Principios de Lenguajes de Programación

Control de Secuencias: Expresiones

Facultad de Informática
Universidad Nacional del Comahue

Primer Cuatrimestre

Índice

Unidad 3. Control de Secuencia y Expresión.

Control de Secuencia: implícito y explícito. Control de Expresión. Estructuras de control a nivel sentencias. Evaluación ansiosa y perezosa.

Índice

- Control
- Control de Secuencias
- Expresiones
- Asignaciones

- Bibliografía:
  - Pratt (cap. 6), Sebesta (cap. 7)

Control

- Vimos:
  - Datos y operaciones
- Veremos:
  - Organización de cómo los datos y operaciones son combinados en programas y conj de programas.

Introducción

- Las expresiones son una forma fundamental de **especificar computación** en un lenguaje de programación.
- Además de la forma de las expresiones (sintaxis: BNF), **su significado** (semántica) **es crucial**.
- La semántica, esta gobernada por **su forma de evaluación**
  - Es necesario estar conscientes de los ordenes de evaluación de operandos y operadores
- La esencia de los lenguajes de programación imperativos es dominada por el rol de las **sentencias de asignación**

Control

- El control de la ejecución de los programas (algoritmos) se basa en estructuras de control:
- Aspectos en que podemos dividir ese control:
  - ***Control de Secuencias***: el control del orden de ejecución de las operaciones (primitivas y definidas por el usuario)
  - ***Control de los Datos***: el control de cómo se transmiten datos entre los subprogramas de un programa

Control de Secuencia

- Niveles de las Estructuras de Control (imperativo):
  - Utilizadas en las *expresiones* (***dentro de las sentencias***) tales como reglas de precedencia y paréntesis.
  - Utilizadas entre sentencias ***(instrucciones) o grupos de sentencias***, tales como condicionales o iteraciones
  - Utilizadas entre subprogramas, tales como llamadas a programas, corutinas, etc.

Control de Secuencia

- Niveles de las Estructuras de Control (imperativo):
- Otros paradigmas
  - La programación declarativa tiene un modelo de ejecución que no depende del orden de las sentencias en el programa fuente
    - Lisp y APL no hay sentencias solo expresiones.

Control

- **Control Implícito**
  - Determinado por el orden de las sentencias (o instrucciones) en el programa fuente
  - *Por el modelo de ejecución de la máquina virtual establecida para el lenguaje*
- **Control Explícito**
  - El programador utiliza sentencias específicas para cambiar el orden de ejecución (cambia el control, ej, sentencia IF)

Control en las expresiones

Expresiones

- Son un medio fundamental para **especificar computaciones** en un lenguaje de programación
- Para comprender la evaluación de una expresión, debe familiarse con el **orden de evaluación de los operadores y operandos**

Expresiones

- En su forma más “pura”, las expresiones no implican consideraciones de control: las subexpresiones se evalúan en orden arbitrario, y el orden de ejecución no afecta el resultado
  - La programación declarativa trata de cumplir con esta meta hasta en los programas completos

Expresiones Aritméticas

- La evaluación aritmética fue una de las motivaciones para el desarrollo de los primeros lenguajes de programación
- Las **expresiones** constan de:
  - Operadores
  - Operandos
  - Paréntesis
  - Llamadas a función
- Los **operadores** pueden ser:
  - Unarios: tienen sólo un operando
  - Binarios: tienen dos operandos

Expresiones Aritméticas

La mayoría de los lenguajes imperativos adoptan una notación infija para los operadores, pero hay otras opciones:

- **Prefija**
  - Común *(+(A,B),-(C,A))
  - Polaca (*(+ A B) (- C A))
  - Polaca Cambridge * + A B – C A
- **Posfija**
  - Sufija ((A,B)+,(C,A)-)*
  - Inversa AB+CA-*
- **Infija** (A + B) * (C – A)

Expresiones Aritméticas: consideraciones de diseño

- Algunas consideraciones para el diseño:
  - Reglas de precedencia para los operadores
  - Reglas de asociatividad para los operadores

  ] control
  - Orden de evaluación de los operandos
  - Generación de efectos colaterales en la evaluación de los operandos
  - Se permite sobrecarga de los operadores
  - Expresiones con combinación de tipos

Expresiones Aritméticas: Operadores

- Número de operandos
  - Operador unario: único operando
  - Operador binario: dos operandos
  - Operador ternario: tres operandos
  - ...

