



Programmation et algorithmique

IN101

ENSTA Paris - TC 1ère année

François Pessaux

U2IS

2022-2023

`prenom.nom@ensta-paristech.fr`

Spécification

- **Spécification** : expression du **besoin**.
- \exists plusieurs formes de spécifications :
 - ▶ Simple formule mathématique : **expression** directement implantable dans un langage.
 - ▶ Formule mathématique plus complexe nécessitant une **mise en forme** « informatique ».
 - ▶ Problème décrit en langage **naturel** et **informel**.
- +/- besoin de **modéliser** et **raffiner**

« Écrire un programme permettant de convertir une température en degrés Fahrenheit vers des degrés Celcius. »

- $x \text{ }^{\circ}\text{F} \longrightarrow \frac{5}{9}(x - 32) \text{ }^{\circ}\text{C}.$
- Expression arithmétique \Rightarrow direct en C.

```
float ftoc (float t) {  
    return (5.0 / 9.0 * (t - 32.0)) ;  
}
```

- Traduction simple.

« Écrire un programme permettant de calculer $f(n)$ sachant que : »

$$\begin{cases} f(0) &= 1 \\ f(n) &= \prod_{k=1}^n k \end{cases}$$

- Deux cas dans la définition.
 - ▶ Faire explicitement 2 cas ?
 - ▶ Possibilité d'un seul cas général ?
- Répétition : introduction d'un mécanisme d'itération.
 - ▶ Faire une boucle ?
 - ▶ Faire de la récursion ?

```
int f (int x) {  
    int res = 1 ;  
    while (x > 1) {  
        res = res * x ;  
        x-- ;  
    }  
    return res ;  
}
```

```
int f (int x) {  
    if (x <= 0) return 1 ;  
    return (x * f (x - 1)) ;  
}
```

Spécification carrément pas triviale

« Écrire un programme permettant de calculer les déformations d'une structure. »

« Écrire un programme permettant de naviguer sur le WEB. »

« Écrire un programme permettant de simuler les cours de la bourse. »

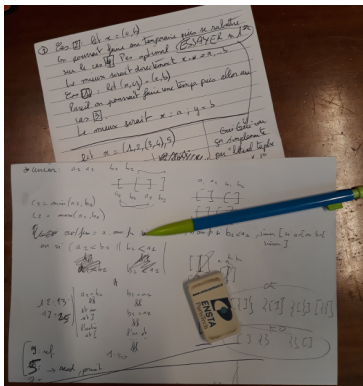


- **Modéliser** le problème.
- **Décomposer** en sous-problèmes.

Concevoir l'algorithme : choses à faire pour commencer

- Ne pas sauter sur le clavier.
- Ne pas penser en terme de langage de programmation.
- Étudier comment on (pas la machine) le ferait à la main.
- Faire des exemples.
- Faire des dessins, des schémas.

Meilleurs outils :



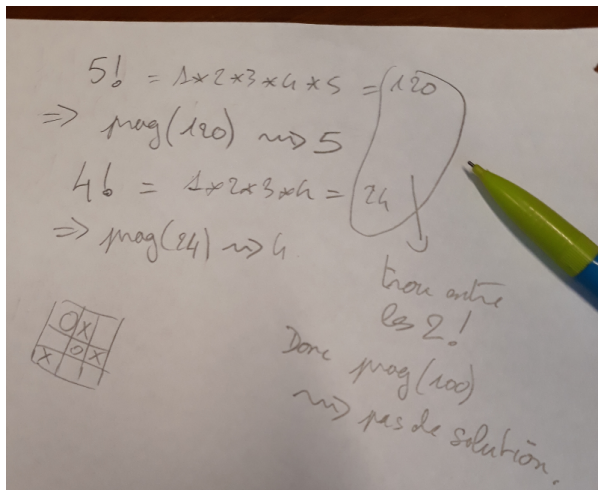
Un premier exemple

« Écrire un programme qui prend un entier et dit de quoi sa valeur absolue est la factorielle. »

- Formuler le problème plus **clairement** :
 - ▶ « Soit $x \in \mathbb{Z}$, trouver y tel que $y! = |x|$. »
 - ▶ C'est en fait la fonction inverse de factorielle qui est demandée.
- Mes **entrées** : un nombre **entier** positif, nul ou négatif.
- Mes **sorties** : un nombre entier **positif**.

Sûr ?

