher le N°, le record et le rang de chaque athlète en donnant le même rang aux ex aequos.

Exemple de vecteur d'enregistrements en C++

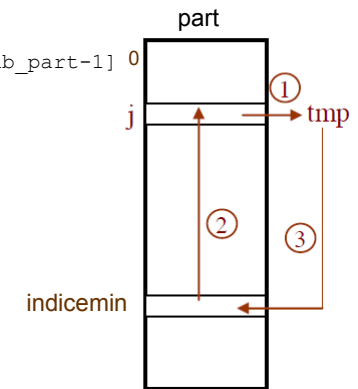
```

// tri du vecteur (tri « rapide »)
for (j = 0; j < nb_part-1; j++)
{
    // on cherche le minimum dans la partie [j,nb_part-1]
    min = part[j].rec;
    indicemin = j;

    for (i = j+1; i < nb_part; i++)
        if (part[i].rec < min)
        {
            min = part[i].rec;
            indicemin = i;
        }

    // on échange si nécessaire
    if (indicemin != j)
    {
        tmp = part[j];
        part[j] = part[indicemin];
        part[indicemin] = tmp;
    }
}

```



Exemple de vecteur d'enregistrements en C++

```
// affichage
cout << "rang participant record" << endl;
cout << "1 " << part[0].num << " " << part[0].rec << endl;

rang = 1;
for (i = 1; i < nb_part; i++)
{
    if (part[i].rec>part[i-1].rec)
        rang = i+1; //pour les records égaux

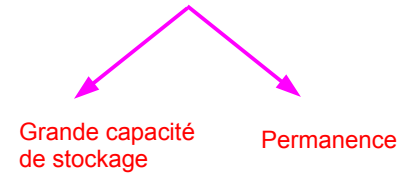
    cout << rang << " " << part[i].num << " " << part[i].rec << endl;
}
}
```


Les fichiers

Et si on apprenait à utiliser
un nouveau type d'entrée sortie

LES FICHIERS

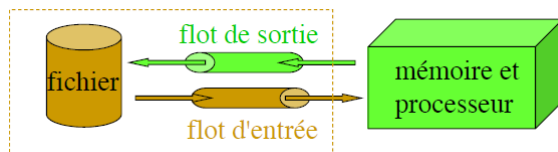
- Il est difficile de trouver UNE notation algorithmique pour les fichiers car ils représentent un concept de programmation hétérogène et multiforme.
- **Définition** : Un fichier est **une séquence d'enregistrements (fiches)** du même type (entier, réel, caractère, ...) rangée sur une **mémoire secondaire** (disque, disquette, ...).



Fichier physique

00100110 | 10100101 | 111101.....

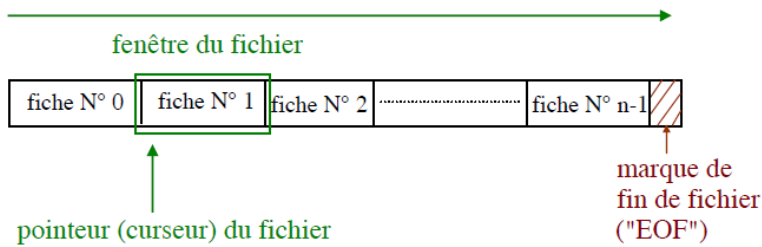
- Un accès au fichier correspond à **un nombre entier d'octets**.
- Les fichiers peuvent être considérés comme des canaux d'entrées et de sorties utilisant des **flots** de données :
 - réception de données dans le cas d'un flot de sortie ;
 - émission de données dans le cas d'un flot d'entrée.



Accès à un fichier

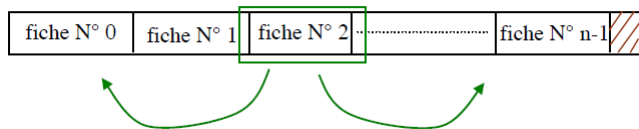
- Il existe 2 types accès à un fichier :
 - accès séquentiel ;
 - accès direct (aléatoire).
- Cette typologie ne se reflète pas dans la structure elle-même du fichier.
 - En fait, tout fichier peut être utilisé avec l'un ou l'autre des deux types d'accès.
 - Le choix du type d'accès n'est pas un choix qui concerne le fichier lui-même, mais uniquement la manière dont il va être traité par la machine. C'est donc dans l'algorithmique/programme, et seulement dans ce dernier, que l'on choisit le type d'accès souhaité.

Accès séquentiel



- On ne peut accéder qu'à la donnée suivant celle qu'on vient de lire.
- On ne peut donc accéder à une information qu'en ayant au préalable examinée celle qui la précède.
- Dans le cas d'un fichier texte, cela signifie qu'on lit le fichier ligne par ligne (enregistrement par enregistrement).

Accès direct



- On peut accéder directement à l'enregistrement de son choix, en précisant le numéro de cet enregistrement.
 - Mais cela veut souvent dire une gestion fastidieuse des déplacements dans le fichier.
- Opérations sur un fichier à accès direct :
 - Les mêmes opérations sur un fichier à accès séquentiel ;
 - Utilisé principalement pour la mise à jour des fichiers.

Accès séquentiel : opérations

- Six opérations possibles :
 - création ou désignation ;
 - ouverture ;
 - lecture ;
 - écriture ;
 - fermeture ;
 - test de fin de fichier.

Manipulation de fichiers

- Définition du type fichier : `Fichier`
- Création d'un fichier :
`Créer(nom_fichier, fichier)`
- Désignation d'un fichier :
`Désigner(nom_fichier, fichier)`
- Ouverture d'un fichier :
`Ouvrir(fichier, <mode>)`
`<mode> ::= lecture | écriture | lecture/écriture`
- Fermeture d'un fichier :
`Fermer(fichier)`

Manipulation de fichiers

- Écriture dans un fichier :

`Ecrire (fichier, données)`

- Lecture d'un fichier :

`Lire (fichier, variable)`

- **Remarque** : Après chaque écriture/lecture, le pointeur (curseur) du fichier est automatiquement avancé.

- Accès direct :

`Décaler (fichier, <position>, décalage)`

`<position> ::= Début_fichier | Fin_fichier | Position_courante`

- Fin d'un fichier :

`Fin(fichier) //fonction du type booléen`

Exemples

Algorithme Lecture d'un fichier d'entiers

```
{
  variable nom_fichier : chaîne de caractères
         f_entier : Fichier
         i : entier

  Afficher("Saisir le nom du fichier d'entiers")
  Saisir(nom_fichier)

  Désigner(nom_fichier, f_entier) // désignation du fichier
  Ouvrir(f_entier, lecture) // ouverture du fichier en lecture

  Tant que (non Fin(f_entier)) // test la fin du fichier
  {
    Lire(f_entier, i) // lecture d'un entier dans le fichier
    Afficher(i, "\n")
  }

  Fermer(f_entier) // fermeture du fichier
}
```

Exemples

Algorithme Ecriture d'un fichier d'entiers

```
{
  variable nom_fichier : chaîne de caractères
         f_entier : Fichier
         i : entier

  Afficher("Saisir le nom du fichier d'entiers")
  Saisir(nom_fichier)

  Créer(nom_fichier, f_entier) // création du fichier
  Ouvrir(f_entier, écriture) // ouverture du fichier en écriture

  Pour i de 1 à 100
    Ecrire(f_entier, i) // écriture dans le fichier

  Fermer(f_entier) // fermeture du fichier
}
```

Attention, cet algorithme est pour un fichier non existant.
Pour un fichier existant, on utilise Désigner au lieu de Créer

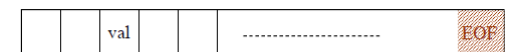
Exemple d'utilisation de fichiers à accès direct : mise à jour d'un fichier d'entiers

```
procédure maj_fich_ent(E f : Fichier)
{
  variable val, n_val, n : entier
         trouvé : booléen

  Afficher("Saisir la valeur à mettre à jour ?")
  Saisir(val)
  Afficher("Saisir la nouvelle valeur ?")
  Saisir(n_val)

  trouvé ← faux;
  Tant que ((non Fin(f)) et (non trouvé)) // recherche de val
  {
    Lire(f, n);
    Si (n=val) alors
      trouvé ← vrai
  }

  Si (trouvé) alors
  {
    Décaler(f, Position_Courante, -1); // positionner le pointeur du fichier
    Ecrire(f, n_val); // mise à jour d'une valeur
  }
  sinon
    Afficher("La valeur ", val, " n'a pas été trouvée.\n");
  }
}
```



Avant Décaler



Après Décaler

Exemple d'utilisation de fichiers à accès direct : mise à jour d'un fichier d'entiers

```
Algorithme Mise à jour
{
    variable nom : chaîne de caractères
           f : Fichier

    Afficher("Saisir le nom du fichier d'entiers")
    Saisir(nom);
    Afficher("mise à jour du fichier d'entiers : ",nom, "\n")

    Désigner(nom,f) // désignation du fichier

    Ouvrir(f,lecture/écriture) // ouverture du fichier en lecture/écriture

    maj_fich_ent(f); // appel à la procédure maj_fich_ent
    Fermer(f);
}
```

Exemple pratique : un fichier d'enregistrements

```
Algorithme Ecriture d'un fichier de Points
{
    constante entier MAX ← 100
    constante chaîne de caractères nom_fichier ← "lespoints.dat"
    variable f : Fichier
           v : vecteur de MAX Point

    Créer(nom_fichier,f) // création du fichier
    Ouvrir(f,écriture) // ouverture du fichier en écriture

    Pour i de 0 à MAX-1
        v[i] ← SaisirPoint()

    Pour i de 0 à MAX-1
        Ecrire(f,v[i]) // écriture dans le fichier

    Fermer(f_entier) // fermeture du fichier
}
```

Exemple pratique : un fichier d'enregistrements

- Écrire un algorithme réalisant la saisie de 100 points et leur stockage dans un fichier créé pour l'occasion et nommé « lespoints.dat ».

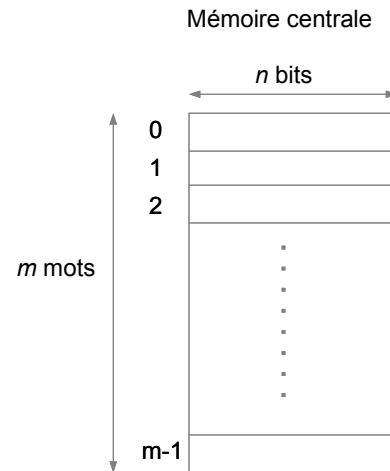
```
structure Point
{
    X,Y : réel
}

fonction SaisirPoint() : Point
{
    variable P : Point
    Afficher("Saisissez les coordonnées du point ")
    Saisir(P.X,P.Y)
    retourne P
}
```

Avant d'aborder les fichiers en C++, nous devons aborder le concept d'adressage et de pointeur.

Souvenons nous de l'organisation de la mémoire centrale

- Un mot mémoire est une unité d'information (un mot binaire – une séquence de bits) adressable en mémoire.
- Suivant l'architecture de la mémoire, le mot pourra être de 8, 16, **32** ou **64** bits.



Mot mémoire et adresses (Théorie)

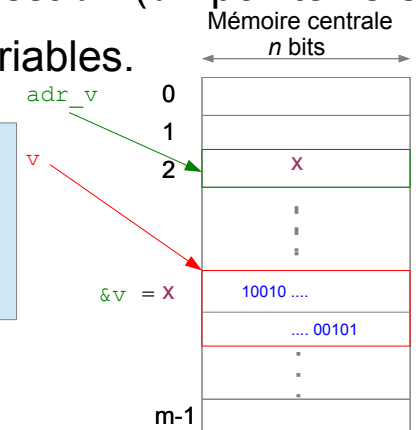
- Ainsi pour une mémoire de 4Go et des mots de 32 bits, il y a 10^9 adresses mémoires.
- Chaque mot (emplacement mémoire) est désigné par une **adresse**.
- Heureusement avec l'exemple ci-dessus, si la technologie utilisée est 32 bits, il est possible d'adresser 2^{32} valeurs, donc de désigner 2^{32} emplacements mémoire. Or avec 4 Go nous avons $10^9 < 2^{32}$
 - Il est en revanche impossible d'adresser plus de 2^{34} octets ≈ 17 Go (2^{32} emplacements * 32 bits pour la taille des emplacements / 8 bits pour avoir la taille en octets).

Opérateur d'adressage : &

- un opérateur unaire qui s'applique à la variable qui le suit ;
- l'adresse de la variable v est $\&v$ ($\&v$ pointe vers v) ;
- $\&$ ne s'applique qu'aux variables.

Ne pas confondre :

- **variable** : un espace mémoire
- **valeur** : contenu d'une zone mémoire
- **adresse** : index d'un mot mémoire
- **pointeur** : variable contenant l'adresse de début d'une zone mémoire



Exemples :

- $\text{adr_v} \leftarrow \&v \Rightarrow \text{adr_v}$ est le pointeur (adresse) vers v
- $\&120 \Rightarrow$ incorrecte
- $\&(a+b) \Rightarrow$ incorrecte.

Mot mémoire / adressage (Pratique)

- Attention ! Certains s'interrogeront peut être sur les données précédentes. En effet, en général, on ne considère qu'on ne peut adresser que 4 Go sur une architecture 32 bits !
- Mais pourquoi est ce que nous trouvons un peu plus de 4 fois plus ?
 - Nous avons considéré une architecture mémoire matérielle appelée « **adressable par mot** » (*word-addressable*) et non une architecture mémoire « **adressable par octet** » (*byte-addressable*). C'est pourtant ce dernier cas qui est le plus fréquent dans la vie courante
 - Ainsi $2^{32} \approx 4\text{Go}$ (les arrondis expliquent le facteur « un peu plus de 4 fois »)



Un entrepôt d'Amazon

La variable : la boîte/le colis/le paquet
 La valeur : le contenu
 L'adresse : la localisation
 Le pointeur : une variable contenant une adresse

L'opérateur de déréférencement : *

- l'opérateur inverse de & pour obtenir le contenu de l'adresse ;
- il s'applique à l'expression qui le suit.

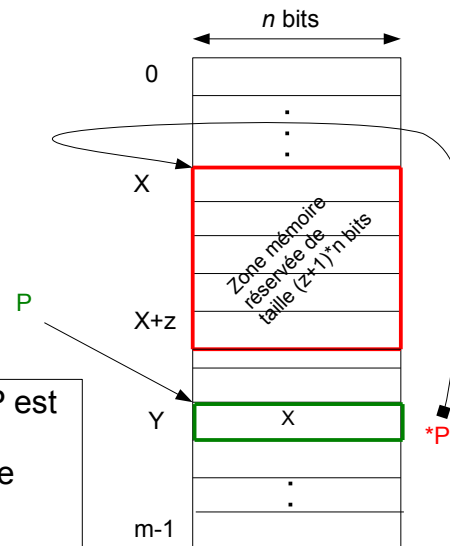
Exemples :

- $v \leftarrow *adr_v$ équivaut à $v \leftarrow *\&v$

c'est à dire $v \leftarrow$ contenu de l'adresse de v .

La notion de pointeur

- Un pointeur est donc une variable qui :
 - a un nom
 - pointe une zone mémoire (désignée par une adresse) qui doit avoir été réservée préalablement à son utilisation



Ici la variable P est un pointeur, pointant la zone mémoire à l'adresse X.

Syntaxe pour les pointeurs en algorithmique et en C++

- Déclaration d'un pointeur :
`nom_pointeur : pointeur sur type_pointé`
- Une valeur particulière NULL (ne pointe sur rien)
`nom_pointeur ← NULL`
- Déclaration d'un pointeur en C++ :
`type_pointé* nom_pointeur;`
- Une valeur particulière NULL (ne pointe sur rien)
`nom_pointeur = NULL;`

Exemple de pointeur

Algorithme Exemple_de_pointeur

```
{
    variable ptr : pointeur sur entier
           A : entier

    ptr ← &A
    ...
}
```

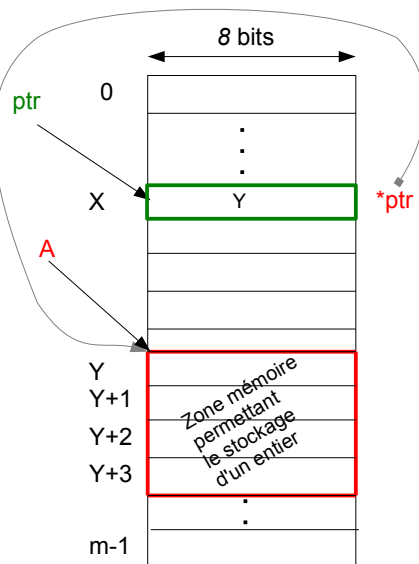
ptr est une variable de type pointeur sur entier. Il occupe donc un espace lui permettant de contenir une adresse en fonction de l'architecture utilisée. Ainsi dans le monde réel sur une architecture 32 bits le pointeur devrait occuper 32 bits (soit la même taille qu'un entier sur 32 bits – ce qui dans le cadre de notre exemple est un peu idiot ! Bref ...).

La déclaration de ptr permet de réserver la zone verte. A ce moment là, la variable ptr, de type pointeur sur entier, contient une valeur particulière par défaut qui s'appelle NULL. Cette valeur indique que le pointeur pointe sur un espace vide.

La déclaration de A permet de réserver la zone rouge.

Une fois que l'instruction d'affectation de l'adresse de A (c'est à dire Y) a eu lieu, il est possible d'accéder à l'espace mémoire permettant le stockage d'un entier (designé par son adresse de début X et représenté en rouge) via l'opérateur de déréférencement *ptr

Pour simplifier, ici, on suppose une mémoire 8 bits et contenant m emplacements, où m = 256. Ainsi, une adresse se représente sur 1 seul emplacement mémoire (sur 8 bits).



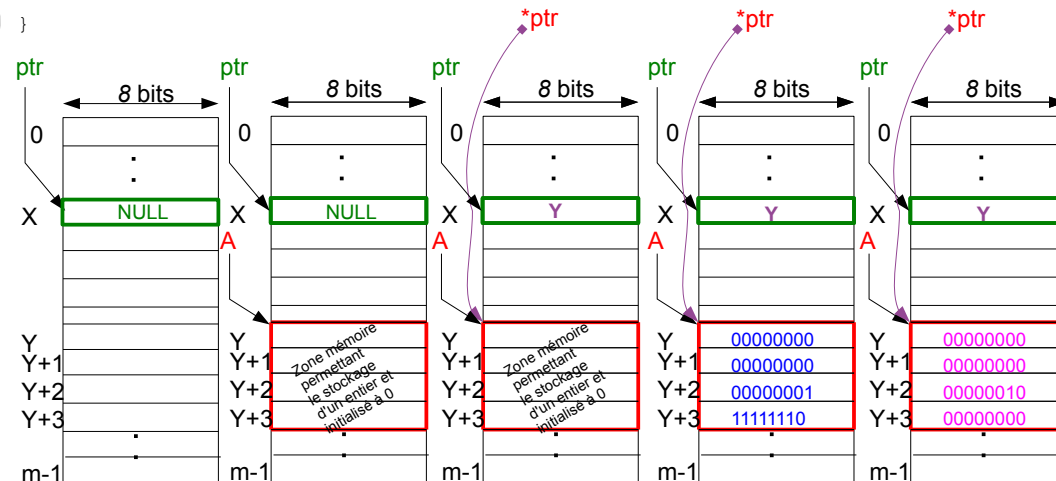
Exemple simpliste d'utilisation de pointeur

Algorithme Exemple_de_pointeur_2

```
{
    variable ptr : pointeur sur entier
           A : entier

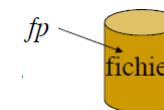
    ptr ← &A
    A ← 510
    *ptr ← *ptr + 514 // identique à A ← *ptr + 514 ou *ptr ← A + 514 ou A ← A + 514
}
```

Pour simplifier, ici, on suppose une mémoire 8 bits et contenant m emplacements, où m = 256. Ainsi, une adresse se représente sur 1 seul emplacement mémoire (sur 8 bits).



Les fichiers en C++ : fopen

- Pour ouvrir un fichier existant ou créer un nouveau fichier.
- Syntaxe : fp = fopen (nom, mode) ;
 - fp : pointeur sur le fichier en utilisant FILE *fp;
 - nom : nom du fichier et le chemin d'accès ;
 - mode : mode d'accès au fichier
 - "r" => lecture ;
 - "w" => écriture ;
 - "a" => ajout ;
 - "r+" => lecture et écriture (mise à jour).
- fopen retourne le pointeur NULL si l'ouverture (création) du fichier est impossible.

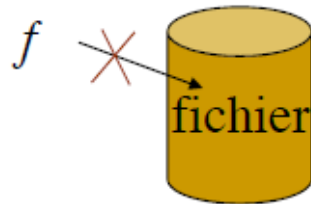


Bon et tout ça pour quoi déjà ?
Ah oui ! Les fichiers en C++ !

Les fichiers en C++ : `fclose`

- Pour **fermer** un fichier.
- Exemple d'utilisation de `fclose` :

```
⋮  
⋮  
fclose(f);  
⋮  
⋮
```



Oublier de fermer un fichier est une erreur fréquente !

Exemple : écriture d'un fichier d'entiers

```
#include <iostream>  
#include <stdio.h>  
using namespace std;  
  
int main()  
{  
    const int n = 512;  
    int i,v[n];  
    char nom[30];  
    FILE *f;  
  
    cout << "Saisir le nom du fichier d'entiers et son chemin d'accès" << endl;  
    cin >> nom;  
    cout << "écriture du fichier d'entiers : " << nom << endl;  
  
    if ((f = fopen(nom,"w")) == NULL) // ouverture du fichier en écriture  
        cout << "erreur d'ouverture (création) du fichier " << nom << endl;  
    else  
    {  
        for (i=0; i<n; i++) // on remplit le vecteur  
            v[i] = i;  
  
        fwrite(v,sizeof(int),n,f); // écriture de v dans le fichier  
        fclose(f); // fermeture du fichier  
    }  
}
```

Un exemple plus complet

```
⋮  
⋮  
f = fopen("/users/etudiants/f_ent.dat","w"); ⇒ EOF  
⋮  
v[0] = 5;  
v[1] = 6;  
  
fwrite(v,sizeof(int),2,f); ⇒ [ 5 | 6 | EOF ]  
⋮  
fclose(f);  
⋮  
⋮
```

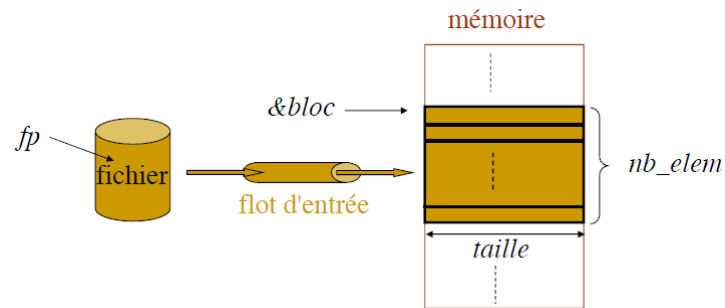
Les fichiers en C++ : `fread`

- Pour la **lecture** d'un fichier.
- Syntaxe :
`nb_lu = fread (&bloc,taille,nb_elem,fp);`
 - `&bloc`, `taille` et `fp` comme précédemment
 - `nb_elem` : nombre d'enregistrements (fiches) du fichier à lire et à placer dans le bloc ("buffer") ;
 - `nb_lu` : nombre d'enregistrements (fiches) du fichier effectivement lus et placés dans le bloc.
- Remarques :
 - `nb_lu < nb_elem` en fin du fichier (dernière lecture) ;
 - `nb_lu = 0` => fin du fichier
 - après la lecture de chaque enregistrement (fiche), le pointeur de fichier (curseur) est automatiquement avancé d'un enregistrement.

Les fichiers en C++ : fread

```
nb_lu = fread (&bloc,taille,nb_elem,fp);
```

- &bloc, taille et fp comme précédemment
- nb_elem : nombre d'enregistrements (fiches) du fichier à lire et à placer dans le bloc ("buffer") ;
- nb_lu : nombre d'enregistrements (fiches) du fichier effectivement lus et placés dans le bloc.

