

# Chapitre 10 : Approches programmatives

## Construction et maintenance de logiciels

Guy Francoeur

basé sur du matériel pédagogique d'Alexandre Blondin Massé, professeur

**UQÀM** | **Département d'informatique**

# Table des matières

1. Justesse et robustesse
2. Programmation par contrat
3. Programmation défensive

## Définition:

- ▶ Une fonction est **juste** (ou **correcte**) si elle ne retourne **jamais** de résultat **inexact**;
- ▶ En particulier, on préfère ne **rien retourner** que retourner quelque chose de **faux**;
- ▶ **Note :** Une fonction est **juste** quand elle retourne un résultat exact **peu importe les paramètres en entrée**.
- ▶ **Exceptions :** Algorithmes **probabilistes**, où on accepte que le résultat soit parfois erroné, mais avec une probabilité **faible**.
- ▶ **Exemple :** test de **primauté** (décider si un nombre est **premier** ou non).

## Définition:

- ▶ Une fonction est **robuste** si elle se termine **toujours** sans faire **planter** le programme;
- ▶ **En C**, c'est une fonction qui
  - ▶ ne déclenche pas d'**erreur de segmentation**,
  - ▶ ne déclenche pas d'**erreur de bus** (*bus error*) et
  - ▶ ne déclenche pas d'**assertion**.

- ▶ On s'assure de la **justesse** et de la **robustesse** à l'aide de deux **approches programmatives** complémentaires :
  - ▶ La programmation **par contrat**;
  - ▶ La programmation **défensive**.
- ▶ Programmation par **contrat**: utilisation de **spécifications**, validées par des **assertions**.
- ▶ Programmation **défensive**: on suppose le **pire** et on prévient à l'aide d'**exceptions** ou de **codes d'erreur**.

# Table des matières

1. Justesse et robustesse
2. Programmation par contrat
3. Programmation défensive

## Définition :

- ▶ Ce type de programmation est plutôt connu et simple d'usage. Nous avons un **module appelant** (client) et un **module appelé** (fournisseur);
- ▶ Un **contrat** est défini par :
  - ▶ des **préconditions**, que le client doit satisfaire lorsqu'il fait appel au fournisseur (**@pre**);
  - ▶ des **postconditions**, que le fournisseur doit satisfaire une fois le service rendu (**@post**).

## Préconditions :

- ▶ Le module **client** considère le service offert par le **fournisseurs** comme une **boîte noire**;
- ▶ Le client doit s'assurer que les **préconditions** soient satisfaites.

## Postconditions :

- ▶ Le module fournisseur a pour rôle de livrer les **postconditions**;
- ▶ Peut importe qui l'appelle et ce qui sera fait par la suite.



# Les contrats et les assertions

Si une **précondition** n'est pas satisfaite :

- ▶ il y a un **bogue** dans le code **client**;
- ▶ le service ne devrait pas être **fourni**.

Si une **postcondition** n'est pas satisfaite :

- ▶ il y a un **bogue** dans le code **fournisseur**;
- ▶ le programme client devrait être **avorté vraiment?**.

Dans le cas du langage **C** :

- ▶ Il existe une **bibliothèque** standard : **assert.h**;
- ▶ Aussitôt qu'une **assertion** n'est pas satisfaite, le programme est **avorté**.

## Activation des assertions :

- ▶ En mode **développement/test**;
- ▶ Permet de vérifier les **contrats**;
- ▶ Permet de vérifier que du code n'est pas exécuté. sections **interdites**;
- ▶ Nous ne voulons pas que les **assertions** soient **déclenchées, bogue**;

## Désactivation des assertions :

- ▶ Pour la **livraison** du code une fois **testé**;
- ▶ Il suffit de passer l'option **-DNDEBUG** lors de la compilation.

