Exceptions

Chapitre 8

Qu'est-ce qu'une exception ?

Une *exception* est une erreur ou bien une action inattendue qui se produit durant l'exécution d'un programme

Une *exception* se traduit par un signal qui interrompt le déroulement normal d'un programme

Intérêt

Contrôler le déroulement des calculs en localisant des évènements exceptionnels pour :

- Traiter des erreurs
- Arrêter un calcul

Rendre plus lisible et plus structuré le texte d'une unité en séparant traitement du cas général et traitement des cas exceptionnels

Point de vue théorique

On peut assimiler un programme au calcul d'une fonction en mathématiques

Tout programme établit une correspondance entre un domaine (les données du programme) et un co-domaine (les résultats)

De fait, les programmes ne traduisent que des fonctions totales (partout définies)

Rien interdit à un programme de s'exécuter sur des données particulières pour lesquelles la fonction n'est pas définie (ex: division par zéro)

Vers des programmes robustes

Le mécanisme des exceptions dans les LP (Ada, C++, Java) permet de prendre en compte le cas des données exceptionnelles (inattendues) de manière à construire des programmes robustes

Par exemple, lorsqu'un programme demande à l'utilisateur de saisir des données numériques dans un certain intervalle, il doit (pour être robuste) déterminer si elles sont numériques et si elles appartiennent à l'intervalle. Dans le cas contraire, il peut prévoir une action spécifique; par exemple, demander une nouvelle saisie.

Gestion traditionnelle des cas d'erreur

La manière traditionnelle de prendre en compte les cas d'erreur est d'utiliser les structures de contrôle classiques

Exemple :

supposons qu'Ada ne traite pas les exceptions. La procédure `get` devrait posséder un paramètre représentant l'erreur éventuelle (le paramètre `C` joue ce rôle)

```
procedure get(X:out String;C:out Integer)
```

Lecture d'une chaîne de caractères

```
declare
 A:String(1..10);
 code:Integer;
begin
 get(A,code);
 case code is
   when –1 => caractère numérique
   when +1 => la chaîne lue ne possède pas 10 caractères
   when others => lecture OK
 end case;
end;
```

Inconvénients

Le code de gestion de erreurs se mélange au code normal.

- Ajout de paramètres non applicatifs aux procédures (le paramètre `code`)
- Surplus de code (`case`)
- Risque de propagation des erreurs => prolifération des paramètres supplémentaires

Exceptions prédéfinies

Elles sont présentes dans l'environnement initial et déclenchées (levées) :

- En cas de manque d'espace mémoire (`STORAGE_ERROR`)
- En cas de tentative de violation d'une structure de contrôle ou d'appel d'une procédure non encore élaborée (`PROGRAM_ERROR`)
- Si la valeur d'une expression dépasse les bornes de l'ensemble des valeurs possibles (`CONSTRAINT_ERROR`)

Exception prédéfinie : exemple

```
with Ada.Text_io;
with Ada.Integer_Text_io;
use Ada.Text_io;
use Ada.Integer_Text_io;
procedure divparzero is
   res,num,den:Integer;
begin
   put( "numérateur : " );get(num);
   put( "dénominateur : " );get(den);
   res:=num/den;
   put( "résultat=" );put(res);
   new_line;
end divparzero;
```

Exception prédéfinie : suite

Si la valeur saisie pour le dénominateur est `0` :

- Le programme ne termine pas son exécution (plantage)
- L'exception prédéfinie `CONSTRAINT_ERROR` est levée
- Le message :
  `raised` `CONSTRAINT_ERROR : divparzero.adb:11` est affiché

Déclaration

Syntaxe

```
<déclaration_exception>::=
        <identificateur_exception>:exception;
```

Sémantique

L'identificateur est introduit dans l'environnement de l'état courant

Exemple

```
erreur,divparzero:exception;
```

Type exception

L'ensemble des valeurs du type exception est l'ensemble des signaux nommés

La seule opération sur les exceptions est :

- la levée (déclenchement)

Levée d'une exception

Lever une *exception* dans le code d'un programme, c'est signaler qu'une situation anormale est détectée.

Syntaxe

```
<levée_exception>::=
    raise <identificateur_exception>
```

Sémantique

- L'exécution de l'instruction **raise** interrompt le déroulement normal du programme et transfert le contrôle à un traitement de cette exception

Quel traitement ?