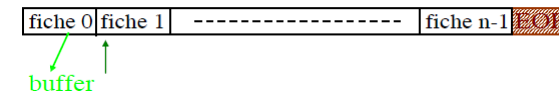


Exemple d'utilisation de fread

```

:
:
const int n = 512;
int v[n];
:
:
nb_lu = fread(v, sizeof(int), n, f);
:
:

```



Un exemple plus complet

```

:
:
f = fopen("/users/etudiants/f_ent.dat", "r"); => [5 | 6 | EOF]

nb_lu = fread(v, sizeof(int), 2, f); => [5 | 6 | EOF]

cout << v[0] << " " << v[1] << endl.

fclose(f);
:
:

```

Exemple : lecture d'un fichier d'entiers

```

#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
    const int n = 512;
    int i,nb_lu,v[n];
    char nom[30];
    FILE *f;

    cout << "Saisir le nom du fichier d'entiers et son chemin d'accès" << endl;
    cin >> nom;
    cout << "lecture du fichier d'entiers : " << nom << endl;

    if ((f = fopen(nom,"r")) == NULL) // ouverture du fichier en lecture
        cout << "erreur d'ouverture du fichier " << nom << endl;
    else
    {
        while ((nb_lu = fread(v,sizeof(int),n,f)) > 0) // lecture du fichier
        {
            for (i=0; i<(nb_lu); i++)
                cout << v[i] << endl; // affichage des éléments lus
        }
        fclose(f); // fermeture du fichier
    }
}

```

Les fichiers en C++ : `feof`

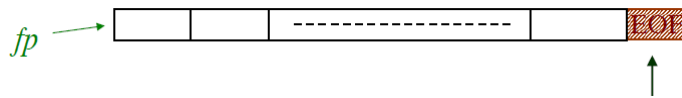
- Pour la détection de la **fin** d'un fichier.

- Syntaxe :

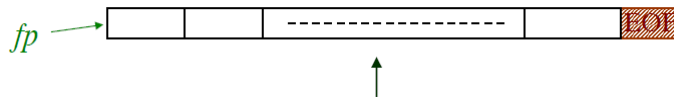
```
feof (fp) ;
```

- `fp` : pointeur sur le fichier en utilisant `FILE *fp`

- `feof = true` si le **pointeur** (curseur) du fichier associé à `fp` est à la fin ;



- `feof = false` sinon.



Les fichiers en C++ : `fseek`

- Pour **positionner le pointeur du fichier** dans le cas d'un **accès direct**

- Syntaxe :

```
fseek (fp, f_pos, origine) ;
```

- `fp` : pointeur sur le fichier en utilisant `FILE *fp`

- `f_pos` : position relative à l'origine en **octets**

- origine :

- `SEEK_SET` => début du fichier ;
- `SEEK_CUR` => position courante ;
- `SEEK_END` => fin du fichier ;

- Remarque : `fseek` retourne une valeur $\neq 0$ si erreur.

Exemple : lecture d'un fichier d'entiers en utilisant `feof`

```
#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
    const int n = 512;
    int i,nb_lu,v[n];
    char nom[30];
    FILE *f;

    cout << "Saisir le nom du fichier d'entiers et son chemin d'accès" << endl;
    cin >> nom;
    cout << "lecture du fichier d'entiers : " << nom << endl;

    if ((f = fopen(nom,"r")) == NULL) // ouverture du fichier en lecture
        cout << "erreur d'ouverture du fichier " << nom << endl;
    else
    {
        while ( !feof(f) ) // tant que la fin du fichier n'est pas atteinte
        {
            nb_lu = fread(v,sizeof(int),n,f); // lecture du fichier
            for (i=0; i<(nb_lu); i++)
                cout << v[i] << endl; // affichage des éléments lus
        }
        fclose(f); // fermeture du fichier
    }
}
```

Exemple : Mise à jour d'un fichier d'entiers

```
#include <iostream>
#include <stdio.h>
using namespace std;

void maj_fich_ent(FILE *f)
{
    int val,n_val,n;
    bool trouve;

    cout << "Saisir la valeur à mettre à jour ?" << endl;
    cin >> val;
    cout << "Saisir la nouvelle valeur ?" << endl;
    cin >> n_val;

    trouve = false;
    while ( (! feof(f)) && (! trouve) ) // recherche de val
    {
        fread(&n,sizeof(int),1,f);
        if (n == val)
            trouve = true;
    }

    if (trouve)
    {
        fseek(f,-sizeof(int),SEEK_CUR); // positionner le pointeur du fichier
        fwrite(&n_val,sizeof(int),1,f); // mise à jour d'une valeur
    }
    else cout << val << " n'est pas trouvée" << endl;
}
```

Exemple : Mise à jour d'un fichier d'entiers

```
#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
    char nom[30];
    FILE *f;

    cout << "Saisir le nom du fichier d'entiers et son chemin d'accès" << endl;
    cin >> nom;
    cout << "mise à jour du fichier d'entiers : " << nom << endl;

    if ((f = fopen(nom,"r+")) == NULL) // ouverture du fichier en "mise à jour"
        cout << "erreur d'ouverture du fichier " << nom << endl;
    else
    {
        maj_fich_ent(f);
        fclose(f); // fermeture du fichier
    }
}
```

Exemple pratique : un fichier d'enregistrements

```
#include <iostream>
#include <stdio.h>
using namespace std;

... // Insérer ici la définition du type Point et la fonction SaisirPoint

int main()
{
    const int MAX = 100;
    const char[] nom = "lespoints.dat";
    Point v[MAX];
    FILE *f;

    if ((f = fopen(nom,"w")) == NULL) // ouverture du fichier en écriture
        cout << "erreur d'ouverture (création) du fichier " << nom << endl;
    else
    {
        for (i=0; i<MAX; i++) // on remplit le vecteur
            v[i] = SaisirPoint();

        for (i=0; i<MAX; i++)
            fwrite(v,sizeof(Point),MAX,f); // écriture de v dans le fichier

        fclose(f); // fermeture du fichier
    }
}
```

Exemple pratique : un fichier d'enregistrements

- Écrire **un programme** réalisant la saisie de 100 points et leurs stockage dans un fichier créé pour l'occasion et nommé « lespoints.dat ».

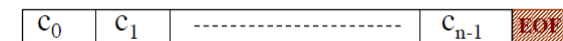
```
struct Point
{
    float X,Y;
};

Point SaisirPoint()
{
    Point P;
    cout << "Saisissez les coordonnées du point ";
    cin >> P.X >> P.Y ;
    return P;
}
```

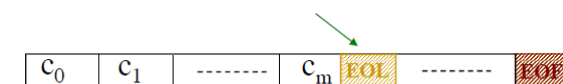
Les fichiers de texte

- Caractéristiques :

- élément de base est **caractère** ;



- fichier peut être structuré en **lignes** : chaque ligne terminée par une marque de fin de ligne (EOL pour End Of Line);



- longueur des lignes est **variable** ;
- les nombres sont représentés sous forme de suite de chiffres (des chaînes de caractères).

Les fichiers en C++ : `fgetc` et `fputc`

- Pour lire un caractère et pour écrire un caractère dans un fichier

- Syntaxe :

```
c = fgetc(fp);
```

- `fp` : pointeur sur le fichier en utilisant `FILE *fp`
- `c` : le caractère lu

```
fputc(c, fp);
```

- `c` : le caractère à écrire
- `fp` : pointeur sur le fichier en utilisant `FILE *fp`

Les fichiers en C++ : `fprintf`

- Format de données

- `%` : spécificateur de format ;
- `%d` : écriture sous forme décimale (entier) ;
- `%o` : écriture sous forme octale (entier) ;
- `%x` : écriture sous forme hexadécimale (entier) ;
- `%u` : écriture sous forme décimale non-signée (entier) ;
- `%f` : écriture sous forme flottante (réels) ;
- `%c` : écriture sous forme caractère ASCII (entier) ;
- `%s` : écriture sous forme chaîne de caractères ;

– Exemple :

```
a = 75;
```

```
fprintf(stdout, "a(déc)=%d , a(oct)=%o , a(hex)=%x , a(car)=%c", a, a, a, a);
```

Affiche à l'écran (`stdout`) – Oui ! L'écran est un fichier ! :-)

=> a(déc)=75 , a(oct)=113 , a(hex)=4b , a(car)=K

Les fichiers en C++ : `fprintf`

- Pour des écritures (sorties) formatées
- Syntaxe :

```
fprintf(fp, "format", arg1, arg2, ...);
```

- `fp` : pointeur sur le fichier en utilisant `FILE *fp`
- `format` est une chaîne de caractères avec 2 types de données :
 - texte à écrire ;
 - format de données (`arg1`, `arg2`, ...) à écrire.

- Exemples :

- `fprintf(fp, "bonjour");` => `bonjour` est écrit dans le fichier.
- `fprintf(fp, "bonjour\n");` => `bonjour` est écrit dans le fichier + 1 nouvelle ligne.

Exemple : écriture d'un fichier de type texte

```
#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
    const int max=30, n = 512;
    int i ;
    char nom[max];
    FILE *f;

    cout << "Saisir le nom du fichier texte et son chemin d'accès" << endl;
    cin >> nom;

    if ((f = fopen(nom, "w")) == NULL) //ouverture du fichier en écriture (création)
        cout << "erreur d'ouverture (création) du fichier " << nom << endl;
    else
    {
        for (i=0; i<n; i++)
            fprintf(f, "%d\n", i); //écriture par ligne dans le fichier

        fclose(f); // fermeture du fichier
    }
}
```

Chaque chiffre est codé dans le fichier en ASCII =>

- 0 à 9 : 1 octet / entier ;
- 10 à 99 : 2 octets / entier ;
- 100 à 999 : 3 octets / entier ;

Les fichiers en C++ : `fscanf`

- Pour des lectures (entrées) **formatées**

- Syntaxe :

```
fscanf(fp, "format", &arg1, &arg2, ...);
```

- `fp` : pointeur sur le fichier en utilisant `FILE *fp`
- `format` est une chaîne de caractères précisant le format de données (`arg1`, `arg2`, ...) à saisir.

- Exemples :

- `fscanf(fp, "%d", &a); =>` lecture d'un entier dans le fichier associé à `fp` et son affectation à `a`.
- `fscanf(fp, "%s", ch); =>` lecture d'une chaîne de caractères dans le fichier associé à `fp` et son affectation à `ch`
 - & `ch` n'a pas raison d'être car en fait, `ch` représente 1 pointeur sur 1 chaîne de caractères.
- `fscanf(stdin, "%d %s %d", &jour, mois, &an); =>` affectation de 2 entiers et d'une chaîne de caractères saisis sur le clavier (`stdin`). – Oui ! Le clavier est aussi un fichier ! :-)

Exemple : Lecture et affichage de tout fichier de type texte. v2

```
#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
    const int max = 30;
    char nom[max], c;
    FILE *f;

    cout << "Saisir le nom du fichier texte et son chemin d'accès" << endl;
    cin >> nom;

    if ((f = fopen(nom, "r")) == NULL) // ouverture du fichier en lecture
        cout << "erreur d'ouverture du fichier " << nom << endl;
    else
    {
        while ((c=fgetc(f))!=EOF) // lecture du fichier caractère par caractère
            putchar(c); // affichage caractère par caractère
            // putchar(c) ≡ fputc(c, stdout) c-à-d écriture sur l'écran !

        fclose(f); // fermeture du fichier
    }
}
```

Exemple : Lecture et affichage de tout fichier de type texte. v1

```
#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
    const int max = 30;
    char nom[max], c;
    FILE *f;

    cout << "Saisir le nom du fichier texte et son chemin d'accès" << endl;
    cin >> nom;

    if ((f = fopen(nom, "r")) == NULL) // ouverture du fichier en lecture
        cout << "erreur d'ouverture du fichier " << nom << endl;
    else
    {
        while (fscanf(f, "%c", &c)>0) // lecture du fichier caractère par caractère
            putchar(c); // affichage caractère par caractère
            // putchar(c) ≡ fputc(c, stdout) c-à-d écriture sur l'écran !

        fclose(f); // fermeture du fichier
    }
}
```

Et on en a fini avec les fichiers !
Ouf !

Gestion dynamique de la mémoire

Allocations de mémoire

- 2 types d'allocations de mémoire :
 - ~ statique
 - Celle que nous avons utilisée jusqu'à présent
 - ~ dynamique
 - Celle que nous allons voir maintenant

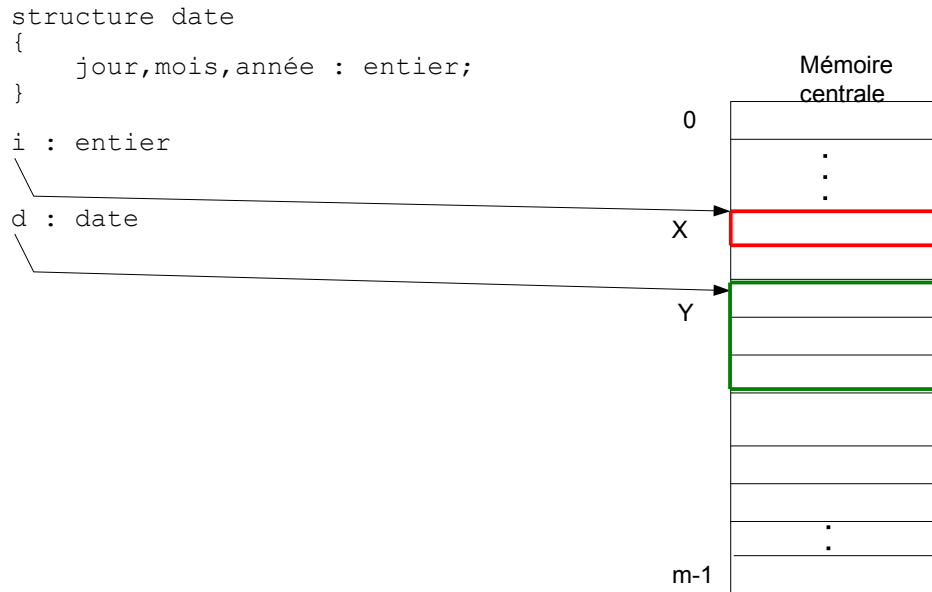
Allocation statique

- Dans les algorithmes que nous avons écrits jusqu'à présent, les variables que nous avons déclarées sont :
 - ~ Soit des variables **globales** (définies dans l'algorithme principal)
 - ~ Soit des variables **locales** (définies dans les sous-programmes)
 - ~ Ces variables doivent être déclarées **explicitement** pendant l'écriture des algorithmes et ont une durée de vie précise (la durée de vie du bloc contenant la déclaration) :
 - Les variables globales sont créées au début de l'exécution et sont conservées jusqu'à la fin de l'exécution
 - Les variables locales sont créées au début de l'exécution du sous-programme et sont supprimées à la fin de son exécution

Allocation statique

- **Remarque 1** : Comme nous l'avons vu lors de l'introduction du concept de **variable**, l'opération de « création » de l'espace mémoire associé à une variable s'appelle **l'allocation**.
- **Remarque 2** : L'allocation est dite « **statique** » car décidée au moment de l'écriture de l'algorithme et non évolutive lors de l'exécution.
- **Conséquence** : Cela a de **nombreux inconvénients**. Par exemple, il n'est pas possible de contrôler au mieux l'espace mémoire disponible.

Allocation statique



Allocation statique Inconvénient 2

- Fréquemment, par exemple dans le cas d'un vecteur, on ne peut pas connaître, a priori lors de l'écriture de l'algorithme, le nombre d'éléments qu'il contiendra. Cette taille n'est bien souvent calculable ou connu **que pendant** l'exécution du programme.
- Il est donc indispensable de disposer d'un mécanisme permettant de **créer** (et de **supprimer**) des variables de différents types pendant l'exécution.
- C'est ce qu'on appelle **la gestion dynamique** (pendant l'exécution) de la mémoire. Elle comprend deux opérations:
 - ~ Allouer
 - ~ Libérer

Allocation statique Inconvénient 1 : durée de vie

- Si pour des variables simples (entiers, réels, ...) il n'est pas très pénalisant de garder des variables jusqu'à la fin de l'exécution (même si elle ne nous servent plus), il n'en est pas toujours ainsi.
 - ~ Par exemple, imaginons que une variable qui ne soit pas non plus un simple entier mais un vecteur de 200000 éléments entier !
Pendant l'exécution du programme (algorithme), il est indispensable de pouvoir supprimer le vecteur s'il n'est plus utilisé et ainsi libérer de la mémoire afin d'éviter une éventuelle saturation.

Allocation dynamique

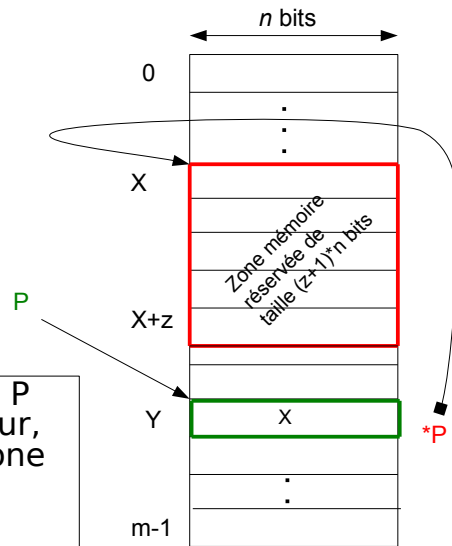
- Il n'y a pas de déclaration explicite de variables.
- L'accès se fait par le moyen d'un **pointeur** (1 adresse mémoire)
- L'espace mémoire existe juste pour la durée nécessaire.
- On utilise l'opérateur `Allouer`
 - ~ Cette opération permet de réserver une zone mémoire et permet d'accéder à cette zone en utilisant la valeur qu'elle retourne
 - ~ On peut stocker cette valeur, **une adresse mémoire**, dans ce que l'on appelle un **pointeur**.

Syntaxe pour les pointeurs

Rappel sur la notion de pointeur

- Un pointeur est une variable de type `pointeur` qui :
 - a un nom
 - pointe une zone mémoire (désignée par une adresse) qui doit avoir été réservée préalablement à son utilisation

Ici la variable P est un pointeur, pointant la zone mémoire à l'adresse X.



Exemple de pointeur

Algorithme Exemple_de_pointeur

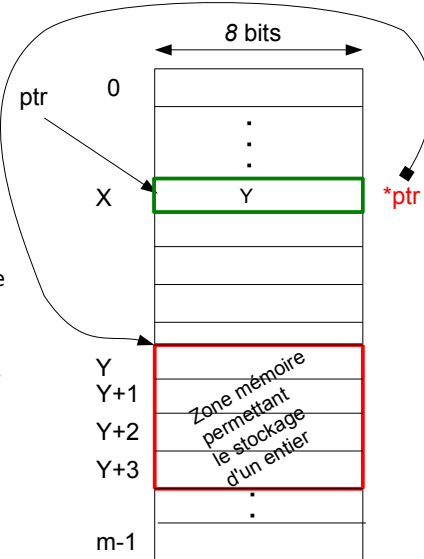
```
{
    variable ptr : pointeur sur entier
    ptr ← Allouer entier
    ...
}
```

ptr est une variable de type pointeur sur entier. Il occupe donc un espace lui permettant de contenir une adresse en fonction de l'architecture utilisée. Ainsi dans le monde réel sur une architecture 32 bits le pointeur devrait occuper 32 bits (soit la même taille qu'un entier sur 32 bits – ce qui dans le cadre de notre exemple est un peu idiot ! Bref ...).

La déclaration permet de réserver la zone verte. À ce moment là, la variable ptr, de type pointeur sur entier, contient une valeur particulière par défaut qui s'appellera NULL. Cette valeur indique que le pointeur pointe sur un espace vide (il n'y a pas eu d'allocation)

Une fois que l'instruction Allouer entier a eu lieu, un espace mémoire permettant le stockage d'un entier a été créé (désigné par son adresse de début X et représenté en rouge) et puisque le résultat de cette opération est affectée (←) au pointeur ptr, celui-ci contient l'adresse de la zone (soit Y ici). Il est possible d'accéder à l'espace mémoire permettant le stockage d'un entier (désigné par son adresse de début X et représenté en rouge) via l'opérateur de déréférencement *ptr

Pour simplifier, ici, on suppose une mémoire 8 bits et contenant m emplacements, où m = 256. Ainsi, une adresse se représente sur 1 seul emplacement mémoire (sur 8 bits).



- Déclaration d'un pointeur :


```
nom_pointeur : pointeur sur type_pointé
```
- Allocation mémoire d'un pointeur


```
nom_pointeur ← Allouer type_pointé
```
- Dans l'algorithme, afin de pouvoir utiliser un pointeur, il doit obligatoirement y avoir
 - tout d'abord, la déclaration (qui permet la création d'une zone en mémoire qui pourra contenir une adresse (la zone désignée par la variable P sur le schéma précédent))
 - puis l'allocation qui permet de créer la zone mémoire correspondant au type_pointé (zone en rouge sur le schéma précédent) et affecte l'adresse (X sur le schéma précédent) comme valeur à la variable de type pointeur.

Syntaxe pour les pointeurs

- Désallocation d'un élément pointé


```
Libérer nom_pointeur
```
- Cette opération libère la zone mémoire occupée par l'élément pointé et affecte NULL comme valeur au pointeur.
- Toutefois, attention dans la pratique, selon le langage de programmation utilisé, lorsque la zone pointée est désallouée, le pointeur conserve sa valeur et par conséquent, il pointe sur une zone mémoire non valide.

Or, il ne faut jamais accéder au contenu d'une zone mémoire libérée sous peine de causer un problème à l'exécution (en effet la zone a pu être réutilisée entre temps par un autre programme).

La bonne pratique veut que dès que l'on a désalloué un pointeur, on lui donne la valeur particulière NULL (on peut aussi le faire en algorithmique)

Exemple de pointeur

Manipulation de pointeur

- Si un pointeur permet de désigner un espace mémoire, il est utile de pouvoir modifier cet espace mémoire. Pour cela l'opérateur unaire `*` (opérateur de déréréférencement) permet d'accéder à la zone mémoire pointée comme s'il s'agissait d'une variable.
- Rappel :
 - ~ L'opérateur `*` permet donc d'accéder au contenu de l'adresse.
 - ~ L'opérateur `*` s'applique à l'expression (en général, il s'agit d'une variable) qui le suit.
- Exemple :

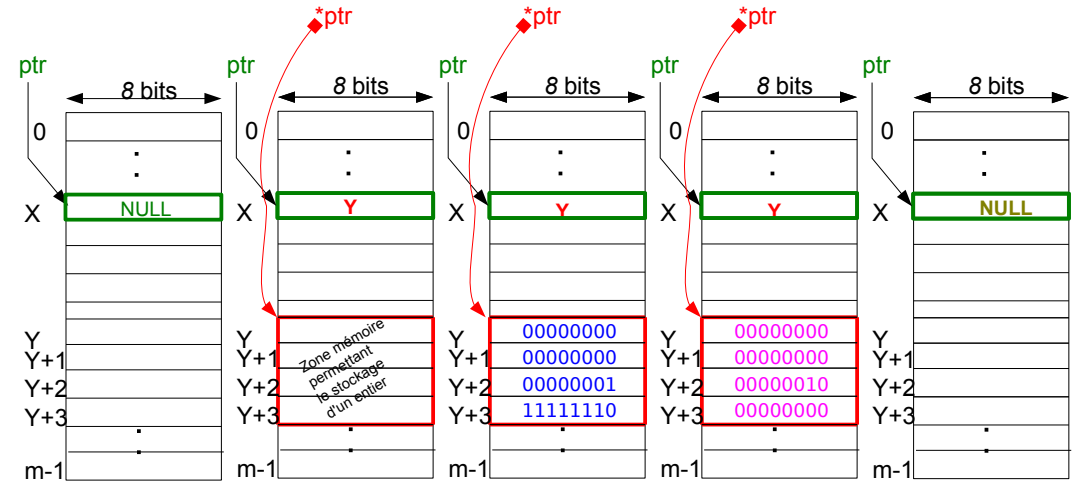
`*nom_pointeur ← une_valeur_compatible_avec_le_type_pointé`

Algorithme Exemple_de_pointeur_2

```

{
  variable ptr : pointeur sur entier
  ptr ← Allouer entier
  *ptr ← 510
  *ptr ← *ptr + 514
  Libérer ptr
}
    
```

Pour simplifier, ici, on suppose une mémoire 8 bits et contenant m emplacements, où m = 256. Ainsi, une adresse se représente sur 1 seul emplacement mémoire (sur 8 bits).



