


# chapitre 1.- Notions de Pré & Post conditions :

## Point de algorithmique & mise en œuvre



Ressource R1.01 : Initiation au développement - Partie 2

# 1.- Rappels : Donnée/Résultat d'un traitement

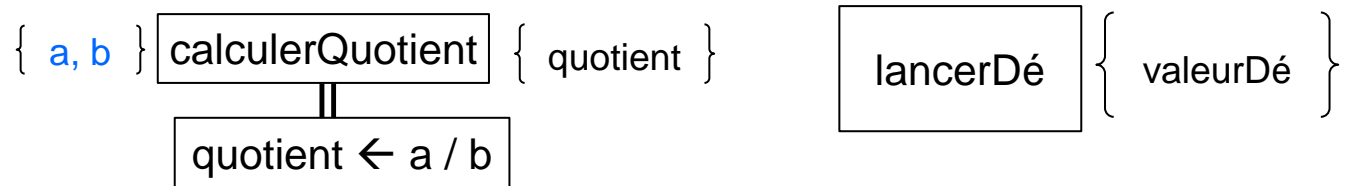
## □ Donnée d'un traitement

### – Rappel Définition

**Information** (variable, constante, ...) dont la **valeur** est nécessaire pour réaliser **ce** traitement

### – Exemples

- a, b : Données du traitement « calculerQuotient »
- pas de Donnée pour le traitement « lancerDé »



# 1.- Rappels : Donnée/Résultat d'un traitement

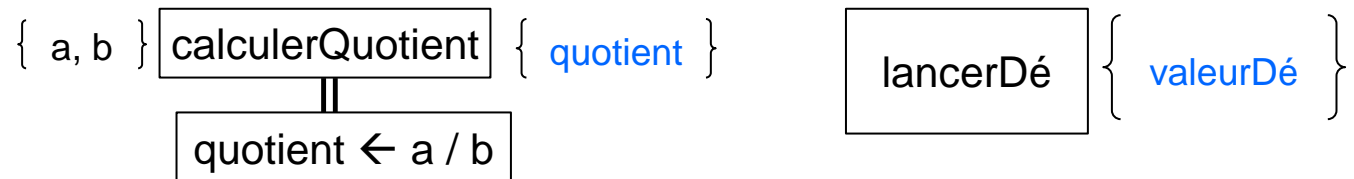
## ❑ **Résultat** d'un traitement

### – Rappel Définition

**Information** (variable, ...) dont la **valeur** est calculée / modifiée / initialisée par **ce** traitement

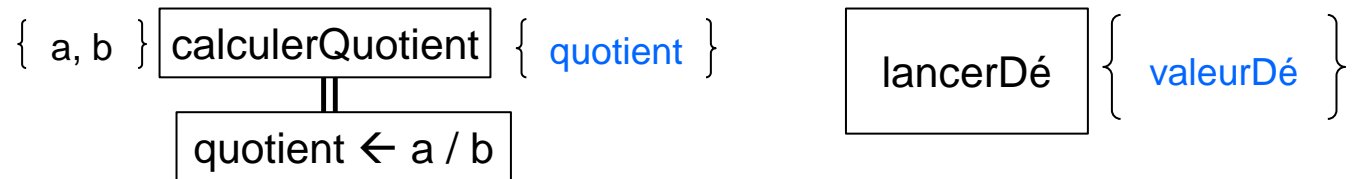
### – Exemples

- quotient : Résultat du traitement « calculerQuotient »
- valeurDé : Résultat du traitement « lancerDé »



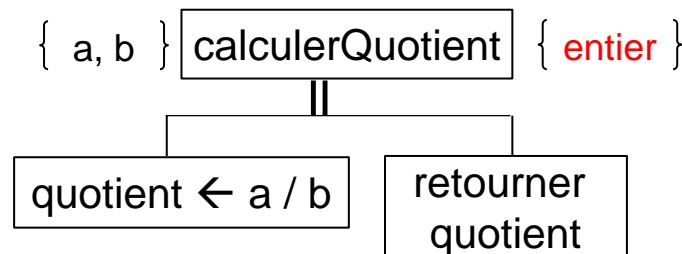
# 1.- Rappels : Donnée/Résultat d'un traitement

- ❑ Ne pas confondre
  - **Résultat** d'un traitement



et

- **Valeur résultante** d'une fonction



## 2.- Pré-condition & Post-condition : Définition algorithmique

### ❑ **Pré-condition** sur la **Donnée** d'un traitement

#### – Définition

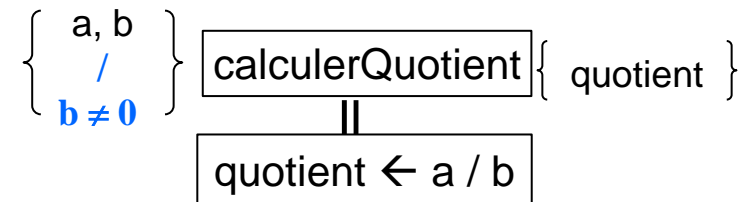
Il s'agit d'une **condition vérifiée par une Donnée d'un traitement**  
Cette condition est exprimée au moyen d'une expression booléenne

