Lenguajes de Programación

Genericidad y Modulos

Ma. Laura Cobo

Universidad Nacional del Sur
Departamento de Ciencias e Ingeniería de la Computación
2018

Unidades: estructuras de control

Las estructuras de control son:

- Jerárquicas
- Simétricas
- Excepciones y eventos
- Planificadas
- Paralelas

Analicemos cada una de ellas

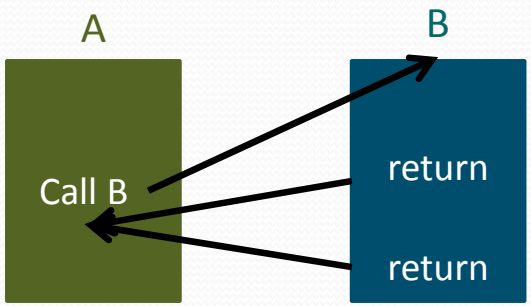
2

Unidades jerárquicas

- **Invocación** a la unidad subordinada a partir de su nombre
- **Retorno** a la unidad llamadora referenciada implícitamente

Abstracción procedural

Abstracción funcional



3

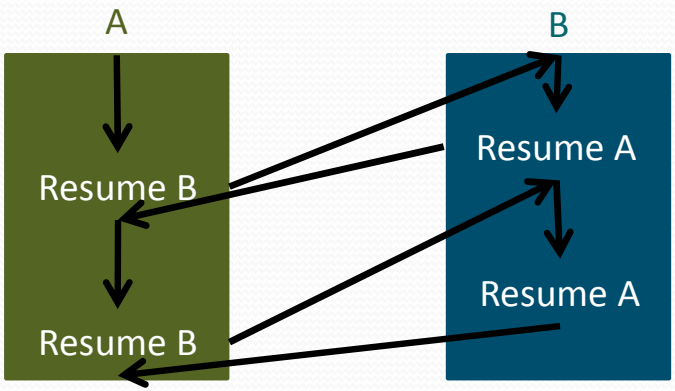
Unidades jerárquicas: ejemplo

```
proc A
label 10
        proc B
                goto 10
        return
        end B
                call B
                ....
10: .....
end A
```

4

Unidades simétricas

- Esquema de control mutuo
- Ejecución entrelazada
- Comportamiento sensible a la historia

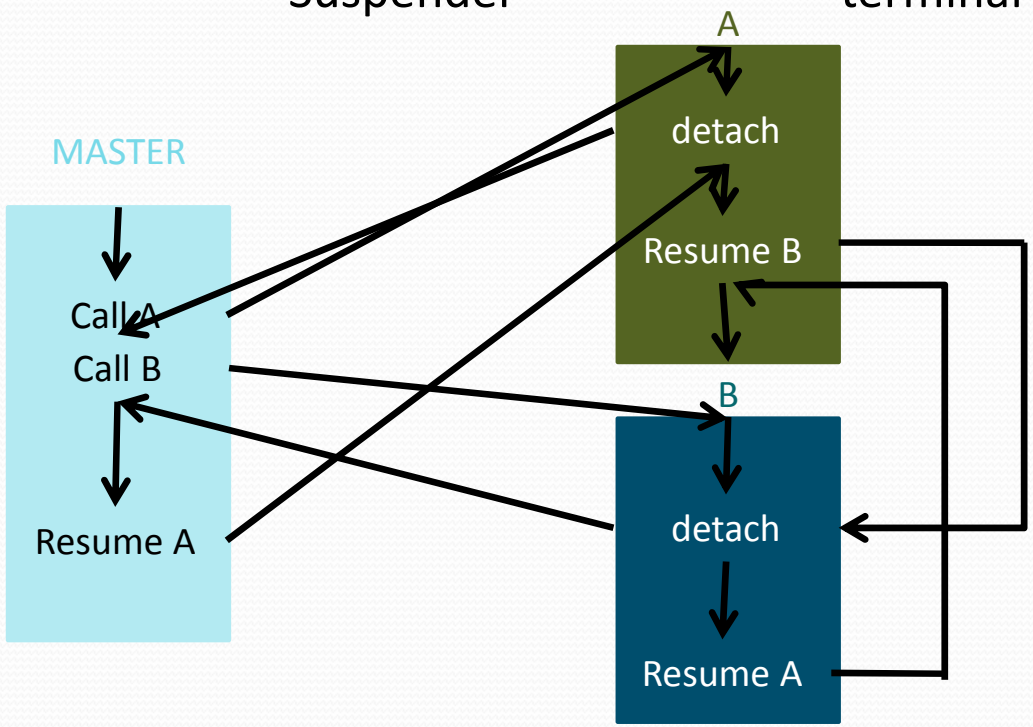


5

Unidades simétricas: ejemplos

Comportamiento:

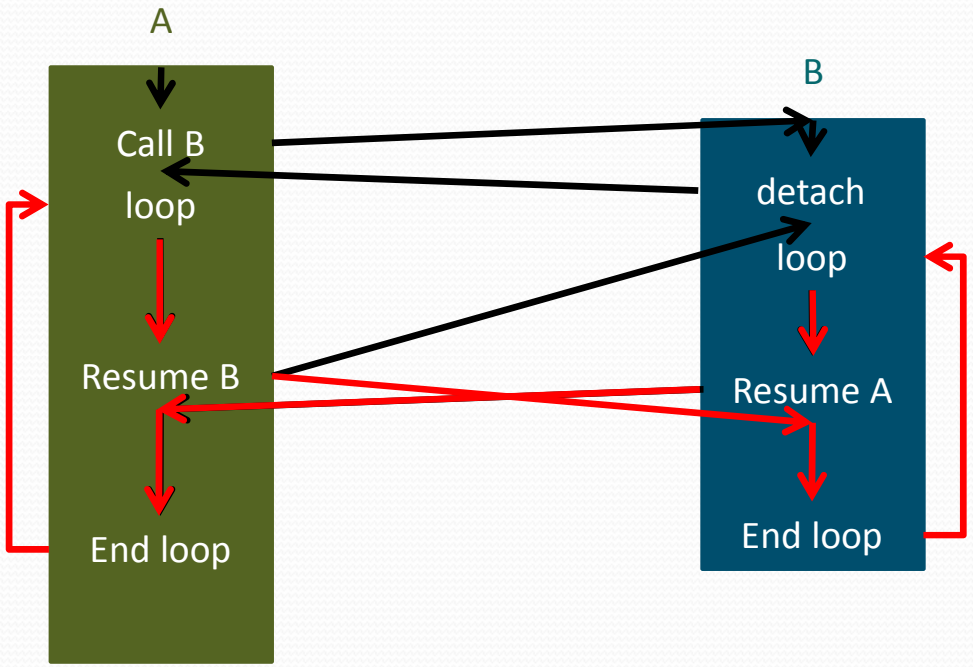
- Invocar
- Suspender
- Reanudar
- terminar



6

Unidades simétricas: ejemplos

Control simétrico en ciclos:



7

Unidades simétricas: ejemplos

```
ref(player) next array p(1:4);
integer i;
integer array cards(1:3)
begin
for i:=1 until 4 do
        begin
            repartir(cards);
            p(i):= new player(i,cards)
        end

resume p(1);
print winner
end
```

```
begin
boolean gameover;
integer winner;
class player(n,hand);
integer n;
integer array hand(1:13);
begin
… detach
while not gameover do
        begin
            move;
            if gameover then winner:= n
                                else resume(next)
        end
end
```

8

Excepciones

Las excepciones son eventos inusuales que deberían controlarse sin oscurecer la legibilidad

Un software que provee excepciones goza de la propiedad de confiabilidad y se dice tolerante a fallas

Las excepciones pueden ser:

- Detectadas por el hardware o por el software
- Situaciones de error o no previstas
- Controladas a través de mecanismos generales o específicos

9

Excepciones: ejemplos

Al intentar copiar el contenido de un archivo en un arreglo:

Operación	Posible excepción
AbrirArchivo	No se pudo abrir
CalcularLongitud	La longitud del archivo excede al máximo del arreglo
AlocarMemoria	No alcanza la memoria disponible
AsignarEnArreglo	Un registro no tiene el formato esperado Índice fuera de rango

10

Excepciones: ejemplos

```
proc Copiar(Arch,Arr,Error)
....
{ openFile(Arch,Error)
  if Error = 0 then {
       calcularLongitud(Arch,N)
       if N > maximo then Error=2
      else {
           alocate(Arreglo,N,Error);
           if Error=0 then { .... }
           }
} }
 end Copiar
```

```
proc Main
....
{ Copiar(Arch,Arr,Error)
  case Error of
       1: write("No se puede abrir el archivo")
       2: write("la cantidad de componentes es mayor que el máximo")
       3: write("no alcanza la memoria disponible")
       otherwise {}
       if error = 0 then
       */ instrucciones sobre el arreglo
end Main
```

