UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

Escuela Técnica Superior de Ingeniería Informática

Procesadores de Lenguajes

Tema 6

Tipos y generación de código

Javier Vélez Reyes
jvelez@lsi.uned.es

Objetivos del Tema

- Aprender a utilizar los ETDS para
  - Obtener información acerca de los tipos
  - Generar código intermedio
- Estudiar las relaciones entre
  - Los tipos del lenguaje
  - La generación de código intermedio
- Estudiar el uso de la tabla de símbolos
- Estudiar los problemas de ámbito
- Estudiar posibles representaciones intermedias

Javier Vélez Reyes jvelez@lsi.uned.es

Índice General

- Tabla de símbolos
  - Los ámbitos y la tabla de símbolos
- Representaciones intermedias
  - Código de tres direcciones
  - Árboles abstractos
  - Grafos dirigidos acíclicos
  - Código de Máquina virtual
- Sistema de tipos
  - Sobrecarga de operadores
  - Comprobaciones y conversiones de tipos

Javier Vélez Reyes jvelez@lsi.uned.es

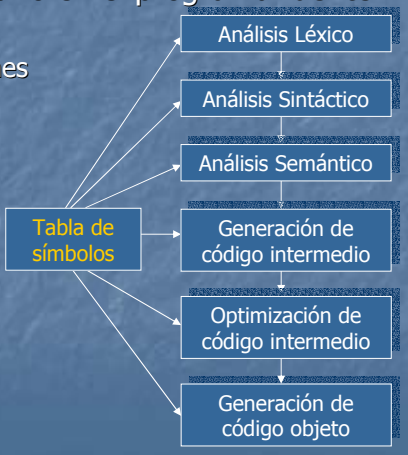
Índice General

- Código intermedio
  - Para expresiones
  - Para sentencias de control de flujo condicional
  - Para sentencias de control de flujo iterativo
- Tipos compuestos
  - La tabla de tipos
  - Ámbitos y la tabla de tipos
  - Código intermedio para tipos compuestos
    - Código intermedio para arrays
    - Código intermedio para registros

Javier Vélez Reyes jvelez@lsi.uned.es

Tabla de símbolos

- Tabla de símbolos
  - Es una tabla que se utiliza para almacenar los nombres definidos por el usuario en el programa fuente
    - Variables
    - Nombres de funciones
    - Nombres de tipos



Javier Vélez Reyes jvelez@lsi.uned.es

Tabla de símbolos

- Se utiliza para comprobar
  - Uso de variables no declaradas
  - Variables declaradas varias veces
  - Incompatibilidad en los tipos de una expresión
  - Etc.
- Para cada entrada hay un registro de información

Tipo de símbolo	Información
Variable	Nombre, tipo, tamaño, dirección
Función	Nombre, tipo, comienzo del código
Tipo	Nombre, tipo, tamaño
Constante	Nombre, tipo, tamaño, valor

Javier Vélez Reyes jvelez@lsi.uned.es

Tabla de símbolos

- Funciones
  - Añadir un símbolo
  - Buscar un símbolo
- Eficiencia en la búsqueda
  - Uso de tablas hash
  - Objetos como registros de información
- Añadir implica buscar
  - Comprobar si existe el símbolo en la tabla
  - Si existe emitir un error
  - Si no existe insertar el símbolo

Javier Vélez Reyes jvelez@lsi.uned.es

Tabla de símbolos

- Ejercicio
  - Dado el programa, ¿Cuál es su tabla de símbolos?

```
int a, b;
float c, d;
char e, f;
```

Tipo de símbolo	Información {TIPO, TAMAÑO, DIR}
a	{ ENTERO, 2, 100 }
b	{ ENTERO, 2, 102 }
c	{ REAL, 4, 104 }
d	{ REAL, 4, 108 }
e	{ CARACTER, 1, 112 }
f	{ CARACTER, 1, 113 }

