C

- langage compiler
- base : fichier text nomer avec un ".c"
- c'est un langage compiler
- on travaille dans le "fichier source" (celui que l'on compile)
- Consaille rester sur de ASCII (sans accents)

Le C :

Fonction particuliere : main()

fonction de "base" de C, c'est celui qui est executer en premier. on peu inclure des librairies (ex : on en a besion pour faire un print)

Ex :

```
#include <stdio.h>
```

On doit declarer les variable avant de pouvoir les utilisers (ex : "int toto;")

Type de variable :

1. int, -> entier (32 bits sur PC)
2. float, -> nombre a virgule (32/64 bits sur PC)
3. double, -> nombre a virgule (+grand) (64/128 bits sur PC)
4. char, -> entier sur 1 octet
5. (viod)

Les tailles :

1. int >= char
2. double >= float

Chaque instruction termine par un ";"

definition d'une variable :

```
(modificateur) type nom.variable ;
ex :
|| float B;
|| int toto;
```

Modificateur

- modifie sur la taille, ex ; short (reduit la taille), long (augmente la taille de la variable)
- modifie la presence du signe, ex ; signet (par defaut), unsigne (enleve le bite de signe) (ex si un "unsigned char L;" de base une variable sur 8 bits vas de -127 a +127 elle ira de 0 a 255)

  int i;

-> de base juste comme ça il vaut ce qui il y avait avant a sont enplacement de RAM.

```
i = 17;
```

-> "=" -> recoit -> opérteur d'affectation -> on peut lui affecter un valeur ou une expression -> l'instrucation est calculer de droite a gauche

```
<variable> **=** valeur;
<variable> = expretion,
```

Autre exenple :

```
a = i = 17
```

on affect "17" a "i" et ce resultat a "a"

Autre operateur :

```
<variable> **+=** valeur;
```

Ex :

```
i = i + 1;
i += 1;
```

Opérateur

- +=
- -=
- *=
- /=
- <<= (decalage a gauche) (utiliser pour simplifier les operation pour eviter les divisions)

- []>>= (decalage a droite)

  int i; i = 12; i = 5/2; !! pas possible car i est un int et 5 et 2 aussi

Autre op

```
== ->Test d'egaliter
!= ->Test different
```

Tandis que :

```
i = ((float)5)/((float)2); -> fonctionne
i = 5.0 / 2.0 -> fonctionne le compilateur comprend
```

Exemple de code :

```
#include <stdio.h>

int main{
    int i;
    printf("i vaut %d \n",i);
    return 0;
}
```

Pour printf

-> pour un int %d -> pour un float %f -> pour un long int %ld -> pour un double %lf -> pour un char %c (-> pour un chaine de char %s) (string = tableau de char)

En C il n'y a pas de type boulein (si c'est 0 c'est vrai si c'est autre chose c'est faux)

Les structures

if

```
if(expression)
    expression; ou {bloc}
```

Ex :

```
if (i==0)
    printf("i est nul \n");
```

```
// le if prend que la premiere ligne donc :

if (i==0){
    printf("i est nul \n");
    printf("i est nul !\n");
}
```

else

```
if(expression)
    expression; ou {bloc}
else
    bloc ou instruction
```

wile

```
wile (expression)
    bloc ou instruction
// 1 je test 2 je fait tant que
do
    bloc ou instruction
while(expretion);
// je fait et je test si je doit reboucler
```

for

```
for(exp1;exp2;exp3)
    bloc ou intruction
```

Ex :

```
for (i=0; i<10; i++)
    //depart;test;pas
    printf("Coucou\n")

// execution : exp1 > contenue > exp3 > exp2
// si vrai : > contenue
// si faux (go next)
```

Ex bouvle infinie :

```
for(;;)
```

Swich

```
switch(expression){
    case valeur :
        bloc ou instruction
        break;
    case valeur :
        bloc ou instruction
        break;
    .
    .
    .
    default : ...
}
```

Ex :

```
int i;
i = 17
switch(i){
    case 14 :
        printf("A");
        break;
    case 17 :
        printf("A");
        break;
    case 18 :
        printf("C");
    case 22 :
        printf("D");
        break;
    default :
        printf("E");
}
```

Resultat :

- i = 17 -> B
- i = 12 -> A
- i = 51 -> E
- i = 18 -> CD
- i = 22 -> D

Chose qui marche pas :

```
case 12+1 :... (pas d'expretion)
case a :... (pas de variavle)
```

Les caracteres

Ex :

```
char d;
char e;

d = 65;
e = 'A';

if (e == d)
    printf("Pareil \n");
```

Utilisation hexa et autre

```
a = 'C';
a = '\n';
a = 17.9;
a = oxFFB2;
a = 0b10011101
    ou a = 2b10011101
```

Declarartion

Une variable est valide que dans son bloque de declaration

Ex :

```
 #include <stdio.h>

int main{
    int i;
    for (i=0; i<10; i++){
        int a;
        a = 17
        printf("%d \n",a),
    }
    printf("%d \n",a);
```

```
    return 0;
}
```

Compiler

Syntaxe :

```
cc -o fablab alaide.c

cc -o (executable generer) (fichier de base)
```

Moins bien :

```
gcc -o fablab alaide.c
```

Pour executer :

```
./fablab
```

Tips bash :

Si on a un return "return 0;" et que l'on fait ./fablab && eject /dev/cdrom

On aura tout executer

Et :

Si on a un return "return 1;" et que l'on fait ./fablab && eject /dev/cdrom

On aura pas le eject

Adresse

Avec un pointeur: int** c; int* b; int a; a = 17;

```
b = &a -> adresse de a

// pour la stoquer on a besion de : int* b; qui sert a stoquer les adresses des
int

c= &b

// pour la stoquer on a besion de : int** c; qui sert a stoquer les adresses des
int*
```

Pour les adress on utilise donc les etoiles ex : "float*"

Tableau

Un tableau n'est que la valeur de la premiere case.

Les entrées

```
int c;
printf("Entrez un chiffre ? \n");
scanf("%d",&c);
```

Faire les premier TP de python en C