

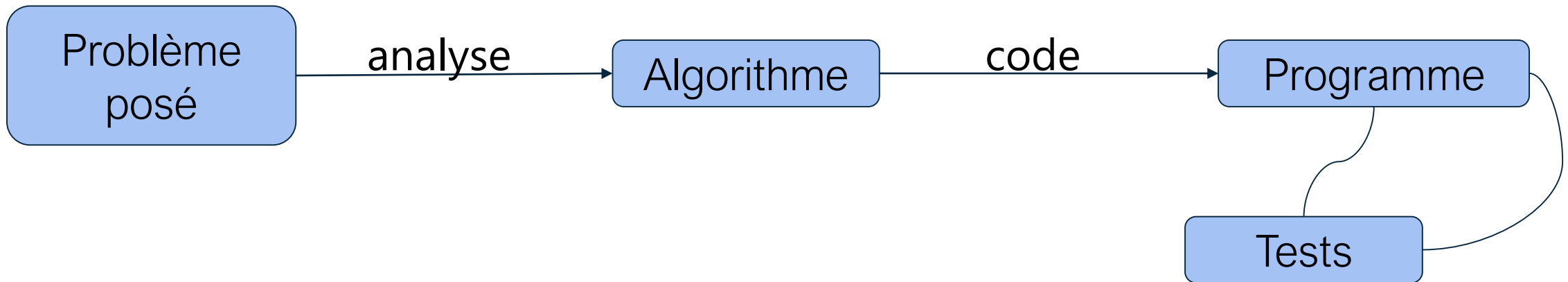
# INFORMATIQUE 1

VII. LE LANGAGE C



# Introduction

L'écriture d'un algorithme, comme celui d'un programme est une étape du développement, non une fin en soi :



Un programme est la traduction (codage) d'un algorithme dans un langage de programmation afin de pouvoir être exécuté sur un ordinateur.

# Introduction

Plusieurs approches de programmation :

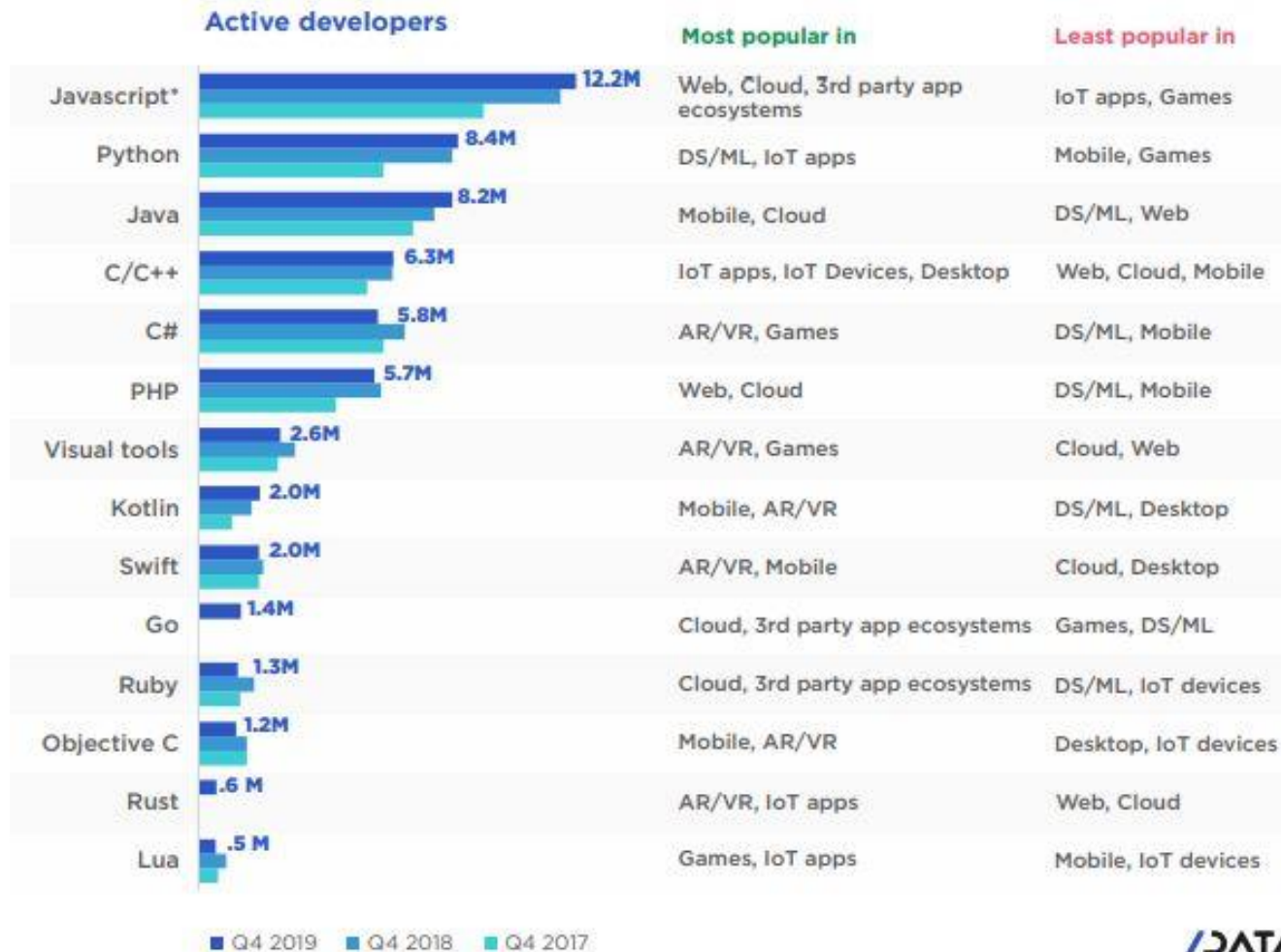
- **fonctionnelle** : appel de fonction mathématiques (Caml, Scheme, Haskell) ;
- **procédurale** : suite d'instructions ou d'appel de fonctions  $\Rightarrow$  automate (**C**, Pascal, Fortan) ;
- **logique** : utilisation de formule logique (Prolog) ;
- **objet** : ensemble d'objets communiquant ensemble (Java, C++, C#).