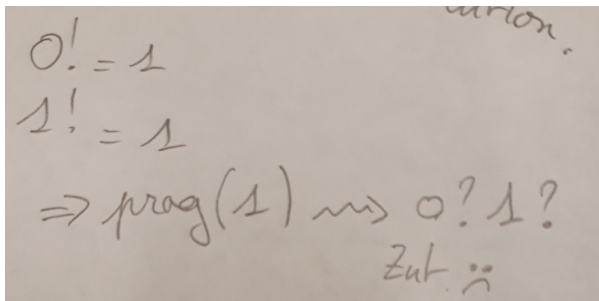
Un peu de papier



- Cas où pas de solution.
- Mes **sorties** : un nombre entier **positif** ou une **erreur**.

Encore un peu de papier

- On refait quelques essais...

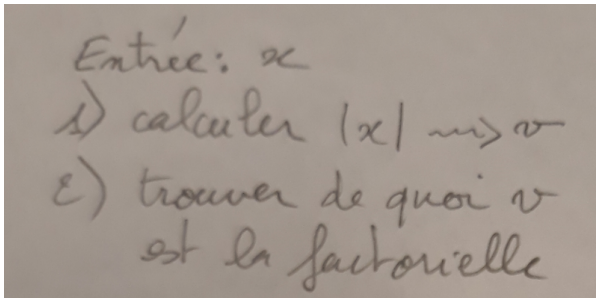


- Même pas une **fonction** en fait : cas avec **2** résultats possibles.
- Prévoir une autre **erreur** si entrée = 0, 1 ...ou -1.

Et maintenant quel algorithme ?

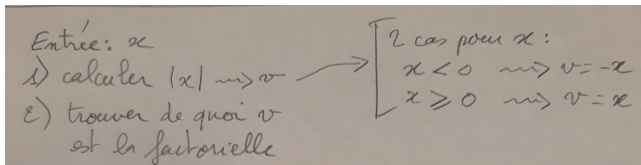
« Écrire un programme qui prend un entier et dit de quoi sa valeur absolue est la factorielle. »

- Découper en sous-problèmes.



Sous-problèmes, sous-sous-problèmes... (1/2)

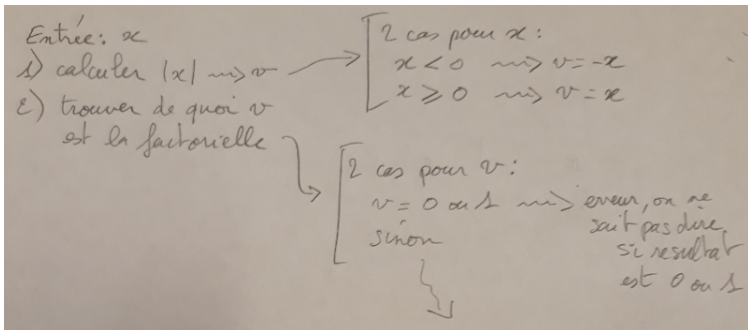
- Problème **actuel** : calcul de la valeur absolue.
- Deux cas.
 - ▶ **Lesquels** ?
 - ▶ Quels **résultats** pour ce problème ?



- Raisonnement **local**.

Sous-problèmes, sous-sous-problèmes... (2/2)

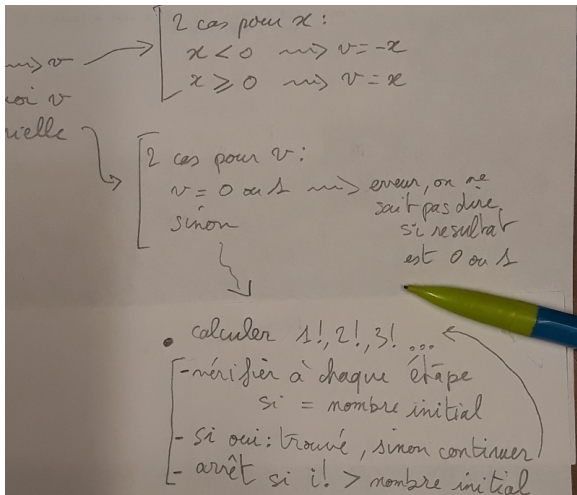
- Problème **actuel** : calcul de la « factorielle inverse ».
- Deux cas.
 - ▶ **Lesquels**?
 - ▶ Quels **résultats** pour ce problème ?
 - ▶ Utilisation de l'**analyse** faite en diapo 11.



- Raisonnement toujours **local**.
- Reste le cas « sinon » à traiter.

Le sous-problème de la factorielle ...(solution 1)

- Comment trouver de quel nombre v est la factorielle ?

