UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

LIASD
LABORATOIRE INFORMATIQUE AVANCÉE DE SAINT-DENIS

Cours 2

Programmation impérative

Revekka Kyriakoglou

Plan du cours

Variables

Variable

Une variable est le nom donné à une zone de stockage que nos programmes peuvent manipuler. Chaque variable en C a un type spécifique, qui détermine la taille et la disposition de la mémoire de la variable, la gamme de valeurs qui peuvent être stockées dans cette mémoire et l'ensemble des opérations qui peuvent être appliquées à la variable.

Type	Size (bytes)	octect	bits	Format Specifier
char	1	1	8	%c
float	4	4	32	%f
double	8	8	64	%lf
int	4	4	32	%d, %i

- Le mot **bit (b)** vient de l'anglais « binary digit ». Un bit est un nombre binaire qui peut prendre les valeurs **0** ou **1**.
- Un **byte (B)** est composé de **8 bits**.
  Un bit peut prendre 2 valeurs (0 ou 1), donc un byte peut prendre $2^8 = 256$ différents valeurs.
- Un **octet (o)** est 1 byte qui est 8 bits.

void signifie "rien" ou "aucun type". Vous pouvez considérer void comme une absence.
Si une fonction ne renvoie rien, son type de retour doit être void.

Exemple

```c
#include <stdio.h>
int main( void ) {
    int a = 1;
    char b ='1';
    float c = 1.0;

    //print the variables and their sizes
    printf("char␣%c;␣a␣taille␣%lu␣B.\n", b, sizeof(b));
    printf("int␣%d;␣a␣taille␣%lu␣B.\n", a, sizeof(a));
    printf("float␣%f;␣a␣taille␣%lu␣B.\n", c, sizeof(c));

    return 0;
}
```

%lu : long unsigned int

Constante

Constante

Le mot clé **const** devant le type de la variable est utilisé lors de la déclaration pour rendre la variable constante (sa valeur ne peut pas être modifiée).

Une variable constante comporte une valeur **dès sa déclaration** !

Exercise 1

Quelle est l'erreur dans le code suivant ?

```
#include <stdio.h>
int main( void ){
    const int CONSTANTE;
    CONSTANTE = 5;
    printf("CONSTANTE = %d", CONSTANTE);
    return 0;
}
```

Operateurs

Opérateur

Les opérateurs sont des symboles qui permettent de manipuler des variables.
Il y a plusieurs types d'opérateurs :

- Opérateurs de calcul.
- Opérateurs d'assignation.
- Opérateurs d'incrémentation.
- Opérateurs de comparaison.
- Opérateurs logiques.

Il y a deux autres catégories que nous ne verrons pas encore :

- Opérateurs bit-à-bit.
- Opérateurs de décalage de bit.

Opérateurs de calcul

Opérateur	Dénomination	Effet
+	Opérateur d'addition	Ajoute deux valeurs
-	Opérateur de soustraction	Soustrait deux valeurs
*	Opérateur de multiplication	Multiplie deux valeurs
/	Opérateur de division	Divise deux valeurs
=	Opérateur d'affectation	Affecte une valeur à une variable

Exemple

- *2+3=5*

Opérateurs d'assignation et opérateurs d'incrémentation

Opérateur d'assignation	Effet
+=	additionne deux valeurs et stocke le résultat dans la variable (à gauche)
-=	soustrait deux valeurs et stocke le résultat
*=	multiplie deux valeurs et stocke le résultat
/=	divise deux valeurs et stocke le résultat

Opérateur d'incrémentation	Effet
++	augmente d'une unité la variable
–	diminue d'une unité la variable

L'opérateur de type x++ permet de remplacer x=x+1 avec x+=1.

Exemple

Soit int x=5 ;

- *x++ ; donne comme résultat 6*
- *x- - ; donne comme résultat 4*

Exercise 2

Quelle est la valeur de x dans les exemples donnés ci-dessous ?