Copyright © 2009 Addison-Wesley. All rights reserved.

Reglas de precedencia - Asociatividad

- Las ***reglas de precedencia***
  - definen el orden en el cual serán evaluados los operadores adyacentes de **diferente** nivel de precedencia son evaluados
- Ejemplo de niveles de precedencia:
  - Paréntesis
  - Operadores unarios
  - Potencia
  - Producto / División
  - Suma / Resta
- Define como se resuelve el orden de evaluación frente a operadores del mismo nivel de precedencia.
- Regla típica: evaluación de **izquierda a derecha**

Las reglas de precedencia y asociatividad pueden modificarse con el uso de paréntesis

Copyright © 2009 Addison-Wesley. All rights reserved.

Expresiones Aritméticas: Operadores

- Las ***reglas de asociatividad*** de los operadores para la evaluación de una expresión definen el orden en el cual serán evaluados los operadores adyacentes del **mismo** nivel de precedencia
- Ejemplo de reglas de asociatividad
  - De izquierda a derecha (en muchos operadores)
    - 2 + 3 + 5 + 6 → ((2 + 3) + 5) + 6
    - a – b – c → (a – b) – c ≠ a – (b – c) = a – b + c
  - De derecha a izquierda en la potencia (**)
    - 2 ** 3 ** 4 → 2 ** (3 ** 4)

Copyright © 2009 Addison-Wesley. All rights reserved.

Expresiones Aritméticas: Operadores

- **Paréntesis**
  - Modifica las reglas “implícitas” de precedencia y asociatividad para modificar evaluación de una expresión
  - (2 + 3) * 4
- Algunos lenguajes actúan en forma especial:
  - **APL, SmallTalk** : todos los operadores tienen igual precedencia
  - **Pascal**
    - A = B | C = D es ≠ que A = (B | C) = D que da error.

Copyright © 2009 Addison-Wesley. All rights reserved.

Expresiones Aritméticas: Operadores

- Algunos lenguajes actúan en forma especial:
  - **Ruby**:
    - **Todos los operadores** aritméticos, relacionales y de asignación (así como el indexado de arreglos, shifts y operadores lógicos nivel bit) se **implementan como métodos**
    - Todos los operadores se pueden “sobreescribir” por los programas de aplicación (reordenar)
  - C
    - Tiene 17 niveles de precedencia (Pratt)

Copyright © 2009 Addison-Wesley. All rights reserved.

Expresiones Aritméticas: Operadores

Precedence	Operators	Operator names
17	tokens, a[k],	f()Literals, subscripting, function call
	.,->	Selection
16	++, --	Postfix increment/decrement
15*	++, --	Prefix inc/dec
	~, -, sizeof	Unary operators, storage
	!,&,*	Logical negation, indirection
14	typename	Casts
13	*, /, %	Multiplicative operators
12	+,-	Additive operators
11	<<, >>	Shift
10	<,>,<=, >=	Relational
9	==, !=	Equality
8	&	Bitwise and
7	^	Bitwise xor
6	|	Bitwise or
5	&&	Logical and
4	||	Logical or
3*	?:	Conditional
2*	=, +=, -=, *=,	Assignment
	/=, %=, <<=, >>=, &=, ^=, |=	
1*	,	Sequential evaluation

** reglas de asociatividad a derecha*

Expresiones Aritméticas: expresiones condicionales

- Lenguajes basados en C (ej C, C++, etc.)
- Ejemplo:
  - `prom = (cont == 0) ? 0 : suma/cont`
- Se evalúa como

```
if (cont == 0)
   prom = 0
else
   prom = suma / cont
```

Copyright © 2009 Addison-Wesley. All rights reserved.

Expresiones Aritméticas: orden de evaluación de operadores

- Evaluación ansiosa
  - Evaluar (o generar el código para evaluar) en primer lugar a los operandos, y luego aplicar la operación
- Evaluación perezosa
  - Nunca evaluar los operandos hasta que no se aplique la operación
    - Pasar los operandos sin evaluar y dejar que la operación decida si es o no necesario evaluar algún operando

Copyright © 2009 Addison-Wesley. All rights reserved.