#### – Exemple

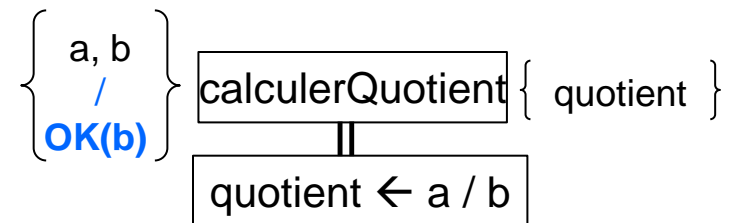
- Traitement « calculerQuotient » : la Donnée **b** doit vérifier la pré-condition :  $b \neq 0$

#### – Notation

- Si la pré-condition est simple à écrire, elle peut figurer sur l'algorithme



- Sinon, elle est signalée dans l'algorithme, au moyen d'une fonction booléenne, puis est et précisée dans la description de l'action :



action calculerQuotient :

pré-condition  $\text{OK}(b)$  : est vrai si  $b \neq 0$

## 2.- Pré-condition & Post-condition : Définition algorithmique

### ❑ **Post-Condition** sur le **Résultat** d'un traitement

#### – Définition

Il s'agit d'une **condition vérifiée par un Résultat d'un traitement**

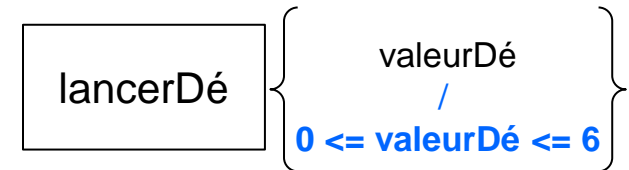
Cette condition est exprimée au moyen d'une expression booléenne

#### – Exemple

- Traitement « lancerDé » : le résultat **valeurDé** vérifie la post-condition :  
(valeurDé  $\geq$  0 **et** valeurDé  $\leq$  6)

#### – Notation

- Si la post-condition est simple à écrire, elle peut figurer sur l'algorithme

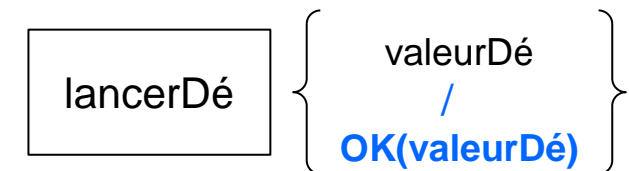


- Sinon, elle est signalée dans l'algorithme au moyen d'une fonction booléenne, puis précisée dans la description de l'action :

action lancerDé :

pos-condition OK(valeurDé) :

est vrai si 0 <= valeurDé <= 6



## 2.- Pré-condition & Post-condition : Définition algorithmique

### ❑ **Post-Condition** sur la **valeur résultante** d'une fonction

#### – Définition

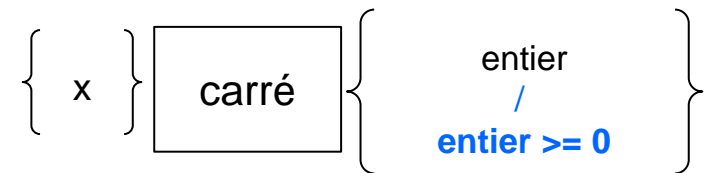
Il s'agit d'une **condition vérifiée par la valeur Résultante de la fonction**

#### – Exemple

- Fonction « carré » : calcule et retourne le carré du paramètre x donné,  
La valeur retournée par la fonction est  $\geq 0$

#### – Notation

- Si la post-condition est simple à écrire, elle peut figurer sur l'algorithme



- Sinon, elle est signalée dans l'algorithme au moyen d'une fonction booléenne, puis précisée dans la description de l'action

# 3.- Pré-condition & Post-condition : Intérêt

---

## ❑ En phase algorithmique

### Consolide la justesse et robustesse de l'algorithme

Les propriétés des informations sont **explicitées**, ce qui permet de :

- en TD {
- Préciser les **conditions d'utilisation** d'un traitement (pré-condition)
  - Préciser **l'état d'éléments** après un traitement (post-condition)
  - **Valider les enchaînements** entre les différents traitements de l'algorithme
  - **Comprendre le fonctionnement** d'un algorithme
  - Identifier les différents scénarios du programme

