Les exceptions

Comment gérer les états de faute.

Opposition et complémentarité de la robustesse et la correction ?

- Deux notions clés relativement à la fiabilité
  - **La correction** : permet de garantir que l'utilisation d'un objet dans l'application ne pourra pas engendrer d'erreurs
  - **La robustesse** : permet de garantir que si un objet (ou une opération effectuée par un objet) est défaillante l'application récupérera de l'erreur

- Opposition entre les deux approches
  - **La correction** : détection avant d'exécuter une opération illicite (compilation/exécution)
  - **La robustesse** : traitement s'effectuant après qu'une opération illicite se soit produite

La gestion des exceptions

- La notion d'exception
  - Evénement anormal se produisant lors de l'exécution du système
- La notion d'échec
  - Une fonction échoue si elle entre dans un état ne respectant pas les termes du contrat

    Ex : violation de la post-condition

La définition d'une exception

Si lors de l'exécution d'une fonction, un événement pouvant conduire à l'interruption de l'exécution se produit, cet événement est une exception.

- Tout dysfonctionnement est constitutif d'une exception
- Toute exception n'entraîne pas systématiquement un dysfonctionnement

Une liste non exhaustive d'exceptions

- Exécution d'une opération produisant une condition anormale détectée par le système (division par 0)
- Appel d'une routine qui échoue
- Violation d'une pré-condition
- Violation d'une post-condition
- Violation d'un invariant de classe
- Violation d'un invariant de boucle
- Déclenchement par le logiciel d'une exception

Causes d’un échec d’une fonction

Une fonction échoue si et seulement si une exception se produit durant son exécution et cette exception n’est pas gérée par la fonction

- Il est possible de capturer des exceptions afin de récupérer de cet état d’erreurs
- La définition d’échec et d’exception dépendent mutuellement l’une de l’autre

La gestion des exceptions dans les langages C++

1. Déclencher une exception

En C++

```
throw   Expression
```

Où Expression est d'un type quelconque

La capture d'une expression en C++

```
try
   { …
   }
catch(type1 E)
   {…}
catch(type2 E)
   {…}
```

- `try` : début du bloc surveillé
- `catch` : capture les exceptions qui dérivent de *type1* et effectue le code qui suit

TD – Partie 1

- Manipuler des exceptions en C++

S’assurer que les exceptions sont bien traitées

- La fiabilisation des logiciels
  - Implique que toute violation/erreur soit détectée
    - Soit à la compilation
    - Soit à l’exécution par la levée d’exception
  - Implique que toute exception soit gérée

- Idée :
  - Faire que le traitement des exceptions soit vérifier au moment de la compilation

Comment s'assurer que toutes les exceptions sont bien capturées ?

- Lever une exception n'est autorisée que dans un bloc « **`try`** … **`catch`**(…) »
- Il doit exister au moins une clause **`catch`**(…) associé au **`try`** telle que l'exception est capturée par cette clause
- les méthodes et constructeurs doivent exposer la liste des exceptions qui peuvent se produire sans être capturées

```
type nom(param1,…)
    throw(Exception1)
 {…}
```

Le compilateur vérifie que le contrat sur les exceptions est respecté

- Détermine dans un bloc `try` les exceptions pouvant être levées
  - Soit par une opération
  - Soit par l'appel d'une méthode
    - La liste des « `throw` … » définit la liste des exceptions possibles
- Détermine ensuite la liste des exceptions capturées par les `catch` et les retirent
- Détermine enfin la liste des exceptions que chaque bloc du `catch` peut lever
  - Soit par une opération
  - Soit par l'appel d'une méthode
  - Soit par la propagation de l'exception capturée

Le compilateur vérifie que le contrat sur les exceptions est respecté

- Détermine pour un constructeur ou une méthode toutes les exceptions pouvant être levées
- Vérifie que toutes les exceptions sont listées dans la liste des exceptions pouvant être levées par la fonction

Le compilateur vérifie que le contrat sur les exceptions est respecté

- Vérifie enfin qu'aucune exception n'est pas capturée par le corps du programme

TD – Partie 2

- Vérifier que l‘ensemble des exceptions est bien capturé en C++

Quelques règles pour gérer correctement les exceptions

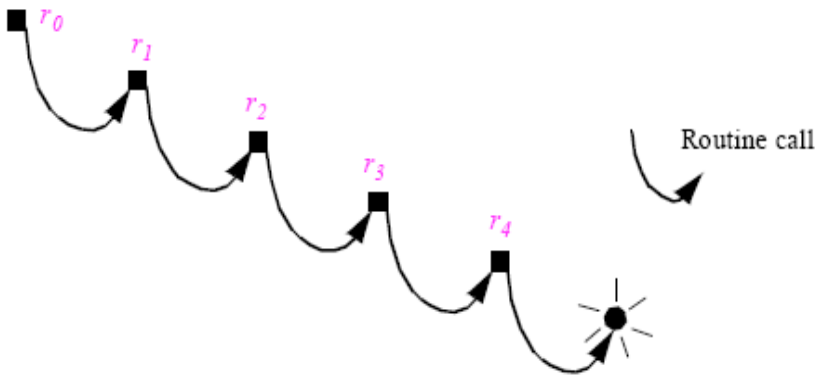
- Trois règles de gestion des exceptions
  1. Nouvel essai : on modifie les paramètres et on effectue de nouveau la procédure
  2. Echec : on nettoie proprement l'environnement et on prévient l'appelant de l'échec de la procédure
  3. Fausse alerte : l'erreur rapportée n'en est pas une, on reprend le processus là où il a été suspendu

Quelques cas où un « nouvel essai » peut être tenté

- Le système d'opération n'a pas pu effectuer une opération dans le temps imparti
  - On réessaye en espérant que la prochaine opération de lecture sera correcte
- Le logiciel doit être tolérant aux pannes
  - On réessaye en espérant que la prochaine opération n'engendrera pas de pannes
- La fonction dépend de données incertaines et fragiles
  - On réessaye en espérant qu'une nouvelle obtention des données permet un exécution correcte
- La procédure implante plusieurs algorithmes
  - On réessaye avec un autre algorithme

Comment gérer correctement l'échec ?

- Deux opérations
  1. Remet le système dans un état consistant
     1. Un état satisfaisant les invariants
     2. Un « état proche » de l'état avant l'appel de la routine
     3. Un « état propre » au niveau allocation Désalloue la mémoire allouée, Libère les ressources allouées, …)
  2. S'assurer que l'appelant est bien prévenu de l'échec de l'opération



Quelques points plus complexes relativement à la gestion des exceptions

- Avoir des informations sur l'exception en cours
  - S'agit-il d'une exception nouvelle ou ancienne ?
  - Quelles ont été les opérations appliquées jusqu'à présent ?
- Empêcher que certaines exceptions puissent se déclencher
  - S'assurer que l'exécution continue même si l'exception se produit
- Lever certaines exceptions au moment où l'on décide
  - Retarder notamment le déclenchement d'une exception en s'assurant que les fonctions ont été réalisées

Conclusion

- Les exceptions sont un concept important pour :
  - Signaler un état pouvant entraîner une erreur
  - Prendre les mesures lorsque cet état s'est réalisé pour
    - Soit réessayer l'opération,
    - Soit remettre le système dans un état consistant
  - Permettre
    - D'augmenter la robustesse des applications
    - De séparer le traitement des erreurs de l'exécution des procédures
- Les exceptions ne sont pas un « moyen » de faire des sauts d'un point à un autre du programme !

TD – Partie 3

- Augmenter le nombre d‘informations contenues dans une exception