En este caso la excepción la maneja la unidad que invocó a la que la produjo

11

Excepciones: ejemplos

```
proc Copiar(Arch,Arr,Error)
….
{ openFile(Arch,Error)
   if Error = 1 then write("No se puede abrir el archivo");
   else { calcularLongitud(Arch,N)
          if N > maximo
           then {Error=2;
             write("la cantidad de componentes es mayor que el máximo")}
         else {alocate(Arreglo,N,Error);
               if Error=3 then { …. }
              }
} }
end Copiar
```

```
proc Main
….
{ Copiar(Arch,Arr,Error)
        if error = 0 then
        */ instrucciones sobre el arreglo
  }
end Main
```

En este caso la excepción la maneja la unidad que la produjo

12

Excepciones: ejemplos

```
proc Copiar(Arch,Arr,label1,label2,label3)
var error
{ openFile(Arch,Error)
  if Error then goto label1
  else { calcularLongitud(Arch,N)
        if N > maximo
         then goto label2
         else {alocate(Arreglo,N,Error);
            if Error then goto label3
                     else {.....
            }
} }
end Copiar
```

```
proc Main
....
{ Copiar(Arch,Arr,1,2,3)
        */ instrucciones sobre el arreglo
1: write("No se puede abrir el archivo")
2:  write("la cantidad de componentes es
      mayor que el máximo")
3: write("no alcanza la memoria disponible")

  }
end Main
```

En este caso la excepción la maneja el bloque etiquetado fuera de la unidad

13

Excepciones: ejemplos

```
proc Copiar(Arch,Arr,cod1,cod2,cod3)
var error
{ openFile(Arch,Error)
  if Error then cod1
  else { calcularLongitud(Arch,N)
       if N > maximo
        then cod2
        else {alocate(Arreglo,N,Error);
            if Error then cod3
                     else {.....
            }
} }
end Copiar
```

```
proc Main
....
{ Copiar(Arch,Arr,ErrOpen,ErrLong,ErrAllocate)
        */ instrucciones sobre el arreglo
}
end Main
```

```
proc ErrOpen
....
end ErrOpen
```

En este caso la excepción la maneja una unidad recibida como parámetro

14

Excepciones: mecanismos no específicos

La situación la detecta la unidad y puede ser manejada:

- **Por la misma unidad**
  - ✓ Decidiendo que acciones ejecutar
  - ✓ Invocando a procedimientos que recibió como parámetros
- **Por la unidad llamadora**
  - ✓ A través de un valor que le indica qué situación se produjo
  - ✓ A través de la transferencia de control

15

Excepciones: mecanismos no específicos

La situación la detecta la unidad y puede ser manejada:

- **Por la misma unidad:** Decidiendo que acciones ejecutar

```
proc A
if Cond1 then Manejador1
        else if Cond2 then Manejador2
                else S
end A
```

Excepciones: mecanismos no específicos

La situación la detecta la unidad y puede ser manejada:

- **Por la misma unidad:** Invocando a procedimientos que recibió como parámetros

```
proc A (proc Man1; proc Man2)
if Cond1 then Man1
        else if Cond2 then Man2
                else S
end A

Call A(Manejador1,Manejador2)
```

17

Excepciones: mecanismos no específicos

La situación la detecta la unidad y puede ser manejada:

- **Por la unidad llamadora:** A través de un valor que le indica qué situación se produjo

```
proc A (var Err:int)
if Cond1 then Err:=1
          else if Cond2 then Err:=2
                        else S
end A

call A (Error);
if Error=1 then  call Manejador1
           else if Error=2 then call Manejador2
                           else …..
```

18

Excepciones: mecanismos no específicos

La situación la detecta la unidad y puede ser manejada:

- **Por la unidad llamadora:** A través de la transferencia de control

```
Label 1,2;
proc A
 if Cond1 then goto 1
          else if Cond2 then goto 2
                        else S
 end A

 call A; ….
1: call Manejador1;
2: call Manejador2;
```

19

Excepciones: mecanismos específicos

Aspectos de diseño:

- Predefinidas o definidas por el programador
- Habilitar o deshabilitar
- Declaración y alcance
- Señalización
- Captura y ligadura al manejador
- Propagación
- Continuación
- Comunicación

Excepciones: mecanismos específicos en LP

Fortran:

```
Read(Unidades=5; Fmt=1000, ERR=100, END=999) Weight


     Read(5,100,Err=200,End=300) A
     Format(E10.2)
200  .....
300  .....
```

La cláusula ERR especifica que el control se transferirá a la etiqueta colocada a continuación si un error ocurre en la operación de lectura. La cláusula END tiene un comportamiento análogo solo que se efectúa en caso de que la operación de lectura alcance el end of file.

