

M1103

STRUCTURES DE DONNÉES ET ALGORITHMES FONDAMENTAUX

Cours 1 : Exceptions

Université Grenoble Alpes
IUT 2 – Département Informatique
2019/2020

Plan de la séance

- ▶ Introduction et définitions
- ▶ Exceptions système courantes
- ▶ Traitement d'une exception
- ▶ Exceptions définies par le programmeur
- ▶ Propagation des exceptions
- ▶ Ce qu'il faut retenir

INTRODUCTION ET DÉFINITIONS

Introduction

- ▶ Que se passe-t-il lors de l'exécution d'un programme, si...
 - *une division par zéro est effectuée ?*
 - *vous demandez à l'utilisateur de saisir un entier positif et qu'il entre une valeur négative ?*
 - *une instruction conduit à lire ou écrire en dehors des bornes d'un vecteur ?*
 - *une fonction appelée ne délivre pas de résultat pour certaines valeurs du ou des paramètres effectifs ?*
- ▶ Le compilateur est capable de détecter certaines anomalies et de les signaler (avertissement ou erreur de compilation).
- ▶ Si une anomalie, non détectée par le compilateur, se produit lors de l'exécution, le système déclenchera ce que l'on appelle une exception et l'exécution sera interrompue.

Introduction

- ▶ Le concept d'**exception** a été introduit pour la première fois en Ada et généralisé à tous les langages « modernes » : C++, Java, ...
- ▶ Le programmeur, pour prévenir une **anomalie** détectée par le système, peut - **et devrait toujours** - traiter l'exception correspondante de façon à prendre en compte le problème associé.
- ▶ Si besoin, le programmeur peut également déclencher volontairement ce mécanisme.

Les exceptions sont ainsi un moyen élégant et efficace pour gérer les « conditions limites » de vos programmes...

Introduction



Le mécanisme d'exceptions est fondamental pour produire des logiciels robustes...

La fusée Ariane (plusieurs millions d'€) du vol 501, le 4 juin 1996, a été perdue parce qu'une exception dans un programme Ada n'était pas traitée ...

Un nombre en représentation flottante a été converti en une valeur qui était supérieure à ce que pouvait exprimer un nombre entier sur 16 bits, ce qui a donné lieu à une erreur d'opérande.

Définitions

- ▶ Une **exception** est un événement déclenché **pendant** l'exécution d'un programme lorsqu'une situation anormale (*exceptionnelle*) est détectée.
- ▶ Quand une exception est déclenchée (on dit aussi **levée**), l'exécution normale du programme est interrompue pour que l'exception soit traitée.
- ▶ **Traitement d'une exception :**
 - **Par défaut** : le système affiche **un message d'erreur** et **stoppe l'exécution** du programme
 - **Sur instruction écrite par le programmeur** : le traitement par défaut est remplacé par un **traitement adapté**

EXCEPTIONS SYSTÈME COURANTES

Exceptions système courantes

► CONSTRAINT_ERROR

déclenchée si :

- une division par zéro est effectuée
- une valeur trop grande ou trop petite est affectée à une variable
- l'accès à un vecteur est fait en dehors de son intervalle d'indices (*forcément contraint lors de la déclaration*)

Exemple 1 : division par 0 non contrôlée

```
1 with p_esiut; use p_esiut;
2 procedure div0proc is
3   x : integer := 40;
4   y, z : integer;
5 begin
6   ecrire_ligne("Entrée procédure principale");
7   ecrire("Donner une valeur du diviseur de 40 : ");
8   lire(y);
9   z := x / y;
10  ecrire(x); ecrire(" divisé par "); ecrire(y);
11  ecrire(" = ");
12  ecrire(z);
13  a_la_ligne;
14  ecrire("Sortie procédure principale");
15 end div0proc ;
```

Trace d'exécution 1/2

```
1 with p_esiut; use p_esiut;
2 procedure div0proc is
3   x : integer := 40;
4   y, z : integer;
5 begin
6   ecrire_ligne("Entrée procédure principale");
7   ecrire("Donner une valeur du diviseur de 40 : ");
8   lire(y);
9   z := x / y;
10  ecrire(x); ecrire(" divisé par "); ecrire(y);
11  ecrire(" = ");
12  ecrire(z);
13  a_la_ligne;
14  ecrire("Sortie procédure principale");
15 end div0proc ;
```

```
Entrée procédure principale
Donner une valeur du diviseur de 40 : 2
40 divisé par 2 = 20
Sortie procédure principale
```

