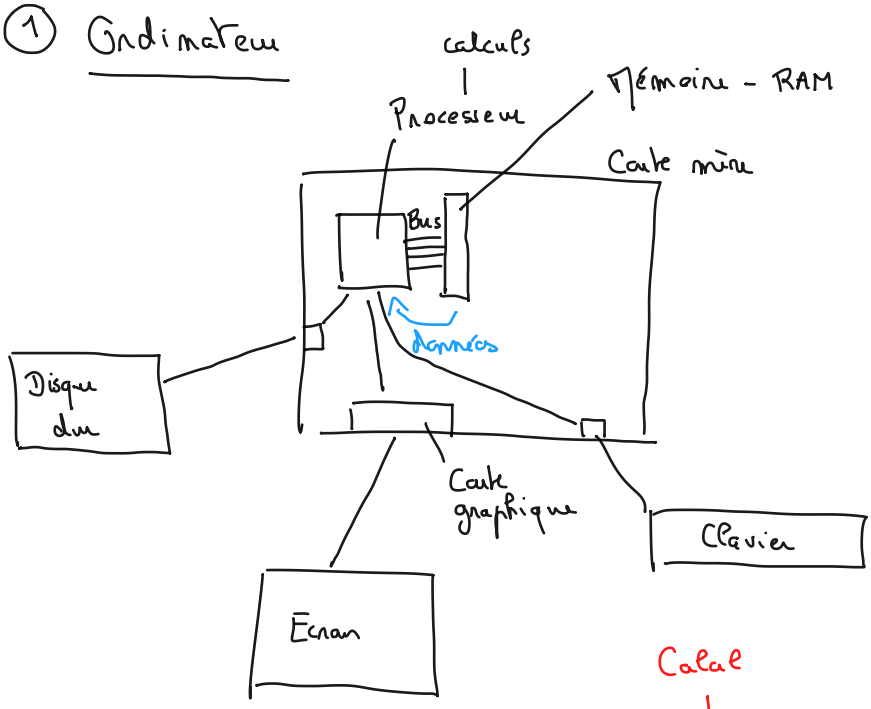
§1. Bases de la prog

I. Intro. aux ordinateurs

① Ordinateur



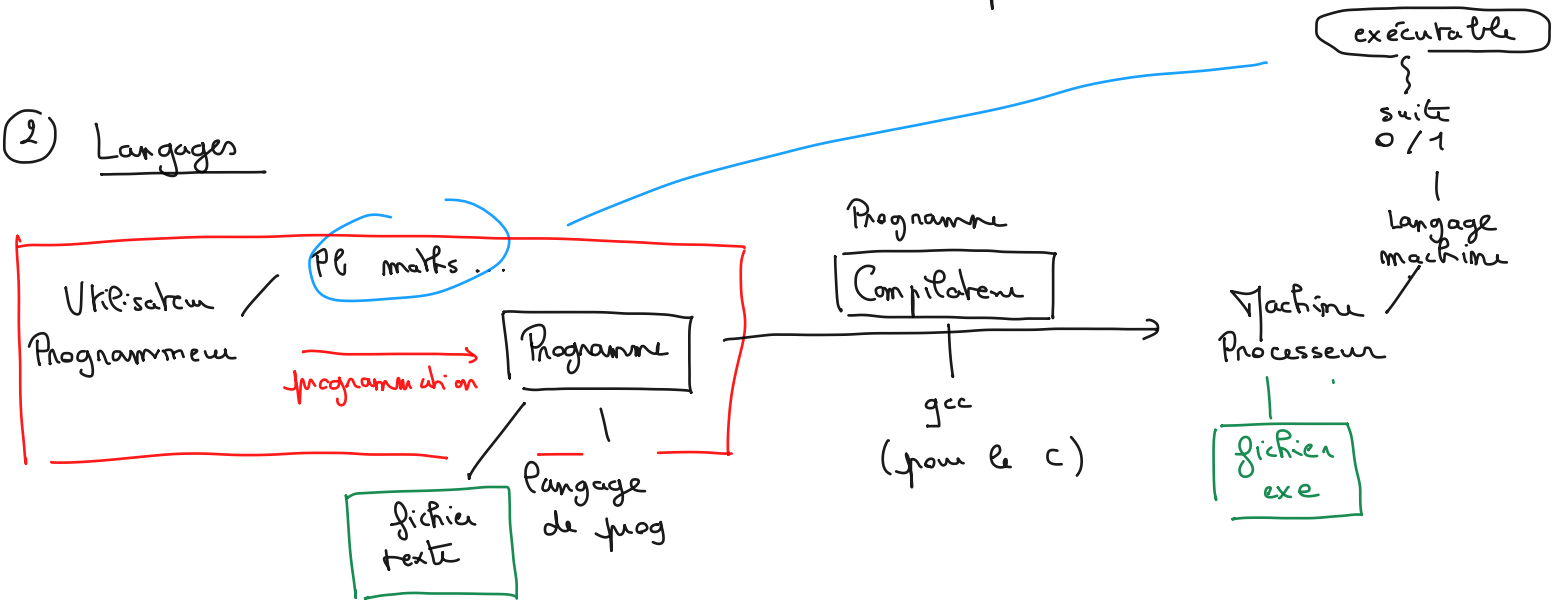
Calcule ← ✓ Carte graphique
✓ Ecran
✓ Clavier
Données ← ✓ Mémoire - RAM
Calcule ← ✓ Processeur
✓ Carte mère
Données ← ✓ Disque dur

