

Candela Rouaux

20 de mayo.

Práctica #8

Ejercicio 1

Una excepción es una condición inesperada o inusual, que surge durante la ejecución del programa y no puede ser manejada en el contexto local

Ejercicio 2

Para que un lenguaje trate excepciones debe proveer:

- Un modo de definirlas
- Una forma de alcanzarlas, invocarlas
- Una forma de manejarlas
- Un criterio de continuación

No todos los lenguajes proveen manejo de excepciones. Por ejemplo: Pascal y C.

Ejercicio 3

Cuando un lenguaje no maneja excepciones, deben contemplarse los casos excepcionales dentro del programa. Por ejemplo, contemplar explícitamente que el divisor en una división sea distinto de 0, o la existencia de un archivo antes de abrirlo, etc. También lo que ocurre es que aborta el programa que produce el error. Se podría simular haciendo todos los controles para manejar todos los posibles casos de excepciones que pudieran ocurrir.

Se podría simular el manejo de excepciones mediante:

- El chequeo manual de las condiciones que generarian la excepción, y
- La invocación, también manual, de la rutina para manejar dicha excepción (esto sería el manejador)

Ejercicio 4

1) Existen dos modelos para saber a donde se cede el control cuando se termina

de atender una excepción:

- Terminación: Se termina la ejecución de la unidad que alcanza la excepción y se transfiere el control al manejador.
- Resolución: Se maneja la excepción y se devuelve el control al punto siguiente donde se invocó a la excepción, permitiendo la continuación.

2) Lenguajes que utilizan el modelo de Resolución:

- PLI → Los manejadores se declaran con la sentencia ON:

ON CONDITION (nombre-excepción) Manejador

→ Las excepciones se lanzan explícitamente con la palabra clave:

Signal condition (nombre-excepción)

→ A medida que se va ejecutando el proceso y se encuentran ON

CONDITION, se apila el manejador en una pila de manejadores. Cuando se levanta una excepción, se busca el nombre de la misma desde el tope de la pila hacia abajo. El primer manejador que se encuentra es el que se ejecuta.

→ Ningún proceso termina cuando se levanta una excepción. Simplemente se detecta la excepción, se la atiende y continua el flujo de la ejecución en el punto siguiente a donde se levantó la excepción.

Lenguajes que utilizan el modelo de Terminación.

- ADA → Las excepciones se definen en la zona de definición de las variables

y tienen el mismo alcance Ejemplo e.Exception. Las excepciones predefinidas no se declaran.

Los manejadores pueden encontrarse en el final de cuatro diferentes unidades de programas Bloque, Procedimiento, Paquete o Tarea.

Ejemplo: Procedure prueba,

e1,e3:Exception

begin

Exception

when e1 → Manejador1;

when others → ManejadorN;

end;

Candela Rouaux

20 de mayo

Solo se
puede relan-
zar una
excepción
dentro de
un maneja-
dor con
raise

- ↳ Se lanzan explícitamente con la palabra clave raise
 - ↳ Cuando se produce una excepción.
 - Se termina la unidad, bloque, paquete o tarea donde se alcanza la unidad.
 - Si el manejador se encuentra en ese ámbito, se ejecuta.
 - Si el manejador no se encuentra en ese lugar, la excepción se propaga dinámicamente. Esto significa que se vuelve a levantar en otro ámbito
 - Siempre tener en cuenta el alcance, puede convertirse en anónima
 - ↳ Si se desea continuar ejecutando las instrucciones de un bloque donde se lanza la excepción, es preciso "crear un bloque más interno" para las instrucciones causantes de la excepción
- CLAVE → Las excepciones que un proceso puede lanzar se declaran en su encabezado. Los manejadores se colocan al lado de una sentencia simple o compleja.

```
<sentencias> except
    when nombre-excepción. Manejador;
    ...
    when others. Manejador;
```
- ↳ Se lanzan explícitamente con la palabra clave signal.
- ↳ Al producirse una excepción; se termina la unidad y se propaga dinámicamente.
 - Si el manejador se encuentra en ese ámbito, se ejecuta y luego se pasa el control a la sentencia siguiente a la que está ligado dicho manejador.
 - Si el manejador no se encuentra en ese lugar, la excepción se propaga estáticamente en las sentencias asociadas. Esto significa que el proceso se repite para las sentencias incluidas

estáticamente.

