Clase 3- Semántica operacional

domingo, 7 de abril de 2024 10:14

ENTIDADES	ATRIBUTOS
Variables	Nombre Tipo área de memoria, etc
Rutinas	nombre, parámetros formales y reales, convención de pasaje de parámetros, etc
Sentencias	acción asociada

Descriptor: donde se almacenan los atributos de las entidades
Si son estáticos se empiezan a llenar en compilación y se llenan en ejecución, si son dinámicos se manejan en ejecución directamente

Los **programas** trabajan con **entidades**

↓

Las **entidades** tienen **atributos**

↓

Estos **atributos** tienen que **establecerse antes** de poder **usar la entidad**

↓

LIGADURA: es el **momento** en el que el **atributo** se **asocia** con un **valor**

Antes de usar la variable tiene que tener un tipo y un valor

Tiene que estar claro cuándo pasa (en ejecución? En compilación?) no tiene que haber ambiguedades

Estabilidad de ligadura: ahora que está ligado, se puede cambiar o queda siempre igual?

Cuándo se hace el binding

- Definición del lenguaje
- Implementación del lenguaje
- Compilación (procesamiento)

↑ ESTATICO

- Ejecución

↓ DINAMICO

Se establece antes de ejecución y no se puede cambiar
Estático tanto el binding como la estabilidad

Ligadura se establece en ejecución y se puede cambiar

Ejemplos:

- En **Definición**
  - Forma de las sentencias
  - Estructura del programa
  - Nombres de los tipos predefinidos
- En **Implementación**
  - Representación de los números y sus operaciones
- En **Compilación**
  - Asignación del tipo a las variables

En lenguaje C

int
Para denominar a los enteros

int
- Representación
- Operaciones que pueden realizarse sobre ellos

int a
- Se liga tipo a la variable

Cuánto espacio ocupa un entero? Qué operaciones se pueden hacer?

Asignarle tipo a variable

- En **Ejecución**
  - Variables con sus valores
  - Variables con su lugar de almacenamiento

int a
- el valor de una variable entera se liga en ejecución y puede cambiarse muchas veces.

Una variable es una abstracción de donde se almacena la variable, en vez de tener que llamar a la dirección donde está la llamamos por el nombre.

VARIABLE	ABSTRACCION	CELDA DE MEMORIA
NOMBRE	ABSTRACCION	DIRECCION
SENTENCIA DE ASIGNACION	ABSTRACCION	MODIFICACION DESTRUCTIVA DE SU VALOR EN LA CELDA

<NOMBRE, ALCANCE, TIPO, L-VALUE, R-VALUE>

- ***Nombre:*** string de caracteres que se usa para referenciar a la variable. **(identificador)**
- ***Alcance:*** es el rango de instrucciones en el que se conoce el nombre
- ***Tipo:*** valores y operaciones
- ***L-value:*** es el lugar de memoria asociado con la variable **(tiempo de vida)**
- ***R-value:*** es el valor codificado almacenado en la ubicación de la variable

L: cuando la variable estrá a la izquierda(necesito darle el valor)
R: cuando está a la derecha (necesito ir a buscarla para operar)

Nombre: longitud máxima, caracteres permitidos, case sensitive?

Alcance: dónde es conocido el nombre de esa variable? Qué rango de instrucciones? Insctrucciones pueden manipular la variable dentro del alcance

Estático: solo viendo la estructura del programa puedo saber su alcance. Según dónde está declarada. No necesito ejecutar el programa para saber

Dinámico: tengo que seguir la ejecución para saber su alcance, no dónde está declarada

Es medio raro, es como que se fija quién llamó al módulo que usa la variable y si ahí está declarada. Sigue cadena de quién llamó a quién

Las reglas de Alcance Estático:

- Son las **más utilizadas** por los LP (C, PASCAL, ADA PYTHON, ETC.)

Las reglas de Alcance dinámico:

- **Menos utilizadas** por los LP
- **Más fáciles de implementar**
- **Poco claras y eficientes**
- En cuanto la programación. **Encontrar una declaración en el flujo de ejecución puede ser duro.** El código se hace **más difícil de leer y seguir,** sobre todo en **grandes programas** con cientos de sentencias es complejo

```
int x;
{
 /*bloque A*/
 int x;
 ....
}
{
 /*bloque B*/
 int x;
 ....
}
{
 /*bloque C*/
 x = ...;
 ...
}
....
```

(a llama a c)

(b llama a c)

Ejecución:

- Con **alcance Dinámico**, si:
  - A C → **x de A**
  - B C → **x de B**
- Con **alcance Estático** en ambos casos hace referencia a **x externa**

Clasificación según alcance:

- **Local:** Son todas la referencias que se han creado dentro del programa o subprograma.
- **No Local:** Son todas las referencias que se utilizan dentro del subprograma pero que no han sido creadas en él.
- **Global:** Son todas las referencias creadas en el programa principal

Declarado dentro de función, rutina, bloque

No global pero tampoco declarado dentro del bloque, declarado dentro del bloque q contiene al bloque en cuestión.