On distingue deux types de langage selon leur proximité avec la machine:

- Bas niveau
- Haut niveau

# Introduction

Active software developers, globally, in millions Q4 2019 (n=12,066)













(\*) JavaScript includes CoffeeScript, TypeScript

# Le langage C

- Créé en 1972 par Ken Thompson & Dennis Ritchie ;
- Compromis entre le langage de bas niveau (assembleur) et de haut niveau (Fortran) ;
- Rapide : pas de vérification de gestion de la mémoire, elle est à la charge du développeur !
- Faiblement typé : chaque variable doit être typée, mais certaines opérations entre différents types sont possibles (par ex. on peut comparer un entier à un caractère).




# Introduction

| Nov 2021 | Nov 2020 | Change | Programming Language  |                   | Ratings | Change |
|----------|----------|--------|---|-------------------|---------|--------|
| 1        | 2        | ▲      |    | Python            | 11.77%  | -0.35% |
| 2        | 1        | ▼      |    | C                 | 10.72%  | -5.49% |
| 3        | 3        |        |    | Java              | 10.72%  | -0.96% |
| 4        | 4        |        |    | C++               | 8.28%   | +0.69% |
| 5        | 5        |        |    | C#                | 6.06%   | +1.39% |
| 6        | 6        |        |    | Visual Basic      | 5.72%   | +1.72% |
| 7        | 7        |        |    | JavaScript        | 2.66%   | +0.63% |
| 8        | 16       | ▲      |   | Assembly language | 2.52%   | +1.35% |
| 9        | 10       | ▲      |  | SQL               | 2.11%   | +0.58% |
| 10       | 8        | ▼      |  | PHP               | 1.81%   | +0.02% |

Index tiobe

# Le langage C

- Base de très nombreux langage : facile d'apprendre un autre langage si on connaît le C.
  - Permet de comprendre le fonctionnement de l'ordinateur
  - Optimisation de la mémoire utilisée et rapidité
  - Syntaxe stricte
  - Certains concepts difficiles (pointeurs, liste chaînée ect.)
  - Un peu de temps avant de pouvoir produire des choses sympas...
- 
- A teal-colored decorative shape, resembling a quarter-circle or a stylized corner, is located in the bottom right corner of the slide.

# Un exemple

```
/**
 * Mon premier programme
 */
#include <stdio.h>

int main(){
    printf("Hello World");
    return 0;
}
```

```
/**
 * Mon premier programme
 */

DEBUT

    ECRIRE("Hello World")

FIN
```