## ❑ Avant le codage

### Facilite la fabrication des jeux d'essai du programme

- Les jeux d'essais devront être composés de valeurs vérifiant les pré-conditions identifiées et de valeurs ne les vérifiant pas

## ❑ En phase de codage

### Facilite le debogage-test du programme

- Vérifier que des valeurs produites par le programme satisfont bien des post-conditions, conformément aux spécifications externes
- Vérifier que des variables nécessaires à un programme satisfont bien certaines pré-conditions



## 4.- Pré-condition & Post-condition : Mise en œuvre en programmation

---

### □ Aide à la mise au point / test des programmes

- En utilisant les dispositifs intégrés dans la plateforme de développement (dans l'outil de débogage)

**Dispositif** : outil 'Espion' du débogueur de VSCode

- En utilisation des Bibliothèques d'instructions spécifiques associées aux langages de programmation

**Dispositif** : bibliothèque <cassert> pour le langage C++

- En utilisant des environnements de tests que l'on peut installer sur la plateforme de développement

**Dispositif** : framework Junit

- ...

## 4.- Pré-condition & Post-condition : Mise en œuvre en programmation

---

- ❑ **Inclusion dans le comportement du programme en cours d'exécution**
  - Le programme réagit en cours d'exécution si des pré/post conditions ne sont pas vérifiées par les éléments (données / résultats) du programme

Exemple : déclenchement automatique des masques à oxygène en cas de dépressurisation de la cabine de l'avion

**Dispositif** : mécanisme des exceptions disponible dans le langage C++

- ...

## 5.- Pré-condition & Post-condition : Mise en œuvre avec <cassert>

---

### ❑ Bibliothèque <cassert> en C++

- Dans le dictionnaire : une assertion est une proposition, affirmative ou négative, que l'on affirme comme vraie.
- En C++, un `assert` est un sous-programme testant une condition, généralement une propriété d'un élément du programme ou d'une zone mémoire
- Si la condition est vérifiée, le programme continue son exécution normalement
- Si la condition n'est pas vérifiée, le programme s'arrête en donnant des informations sur l'erreur :
  - A quelle ligne du programme l'erreur s'est produite
  - Quelle est la condition non vérifiée
- **Syntaxe**

```
void assert (int condition);
```

## 5.- Pré-condition & Post-condition : Mise en œuvre avec <cassert>

---

- ❑ **Utilisation de la bibliothèque <cassert> en C++**
  - **Uniquement pour la mise au point d'un programme**
  - **Pas du tout** pour gérer les erreurs d'exécution du programme une fois mis en production (= installé)
    - Les erreurs d'exécution sont gérées avec d'autres mécanismes, par exemple des exceptions

## 5.- Pré-condition & Post-condition : Mise en œuvre avec <cassert>

### ❑ Exemple 1 – mise en place

- Activation des assert et Utilisation de la bibliothèque

```
1  #include <iostream>
2  using namespace std;
3  |
4  ✓ // #define NDEBUG           // si décommenté, désactive assert
5  | | | | |                   // placer AVANT #include <cassert>
6  #include <cassert>
7
```

- Positionnement d'un assert exprimant une pré-condition

```
29  float moyenne (int somme, unsigned short int nb)
30  {
31      assert (nb != 0);
32      return ( (float) somme / nb);
33  }
```

## 5.- Pré-condition & Post-condition : Mise en œuvre avec <cassert>

### ❑ Exemple 1: exécution quand condition non vérifiée

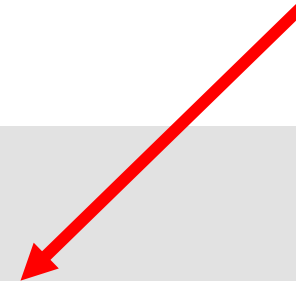
- .. et assert **activé**

`// #define NDEBUG`

- Appel de la fonction

```
int main(void)
{
    cout << moyenne (10, 0) << endl;
    return 0;
}
```

diviseur = 0



## 5.- Pré-condition & Post-condition : Mise en œuvre avec <cassert>

- ❑ **Exemple 1 : exécution quand condition non vérifiée  
et assert **activé****

ligne de l'erreur

```
Assertion failed!
```

Program: C:\Users\Pantxika\OneDrive - IUT de Bayonne\but\r101\_partie 2\1\_Assertions\workspace-plusplus\assertionExemple1\main.exe

File: main.cpp, Line 31

Expression: nb != 0

assertion non vérifiée

```
29 float moyenne (int somme, unsigned short int nb)
30 {
31     assert (nb != 0);
32     return ( (float) somme / nb);
33 }
```

