IC-5701
Compiladores e Intérpretes

Profesor:
Ing. Allan Rodríguez Dávila, MGP

Tipos y Declaraciones

Tipos y Declaraciones

- La comprobación de tipos utiliza reglas lógicas. Asegura que el tipo de operandos coincida con el tipo del operador.
- Con el tipo de un nombre se determina el almacenamiento necesario en tiempo de ejecución, conversiones explícitas, calcular la dirección de un arreglo.

Expresiones de tipos

- Un tipo compuesto es denotado por una expresión de tipo
- Una expresión de tipo es
  - Un tipo básico
  - La aplicación de un constructor de tipo a otras expresiones de tipo

Expresiones de tipos

- Tipos atómicos
  - floats, void
- Nombres
  - Literales, lógicas, aritméticas
- Productos
  - Rpducto cartesiano, tuplas, record
- Arreglos
  - `int foo[128]; // array(integer, 128)`

TEC

Equivalencia de tipos

- ¿Cuándo son equivalentes dos expresiones de tipos?
  - Son el mismo tipo básico
  - Se forman mediante la aplicación del mismo constructor de tipos equivalentes en estructura
  - Uno es el nombre de un tipo que denota al otro

TEC

Declaraciones

$D \rightarrow T$ **id** ; $D \mid \varepsilon$

$T \rightarrow B\ C \mid$ **record** { D }

$B \rightarrow$ **int** | **float**

$C \rightarrow \varepsilon \mid$ [**num**] C

Almacenamiento nombres locales

- En compilación se calcula el espacio para tipos estáticos
  - Dirección relativa en la pila
- En ejecución para tipos dinámicos
- La anchura de un tipo es el número de unidades de almacenamiento necesario para los objetos de ese tipo

TEC

Almacenamiento nombres locales

$T \rightarrow B$ C	{ *t = B.tipo; a = B.anchura;* }
$B \rightarrow$ **int**	{ *B.tipo=integer; B.anchura=4;* }
$B \rightarrow$ **float**	{ *B.tipofloat; B.anchura=8;* }
$C \rightarrow \varepsilon$	{ *C.tipo = t; C.anchura = a;* }

TEC

Secuencias de las declaraciones

- Se maneja la variable *desplazamiento* para llevar el registro de las posiciones.

```
P →              { desplazamiento = 0;}
    D
D → T id;        { superior.put(id.lexeme,
                   T.tipo, desplazamiento);
                   desplazamiento += T.tamaño;}
    D1
D → ε
```

Campos en registros

- Utilizan una tabla de símbolos para cada tipo de registro
- El tope de la pila apunta a la nueva tabla de símbolos. Se reinicia *desplazamiento*

Traducción de expresiones

TEC

Operaciones dentro de expresiones

- Se debe generar recursivamente código para variables temporales según se deriva la expresión.

- Se administra la dirección en la tabla de símbolos para para variable
  - Nombre, constante, o valor temporal.

Operaciones dentro de expresiones

Producción	Reglas semánticas
S → id = E;	$S.codigo = E.codigo \;\|\|\; gen(tope.get(\mathbf{id}.lexema)\ '='\ E.dir)$
E → E_1 + E_2	$E.dir = \mathbf{new}\ Temp()$ $E.codigo = E_1.codigo \;\|\|\; E_2.codigo \;\|\|\; gen(E.dir\ '='\ E_1.dir\ '+'\ E_2.dir)$
E → - E_1	$E.dir = \mathbf{new}\ Temp()$ $E.codigo = E_1.codigo \;\|\|\; gen(E.dir\ '='\ '\mathbf{menos}'\ E_1.dir)$
E → (E_1)	$E.dir = E_1.dir$ $E.codigo = E_1.codigo$
E → id	$E.dir = tope.get(\mathbf{id}.lexema)$ $E.codigo = ''$

TEC

Operaciones dentro de expresiones

```
a = b + (- c);
```

```
t2 = menos c
t1 = b + t2
a = t1
```