# Un exemple

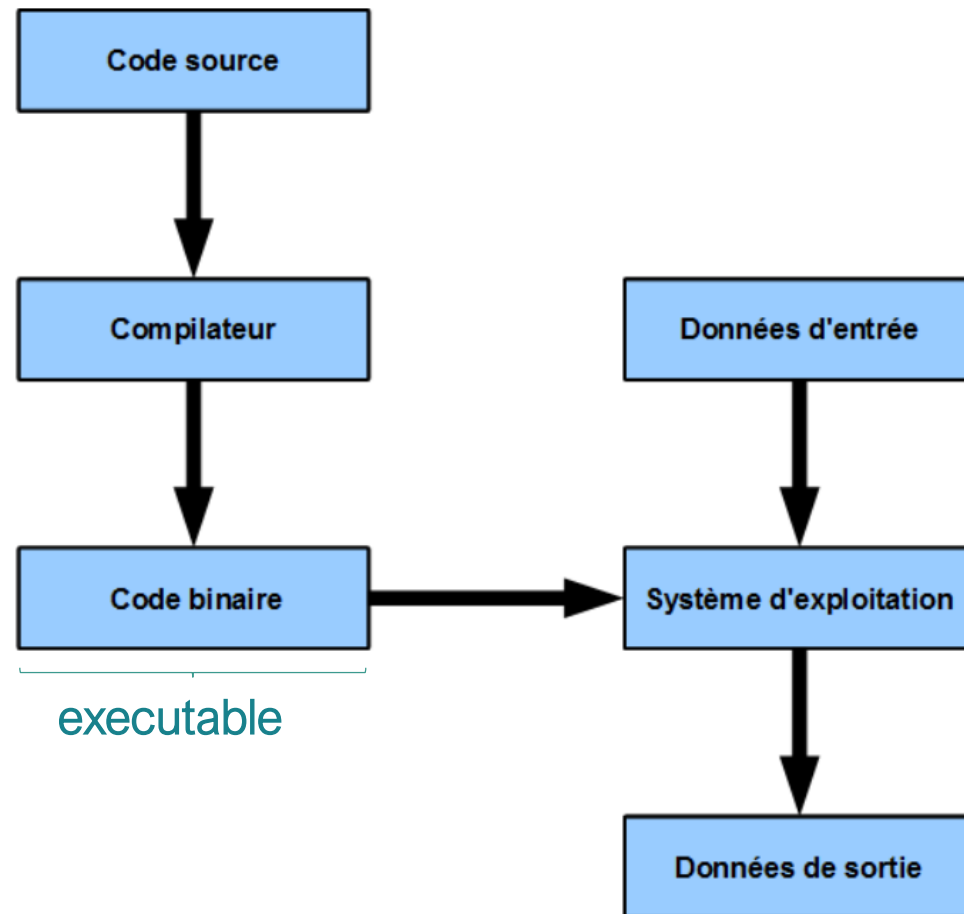
```
/**  
 * Mon premier programme  
 */  
#include <stdio.h>  
  
int main(){  
    printf("Hello World");  
    return 0;  
}
```

```
/**  
 * Mon premier programme  
 */  
  
DEBUT  
    ECRIRE("Hello World")  
FIN
```

Toute instruction se finie par un ;

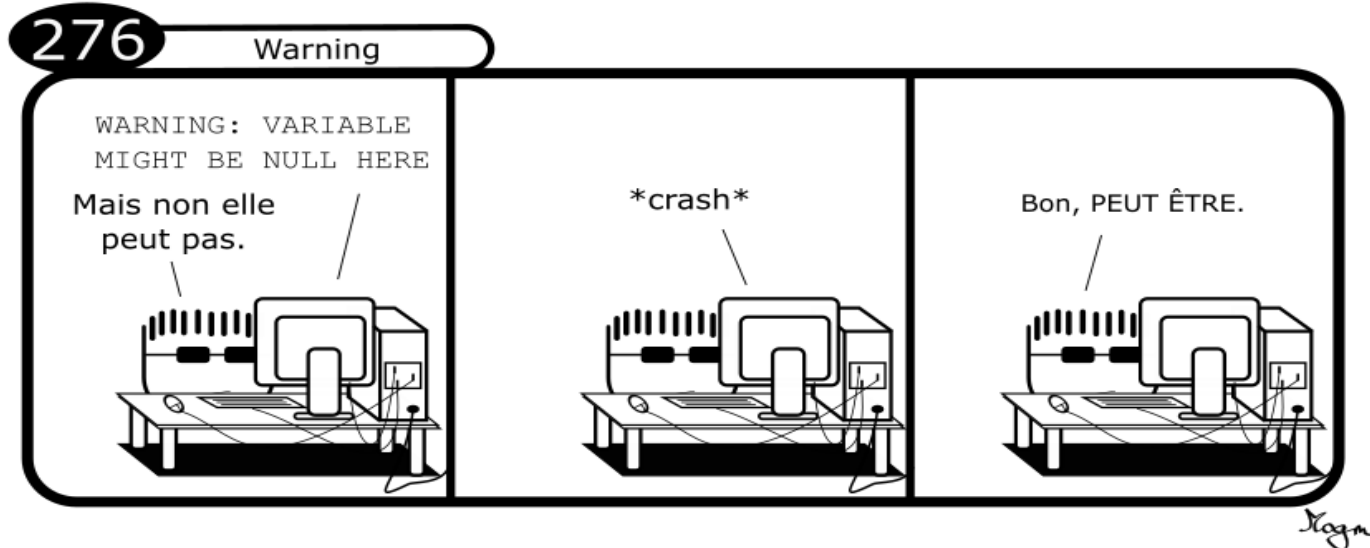
# La compilation

- Le code source (compréhensible par un humain) est enregistré dans un fichier **.c** ;
- C'est un langage **compilé** (vs interprété)