Calcul

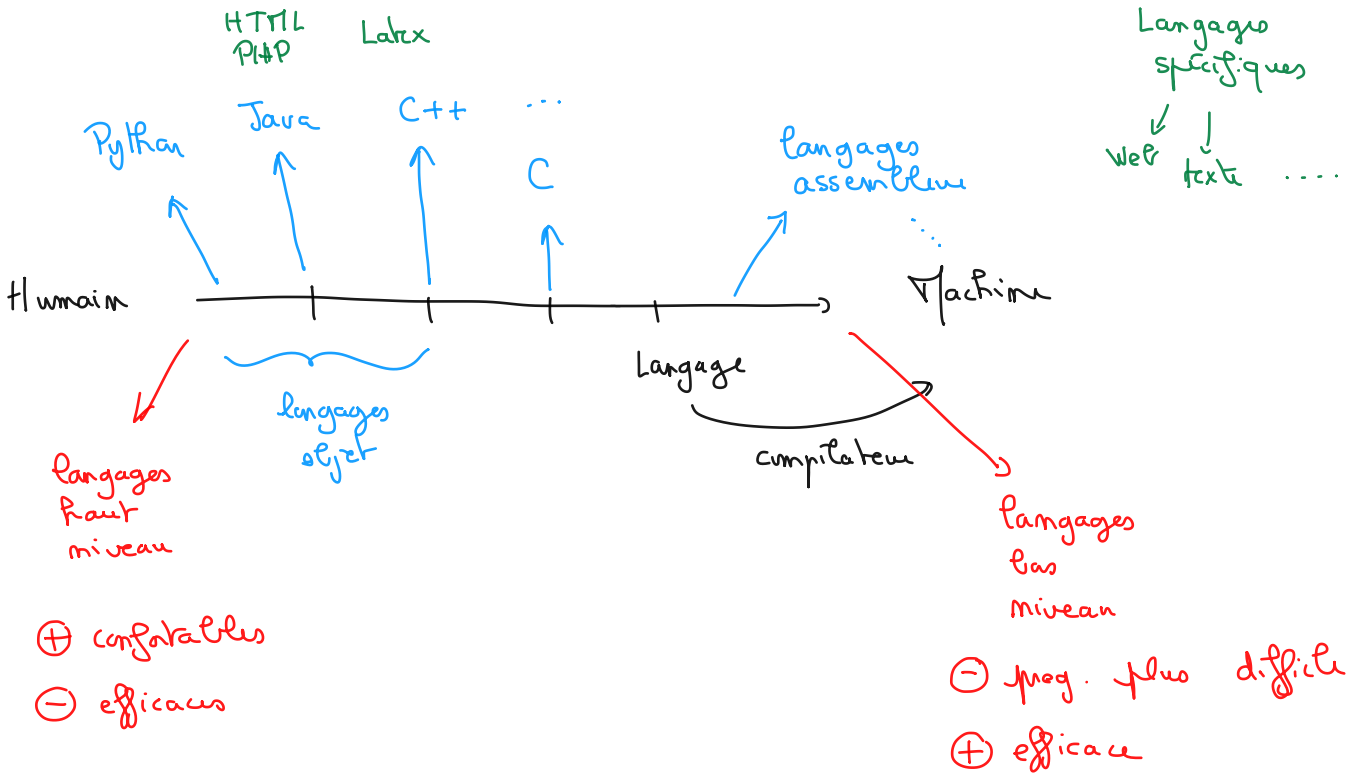
Données : → mémoire RAM (+ rapide, − court terme volatile) | limité
↳ changement
↳ disque dur (+ stockage important | − non volatile, plus lent)