Trace d'exécution 2/2

```
1 with p_esiut; use p_esiut;
2 procedure div0proc is
3   x : integer := 40;
4   y, z : integer;
5 begin
6   ecrire_Ligne("Entrée procédure principale");
7   ecrire("Donner une valeur du diviseur de 40 : ");
8   lire(y);
9   z := x / y;
10  ecrire(x); ecrire(" divisé par "); ecrire(y);
11  ecrire(" = ");
12  ecrire(z);
13  a_la_ligne;
14  ecrire("Sortie procédure principale");
15 end div0proc ;
```

```
Entrée procédure principale
Donner une valeur du diviseur de 40 : 0
raised CONSTRAINT_ERROR : div0proc.adb:9
divide by zero
```

Exemple 2 : accès non contrôlé à un vecteur

```
1 with p_esiut; use p_esiut;
2 procedure sortievect is
3   tab : array(1..10) of integer := (1,2,3,4,5,6,7,8,9,10);
4   i : integer;
5 begin
6   ecrire_ligne("Entrée procédure principale");
7   ecrire("Pour quel indice voulez-vous la valeur : ");
8   lire(i);
9   ecrire(x); ecrire(i); ecrire("est égal à :");
10  ecrire(tab(i)); a_la_ligne;
11  ecrire("Sortie procédure principale");
12 end sortievect;
```

Trace d'exécution 1/2

```
1 with p_esiut; use p_esiut;
2 procedure sortievect is
3   tab : array(1..10) of integer := (1,2,3,4,5,6,7,8,9,10);
4   i : integer;
5 begin
6   ecrire_ligne("Entrée procédure principale");
7   ecrire("Pour quel indice voulez-vous la valeur : ");
8   lire(i);
9   ecrire("tab("); ecrire(i); ecrire(") est égal à :");
10  ecrire(tab(i)); a_la_ligne;
11  ecrire("Sortie procédure principale");
12 end sortievect;
```

```
Entrée procédure principale
Pour quel indice voulez-vous la valeur : 4
tab(4) est égal à : 4
Sortie procédure principale
```

Trace d'exécution 2/2

```
1 with p_esiut; use p_esiut;
2 procedure sortievect is
3   tab : array(1..10) of integer := (1,2,3,4,5,6,7,8,9,10);
4   i : integer;
5 begin
6   ecrire_ligne("Entrée procédure principale");
7   ecrire("Pour quel indice voulez-vous la valeur : ");
8   lire(i);
9   ecrire("tab("); ecrire(i); ecrire(") est égal à :");
10  ecrire(tab(i)); a_la_ligne;
11  ecrire("Sortie procédure principale");
12 end sortievect;
```

```
Entrée procédure principale
Pour quel indice voulez-vous la valeur : 13
tab(13) est égal à :
raised CONSTRAINT_ERROR :
sortievect.adb:10 index check failed
```

Exemple 3 : division par 0 programmée

```
1 with p_esiut; use p_esiut;
2 procedure div0static is
3   x : integer := 40;
4   z : integer;
5 begin
6   ecrire_ligne("Entrée procédure principale");
7   z := x / 0 ;
8   ecrire("Sortie procédure principale");
9 end div0static;
```

Le compilateur est capable de détecter la division par zéro programmée en ligne 7...

Trace de compilation

```
1 with p_esiut; use p_esiut;
2 procedure div0static is
3   x : integer := 40;
4   z : integer;
5 begin
6   ecrire_ligne("Entrée procédure principale");
7   z := x / 0 ;
8   ecrire("Sortie procédure principale");
9 end div0static;
```

```
xxxxxx@transit:~/Ada$ gm div0static.adb
```

```
Compiling: div0static.adb (source file time stamp: 2019-08-11 11:15:20)
```

```
7.  z := x / 0;
    |
    >>> warning: division by zero
    >>> warning: "Constraint_Error" will be raised at run time
```

```
9 lines: No errors, 2 warnings
```

Exemple 4 : accès à un vecteur hors bornes

```
1 with p_esiut; use p_esiut;
2 procedure sortievectstatic is
3   tab : array(1..10) of integer := (1,2,3,4,5,6,7,8,9,10);
4 begin
5   ecrire_ligne("Entrée procédure principale");
6   ecrire("tab(13) est égal à "); ecrire(tab(13));
7   a_la_ligne;
8   ecrire("Sortie procédure principale");
9 end sortievectstatic;
```

Le compilateur est capable de détecter l'accès au vecteur hors de son intervalle d'indices, **programmé** en ligne 6...