- a = b + (- c)
- x = (y + w) + (z + v)
- a = d + f + b - c

TEC

Actividad 4

```
a = b + - c;
```

```
t2 = menos c
t1 = b + t2
a = t1
```

- a = b + (- c)
- x = (y + w) + (z + v)
- a = d + f + b - c

Comprobación de tipos

Comprobación de tipos

- Se debe asignar una expresión de tipos a cada componente del programa fuente.
- Se le aplican reglas lógicas
- Lenguaje fuertemente tipado garantiza que el programa en ejecución no tendrá errores de tipo

TEC

Reglas de comprobación de tipos

- La síntesis de tipos construye el tipo de una expresión a partir de los tipos de sus subexpresiones
- Declaración previa de variables

if *f* tiene el tipo s → t **and** *x* tiene el tipo s,

then la expresión *f(x)* tiene el tipo t

TEC

Reglas de comprobación de tipos

- La inferencia de tipos determina el tipo de una construcción a partir de la forma en que se utiliza.
- Las variables tienen tipos desconocidos

if *f(x)* es una expresión,
then para cierto *α* y β, *f* tiene el tipo
α → β **and** x tiene el tipo *α*

Conversión de tipos

- El lenguaje máquina, o de bajo nivel, utiliza distintas instrucciones para los diferentes tipos.
- Esto puede requerir la conversión de algún operando.

```
t1 = (float) 2
t2 = t1 * 3.14
```

TEC

Conversión de tipos

- El lenguaje máquina, o de bajo nivel, utiliza distintas instrucciones para los diferentes tipos.

```
if (E1.tipo = integer and E2.tipo = integer)
      E.tipo = integer;
else if (E1.tipo = float and E2.tipo = integer)
      …
…
```

TEC

Conversión de tipos

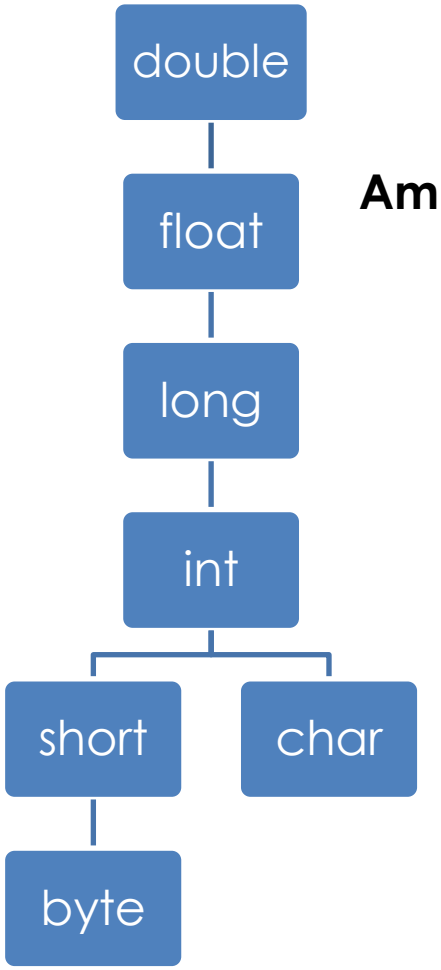
- Con números extensos de tipos sujetos a conversión se debe organizar las acciones semánticas.
- Se aplican reglas de ampliación o reducción.
- Los lenguajes permiten coerción