Le traitement d'une exception peut :

- être prédéfini (cas des exceptions prédéfinies)
- être codé par le programmeur
- consister en une propagation du signal vers les unités appelantes

Levée d'une exception : exemple (1/3)

```
with Ada.Text_io;
use Ada.Text_io;
with Ada.Characters.Handling;
use Ada.Characters.Handling;

-- saisie d'une chaîne de caractères alphabétiques
procedure saisie_alpha is
   ligne:String(1..80);
   lg:Natural;
   non_alpha:exception;
begin
```

Levée d'une exception : exemple (2/3)

```
   put( "entrer une chaine : " );
   get_line(ligne,lg);
   declare
      chaine:String(1..lg):=ligne(1..lg);
   begin
      for i in chaine'range loop
         if is_digit(chaine(i))
          then raise non_alpha;
          end if;
      end loop;
   end;
end saisie_alpha;
```

Levée d'une exception : exemple (3/3)

Pour la donnée saisie

```
Nestor Martin
```

le résultat est :

```
Nestor Martin
```

Pour la donnée

```
Nest0r Mart1
```

le résultat est :

```
raised SAISIE_ALPHA.NON_ALPHA:saisie_alpha.adb:18
```

Récupération d'une exception

Récupérer une exception, c'est se donner la possibilité d'associer un traitement particulier à la situation détectée

Permet de prévenir une interruption brutale du programme en prévoyant un mode de fonctionnement dégradé ou de substitution

Récupération d'une exception : syntaxe

Syntaxe

```
  <récupération_exception>::=
  exception
  when <identificateur_exception>=><traitement>;
 [when <identificateur_exception>=><traitement>;]
 [when others=><traitement>;]
```

La partie `exception` constitue la partie optionnelle d'un bloc :

```
declare
begin
exception
end;
```

Récupération d'une exception : sémantique

Les identificateurs d'exception servent de filtre pour l'exécution des traitements

Ils doivent donc être présents dans l'environnement de récupération

Le filtre universel **`others`** permet de filtrer toutes les exceptions présentes ou non dans l'environnement

Récupération d'une exception : exemple 1

```
function demain(X:Jour) return Jour is
begin
   return Jour'succ(X);
exception
   when CONSTRAINT_ERROR => return Jour'first;
end demain;
```

Récupération d'une exception : exemple 2

```
declare
 chaine:String(1..lg):=ligne(1..lg);
begin
 for i in chaine'range loop
   if is_digit(chaine(i))
   then raise non_alpha;
   end if;
 end loop;
   put_line(chaine);
exception
   when non_alpha=>
    put ( "certains caracteres saisis sont numeriques" );
    new_line;
end;
```

Récupération d'une exception : exemple 2 (suite)

Pour la donnée saisie

```
Nestor Martin
```

le résultat est :

```
Nestor Martin
```

Pour la donnée

```
Nest0r Mart1
```

le résultat est :

```
certains caracteres saisis sont numériques
```

```
--import de Ada.Text_Io et Ada.Characters.Handling
procedure saisie_alpha_2 is
 ligne:String(1 .. 80); lg:Natural; non_alpha:exception;
begin
   loop
   begin
     put( "entrer une chaine : " ); get_line(ligne,lg);
     declare
       chaine:String(1 .. Lg):=ligne(1 .. lg);
     begin
       for I in chaine'range loop
         if is_digit(chaine(I)) then raise non_alpha; end if;
       end loop;
       put_line(chaine);exit;
     end;
   exception
     when non_alpha=>
     put_line( "certains caracteres saisis sont numeriques" );
     put_line( "recommencer la saisie !" );
   end;
   end loop;
end saisie_alpha_2;
```

Portée des exceptions

Une exception est connue par son nom dans son unité de déclaration ainsi que dans toutes les unités englobées

Si une exception est levée dans une unité, le traitement est interrompu et le contrôle est transféré au traitant d'exception