Trace de compilation

```
1 with p_esiut; use p_esiut;
2 procedure sortievectstatic is
3   tab : array(1..10) of integer := (1,2,3,4,5,6,7,8,9,10);
4 begin
5   ecrire_ligne("Entrée procédure principale");
6   ecrire("tab(13) est égal à "); ecrire(tab(13));
7   a_la_ligne;
8   ecrire("Sortie procédure principale");
9 end sortievectstatic;
```

```
xxxxx@transit:~/Ada$ gm sortievectstatic.adb

Compiling: sortievectstatic.adb (source file time stamp: 2019-08-11 11:05:33)

6. ecrire("tab(13) est égal à "); ecrire(tab(13)); a_la_ligne;
    |
    >>> warning: value not in range of subtype of "Standard.Integer"
        defined at line 3
    >>> warning: "Constraint_Error" will be raised at run time

9 lines: No errors, 2 warnings
```

Exceptions système courantes

► PROGRAM_ERROR

déclenchée si :

- on essaye d'exécuter une action erronée

*(par exemple si une fonction n'a pas d'instruction **return** prévue pour certaines valeurs d'un de ses paramètres effectifs)*

Exemple : fonction sans résultat

```
1 procedure foncsansresult is
2   adult : boolean;
3   function majeur(x : in positive) return boolean is
4     --{} => {résultat = vrai si x >=18, faux sinon}
5   begin
6     if x >= 18 then return true; end if;
7   end majeur;
8 begin
9   adult := majeur(12);
10 end foncsansresult ;
```

Le compilateur est capable de détecter que la fonction n'a pas de résultat dans le cas où $x < 18$...

Trace de compilation

```
1 procedure foncsansresult is
2   adult : boolean;
3   function majeur(x : in positive) return boolean is
4     --{} => {résultat = vrai si x >=18, faux sinon}
5   begin
6     if x >= 18 then return true; end if;
7   end majeur;
8 begin
9   adult := majeur(12);
10 end foncsansresult ;
```

```
xxxxx@transit:~/Ada$ gm foncsansresult.adb
```

```
Compiling: foncsansresult.adb (source file time stamp: 2019-08-11 14:22:20)
```

```
6.           if x >= 18 then return true;
              |
```

```
>>> warning: "return" statement missing following this statement
>>> warning: Program_Error may be raised at run time
```

```
10 lines: No errors, 2 warnings
```

Exceptions système courantes

► **STORAGE_ERROR** :

- déclenchée lorsque le système manque de mémoire (pour les allocations dynamiques)

► **DATA_ERROR, STATUS_ERROR, MODE_ERROR, NAME_ERROR, DEVICE_ERROR, END_ERROR** :

- exceptions spécifiques aux **opérations d'E/S**.
- définies dans les spécifications des modules `Ada.Text_io`, `Ada.Sequential_io`, `Ada.Direct_io`

À titre d'exemple regarder comment le package `p_eslut` est réalisé à partir du module `Ada.Text_io` et comment il utilise les exceptions pour contrôler les opérations d'E/S

TRAITEMENT D'UNE EXCEPTION

Traitement d'une exception

- ▶ Ada permet au programmeur de définir le traitement à accomplir lors de la détection d'une exception au cours de l'exécution d'un programme
- ▶ Il est possible de définir à différents endroits du programme, des **traitements différents** pour des exceptions semblables
 - **globalement** (à la fin du programme principal)
 - **localement** (à la fin d'un sous-programme ou d'un bloc d'instructions défini explicitement)

Traitement d'une exception

- ▶ Le traitement doit être précédé du mot clé **exception** et respecter la forme suivante :

```
exception  
  when Nom_D_Exception1 => Traitement1  
  when Nom_D_Exception2 => Traitement2
```

- ▶ On peut utiliser l'opérateur « | » et **when others**
 - | pour faire un même traitement pour plusieurs exceptions
 - **when others** pour faire un même traitement pour les exceptions non nommées explicitement dans les **when** précédents
- ▶ Le traitement doit être programmé avant le **end** final de l'ensemble des instructions ciblées par le traitement

Traitement d'une exception

- Si l'on veut définir un traitement d'exception pour un ensemble d'instructions **plus petit** qu'un programme ou sous-programme complet, il faut définir **explicitement** un **bloc d'instructions**



1. Encadrer l'ensemble d'instructions où l'on veut traiter une ou plusieurs exceptions par les mots **begin** et **end** (\Rightarrow *création explicite d'un bloc d'instructions*)
2. Définir le traitement des exceptions juste avant le **end** de fin du bloc défini.