Conversión de tipos

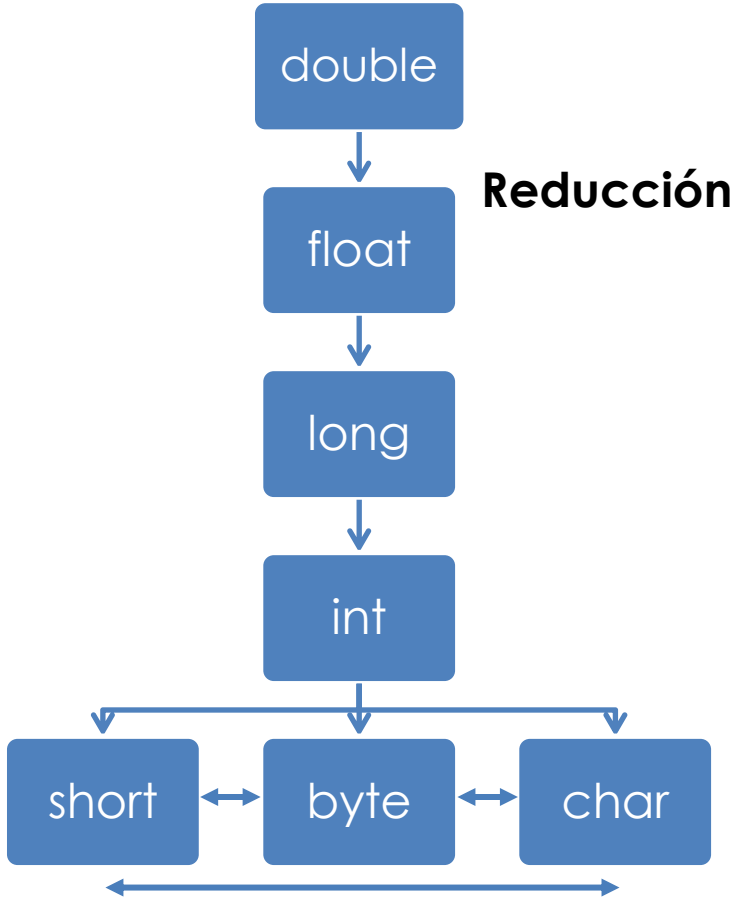
- La ampliación preserva la información.
- La ampliación se proporciona por jerarquía
  - Un tipo menor se amplía a un nivel mayor
- La reducción puede perder información.
  - El lenguaje "habilita el camino"

TEC

Conversión de tipos



Ampliación



Reducción

Conversión de tipos

- La acción semántica de para comprobar $E \rightarrow E_1 + E_2$ utiliza dos funciones.
- $max(t_1,t_2)$ recibe dos tipos y devuelve el máximo. Declara error si no encuentra relación.
- `ampliar(a,t,w)` genera la conversión de tipos, para ampliar una dirección *a* de tipo *t* en una variable de tipo *w*

TEC

Conversión de tipos

```
Dir ampliar(Dir a, Tipo t, Tipo w)
   if ( t = w ) return a;
   else if ( t = integer and w = float ) {
      temp = new Temp();
      gen(temp = (float) a);
      return temp;
   }
   else error;
}
```

TEC

Conversión de tipos

Producción	Reglas semánticas
$E \rightarrow E_1 + E_2$	{ *E.tipo = max(E1.tipo, E2.tipo);* *a1 = ampliar(E1.dir, E1.tipo, E.tipo);* *a2 = ampliar(E2.dir, E2.tipo, E.tipo);* *E1.dir =* **new** *Temp();* *gen(E.dir = a1 + a2);* }

Flujo de Control

Flujo de control

- Las instrucciones if-else y while están enlazadas a las expresiones booleanas.
  - Se utilizan como expresiones condicionales para alterar el flujo de control.
  - Pueden representar true o false para calcular valores lógicos.

B → *B* **||** *B* | *B* **&&** *B* | **!***B* | **(** *B* **)** | *E* **rel** *E* | **true** | **false**

TEC

Código corto circuito

- En el código de corto circuito (o de salto), los operadores booleanos &&, | | y ! se traducen en saltos.

```
if (x < 100 || x > 200 && x != y) x = 0;
```

```
    if x < 100 goto L2
    ifFalse x > 200 goto L1
    ifFalse x != y goto L1
L2: x = 0
L1:
```

Actividad #4

- Muestra (pseudocódigo o algoritmo) de generación de código intermedio de la estructura de control switch (no detallar bloque)
  - Etiquetas de salto a bloque
    - Condición
    - Continua (si o no)

Actividad #4

- Actividad #4:
  - Lectura y resumen de Instrucciones de flujo de control

Bibliografía

- Programming Language Processors in Java: compilers and interpreters. Watt, David, Brown, Deryck. Pearson Education. 2000

- Compilers: principles, techniques and tools (2da. ed.). Aho, Alfred. Pearson Education. 2007


TEC
Tecnológico
de Costa Rica