Por lo tanto, es claro que Fortran utiliza simples saltos como mecanismo para el manejo de excepciones

21

Excepciones: ejemplo

Ada:

El lenguaje provee cinco categorías de excepciones predefinidas:

- Constaint_Error: falla en el chequeo de alguna restricción en ejecución, por ejemplo el índice de un arreglo.
- Numeric_Error: similar al anterior pero con números, ejemplo: división por cero
- Program_Error: falla en el chequeo de alguna regla del lenguaje. Ejemplo: se espera que una función culmine su ejecución y retorne un valor al llamador. Si esto no sucede se levanta una excepción
- Storage_Error: falla en el chequeo de memoria disponible. Ejemplo: puede suceder a la hora de realizar un new
- Tasking_Error: falla en tiempo de ejecución en una tarea de sistema (programación concurrente)

El manejador de cada excepción puede ser redefinido

22

Excepciones: ejemplo

Ada:

Permite la definición de nuevas excepciones a través de declaraciones

```
Proc Leer
        bad_data_value: exception;
Begin
Read(x);
If X<0 then raise bad_data_value
end
```

Estas excepciones son tratadas igual que las predefinidas pero deben ser levantadas en forma explícita

La declaración determina el alcance.

El ***raise*** la señalización

23

Excepciones: ejemplo

Ada: Los manejadores de excepciones pueden ser incluidos en bloques o en el cuerpo de unidades, paquetes o tareas.

Sin importar dónde aparezcan, los manejadores están agrupados en la cláusula *exception*. La cuál debe estar ubicada al final del bloque o unidad

```
Begin
-- cuerpo o cuerpo de la unidad –
exception
        when excepción1 =>
                --primer manejador
        when excepción2 =>
                --segundo manejador
                -- otros manejadores
End;
```

La clausula ***exception*** resuelve la ligadura del manejador

24

Excepciones: ejemplo

Ada:

Si la unidad que levanta la excepción tiene un manejador para ella, el control se transfiere al manejador.

Las acciones que se deberían haber ejecutado luego de la sentencia que provocó la excepción son salteadas.

Se ejecuta el manejador y el programa continua la ejecución como si la unidad que levantó la excepción hubiese terminado normalmente. Retoma la ejecución desde el **end** del bloque o unidad.

Si la unidad en ejecución no provee un manejador para la excepción, la unidad termina y el excepción es propagada al llamador. La propagación significa que la excepción es relanzada en otro contexto

25

Eventos

Por conveniencia, en algunos casos, los programas son estructurados como *sistemas* reactivos, sistemas donde determinados eventos que ocurren en el entorno determinan el orden de ejecución

En la programación basada en eventos el control de ejecución se basa en acciones producidas por el usuario o por el entorno

Un manejador de eventos es un código que se ejecuta en respuesta a un evento.

Ejemplos donde se requiere programación orientada a eventos:

- Sistemas reactivos
- Interfaces gráficas
- Sistemas operativos

26

Eventos

En una interfaz gráfica los eventos se producen a partir de la interacción del usuario con las **componentes reactivas** de la interfaz

Un **evento** es una notificación de que el usuario ha realizado un acción que requiere respuesta por parte del sistema

Un **manejador de eventos** es el segmento de código que ejecuta en respuesta al evento

```
on event
when condition
    do action
```

27

Eventos

Una interfaz gráfica es una **colección de componentes** con una **representación gráfica** y capacidad para **percibir eventos** generados por el usuario.

Las componentes son las partes individuales a partir de las cuales se conforma una interfaz gráfica. Por ejemplo, el botón para cerrar una ventana, la barra de desplazamiento de una ventana o la ventana misma

Un usuario realiza una **acción** que provoca un evento ante el cual una componente tiene una **reacción**.