Exemple de traitement global

```
1  with p_esiut; use p_esiut;
2  procedure Lecturenote1 is
3      N : float range 0.0..20.0;
4  begin --début bloc
5      ecrire("Donner une note entre 0.0 et 20.0 : ");
6      lire(N);
7      ecrire("La note saisie est : ");
8      ecrire(N); a_la_ligne;
9      exception --traitement exception
10         when Constraint_Error =>
11             ecrire_ligne("La note doit être entre 0 et 20");
12             ecrire_ligne("Arrêt du programme");
13 end Lecturenote1; --fin bloc
```

Traces d'exécution

► Différents cas...


```
xxxxx@transit:~/Ada$ ./lecturenote1
Donner une note entre 0.0 et 20.0 : 5
La note saisie est 5.00
```

```
xxxxx@transit:~/Ada$ ./lecturenote1
Donner une note entre 0.0 et 20.0 : 10.75
La note saisie est 10.75
```

```
xxxxx@transit:~/Ada$ ./lecturenote1
Donner une note entre 0.0 et 20.0 : 30
La note saisie doit être entre 0 et 20
Arrêt du programme
```

Exemple de traitement local 1

```
1 with p_esiut; use p_esiut;
2 procedure Lecturenote2 is
3   N : float range 0.0..20.0;
4   begin --début bloc
5     begin --début nouveau bloc explicite
6       ecrire("Donner une note entre 0.0 et 20.0 : ");
7       lire(N);
8     exception --traitement local exception
9       when Constraint_Error =>
10        ecrire_ligne("La note doit être entre 0 et 20");
11        ecrire_ligne("Note forcée à 10 !"); N := 10.0;
12        ecrire_ligne("L'exécution continue...");
13    end; --fin bloc explicite
14    ecrire("La note prise en compte est : ");
15    ecrire(N); a_la_ligne;
16 end Lecturenote2; --fin bloc
```



contrôle de
saisie avec
forçage en
cas d'erreur

Traces d'exécution

► Différents cas...


```
xxxxx@transit:~/Ada$ ./lecturenote2
Donner une note entre 0.0 et 20.0 : 5
La note prise en compte est 5.00
```

```
xxxxx@transit:~/Ada$ ./lecturenote2
Donner une note entre 0.0 et 20.0 : 10.75
La note prise en compte est 10.75
```

```
xxxxx@transit:~/Ada$ ./lecturenote2
Donner une note entre 0.0 et 20.0 : 30
La note saisie doit être entre 0 et 20
Note forcée à 10 !
L'exécution continue...
La note prise en compte est 10.00
```

Exemple de traitement local 2

```
1  with p_esiut; use p_esiut;
2  procedure lecturenote2 is
3      N : float range 0.0..20.0;
4  begin --début bloc
5      loop
6          begin --début nouveau bloc explicite
7              ecrire("Donner une note entre 0.0 et 20.0 : ");
8              lire(N);
9              exit; -- ne sera pas exécuté si l'exception est levée
10         exception --traitement local exception
11             when Constraint_Error =>
12                 ecrire_ligne("La note doit être entre 0 et 20");
13         end; --fin bloc explicite
14     end loop;
15     ecrire("La note saisie est : ");
16     ecrire(N); a_la_ligne;
17 end lecturenote2; --fin bloc
```



contrôle de
saisie en
boucle tant
qu'il y a erreur

Traces d'exécution

- ▶ Répétition de la saisie tant qu'elle n'est pas valide...

```
xxxxx@transit:~/Ada$ ./lecturenote2
Donner une note entre 0.0 et 20.0 : 30
La note saisie doit être entre 0 et 20
Donner une note entre 0.0 et 20.0 : -2
La note saisie doit être entre 0 et 20
Donner une note entre 0.0 et 20.0 : 15
La note saisie est 15.00
```

EXCEPTIONS DÉFINIES PAR LE PROGRAMMEUR

Types d'exceptions

- ▶ On peut distinguer deux sortes d'exceptions :
 - Celles déclenchées par le système sans demande explicite du programmeur
 - Celles définies et déclenchées explicitement par le programmeur : on parle d'exceptions **programmées** ou d'exceptions **utilisateur**