Javier Vélez Reyes jvelez@lsi.uned.es

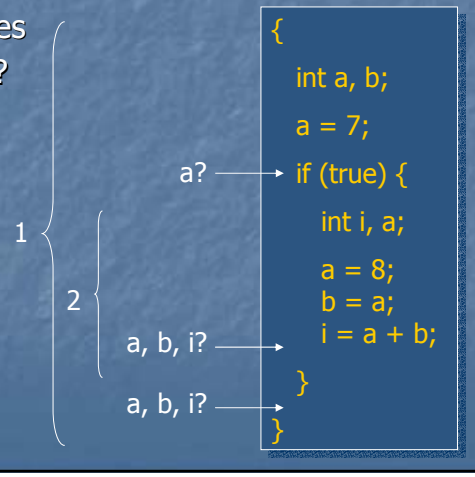
Tabla de símbolos

- La tabla de símbolos y los ETDS
  - Los símbolos deben insertarse cuando aparecen en G
  - Deben ubicarse correctamente las acciones semánticas
  - Puede ser preciso rediseñar la gramática
- En relación con la tabla de símbolos, los errores son
  - Utilizar una variable no declarada
  - Declarar dos veces una variable
  - Desbordamiento en memoria de la tabla de símbolos

Javier Vélez Reyes jvelez@lsi.uned.es

Los ámbitos y la Tabla de símbolos

- Ámbitos anidados
  - Pueden existir distintos ámbitos de declaración anidados
  - Cada ámbito declara variables
  - Pueden solaparse nombres
- ¿Cómo se resuelve b en (2)?
  - Primero busca en (2)
  - Si no existe busca en (1)



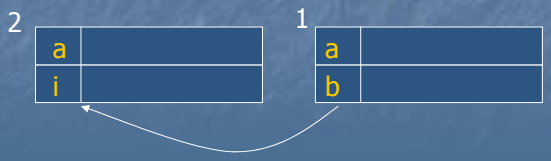
Javier Vélez Reyes jvelez@lsi.uned.es

Los ámbitos y la Tabla de símbolos

- Cuando se sale de un ámbito
  - Deben olvidarse las variables declaradas en él
  - Deben eliminarse las entradas de la tabla de símbolos
- Solución
  - O asociar un campo de ámbito a las entradas de la tabla

1	a	
1	b	
2	a	
2	i	

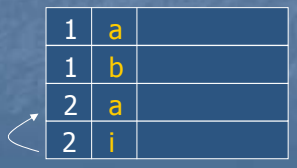
  - O crear una pila de tabla de símbolos



Javier Vélez Reyes jvelez@lsi.uned.es

Los ámbitos y la Tabla de símbolos

- Añadir un nuevo símbolo
  - Al añadir un nuevo símbolo se debe buscar sólo entre los símbolos del mismo ámbito
  - Se comienzan las búsquedas comenzando desde el final de la tabla y terminando cuando se llega al principio de la tabla o cuando el nivel de anidamiento es menor que el actual
- Ejemplo



Javier Vélez Reyes jvelez@lsi.uned.es

Los ámbitos y la Tabla de símbolos

- Buscar un símbolo
  - Al buscar un símbolo que aparece en una instrucción se debe buscar el símbolo desde el final de la tabla hasta el principio, de manera que se encuentre el símbolo del ámbito no cerrado más cercano
- Ejemplo
  - Buscando b…

