Conceptos de Lenguajes de Programación

Names, Scopes and Bindings

Guzman Pieroni y Gianfranco Stefanoli

FACULTAD DE INGENIERIA

Introducción

Se van a analizar los conceptos de Names, Scopes y Binding, vinculados a lenguajes de alto nivel.

Los lenguajes de programación de alto nivel son denominados así debido al nivel de abstracción que ofrecen en comparación con el lenguaje ensamblador, el cual opera mucho más cerca del código de máquina y del hardware.

La abstracción se mide con dos conceptos principales:

- La separación de cualquier arquitectura de computadora en particular.
- La facilitación del desarrollo al programador.

Binding Time

- Binding: Definición del libro

  Asociación entre dos cosas, un nombre y la cosa a la que este referencia.

Luego de esto se desprende la noción de **Binding Time**, como el tiempo en el que la asociación anterior se formó.

- Definición “más formal” del Binding Time

  Momento en el proceso de desarrollo de software en el cual una propiedad, como el tipo de una variable o el valor de una constante, queda determinada o "vinculada" a un nombre.

Binding Time

- Se evidencian distintos tiempos
  - Tiempo de Diseño del Lenguaje
    - Es el momento en el que se definen las características y restricciones del lenguaje en sí.
  - Tiempo de Implementación del Lenguaje
    - Se refiere al momento en que se toman decisiones de implementación específicas que no estaban determinadas por el diseño del lenguaje en sí, pero se realizan antes de la compilación.
  - Tiempo de Redacción del Lenguaje
    - El binding en este tiempo incluye decisiones como el algoritmo a utilizar, la estructura de los datos y la modularidad del código.
  - Tiempo de Compilación del Lenguaje
    - Es cuando el código fuente es transformado en código objeto por el compilador. En este momento se realizan vinculaciones de tipos de datos, resolución de nombres de variables y funciones, y la evaluación de expresiones constantes, entre otros.
  - Tiempo de linkeo del Lenguaje
    - Ocurre después de la compilación, cuando se combinan distintos módulos de código objeto para formar un único ejecutable.
  - Tiempo de Carga del Lenguaje
    - Es cuando el programa es cargado en memoria para su ejecución. Las direcciones de memoria se asignan a las variables y funciones en este momento.
  - Tiempo de Ejecución del Lenguaje
    - Es el tiempo durante el cual el programa está en ejecución y realizando tareas. Aquí se realizan vinculaciones dinámicas.

Binding Time

- Binding Time temprano vs tardío
  - En general, los tiempos de vinculación tempranos se asocian con una mayor eficiencia, mientras que los tiempos de vinculación posteriores se asocian con una mayor flexibilidad.

Conceptos Importantes

- Creación de Objetos
- Creación de Bindings
- Referencias
  - a variables, subrutinas, tipos, etc.
- Desactivación y Reactivación de Bindings
  - vinculadas a los tiempos de inutilización.
- Destrucción de Bindings
- Destrucción de Objetos

Vida útil : *Tiempo entre creacion y destruccion.*

Asignaciones de Memoria

Tipos de Asignación

1. **Static Allocation**
   - A los objetos se les asigna una dirección absoluta que se conserva durante la ejecución del programa
2. **Stack Allocation**
   - A los objetos se asignan y desasignan en orden de último en entrar, primero en salir, generalmente junto con llamadas y devoluciones de subrutinas.
3. **Heap Allocation**
   - Los objetos pueden ser asignados y desasignados en momentos arbitrarios. (Más costoso)

1. Static Allocation

Definición: La asignación estática se refiere a la reserva de espacio en memoria para objetos que no cambian de ubicación durante la ejecución del programa.

- Ejemplos : Variables globales, literales constantes, tablas del compilador.
- Objetos estáticos cuyo valor no debe cambiar durante la ejecución del programa.
- Se colocan frecuentemente en memoria protegida de solo lectura para evitar escrituras accidentales que podrían causar errores en tiempo de ejecución.