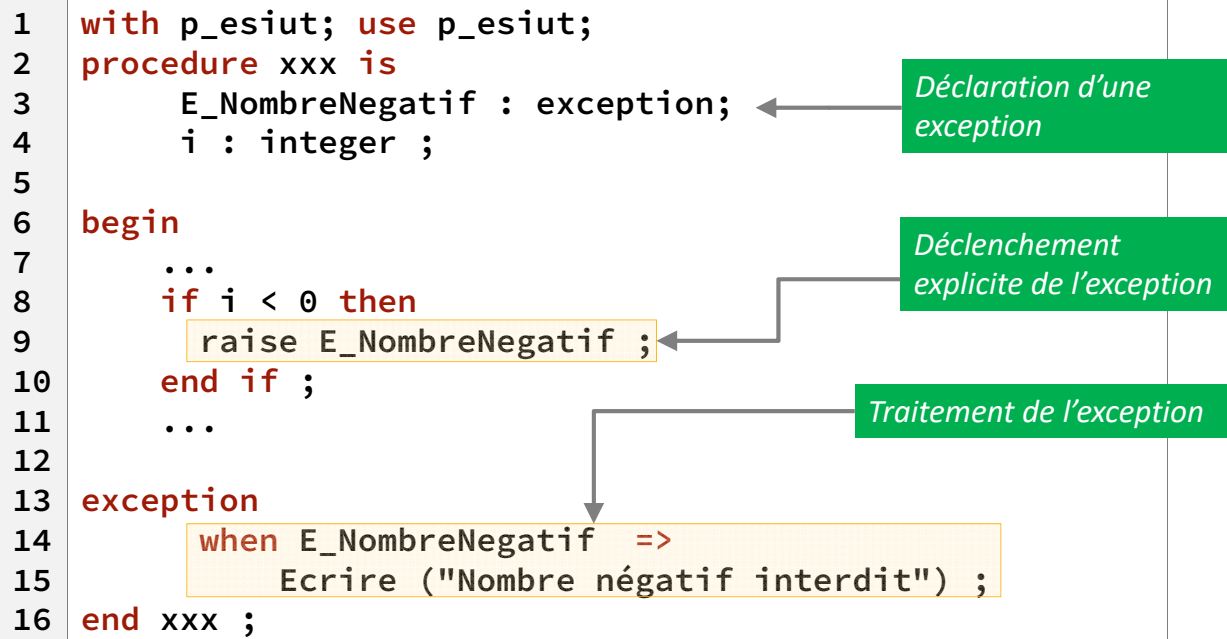
Exceptions programmées

- ▶ Pour définir des exceptions par programme, il faut :
 - Déclarer des variables de type **exception**

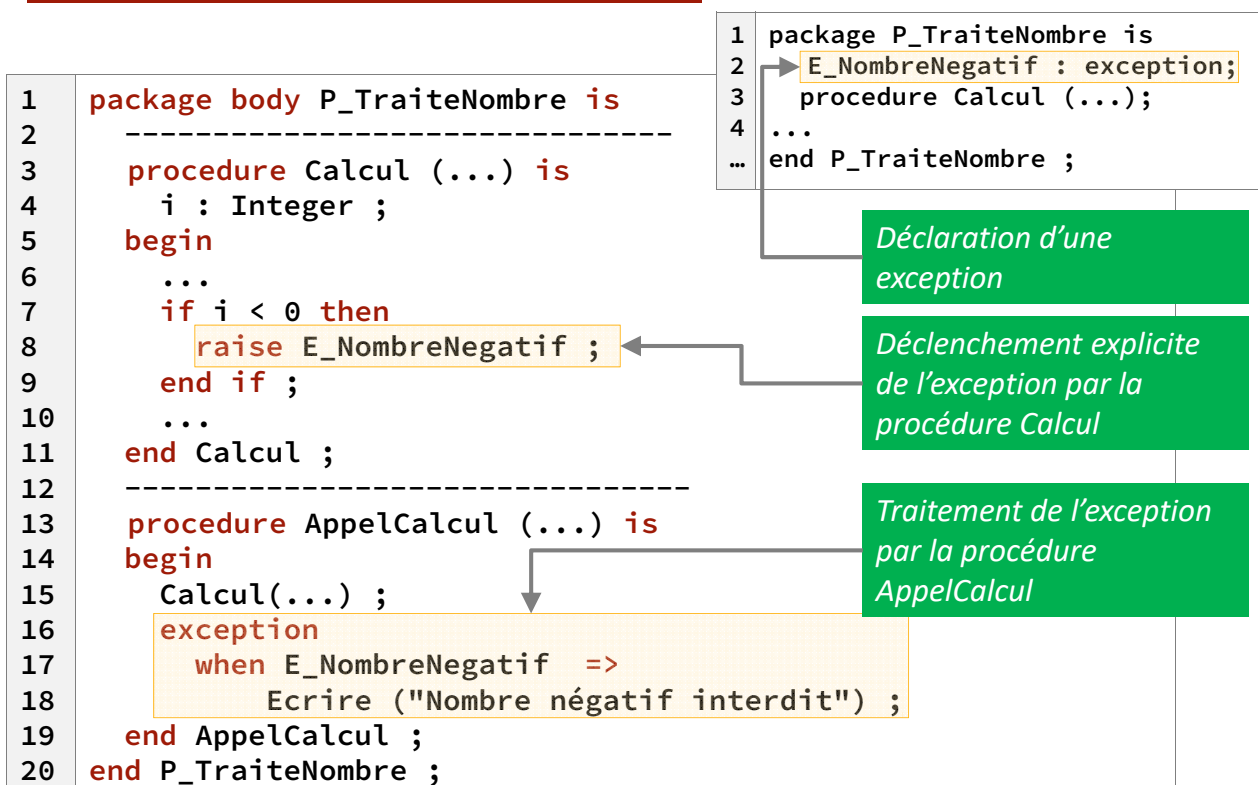
```
nom_exception : exception;
```
 - Déclencher explicitement ces exceptions par l'instruction **raise**

```
raise nom_exception ;
```
- ▶ Convention de nommage :
 - les noms d'exceptions seront préfixées par **E_** (exemple : **E_MonErreur**)