Javier Vélez Reyes jvelez@lsi.uned.es

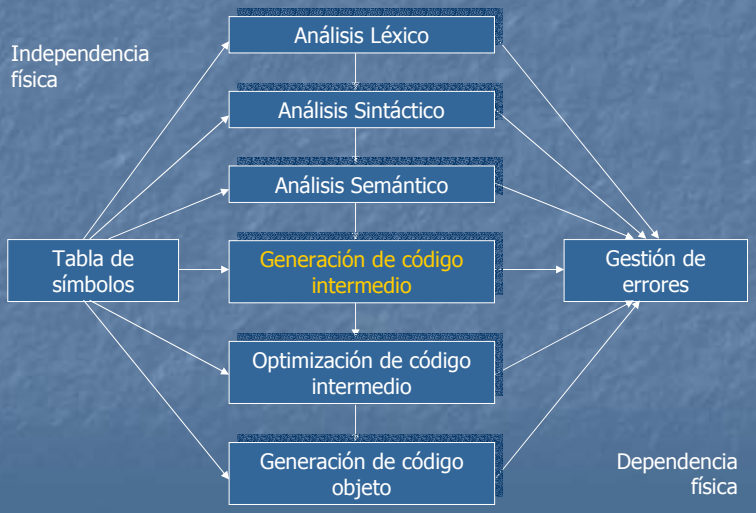
Los ámbitos y la Tabla de símbolos

- Consideraciones sobre variables
  - Si el compilador asigna direcciones de memoria a las variables, éstas deben ser reutilizadas cuando se cierra el ámbito
- Consideraciones sobre funciones
  - El cuerpo de una función es considerado un ámbito
  - Los argumentos son lo primero que se almacena en la tabla de símbolos del ámbito de la función

Javier Vélez Reyes jvelez@lsi.uned.es

Representaciones intermedias

- El código intermedio como frontera…



Javier Vélez Reyes jvelez@lsi.uned.es

Representaciones intermedias

- Código de tres direcciones
  - Representación mediante tercetos

Operador	Resultado / Operando 1	Operando 2

  - Representación mediante cuartetos

Operador	Operando 1	Operando 2	Resultado

- Ejemplo
  - Cuartetos
  - (2+3) * (2+3+5)

ADD	2	3	t1
ADD	2	3	t2
ADD	t2	5	t3
MUL	t1	t3	t4

Javier Vélez Reyes jvelez@lsi.uned.es

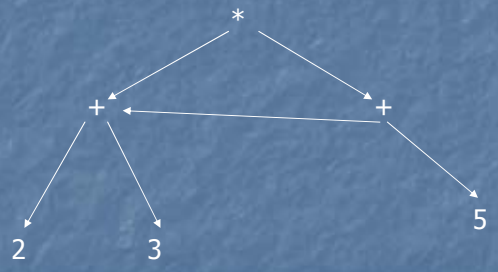
Representaciones Intermedias

- Árboles sintácticos abstractos
  - El código se representa en forma de árbol donde
    - Cada nodo no terminal representa un operador
    - Cada nodo terminal representa un operando
  - Más próximo al lenguaje fuente
- Ejemplo
  - (2+3)*(2+3+5)

Javier Vélez Reyes jvelez@lsi.uned.es

Representaciones Intermedias

- Grafos dirigidos acíclicos
  - Surgen de árboles sintácticos abstractos
  - Se reutilizan expresiones comunes
  - Más óptimo
- Ejemplo
  - (2+3)*(2+3+5)



Javier Vélez Reyes jvelez@lsi.uned.es

Representaciones intermedias

- Código de una máquina virtual
  - Se codifica a una máquina virtual de pila
  - La transformación a una máquina concreta es compleja
  - También se utilizan máquinas virtuales con registros
  - Java & JVM es un ejemplo
- Ejemplo
  - (2+3) * (2+3+5)
  - Máquina virtual a pila

```
lda 2
lda 3
add
lda 2
lda 3
add
lda 5
add
mul
```

Javier Vélez Reyes jvelez@lsi.uned.es

Representaciones intermedias

- Operadores sobrecargados
  - Los lenguajes intermedios no admiten sobrecarga
  - Los lenguajes fuente sí admiten sobrecarga
- Solución
  - Si los dos operandos son del mismo tipo se genera una operación para ese tipo
  - Si los dos operandos son de dos tipos compatibles debe generar código para convertir el tipo
- Ejemplo
  - 2*3+0.5

```
IMUL   2    3    t1
ITOR   t1   –    t2
RADD   t2   0.5  t3
```