Una componente está asociada a un **objeto gráfico** que puede interactuar con el usuario

28

Eventos

La **implementación** de una interfaz gráfica consiste en:

- Crear un objeto gráfico para cada componente de la GUI e insertarlo en otras componentes contenedoras
- Definir el comportamiento de las componentes reactivas en respuesta a las acciones del usuario

El desarrollo de una interfaz gráfica está fuertemente ligado a los conceptos de evento, herencia, polimorfismo y ligadura dinámica.

29

No determinismo y backtracking

Otra manera de abordar la solución a un problema es describiéndolo como un descomposición en sub-problemas.

De esta manera la solución a un problema se traduce en una combinación (AND) y elección (OR) de sub-problemas.

Idealmente la exploración de la solución no debería requerir un determinado orden. Sin embargo es una decisión de diseño del lenguaje determinar esto.

30

No determinismo y backtracking

Si el orden en el cual los sub-problemas se resuelven no es especificado, se dice que el programa es **no determinista**.

En este caso el lenguaje estable que el orden de evaluación, no resulta relevante en la resolución

Lo que resulta vital es que de existir una solución la misma sea encontrada.

También que si no hay solución se indique o retorne la falla.

31

No determinismo y backtracking

La estrategia básica es recorrer los nodos del árbol de búsqueda, evaluando un nodo a la vez.

Elegir luego una submeta a evaluar y repetir el proceso.

1. Búsqueda depth-first
2. Búsqueda breath-first

La primera requiere de un mecanismo que retroceda en el árbol, para permitir la evaluación de otra ramas no exploradas. Este mecanismo se conoce como **backtracking**.

El **backtracking** asegura la completitud del proceso de búsqueda.

32

Unidades planificadas

El concepto de unidad planificada aparece como una relajación de la máxima que establece que la ejecución de toda unidad debe comenzar inmediatamente a continuación de su llamada.

1. Las unidades planificadas pueden ejecutarse antes o después que otras unidades.

   Ejemplo: **call B after A**, en el cual se planifica la ejecución de B para algún momento posterior a la culminación de la ejecución de A

2. Pueden ejecutarse cuando una expresión booleana arbitraria es verdadera.

   Ejemplo: **call B when X=5 and Y*Z>24**, la ejecución toma lugar cuando se garantizan determinadas condiciones

33

Unidades planificadas

3. Pueden ejecutarse sobre la base de una escala temporal simulada

   Ejemplo: **call B at time = 25** o

   call B at time = currentTime+24

4. Pueden ejecutarse de acuerdo a un orden de prioridad

   Ejemplo: **call B with priority 5**, la activación de B tendrá lugar cuando no haya otra unidad con mayor prioridad planificada

Esta característica aparece en lenguajes diseñados para realizar simulación discreta de sistemas, como por ejemplo GPSS, SMSCRIPT y SIMULA

34

Unidades planificadas

Para programas con este tipo de unidades, ya no hay programa principal. Se cuenta con un programa planificador que va determinando que unidades están planificadas para ejecutar, las mantiene en una especie de lista. La misma mantiene el orden en el cual las unidades se activaran o ejecutaran.

35

Unidades paralelas

En este tipo de control de ejecución de unidades se elimina la restricción de ejecutar en forma secuencial, ya que varias unidades pueden ejecutarse simultáneamente.

Las unidades que pueden ejecutarse concurrentemente con otras unidades son llamadas **tareas** o **procesos**

Hasta ahora todas las estructuras de control para unidades están basadas en la existencia de un hilo de ejecución predeterminado que describe la ejecución del programa

36

Unidades paralelas

Un sistema *multiprocesador* tiene varias unidades de procesamiento central o CPU compartiendo memoria.

Un sistema computacional *distribuido* o *paralelo* esta conformado por varias computadoras, cada una con su propia CPU y memoria, conectadas entre si de manera que todas pueden comunicarse

37

Unidades paralelas

Principios de la programación paralela:

1. Definición de variables: declaración de las variables sobre las que se requiere sincronización y sobre las que no
2. Composición paralela: agregado de la composición paralela que ocasiona la ejecución de mas de un hilo
3. Estructura del programa:
   - *Transformacional*: obtener una salida de un grupo de valores de entrada
   - *Reactiva:* el sistema reacciona a múltiples eventos
4. Comunicación:
   - *Memoria compartida*
   - *Mensajes*
5. Sincronización: en algunos sistemas se requiere un ordenamiento en la ejecución de los hilos

38