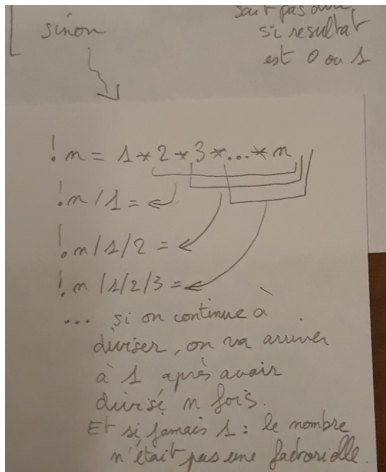


- \Rightarrow Boucle.

- Remarque : inutile de calculer $0!$ et $1!$ (déjà gérés).

Le sous-problème de la factorielle ... (solution 2)

- Comment trouver de quel nombre v est la factorielle?



- \Rightarrow Boucle.
- Remarque : inutile de diviser par 1.
- Remarque : plus besoin de calculer la factorielle.

L'algorithme complet

Entrée: x

1) calculer $|x| \rightarrow v$

2) trouver de quoi v est la factorielle

Entrée: x

1) calculer $|x| \rightarrow v$

2) trouver de quoi v est la factorielle

2 cas pour x :

- $x < 0 \rightarrow v = -x$
- $x \geq 0 \rightarrow v = x$

2 cas pour v :

- $v = 0$ ou $1 \rightarrow$ erreur, on ne sait pas dire si résultat est 0 ou 1
- sinon

! $m = 1 \times 2 \times 3 \times \dots \times m$

! $m/1 =$

! $m/4/2 =$

! $m/4/2/3 =$

... si on continue à diviser, on va arriver à 1 après avoir divisé m fois.

Et si jamais 1: le nombre n'était pas une factorielle

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

```
int factinv (int x) {
    int i, div, v ;
    /* Compute v as |x|. */
    if (x < 0) v = -x ;
    else v = x ;
    /* Check error. */
    if ((v == 0) || (v == 1))
        return -1 ;

    div = v ;
    for (i = 2; i <= v; i++) {
        div = div / i ;
        if (div == 1) return i ;
    }
    return -1 ;
}
```

- Création de l'algorithme par **décomposition en sous-problèmes**.
- Raisonnement **local** à chaque sous-problème.
 - ▶ **Questions** et **solutions** petit-à-petit.
- Synthèse des **solutions** des **sous-problèmes** imbriqués → solution du problème **global**.
- Algorithme conçu **hors langage de programmation spécifique**.
- Langage de programmation : à la toute **fin**, une fois l'**algorithme conçu**.

La satisfaction du programme qui tourne

- Vérifier les résultats : tests.

```
factinv (120) → 5  
factinv (-5040) → 7  
factinv (0) → -1  
factinv (1) → -1  
factinv (119) → -1  
factinv (-119) → -1
```

- Encore...

```
factinv (3) → 2  
factinv (-25) → 4
```



Analyse de la défaillance (1/2)

- **Tracer** le comportement du programme : des printf.
- **Afficher** les variables pertinentes.

```
int factinv (int x) {
    int i, div, v ;
    /* Compute v as |x|. */
    if (x < 0) v = -x ;
    else v = x ;
    /* Check error. */
    if ((v == 0) || (v == 1)) return -1 ;
    div = v ;
    for (i = 2; i <= v; i++) {
        printf ("Before div = %d\n", div);
        printf ("i = %d\n", i);
        div = div / i ;
        printf ("After div = %d\n", div);
        if (div == 1) return i ;
    }
    return -1 ;
}
```

Analyse de la défaillance (2/2)

```
—> factinv (3)
Before div = 3
i = 2
After div = 1
2
```

- Lien entre `div` et `i` : `div = div / i`
- Division **entière**.
- Erreur dans notre **modélisation** : pas de prise en compte du **type** des « nombres ».
- Vocabulaire « parlé » **imprécis**.
 - ▶ En mathématiques : $\mathbb{Z} \neq \mathbb{R}$.
 - ▶ En informatique : `int` \neq `float`.

Régler son compte au problème

- Solution simple et rapide : utiliser des `float`.
- Domaine de factorielle : $\mathbb{N} \Rightarrow$ pas très élégant d'introduire \mathbb{R} .
- Solution « contre nature » apportant d'autres problèmes.
- **Réflexion** ...
« Quand est-on certain que le calcul réussit lors de la division ? »
- Réponse : « Quand la division tombe juste ».
- « Division tombe juste » \equiv reste = 0.
- Opérateur `%` : reste division entière.