# La compilation

- Lorsqu'il y a des problèmes dans le codes (syntaxes, problème de types ect...), le compilateur peut faire apparaitre des messages :
  - **Des erreurs** : la compilation ne peut pas se faire. Il s'agit souvent d'erreurs de syntaxe (absence de ; parenthèse non fermée ect.)
  - **Des warnings** : la compilation peut se faire mais il est possible qu'il y est un problème à la compilation. Il s'agit souvent d'erreurs de type.



# La compilation

- Lorsqu'il y a des problèmes dans le codes (syntaxes, problème de types ect...), le compilateur peut faire apparaitre des messages :
  - **Des erreurs** : la compilation ne peut pas se faire. Il s'agit souvent d'erreurs de syntaxe (absence de ; parenthèse non fermée ect.)
  - **Des warnings** : la compilation peut se faire mais il est possible qu'il y est un problème à la compilation. Il s'agit souvent d'erreurs de type.
- Ce n'est pas parce qu'il code compile qu'il fonctionne correctement ou qu'il fait ce que vous avez prévu !

# La compilation

- Sous Linux, le **compilateur gcc** s'occupe de traduire le langage C en langage machine. Il produit un fichier exécutable :
  - utilisation : `gcc -o monExecutable monCode.c`
  - exécution : `./monExecutable`
- L'exécution d'un programme signifie l'exécution du programme principale : la fonction **main**.
- Options
  - **-Wall** : Affiche tous les warnings par le compilateur
  - **-Werror** : Transforme tous les warnings en erreurs
  - **-o nomFichierExecutable** : Précise le nom du fichier en sortie
  - Pour plus d'options, voir le manuel de gcc

# Les variables

```
int main() {  
    // déclaration  
    int a, b;  
    float d;  
  
    // affectation  
    a = 12;  
    b = a;  
    d = 3.14;  
  
    return 0;  
}
```

```
VARIABLE  
    a, b : ENTIER  
    d : REEL  
DEBUT  
  
    // affectation  
    a <- 12  
    b <- a  
    d <- 3,14  
  
FIN
```

# Les variables

- Déclaration : nécessite un nom et un type : **type nom;**
  - `int a;`
  - `float x;`
- Affectation : =
  - `a = 1;`
- Attention au nommage de variables !
  - Commence par une **lettre minuscule**
  - Pas de nom sur plusieurs mots
- Il faut déclarer les variables **avant** de les utiliser !

```
int main() {  
    // déclaration  
    int a, b;  
    float d;  
  
    // affectation  
    a = 12;  
    b = a;  
    d = 3.14;  
  
    return 0;  
}
```

# Les variables

- Types de base :
  - char : caractères ASCII                      codé sur 1 octet,
  - int : -2147483648 à 2147483647              codé sur 4 octets,
  - float : (+-) :  $3.4 * 10^{-38}$  à  $3.4 * 10^{38}$       codé sur 4 octets ;
- **Pas de booléen en C** (on gère avec des entiers (0 : faux, autre pour vrai)) ;
- Pas de type *chaîne de caractère* (on verra ça plus tard!)

| Type (pseudo-code) | Type en C |
|--------------------|-----------|
| Entier             | int       |
| Caractère          | char      |
| Réel               | float     |
| “gros” entier      | long      |
| “gros” reel        | double    |
| ...                | ....      |



# Opérations

- Numériques +, -, \*, Division : / (entiers  $\Leftrightarrow$  DIV)
- Modulo : % (entiers  $\Leftrightarrow$  MOD)
- Quelques raccourcis:
  - `a++;`  $\Leftrightarrow$  `a=a+1;`
  - `a--;`  $\Leftrightarrow$  `a=a-1`
  - `a+=5;`  $\Leftrightarrow$  `a=a+5;`
  - `a-=5;`  $\Leftrightarrow$  `a=a-5;`
  - `a*=y`  $\Leftrightarrow$  `a=a*y` ect..

```
int main() {  
    // déclaration  
    int a, b = 0;  
    float d;  
  
    // affectation  
    a++; //a=1  
    b-=a; //b=-1  
    d=a/(2*b); //d=0.5  
    a+=2; //a=3  
  
    return 0;  
}
```