Expresiones Aritméticas: orden de evaluación de operadores

- **Orden usual de operación ansiosa**
  - Variables (traerlas de memoria)
  - Constantes
    - Traerlas de memoria
    - Incorporadas a la instrucción en lenguaje de bajo nivel
  - Expresiones en paréntesis (evaluar los operandos y luego sus operadores)
  - Un operando puede ser una función

Copyright © 2009 Addison-Wesley. All rights reserved.

Expresiones Aritméticas: problemas

- Problemas que pueden presentarse al momento de generar la traducción:
  - Reglas de evaluación uniforme
  - Efectos Colaterales
  - Condiciones de Error
  - Evaluación de Circuito Corto

Expresiones Aritméticas: problemas

- Reglas de evaluación uniforme
  - Ejemplo:
    - Z + (X == 0 ? Y : Y/X)
  - Distinto comportamiento para evaluación ansiosa y perezosa
    - Ansiosa: división por cero
    - Perezosa: no genera error
      - Muy difícil de implementar en algunos lenguajes
- Comparar paso de parámetros (por valor y por referencia)

Expresiones Aritméticas: problemas

- **Efectos colaterales de una función**:
  - Una función cambia el valor de un parámetro o una variable no local
- **Problemas:**
  - Cuando una expresión que referencia a una función que altera un operando de la expresión (por ejemplo una variable):

Expresiones Aritméticas: problemas

```
a = 10;
  /* fun puede cambiar el valor de a */
b = a * fun(&a) + a;
```

Si fun cambia el valor de *a* el orden de evaluación es crítico:

- Sea ***a*** = 1, ***fun*** devuelve 3 y cambia ***a*** a 2
- Posibles evaluaciones
  - Izquierda a derecha: 5
  - Derecha a izquierda: 7
  - Evaluar ***a*** una vez: 4
  - Llamar a fun antes de evaluar ***a:*** 8

Expresiones Aritméticas: problemas

- Si fun cambia el valor de *a* el orden de evaluación es crítico:
  - Sea ***a*** = 1, ***fun*** devuelve 3 y cambia ***a*** a 3
  - Posibles evaluaciones
    - Izquierda a derecha: 5 = (1 x 3) + 2
    - Derecha a izquierda: 7 = 1 + (3 x 2)
    - Evaluar ***a*** una vez: 4 = 1 x 3 + 1
    - Llamar a fun antes de evaluar ***a:*** 8 = (3 x 2) + 2

Expresiones Aritméticas: problemas

- **Efectos colaterales:** soluciones posibles:
  - El diseñador del lenguaje de programación **deshabilita los efectos colaterales** de las funciones
    - No hay parámetros de entrada-salida en las funciones
    - No se pueden realizar referencias a OD no locales
    - *Ventaja*: funciona!!
    - *Desventaja*: inflexible (parámetros de sólo entrada) y falta de acceso a los OD no locales
  - Escribir la definición del lenguaje de manera que **fije la evaluación de los operandos**
    - *Desventaja:* limita las posibilidades de optimizar el código
  - **Ignorar** el problema (suerte !!!)

Expresiones Aritméticas: problemas

- **Condiciones de error:** clase especial de efecto colateral operaciones que pueden fallar y generan condición de error.
  - Dependen de las implementaciones
  - Por ejemplo, sobre operaciones primitivas (overflow, división por cero, etc.)
  - Las soluciones pueden ser ad-hoc (implementación particular)

Expresiones Aritméticas: problemas

- **Evaluación de circuito corto:**
  - El resultado de la expresión se obtiene sin evaluar todos los operandos/operadores:
    - Ejemplo: (13*a) * (b/13–1), con a = 0

Expresiones Aritméticas: problemas

- **Evaluación de circuito corto:**
  - De expresiones booleanas:

```
if (( A == 0 ) || (B/A > C)) { … }
```

```
while (( I <= UB ) && (v[I] > C)) {…}
```

  - El segundo operando puede generar condiciones de error
  - Algunas soluciones:
    - C, C++, y Java: usan circuito corto para operadores habituales booleanos (**`&&`** and **`||`**)
    - Ada: el programador puede establecer si es corto o no (cláusulas **`and then`** y **`or else`**)

```
If (A = 0) or else ( B/A > C) then ...
```