Correction du problème

```
int factinv (int x) {  
    int i, div, v ;  
    /* Compute v as |x|. */  
    if (x < 0) v = -x ;  
    else v = x ;  
    /* Check error. */  
    if ((v == 0) || (v == 1))  
        return -1 ;  
    div = v ;  
    for (i = 2; i <= v; i++) {  
        int rem = div % i ;  
        if (rem != 0) return -1 ;  
        div = div / i ;  
        if (div == 1) return i ;  
    }  
    return -1 ;  
}
```

```
factinv (120) → 5  
factinv (-5040) → 7  
factinv (0) → -1  
factinv (1) → -1  
factinv (119) → -1  
factinv (-119) → -1  
factinv (-25) → -1  
factinv (3) → -1
```



- Moins de tours de boucle effectués.

- Concevoir un algorithme : besoin de **rigueur** et de **détails**.
 - ▶ **Type** des données manipulées à exprimer **formellement**.
 - ▶ Étapes de calcul **toutes explicitées**.
 - ▶ Identifier les **limites** de la **solution** apportée au **problème**.
 - ▶ « *Ok, je ne traite que les solides indéformables.* »
 - ▶ Traiter **tous** les cas possibles dans le périmètre décidé.
 - ▶ *Ne pas oublier le cas où le solide a une masse nulle (sans doute une erreur).*
- Implantation : prendre en compte des aspects **techniques** du langage de programmation.
 - ▶ **Après** avoir **décomposé** le problème « suffisamment ».

Domaine des entrées, domaine des sorties

Importance des domaines (1/2)

« Écrire un programme qui calcule la racine carrée d'un nombre donné en entrée. »

- Domaine de définition : \mathbb{R}^+ .
- Type informatique float : toujours signé.
 - ▶ 2 cas : < 0 et $\geq 0 \Rightarrow$ condition à tester.
- Hop, méthode itérative de Newton...

```
#define X0 (3.0)    /* Random seed > 0. */

float squre (float a, int max_iter) {
    int i ;
    float xn = X0 ;



    if (a < 0) return -1 ; /* Assume -1 is the error value. */
    for (i = 0; i < max_iter; i++)
        xn = 0.5 * (xn + (a / xn)) ;

    return xn ;
}
```

« *Écrire un programme qui calcule la racine carrée d'un nombre donné en entrée.* »

- « **Nombre** » : qui a dit que c'est un réel ?
- Qui a dit que le **résultat** était un réel ?
- Autre choix : résultat entier \Rightarrow absence de racine.
- Autre choix : résultat complexe \Rightarrow aucun cas d'erreur, nécessité de créer des nombres complexes.
- Autres domaines d'entrées / sorties \Rightarrow algorithme totalement **différent**.
- \Rightarrow Nécessité de définir **clairement** les **domaines** (entrées **et** sorties).

- C'est un entier ?
- C'est un entier positif ou nul ?
- C'est un réel ?
- C'est suite finie ordonnée de lettres (mot, chaîne de caractères) ?
- Valeur de vérité (booléen, vrai / faux) ?
- C'est un couple (entier \times booléen) ?
- C'est un ensemble non ordonné de réels ?
- Indique les **structures** manipulées par l'algorithme.
- Exemples :
 - ▶ Date \Rightarrow 3 entiers.
 - ▶ Âge \Rightarrow 1 entier.
 - ▶ Température \Rightarrow 1 réel ($^{\circ}\text{K} \Rightarrow \geq 0$).

- C'est une distance ?
 - ▶ À échelle astronomique ?
 - ▶ À échelle subatomique ?
- C'est une vitesse ?
 - ▶ Linéaire ?
 - ▶ Angulaire ?
- Dirige les opérations **autorisées** sur les données.
 - ▶ Vitesse + distance = 
 - ▶ Vitesse linéaire + vitesse angulaire = 
- Définit les **ordres de grandeur**, les ϵ .
 - ▶ Collision de 2 **planètes** : distance calculée = 10 m \Rightarrow **oui**.
 - ▶ Collision de 2 **particules** : distance calculée = 10 m \Rightarrow **non**.

- Scalaires informatique = **approximation** des « nombres » mathématiques.
- Arrondis, finitude, etc.
- Parfois, l'algorithme doit **prendre en compte** ces différences.
- Exemple : programme de la racine carrée (diapo 26).
- Flottant float toujours signés.
- Impossible de restreindre **structurellement** (par le type de données) à \mathbb{R}^+ .
- \Rightarrow Un **test** (if) à faire dans l'algorithme.

- Commencer par déterminer les **domaines** des données :
 - ① ... mathématique,
 - ② ... sémantique,
 - ③ ... informatique (type),
- ... pour entrées **et** sorties.
- Permet de préciser la **spécification** (besoin attendu).
 - ▶ **Que faire** dans un cas non (clairement) spécifié ?
- \Rightarrow **Influence** l'algorithme (cas admis, cas d'erreur).