# Entrée / Sortie

```
#include <stdio.h>
int main() {
    // déclaration
    int a,res;
    // affectation
    a = 4;
    res=a*a;
    printf("le carré de a est
%d",res);
    return 0;
}
```

```
VARIABLE
    a, res : ENTIER
DEBUT
    // affectation
    a <- 4
    res <- a*a
    ECRIRE("le carré de a
est"+res)
FIN
```

# Entrée / Sortie

```
#include <stdio.h>
int main() {
    // déclaration
    int a,res;
    // affectation
    a = 4;
    res=a*a;
    printf("le carré de %d est %d", a, res);
    return 0;
}
```

# Entrée / Sortie

- Les sorties en C sont formatées : on sépare les valeurs et la chaîne affichée (la forme et le fond)
  - `printf("coucou");`
  - `printf("carre(%d) = %d", x, x*x);`
- Formats :

| Type (pseudo-code) | Type en C | Format associé |
|--------------------|-----------|----------------|
| Entier             | int       | %d             |
| Caractère          | char      | %c             |
| Réel               | float     | %f             |
| "gros" entier      | long      | %ld            |
| "gros" reel        | double    | %lf            |
| ...                | ....      | ...            |

# Entrée / Sortie

```
#include <stdio.h>
int main() {
    // déclaration
    int a,res;
    // affectation
    printf("saisir une valeur");
    scanf("%d",&a);
    res=a*a;
    printf("le carré de %d est
%d",a,res);
    return 0;
}
```

```
VARIABLE
    a, res : ENTIER
DEBUT
    // affectation
    ECRIRE("saisir une valeur")
    LIRE(a)
    res <- a*a
    ECRIRE("le carré de" +a+
est"+res)
FIN
```

# Entrée / Sortie

- Les entrées en C sont également formatées : on annonce ce que l'on va lire :
  - `scanf("%d", &monEntier);`
  - `scanf("%f", &monReel);`

| Type (pseudo-code) | Type en C | Format associé |
|--------------------|-----------|----------------|
| Entier             | int       | %d             |
| Caractère          | char      | %c             |
| Réel               | float     | %f             |
| "gros" entier      | long      | %ld            |
| "gros" reel        | double    | %lf            |

# Entrée / Sortie

- Pourquoi le symbole & ?
- &nomVariable indique l'**adresse** de la variable dans la mémoire

|   | Adresse                                  | Valeur    |
|---|--|-----------|
|   | 0  | 145       |
| a | 1  | 3.8028322 |
|   | 2  | 0.827551  |
| b | 3  | 3901930   |
|   | ...                                      | ...       |
|   | 3 448 765 900 126<br>(et des poussières) | 940.5118  |

a=3.802 ... &a=1

b=3901930 ... &b=3

# Entrée / Sortie

- Les entrées en C sont également formatées : on annonce ce que l'on va lire :
  - `scanf("%d", &monEntier);`
- Cette commande se traduit donc littéralement par : « **placer la valeur saisie par l'utilisateur dans la case mémoire de monEntier** »
- Il sera interdit :
  - de lire plus d'une variable en une instruction scanf  
`Scanf("%d %d",&a,&b);`
  - de mettre du texte dans les "%..." du scanf ;  
`Scanf("saisir un nombre %d",&a);`



# Les structures conditionnelles

```
if(condition1){  
    instructions1  
}  
else if (condition2){  
    instructions2  
}  
  
else{  
    instruction3  
}
```

```
SI (condition1) ALORS  
    instructions1  
SINON SI (condition2) ALORS  
    instruction2  
SINON  
    instruction3  
FIN SI
```

# Les conditions

- Les opérateurs de comparaison :

- < , > , <= , >=, !=, ==

- Les opérateurs logiques :

- ET : &&

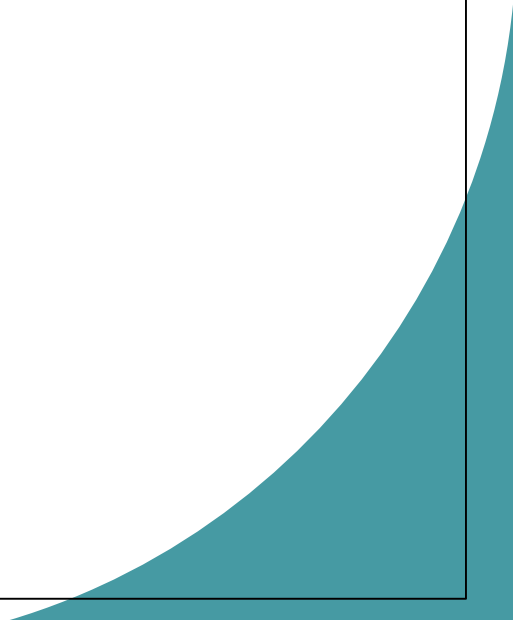
- OU : ||

- Exemple: `a<=2 || b !=5 && c==10`

# Les structures conditionnelles

```
switch (a) {  
    case v1 :  
        ...  
    break;  
    case v2 :  
        ...  
    break;  
    case v3 :  
        ...  
    break;  
    default :  
        ...  
    break;  
}
```

```
Selon(a)  
    cas v1 : ...  
    cas v2 : ...  
    cas v3 : ...  
        default : ...  
Fin selon
```