programmation : gestion mémoire / RAM

② Langages



C / C++ ....
Langages compilés ≠ interprétés



II. Bases de la programmation impérative

Langages qui s'inspirent des instruction machine

→ suite d'instructions

→ exécuté ligne par ligne

≠

prog. fonctionnelle
OCAML

prog. par contrainte
PROLOG

Prog impérative

→ calculs → structures ~~instructions~~ de contrôle / if for while

→ données — machine — contraintes sur données

- type
  - stockage (taille 2)
    - flottants
    - entiers ≠ relatifs (≠ caractères)
  - taille
- stockage variables

type

① Données

Types → valeurs

Booléens

99 % Langages

1) caractères (char) 'a' 'b' ...

2) entiers 1 10 -5

3) réels 1.2 -3.7 ~~0.000003~~ 3e-6 $(3 \cdot 10^{-6})$

+ Langages haut niveau (pas en C)

chaînes de caractères (string) "abc"

② Stockage des données

↓

dans des **variables** — case dans la mémoire — baptisée par un nom i, j ....

réservée d'une taille correspondant au type stocké

déclaration d'une variable (C / C++)

nbre de 0/1

Taille — type

32 bits — entiers (int)

réels

32 bits → float

64 bits → double ← calcul scientifique

$x = 10^m$

$\frac{1}{x} + \dots + \frac{1}{x} = 1$ (10^m fois)

Pseudo-code → section Variables au début où on liste toutes les variables / type + à quoi elles servent

③ Structures de contrôle → contrôle l'enchaînement des calculs

- conditionnelles
- boucles
  - pour
  - tant que

a) Conditionnelles — si / alors / sinon

Syntaxe

```
si (cond)
    | code si la cond est vraie
sinon
    | code si la cond est fausse
fsi
```

ex: calcule le max de 2 nombres entiers stockés ds x, y → stocker le résultat dans res

↳ idée :

```
si (x ⩾ y) : max : x
sinon      : max : y
```
→ résultat ?

Variables

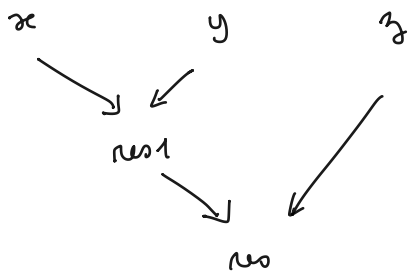
```
entiers : x, y   // données
entiers : res    // pour le résultat
```
(// : commentaire)

Pseudo-code

```
si (x ⩾ y)
    res ← x
sinon
    res ← y
fsi
```

ex2 : calculer le max de 3 nombres réels x, y, z → résultat dans res

Idée :



7 9 10

Variables réels : x, y, z // données

~~réel : res1 // résultat intermédiaire~~

réel : res // ~~résultat final~~ plus grand nombre rencontré à un moment donné → fin : résultat

Pseudo-code

```
si (x ≤ y)
    res ← y
sinon
    res ← x
fsi
si (res ≤ z)
    res ← z
sinon
    res ← res1
fsi
```

e) Boucles : pour / tantque → répéter du code un certain nombre de fois

Pour : répéter le code avec une variable variant de val 1 → val_finale

≠

Tantque : répéter le code tant qu'une condition est vraie

↳ nombre de fois non connu à l'avance

n fixé

$\sum_{i=0}^{n} i^2$ — pour

$\prod_{i=1}^{n} i$ ↳ $n!$ — pour

$\lfloor x \rfloor$ ↳ partie entière du réel x — tantque

chercher si $x \in$ tableau — tantque

Syntaxe

```
pour var de v1 à v2 (pas ...)
{ ⋮ code à répéter
fpour

tantque ( cond )
{ ⋮ code à répéter
ftq
```

ex 1/2 : $\sum_{i=0}^{n} i^2$ / $\prod_{i=1}^{n} i = n!$ — n fixé

acc : $0 + 1^2 + 2^2 + 3^2 + \dots + n^2$

i : ↑ .....