## Exemple d'assertions (1/2)

- ▶ Mettre en place des **contrats**;
- ▶ Signaler que certaines sections de code ne devraient pas être **appelées**;
- ▶ Les assertions permettent de détecter les cas **critiques**;
- ▶ Par exemple, un **arbre AVL** devrait demeurer **équilibré** après une **insertion**, une **suppression**;
- ▶ Un autre exemple, un **tableau trié** devrait demeurer **trié** après une **insertion** et une **suppression** (on parle alors d'**invariants**);
- ▶ Nous comprenons que les assert **arrêtent** le programme;

- ▶ Par conséquent la gestion des **erreurs** ne devrait pas être traitée à l'aide **des assertions** (par exemple, mauvaise entrée de l'**utilisateur**);
- ▶ Si une **assertion** est activée, cela signifie qu'il y a un **bogue** dans le programme (quelque chose de grave);
- ▶ Les assertions ne devraient pas avoir **d'effets de bord**<sup>1</sup>.

<sup>1</sup> qui n'a pas d'interaction observable avec le monde extérieur.

# Exemple

```
1 // assert.c
2 #include <assert.h>
3
4 int main(void) {
5     int a = 5;
6     assert(a == 5 && "mon premier assert est invalide");
7 #ifdef ERR
8     assert(a == 6 && "mon deuxieme assert est invalide");
9 #endif
10    return 0;
11 }
```

# Table des matières

1. Justesse et robustesse
2. Programmation par contrat
3. Programmation défensive

## Définition :

- ▶ Type d'approche qui préconise d'avertir le **module appelant** aussitôt qu'une **erreur** est rencontrée;
- ▶ Plusieurs modes de **communication**
  - ▶ En **déclenchant** une **exception**;
  - ▶ En retournant une **valeur spéciale** : -1 pour un **entier**, NULL pour un pointeur, etc.
  - ▶ Approche typique en C :
    - ▶ En retournant un **code**, nommé **code de retour**;
    - ▶ Généralement ce code devrait être documenté;
    - ▶ Le code (int) a une signification précise;

# Exceptions

- ▶ Sont soulevées lorsque des événements **indésirables**, mais **prévisibles** surviennent;
  - ▶ Il n'y a plus de **mémoire disponible**, lors d'un malloc ou realloc;
  - ▶ Le disque est plein;
  - ▶ Une connexion vers un système distant a échoué;
  - ▶ L'utilisateur entre une **chaîne de caractères** plutôt qu'un **nombre entier**;
- ▶ Un autre mot pour remplacer **prévisible** (ci-haut)?
- ▶ Par opposition, les **assertions** sont réservées aux événements **critiques**;



# Signalement d'exceptions

- ▶ Affichage de messages d'erreur sur le canal **stderr**;
- ▶ Écriture de messages d'erreur dans un **journal** (en anglais **log file**);
- ▶ Terminer l'exécution d'un programme, en cas d'**erreur fatale** :
  - ▶ Dans la bibliothèque **stdlib.h**, on trouve les constantes **EXIT\_SUCCESS** et **EXIT\_FAILURE**;

# Signaler une exception à l'aide d'un code d'erreur

- ▶ Lorsque l'erreur n'est pas **fatale**, on retourne une valeur de **statut** ou un **code d'erreur**.
- ▶ La valeur **0** indique que tout s'est **bien déroulé**;
- ▶ Sinon, on retourne une **valeur appropriée** en utilisant des **constantes**;
- ▶ Autre stratégie, utiliser une valeur **spécifique** pour indiquer un problème (**-1**, **NULL**, etc.);
  - ▶ Problème : n'indique rien sur l'**erreur** survenue;
  - ▶ Potentiellement **dangereux** avec d'autres types de données.

# Exemple

```
1 #include <stdio.h>
2 #include <math.h>
3
4 typedef unsigned long u64_t;
5
6 // estPremier
7 int estPremier(u64_t *n){
8     u64_t nLimit = sqrt(*n);
9     u64_t i = 2;
10
11     while (i++ < nLimit){
12         if (*n % i == 0) return 1;
13     }
14     return 0;
15 }
```

# Conclusion et questions

Questions ?