Si l'exception n'est pas traitée dans cette même unité, alors elle est propagée vers le traitant d'exception de l'unité appelante et ainsi de suite vers l'unité principale jusqu'à l'exécutif Ada.

```
declare
  erreur:exception;
  procedure B is
  begin
    raise erreur;
  end B;
  procedure C is
  begin
    B;
  exception
    when erreur=>traitement_erreur_C;raise;
  end C;
  procedure A is
  begin
    C;
  exception
    when erreur=>traitement_erreur_A;
  end A;
begin
  A;
end;
```

Propagation d'une exception en dehors de sa portée

Lorsqu'une exception n'est pas traitée dans son unité de déclaration, elle est propagée vers son unité appelante
=> elle se retrouve *en dehors de sa portée*

Pour être traitée, elle doit alors être filtrée par le filtre universel
`others`

```
procedure recupere is
  procedure declenche is
    exception_locale:exception;
  begin
    raise exception_locale;
  end declenche;
begin
  declenche;
exception
  when others=> traitement de toutes les exceptions;
end recupere;
```

Exception dans une déclaration

Si une exception est déclenchée dans une déclaration, elle est propagée vers l'unité appelante

Dans l'exemple qui suit, le résultat affiché sera :

```
recupere externe
```

```
declare
  ex:exception;
  function declenche return Integer is
  begin
    raise ex;
  end declenche;
begin
  declare
    X:Integer:=declenche;
  begin
    put( "interne" );
  exception
    when ex=>put( "recupere interne" );
  end;
exception
  when ex=>put( "recupere externe" );
end;
```

Exception levée dans un traitant d'exception

Une exception levée dans un traitant d'exception est propagée vers l'unité appelante

Pour un traitement couche par couche de l'exception, on peut propager la même exception en la levant de nouveau par `raise;`

Lorsque plusieurs exceptions sont filtrées par `others`, il n'y a pas moyen de nommer explicitement l'exception survenue. Dans ce cas, il est possible de la lever de nouveau par `raise;`

```
declare
begin
 declare
   X:Integer;
 begin
  declare
    ex2,ex1:exception;
  begin
    get(X);
    if X=0 then raise ex1; else raise ex2 end if;
  exception
    when ex2=>put( "traitement ex2" );raise;
    when ex1=>put( "traitement ex1" );raise;
  end;
 exception
  when others=>put( "traitement complémentaire de ex1 si X=0" );
   put( "ou traitement complémentaire de ex2 si x/=0" );raise;
 end;
exception
 when others=>put( "traitement complémentaire de ex1 si X=0" );
             put( "ou traitement complémentaire de ex2 si x/=0" );
end;
```

Exception : utilisation

Pour éviter un « plantage » et améliorer la structure et la lisibilité

```
declare
  N:Integer;
begin
  loop
    begin
     get(N);exit;
    exception
     when   DATA_ERROR=>skip_line;put_line( "recommencez!" );
    end;
  end loop;
end;
```

Exception et récursion (1/5)

```
declare
  ex:exception;
  function fac(N:Integer) return Integer is
  begin
    if N=1 then raise ex;
    else return N*fac(N-1);
    end if;
  exception
    when ex => Ada.Integer_Text_io.put(N);raise;
  end fac;
begin
  put(fac(3));
exception
  when ex => Ada.Text_io.put_line( "stop" );
end;
```

Exception et récursion (2/5)

```
Le résultat affiché sera :

1       2       3stop
```

Exception et récursion (3/5)

```
declare
  ex:exception;
  function fac(N:Integer) return Integer is
  begin
    if N = 1 then raise ex;
    else return N*fac(N-1);
    end if;
  exception
    when ex => Ada.Integer_Text_io.put(N);
  end fac;
begin
  put(fac(1));
exception
  when ex=>Ada.Text_io.put_line( "stop" );
end;
```

Exception et récursion (4/5)

```
Résultat de l'exécution

1

raised PROGRAM_ERROR : fonction.adb:7
```

Exception et récursion (5/5)

Puisque l'exécution de `fac(1)` se termine par un traitement d'exception non propagée, la fonction doit retournée une valeur de type `Integer` pour que l'exécution se poursuive normalement. Le compilateur affiche des "warnings" pour signaler que l'instruction **`return`** manque à la fin du traitant d'exception de la fonction `fac` et que l'exception `PROGRAM_ERROR` pourra être levée.