Javier Vélez Reyes jvelez@lsi.uned.es

Comprobación y conversiones de tipos

- Durante el análisis semántico el compilador realiza
  - Inferencia de tipos
  - Conversiones de tipos
- La información de tipos es importante
  - Para generar el código intermedio en consonancia
    - Realizar inferencia de tipos
    - Realizar conversiones implícitas
    - Detectar errores de tipo
  - Para dar soporte a la sobrecarga de operadores
- Utilizamos
  - ETDS
  - Tabla de símbolos

Javier Vélez Reyes jvelez@lsi.uned.es

Comprobación y conversiones de tipos

- Ejemplo 8.4 (Página 162, texto base)

I ::= id :=	{ Si Busca (id.lexema) no encontrado ErrorSemantico (…) }
E	{ Si Tipo (id.lexema) == ENTERO y E.Tipo == REAL ErrorSemantico (…) }
E ::= E_1 + T	{ Si E_1.Tipo == ENTERO y T.Tipo == ENTERO E.Tipo = ENTERO Si E1.Tipo == ENTERO y T.Tipo == REAL E.Tipo = REAL Si E1.Tipo == REAL y T.Tipo == ENTERO E.Tipo = REAL Si E1.Tipo == REAL y T.Tipo == REAL E.Tipo = REAL }
E ::= T	{ E.Tipo = T.Tipo }
T ::= nint	{ T.Tipo = ENTERO }
T ::= nreal	{ T.Tipo = REAL }
T ::= id	{ Si Busca (id.lexema) no encontrado ErrorSemantico (…) Si no T.Tipo = Tipo (id.lexema) }

Javier Vélez Reyes jvelez@lsi.uned.es

Código intermedio para expresiones

- Ejemplo
  - Genera (…) — Genera código de 3 direcciones
  - NuevoTemp() — Genera un nuevo temporal
  - X.lugar — Nombre del temporal para X
  - X.código — Código de X

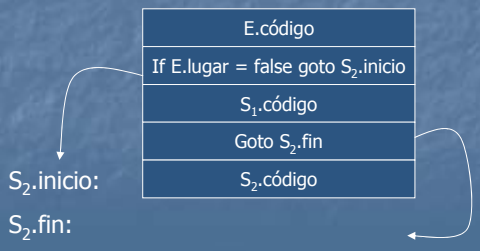
S ::= id := E	{ S.código := E.código || genera(id.lugar ':=' E.lugar) }
E ::= E_1 * E_2	{ E.lugar := nuevoTemp() E.código := E_1.código || E_2.código || genera (E.lugar ':=' E_1.lugar '*' E_2.lugar) }
E ::= E_1 + E_2	{ E.lugar := nuevoTemp() E.código := E_1.código || E_2.código || genera (E.lugar ':=' E1.lugar '+' E2.lugar) }
E ::= - E_1	{ E.lugar := nuevoTemp() E.código := E_1.código || genera (E.lugar ':=' '-' E_1.lugar) }
E ::= (E_1)	{ E.lugar := E_1.lugar E.código := E_1.código || genera (E.lugar ':=' '-' E1.lugar) }
E ::= id	{ E.lugar := id.Lugar E.código := '' }

Javier Vélez Reyes jvelez@lsi.uned.es

Código intermedio para instrucciones

- Control de flujo condicional
  - nuevaEtiqueta — Genera una nueva etiqueta de salto

S ::= if E then S_1 else S_2	{ S_2.inicio := nuevaEtiqueta() S_2.fin := nuevaEtiqueta() S.código = E.código || genera('if' E.lugar '=' 'false' 'goto' S_2.inicio) || S_1.código || genera('goto' S_2.fin) || genera(S_2.inicio ':') || S_2.código || genera(S_2.fin ':') }