Espacios de nombres

- Definición:
  - Un espacio de nombre es una zona separada donde se pueden declarar y definir objetos, funciones y en general, cualquier identificador de tipo, clase, estructura, etc.; al que se asigna un nombre o identificador propio.
- Utilidad:
  - Ayudan a evitar problemas con identificadores con el mismo nombre en grandes proyectos o cuando se usan bibliotecas externas.

<NOMBRE, ALCANCE, **TIPO**, L-VALUE, R-VALUE>

- **Definición:**
  - **Conjunto** de **valores** posibles
  - **Conjunto** de las **operaciones**
- Antes de que una variable pueda ser referenciada debe **ligársele un tipo**
- **Protege a las variables de operaciones no permitidas**

Reglas de compatibilidad/conversión: en algunos lenguajes se puede combinar tipos, pero tienen que estar declaradas

A

Clases de tipo

- **Predefinidos - por el lenguaje**
  - **Tipos base** definidos en el lenguaje
- **Definidos - por el usuario**
  - **Constructores**, permiten crear otros tipos
  - **TAD - Tipo Abstracto de Datos**
    listas, colas, pilas, arboles, grafos, etc...

Descriptos en la definición del lenguaje

Nuevos tipos a partir de los que ya hay

Se compone de rutinas que son operaciones definidas y se usan para manipular los objetos del tipo. No hay ligadura por defecto, el programador se encarga de especificar representación y operaciones.

Momento de ligadura:

Estático: ligadura se hace en compilación, puede ser de forma explícita, implícita o inferida.
Explícita: es cuando digo x es de tipo entero
Implícita: se deduce según las reglas, creo que es tipo assembler en que según el registro es entereo o real
Inferida: se deduce de los tipos de los componentes

Dinámico: Dinámico: se liga en ejecución, lo puedo ir cambiando. Es más flexible pero más dificil de mantener y menos legible

<NOMBRE, ALCANCE, TIPO, **L-VALUE**, R-VALUE>

- **Área de memoria ligada a la variable**
- **Tiempo de vida** (lifetime) o extensión:

  Periodo de tiempo que existe la ligadura
- **Alocación**:

  Momento que se reservar la memoria

El tiempo de vida es el tiempo en que la variable esté alocada en memoria

A dónde tengo que ir para guardar el valor que está a la derecha.
El lugar de mem donde se almacena la variable es un atributo.

No se cargan todas las memorias del programa a la vez.
Tiempo de vida es tiempo desde que se aloca (pone en memoria) hasta que se pierde la referencia.

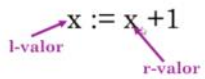
Estática: incluso cuando ya se está cargando en memoria el programa ya se les aloca espacio. Cuando termina el programa se borran. Le doy espacio a todas a la vez cuando lo cargo

Persistente: el tiempo de vida no depende de la ejecución, existe en el ambiente. Archivos, bases de datos

Dinámica:
automática: se alocan cuando se empieza a ejecutar el program o cuando se empieza a ejecutar y arranca la rutina en la que están.
Explícita: lo decide el programador

<NOMBRE, ALCANCE, TIPO, L-VALUE, R-VALUE>

- Valor almacenado en el l-valor de la variable
- Se interpreta de acuerdo al tipo de la variable
- Objeto: (l-valor, r-valor)



Se accede a las variable a través del **l-valor**

Se puede modificar el **r-value**

- **¿Cuál es el r-valor luego de crearse la variable?**
  - **Estrategia de inicialización:**
    1. **Inicialización por defecto:**
       - **Enteros** se inicializan en **0**
       - **Caracteres** en **blanco**
       - **Funciones** en **VOID**, etc.
    2. **Inicialización en la declaración:**
       C int i =0, j= 1 ADA I,J INTEGER:=0

Binding Dinámico de una variable a su valor

- el **valor (*r-valor)*** puede **cambiar durante la ejecución** con **una asignación.**
- el **valor (*r-valor) no* puede cambiar** si se define como **constante simbólica definida por el usuario**

  - **b := a (**copia el *r-valor* de **a** en el *l-valor* de **b** y **cambia** el *r-valor* de **b)**
  - **a :=17** (asigna un valor directamente)
  - **Constante: se congela el valor**

68