Exemple de pointeur

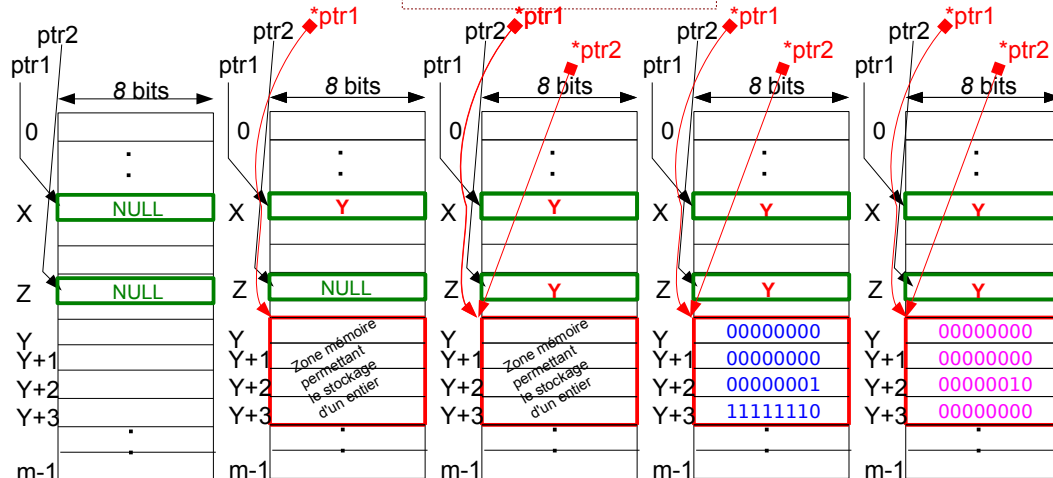
Algorithme Exemple_de_pointeur_3

```

{
  variable ptr1, ptr2 : pointeur sur entier
  ptr1 ← Allouer entier
  ptr2 ← ptr1
  *ptr1 ← 510
  *ptr2 ← *ptr2 + 514
}
    
```

On peut affecter un pointeur à un autre. Deux pointeurs peuvent désigner et utiliser la même zone mémoire.

Pour simplifier, ici, on suppose une mémoire 8 bits et contenant m emplacements, où m = 256. Ainsi, une adresse se représente sur 1 seul emplacement mémoire (sur 8 bits).



Pointeur : des erreurs à éviter

- Une première erreur dans l'utilisation des pointeurs, consiste à oublier de désallouer l'espace mémoire avant la fin de l'algorithme/programme (auquel cas l'espace mémoire continue à être réservé et donc ne peut pas être utilisé par les autres algorithmes/programmes en cours d'exécution ou allant s'exécuter dans l'avenir.

Illustration de l'erreur 1

Algorithme Exemple_de_pointeur_4

```

{
  variable ptr : pointeur sur entier
  ptr ← Allouer entier
  *ptr ← 510
}
  
```

Pour simplifier, ici, on suppose une mémoire 8 bits et contenant m emplacements, où m = 256. Ainsi, une adresse se représente sur 1 seul emplacement mémoire (sur 8 bits).

Après la fin de l'exécution de l'algorithme, l'entier alloué continue à occuper inutilement de la mémoire. Cette zone ne sera plus accessible pour aucun algorithme/programme. La mémoire est « perdue ».

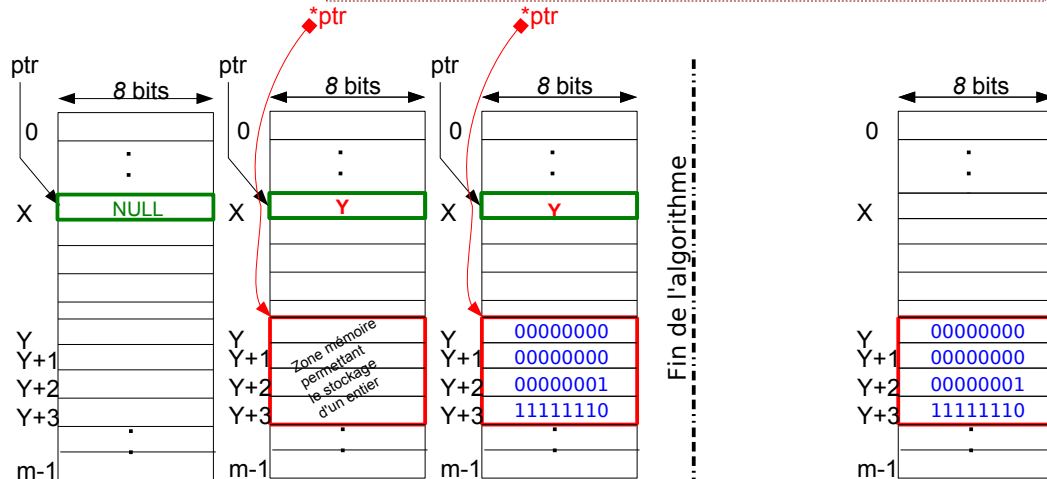


Illustration de l'erreur 2

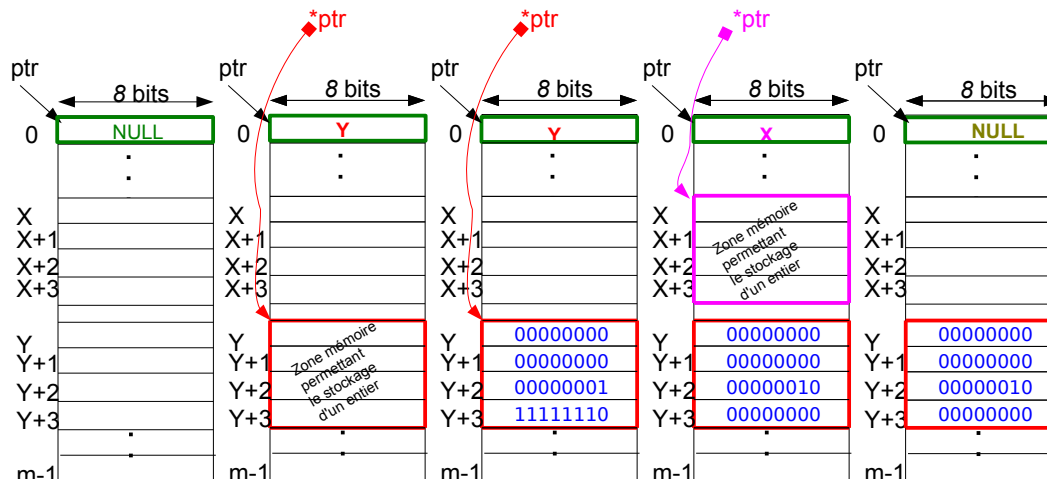
Algorithme Exemple_de_pointeur_5

```

{
  variable ptr : pointeur sur entier
  ptr ← Allouer entier
  *ptr ← 510
  ptr ← Allouer entier
  Libérer ptr
}
  
```

Pour simplifier, ici, on suppose une mémoire 8 bits et contenant m emplacements, où m = 256. Ainsi, une adresse se représente sur 1 seul emplacement mémoire (sur 8 bits).

Après la seconde allocation effectuée au pointeur, l'espace mémoire rouge est « perdu ». Plus aucun pointeur ne connaissant son adresse pour y accéder, il n'existe plus de moyen de libérer cette zone. Il aurait fallu copier le pointeur dans un autre, préalablement à la nouvelle allocation afin de toujours pointer cette zone.



Pointeur : des erreurs à éviter

- Une seconde erreur consiste à allouer un nouvel espace mémoire et à l'assigner à un pointeur en oubliant de désallouer l'espace mémoire précédemment pointé par ce dernier (ce qui est problématique si aucun autre pointeur ne désigne cet espace – en effet cet espace est désormais considéré comme « perdu »)

Pointeur : des erreurs à éviter

- Une troisième erreur est de considérer que ce n'est pas parce qu'un pointeur n'est pas NULL qu'il pointe sur une zone valide.
- Illustration sur le transparent suivant !
- Bonne pratique : mettre à NULL tous les pointeurs désignant la même zone mémoire dès que l'un d'eux est désalloué.

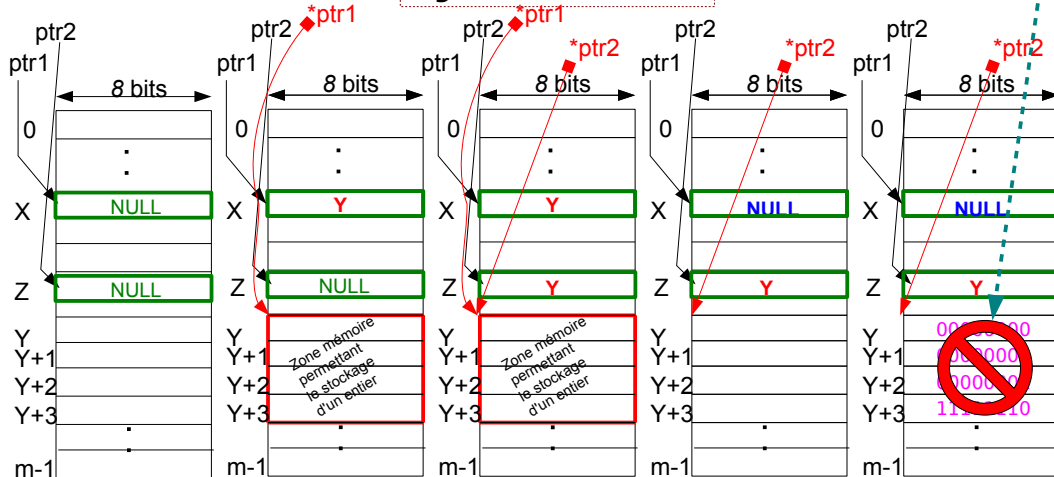
Illustration de l'erreur 3

Algorithme Exemple_de_pointeur_6

```
{
  variable ptr1, ptr2 : pointeur sur entier
  ptr1 ← Allouer entier
  ptr2 ← ptr1
  Libérer ptr1
  *ptr2 ← 510
}
```

Impossible puisque la zone mémoire n'est plus réservée (elle peut donc être utilisée par un autre algorithme).

Pour simplifier, ici, on suppose une mémoire 8 bits et contenant m emplacements, où m = 256. Ainsi, une adresse se représente sur 1 seul emplacement mémoire (sur 8 bits).



Exemple de bonnes pratiques

Algorithme Un_Exemple_Pour_Finir_Avec_Les_Pointeurs_Illustrant_De_Bonnes_Pratiques

```
{
  variable ptr : pointeur sur entier
  ptr ← Allouer entier
  Afficher("Saisissez un entier :")
  Saisir(*ptr)
  Afficher("Vous avez saisi :", *ptr)
  Libérer ptr
  ptr ← NULL
  ...
  Si (ptr = NULL) alors
  {
    ...
    ptr ← Allouer entier
  }
  ...
  Si (ptr ≠ NULL) alors
  {
    ...
    Libérer ptr
  }
}
```

Exemple de bonne pratique. Libérer dès qu'on n'a plus besoin de l'espace mémoire et mettre à NULL le pointeur

Exemple de bonne pratique. Tester le pointeur avant de le réallouer

Exemple de bonne pratique. Tester le pointeur avant de le libérer

Remarque sur les pointeurs

- Il est impossible d'affecter directement une valeur à un pointeur

```
variable ptr : pointeur sur entier
ptr ← Allouer entier
ptr ← 10
```

- En revanche l'affectation suivante est valide : *ptr ← 10

- Il n'est pas possible d'afficher une variable pointeur

```
Afficher(ptr)
```

- En revanche il est possible d'afficher le contenu d'une variable pointeur

```
Afficher(*ptr)
```

Allocation dynamique et vecteur

- Il arrive fréquemment que l'on ne connaisse pas à l'avance le nombre d'éléments que devra comporter un vecteur. Bien sûr, une solution consisterait à déclarer un vecteur d'une taille très importante, mais engendrerait beaucoup de problèmes.
- Aussi, pour parer à cette situation, nous avons la possibilité de déclarer le vecteur sans préciser au départ son nombre d'éléments (souvenez vous certains passages de vecteur en paramètre de fonction ou procédure).
- Ce n'est que dans un second temps, au cours de l'algorithme, que l'on va fixer ce nombre via l'opérateur Allouer
- Un vecteur sans taille initiale se déclare ainsi : V: vecteur d'entier

Exemple de déclaration dynamique

Algorithme Exemple_Declaration_Dynamique_vecteur_Et_Tri

```
{
    variable tab1, tab2, tab_fusion : vecteur d'entier
           n1, n2, n3 : entier

    Saisir_tab(tab1, n1) //La procédure Saisir_tab est donnée après
    Saisir_tab(tab2, n2)

    // Tri les deux vecteurs avec la procédure vue dans le cours
    Tri_min_max(tab1, n1)
    Tri_min_max(tab2, n2)

    n3 ← n1 + n2
    tab_fusion ← Allouer vecteur de n3 entier

    // Appel à la procédure de fusion vue dans le cours.
    Fusion_2vecteurs(tab1, tab2, n1, n2, tab_fusion)

    Libérer tab1
    tab1 ← NULL

    Libérer tab2
    tab2 ← NULL

    // Faire ici un affichage de tab_fusion

    Libérer tab_fusion
    tab_fusion ← NULL
}
```

L'allocation dynamique peut avoir lieu dans un sous programme et la libération intervenir dans l'algorithme principal (et vice versa). Il faut juste faire attention à ce que l'on fait.

Pointeur et enregistrements

- Il est bien sur possible de déclarer un pointeur sur un type structuré. Ainsi l'accès au champ se fera de la façon suivante :
- Toutefois pour des raisons de commodité, une autre syntaxe existe.

`(*nom_pointeur).champ_de_l_enregistrement`

`nom_pointeur->champ_de_l_enregistrement`

Exemple de déclaration dynamique

```
procedure Saisir_tab(E/S V: vecteur d'entier, S N: entier)
{
    variable i: entier
    Afficher("Saisissez le nombre d'éléments à saisir")
    Saisir(N)
    V ← Allouer vecteur de N entier
    Pour i de 0 à N-1
    {
        Afficher("Saisir la valeur de l'élément ", i)
        Saisir(V[i])
    }
}
```

Exemple : Pointeur et enregistrement

```
structure Pays
{
    nom : chaine de caractère
    nb_hab : entier
}

Algorithme Pointeur_Enregistrement
{
    variable ppays : pointeur sur Pays
    ppays ← Allouer Pays
    Afficher("Saisissez le nom du pays et sa population")
    Saisir(ppays->nom, ppays->nb_hab) // équivalent de
                                   // Saisir((*ppays).nom, (*ppays).nb_hab)
    Afficher("Vous avez saisi que le ", ppays->nom, " a ", ppays->nb_hab, "habitant(s).")
    Libérer ppays
}
```

Exemple : Vecteur, allocation dynamique, enregistrement, tri

- Écrire l'algorithme demandant à l'utilisateur de saisir le nombre de pays membres de l'ONU, de saisir les noms et nombre d'habitants de chacun d'eux puis qui triera ces pays dans l'ordre décroissant du nombre d'habitants avant des les afficher.

```
procédure Tri_Pays_par_nb_hab(E/S v:vecteur de Pays, E n:entier)
{
    variable i, etape, i_min: entier
           p: Pays
    Pour etape de 1 à (n - 1)
    {
        /* Recherche du min */
        i_min ← 0
        Pour i de 1 à (n - etape)
        {
            Si (v[i].nb_hab < v[i_min].nb_hab) alors
                i_min ← i
        }
        /* permutation */
        p ← v[i_min]
        v[i_min] ← v[n - etape]
        v[n - etape] ← p
    }
}
```

Adaptation du tri par recherche du minimum.

```
procédure Afficher_Pays(E/S v:vecteur de Pays, E n:entier)
{
    variable i : entier

    Afficher("La liste des ",n," pays membres de l'ONU classés par l'importance de leur population sont : ")

    Pour i de 0 à n-1
    {
        Afficher(i+1,") ",v[i].nom," avec ",v[i].nb_hab," habitants.\n")
    }
}
```

```
structure Pays
{
    nom : chaîne de caractère
    nb_hab : entier
}

Algorithme Tri_Pays_ONU
{
    variable v_pays : vecteur de Pays
           nb_pays :

    Saisie_Pays_ONU(v_pays, nb_pays)

    Tri_Pays_par_nb_hab(v_pays, nb_pays)

    Afficher_Pays(v_pays, nb_pays)

    Libérer v_pays
    v_pays ← NULL
}

procédure Saisie_Pays_ONU(E/S v:vecteur de Pays, S n:entier)
{
    variable i : entier

    Afficher("Saisissez le nombre de pays actuellement membre de l'ONU :")
    Saisir(n)

    v ← Allouer vecteur de n Pays
    Pour i de 0 à n-1
    {
        Afficher("Saisir le nom du pays :")
        Saisir(v[i].nom)
        Afficher("Saisir le nombre d'habitants du pays :")
        Saisir(v[i].nb_hab)
    }
}
```

Allocation dynamique en C++

- Pour une allocation dynamique l'opérateur **new** est utilisé.
- Exemples :

```
int *p; // déclaration d'un pointeur sur 1 entier
p = new int; // allocation de mémoire pour 1 entier
ou directement : int *p = new int;
```

```
float *tab;
tab = new float[30]; // allocation de mémoire pour 30 réels
ou directement : float *tab = new float[30];
```

```
char *ch;
ch = new char[30]; // allocation de mémoire pour 30 caractères
```

```
ou directement : char *ch = new char[30];
```

Désallocation en C++

- Pour libérer physiquement la place allouée l'opérateur `delete` est utilisé.
- Exemples :

```
delete p; // libérer la place allouée par new int
```

=> p reçoit NULL

```
delete tab;
```

```
delete ch;
```

Pointeur sur structure en C++

- L'accès au champ se fera de la façon suivante :
- Ou avec une syntaxe plus commode, une autre syntaxe existe.

```
(*nom_pointeur).champ_de_l_enregistrement
```

```
nom_pointeur->champ_de_l_enregistrement
```

- Exemple

```
struct point  
{  
    float x,y;  
};
```

...

```
point *pp = new point;  
*pp.x = 12;           // première façon  
pp->y=7;              // deuxième façon
```

VOUS AVEZ DIT ARDUINO ?

Introduction à Arduino

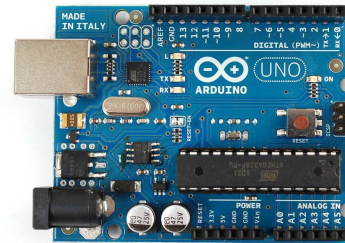
- **Arduino** est une **plate-forme de prototypage** d'objets interactifs à usage créatif constituée d'une **carte électronique** et d'un **environnement de programmation**.
- Sans tout connaître ni tout comprendre de l'électronique, cet environnement matériel et logiciel permet à l'utilisateur de formuler ses projets par l'expérimentation directe avec l'aide de nombreuses ressources disponibles en ligne.

VOUS AVEZ DIT ARDUINO ?

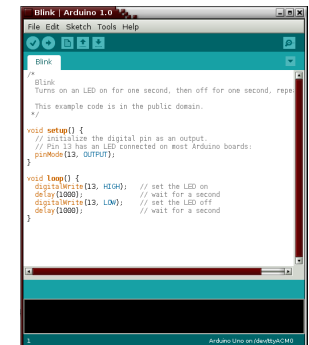
- Pont tendu entre le **monde réel** et le **monde numérique**, **Arduino** permet d'**étendre** les capacités de relations **humain/machine** ou **environnement/machine**.
- **Arduino** est un projet en **source ouverte (open source)** : la communauté importante d'utilisateurs et de concepteurs permet à chacun de trouver les réponses à ses questions.

Arduino en résumé

Une carte électronique



Un environnement de programmation

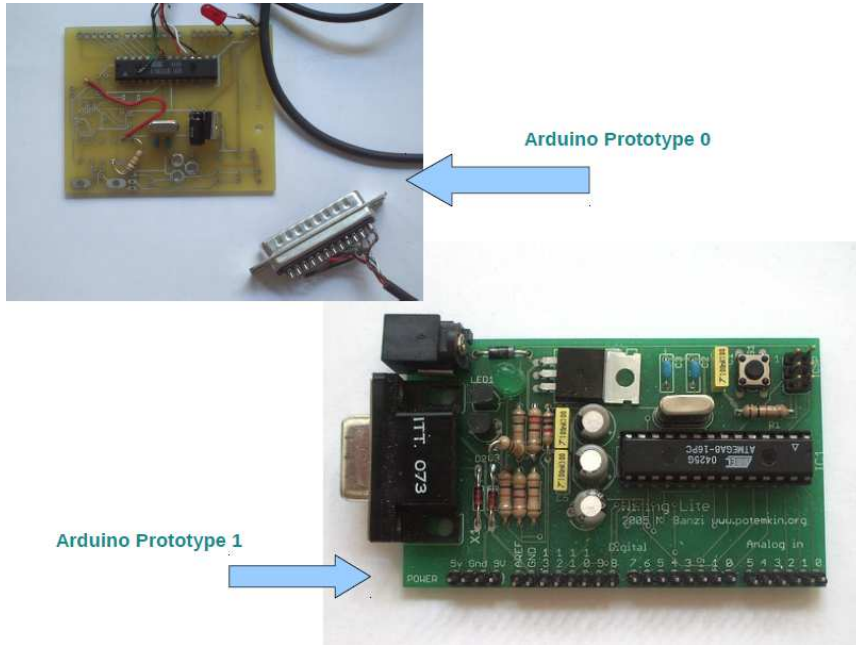


Une communauté qui échange
<http://arduino.cc/>



Photo by the Arduino Team

Historique



Leur objectif : Processing pour le Hardware !

- Qu'est ce que **Processing** ?
 - un langage de programmation et un environnement de développement créé par Benjamin Fry et Casey Reas, deux artistes américains.
 - particulièrement adapté à la **création plastique et graphique** interactive
 - Le logiciel fonctionne sur Macintosh, sous Windows et sous Linux, car il est basé sur la plate-forme Java — il permet d'ailleurs de programmer directement en langage Java.
- Pourquoi ?
 - Matériel robotique excessivement cher



Les créateurs : des artistes au sens premier du terme

Interaction Design Institute Ivrea : Team Arduino

- > David Mellis
- > Tom Igoe
- > Gianluca Martino
- > David Cuartielles
- > Massimo Banzi



Arduino : une philosophie

- Le matériel est « open source » :  
 - On peut le copier, le fabriquer et le modifier librement.



- Le logiciel est libre :
 - On peut l'utiliser et le modifier librement.



- Sur l'Internet, on trouve :
 - Une communauté d'utilisateurs.
 - Des guides d'utilisation.
 - Des exemples.
 - Des forums d'entraide.