# Les structures conditionnelles

```
switch (a) {  
    case v1 :  
        ...  
        break;  
    case v2 :  
        ...  
    case v3 :  
        ...  
        break;  
    default :  
        ...  
        break;  
}
```

Si on écrit pas le break, on fait également les cas suivants jusqu'à un break.

# Répétitions : Tant que

Tant que (condition) faire  
...  
Fin tant que

Répéter  
...  
Tant que (condition)

```
while (condition) {  
    ...  
}
```

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

# Répétitions : Pour

```
Pour i de 1 à n faire  
    ...  
Fin pour
```

```
for (i=1; i<n; i++) {  
    ...  
}
```

```
Pour i de 1 à n+1 par pas de 2  
faire  
    ...  
Fin pour
```

```
for (i=1; i<=n; i=i+2) {  
    ...  
}
```

# Répétitions : Pour

- Syntaxe :

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

```
for (i=1; i<=n; i=++) {  
    instructions;  
}
```

- Déroulement :
  1. exécute **initialisation**
  2. vérifie **cond**, si faux -> sortie
  3. exécute **instructions**
  4. exécute **iteration**
  5. retour à 2

# Répétitions : Pour

- Syntaxe :

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

```
for (i=1; i<=n; i=++) {  
    instructions;  
}
```

- Déroulement :
  1. exécute **initialisation**
  2. vérifie **cond**, si faux -> sortie
  3. exécute **instructions**
  4. exécute **iteration**
  5. retour à 2




# Répétitions : Pour

- Exemple

# Procédures

```
void proc1(int a, float b) {  
    instructions  
}
```

```
Procédure proc1(a: entier,  
b: réel) {  
    instructions  
}
```

A teal-colored decorative shape, resembling a quarter-circle or a stylized wave, is located in the bottom right corner of the slide.

# Procédures

- Utilisation du mot clé **void**

- Syntaxe

```
void nomProc(paramètres) {  
    instructions ;  
}
```

```
void afficher(int n) {  
    int i ;  
    for (i=1; i<=n; i++) {  
        printf("%d", i) ;  
    }  
}
```

# Fonctions

```
int ma_fonction(int a, float b) {  
    return () ;  
}
```

```
Fonction ma_function(a: entier,  
b:réel): entier  
DEBUT  
    instruction  
    RETOURNER ()  
FIN
```

# Fonctions

- Syntaxe

```
typeRetourné nomFonction(paramètres) {  
    ...  
    return()  
}
```

```
int factorielle(int n) {  
    int f = 1 ;  
    int i ;  
    for (i=1; i<=n; i++) {  
        f = f * i;  
    }  
    return f ;  
}
```

```
int fact (int n) {  
    if (n == 0) {  
        return 1 ;  
    }  
    else {  
        return n * fact (n-1) ;  
    }  
}
```

# Fonctions

- Syntaxe

```
typeRetourné nomFonction(paramètres) {  
    ...  
    return()  
}
```

ON NE RETOURNE QU'UNE VARIABLE !

```
int factorielle(int n) {  
    int f = 1 ;  
    int i ;  
    for (i=1; i<=n; i++) {  
        f = f * i;  
    }  
    return f ;  
}
```

```
int fact (int n) {  
    if (n == 0) {  
        return 1 ;  
    }  
    else {  
        return n * fact (n-1) ;  
    }  
}
```

# La fonction main

- En C le programme principal est vu comme une fonction/procédure.
- C'est cette fonction qui sera parcouru lors de l'exécution du programme.