Variables

entier i * // var. incrémentée → entier n donné ....

entier acc // résultat partiel (quand i = n ⇒ résultat) **

Pseudo-code

```
acc ← ∅ (1)   **
pour i de ∅ (1) à n   *
{ acc ← acc + i²
  acc * i
fpour
```

Règle 1
Toute variable doit être initialisée

ex 3 : x donné (≥ 0) : calculer $\lfloor x \rfloor$

$x = 3{,}27$ ~~3,27~~

10,525

i → 0, 1, 2, 3, ... 4 > x → tantque

Variables

réel x (donné)

entier i

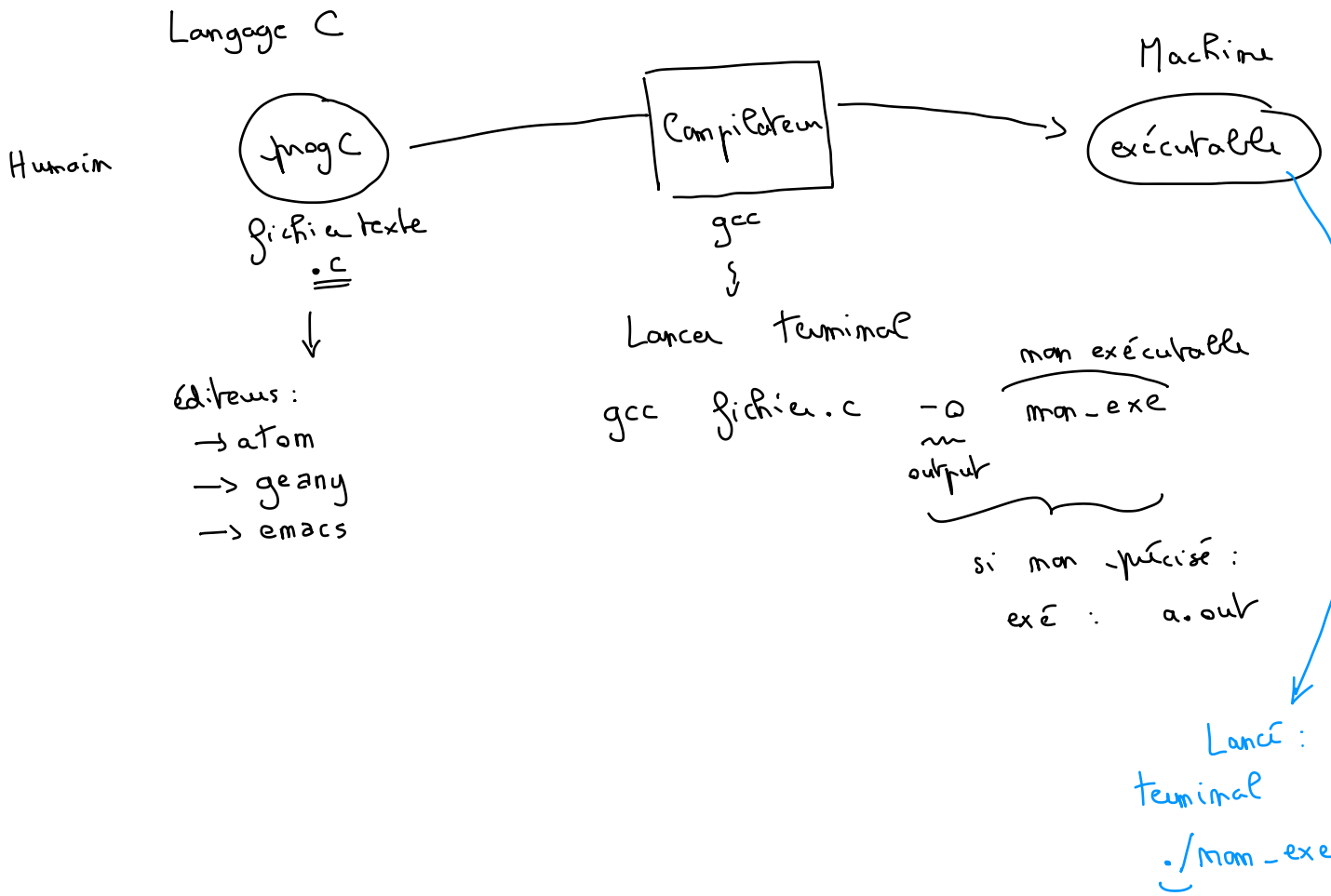
9

Pseudo-code