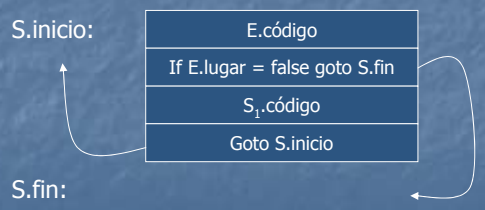


Javier Vélez Reyes jvelez@lsi.uned.es

Código intermedio para instrucciones

- Control de flujo iterativo
  - nuevaEtiqueta Genera una nueva etiqueta de salto

S ::= while E do S_1	{ S.inicio := nuevaEtiqueta() S.fin := nuevaEtiqueta() S.código = genera(S.inicio ':') || E.código || genera('if' E.lugar '=' 'false' 'goto' S.fin) || S_1.código || genera('goto' S.inicio) || genera(S_2.fin ':') }



Javier Vélez Reyes jvelez@lsi.uned.es

Tipos compuestos

- Tipos compuestos
  - Punteros
  - Arrays
  - Registros
  - Funciones
- Notación
  - Puntero (tipo)
  - Array (min..max, tipo)
  - Registro ((nombre: tipo) x … x (nombre: tipo))
  - (tipo x tipo) -> tipo

Javier Vélez Reyes jvelez@lsi.uned.es

Tipos compuestos

- Tabla de tipos
  - La tabla de tipos se utiliza para almacenar toda la información necesaria para la gestión de los tipos compuestos
  - Es una variable global del compilador (como TS)
  - Es necesario definir funciones
    - Añadir Tipos
    - Buscar Tipos
- Proceso

1. Se almacenan todos los tipos básicos
2. Por cada nuevo tipo compuesto que aparece en el programa
   1. Se añade su tipo a la tabla de tipos
   2. Se relaciona con el tipo base que utiliza
   3. Se añaden los símbolos de ese tipo en la tabla de símbolos

Javier Vélez Reyes jvelez@lsi.uned.es

Tipos compuestos

- Ejemplo

```
int a;
float **b;
char c[10];
int d[4][7];
float *e[15];
struct {
        float f;
        int g;
} h;
int funcion (char, float, int);
```

Tabla de Tipos

Número	Tipo	D/T	Tipo Base
0	ENTERO		
1	REAL		
2	CARÁCTER		
3	PUNTERO		1
4	PUNTERO		3
5	ARRAY	10	2
6	ARRAY	7	0
7	ARRAY	4	6
8	PUNTERO		1
9	ARRAY	15	8
10	REGISTRO		TS1
11	P. CARTESIANO	2	1
12	P. CARTESIANO	11	0
13	FUNCION	12	0

Tabla de Símbolos Global

Nombre	Tipo	Dirección
a	0	100
b	4	102
c	5	104
d	7	114
e	9	170
h	10	200
funcion	13	-

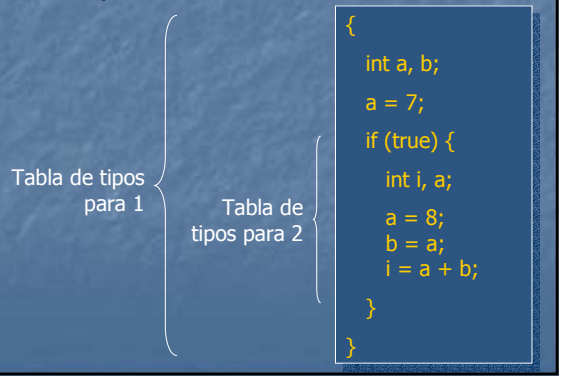
Tabla de Símbolos TS1

Nombre	Tipo	Dirección
f	1	0
g	0	4

Javier Vélez Reyes jvelez@lsi.uned.es

Tipos compuestos

- Ámbitos y la tabla de tipos
  - La gestión es igual a la de la tabla de símbolos
  - Cuando se entra en un ámbito se define una nueva TT
  - Cuando se abandona un ámbito se destruye la TT
  - Una excepción a esto es el procesamiento de funciones



Javier Vélez Reyes jvelez@lsi.uned.es

Tipos compuestos

- Equivalencia de tipos
  - Equivalencia de nombres

    Dos tipos son considerados iguales o equivalentes si tienen exactamente el mismo nombre
  - Equivalencia estructural

    Dos tipos son considerados iguales o equivalentes si tienen la misma estructura
- Ejemplo

```
int *a;
typedef int *punteroAEntero;
punteroAEntero b;
```

¿ Son a y b de tipos equivalentes ?