Mode fonction :

```
int main(){  
    return 0 ;  
}
```

Mode procédure:

```
void main(){  
}
```

# Prototypage

- Le compilateur C lit les fichiers source du haut en bas. Il constatera donc une **erreur** si une fonction/procédure est déclarée **APRÈS** sa première utilisation. L'exemple suivant provoque une erreur de compilation.

```
#include <stdio.h>
int main () {
    printf("%d",toto(3)) ;
    return 0 ;
}

int toto (int n) {
    return 2 * n ;
}
```

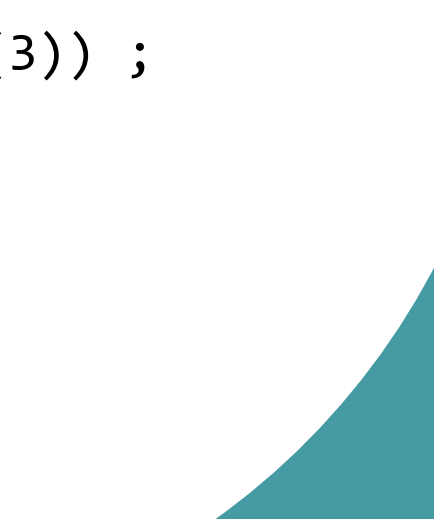


# Prototypage

- Solutions :
  - déclarer les fonctions dans le bon ordre ;

```
int toto (int n) {  
    return 2 * n ;  
}  
  
int main () {  
    printf("%d",toto(3)) ;  
    return 0 ;  
}
```

```
int toto (int n) ; // prototypage !  
  
int main () {  
    printf("%d",toto(3)) ;  
    return 0 ;  
}  
  
int toto (int n) {  
    return 2 * n ;  
}
```



# Prototypage

- Solutions :
  - déclarer les fonctions dans le bon ordre ;

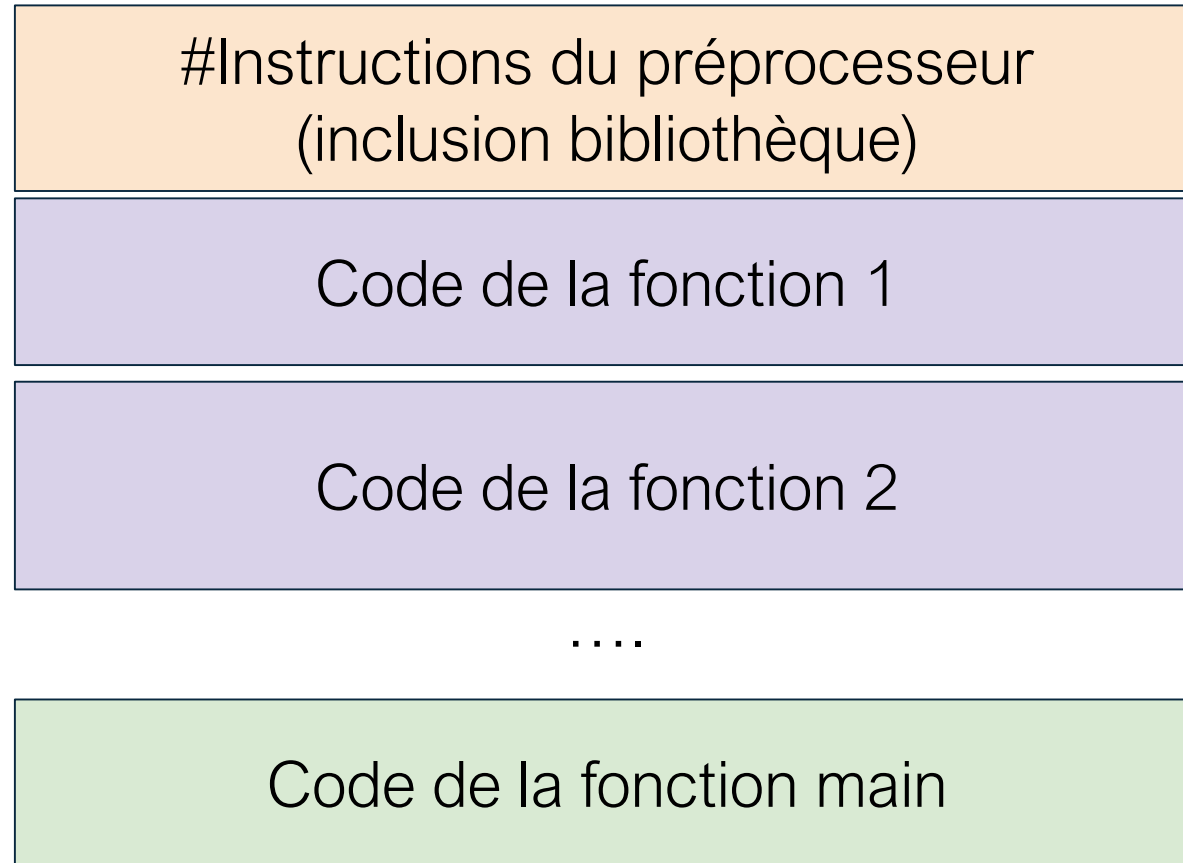
```
int toto (int n) {  
    return 2 * n ;  
}  
  
int main () {  
    printf("%d",toto(3)) ;  
    return 0 ;  
}
```

```
int toto (int n) ; // prototypage !  
  
int main () {  
    printf("%d",toto(3)) ;  
    return 0 ;  
}  
  
int toto (int n) {  
    return 2 * n ;  
}
```