- x = (2 + 3) * 2 + 3 ;
- x = 2 ; x + = 3 ;
- 9/3

Opérateurs de comparaison

Opérateur	Dénomination	Effet
==	égalité	vérifie l'égalité entre deux valeurs
<	infériorité stricte	vérifie qu'une variable est strictement inférieure à une valeur
<=	infériorité	vérifie qu'une variable est inférieure ou égale à une valeur
>	supériorité stricte	vérifie qu'une variable est strictement supérieure à une valeur
>=	supériorité	vérifie qu'une variable est supérieure ou égale à une valeur
!=	différence	vérifie qu'une variable est différente d'une valeur

Exemple

- *x==3 retourne* 1 *si x est égal à* 3, *sinon* 0.

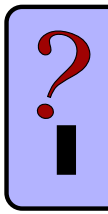
La valeur 1 correspond à **Vrai** et la valeur 0 à **Faux**.

Opérateurs logiques

Opérateur	Dénomination	Effet
||	OU logique	vérifie qu'une des conditions est réalisée
&&	ET logique	vérifie que toutes les conditions sont réalisées
!	NON logique	inverse l'état d'une variable booléenne

Opérateurs logiques

Opérateur	Dénomination	Effet
||	OU logique	vérifie qu'une des conditions est réalisée
&&	ET logique	vérifie que toutes les conditions sont réalisées
!	NON logique	inverse l'état d'une variable booléenne