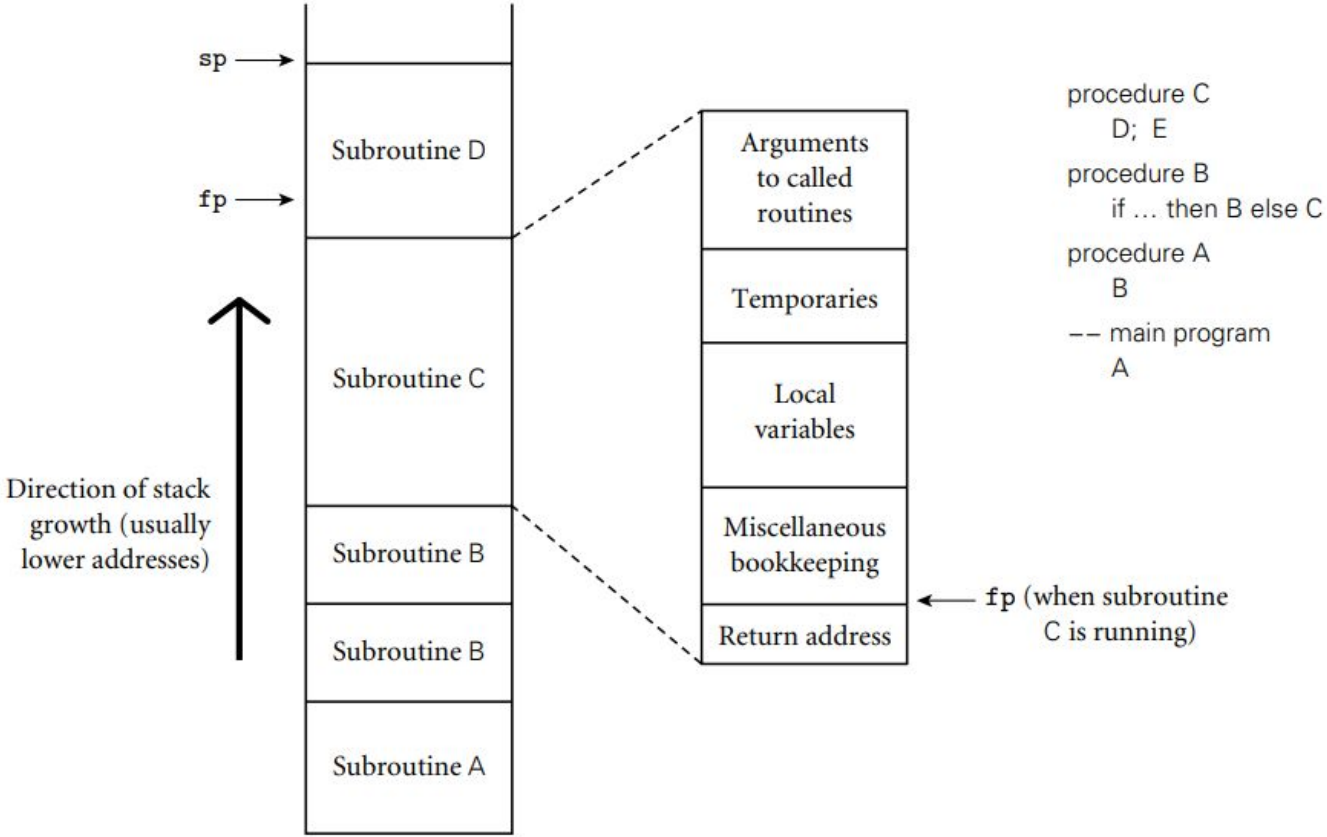
2. Stack Allocation

- La asignación basada en pila es esencial para lenguajes que permiten recursión, dado que el número de instancias de una variable local puede ser conceptualmente ilimitado.
- Funcionamiento del stack: Cada invocación de una subrutina en tiempo de ejecución tiene su propio frame en la pila, que contiene:
  - Argumentos y valores de retorno.
  - Variables locales.
  - Valores temporales.
  - Información de control (como la dirección de retorno y el enlace dinámico).
- Ventaja sobre Static Allocation : Un stack puede requerir sustancialmente menos memoria en tiempo de ejecución en comparación con la asignación estática.