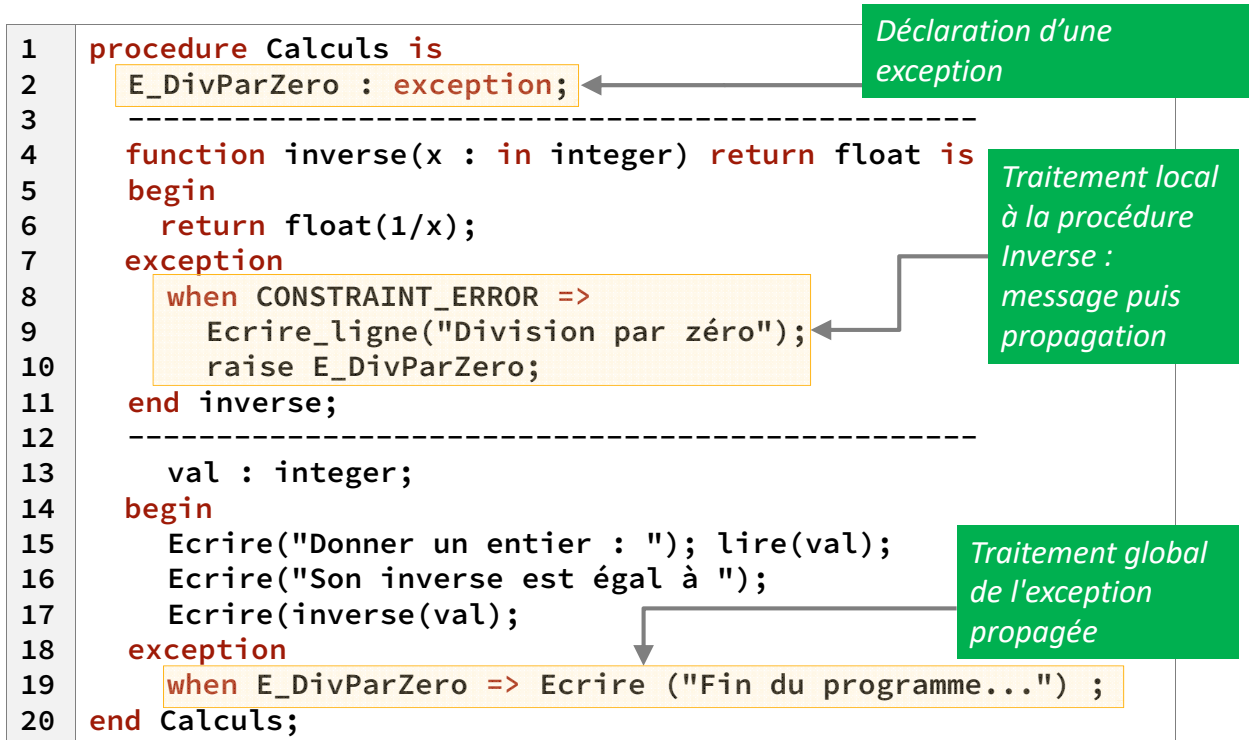
Exemple 1



Exemple 2



Exemple 3



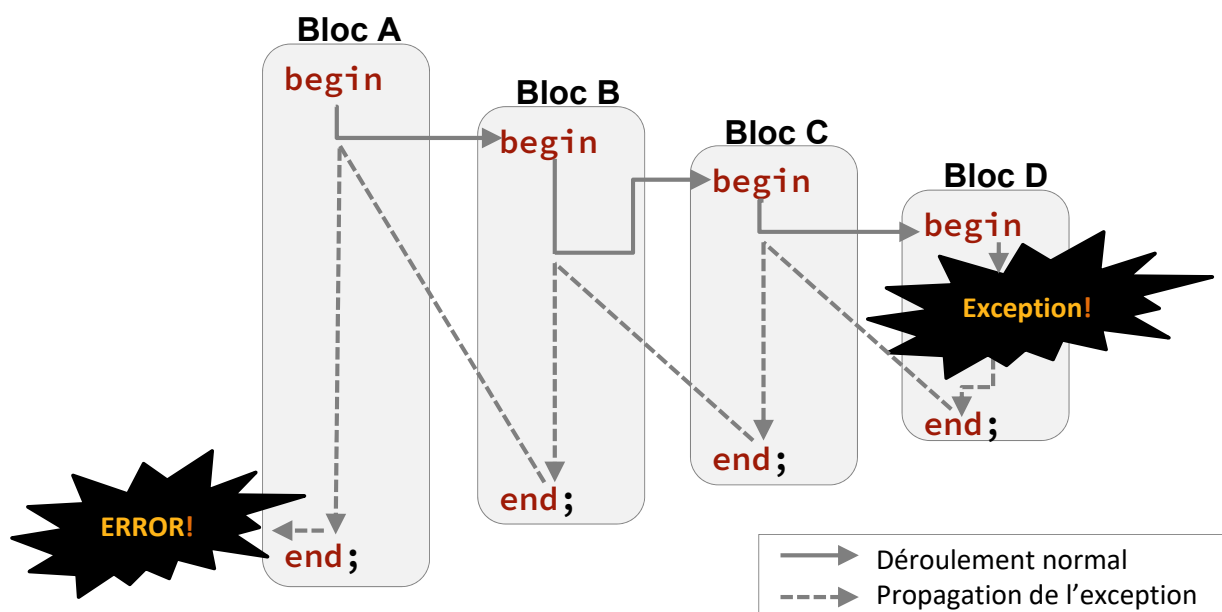
PROPAGATION DES EXCEPTIONS

Propagation des exceptions

- ▶ Lorsqu'une exception est traitée dans un bloc d'instructions :
 - si elle ne se produit pas, l'exécution du bloc se déroule normalement
 - si elle se produit, elle est immédiatement traitée avant de quitter le bloc