## 5.- Pré-condition & Post-condition : Mise en œuvre avec <cassert>

### ❑ Exemple 2 : Améliorer le message de l'assert

Message plus précis

assertion

```
29 float moyenne (int somme, unsigned short int nb)
30 {
31     assert (("pre-cond : diviseur de moyenne() doit etre non nul ", nb != 0 ));
32     |
33     return ( (float) somme / nb);
34 }
```

– Attention syntaxe

```
assert ( ( "message", assertion ) );
```

– A l'exécution

```
C:\WINDOWS\system32\cmd.exe
Assertion failed!

Program: C:\Users\Pantxika\OneDrive - IUT de Bayonne\but\r101_partie2\1_Assert
ions\workspace_cplusplus\assertionExemple1\main.exe
File: main.cpp, Line 31

Expression: ("pre-cond : diviseur de moyenne() doit etre non nul ", nb != 0 )
```




## 5.- Pré-condition & Post-condition : Mise en œuvre avec <cassert>

---

### ❑ Exemple 3 – Désactivation de l'assert

- Décommenter la définition de la variable NDEBUG



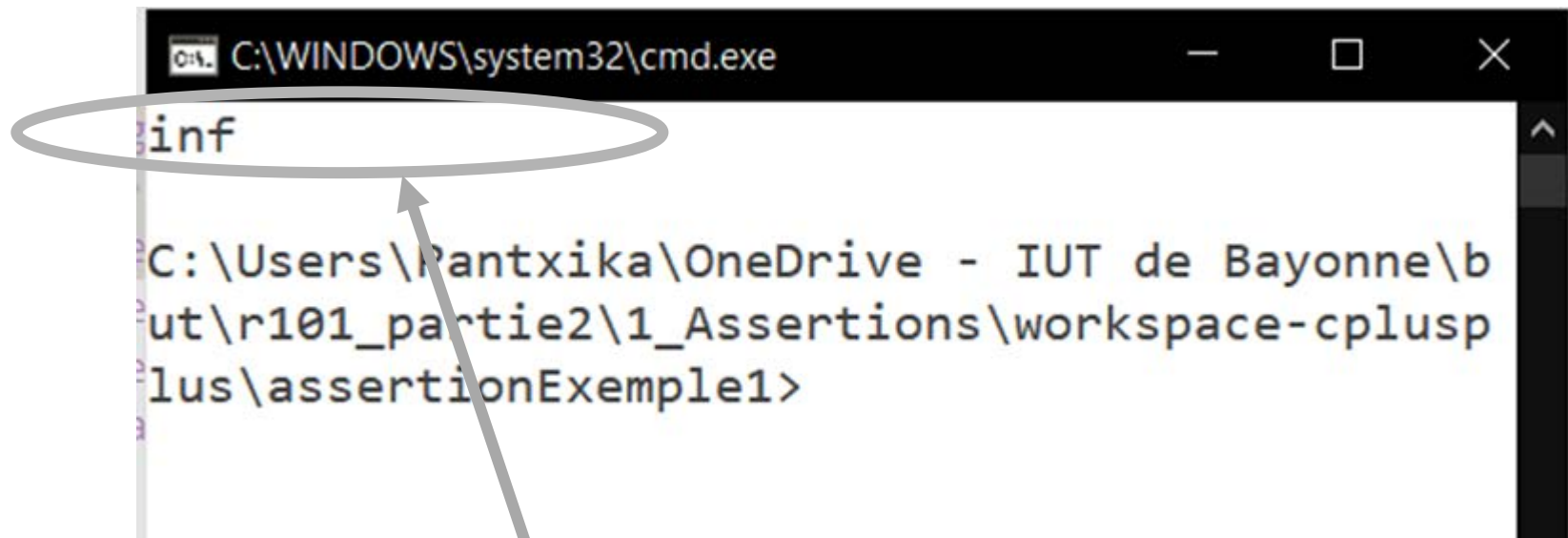
```
1  #include <iostream>
2  using namespace std;
3
4  #define NDEBUG           // si décommenté, désactive assert
5                           // placer AVANT #include <cassert>
6  #include <cassert>
```

- Recompiler le programme

## 5.- Pré-condition & Post-condition : Mise en œuvre avec <cassert>

### ❑ Exemple 3 : exécution quand condition non vérifiée et assert **désactivé**

– Résultat d'exécution



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The command prompt displays the text "inf" on the first line, which is circled in grey. Below it, the directory path "C:\Users\Pantxika\OneDrive - IUT de Bayonne\b" is shown, followed by "ut\r101\_partie2\1\_Assertions\workspace-cplusp" and "lus\assertionExemple1>" on the next line. A grey arrow points from the text "Erreur système non explicite" below the screenshot to the circled "inf" text.

Erreur système non explicite

# chapitre 1.- Pré-Post conditions algorithmiques & mise en œuvre



Ressource R1.01 : Initiation au développement - Partie 2

**Merci pour votre attention !**