Avantages

- **Pas cher !**
- Environnement de programmation clair et **simple**.
- **Multiplate-forme** : tourne sous Windows, Macintosh et Linux.
- Nombreuses bibliothèques disponibles avec diverses fonctions implémentées.
- Logiciel et matériel open source et extensible.
- Nombreux conseils, tutoriaux et exemples en ligne (forums, site perso, etc.)
- Existence de « shield » (boucliers en français)

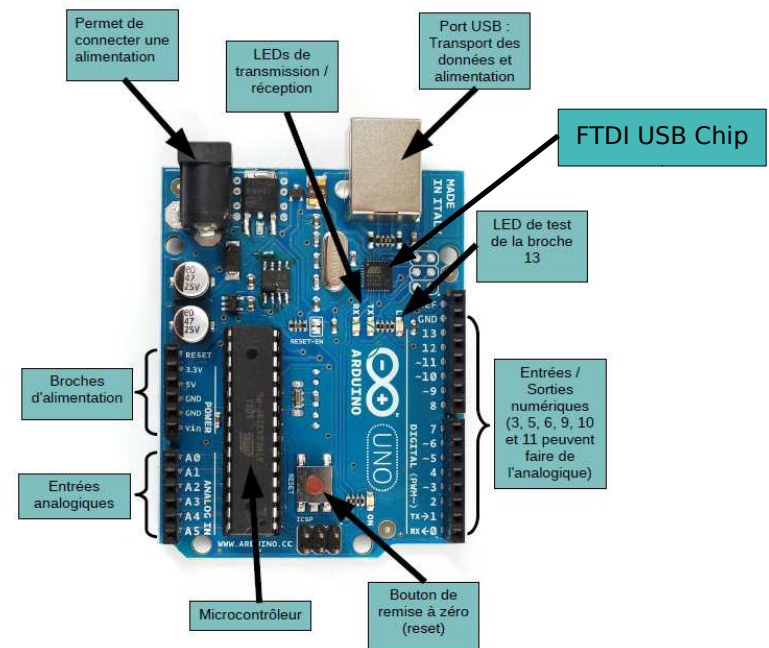
Domaine d'utilisation

- **Physical computing** : Au sens large, **construire des systèmes physiques interactifs** qui utilisent **des logiciels et du matériel** pouvant s'interfacer avec **des capteurs et des actionneurs**.
- Électronique industrielle et embarquée
- Art / Spectacle
- Domotique
- Robotique
- Modélisme
- DIY (Do-It-Yourself), Hacker, Prototypage, Education, Etc.

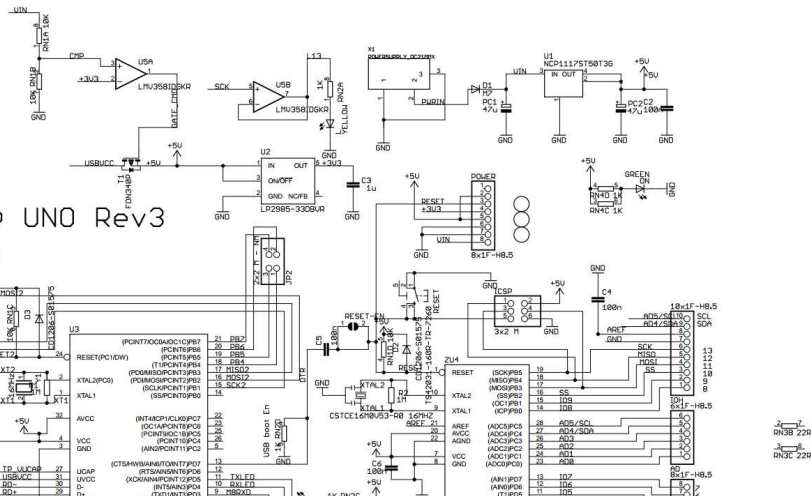
C'est quoi « pas cher » ?

- Prix d'une carte Arduino Uno = 25 euros
- Logiciel = 0 euros
- Support et assistance = 0 euros (forums)

La carte électronique Arduino



La schématique électronique

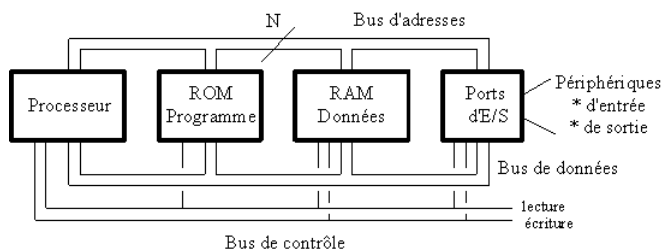


Qu'est ce qu'un microcontrôleur ?

- **μcontrôleur** : **circuit intégré qui rassemble les éléments essentiels d'un ordinateur** : processeur, mémoires (mémoire morte et/ou programmable pour le programme, mémoire vive pour les données), **unités périphériques et interfaces d'entrées-sorties**
- Ils sont fréquemment **utilisés dans les systèmes embarqués**, comme les contrôleurs des moteurs automobiles, les télécommandes, les appareils de bureau, l'électroménager, les jouets, la téléphonie mobile, etc.

Qu'est ce qu'un microcontrôleur ?

- Un microcontrôleur intègre sur un unique die (circuit intégré) :
- un **processeur** (CPU), avec une largeur du chemin de données allant de 4 bits pour les modèles les plus basiques à 32 ou 64 bits pour les modèles les plus évolués ;
- de la **mémoire vive** (RAM) pour stocker les données et variables ;
- de la **mémoire pour stocker le programme** : ROM (mémoire morte) et/ou EPROM, EEPROM, Flash ;
- souvent un oscillateur pour le cadencement. Il peut être réalisé avec un quartz, un circuit RC ou encore une PLL¹ ;



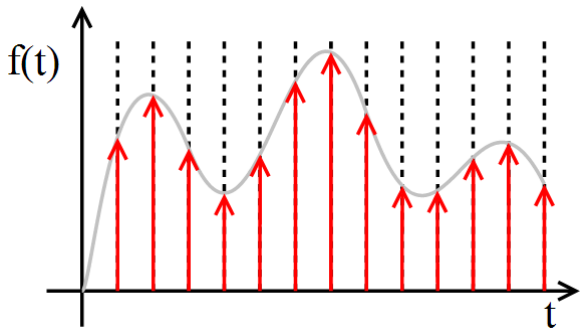
Qu'est ce qu'un microcontrôleur ?

- des périphériques, capables d'effectuer des tâches spécifiques. On peut mentionner entre autres :
 - les **convertisseurs analogiques-numériques** (CAN) (donnent un nombre binaire à partir d'une tension électrique),
 - les **convertisseurs numériques-analogiques** (CNA) (effectuent l'opération inverse),
 - les **générateurs de signaux à modulation de largeur d'impulsion** (MLI, ou en anglais, **PWM pour Pulse Width Modulation**),
 - les **timers/compteurs** (compteurs d'impulsions d'horloge interne ou d'événements externes),
 - les chiens de garde (watchdog),
 - les comparateurs (comparent deux tensions électriques),
 - les contrôleurs de bus de communication (UART, I²C, SSP, CAN, FlexRay, USB, Ethernet, etc.).

CAN = ADC (Analog-to-Digital Converter)
 CNA = DAC (Digital-to-Analog Converter)
 liaison série = UART (Universal Asynchronous Receiver Transmitter)

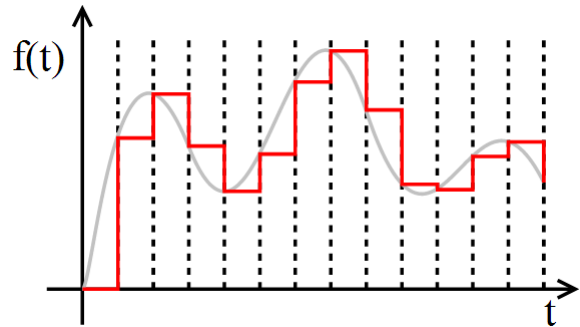
1 : Phase-Locked Loop ou boucle à phase asservie, ou encore boucle à verrouillage de phase

Analogique vers numérique



Résolution : le nombre de niveaux de sortie que l'ADC peut reproduire

Fréquence d'échantillonnage : le nombre de mesures par unité de temps



Par exemple avec une résolution de 8 bits sur un signal variant entre 0V et 5V, le nombre de niveaux est de 2^8 et donc la résolution en volt est de $(5-0)/(2^8)$ soit environ 19,53125 mV

En pratique, on considère que les bits de poids faibles ne sont pas assez précis. Si on omet 2 bits, on arrive à 78,125mV

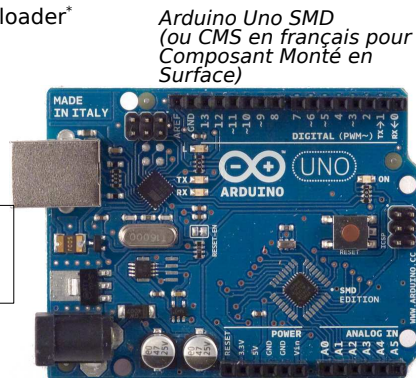
Numérisation

- Prenons un exemple pratique:
 - une fréquence d'échantillonnage de 10Kéch./s
 - une résolution de 10 bits (par échantillon)
 - une mémoire RAM de 2Ko
 - 10Kéch./s, soit 100Kbits/s soit 12,5Ko/s
 - Ce qui signifie qu'on ne peut récupérer que $2/12,5=160$ ms
- Les solutions ?
 - Réduire la fréquence d'échantillonnage si le phénomène à observer ne requière pas une telle précision temporelle
 - Réduire la résolution si une approximation plus grande est admise
 - Augmenter la mémoire :
 - si c'est une mémoire rapide, pas de problème
 - si c'est un support externe, plus lent, on va rater des événements (des mesures) le temps d'écrire.

Heureusement, pour nous, la température à observer variera lentement.

Notre Arduino : le Uno

- 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 (avec une résolution de 10 bits => 1024 valeurs différentes)
- Courant max par broches E/S = 40 mA
- Courant max sur sortie 3,3V = 50mA
- Mémoire Flash 32 KB dont 0.5 KB utilisée par le bootloader*
- Mémoire SRAM 2 KB
- mémoire EEPROM 1 KB
- Fréquence horloge = 16 MHz
- Dimensions = 68.6mm x 53.3mm



Arduino Uno SMD (ou CMS en français pour Composant Monté en Surface)

La carte s'interface au PC par l'intermédiaire de sa prise USB.
La carte s'alimente par le jack d'alimentation (utilisation autonome) mais peut être alimentée par l'USB (en phase de développement par exemple).

*Bootloader : un petit programme chargé sur le microcontrôleur. Il permet de charger le code sans programmeur. Il est activé quelques secondes lorsque la carte est « resetée ». Ensuite, il démarre le sketch (programme) qui a été chargé sur le microcontrôleur.

Des capteurs

	Mini Arduino Mercury Type TIR Sensor Module		Line Finder		Capteur infrarouge émetteur récepteur
	Smoke Sensor		Photorésistance (CdS)		Capteur Piezoelectrique
	Capteur de Poussieres - PPD42NS		Capteur de temperature - Grove		Capteur de mesure à ultrasons URM37 V3.2
	Capteur de qualité d'air - Grove		Capteur de temperature et d'humidité		Capteur de mouvement à infrarouge
	Détecteur d'alcool MQ303A		Thermographe infrarouge - Grove		Accéléromètre 3 axes +/-8g ADXL362
	Détecteur de gaz (MQ2)		TMP36 - Capteur de Temperature		Cyroscope 3 axes (Grove)
	Capteur de débit d'eau		Capteur Tactile - Grove		Module RFID 125kHz - UART
	Capteur de débit d'eau - G3/4		Capteur Touch		Module RFID 13.56MHz IOS/IEC 14443 type a

Encore des capteurs

	Capteur de poids - 50kg		Scanner code barre USB Série		Grove - RTC
	Capteur de son		GPS Bee kit (with Mini Embedded Antenna)		Grove - Serial Camera
	Carte magnétique LoCo		Grove - Barometer Sensor		Omnidirectional Sensor
	Compteur Geiger		Grove - Moisture Sensor		Grove - Electricity Sensor
	Lecteur - Encodeur de cartes magnétiques		Photo interrupter (OS25B10)		Grove - Hall Sensor

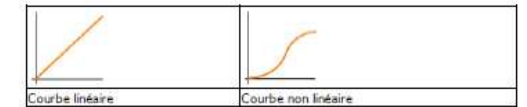
	Waterproof Metal Pushbutton with Blue LED Ring These chrome-plated metal buttons are rugged and waterproof and look real good while doing it! Simply drill a 16mm hole into any material up to 1/2".... ID : 481		Non-invasive AC current sensor (100A max)
	Rotary Encoder + Extras This rotary encoder is the best of the best, its a high quality 24-pulse encoder, with detents and a nice feel. It is panel mountable for placement... ID : 377 43 IN STOCK		Non-invasive AC current sensor (30A max)

Toujours des capteurs

	Extra-long force-sensitive resistor (FSR) FSRs are sensors that allow you to detect physical pressure, squeezing and weight. They are simple to use and low cost. This sensor is a Interlink... ID : 1071		pH Sensor Kit SEN-10972 Description: Need to measure precise pH? This kit includes everything you need including buffer solutions for calibration, pH probe, and even a board to connect directly to your favorite microcontroller. This is laboratory-grade equipment and not just a toy or educational kit.
	Color Sensor Module Description The Color sensor module base on TCS3200 which is a programmable color light-to-frequency converter, it could filter RGB data from source...		
	Programmable Color Light-to-Frequency Converter Module Description The Color sensor module base on TCS230 which is a programmable color light-to-frequency converter, it could filter RGB data from source...		
	Electronic brick - Diffuse reflection IR Switch sensor (Analog) Description This is a Diffuse reflection IR switch sensor module, use it to detect objects in front of the sensor. It comes with a transmitter and...		

ETC.

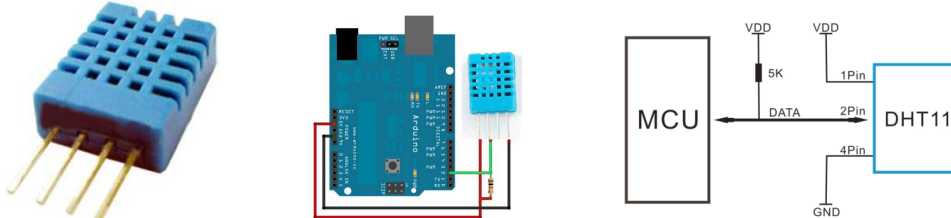
Attention au fonction régissant un capteur analogique après la conversion



Capteurs analogiques et numériques

- Parmi les capteurs que nous venons de voir, il existe :
 - des capteurs analogiques pour lesquels le signal devra être numérisé par le CAN du microcontrôleur. Il nous appartiendra de faire appliquer la loi régissant la mesure.
 - des capteurs numériques qui ont leur propre CAN embarqué. Il gère eux même la loi régissant la mesure. La communication avec ces capteurs se fait souvent selon un protocole particulier (I2C, 1-wire, etc.).

Par exemple le DHT11



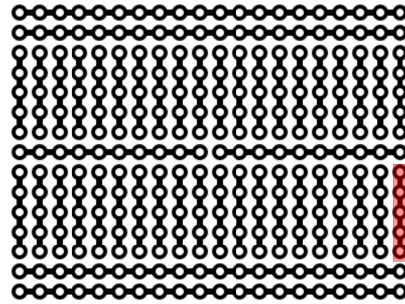
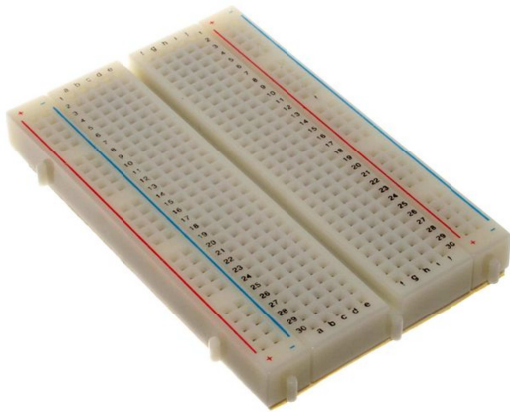
Quelques actionneurs

	Sirène d'alarme 110 dBA		Plateforme robot 2 roues motrices pour Arduino
	Module laser Ligne Rouge 5mW		Mini moteur à vibration 3V
	Module laser Point Rouge 5mW		Moteur pas à pas 12V 0.4A
	Module laser vert 5mW		Motoréducteur miniature
			Vibreur - Grove

	Plastic Water Solenoid Valve - 12V - 1/2 NPT Control the flow of fluid using the flow of electrons! This liquid valve would make a great addition to your robotic gardening project. There are two... ID : 997 IN STOCK		2 Channel 5V/12V/24V Relay Module Description This is a 2 channel relay module. It can be used to control the lighting, electrical and other equipments. The modular design makes it...
	Servo - Large ROB-09064 Description: Here is a simple, low-cost, high quality servo for all your mechatronic needs. Large servo with a standard 3 pin power and control cable. Includes hardware as shown. Weight: 41g Dimensions: 41 x 20 x 38mm		4 Channel 5V/12V/24V Relay Module Description This is a 4 channel relay module. It can be used to control the lighting, electrical and other equipments. The modular design makes it...

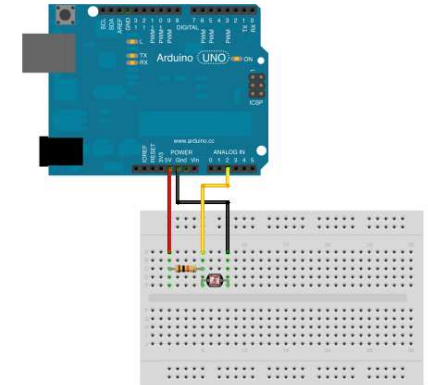
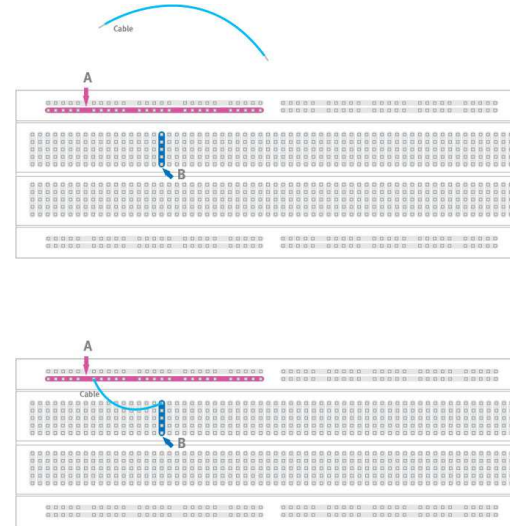
Breadboard (Planche à pain ... pour le prototypage)

Élément essentiel pour le prototypage et essai en tout genre



Les points reliés ensemble sont au même potentiel

Breadboard : exemple d'utilisation



1 photorésistance

Différents shields

- Existence de « shields » (boucliers en français) : ce sont des cartes supplémentaires qui se connectent sur le module Arduino pour augmenter les possibilités comme par exemple : afficheur graphique couleur, interface ethernet, GPS, etc...



Des shields



Arduino Ethernet Shield



Arduino Motor Shield



Shield GPRS



Arduino WiFi Shield



Arduino Wireless Proto Shield



2.8" TFT Touch Shield V2.0

Adafruit Wave Shield for Arduino Kit - v1.1



Arduino Wireless SD Shield



Arduino Proto Shield



CAN-BUS Shield

Go-Between Shield

Encore des shields

E-ink Display Shield



SD card shield



Solar Charger Shield V2

ETC.



RGB LED Shield V1.4 Kit



LoL Shield RED - A charlieplexed LED matrix kit for the Arduino
 The LoL Shield is a charlieplexed LED matrix for the Arduino. The LEDs are individually addressable, so you can use it to display anything in a 9 x...
 ID : 274
 20 IN STOCK



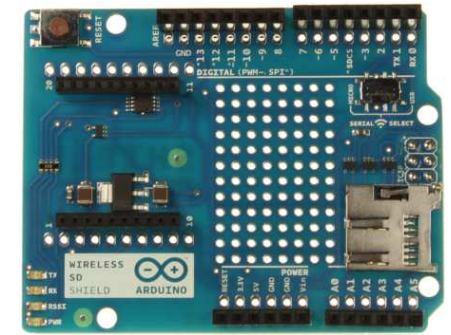
NFC Shield V2.0



LCD Shield Kit w/ 16x2 Character Display - Only 2 pins used!
 This new Adafruit shield makes it easy to use a 16x2 Character LCD. We really like the Blue & White 16x2 LCDs we stock in the shop. Unfortunately....
 ID : 772

Le shield Wireless SD

<http://arduino.cc/en/Main/ArduinoWirelessShield>



Communications sur le shield



Wifi Bee



Bluetooth Bee



XBee PCB Antenna - S1 (802.15.4)



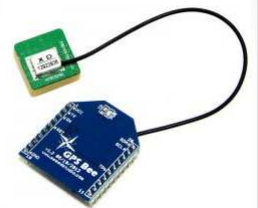
XBee Wi-Fi PCB Antenna - S6

En général on fonctionne sur une communication série (RX/TX). On utilisera la bibliothèque Serial ou SoftwareSerial sur Arduino.

C'est très simple !

Et ici c'est modulaire !

GPS Bee kit (with Mini Embedded Antenna)



Différents autres modules

JY-MCU Arduino Bluetooth Wireless Serial Port Module
 The **Bluetooth** wireless serial port module, Drop-in replacement for wired connections, transparent usage. You can use it. It is simply for a serial port replacemen
 free shipping
 ⭐⭐⭐ (77 reviews)

Wii WiiChuck Nunchuck Adapter for Arduino - Blue
 Model/AQuantity1ColorBlueMaterialPCBFeaturesUse this shield to achieve communication between nunchuck and ArduinoApplicationDIY projectPacking List1 x **Wii** wiiChuck adap
 free shipping
 (4 reviews)

Arduino Compatible NRF24L01 + PA + LNA V3.1 Wireless Module
 Model/AQuantity1ColorBlack + GoldenMaterialFR4FeaturesUnder 250K transmission rate, effective range: 1000m (open land); 1M transmission rate, 750m; 2M transmission rate, 520mApplicationArduino DIY pr
 free shipping

NRF905 Wireless Communication Transmission Module for Arduino
 Model/I061808Quantity1ColorGreenMaterialCCLFeaturesOperating frequency: 433/868/915MHz; Number of channel: 170; Modulation Mode: FSK/GMSK; Max output power: +10dBm; Sensitivity: -100dBm; Max operating
 free shipping

DIY 433MHz Wireless Receiving Module for Arduino - Green
 + PCBFeaturesOperating voltage: DC 5V; Quiescent current: 4mA; Modulation mode: OOK; Receiving sensitivity: -105dBm; Operating frequency: 433.90MHzApplicationVarious types of electronic products or DI
 free shipping

NRF2401B 2.4GHz Wireless RF Transceiver Module
 PCB - With RF2401B **wireless** communication module - 2.4GHz ISM frequency range - Up to 1Mbps working speed - 125 channels - High anti-jamming GFSK modulation - 8 / 16-bit
 free shipping
 > Arduino & SCM Supplies



315Mhz RF link kit



315Mhz RF link kits - with encoder and decoder



PLL FM(88-108MHz) Stereo Audio Receiver Module

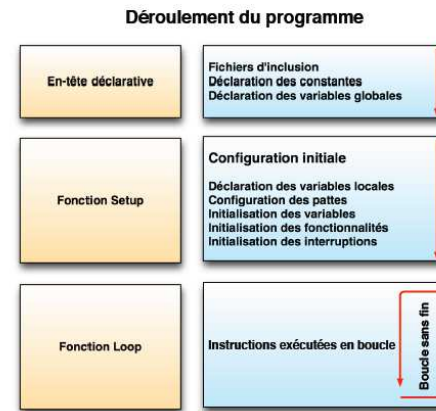


PLL FM(88-108MHz) Stereo Audio Transmitter Module

Programmmons notre Arduino

- Le langage Arduino est basé sur le C/C++.
 - Le langage de programmation d'Arduino est en effet une implémentation de Wiring (une plateforme open source similaire de *physical computing* qui proposait elle-même une bibliothèque appelée **Wiring** qui simplifie les opérations d'entrée/sortie).
- Un programme Arduino est aussi appelé un sketch.

Structure d'un programme



- Prise en compte des instructions de la partie déclarative
- Exécution de la partie configuration (fonction `setup()`),
- Exécution de la boucle sans fin (fonction `loop()`): le code compris dans la boucle sans fin est exécuté indéfiniment.

```
Finalemnt similaire au code suivant sur PC :  
int main()  
{  
    setup();  
    while(true)  
        loop();  
}
```

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.