# Bibliothèques

- Certaines fonctions déjà écrites se trouvent dans des **bibliothèques** qu'il faut inclure au début du code.
- Exemples:
  - `printf` fait partie de `stdio.h`
  - `sqrt`, `pow` font partie de `math.h`
  - `srand` fait partie de `time.h`
- Pour inclure une bibliothèque on écrit au début du programme
  - `#include<nomBibliothèque.h>`
- `stdio.h` est très utile : on l'inclura tout le temps !

# Structure générale d'un fichier .c



# Commentaires

- **LE CODE DOIT ÊTRE COMMENTÉ**
  - Pour vous, pour les autres
  - Normalisation pour exporter une documentation “propre”
- Un commentaire sur une ligne est précédé de `//`
  - `//Ceci est un commentaire`
- Un commentaire sur plusieurs ligne est écrit entre `/*` ... `*/`
  - `/* Ceci est un commentaire`  
`sur plusieurs lignes */`

# Example

```
int main() {  
    int i,n ;  
    n=4;  
    for (i=1; i<=n; i++) {  
        printf("%d", i) ;  
    }  
    return 0;  
}
```

# Example

```
→ int main() {  
    int i,n ;  
    n=4;  
    for (i=1; i<=n; i++) {  
        printf("%d", i) ;  
    }  
    return 0;  
}
```

| n | i |
|---|---|
|   |   |

# Example

```
int main() {  
    int i,n ;  
    → n=4;  
    for (i=1; i<=n; i++) {  
        printf("%d", i) ;  
    }  
    return 0;  
}
```

| n | i |
|---|---|
| 4 |   |



# Example

```
int main() {  
    int i,n ;  
    n=4;  
    → for (i=1; i<=n; i++) {  
        printf("%d", i) ;  
    }  
    return 0;  
}
```

| n | i |
|---|---|
| 4 | 1 |

# Exemple

```
int main() {  
    int i,n ;  
    n=4;  
    for (i=1; i<=n; i++) {  
→      printf("%d", i) ;  
    }  
    return 0;  
}
```

| n | i |
|---|---|
| 4 | 1 |

Ecran

1

# Exemple

```
int main() {  
    int i,n ;  
    n=4;  
    → for (i=1; i<=n; i++) {  
        printf("%d", i) ;  
    }  
    return 0;  
}
```

| n | i |
|---|---|
| 4 | 2 |

Ecran

1

# Exemple

```
int main() {  
    int i,n ;  
    n=4;  
    for (i=1; i<=n; i++) {  
→      printf("%d", i) ;  
    }  
    return 0;  
}
```

| n | i |
|---|---|
| 4 | 2 |

Ecran

1 2

# Exemple

```
int main() {  
    int i,n ;  
    n=4;  
    → for (i=1; i<=n; i++) {  
        printf("%d", i) ;  
    }  
    return 0;  
}
```

| n | i |
|---|---|
| 4 | 3 |

Ecran

1 2

# Exemple

```
int main {  
    int i,n ;  
    n=4;  
    for (i=1; i<=n; i++) {  
→      printf("%d", i) ;  
    }  
    return 0;  
}
```

| n | i |
|---|---|
| 4 | 3 |

Ecran

1 2 3

# Exemple

```
int main() {  
    int i,n ;  
    n=4;  
    → for (i=1; i<=n; i++) {  
        printf("%d", i) ;  
    }  
    return 0;  
}
```

| n | i |
|---|---|
| 4 | 4 |

Ecran

1 2 3

# Procédures

```
int main() {  
    int i,n ;  
    n=4;  
    for (i=1; i<=n; i++) {  
→      printf("%d", i) ;  
    }  
    return 0;  
}
```

| n | i |
|---|---|
| 4 | 4 |

Ecran

1 2 3 4



# Exemple

```
int main() {  
    int i,n ;  
    n=4;  
    → for (i=1; i<=n; i++) {  
        printf("%d", i) ;  
    }  
    return 0;  
}
```


| n | i |
|---|---|
| 4 | 5 |

Ecran

1 2 3 4

# Exemple

```
int main() {  
    int i,n ;  
    n=4;  
    for (i=1; i<=n; i++) {  
        printf("%d", i) ;  
    }  
    return 0;  
}
```



| n | i |
|---|---|
| 4 | 5 |

Ecran

1 2 3 4

# Un exemple pratique