Dans l'exemple précédent l'exécution se poursuivait jusqu'à la fin, de traitant d'exception en traitant d'exception. Dans ce cas tout se passe comme si l'appel à la fonction `fac` se terminait avec le programme.

Exemple détaillé (1)

Réalisation d'un programme de `login` sur une machine hôte.

- Le programme affiche tout d'abord l'invite :
  `login:`

- L'utilisateur tape alors son nom d'utilisateur

Si ce nom est inconnu du système, la tentative est arrêtée,
l'exception `login_errone` est levée et propagée

Exemple détaillé (2)

Si le nom est connu du système, le programme affiche :

```
passwd:
```

et l'utilisateur est invité à taper son mot de passe.

- Si le mot de passe est correct, c'est terminé, sinon le programme imprime un message ad hoc et l'utilisateur doit retaper son mot de passe.

- L'utilisateur a trois essais, à la suite desquels l'exception `login_errone` est levée et propagée

Exemple détaillé (3)

On suppose que les procédures suivantes appartiennent à l'environnement du programme de `login` :

```
procedure teste_nom(N:in String);
   -- vérifie que N est connu du système
   -- si ce n'est pas le cas, l'exception
   -- nom_errone est levée
procedure teste_mot_de_passe
                      (M:in String;N:in String);
   -- vérifie que M est connu et associé à N
   -- sinon l'exception mot_errone est levée
```

```
with Ada.Text_io; use Ada.Text_io;with Test;use Test;

procedure login is
   nom,mot:String(1..8);lg,lgm:Natural;
   login_errone:exception;
begin
   put("login: ");get_line(nom,lg);
   teste_nom(nom(1..lg));
   put("passwd: ");

   for i in 1..3 loop
      begin
         get_line(mot,lgm);
         teste_mot_de_passe(mot(1..lgm),nom(1..lg));
         put("mot de passe : ****");
         exit;
      exception
         when mot_errone => put_line("mot de passe errone");
                            if i<3 then
                               put_line(" Essayez encore");
                               put("passwd:");
                            else raise;
                            end if;
      end;
   end loop;
```

```
exception
   when nom_errone =>
        put_line("nom "&nom(1..lg)&" inconnu");
        put("login interrompu");
        raise login_errone;
   when mot_errone => raise login_errone;
end login;
```

Exemple

Saisie et affichage d'un tableau de mesures en nombre quelconque

schéma du programme

```
begin
  get(tabMesures);
  put(tabMesures);
end;
```

On souhaite écrire un programme robuste avec contrôle de saisie :

- des mesures
- de la taille du tableau

```
with Ada.Integer_Text_io;use Ada.Integer_Text_io;
with Ada.Float_Text_io;use Ada.Float_Text_io;
with Ada.Text_io;use Ada.Text_io;

procedure mesures is
  entier_negatif:exception;
  type Tableau is array(Positive range<>) of Float;
  N:Positive;


  procedure get(t:out Tableau);
-- déclaration du corps de la procédure
-- (cf transparents suivants)

  procedure put(t:in Tableau);
-- déclaration du corps de la procédure
-- (cf transparents suivants)
```

```
begin
   put("Combien d'entiers voulez-vous saisir ?");
   loop
    begin
      get(N);
      if N<=0 or N<=Integer'last then exit;
      else raise entier_negatif; end if;
    exception
     when DATA_ERROR => put("Saisir un entier,Recommencez !");
                        skip_line;
     when entier_negatif =>
                   put("Saisir un entier positif,Recommencez !");
                   skip_line;
    end;
   end loop;
  declare
    tabMesures:Tableau(1..N);
  begin
     get(tabMesures);
     put(tabMesures);
  end;
end mesures;
```

```
procedure get(t:out Tableau) is
   i:Positive:=t'first;
begin
   loop
    begin
      put("Tapez une mesure (Float) : ");
      get(t(i));
      i:=i+1;
      if i>t'last then exit;
      end if;
    exception
     when DATA_ERROR=>
         put_line("Saisir un float:recommencez");
         skip_line;
    end;
   end loop;
end get;
```

```
procedure put(t:in Tableau) is
begin
    put("(");
    for i in t'first..t'last-1 loop
      put(t(i));put(",");
    end loop;
    put(t(t'last));
    put_line(")");
  end put;
```