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

```
void loop() //fonction principale qui se répète (s'exécute) à l'infini  
{  
    //contenu du programme  
}
```

La syntaxe du langage

- Voir aussi la section reference

<http://www.arduino.cc/reference>

Program structure

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

Statements

```
; {}  
// /* */
```

```
#define #include
```

Control structures

```
if, if...else, for, switch case, while, do... while  
break, continue, return, goto
```

Control structures

```
type func(type param, ...)
```


La syntaxe du langage

Data Types

void, boolean, char, unsigned char, byte,
int, unsigned int, word, long, unsigned long
float, double, string (char[])

Constructor

{}

Variables

local, global, static local, volatile, const, sizeof()

Operators

Arithmetic = + - * / %

Comparison == != < > <= >=

Boolean && || !

Bitwise & | ^ ~ « »

Pointer Access * &

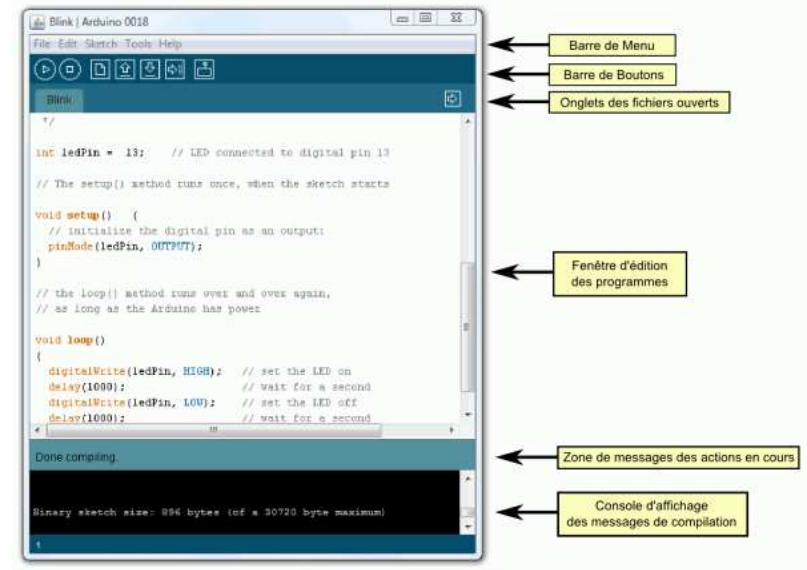
Compound == -- += -= *= /=&= |=

L'environnement de développement



- ✓ Vérifier (Verify) : vérifier les erreurs dans le code
- ➔ Charge (Upload) : compiler le code et charge le programme sur la carte Arduino
- 📄 Nouveau (New) : créer un nouveau sketch
- 📂 Ouvrir (Open) : ouvrir un des sketches déjà présent
- 💾 Sauvegarder (Save) : sauvegarder le sketch
- 🗨️ Serial Monitor : permet d'accéder au port série (en RX/TX)

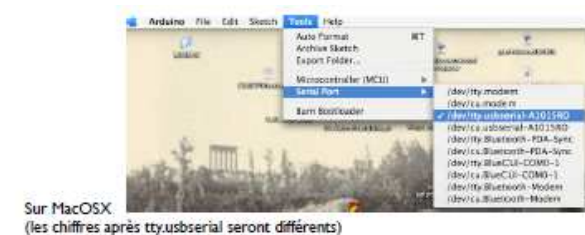
L'environnement de développement



Attention la barre d'outils n'est pas la même dans notre version !

Configuration de l'environnement de développement

- Pour un Arduino Uno dans Microcontroller (ou « type de carte » suivant les versions d'IDE)
- Désigner le bon port Série



Si vous changez de carte arduino, il faudra ré-indiquer le bon port.

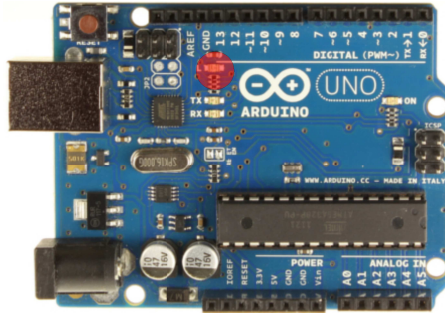


Un premier exemple : la led clignotante sur la carte

```
/*
Objectif: faire clignoter la LED montée sur la carte (et reliée à la patte 13)
*/
```

```
void setup()
{
  // Initialise la patte 13 comme sortie
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH); // allume la LED
  delay(500); // attend 500ms
  digitalWrite(13, LOW); // éteint la LED
  delay(500); // attend 500ms
}
```

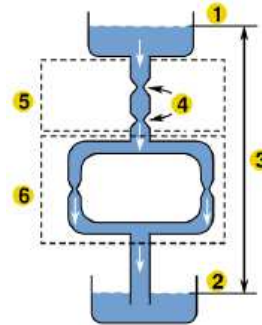


Quelques explications

- La ligne `pinMode(13, OUTPUT);` initialise la patte 13 du microcontrôleur comme sortie, c'est-à-dire que des données seront envoyées depuis le microcontrôleur vers cette patte.
- Avec l'instruction `digitalWrite(13, HIGH);`, le microcontrôleur connecte la patte D13 au +5V ce qui a pour effet d'allumer la LED.
- L'instruction `delay(500);` indique au microcontrôleur de ne rien faire pendant 500 millisecondes, soit 1/2 seconde.
- Avec l'instruction `digitalWrite(13, LOW);`, le microcontrôleur connecte la patte D13 à la masse (Gnd) ce qui a pour effet d'éteindre la LED.
- L'instruction `delay(500);` indique au microcontrôleur à nouveau de ne rien faire pendant 500ms soit 1/2 seconde.

Notions électriques fondamentales

- La **tension** est la différence de potentiel (Volts, V)
- Le **courant** (Ampères, A)
 - équivalent du débit
- La **résistance** (Ohms, Ω)
 - Équivalent du rétrécissement
 - $U=R*I$
- **AC/DC**
 - Direct Current (Courant Continu) (en électronique)
 - Alternative Current (Courant Alternatif)
 - courant qui change de direction continuellement
 - peut être périodique, c-à-d que sa fréquence est constante.
 - Souvent sinusoïdal, il est caractérisé par sa fréquence notée f et exprimée en Hertz, qui correspond au nombre d'aller-retour par seconde



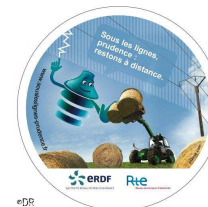
D'où

- DANGER



- Pas de danger pour nous avec la basse tension

Même une immense retenue d'eau peu profonde qui se déverserait à grand débit n'est pas un problème



Limites électroniques

- Nous allons confronter la théorie à l'épreuve de la réalité

La carte Arduino est une petite chose fragile, il convient de bien en comprendre ses limites d'utilisation et vérifier la compatibilité avec les matériels que l'on souhaite y raccorder...

Tension :

Le microcontrôleur placé sur la carte est prévu pour fonctionner entre 3,3V et 5V.

Intensité :

Le courant de sortie de chaque broche (D0 à D13) ne doit pas dépasser 40mA

Le courant issu du port USB ne doit pas dépasser 500mA

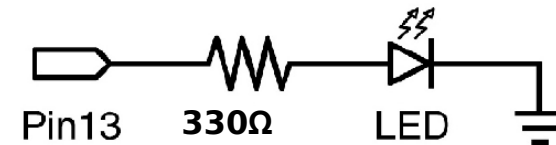
Le courant soutiré à la broche « 3,3V » ne doit pas dépasser 50mA

Les différents niveaux de protections

- Protection du port USB
- Protection de la carte Arduino
 - Protection des entrées numériques
- Protection des composants

Un second exemple : clignotement d'une led externe

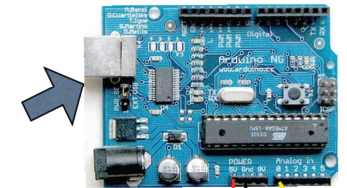
- **Attention**, on ne branche pas n'importe quoi n'importe comment.



- **Pas** de moteur/lampe, de composants nécessitant de la puissance directement connecté à un port d'entrée/sortie qui est **en mode sortie** !

Protection du port USB

- Tout d'abord, il faut savoir que le port USB de votre ordinateur délivre une quantité limitée de courant électrique. En général un port USB peut fournir au maximum 500 mA en 5 V.
- Si vous souhaitez réaliser un montage qui nécessite plus de courant, il s'agira de prévoir une alimentation externe suffisamment puissante pour répondre à votre besoin.



- **Conseils :**
 - Pour éviter qu'un fil ou qu'un composant branché au + vienne endommager un port USB dans l'ordinateur, **isoler le métal du port USB** avec un adhésif d'électricien
 - Attention également au dessous de la carte, à ne pas poser sur un support conducteur.

Protection de la carte Arduino

- Sachez que chacune des entrées/sorties de la carte ne peut pas délivrer plus de 40 mA (pour nos cartes Arduino Uno – voir les spécifications sur <http://arduino.cc/en/Main/ArduinoBoardUno>).
- Cette quantité de courant est relativement faible mais **permet**, par exemple, **de contrôler** une diode électroluminescente DEL (ou LED en anglais) ainsi que des actionneurs de faible puissance tel qu'un piézoélectrique ou encore un petit servomoteur.

La protection des entrées numériques sera vue plus loin.

Circuits de commande et de puissance

- Lorsqu'un système comporte des moteurs ou des **actionneurs demandant un certain niveau de puissance** que le micro-contrôleur ne peut pas fournir, **il faut avoir recours à deux circuits distincts interconnectés**, appelés « circuit de commande » et « circuit de puissance ».
- Suivant la puissance présente, les interconnexions entre les deux circuits nécessiteront des précautions particulières.
- Il est important de noter que ces deux circuits sont montés distinctement et sont isolés l'un de l'autre.
 - Toutefois, **il est primordial de lier leur masse** (« Gnd ») afin qu'ils partagent le même potentiel de référence.

Protection des composants

- Chaque composant possède ses propres conventions d'utilisation.
- Par exemple, il existe des composants qui possèdent un sens de branchement à respecter. On dit que ces composants sont polarisés.
 - C'est le cas des LEDs, de certains condensateurs, des diodes, etc.
- La plupart des composants ne peuvent pas fonctionner seuls, par exemple **une LED a besoin d'une résistance appropriée pour ne pas s'user ou « brûler »**. Cette résistance permet de limiter le courant qui traverse la LED. Le courant supprimé est alors dissipé en chaleur par la résistance (« Loi d'Ohm »).

Circuit de commande

- C'est dans ce circuit que sont rassemblés tous les éléments de contrôle comme les boutons, les interfaces et le micro-contrôleur.
 - Il est alimenté en basse tension : moins de 50V, souvent 12V, ou avec la carte Arduino 5V.
- On pourrait l'assimiler au système nerveux d'un organisme : c'est ici que se prennent les décisions mais peu d'énergie y circule.

Circuit de commande

- La manière la plus simple de relayer les commandes émergeant de ce circuit **pour les transmettre au circuit de puissance** est d'**utiliser des transistors ou encore des relais**.
- Lorsque les tensions d'alimentation des deux circuits sont plus importantes ou **si l'on veut protéger la commande de retours accidentels de courant provenant de la puissance, des optocoupleurs** (plutôt que des transistors) **assurent une isolation galvanique** : l'information est transmise sous forme de lumière. Ainsi, les deux circuits sont complètement isolés électriquement.

Courts-circuits

- Votre carte **ne doit pas être posée sur un support conducteur** car elle possède sur son verso des zones nues qui ne doivent pas être mises en contact afin de ne pas court-circuiter les composants entre eux.
- Il faut aussi **ne jamais connecter directement le port noté « Gnd »** (pôle négatif) **avec la broche 5 V** (pôle positif).

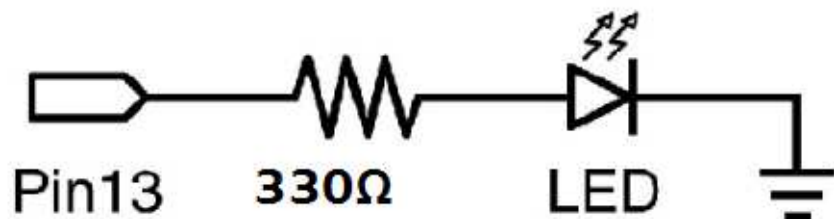
Circuit de puissance

- **Ce circuit alimente les composants nécessitant beaucoup d'énergie** (habituellement les moteurs et autres actionneurs). Sa tension d'alimentation dépend des composants en question.
- En fonction de cette tension et des conditions d'utilisation, les précautions à prendre sont variables :
 - dans tous les cas, une **protection contre les courts-circuits** est conseillée : fusible ou disjoncteur pour éviter de détruire les composants ;
 - **au dessus de 50 V, la tension peut menacer directement les humains** : protéger les pièces nues sous tension ;
 - le 230 V nécessite un interrupteur différentiel qui protège les humains des contacts accidentels (en général tout local est doté de ce dispositif). **Ne pas manipuler de 230 V sans connaissances appropriées.**

Quelques conseils (qu'on ne suivra pas tous ...)

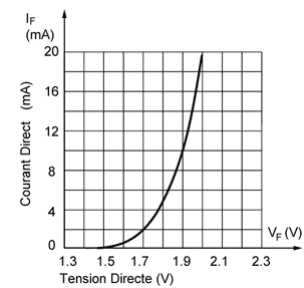
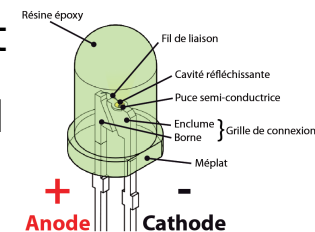
- Certains composants se ressemblent mais n'ont pas du tout la même fonction : toujours bien regarder leur signalétique.
- **Ne pas manipuler de 240V ou 110 V sans connaissances appropriées.**
- **Préférer faire les essais et les montages avec une alimentation externe plutôt que celle de l'USB (hummm)**, ce qui évitera de griller votre port USB de l'ordinateur (très courant)
- **À NE PAS FAIRE** : 10 façons de tuer son Arduino
 - <http://ruggedcircuits.com/html/ancp01.html>

Retour sur le second exemple : clignotement d'une led externe



Led

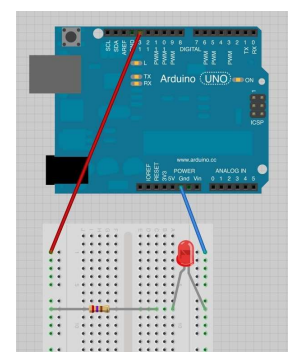
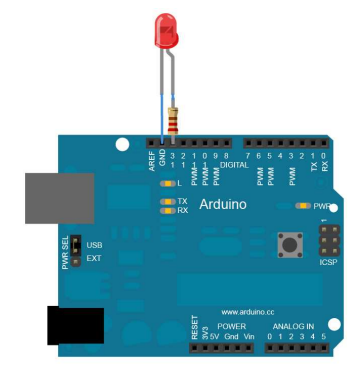
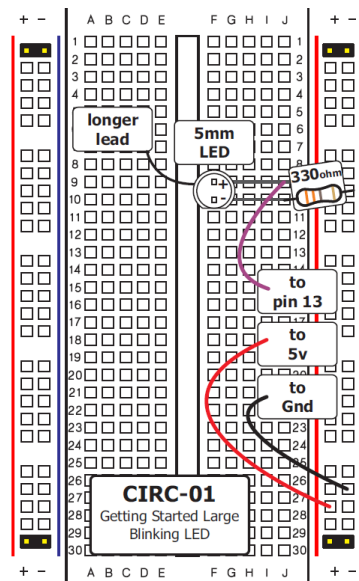
- Une diode électroluminescente (DEL ou LED) est un composant optoélectronique capable d'émettre de la lumière lorsqu'il est parcouru par un courant électrique.
- Une diode électroluminescente ne laisse passer le courant électrique que dans un seul sens (le sens passant).
- lorsqu'elle est traversée par un courant, la LED oppose une tension fixe
- L'intensité admissible dans la LED est aussi limitée (pour les LED ordinaires de 5mm, 24mA environ)



Résistance

Chiffres significatifs	Multiplicateur	Tolérance
0	x 0,01 Ω	± 10 %
1	x 0,1 Ω	± 5 %
2	x 1 Ω	± 20 %
3	x 10 Ω	± 1 %
4	x 100 Ω	± 2 %
5	x 1 k Ω	
6	x 10 k Ω	± 5 %
7	x 100 k Ω	± 0,5 %
8	x 1 M Ω	± 0,25 %
9	x 10 M Ω	± 0,1 %
		± 0,05 %
		± 20 %

Le circuit du second exemple



Le code du second exemple

Commentaires

Toujours écrire des commentaires sur le programme: soit en multiligne, en écrivant entre des `/*...*/`, soit sur une ligne de code en se séparant du code avec `//`

Définition des variables:

Pour notre montage, on va utiliser une sortie numérique de la carte, qui est par exemple la 13^{ème} sortie numérique. Cette variable doit être définie et nommée ici: on lui donne un nom arbitraire `BrocheLED`. Le mot de la syntaxe est pour désigner un nombre entier est `int`

Configuration des entrées-sorties `void setup():`

Les broches numériques de l'Arduino peuvent aussi bien être configurées en entrées numériques ou en sorties numériques. Ici on va configurer `BrocheLED` en sortie. `pinMode (nom, état)` est une des quatre fonctions relatives aux entrées-sorties numériques.

Programmation des interactions `void loop():`

Dans cette boucle, on définit les opérations à effectuer, dans l'ordre:

- `digitalWrite (nom, état)` est une autre des quatre fonctions relatives aux entrées-sorties numériques.
- `delay(temps en millisecondes)` est la commande d'attente entre deux autres instruction
- Chaque ligne d'instruction est terminée par un point virgule
- Ne pas oublier les accolades, qui encadrent la boucle.

(Syntaxe en marron, paramètres utilisateur en vert)

```
/* Ce programme fait clignoter une LED branchée sur la broche 13
 * et fait également clignoter la diode de test de la carte
 */

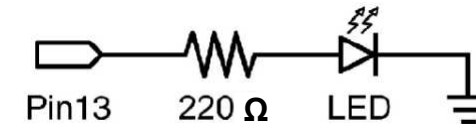
int BrocheLED = 13; // Définition de la valeur 13 et du nom de la broche à
utiliser

void setup()
{
  pinMode(BrocheLED, OUTPUT); // configure BrocheLED comme une
sortie
}

void loop()
{
  digitalWrite(BrocheLED, HIGH); // met la sortie num. à l'état haut (led
allumée)
  delay(3000); // attente de 3 secondes
  digitalWrite(BrocheLED, LOW); // met la sortie num. à l'état bas (led
éteinte)
  delay(1000); // attente de 1 seconde
}
```

Le second exemple en changeant la résistance

- Que se passe-t-il si on réduit la résistance ?



Vous le verrez en TP !

Serial (ou comment lire et écrire)

- La librairie **Serial** est une librairie essentielle du langage Arduino qui permet de visualiser sur le PC des messages reçus depuis la carte Arduino ou de commander la carte Arduino.
- En couplant l'utilisation de cette librairie avec l'interface programmable graphique **Processing** côté PC, on dispose d'un outil extrêmement puissant pour réaliser toute sortes d'affichages graphiques sur le PC ou d'interactions entre la carte et le PC (commande de la carte Arduino avec la souris ou le clavier !).

Serial

- Elle est utilisée **pour les communications par le port série** entre la carte Arduino et un ordinateur ou d'autres composants. Toutes les cartes Arduino ont au moins un port Série (également désigné sous le nom de UART ou USART) : **Serial**. **Ce port série communique sur les broches 0 (RX) et 1 (TX) avec l'ordinateur via le port USB.**
 - **C'est pourquoi, si vous utilisez cette fonctionnalité, vous ne pouvez utiliser les broches 0 et 1 en tant qu'entrées ou sorties numériques.**
- Vous pouvez utiliser le terminal série intégré à l'environnement Arduino pour communiquer avec une carte Arduino. Il suffit pour cela de **cliquer sur le bouton du moniteur série** dans la barre d'outils puis de **sélectionner le même débit** de communication que celui utilisé dans l'appel de la fonction `begin()`.
- La carte Arduino Mega dispose de trois ports série supplémentaires : **Serial1** sur les broches 19 (RX) et 18 (TX), **Serial2** sur les broches 17 (RX) et 16 (TX), **Serial3** sur les broches 15 (RX) et 14 (TX).

Serial

- Les fonctions de la librairie

- begin()
- available()
- read()
- flush()
- print()
- println()
- write()

Le baud

- Le **baud** est une unité de mesure utilisée dans le domaine des télécommunications en général, et dans le domaine informatique en particulier. Le baud est l'unité de mesure du **nombre de symboles transmissibles par seconde**.
- Le terme « baud » provient du patronyme d'Émile Baudot, l'inventeur du code Baudot utilisé en télégraphie.
- Il ne faut pas confondre le **baud** avec le **bps** (bit par seconde), ce dernier étant l'unité de mesure du nombre d'information effectivement transmise par seconde. Il est en effet souvent possible de transmettre **plusieurs bits par symbole**. La mesure en bps de la vitesse de transmission est alors supérieure à la mesure en baud.

Serial.begin()

- Description
 - **Fixe le débit de communication en nombre de caractères par seconde** (l'unité est le baud) pour la communication série.
 - Pour communiquer avec l'ordinateur, utiliser l'un de ces débits : 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. Vous pouvez, également, spécifier d'autres débits - par exemple, pour communiquer sur les broches 0 et 1 avec un composant qui nécessite un débit particulier.
- Syntaxe
 - Serial.begin(debit);
- Paramètres
 - int debit: débit de communication en caractères par seconde (ou baud). Pour communiquer avec l'ordinateur, utiliser l'un de ces débits : 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200.
- En pratique utiliser une valeur comprise entre 9600 et 115200. Typiquement 115200 fonctionne très bien ! (Plus le débit est élevé et plus la communication est rapide...)
 - **En cas d'utilisation du terminal série, IL FAUDRA FIXER LE MEME DEBIT dans la fenêtre du Terminal !**

Serial.available()

- Description
 - **Donne le nombre d'octets (caractères) disponible pour lecture** dans la file d'attente (buffer) du port série.
- Syntaxe
 - Serial.available();
- Valeur renvoyée
 - Le nombre d'octet disponible pour lecture dans la file d'attente (buffer) du port série, ou 0 si aucun caractère n'est disponible. Si une donnée est arrivée, Serial.available() sera supérieur à 0. La file d'attente du buffer peut recevoir jusqu'à 128 octets.