- En caso de no encontrar ningún manejador en el procedimiento que hizo la llamada se levanta una excepción failure y devuelve el control, terminando todo el programa.

→ El modelo de terminación es a nivel de sentencia. El proceso que levanta la excepción siempre termina (aunque tenga manejadores). El proceso que tiene el manejador no termina. La ejecución continúa en la línea siguiente al manejador de la excepción.

3) El modelo de reasunción es el más inseguro. El principal problema de este modelo es que normalmente es bastante difícil "reparar" los errores que han sido elevados por el entorno de ejecución. Por ejemplo, un error aritmético de overflow que ocurra en la mitad de la evaluación de una expresión compleja puede conllevar que varios registros contengan evaluaciones parciales. Como consecuencia de la llamada al manejador, estos registros se pierden "madrugar".

Ejercicio 5

CLU	PL/I	ADA	Python
Propaga un nuevo error dinámicamente (no estáticamente) dentro de la unidad que lanzó el error y luego busca estáticamente a la ejecución en el nivel sentencia. Si no se encuentra en la unidad llamadora, se levanta la excepción failure y termina el programa.	Se detecta la excepción, se busca un manejador en la pila de manejadores, se ejecuta y continua en la ejecución en el punto siguiente a donde se levantó la excepción. Si no se encuentra el manejador, termina el programa.	Se termina el bloque que levantó la excepción y se propaga dinámicamente en busca del manejador. Si el alcance de "madrugar" si no se encuentra manejador en ese bloque try, continúa ejecutando el código debajo del bloque try que tiene el manejador que se ejecutó.	La excepción se propaga a try de "madrugar" si no se encuentra manejador en ese bloque try.

Ejercicio 6

Está simulando el modelo de reasunción porque cuando termine de ejecutarse el "manejador" se vuelve a la siguiente linea luego de la llamada a X

20 de mayo

Candela Ravaux

Para simular el modelo por terminación se podría usar break para terminar el proceso que se está ejecutando.

Ejercicios *

a) Prog Principal:

```

declare x integer;
declare b1 boolean;
declare b2 boolean;
Proc P (b1 boolean);
declare x integer;
ON CONDITION Manejador1 begin x := x + 1; end;
x := 1;
if (b1) then
  SIGNAL CONDITION Manejador1;
end,
x := x + 4;
end,
begin
  ON CONDITION Manejador2 begin x := x + 100; end;
  x = 4
  b2 := true;
  b1 := false;
  if (!b1) then
    SIGNAL CONDITION Manejador2;
  end;
  P(b2);
  write(x);
end;
```

b) Procedure Principal is

```

x : integer,
b1, b2 : boolean, e1 : Exception;
Procedure P (b1 : boolean) is
  x : integer; e2 : Exception;
begin
  x := 1;
  if (b1) then
    raise e2;
  endif;
  exception
    when e2 : begin x := x + 1; x := x + 4; end;
  end P
begin
  x = 4; b2 = true; b1 = false;
  if (!b1) then raise e1 endif;
  exception
    when e1 : begin x := x + 100; P(b2); write(x); end;
  end Principal;
```

* También se puede encerrar el raise y los manejadores en un bloque declare así continua la ejecución del proceso luego de la excepción

Ejercicio 8

a) Si $m=0$, se levanta la excepción tipo1 y se termina el proceso Proc-1 (no se ejecuta $m := m * 10$). Se busca un manejador estáticamente en la unidad llamadra. Se ejecuta el manejador asociado a la sentencia while que contiene el if. y se imprime "se produjo un error de tipo2 en Proc-2". Se continua ejecutando a partir del manejador del while. La variable m queda en 0.

b) Para simular el efecto en ADA, ningún manejador de Proc-1 debería resolver la excepción levantada. Así se propaga a Proc-2. El alcance de la excepción debe abarcar Proc-2 o poner un manejador when others y que se propague como anónima.

