PROGRAMMATION PAR CONTRATS GESTION DES EXCEPTIONS

2012-10-01

© Emmanuel Chieze
Département d'informatique, UQAM
INF3135

Plan

- Programmation par contrats
- Gestion des exceptions

Programmation par contrats

3

- Contexte d'une relation client-fournisseur
  - client = module appelant une fonction d'un autre module, le fournisseur
- Nécessité d'un contrat de service spécifiant :
  - des préconditions : doivent être satisfaites par le client
  - des postconditions : doivent être satisfaites par le fournisseur

Programmation par contrats

4

- Avantages :
  - pour le client : la garantie que les post-conditions seront satisfaites
  - pour le fournisseur : la garantie que les pré-conditions seront satisfaites
- Un fournisseur de service devrait être paresseux
  - exiger les pré-conditions les plus strictes
  - acquiescer aux post-conditions les plus faibles

Bogues

5

- Lorsqu'une pré-condition n'est pas satisfaite
  - bogue dans le code client
  - le service ne doit pas être fourni
- Lorsqu'une post-condition n'est pas satisfaite
  - bogue dans le code fournisseur
  - le programme client devrait être avorté

Exemple de contrat

6

- Service = calcul de l'inverse d'un long double x
- Contrat possible :
  - pré-condition : x != 0
  - post-condition : | y * x -1 | < p
    - où y dénote la valeur retournée par le service,
    - et p un niveau de précision fixée dans le contrat
- Vérification du contrat par des assertions

Assertions en C

7

- Macro-fonction assert (en-tête de la librairie standard <assert.h>)
- Avorte le programme si la condition en argument n'est pas vérifiée

```
exemple7.1.c:26: failed assertion `x != 0'
```

- cf. exemple7.1.c
  - en spécifiant -DPRECISION=0.0000001 à la compilation
  - en spécifiant -DPRECISION=0.0 à la compilation
    - permet de voir les effets sur les post-conditions
    - représente une contrat aberrant cependant

© Emmanuel Chieze, Département d'Informatique, UQAM.

Approches exigeante et tolérante

8

- Approche exigeante
  - pré-conditions vérifiées par des assertions
  - préférable pour des données internes au système
  - exemple7.1.c
- Approche tolérante
  - vérifiées par des if et des traitements appropriés
  - préférable pour des données provenant de l'extérieur du système
  - exemple7.2.c

© Emmanuel Chieze, Département d'Informatique, UQAM.

Utilité des assertions

9

- Mettre en place des contrats
- Rendre explicites des hypothèses implicites
  - expliciter le cas représenté par else/default
- Signaler qu'on appelle des sections de code qui ne devraient pas l'être
  - else/default dans lesquels on ne devrait pas aller
- Signaler une portion de programme non encore développée

Utilité des assertions

10

- Les assertions servent à gérer des cas supposément impossibles
  - ne servent pas à gérer des cas d'erreurs normales
    - mauvaise entrée de l'utilisateur
      - le problème est avec les données, pas le programme !
    - absence de mémoire disponible
  - ne doivent pas avoir d'effets de bord i.e. elles ne doivent pas modifier l'environnement
- L'activation d'une assertion est le signe d'un bogue

Utilisation des assertions en C

11

- Activation des assertions
  - en mode développement/tests
  - permet de vérifier les contrats, le non-passage dans des sections de codes interdites …
  - les assertions ne devraient pas être déclenchées !
- Désactivation des assertions
  - pour livraison du code testé
  - en définissant NDEBUG à la compilation (-DNDEBUG)

Utilisation des assertions en C

12

- Combinaison de conditions
  - assert (x>=0 && x<NB && y>=0 && y< NB && "Coordonnées hors bornes")
- Ou
  - assert (x>=0 && x<NB && "x hors bornes")
  - assert (y>=0 && y< NB && "y hors bornes")
- Ou
  - assert (x>=0 && "x négatif")
  - assert (x>=0 && "x trop grand")
  - assert (y>=0 && y< NB && "y négatif")
  - assert (y>=0 && y< NB && "y trop grand")

Limites des assertions

13

- Exemple : recherche dichotomique

```
int recherche_dicho(int cle, int t[], int taille) {
   int d = 0, f = taille - 1, m, trouve = FAUX;
   while (d <= f && trouve == FAUX) {
      m = (d+f)/2;
      if (t[m] == cle)
         trouve = VRAI;
      else if (t[m] < cle)
         d = m + 1;
      else
         f = m - 1;
   }
   if (trouve == TRUE)
     return m;
   else
     return -1;
}
```


Impact des préconditions et postconditions

14

- Pré-conditions :
  - taille >= 1
  - ∀ i ∈ [0, taille -2], t[i] <= t[i+1]
  - VÉRIFICATION ?
- Post-conditions :
  - (vrai et t(i) = clé) ou (faux et ∀ i ∈ [0, taille -1], t[i] != cle)
  - VÉRIFICATION ?

Plan

15

- Programmation par contrats
- Gestion des exceptions

Exceptions

16

- Exception : évènement indésirable mais prévisible
  - un vecteur est plein
  - il n'y a plus de mémoire disponible
- Ne peut être gérée par des assertions
  - réservées aux évènements supposément impossibles
- Mécanisme de gestion
  - signaler au programme appelant l'exception
  - le laisser gérer l'exception

Signalement d'exceptions

17

- Pour une application
  - affichage de messages d'erreur sur stderr
  - écriture de messages d'erreur dans un journal (*log file*)
  - terminer l'exécution d'un programme, en cas d'erreur fatale
    - fonction `void exit(int statut)`, définie dans stdlib.h
      - 2 statuts prédéfinis : EXIT_SUCCESS, EXIT_FAILURE
      - dans main() : exit(valeur) équivaut à return(valeur)
      - ferme les fichiers ouverts, vide les tampons

Signalement d'exceptions

18

- Pour une fonction dans un module auxiliaire
  - utilisation d'un statut comme résultat d'une fonction
    - approche traditionnelle en C
      - return(0) si OK
      - return(code d'erreur) si NOK
  - utilisation d'une valeur spécifique comme résultat d'une fonction
    - pointeur : retourner NULL en cas d'erreur
    - ne dit rien sur l'erreur survenue
    - potentiellement dangereux avec d'autres types de données