Serial.read()

- Description
 - Lit les données entrantes sur le port Série.
- Syntaxe
 - Serial.read();
- Valeur renvoyée
 - Renvoi le premier octet de donnée entrant disponible dans le buffer du port série, ou -1 si aucune donnée n'est disponible. (int)

Serial.print()

- Description
 - Affiche les données sur le port série sous forme lisible pour les humains (texte ASCII). Cette instruction peut prendre plusieurs formes.
 - Les nombres entiers sont affichés en utilisant les caractères ASCII pour chaque chiffre.
 - Les nombres à virgules (float) sont affichés de la même façon sous forme de caractères ASCII pour chaque chiffre, par défaut avec 2 décimales derrière la virgule.
 - Les valeurs de type byte sont affichés sous la forme d'un caractère ASCII.
 - Les caractères et les chaînes sont affichés tels quels.
- Syntaxe
 - Serial.print(val) ou Serial.print(val, format)
- Paramètre
 - val: la valeur à afficher. N'importe quel type de données.
 - format (optionnel) : spécifie
 - la base utilisée pour les nombres entiers
 - Les valeurs autorisées sont BIN (binaire, ou base 2), OCT (octal, ou base 8), DEC (décimal, ou base 10), HEX (hexadécimal, ou base 16) et plus ~~BYTE~~
 - le nombre de décimales (pour les nombres de type float)
- Valeur renvoyée
 - Un size_t (long) représentant le nombre d'octets écrits (même si il est optionnel de récupérer cette valeur)

Serial.write()

- Description
 - Écrit des données binaires sur le port série. Ces données sont envoyées comme une série d'octet; pour envoyer les caractères correspondants aux chiffres d'un nombre, utiliser plutôt la fonction print().
- Syntaxe
 - serial.write(val)
 - serial.write(str)
 - serial.write(buf, len)
- Paramètres
 - val: une valeur à envoyer sous forme d'octet simple
 - str: une chaîne à envoyer sous forme d'une série d'octets
 - buf: un tableau pour envoyer une série d'octets
 - len: la largeur du tableau
- Valeur renvoyée
 - Un octet représentant le nombre d'octets écrits (même si il est optionnel de récupérer cette valeur)

Exemples

```
Serial.print(78); // affiche "78"
Serial.print(1.23456); // affiche "1.23"
Serial.print(byte(78)); // affiche "N" (dont la valeur ASCII est 78)
Serial.print('N'); // affiche "N"
Serial.print("Hello world."); // affiche "Hello world."

Serial.print(78, BYTE); // N'est plus possible depuis la version 1.0
// Il faut utiliser Serial.write(78) qui affiche "N"

Serial.print(78, BIN); // affiche "1001110"
Serial.print(78, OCT); // affiche "116"
Serial.print(78, DEC); // affiche "78"
Serial.print(78, HEX); // affiche "4E"
Serial.print(1.23456, 0); // affiche "1"
Serial.print(1.23456, 2); // affiche "1.23"
Serial.print(1.23456, 4); // affiche "1.2346"
```

Serial.println()

- Description
 - Affiche les données sur le port série suivi d'un caractère de "retour de chariot" (ASCII 13, or '\r') et un caractère de "nouvelle ligne" (ASCII 10, or '\n'). Cette instruction a par ailleurs la même forme que l'instruction Serial.print()
- Syntaxe
 - Serial.println(val) ou Serial.println(val, format)
- Paramètres
 - val: la valeur à afficher. N'importe quel type de données.
 - format (optionnel) : spécifie
 - la base utilisée pour les nombres entiers
 - Les valeurs autorisées sont BIN (binaire, ou base 2), OCT (octal, ou base 8), DEC (décimal, ou base 10), HEX (hexadécimal, ou base 16) et plus BYTE
 - le nombre de décimales (pour les nombres de type float)
- Valeur renvoyée
 - Un size_t (long) représentant le nombre d'octets écrits (même si il est optionnel de récupérer cette valeur)

Exemple

```
/*
 * Lit une valeur analogique en entrée sur analog 0 et affiche la valeur.
 */

int analogValue = 0; // variable pour stocker la valeur analogique

void setup() {
  // initialise le port série à 9600 bauds
  Serial.begin(9600);
}

void loop() {
  // lit la valeur analogique sur la broche analog 0
  analogValue = analogRead(0);

  // affiche cette valeur dans différents formats
  Serial.println(analogValue); // affichage décimal
  Serial.println(analogValue, DEC); // affichage décimal
  Serial.println(analogValue, HEX); // affichage hexadécimal
  Serial.println(analogValue, OCT); // affichage octal
  Serial.println(analogValue, BIN); // affichage binaire
  Serial.write(analogValue); // affichage octet simple
  Serial.println(); // affichage retour à la ligne

  // pause de 10 millisecondes avant la mesure suivante
  delay(10);
}
```

Serial.print() et Serial.println()

- Un problème majeur que l'on va rencontrer avec les chaînes de caractères, c'est l'utilisation de la RAM qui va rapidement saturer (2K pour une UNO contre 32K de mémoire programme) et donner des comportements inattendus. Pour contourner le problème :
 - il faut mettre les chaînes de caractères en mémoire programme, ce qui peut se faire simplement à l'aide de la librairie flash
 - on peut évaluer le niveau d'utilisation de la RAM en intégrant une fonction spécifique dans le programme. Voir : <http://www.arduino.cc/playground/Code/AvailableMemory>

Exemple de lecture et d'affichage sur le port série

```
int incomingByte = 0; // variable pour lecture de l'octet entrant

void setup() {
  Serial.begin(9600); // ouvre le port série et fixe le débit à 9600 bauds
}

void loop() {

  // envoie une donnée sur le port série seulement quand reçoit une donnée
  if (Serial.available() > 0) { // si données disponibles sur le port série
    // lit l'octet entrant
    incomingByte = Serial.read();

    // renvoie l'octet reçu
    Serial.print("Octet reçu: ");
    Serial.println(incomingByte, DEC);
  }
}
```

Serial.flush()

- Description
 - Cette fonction s'assure que les données écrites sur le port série aient été physiquement envoyés avant de rendre la main (la fonction est donc bloquante).
- Syntaxe
 - Serial.flush();
- Ancienne description
 - Vide le buffer de réception de données du port série. Par conséquent, tout appel de la fonction Serial.read() ou Serial.available() renverra seulement les données reçues après le plus récent appel de la fonction Serial.flush().

Mais ça c'était avant !

Bibliothèque pour la carte SD

- Cette librairie est très intéressante et ouvre des **possibilités de stockage de données** impressionnantes puisque les cartes SD disposent de 4 à 8 Go d'espace.
- De plus, les cartes SD une fois retirée de leur emplacement sur le shield Arduino, elles peuvent être **lues directement sur un PC**. En enregistrant les données dans un **fichier texte au format CSV** par exemple, on rend possible une **utilisation immédiate des données dans un tableur** avec réalisation de graphique, etc. Que du bon, et même du très bon !
- A noter que la librairie sdfatlib donne accès à davantage de fonctions, notamment des fonctions d'accès aux informations de la carte SD, ce qui peut être intéressant dans certains cas.
- Pour l'utiliser, **il suffira de l'inclure** et d'utiliser les sous programmes qu'elle propose :

```
#include <SD.h>
```

Quelques autres fonctions de Serial

- if (Serial)
- end()
- find()
- findUntil()
- parseFloat()
- parseInt()
- peek()
- readBytes()
- readBytesUntil()
- setTimeout()
- serialEvent()

Bibliothèque pour la carte SD

- INFORMATIONS SUR LES FICHIERS SUPPORTÉS
 - La librairie SD permet de lire et d'écrire les cartes mémoires SD, par exemple sur le shield Wireless SD.
 - La librairie supporte les systèmes de fichier FAT16 et FAT 32 sur les **cartes mémoires SD standard et les cartes SDHC**.
 - La librairie supporte l'**ouverture de plusieurs fichiers à la fois** et n'utilise toujours que les noms de fichiers courts au format "8.3" (c'est à dire **des noms en 8 lettres maxi, un point et trois lettres**, par exemple filename.txt).
 - Les noms de fichiers utilisés avec les fonctions de la librairie SD peuvent inclure des chemins séparés par des slash /, par exemple "directory/filename.txt).
 - Le répertoire de travail étant toujours la racine de la carte SD, un nom se réfère au même fichier, qu'il inclue ou non un / (par exemple, "/file.txt" est équivalent à "file.txt").

Bibliothèque pour la carte SD

- INFORMATIONS MATÉRIELLES
 - La communication entre la carte Arduino et la carte mémoire SD utilise la **communication SPI**, laquelle utilise les broches **11, 12 et 13** de la plupart des cartes Arduino ou les broches 50, 51 et 52 sur la carte Arduino Mega (à noter que la carte d'extension Arduino Wireless SD assure la connexion pour la carte Mega via le connecteur droit ICSP* de la carte Arduino).
 - En plus, une autre broche peut être utilisée pour sélectionner la carte SD. Cette broche peut être la broche matérielle SS (broche **10** sur la plupart des cartes Arduino et broche 53 sur la Mega) ou une autre broche spécifiée dans la fonction d'initialisation **SD.begin()**.
 - **Noter que même si vous n'utilisez pas la broche matérielle SS, elle doit être laissée en sortie, sinon la librairie ne fonctionnera pas.**

*In Circuit Serial Programming

Bibliothèque pour la carte SD

- La classe `File` permet de lire et d'écrire dans les fichiers individuels sur la carte SD.

- `available()`
- `close()`
- `flush()`
- `peek()`
- `position()`
- `print()`
- `println()`
- `seek()`
- `size()`
- `read()`
- `write()`
- `isDirectory()`
- `openNextFile()`
- `rewindDirectory()`

Bibliothèque pour la carte SD

- La classe `SD` fournit les fonctions pour accéder à la carte SD et manipuler ses fichiers et répertoires.
- L'objet `SD` est directement disponible à l'inclusion de la librairie.
 - `begin()`
 - `exists()`
 - `mkdir()`
 - `open()`
 - `remove()`
 - `rmdir()`

SD.begin()

- Description
 - **Initialise la librairie SD et la carte SD.** Cela lance l'utilisation du bus SPI (broches numériques 11,12 et 13 sur la plupart des cartes Arduino) et initialise la broche de sélection du circuit intégré (chip select), qui est par défaut la broche matérielle SS (broche 10 sur la plupart des cartes Arduino).
 - Notez que même si vous utilisez une broche de sélection différente, la broche matérielle SS doit être laissée en SORTIE, sinon les fonctions de la librairie SD ne fonctionneront pas.

SD.begin()

- Syntaxe
 - SD.begin()
ou
 - SD.begin(cspin)
- Paramètres
 - SD : l'objet racine de la librairie SD
 - cspin (optionnel): la broche connectée à la broche de sélection "chip select" de la carte SD. Cette broche est la broche matérielle SS du bus SPI par défaut.
- Valeur renvoyée
 - true si réussite;
 - false si échec.

SD.begin() : exemple

```
#include <SD.h>

void setup()
{
  Serial.begin(115200); // utiliser le meme debit coté Terminal Serie
  Serial.println("Initialisation de la SD card...");

  pinMode(10, OUTPUT); // laisser la broche SS en sortie - obligatoire avec librairie SD

  if (!SD.begin(10)) { // si initialisation avec broche 10 en tant que CS n'est pas réussie
    Serial.println("Echec initialisation!"); // message port Série
    return; // sort de setup()
  }
  Serial.println("Initialisation reussie !"); // message port Série
}

void loop(){
}
```

SD.exists()

- Description
 - Cette fonction teste si un fichier ou un répertoire existe sur la carte mémoire SD.
- Syntaxe
 - SD.exists(filename)
- Paramètres
 - SD : l'objet racine de la librairie SD
 - filename : le nom du fichier dont il faut tester l'existence, lequel peut inclure des répertoires (délimités par les slash /).
- Valeur renvoyée
 - true : si le fichier ou répertoire existe
 - false : si le fichier ou le répertoire n'existe pas

SD.mkdir()

- Description
 - Cette fonction crée un répertoire sur la carte mémoire SD. Cette fonction créera également les répertoires intermédiaires qui n'existe pas encore; par exemple :
SD.mkdir("a/b/c");
- créera les répertoires a, puis b dans a et c dans b.
- Syntaxe
 - SD.mkdir(filename)
- Paramètres
 - SD : l'objet racine de la librairie SD
 - filename : le nom du répertoire ou chemin à créer
- chaîne de caractères
- Valeur renvoyée
 - true : si la création du chemin a réussi
 - false : si la création du chemin a échoué

SD.open()

- Description
 - Cette fonction ouvre un fichier sur la carte mémoire SD. Si le fichier est ouvert pour écriture, il sera créé si il n'existe pas déjà (cependant le chemin le contenant doit déjà exister).
- Syntaxe
 - SD.open(filename) ou SD.open(filename, mode)
- Paramètres
 - SD : l'objet racine de la librairie SD
 - filename : le nom du fichier à ouvrir, qui peut inclure le chemin (délimité par les slash /) - pointeur caractères char *
 - mode (optionnel): le mode d'ouverture du fichier, par défaut FILE_READ - byte. Les paramètres possibles sont :
 - FILE_READ: ouvre le fichier pour lecture, en démarrant au début du fichier.
 - FILE_WRITE: ouvre le fichier pour lecture et écriture, en démarrant à la fin du fichier.
- Valeur renvoyée
 - un objet File qui se réfère au fichier ouvert; si le fichier n'a pas pu être ouvert, cet objet sera évalué comme false dans une situation "booléenne", autrement dit, il est possible de tester la valeur renvoyée avec if (file).

SD.rmdir()

- Description
 - Cette fonction efface un répertoire (remove dir) de la carte mémoire SD. Le répertoire doit être vide.
- Syntaxe
 - SD.rmdir(filepath)
- Paramètres
 - SD : l'objet racine de la librairie SD
 - filepath : le nom du répertoire à effacer, incluant les sous-répertoires délimités par des slashes /.
- Valeur renvoyée
 - true : si le répertoire a bien été effacé
 - false : si le répertoire n'a pas pu être effacé.
 - Si le répertoire n'existe pas, la valeur renvoyée n'est pas spécifiée.

SD.remove()

- Description
 - Cette fonction efface un fichier de la carte mémoire SD.
- Syntaxe
 - SD.remove(filename)
- Paramètres
 - SD : l'objet racine de la librairie SD
 - filename : le nom du fichier, qui peut inclure le chemin délimité par des slashes /.
- Valeur renvoyée
 - true : si le fichier a bien été effacé
 - false : si le fichier n'a pas pu être effacé.
 - A noter que la valeur renvoyée n'est pas spécifiée si le fichier n'existe pas

Les fichiers

- On déclare une variable de type fichier :
 - File nomFile;
- On ouvre un fichier ainsi :
 - File f = SD.open("data.txt", FILE_WRITE);

file.close()

- Description
 - Cette fonction ferme le fichier, et s'assure que toute donnée écrite dans ce fichier est physiquement enregistrée sur la carte mémoire SD.
- Syntaxe
 - `file.close()`
- Paramètres
 - `file` : une instance de l'objet File (renvoyée par la fonction `SD.open()`)
- Valeur renvoyée
 - Aucune valeur

file.peek()

- Description
 - Cette fonction lit un octet dans un fichier sans avancer au suivant. Ainsi, des appels successifs de la fonction `peek()` renverront la même valeur, jusqu'au prochain appel de la fonction `read()`.
- Syntaxe
 - `file.peek()`
- Paramètres
 - `file` : une instance de l'objet File (renvoyée par la fonction `SD.open()`)
- Valeur renvoyée
 - l'octet suivant (ou caractère)
 - ou -1 si aucun n'est disponible

file.read()

- Description
 - Cette fonction lit un octet dans un fichier.
- Syntaxe
 - `file.read()`
- Paramètres
 - `file` : une instance de l'objet File (renvoyée par la fonction `SD.open()`)
- Valeur renvoyée
 - l'octet (ou caractère) suivant
 - -1 si rien n'est disponible

file.write()

- Description
 - Cette fonction écrit des données dans un fichier.
- Syntaxe
 - `file.write(data)`
 - `file.write(buf, len)`
- Paramètres
 - `file` : une instance de l'objet File (renvoyée par la fonction `SD.open()`)
 - `data` : l'octet(byte), le caractère(char) ou la chaîne de caractère (char *) à écrire dans le fichier
 - `len` : le nombre d'éléments contenus dans `buf`
- Valeur renvoyée
 - Un octet représentant le nombre d'octet écrits (même si il est optionnel de récupérer cette valeur)

• Description *file.print()*

- Cette fonction écrit des données dans un fichier, lequel doit être ouvert pour écriture. Cette fonction affiche les nombres comme une séquence de chiffres, chacun comme un caractère ASCII (par exemple le nombre 123 est écrit sous la forme de 3 caractères '1', '2', '3').
- Syntaxe
 - file.print(data)
 - file.print(data, BASE)
- Paramètres
 - file : une instance de l'objet File (renvoyée par la fonction SD.open())
 - data: les données à écrire dans le fichier - type possibles : char, byte, int, long, ou string
 - BASE (optionnel): la base dans laquelle écrire les nombres : BIN pour binaire (base 2), DEC pour décimal (base 10), OCT pour octal (base 8), HEX pour hexadécimal (base 16). La base par défaut est la base décimale.
- Valeur renvoyée
 - Un octet représentant le nombre d'octets écrits (même si il est optionnel de récupérer cette valeur)

file.flush()

- Description
 - Cette fonction s'assure que les données écrites dans un fichier ont été physiquement enregistrées sur la carte mémoire SD. Ceci est fait automatiquement également lorsque le fichier est fermé avec la fonction close()
- Syntaxe
 - file.flush()
- Paramètres
 - file : une instance de l'objet File (renvoyée par la fonction SD.open())

file.println()

Cette fonction est identique à *file.print()*, elle écrit des données **suivies** d'un **saut de ligne** (retour chariot + nouvelle ligne)

• Description *file.size()*

- Cette fonction permet de connaître la taille d'un fichier (en nombre d'octets).
- Syntaxe
 - file.size()
- Paramètres
 - file : une instance de l'objet File (renvoyée par la fonction SD.open())
- Valeur renvoyée
 - la taille du fichier en nombre d'octets - type unsigned long

file.available()

- Description
 - Cette fonction vérifie si des octets sont disponibles en lecture dans le fichier.
- Syntaxe
 - file.available()
- Paramètres
 - file : une instance de l'objet File (renvoyée par la fonction SD.open())
- Valeur renvoyée
 - le nombre d'octets disponibles - int

file.seek()

- Description
 - Cette fonction permet de se placer à une nouvelle position, qui doit être comprise entre 0 et la taille du fichier (inclusif).
- Syntaxe
 - file.seek(pos)
- Paramètres
 - file : une instance de l'objet File (renvoyée par la fonction SD.open())
 - pos : la position à laquelle se placer - type unsigned long
- Valeur renvoyée
 - true : en cas de réussite
 - false : en cas d'échec.

file.position()

- Description
 - Cette fonction renvoie la position courante à l'intérieur du fichier, c'est-à-dire la position à laquelle le prochain octet sera lu ou écrit).
- Syntaxe
 - file.position()
- Paramètres
 - file : une instance de l'objet File (renvoyée par la fonction SD.open())
- Valeur renvoyée
 - la valeur de la position à l'intérieur du fichier
 - type long

file.isDirectory()

- Description
 - Les répertoires (ou dossiers) sont une sorte particulière de fichiers : cette fonction permet de savoir si le fichier courant est un répertoire ou non.
- Syntaxe
 - file.isDirectory()
- Paramètres
 - file : une instance de l'objet File (renvoyée par la fonction SD.open())
- Valeur renvoyée
 - Renvoie un boolean : true si le fichier courant est un répertoire, false sinon.

file.openNextFile()

- Description
 - Renvoie le fichier ou dossier suivant dans un répertoire.
- Syntaxe
 - file.openNextFile()
- Paramètres
 - file : une instance de l'objet File (renvoyée par la fonction SD.open()) qui ici représente un répertoire
- Valeur renvoyée
 - char : le nom du fichier ou répertoire suivant.

#include <SD.h> Exemple : liste le contenu

```
File root;
void setup()
{
  Serial.begin(9600);
  pinMode(10, OUTPUT);
  SD.begin(10);
  root = SD.open("/");
  printDirectory(root, 0);
  Serial.println("done!");
}

void loop()
{
  // nothing happens after setup finishes.
}

void printDirectory(File dir, int numTabs) {
  while(true) {
    File entry = dir.openNextFile();
    if (! entry) {
      // si pas de nouveau fichier
      // renvoie le premier fichier dans le répertoire
      dir.rewindDirectory();
      break;
    }
    for (uint8_t i=0; i<numTabs; i++) {
      Serial.print('\t');
    }
    Serial.print(entry.name());
    if (entry.isDirectory()) {
      Serial.println("/");
      printDirectory(entry, numTabs+1);
    }
    else {
      // les fichiers ont une taille, pas les répertoires
      Serial.print("\t\t");
      Serial.println(entry.size(), DEC);
    }
  }
}
```

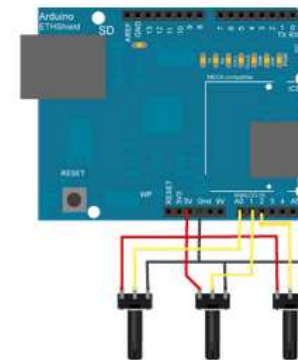
Un appel récursif

file.rewindDirectory()

- Description
 - Se place au niveau du premier fichier dans un répertoire. Utilisée en association avec openNextFile()
- Syntaxe
 - file.rewindDirectory()
- Paramètres
 - file : une instance de l'objet File (renvoyée par la fonction SD.open())

Un exemple de Datalogger sur SD

- 3 potentiomètres

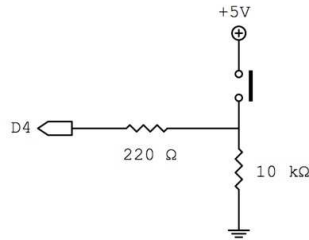


Résistances Pull-Up / Pull-Down

- Une **résistance de rappel**, aussi appelée résistance de tirage (en anglais pull-up resistor), est une résistance dans un circuit électronique, située entre la source d'alimentation et une ligne, et qui amène délibérément cette même ligne soit à l'état haut (1 logique) pour une résistance de rappel haut, soit à l'état bas (0 logique) pour une résistance de rappel bas.
- Et en clair ?

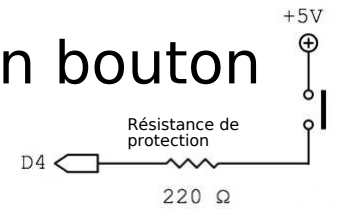
Avec une résistance pull-down

- Observons ce schéma :



- Ce montage contient une **résistance de 10 kΩ** (soit 10000 Ω), qu'on appelle **pull-down** (littéralement tirer-en-bas). Cette résistance permet de tirer le potentiel vers le bas (pulldown). En français, on appelle aussi ceci **un rappel au moins**.
- **Le but d'une résistance de pull-down est donc d'évacuer les courants vagabonds et de donner un signal clair.**

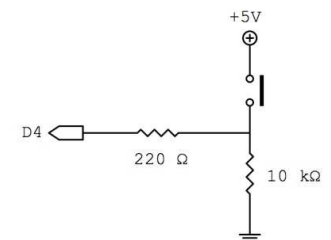
Observons l'état d'un bouton



- **Modèle théorique :**
 - Lorsque l'on presse le bouton poussoir, on envoie +5 Volts sur D4, qui sera interprété comme un 1.
 - Lorsqu'on relâche le bouton, il n'y a plus de courant qui arrive sur D4, ce qui sera interprété comme un 0. On a donc un signal binaire: allumé/éteint (on/off).
- **Pourtant contrairement au cas théorique**, fermer ou ouvrir un circuit (via un interrupteur ou un bouton poussoir) ne génère pas forcément un signal clair :
 - Le circuit non connecté à la source peut agir comme **une antenne** dans un environnement pollué d'ondes électromagnétiques (proches des moteurs par exemple). Cela va générer dans le circuit un courant qui **peut être interprété comme un signal**.
 - Il peut **rester** dans un circuit récemment ouvert **un niveau d'énergie**. Cela **provoque des cas d'indétermination**. Le signal qui a été coupé peut être considéré comme encore actif par un microcontrôleur.

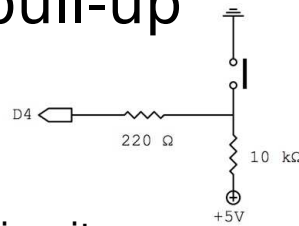
Avec une résistance pull-down

- Si on presse le bouton, 5 Volts sont alors appliqués sur l'entrée. Le courant va prendre le chemin le plus simple, soit par la résistance de 220 Ω et finit par arriver sur D4. Si on relâche le bouton, la résistance pull-down ramène l'entrée à la masse (puisque'elle est connectée à la masse).



Avec une résistance pull-up

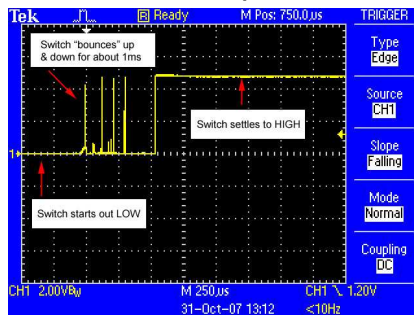
- Observons ce schéma :



- Si on le compare au schéma d'un circuit avec une résistance pull-down, on constate une **inversion entre la terre et le +5V**.
- En effet, lorsque le circuit est ouvert, une tension de +5V est appliquée à l'entrée D4.
- Lorsqu'on appuie sur le bouton poussoir, le courant électrique va passer par le chemin offrant le moins de résistance, soit directement par la masse (Gnd), sans passer par l'entrée D4.
- **Le fonctionnement est donc inversé par rapport à la résistance pull-down.**

On veut aller plus loin ?