Javier Vélez Reyes jvelez@lsi.uned.es

Código intermedio para arrays

- Declaraciones de variables simples y arrays

D ::=	D Var	
D ::=	Var	
Var ::=	Tipo	{ L.Th := Tipo.t ; L.tamh := Tipo.tam }
	L ;	
Tipo ::= int		{ Tipo.t := ENTERO ; Tipo.tam := 2 }
Tipo ::= float		{ Tipo.t := REAL ; Tipo.tam := 4 }
Tipo ::= char		{ Tipo.t := CARÁCTER ; Tipo.tam := 1 }
L ::=		{ L_1.th := L.th ; L_1.tamh := L.tamh }
	L_1,	{ V.th := L.th ; V.tamh := L.tamh }
	V	
L ::=		{ V.th := L.th ; V.tamh := L.tamh }
	V	
V ::=	id	{ A.th := V.th ; A.tamh := V.tamh }
	A	{ GuardaSimbolo (id.lexema, A.tipo, A.tam) }
A ::=	[nint]	{ A_1.th := A.th ; A_1.tamh := A.tamh}
	A_1	{ A.tipo := NuevoArray(nint.valex, A_1.tipo); A.tam := A_1.tam * nint.valex }
A ::=	€	{ A.tipo := A.th ; A.tam := A.tamh }

Javier Vélez Reyes jvelez@lsi.uned.es

Código intermedio para arrays

- Acceso a elementos de array
  - Dada la declaración de un array Tipo nombre $[D_1][D_2][D_3]$
  - Acceso a posición del array nombre [i][j][k];
  - La dirección de memoria asociada es

$$\text{DirBase (nombre)} + i \times (D_2 \times D_3 \times \text{Tam (Tipo)}) + j \times (D_3 \times \text{Tam (Tipo)}) + k \times \text{Tam (Tipo)})$$

  - Recursivamente

$$t1 := 0$$
$$t2 := t1 \times D_1 + i$$
$$t3 := t2 \times D_2 + j$$
$$t4 := t3 \times D_3 + k$$
$$t5 := \text{DirBase (nombre)} + t4 \times \text{Tam (Tipo)}$$

Javier Vélez Reyes jvelez@lsi.uned.es

Código intermedio para arrays

- Comprobaciones semánticas
  - Comprobar que la variable indexada es un array
  - Comprobar que la dimensión coincide con la declarada
  - Comprobar que los índices sean de tipos apropiados
  - Comprobar que el índice no excede el rango declarado

Javier Vélez Reyes jvelez@lsi.uned.es

Código intermedio para registros

- Código para registros
  - Si sólo se hay registros y tipos simples no hay problema
  - La dirección de los campos se calcula en compilación
    - La dirección base se suma al desplazamiento del campo
    - Código intermedio igual que un tipo base
- Problema
  - Se puede declarar un array de registros
  - Un campo de un registro puede ser un array
- Por tanto…
  - La dirección base debe calcularse dinámicamente
  - Es necesario sustituir la dirección base por 2 atributos
    - Una para el código
    - Otra para la variable temporal

Javier Vélez Reyes jvelez@lsi.uned.es

Bibliografía

[AJO] AHO, SETHI, ULLMAN: *Compiladores: Principios, técnicas y herramientas*,: Addison-Wesley Iberoamericana, 1990

[GARRIDO] A. Garrido, J. Iñesta, F. Moreno y J. Pérez. 2002. *Diseño de compiladores*. Universidad de Alicante.