- ▶ Lorsqu'une exception se produit dans un bloc d'instructions où son traitement n'est pas prévu, elle se propage de blocs en blocs...
 - si un des blocs a prévu son traitement, l'exception est traitée avant de quitter ce bloc
 - si aucun bloc n'a prévu son traitement, le programme s'arrête avec un message d'erreur système, relatif à l'exception qui n'a pas été traitée.

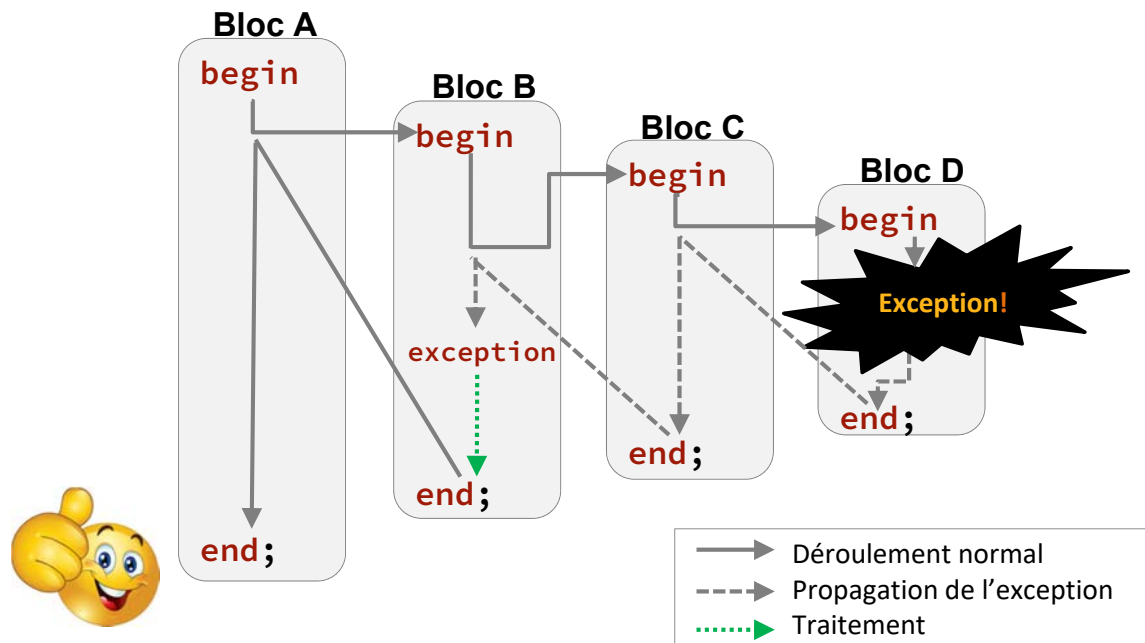
Propagation d'une exception ...