- Lorsque l'on presse/relâche un bouton, il y a souvent l'apparition de parasites pendant quelques millisecondes. Ces parasites n'apparaissent que durant les moments où l'on enfonce/relâche le bouton poussoir.



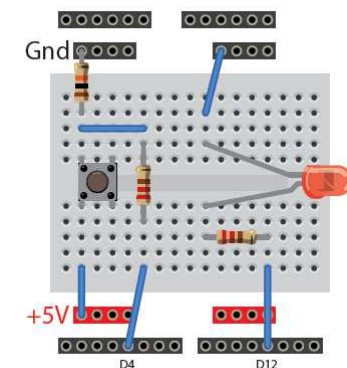
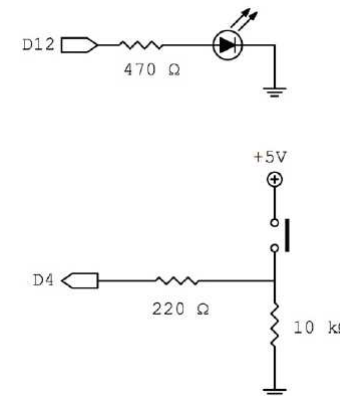
- Si l'on compte le nombre de pressions (pour faire un compteur), ces parasites viennent justement perturber le bon fonctionnement du logiciel. Le problème est matériel... et les parasites viennent ajouter des pressions fantômes.
- Il y a deux façon de corriger le problème :
 - Utiliser une capacité de déparasitage
 - Introduire un délai logiciel dans le programme.

Voir les références à la fin pour les détails

Résistance pull-down ou pull-up ?

- À notre niveau, la seule chose qui va changer entre une résistance pull-down et une résistance pull-up est la lecture de l'information:
- Avec une résistance pull-down, par défaut, l'entrée sur la patte est égale à 0.
- Avec une résistance pull-up, par défaut, l'entrée sur la patte est égale à 1.
- Dans le code `if (digitalRead(bouton) == 1)`, sera respectivement la valeur d'entrée lorsque le circuit est fermé (pull-down) ou ouvert (pull-up).
- Remarque :
 - La résistance est ici de 10KOhms, c'est une valeur assez commune permettant d'identifier facilement un circuit de pull-up/down.
 - Il est possible de diminuer cette résistances à 4.7 KOhms mais cela consommera aussi plus de courant.
 - Il n'est pas conseillé d'utiliser une résistance de pull-up de plus de 10KOhms. À partir de 100 KOhms, la résistance de pull-up/down interfère avec la circuiterie interne du micro-contrôleur et le résultat (la détection) deviendrait incertain.

Montage avec résistance pull-down



Indiquez sur le schéma ci-dessus où se trouve la résistance pull down et la résistance de protection du microcontrôleur

Allume la LED quand on presse

```
/*
 * Allume LED en fonction de l'état du bouton poussoir
 */
const int bouton = 4;    // la patte 4 devient bouton
const int led = 12;     // la patte 12 devient led

void setup()
{
  pinMode(bouton, INPUT); // Initialise la patte 4 comme entrée
  pinMode(led, OUTPUT); // Initialise la patte 12 comme sortie
}

void loop()
{
  if (digitalRead(bouton) == 1) //teste si le bouton a une valeur de 1 (appuyé en pull-down)
  {
    digitalWrite(led, HIGH); // allume la LED
  }
  else
  {
    digitalWrite(led, LOW); // éteint la LED
  }
}
```

Si on échange, on a le comportement opposé. En électronique pur, il aurait fallu recâbler. On peut même envisager des effets (clignotements, fading, etc).

Les PIN et modes

- Les broches analogiques peuvent être utilisées en tant que broches numériques, représentées par les nombres 14 (entrée analogique 0) à 19 (entrée analogique 5).
- Chacune des broches numériques possède une résistance interne de pull-up de 20 à 50 KOhms déconnectée par défaut.
- La broche numérique 13 est plus difficile à utiliser que les autres en tant qu'entrée numérique car elle est associée à une résistance et sa LED soudées sur le circuit imprimé de la carte sur la plupart des cartes.
 - Si vous activez la résistance interne de rappel au plus de 20KOhms, cela mettra la broche à 1,7V au lieu des 5V théoriques car la LED et la résistance associées à la broche abaisse la tension, qui est toujours considérée au niveau BAS (LOW). Ainsi, si vous devez utiliser la broche 13 en tant qu'entrée numérique, utiliser une résistance de rappel au plus externe.

Allume la LED quand on presse

```
/*
 * Allume LED en fonction de l'état du bouton poussoir
 */
const int bouton = 4;    // la patte 4 devient bouton
const int led = 12;     // la patte 12 devient led
int etatbouton;         // variable qui enregistre l'état du bouton

void setup()
{
  pinMode(bouton, INPUT); // Initialise le bouton comme entrée
  pinMode(led, OUTPUT); // Initialise la led comme sortie
  etatbouton = LOW; // Initialise l'état du bouton comme relâché
}

void loop()
{
  etatbouton = digitalRead(bouton); //On mémorise l'état du bouton

  if(etatbouton == LOW) //teste si le bouton a un niveau logique BAS
  {
    digitalWrite(led,LOW); //la LED reste éteinte
  }
  else //teste si le bouton a un niveau logique différent de BAS (donc HAUT)
  {
    digitalWrite(led,HIGH); //le bouton est appuyé, la LED est allumée
  }
}
```

pinMode()

- Description
 - Configure la broche spécifiée pour qu'elle se comporte soit en entrée, soit en sortie.
- Syntaxe
 - pinMode(broche, mode)
- Paramètres
 - broche: le numéro de la broche de la carte Arduino dont le mode de fonctionnement (entrée ou sortie) doit être défini.
 - mode: soit INPUT (entrée en anglais) ou OUTPUT (sortie en anglais)

digitalWrite()

• Description

- Met un niveau logique HIGH (HAUT en anglais) ou LOW (BAS en anglais) sur une broche numérique.
- Si la broche a été configurée en SORTIE avec l'instruction pinMode(), sa tension est mise à la valeur correspondante : 5V (ou 3.3V sur les cartes Arduino 3.3V) pour le niveau HAUT, 0V (masse) pour le niveau BAS.

• Syntaxe

- digitalWrite(broche, valeur)

• Paramètres

- broche: le numéro de la broche de la carte Arduino
- valeur : HIGH ou LOW (ou bien 1 ou 0)

digitalRead()

• Description

- Lit l'état (= le niveau logique) d'une broche précise en entrée numérique, et renvoie la valeur HIGH (HAUT en anglais) ou LOW (BAS en anglais).

• Syntaxe

- digitalRead(broche)

• Paramètres

- broche : le numéro de la broche numérique que vous voulez lire. (int)

• Valeur retournée

- Renvoie la valeur HIGH (HAUT en anglais) ou LOW (BAS en anglais)

- Remarque : Si la broche numérique en entrée n'est connectée à rien, l'instruction digitalRead() peut retourner aussi bien la valeur HIGH ou LOW (et cette valeur peut changer de façon aléatoire)

digitalWrite()

- Si la broche est configurée en ENTRÉE, écrire un niveau HAUT sur cette broche a pour effet d'activer la résistance interne de 20KOhms de "rappel au plus" (pullup) sur cette broche. À l'inverse, mettre un niveau BAS sur cette broche en ENTRÉE désactivera le pullup interne.

• Exemple :

```
pinMode(pin, INPUT); // configure la broche en entrée
digitalWrite(pin, HIGH); // écrit la valeur HIGH (=1) sur la broche en entrée
// ce qui active la résistance de "rappel au +" (pullup) au plus de la broche
```

- Cette instruction met la valeur 0/1 dans le bit de donnée qui est associé à chaque broche, ce qui explique qu'on puisse le mettre à 1, même si la broche est en entrée.

Exemple : on allume encore une LED avec un bouton mais avec le pull up interne !

```
int ledPin = 13; // LED connectée à la broche n°13
int inPin = 7; // un bouton poussoir connecté à la broche 7
int val = 0; // variable pour mémoriser la valeur lue

void setup()
{
  pinMode(ledPin, OUTPUT); // configure la broche 13 en SORTIE
  pinMode(inPin, INPUT); // configure la broche 7 en ENTREE

  digitalWrite(inPin, HIGH); // écrit la valeur HIGH (=1) sur la broche en entrée
  // ce qui active la résistance de "rappel au +"
  // (pullup) au plus de la broche
}

void loop()
{
  val = digitalRead(inPin); // lit l'état de la broche en entrée
  // et met le résultat dans la variable

  digitalWrite(ledPin, val); // met la LED dans l'état du BP
  // (càd allumée si appuyé et inversement)
}
```


analogRead()

- Description

- **Lit la valeur de la tension présente sur la broche spécifiée.** La carte Arduino comporte 6 voies (8 voies sur le Mini et le Nano, 16 sur le Méga) connectées à un convertisseur analogique-numérique 10 bits. Cela signifie qu'il est possible de transformer la tension d'entrée entre 0 et 5V en une valeur numérique entière comprise entre 0 et 1023. Il en résulte une résolution (écart entre 2 mesures) de : 5 volts / 1024 intervalles, autrement dit une précision de 0.0049 volts (4.9 mV) par intervalle !
- Une **conversion analogique-numérique** dure environ 100 µs (100 microsecondes soit 0.0001 seconde) pour convertir l'entrée analogique, et donc la fréquence maximale de conversion est environ de 10 000 fois par seconde.

Exemple de lecture et d'envoi sur le Serial

```
int analogPin = 3; // une résistance variable (broche du milieu)
                  //connectée sur la broche analogique 3
                  // les autres broches de la résistance sont connectées
                  // l'une au 5V et l'autre au 0V

int val = 0; // variable de type int pour stocker la valeur de la mesure

void setup()
{
  Serial.begin(9600); // initialisation de la connexion série
  // IMPORTANT : la fenêtre terminal côté PC doit être réglée sur la même valeur
}

void loop()
{
  // lit la valeur de la tension analogique présente sur la broche
  val = analogRead(analogPin);

  // affiche la valeur (comprise en 0 et 1023) dans la fenêtre terminal PC
  Serial.println(val);
}
```

analogRead()

- Syntaxe

- analogRead(broche_analogique)

- Paramètres

- broche_analogique : le numéro de la broche analogique (et non le numéro de la broche numérique) sur laquelle il faut convertir la tension analogique appliquée (0 à 5 sur la plupart des cartes Arduino, 0 à 7 sur le Mini et le Nano, 0 à 15 sur le Mega)

- Valeur renvoyée

- valeur int (0 to 1023) correspondant au résultat de la mesure effectuée

- Remarque :

- Si la broche analogique est laissée non connectée, la valeur renvoyée par la fonction analogRead() va fluctuer en fonction de plusieurs facteurs (tels que la valeur des autres entrées analogiques, la proximité de votre main vis à vis de la carte Arduino, etc.).

analogWrite()

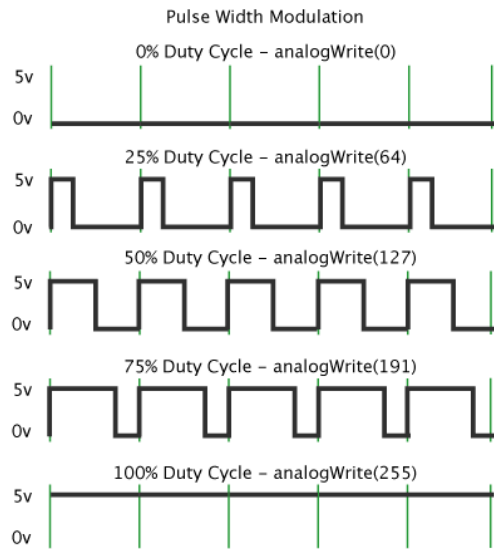
- Description

- **Génère une impulsion de largeur / période voulue sur une broche de la carte Arduino** (onde PWM - Pulse Width Modulation en anglais ou MLI - Modulation de Largeur d'Impulsion en français).

- Peut-être utilisé pour faire briller une LED avec une luminosité variable ou contrôler un moteur à des vitesses variables.

- Après avoir appelé l'instruction analogWrite(), la broche générera une onde carrée stable avec un "duty cycle" (fraction de la période où la broche est au niveau haut) de longueur spécifiée (en %), jusqu'à l'appel suivant de l'instruction analogWrite() (ou bien encore l'appel d'une instruction digitalWrite() ou digitalWrite() sur la même broche). La fréquence de l'onde PWM est approximativement de 490 Hz (soit 490 périodes par seconde).

analogWrite()



- Remarque : Sur les nouvelles cartes Arduino (incluant le Mini et le BT) avec le microcontrôleur ATmega168, cette fonction fonctionne sur les broches 3, 5, 6, 9, 10 et 11. Les cartes plus anciennes USB et série basées sur l'ATmega8 ne supportent l'instruction analogWrite() que sur les broches 9, 10 et 11.

Exemple

- Fixer la luminosité d'une LED proportionnellement à la valeur de la tension lue depuis un potentiomètre.

```
int ledPin = 9; // LED connectée sur la broche 9
int analogPin = 3; // le potentiomètre connecté sur la broche analogique 3
int val = 0; // variable pour stocker la valeur de la tension lue

void setup()
{
  pinMode(ledPin, OUTPUT); // configure la broche en sortie
}

void loop()
{
  val = analogRead(analogPin); // lit la tension présente sur la broche en entrée
  analogWrite(ledPin, val / 4); // Résultat d'analogRead entre 0 to 1023,
  // résultat d'analogWrite entre 0 to 255
  // => division par 4 pour adaptation
}
```

analogWrite()

- Syntaxe
 - analogWrite(broche, valeur);
- Paramètres
 - Broche : la broche utilisée pour "écrire" l'impulsion. Cette broche devra être une broche ayant la fonction PWM, Par exemple, sur la UNO, ce pourra être une des broches 3, 5, 6, 9, 10 ou 11.
 - valeur: la largeur du "duty cycle" (proportion de l'onde carrée qui est au niveau HAUT) : entre 0 (0% HAUT donc toujours au niveau BAS) et 255 (100% HAUT donc toujours au niveau HAUT).
- Notes
 - Il n'est pas nécessaire de faire appel à l'instruction pinMode() pour mettre la broche en sortie avant d'appeler la fonction analogWrite().
 - L'impulsion PWM générée sur les broches 5 et 6 pourront avoir des "duty cycle" plus long que prévu. La raison en est l'interaction avec les instructions millis() et delay(), qui partagent le même timer interne que celui utilisé pour générer l'impulsion de sortie PWM.

Quelques fonctions bien utiles

- Mathématiques
 - min(x, y)
 - max(x, y)
 - abs(x)
 - constrain(x, a, b)
 - map(valeur, fromLow, fromHigh, toLow, toHigh)
 - pow(base, exposant)
 - sq(x)
 - sqrt(x)
- Temps
 - unsigned long millis()
 - unsigned long micros()
 - delay(ms)
 - delayMicroseconds(us)

constrain(x, a, b)

- Description
 - **Contraint un nombre à rester dans une fourchette précise.**
- Syntaxe
 - constrain(x,a,b)
- Paramètres
 - x: le nombre qui doit être contraint, tout type de donnée.
 - a: la limite inférieure de la fourchette, tout type de donnée.
 - b: la limite supérieure de la fourchette, tout type de donnée.
- Valeur renvoyée
 - Renvoie x si x a une valeur comprise entre a et b
 - Renvoie a si x est inférieur à a
 - Renvoie b si x est plus grand que b

map(value, fromLow, fromHigh, toLow, toHigh)

- Description (suite)
 - La limite basse de chaque fourchette peut être supérieure ou inférieure à la limite haute, dès lors l'instruction map() peut être utilisée pour inverser l'ordre des valeurs, par exemple :
 - `y = map(x, 1, 50, 50, 1);`
 - y évolue en sens inverse de x (càd si `x = 1, y=50` et inversement)
 - Cette instruction supporte également des valeurs négatives, tel que dans cet exemple :
 - `y = map(x, 1, 50, 50, -100);`
 - Cette utilisation est aussi valide et fonctionne normalement.
 - L'instruction map() utilise des valeurs **entières** qui, par définition, ne peuvent fournir les décimales, alors que les calculs le devraient. Aussi **la partie décimale est tronquée**, et les valeurs ne sont pas arrondies ou moyennées.

map(value, fromLow, fromHigh, toLow, toHigh)

- Description
 - **Ré-étalonne un nombre d'une fourchette de valeur vers une autre fourchette.**
 - Ainsi, une valeur basse source sera étalonnée en une valeur basse de destination, une valeur haute source sera étalonnée en une valeur haute de destination, une valeur entre les deux valeurs source sera étalonnée en une valeur entre les deux valeurs destinations, **en respectant la proportionnalité.**
 - Cette fonction est très utile pour effectuer des changements d'échelle automatiques.
 - Cette fonction **ne contraint pas les valeurs** à rester dans les limites indiquées, car les valeurs en dehors de la fourchette sont parfois attendues et utiles. L'instruction constrain() doit être utilisée également avant ou après cette fonction, si les limites de la fourchette utilisée doivent être respectées.

map(value, fromLow, fromHigh, toLow, toHigh)

- Syntaxe
 - map (valeur, limite_basse_source, limite_haute_source, limite_basse_destination, limite_haute_destination)
- Paramètres
 - valeur : le nombre à ré-étalonner
 - limite_basse_source: la valeur de la limite inférieure de la fourchette de départ
 - limite_haute_source: la valeur de la limite supérieure de la fourchette de départ
 - limite_basse_destination: la valeur de la limite inférieure de la fourchette de destination
 - limite_haute_destination: la valeur de la limite supérieure de la fourchette de destination
- Valeur renvoyée
 - La valeur ré-étalonnée

Exemple de l'utilisation de map

- Fixer la luminosité d'une LED proportionnellement à la valeur de la tension lue depuis un potentiomètre.

```
int ledPin = 9;      // LED connectée sur la broche 9
int analogPin = 3;  // le potentiomètre connecté sur la broche analogique 3
int val = 0;        // variable pour stocker la valeur de la tension lue

void setup()
{
  pinMode(ledPin, OUTPUT); // configure la broche en sortie
}

void loop()
{
  val = analogRead(analogPin); // lit la tension présente sur la broche en entrée
  // Ré-étalonne la valeur entre 0 et 1023 sur une fourchette entre 0 et 255
  val = map(val, 0, 1023, 0, 255);
  analogWrite(ledPin, val);
}
```

delay(ms)

• Description

- Réalise une pause dans l'exécution du programme pour la durée (en millisecondes) indiquée en paramètre. (Pour mémoire, il y a 1000 millisecondes dans une seconde !)

• Syntaxe

- delay (ms);

• Paramètres

- ms (unsigned long): le nombre de millisecondes que dure la pause

Pour les matheux

- Voici la fonction entière map :

```
long map(long x, long in_min, long in_max, long out_min, long out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

- Comme nous l'avons vu **map est fonction très puissante et très utile en pratique** puisqu'elle permet de réaliser une adaptation d'échelle de valeur (entre la valeur d'une conversion analogique numérique brute et la valeur réelle de la tension par exemple) en une seule ligne, là où plusieurs lignes de calcul seraient normalement nécessaires.

Avertissements concernant delay(ms)

- Bien qu'il soit facile de faire clignoter une LED avec l'instruction delay(), et que de nombreux programmes utilisent de courtes pauses pour de nombreuses tâches telles que la commutation, **l'utilisation de l'instruction delay() a des inconvénients non négligeables.**
 - **Aucune lecture de valeur sur un capteur, opération mathématique ou manipulation de bits ne peut avoir lieu durant une instruction delay(),** et en effet, cela fait stopper toute autre activité le temps de l'exécution de la pause.
 - Pour une autre approche du contrôle des temporisations, voir l'instruction millis() et le programme d'exemple de clignotement sans délai.
 - Les programmeurs avancés (vous ?) évitent habituellement d'utiliser l'instruction delay() pour des pauses supérieures à 10 millisecondes, à moins que le programme Arduino ne soit très simple.
- **Certaines choses se font cependant même lorsque l'instruction delay() est exécutée** sur le microcontrôleur ATmega, car l'instruction **delay() ne désactive pas les interruptions.**
 - Les communications série qui arrivent sur la broche RX sont prises en compte, la génération d'impulsion PWM (analogWrite) et l'état des broches stables sont maintenus, et les interruptions fonctionnent comme elles le doivent.

delayMicroseconds(us)

• Description

- Stoppe le programme pendant la durée (en microsecondes) indiquée en paramètre. (Pour mémoire, il y a 1000 microsecondes dans une milliseconde, et un million de microsecondes dans une seconde).
- Actuellement, la valeur la plus grande qui peut produire une pause précise est 16383. Ceci pourra changer dans les versions futures d'Arduino. Pour des délais plus longs que quelques milliers de microsecondes, vous devriez plutôt utiliser l'instruction delay().

• Syntaxe

- delayMicroseconds(us)

• Paramètres

- us: le nombre de microsecondes que doit durer la pause.

millis()

• Description

- Renvoie le nombre de millisecondes depuis que la carte Arduino a commencé à exécuter le programme courant. Ce nombre débordera (c'est-à-dire sera remis à zéro) après 50 jours approximativement.

• Syntaxe

- variable_unsigned_long=millis();

• Valeur retournée

- Le nombre de millisecondes depuis que le programme courant a démarré. Renvoie un long non signée.

Avertissements concernant delayMicroseconds(us)

- Cette fonction fonctionne de façon très précise pour des valeurs de 3 microsecondes et plus. Nous ne pouvons pas assurer que l'instruction delayMicroseconds donnera un résultat correct pour des durées plus courtes.
- L'instruction delayMicroseconds(0) générera un délai beaucoup plus long qu'attendu (1020 µs environ), comme si on utilisait un nombre négatif comme paramètre.

Exemple

```
unsigned long time;

void setup(){
  Serial.begin(9600);
}

void loop(){
  Serial.print("Time: ");
  time = millis();
  //affiche sur le PC le temps depuis que le programme a démarré
  Serial.println(time);
  // pause d'une seconde afin de ne pas envoyer trop de données au PC
  delay(1000);
}
```

Faire clignoter sans « delay »

```
const int ledPin = 13;
int ledState = LOW; // l'état de la LED
long previousMillis = 0; // stocke le temps à laquelle la LED a été changée

long interval = 1000; // intervalle auquel on veut clignoter (en millisecondes)

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  // vérifie si c'est le moment de faire clignoter la LED. C'est à dire si
  // la différence entre le temps actuel et le temps du précédent clignotement
  // de la LED est supérieur à l'intervalle auquel on veut qu'elle clignote.
  unsigned long currentMillis = millis();

  if(currentMillis - previousMillis > interval) {
    // sauvegarde le temps du précédent changement d'état de la LED
    previousMillis = currentMillis;

    if (ledState == LOW)
      ledState = HIGH;
    else
      ledState = LOW;

    digitalWrite(ledPin, ledState);
  }
  // ici on peut mettre d'autres instructions pour faire du boulot
  // en « parallèle » si ce n'est pas le moment de faire clignoter
```

Quelques mots encore sur les entrées/sorties

- Vous vous êtes peut être rendu compte que les **entrées/sorties vues sont non bloquantes** contrairement au `Saisir()` vu en algorithmique ou au `cin` << vu en C++ qui stoppe l'exécution jusqu'à ce qu'une valeur ait été saisie

- C'est à dire que les **entrées/sorties sont asynchrones** (non-bloquantes)
- Ainsi, par exemple, lors d'un appel à `digitalRead(unPin)`, on récupère la valeur sur la broche `unPin`, soit HIGH, soit LOW.



micros()

• Description

- **Renvoie le nombre de microsecondes depuis que la carte Arduino a démarré le programme en cours.** Ce nombre déborde (re passe à 0) après approximativement 70 minutes.

Sur les cartes Arduino à 16Mhz (par exemple le Nano), **cette fonction a une résolution de 4 microsecondes** (ce qui veut dire que la valeur retournée est toujours un multiple de quatre, autrement dit que la fonction compte de quatre en quatre). Sur les cartes Arduino à 8Mhz (par exemple LilyPad), cette fonction a une résolution de 8 microsecondes.

• Syntaxe

- `variable_unsigned_long=micros();`

• Valeur renvoyée

- Le nombre de microsecondes écoulées depuis que le programme en cours a démarré, sous la forme d'un nombre de type `unsigned long`.

Problèmes d'entrées/sorties non bloquantes

- **Il faut être conscient qu'on peut « rater » des états**