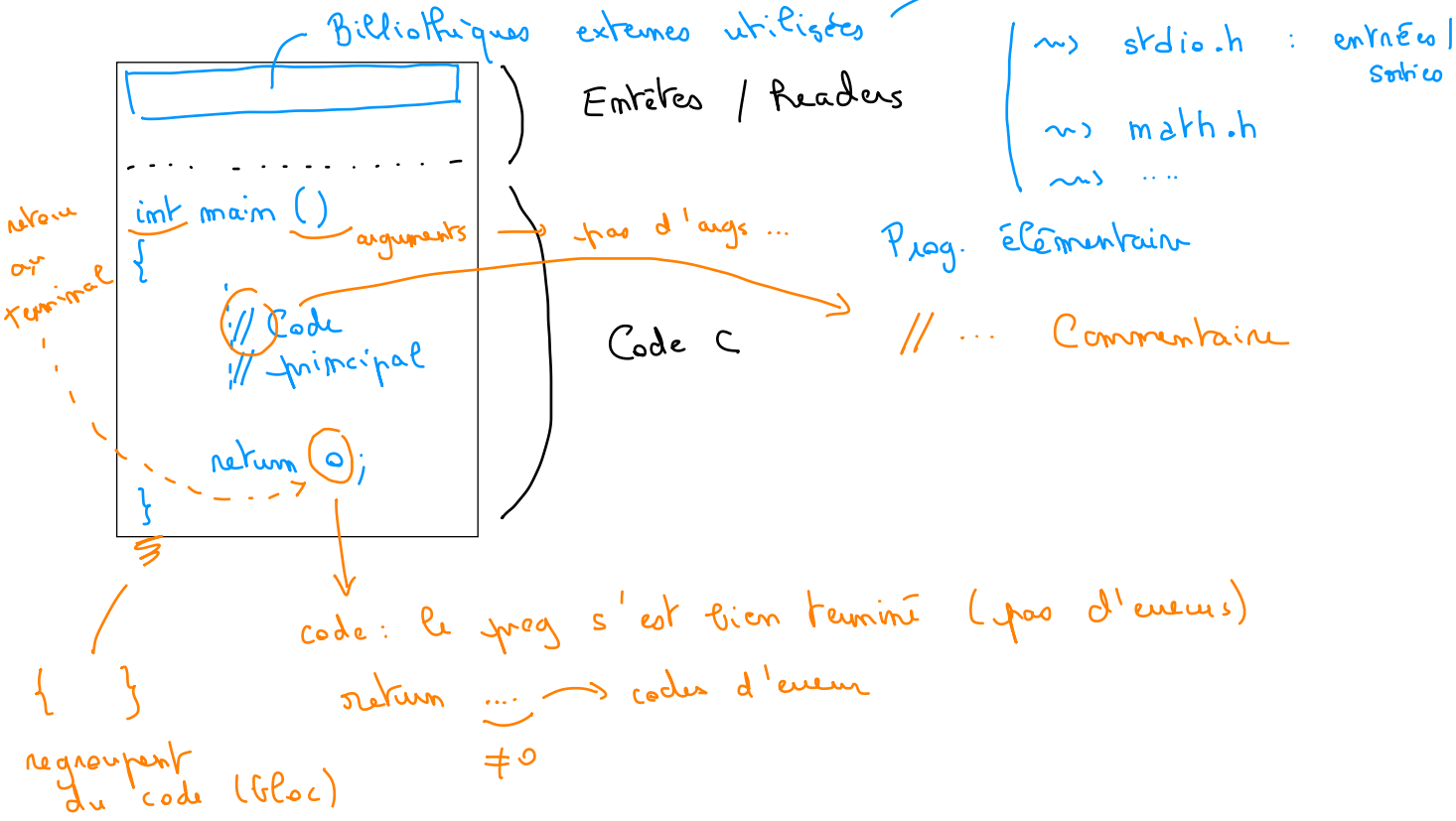
```
init i ?
tantque ( i ≤ x)
| i ← i+1
ftq
res ← i-1
```

III. Bases du C

① Logistique de programmation



② Structure d'un prog C (v.1)



③ Variables et types

Stocker des données —codage→ machine ←décodage—

- représentées en base 2
- taille finie et déterminée à l'avance

1 encodage par type de données

11 (entier) → 1011 (base 2)

- 11 : mod 2 → 1 ; /2 → 5
- 5 : mod 2 → 1 ; /2 → 2
- 2 : mod 2 → 0 ; /2 → 1
- 1 : mod 2 → 1 ; /2 → 0 ×

−11 ?