Operadores sobrecargados

- Uso de operadores para más de un propósito (ej. + para enteros y reales)
- Características
  - Pérdida detección de errores por el compilador
  - Genera reducción de legibilidad
  - Puede ser evitada con nuevos símbolos

Conversión de Tipo de Dato

- Conversión
  - explícita (cast)
  - implícita (coerción)
- Información
  - Con pérdida (narrowing)
  - Sin pérdida (widening)

Expresiones mixtas

- Una expresión en la cual se presentan operadores de diferentes tipo de dato
- Problemas
  - **Coerción**: resultados no deseados
    - Disminuyen la habilidad de detección de errores
    - Uso habitual en la mayoría de lenguajes sobre los tipos numéricos

Expresiones Relacionales

- **ER:**
  - Utilizan operadores relacionales y operandos de varios tipos
  - Evalúan en alguna representación booleana
  - Notación: las operaciones son similares pero varía la simbología entre lenguajes (!=, /=, ~=, .NE., <>, #)
- **Operadores especiales:**
  - JavaScript y PHP tienen === y !==
    - Similares a tradicionales
    - No permiten coerción entre sus operandos

Expresiones Booleanas

- Resultan de operandos booleanos (que pueden surgir de expresiones relacionales)
- Resultado es booleano
- Ejemplos de Operadores:

FORTRAN 77	***FORTRAN 90***	***C***	***Ada***
.AND.	*and*	*&&*	*and*
.OR.	*or*	*||*	*or*
.NOT.	*Not*	*!*	*not*
			xor

Expresiones Aritméticas y Booleanas

- Algunos lenguajes no tienen tipo boolean:
  - Utiliza el tipo ***int*** con 0 para *falso* y <> de 0 para *verdadero*
  - Algunos problemas
    - a < b < c
    - Es válida, pero no evalúa como aparenta:
    - a < b < c → R < c , con R 0 o 1, según evaluación de expresión relacional

Sentencias de Asignación

- Sintaxis habitual:

  <OD_destino> <op_asignación> <expresion>

- Operador de Asignación
  - = FORTRAN, BASIC, the C-based languages
  - := ALGOLs, Pascal, Ada
  - = diferente de == (pe, C y familias)
- Asignación con Destino condicionado (pe, lenguaje Perl)

  ```
  ($flag ? $total : $subtotal) = 0
  ```

  - Similar a

  ```
  if ($flag){
       $total = 0
  } else {
       $subtotal = 0
  }
  ```

Sentencias de Asignación: compuesta

- Asignaciones compuestas con operaciones habituales en formato comprimido
  - Incorpora ALGOL, adoptado por C

    a = a + b → a += b
  - El operador *op=* con op alguno de

    `+ - * / % << >> & ^ |`
- Ejemplos:

```
<<= :   result <<= num     result = result << num
>>= :   form >>= 1         form = form >> 1
&=  :   mask &= 2          mask = mask & 2
^=  :   test ^= pre_test     test = test ^ pre_test
|=  :   flag |= ON         flag = flag | ON
```

Sentencias de Asignación: operadores unarios

- Presentes en lenguajes tipo C, combinan incremento y decremento con operaciones de asignación
- Ejemplos
  - sum = ++count
  - sum = count++
  - count++
  - -(count++)

Asignación como expresión

- En C, C++, y Java, la sentencia de asignación genera un resultado que puede utilizarse como operando
- Ejemplo

```
while ((ch = getchar())!= EOF){…}
```

Lista de Asignación

- Algunos lenguajes como Perl y Ruby

```
($first, $second, $third) = (20, 30, 40);
```

Conclusiones

- Se define el control y distintos niveles de control en los Lenguajes de programación
- Se analiza el Control de Secuencias en sus 3 niveles y tipos (implícito y explícito)
- Control en una Sentencia:
  - Expresiones
    - Aritméticas, Booleanas y Mixtas
    - Problemas y Soluciones:
      - Reglas de Evaluación Uniforme
      - Efectos Colaterales
      - Condiciones de Error
      - Evaluación de Circuito Corto
  - Sentencias de Asignación:
    - Compuestas y Simples, Expresiones y Listas.