```
int ledPin = 13; // LED connectée à la broche n°13
int inPin = 7; // un bouton poussoir connecté à la broche 7
int val = 0; // variable pour mémoriser la valeur lue

void setup()
{
  pinMode(ledPin, OUTPUT); // configure la broche 13 en SORTIE
  pinMode(inPin, INPUT); // configure la broche 7 en ENTREE

  digitalWrite(inPin, HIGH); // écrit la valeur HIGH (=1) sur la broche en entrée
  // ce qui active la résistance de "rappel au +"
  // (pullup) au plus de la broche
}

void loop()
{
  val = digitalRead(inPin); // lit l'état de la broche en entrée
  // et met le résultat dans la variable

  digitalWrite(ledPin, val); // met la LED dans l'état du BP
  // (càd allumée si appuyé et inversement)

  // ici un processus très long ou complexe pouvant durer plusieurs secondes, minutes
  // implique qu'on rate les événements d'appui sur le bouton
  for (int i = 0; i < 100; i++)
    delay(10);
}
```


Mais comment faire ?

- Dans l'exemple précédent on « ratait » les appuis sur un bouton pendant le processus, mais l'information aurait pu venir d'ailleurs, un capteur ultrason sur un robot par exemple et si la tâche longue consiste à avancer (ou faire d'autres opérations) et bien pendant ce temps là on peut rencontrer un obstacle sans en être capable de récupérer l'information !
 - On parle ici de rouler, mais toute opération bloquante (comme un délai) peut être un handicap.
- Comment faire si on veut exécuter une action particulière quand une broche change d'état ?

Polling versus Interruption

- On peut utiliser une interruption matérielle (hardware) qui appellera une fonction associée lors du changement d'état sur une broche.
- Le processeur exécutera le programme principal jusqu'à ce qu'il soit interrompu à l'arrivée d'une interruption qui déclenchera le code de la fonction associée à cette interruption.
- Une fois le code de la fonction associée à l'interruption sera fini, la « main » sera rendu au programme principal à l'endroit où l'exécution s'était arrêtée.
 - Attention entre temps (c'est-à-dire durant l'exécution de la fonction associée à l'interruption) des variables ont pu changer de valeurs (il peut y avoir des problèmes de cohérence)

Polling versus Interruption

- On peut faire du Polling (attente active ou « scrutation »)
 - Par exemple dans le loop si on veut attendre que la broche inPin passe à un état LOW, on peut utiliser le code suivant :

```
int val;  
do  
{  
  val = digitalRead(inPin);  
} while (val!= LOW);
```

// ici du code à exécuter

...

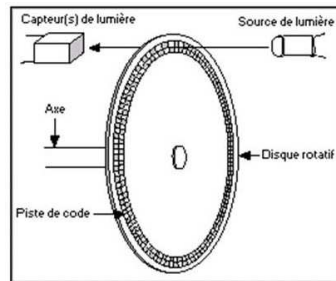
- L'inconvénient est que l'on ne fait rien d'autre que lire, dans l'attente d'un changement, alors que dans le code qui suit cette attente, il y a peut être des tâches indépendantes de la valeur val qui pourraient être exécutées (mais en les exécutant, on risque de « rater » le changement d'état)
- On consomme beaucoup de cycles CPU pour rien

Exemple d'utilisation d'interruption

```
int ledPin = 13; // LED connectée à la broche n°13  
int inPin = 2; // un bouton poussoir connecté à la broche 2 car interruption 0  
int INT0 = 0;  
volatile int val = LOW; // variable pour mémoriser la valeur lue  
  
void setup()  
{  
  pinMode(ledPin, OUTPUT); // configure la broche 13 en SORTIE  
  
  digitalWrite(inPin, HIGH); // écrit la valeur HIGH (=1) sur la broche en entrée  
                             // ce qui active la résistance de "rappel au +"  
                             // (pullup) au plus de la broche  
  
  // Attache la fonction à la patte ayant ce numéro d'interruption  
  // et surveille tout CHANGEMENT d'état  
  attachInterrupt(INT0, etatChange, CHANGE);  
}  
  
void loop()  
{  
  // ici un processus très long ou complexe pouvant durer plusieurs secondes, minutes  
  // SANS qu'on rate les événements d'appui sur le bouton  
  for (int i = 0; i < 100; i++)  
    delay(10);  
}  
  
void etatChange()  
{  
  val = !val; // l'état vient de changer, donc on le change dans val !  
  
  digitalWrite(ledPin, val); // met la LED dans l'état du BP  
                             // (càd allumée si appuyé et inversement)  
}
```

Avantages des interruptions

- On économise du CPU par rapport au polling
- On a un modèle de programmation événementiel permettant de simuler plusieurs tâches quand on ne dispose pas de threads.
- Elles sont très utiles pour faire des choses de façon automatique dans un programme pour microcontrôleur, et peuvent aider à résoudre les problèmes de temporisation. Une bonne utilisation pour une interruption peut-être la lecture d'un encodeur rotatif, pour contrôler les entrées de l'utilisateur.



Principe des souris à boule !

Quelques cas d'utilisation

- **Encodeur rotatif** :
 - Si vous voulez que le programme prenne toujours en compte les impulsions venant d'un encodeur rotatif, sans jamais négliger une impulsion, il serait très compliqué d'écrire un programme pour faire quoique ce soit, parce que le programme devrait en permanence examiner les broches connectées à l'encodeur, afin de prendre en compte les impulsions lorsqu'elle surviendront.
- D'autres capteurs ont un rôle d'interface dynamique semblable, tel que :
 - **capteur de sons** essayant de détecter un bruit
 - **capteur infra-rouge** (photo-transistor) essayant de détecter un obstacle
- Dans toutes ces situations, l'utilisation des interruptions libère le microcontrôleur pour faire d'autres choses tant que l'événement attendu ne survient pas.

Opto-Interrupteur (interruption)

```
const int OIPin = 2; // Patte de l'Opto-interrupteur
const int ledPin = 13; // Patte de la LED
int INTO = 0;

volatile int beamState = LOW; // état du faisceau

void setup() {

    pinMode(ledPin, OUTPUT); // initialise la patte de la LED comme sortie
    // Attache la fonction à la patte ayant ce numéro d'interruption
    //et surveille tout front montant (LOW->HIGH) : RISING
    attachInterrupt(INT0, faisceauCoupe, RISING);
}

void loop() {

}

void faisceauCoupe()
{
    beamState = !beamState; // l'état vient de changer, donc on le change dans val !

    digitalWrite(ledPin, beamState); // met la LED dans l'état correspondant au faisceau
    // (câd allumée si il est coupé et inversement)
}
```

Opto-Interrupteur (polling)

```
const int OIPin = 2; // Patte de l'Opto-interrupteur
const int ledPin = 13; // Patte de la LED

int beamState = LOW; // état du faisceau

void setup() {

    pinMode(ledPin, OUTPUT); // initialise la patte de la LED comme sortie
    pinMode(OIPin, INPUT); // initialise la patte de l'opto-interrupteur comme entrée
}

void loop() {
    beamState = digitalRead(OIPin); // lit l'état de l'opto-interrupteur

    // Vérifie si le faisceau a été interrompu
    // si il l'est, beamState est HIGH:
    if (beamState == HIGH) {
        digitalWrite(ledPin, HIGH); // allume la LED
    }
    else {
        digitalWrite(ledPin, LOW); // éteint la LED
    }
}
```



Aussi appelé opto-coupleur

J'en ai marre d'être interrompu !

Allons plus loin avec l'Opto-Interrupteur (et les interruptions)

```
/* Ce programme compte la vitesse d'un moteur CC en nombre de tours par minute.
 * Ce programme utilise un optocoupleur en fourche et une interruption externe.
 * Le résultat est affiché dans la fenêtre Terminal du logiciel Arduino. */
```

```
volatile long comptageImpuls=0; // variable accessible dans la routine interruption externe 0

long timeRef=0; // variable pour temps de référence en millisecondes
long delai=1000; // variable pour délai de comptage en millisecondes

void setup(){
  Serial.begin(115200); // initialise connexion série à 115200 bauds
  attachInterrupt(0, gestionINT0, RISING); // attache l'interruption 0 à la fonction gestionINT0()
  timeRef=millis(); // initialisation de la référence du comptage
}

void loop(){
  if (millis()>(timeRef+delai)) { // si le delai de comptage est écoulé
    timeRef=timeRef+delai; // réinitialise le délai de comptage

    Serial.print("Nombre Impulsions par seconde =");
    Serial.println(comptageImpuls); // affiche impulsions/sec
    Serial.print("Nombre tours par seconde =");
    Serial.println(comptageImpuls/2); // affiche nombre tours/sec

    comptageImpuls=0; // RAZ comptage impulsions
  }
}

void gestionINT0() { // la fonction appelée par l'interruption externe n°0
  comptageImpuls=comptageImpuls+1; // incrémente comptage Impulsion
}
```



Est ce qu'il peut y avoir des problèmes ?

attachInterrupt (interruption, fonction, mode)

• Paramètres

- interruption : le numéro de l'interruption (type int)
- fonction: la fonction à appeler quand l'interruption survient; la fonction ne doit recevoir aucun paramètres et ne renvoie rien. Cette fonction est également appelée une routine de service d'interruption (ou ISR).
- mode : définit la façon dont l'interruption externe doit être prise en compte. Quatre constantes ont des valeurs prédéfinies valables :
 - LOW : pour déclenchement de l'interruption lorsque la broche est au niveau BAS
 - CHANGE : pour déclenchement de l'interruption lorsque la broche change d'état BAS/HAUT
 - RISING : pour déclenchement de l'interruption lorsque la broche passe de l'état BAS vers HAUT (front montant)
 - FALLING : pour déclenchement de l'interruption lorsque la broche passe de l'état HAUT vers l'état BAS (front descendant)
 - HIGH : pour déclenchement de l'interruption lorsque la broche est au niveau HAUT

attachInterrupt (interruption, fonction, mode)

• Description

- Spécifie la fonction à appeler lorsqu'une interruption externe survient. Remplace tout autre fonction qui était attaché à cette interruption.

La plupart des cartes Arduino ont deux interruptions externes : interruption externe n°0 sur la broche numérique 2 et interruption externe n°1 sur la broche numérique 3. (La carte Arduino Mega en possède quatre de plus : interruption externe n°2 sur la broche 21, n°3 sur la broche 20, n°4 sur la broche 19 et n°5 sur la broche 18.)

• Syntaxe

- attachInterrupt (interruption, fonction, mode)

Interruptions : Important

• Attention

- À l'intérieur de la fonction attachée à l'interruption, la fonction delay ne fonctionne pas et la valeur renvoyée par millis ne s'incrémente pas. Les données séries reçues pendant l'exécution de la fonction sont perdues.
- La fonction delayMicroseconds fonctionne normalement à l'intérieur de la fonction attachée.
- Vous devrez déclarer en tant que volatile toute les variables que vous modifier à l'intérieur de la fonction attachée à l'interruption.

volatile



Oups ! Il est temps de finir ce cours !

volatile

- Le mot-clé `volatile` est connu en tant que qualificateur de variable et il est utilisé usuellement avant la déclaration d'une type de variable, afin de modifier la façon dont le compilateur, et par voie de conséquence le programme, traitera la variable.
- Déclarer une variable volatile est une information pour le compilateur. Le compilateur est la partie du logiciel Arduino qui transforme votre code C/C++ en un code machine (c'est à dire en une suite de 0 et 1), lequel correspond aux instructions réelles pour le microcontrôleur Atmega de votre carte Arduino.
- Plus précisément, il indique au compilateur de charger la variable depuis la RAM et pas depuis un registre de stockage, lequel est un emplacement temporaire de la mémoire où les variables sont stockées et manipulées. Sous certaines conditions, la valeur de la variable stockée dans les registres peut-être imprécise.
- Une variable devra être déclarée volatile chaque fois que sa valeur pourra être changée par quelque chose d'autre que le code dans laquelle elle apparaît, tel qu'un fil d'exécution concurrent. Avec Arduino, la seule situation où cela risque d'arriver concerne les sections de codes associées aux interruptions, appelées également routines de service des interruptions.

detachInterrupt(interruption)

- Description
 - Désactive l'interruption externe spécifiée
- Syntaxe
 - `detachInterrupt(interruption)`
- Paramètres
 - `interruption`: le numéro de l'interruption externe à désactiver.

noInterrupts()

- Description
 - Désactive les interruptions (vous pouvez les réactiver avec l'instruction `interrupts()`). Les interruptions permettent à certaines tâches importantes de survenir à tout moment, et elles sont activées par défaut avec Arduino. Plusieurs fonctions ne fonctionnent pas lorsque les interruptions sont désactivées et les communications sont ignorées (série en particulier). Les interruptions peuvent perturber le déroulement du code et peuvent être désactivées pour des sections critiques du code.
- Syntaxe
 - `noInterrupts()`

interrupts()

- Description
 - Réactive les interruptions (après qu'elles aient été désactivée par l'instruction noInterrupts()).
- Syntaxe
 - interrupts()

Exemple

```
void setup() {}

void loop()
{
  noInterrupts(); // désactivation des interruptions

  // code critique, sensible au temps, ici

  interrupts(); // réactivation des interruptions

  // le reste du code ici
}
```

Les interruptions

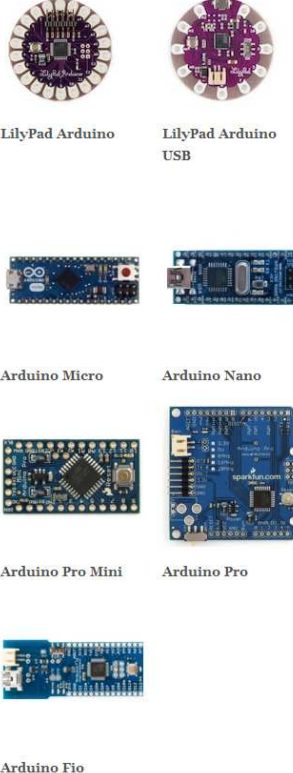
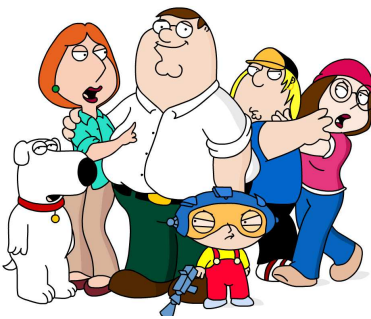
- Nous venons de voir des exemples d'interruptions matérielles.
- Mais sachez qu'il existe aussi des interruptions logicielles (par exemple déclenchées par des timers)

Des choses beaucoup trop avancées pour vous

- La manipulation des ports via les registres de port :

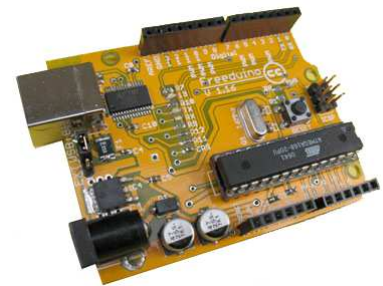
http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.PortManipulation

La famille

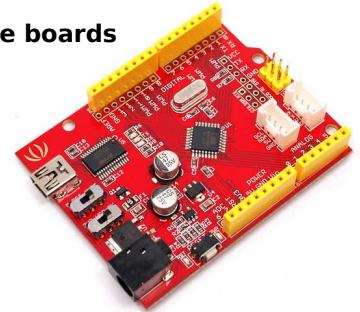


Des cousins plus ou moins proches

Arduino footprint-compatible boards



Freeduino 0603 SMT prototype

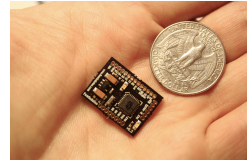


Seeeduino V3.0 (Atmega 328P)

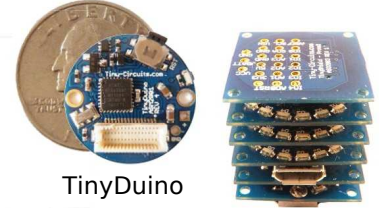


Scooby-Doo and his cousin Scooby-Dum.

Software-compatibility only



Femtoduino Kit

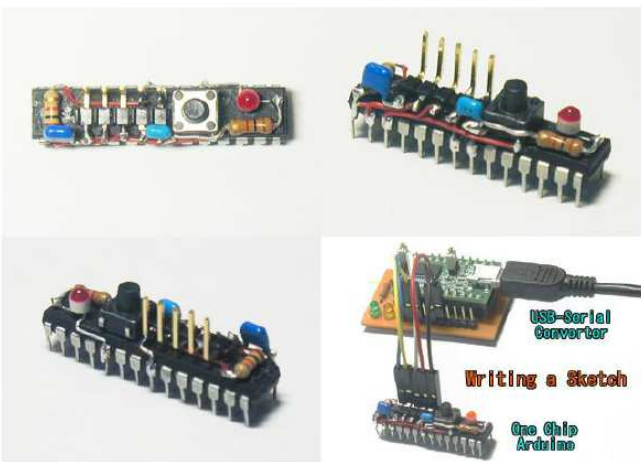


TinyDuino

En savoir plus http://en.wikipedia.org/wiki/List_of_Arduino_boards_and_compatible_systems

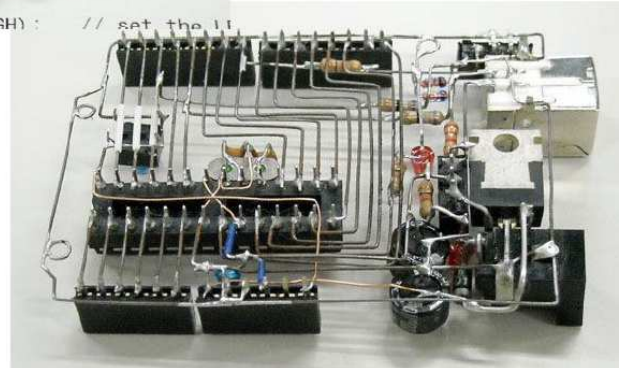
Des jolis Arduino

One chip Arduino
Kimio Kosaka



http://www.geocities.jp/arduino_diecimila/obaka/project-2/index_en.html

Des jolis Arduino

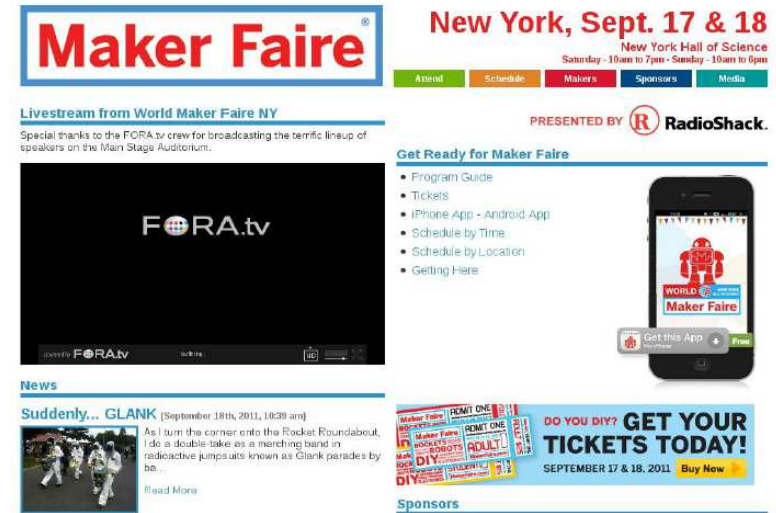


Apprendre soit même



<http://makezine.com/>

Apprendre soit même



<http://makerfaire.com/>

Apprendre soit même



<http://www.instructables.com/>

- Pour en savoir plus
- Le documentaire sur Arduino :
<http://www.archive.org/details/Arduino.TheDocumentary.English>
- Arduino en BD :
<http://www.jodyculkin.com/wp-content/uploads/2011/09/arduino-comic-latest1.pdf>
- Arduino (livre gratuit)
http://fr.flossmanuals.net/_booki/arduino/arduino.pdf
- ADC et Arduino
<https://sites.google.com/site/measuringstuff/the-arduino>
- En savoir plus sur les LED
<http://www.positron-libre.com/cours/electronique/diode/led/diode-led.php>
- Pour une découverte ludique de l'électricité, je vous conseille de regarder le *C'est pas sorcier* consacré à ce thème
 - Partie 1: <http://00.lc/cl>
 - Partie 2: <http://00.lc/cm>

Pour en savoir plus

- Bouton poussoir (pull-up/pull-down et déparasitage)

<http://arduino103.blogspot.fr/2011/05/entree-bouton-resistance-pull-up-pull.html>

- Asservissement en vitesse d'un moteur avec Arduino

<http://www.ferdinandpiette.com/blog/2012/04/asservissement-en-vitesse-dun-moteur-avec-arduino/>

- Les 10 points essentiels pour choisir un numériseur/oscilloscope

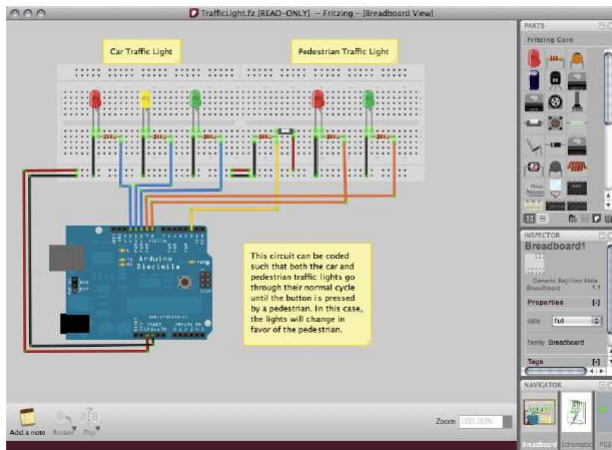
- Contient des choses sympas que je voulais mettre dans le cours

<http://www.ni.com/white-paper/4333/fr>

Des conseils

- Installez l'éditeur de montage Fritzing. Il vous sera utile pour garder en mémoire les montages réalisés. <http://fritzing.org/>

- Pour dessiner des dessins de type breadboard, **des schémas**, des PCB (typon) en vue de réaliser des circuits.
- Le site¹¹ regorge de schémas.



Des ressources

- Web :

- <http://www.arduino.cc/>
- <http://www.ladyada.net/learn/arduino/>
- <http://www.todbot.com/blog/category/arduino/>
- <http://www.freeduino.org/>
- <http://www.adafruit.com/>
- <http://www.sparkfun.com/>
- <http://www.seeedstudio.com/>
- <http://www.evola.fr/>

- Livres :

- “Arduino Programming Notebook”, Brian W. Evans
- “Physical Computing”, Dan O’Sullivan & Tom Igoe
- “Making Things Talk”, Tom Igoe
- “Hacking Roomba”, TodE. Kurt

Electronique / électricité: recommandations

- **Attention** : entre 0v et 30v pas de risque, à 220V c'est mortel
- **Les manipulation à 110v ou 220v** : demande un minimum de connaissance et une rigueur stricte (demandez conseil à une personne expérimentée)
- Consultez les schémas et les programmes sur les sites⁶ et⁷ (très nombreux exemples)
- Pour des simulations de la partie électronique utilisez des simulateurs et expérimentez : icircuit⁸ ou Ktechlab⁹

⁶<http://www.arduino.cc/>

⁷<http://www.fritzing.org/>

⁸<http://www.falstad.com/circuit/>

⁹<http://sourceforge.net/projects/ktechlab/>