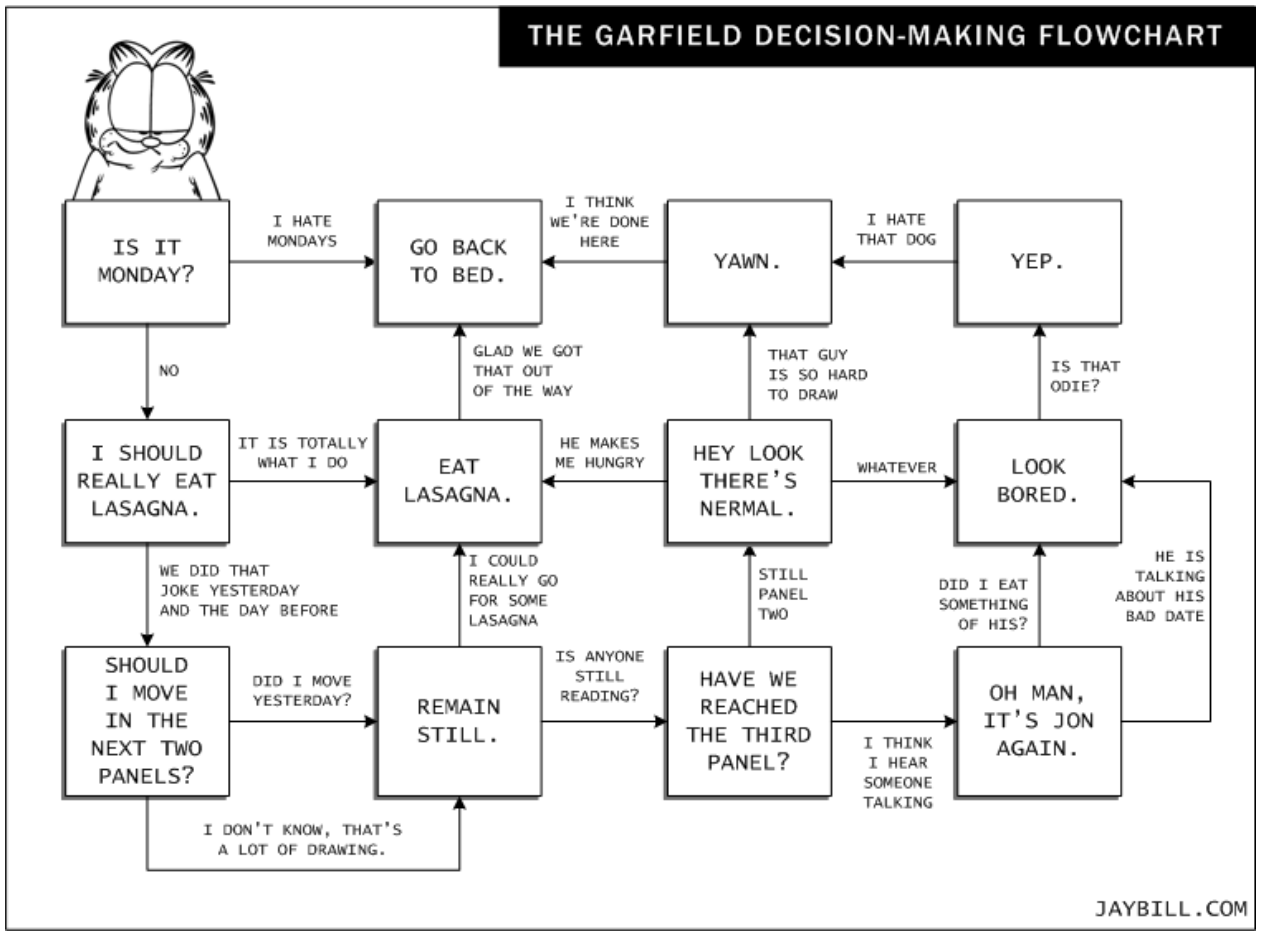
Exercise 3

Essayer de prévoir le résultat et de l'expliquer.

```
#include <stdio.h>
int main( void ){
    printf("1 == 30 vaut %d\n", 1 == 30);
    printf("1 != 30 vaut %d\n", 1 != 30);
    printf("1 < 30 vaut %d\n", 1 < 30);
    printf("1 > 30 vaut %d\n", 1 > 30);
    return 0;
}
```

Structure conditionnelle

On appelle **structure conditionnelle** les instructions qui permettent de tester si une condition est vraie ou non.



if-else

if	si
else	srinon

L'instruction **if-else** permet d'exprimer des prises de décision.

```
if (/* Condition */){
    // Instruction 1
}
else {
    // Instruction 2
}
```

La **condition** est évaluée

- Si elle est **VRAI**, l'instruction 1 s'éxecute.
- Si elle est **FAUSSE**, c'est l'instruction 2 qui s'éxecute.

Chaque **else** s'associe automatiquement au plus proche des if précédents qui n'ont pas encore de else.

```
if ( n > 0 )
    if ( a > b)
        z = a ;
    else
        z = b ;
```

Sinon il faut utiliser des accolades :

```
if ( n > 0 ) {
    if ( a > b)
        z = a ;
}
else
    z = b ;
```

if, else if, else

if	si
else if	sinon si
else	srinon

```
if (/* Condition */){
    // Instruction 1
}
else if (/* Condition */){
    // Instruction 2
}
else {
    // Instruction 3
}
```


Le code de chaque instruction peut être constitué d'une seule instruction ou d'un groupement d'instructions entre accolades.

Exemple

Ce code prend un nombre entier de l'utilisateur et imprime s'il est pair ou impair.

L'opérateur modulo, est représenté par %.

```
#include <stdio.h>
int main( void ){
    int number;
    printf("Entrez un nombre entier: ");
    scanf("%d", &number);

    // Vrai si le reste est 0
    if  (number%2 == 0){
        printf("%d est un nombre pair.", number);
    }
    // Sinon
    else{
        printf("%d est un nombre impair.", number);
    }
```

Instruction switch

switch

Le **switch** est une instruction de prise de décision à choix multiple qui regarde si la valeur d'une expression fait partie d'un certain nombre de **constantes entières**, et effectue les instructions associés à la valeur correspondante.

```c
//Programme 1
int x = 2;

if (x == 1)
    printf("x est 1");
else if(x == 2)
    printf("x est 2");
else if(x == 3)
    printf("x est 3");
else
    printf("no 1,2,3");
```

```c
//Programme 2
int x = 2;

switch(x){
    case 1: printf("x est 1");
            break;
    case 2: printf("x est 2");
            break;
    case 3: printf("x est 3");
            break;
    default: printf("no 1,2,3");
            break;
}
```

Attention :

- Nous ne sommes pas autorisés à ajouter des cas en doublon.
  error : duplicate case value
- Les seules expressions autorisées dans le switch sont les expressions dont le résultat est une valeur **constante intégrale**.
  error : switch quantity not an integer
- La valeur type float n'est pas autorisée comme valeur dans le case.
  error : case label does not reduce to an integer constant
- Les expressions variables ne sont pas autorisées dans les étiquettes de cas (nous verrons plus tard que les macros sont autorisées).
  error : case label does not reduce to an integer constant
- **default** peut être n'importe où dans le switch.