- ▶ ... non interceptée



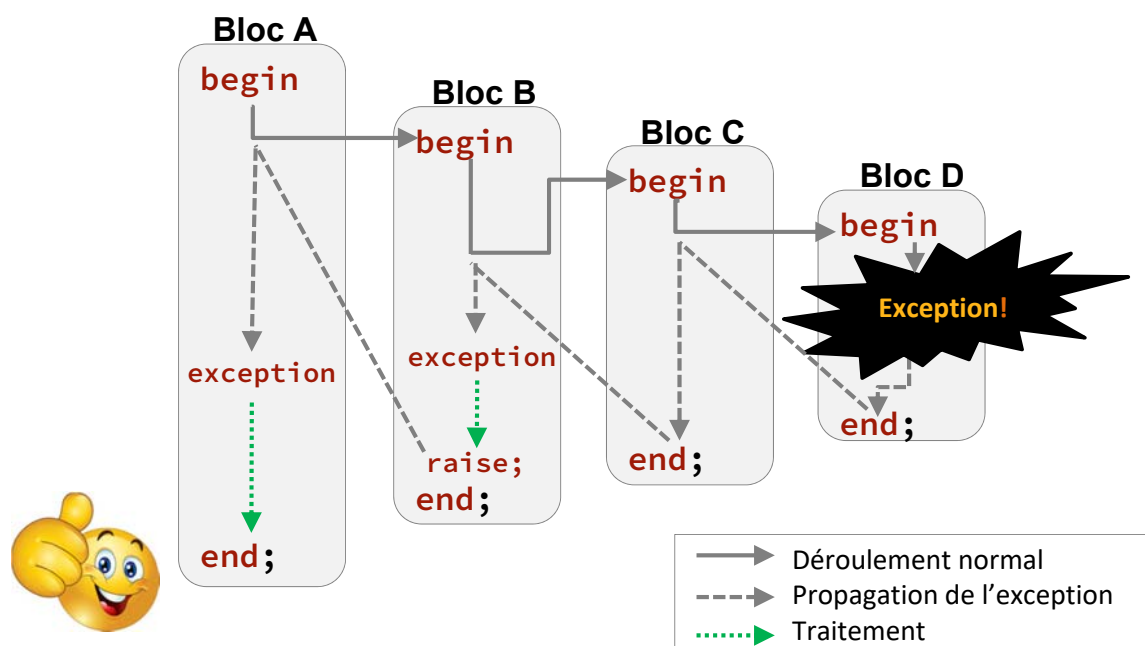
Propagation d'une exception ...

- ... interceptée et traitée



Propagation d'une exception ...

- ... interceptée, traitée, explicitement propagée et retraitée



A RETENIR...

Ce qu'il faut retenir

- ▶ Une exception est un **événement**, portant un nom
 - exemple **CONSTRAINT_ERROR**
- ▶ Une exception **programmée** doit être **déclarée**, comme toutes les autres entités d'Ada
- ▶ Le déclenchement d'une exception signale l'apparition d'une **situation exceptionnelle**
 - Ceci peut arriver implicitement (si une règle du langage n'est pas satisfaite) ou explicitement si une instruction **raise** est exécutée.
- ▶ Tout traitement qui déclenche une exception est **interrompu**
 - Le contrôle est alors passé à la séquence de traitement d'exception - **si elle a été prévue** - selon les règles de portée correspondant à l'imbrication de blocs.