Ejercicio 9

Python	Java
- bloques try except	- bloques try catch
- Se levantan con raise	- Se levantan con throw.
- finally: se ejecuta aunque se levante una excepción	- finally: se ejecuta aunque se levante una excepción
- else: se ejecuta si no se levanta una excepción.	- Objeto subclase de java.lang.Object

Ejercicio 10

El modelo de excepción que implementa Ruby es por terminación. Para levantar una excepción se utiliza la palabra clave raise. Para manejar excepciones, incluimos el código que pueda generar una excepción en un bloque begin-end y osamos una o más cláusulas rescue. Una vez que la excepción es manejada, la ejecución continua inmediatamente después del bloque begin que la generó. Se pueden aplicar cláusulas rescue en un bloque begin-end. Las excepciones que no sean manejadas por una cláusula rescue fluirán a la siguiente.

Candela Ravaux

20 de mayo

Ejercicio 11

La forma más sencilla para lanzar errores es utilizando "throw". Este comando permite enviar al navegador un evento similar al que se produce cuando ocurre algo inesperado o nos encontramos ante un ingreso inesperado de datos.

La sentencia try catch consiste en un bloque try, el cual contiene una o más instrucciones, y nunguno o varios bloques catch, conteniendo sentencias que especifican qué hacer si una excepción es lanzada en el bloque try. Por último, se ejecuta el bloque finally luego de que los bloques try y catch hayan sido ejecutados pero antes de las instrucciones que encuentren a continuación de la sentencia try-catch.

Ejercicio 12

a) Camino 1:

- ↳ Se lee b ($b = \text{true}$)
- ↳ $y = 3$
- ↳ Llama a prueba 1
 - ↳ $m = 20$
 - ↳ como $b = \text{true} \rightarrow \text{raise } X$
 - ↳ Termina Prueba 1 $\rightarrow y$ vale 20
- ↳ Se ejecuta el manejador de X en Principal
 - ↳ $y = 600$
 - ↳ Imprime 600

Camino 2:

- ↳ Se lee b ($b = \text{false}$)
- ↳ $y = 1$
- ↳ Llama a prueba 1
 - ↳ $m = 20$
 - ↳ $b = \text{false} \rightarrow$ no entra al if
 - ↳ $m = 22$
 - ↳ Termina prueba 1 $\rightarrow y = 22$
- ↳ Llama a prueba 2
 - ↳ División por 0 antes del begin de prueba 2 $\rightarrow \text{raise constraint-error}$
 - ↳ Termina prueba 2
- ↳ Se ejecuta el manejador constraint-error del main
 - ↳ $y = 22 + 1 = 26$
 - ↳ Imprime 26

Comando 3:

- ↳ Se lee b (no es un valor booleano) → raise constraint-error
- ↳ Se ejecuta el manejador constraint-error de main
 - ↳ Se levanta una excepción debido a que y no está inicializada
- ↳ Termina el programa con error

b) Procedure Principal;

y: integer; b: boolean,
Procedure Prueba1 (out m: integer),
x: exception;

```
begin
  m := 20,
  if (b=true) then raise x;
  m := m + 2;
end;

Procedure Prueba2;
  a: int := 0, b := 4/a,
begin
  y := y + 8,
  exception
    when constraint-error => y := y + 10,
end;

begin
  read(b);
  y := 1;
  Prueba1(y);
  Prueba2;
  write(y),
  exception
    when constraint-error =>
      begin
        y := y + 4;
        write(y);
      end;
    when x =>
      begin
        y := y * 30;
        write(y);
      end;
  end;
end;
```

* Como x está definido dentro de prueba1, cuando la excepción se propaga al procedure Principal, la excepción sale como anónima.
Para que pueda manejarse, se debería agregar un when others en el programa Principal.

Ejercicio 13

a)

• Comando 1:

Llama a uno

y < x → entra al while

y <= 0 → no entra al if

Llama a dos

m = 0 → entra al if

se levanta la excepción error!

se termina el proceso que la levantó (Dos)

No se encuentra manejador asociado al llamado de Dos → se propaga estáticamente

Candela Rouaux

20 de mayo

Se ejecuta el manejador asociado al while
Relanza la excepción
Se termina el proceso que la levantó (Uno)
Se ejecuta el manejador asociado al llamado de Uno
Llama a Dos
 $m < 0 \rightarrow$ no entra al if
Termina Dos
Termina Main

Camino 2:

Llama a Uno
 $y < x \rightarrow$ entra al while
 $y < 0 \rightarrow$ no entra al if
Llama a Dos
 $m = 0 \rightarrow$ entra al if
se levanta la excepción error!
se termina el proceso que la levantó (Dos)
No se encuentra manejador asociado al llamado de Dos \rightarrow se propaga estáticamente
Se ejecuta el manejador asociado al while
Relanza la excepción
Se termina el proceso que la levantó (Uno)
Se ejecuta el manejador asociado al llamado de Uno
Llama a Dos
 $m = 0 \rightarrow$ entra al if
se levanta la excepción error!
se termina el proceso que la levantó (Dos)
Se ejecuta el manejador asociado al llamado de Dos
Relanza la excepción
se termina el proceso que la levantó (Main)
Termina Main con error

Camino 3:

Llama a Uno
 $y < x \rightarrow$ entra al while
 $y < 0 \rightarrow$ no entra al if
Llama a Dos
 $m < 0 \rightarrow$ no entra al if
Termina Dos
Termina Uno
Llama a Dos
 $m < 0 \rightarrow$ no entra al if
Termina Dos
Termina Main

Camino 4:

Llama a Uno
 $y < x \rightarrow$ entra al while
 $y < 0 \rightarrow$ no entra al if
Llama a Dos
 $m < 0 \rightarrow$ no entra al if

Termina Dos

Termina Uno

Llama a Dos

$m = 0 \rightarrow$ entra al if

se levanta la excepción error!

se termina el proceso que la levantó (Dos)

se ejecuta el manejador asociado al llamado de Dos

Relanza la excepción

se termina el proceso que la levantó (Main)

Termina Main con error

• Camino 5:

Llama a Uno

$y < x \rightarrow$ entra al while

$y = 0 \rightarrow$ entra al if

se levanta la excepción errors

se termina el proceso que la levantó (Uno)

se ejecuta el manejador asociado al llamado de Uno

Llama a Dos

$m < 0 \rightarrow$ no entra al if

Termina Dos

Termina Main

• Camino 6:

Llama a Uno

$y < x \rightarrow$ entra al while

$y = 0 \rightarrow$ entra al if

se levanta la excepción errors

se termina el proceso que la levantó (Uno)

se ejecuta el manejador asociado al llamado de Uno

Llama a Dos

$m = 0 \rightarrow$ entra al if

se levanta la excepción errors

se termina el proceso que la levantó (Dos)

se ejecuta el manejador asociado al llamado de Dos

Relanza la excepción

se termina el proceso que la levantó (Main)

Termina Main con error

• Camino 7:

Llama a Uno

$y > x \rightarrow$ no entra al while

Termina Uno

Llama a Dos

$m < 0 \rightarrow$ no entra al if

Termina Dos

Termina Main

• Camino 8:

Llama a Uno

$y > x \rightarrow$ no entra al while

Termina Uno

Llama a Dos

20 de mayo

Candela Rovauk

$m=0 \rightarrow$ entra al if
 se levanta la excepción error!
 se termina el proceso que la levantó (Dos)
 Se ejecuta el manejador asociado al llamado de Dos
 Relanza la excepción
 se termina el proceso que la levantó (Main)
 Termina Main con error.

- b) El manejador asociado al llamado de Uno utiliza la variable x del procedure Main. Para hacer que use la variable de Uno se debería pasar por parámetro al manejador la variable x de Uno:

```

Procedure Main
  error1:exception;
  x,y:integer;
  Procedure Uno () signals error1;
    x:integer;
    begin
      x:=2, ...
      while y < x do
        if y=0 then signal error1(x);
        end if; exception
        when error1 → y:=y+1, x:=x+2, resignal, end;
        Dos();
        y:=y+1;
      end; exception
      when error1 → y:=x+3, x:=x+3, resignal, end;
    end;
  Procedure Dos () signals error1;
    m:integer;
    begin
      ...
      if m=0 then signal error1;
    end;
  begin // Main
    x:=1, y:=0;
    Uno(); exception
    when error1(x:integer) → x:=x+1; y:=y+1; end;
    ...
    Dos(); exception
    when error1 → resignal, end;
  end; // Main
  
```

Nota: No importa que Uno haya terminado (desalocado de memoria) a causa de la excepción. La variable x sigue en memoria.

Ejercicio 14

- En (7) se levanta e y se maneja en 4, luego se relanza la excepción que es manejada en (1) porque e pertenece a Main.
- En (8) se levanta $e2$ y se maneja en (2), y el programa termina.