Chaque donnée — associée à un type → encodage, → taille mémoire

explicite ↓

déclarer toutes les variables en début du main

Types en C :

→ caractère : char (1 caractère)

valeurs :
- 'a' 'b' .....
- '\...' caractères spéciaux :
  - '\n' new line (saut à la ligne)
  - '\t' tabulation (alignement en colonnes de 8 caract)

→ entiers : int (+ ....)

(codé sur 32 bits)

valeurs | 1 −5 157 ....

→ réels : float / double

pour tous les calculs scientifiques

float → 32 bits

double → 64 bits

nbre de chiffres représentables après la virgule ...

1.57 −3.27

$3{,}25 \cdot 10^{15}$ → 3.25 e15

0,0000001 → 1e−5

- ~~Chaînes de caractères~~ string codées → tableau de caractères (§2) + ...

Syntaxe pour écrire des chaînes : "abc ..." ↳ fabrique ce tableau ...

- ~~Booléens~~ codés → entiers | FAUX : 0 | VRAI : $i \neq 0$ (quelconque)

Opérations

int / float double

\+ − *

/ sur réels : division

sur entiers : division euclidienne (quotient)

17 / 4 ⇝ 4

% reste de la div. euclidienne

op. booléennes (logiques)

ET	&&
OU	||
≠	!=
EGAL	==
NON	!

& | : existent mais pas op. logiques (op. bit à bit...)

⚠ 1 / 3 ⟶ div. eucl. ⇝ 0

1.0 / 3 ou 1 / 3.0

L'exposant n'existe pas

~~**~~ ~~*~~ ⚠ xor en binaire

\ pas exp

Variables ⟶ emplacement de stockage de données en mémoire

⟶ typées

⟶ nom associé ⇒

↳ toutes les variables sont déclarées avec leur type

↳ début du main (bloc)

⚠

Syntaxe (s) | type nom ; ← en C ; finit toutes les instructions

ex:
```
int i;
float x;
```

Déclarations multiples:

| type nom1, nom2, ... ;

ex: `int i, j;`

Déclaration et init:

| type nom = val, ... ;

ex: `int i=0, j, k=3;`

Affectation: = ⚠

(4) Structures de contrôle

a) Si ...

Syntaxe :

```
if ( cond )
{
    // code si vrai
}
else
{
    // code si faux
}
```

(else : optionnelle)

b) Boucle tantque

Syntaxe :

```
while ( cond )
{
    // code à répéter
}
```

c) Boucle pour

Syntaxe :

```
for ( init ; cond. tantque ; incrémentation )
{
    // code
}
```

⋮

~~i <= 16~~

i < 16

pour i de 1 à 15 → for(i=1 ; i<=15 ; i=i+1)

pour j de 20 à 0 décr. → for(j=20 ; j>=0 ; j=j-1)

pour k ∈ {nombres impairs entre 1 et 20} → for(k=1 ; k<=20 ; k=k+2)

(5) Entrées / sorties

a) Affichage à l'utilisateur

Pb : calcul → résultat ds res

on veut afficher : le resultat est (texte pas fixe) 1.5 ← val. de res ....

- le resultat est → texte fixe
- 1.5 → contenu de variables

Syntaxe

```
printf( " ....\n" , var1, .... );
```

chaîne format décrit

Chaîne format :
- → texte fixe : écrites tel que
- → contenu de variable à insérer :

%... (... = type) →
- %c : caractère
- %d : entier
- %f : float
- %g : double
- ⋮

ex. précédent :

```
printf("Le resultat est : %f\n", res);
printf(" i = %d , j = %d\n", i, j );
```

b) Saisie de valeurs au clavier

```
scanf( " ... " , &var );
```

chaîne format

⚠ pas \n

& obligatoire avant le nom des variables (cf. §4 pointeurs)

Lire un entier ↘ stocké ds i :

```
scanf("%d", &i);
```

rem:

```
scanf("%d/%d/%d", &j, &m, &a);
```

entier / entier / entier