2. Stack Allocation

- Acceso a Variables: Mientras que la ubicación de un marco de pila no puede predecirse en tiempo de compilación, los desplazamientos de los objetos dentro de un marco pueden determinarse estáticamente.
- Crecimiento del stack: La pila crece hacia direcciones de memoria más bajas en la mayoría de las implementaciones

2. Stack Allocation



3. Heap Allocation

- Heap: Es una región de almacenamiento.
- Característica principal: Los sub bloques pueden asignarse y liberarse en momentos arbitrarios.
- Es esencial para estructuras de datos dinámicas y objetos cuyo tamaño puede cambiar.
    - Ejemplos: Cadenas de caracteres, listas y conjuntos.
- Gestión del Espacio en el Heap: Dos estrategias claras.
    - Fragmentación Interna: Ocurre cuando se asigna un bloque más grande de lo necesario, dejando espacio sin usar dentro del bloque asignado.
    - Fragmentación Externa: Sucede cuando el espacio libre del heap está disperso en múltiples bloques pequeños, de tal manera que puede que no haya un bloque suficientemente grande para satisfacer una solicitud futura.
        - La misma puede empeorar con el tiempo.

3. Heap Allocation

- Listas de Bloques Libres: Para asignar bloques.
  - First Fit: Selecciona el primer bloque que sea lo suficientemente grande para satisfacer la solicitud.
  - Best Fit: Busca en toda la lista para encontrar el bloque más pequeño que cumpla con la solicitud.
- Optimización de la asignación: Algunos algoritmos mantienen listas separadas para bloques de diferentes tamaños, dividiendo el heap en "pools" de tamaño estándar.
  - Se visualiza una reducción del tiempo O(n) a constante.
  - Dos tipos: Buddy Systems, Fibonacci Heaps.

Garbage Collection

- Definición: Mecanismo para la desasignación implícita de objetos en el heap que ya no son accesibles desde ninguna variable del programa.
- Algunos lenguajes requieren liberación explícita (C,C++,etc), y otros aplican esta técnica.
  - Ejemplos: Modula-3, Java y C#.
- Ventajas a favor de la liberación explícita: Simplicidad de implementación, velocidad de ejecución.
- Ventajas a favor de la Garbage Collection: Reducción de errores, disminución de la dificultad de depuracion, mas utilizable en programas complejos, algoritmos cada vez mas óptimos.

Reglas de Scope

- Scope: El alcance (scope) de una variable se refiere a la región textual del programa en la que un binding está activo. En su uso más general, es una región del programa donde no cambian los enlaces, o al menos no se destruyen.
- La mayoría de los lenguajes modernos determinan el scope de una variable de forma estática, es decir, en tiempo de compilación.
    - Por ejemplo, en C, al entrar a una subrutina se introduce un nuevo scope y se crean enlaces para los objetos locales, mientras que los enlaces a objetos globales que son ocultados por objetos locales con el mismo nombre se desactivan. Al salir de la subrutina, se destruyen los enlaces a las variables locales y se reactivan los enlaces a los objetos globales ocultos. Estas manipulaciones de enlaces se determinan en tiempo de compilación y no requieren la ejecución de ningún código.

Reglas de Scope

- C es considerado un lenguaje de scope estático o léxico, ya que la relación entre nombres y objetos está basada en reglas textuales y puede conocerse durante la compilación.
- Elaboración: Proceso por el cual las declaraciones se activan al entrar por primera vez a un scope.
    - Usado en ADA y Algol 68
    - La elaboración implica la creación de enlaces y, en muchos lenguajes, también la asignación de espacio en la pila para objetos locales y la asignación de valores iniciales.

Scoping Estático

- Las vinculaciones entre nombres y objetos se pueden determinar en tiempo de compilación examinando el código del programa, sin considerar el flujo de control en tiempo de ejecución.
- Reglas de Scope en Basic: Las primeras versiones de Basic tenían una regla de scope estático muy simple, con un único scope global y un número limitado de nombres posibles para las variables, que no requerían declaraciones explícitas.
- Reglas de Scope en Fortran: Fortran (antes de Fortran 90) distingue entre variables globales y locales.
- Variables Estáticas en C: En C, la palabra clave **static** se utiliza para declarar variables que retienen su valor entre invocaciones de la subrutina en la que aparecen, extendiendo así su vida útil a la ejecución completa del programa. Aunque la vinculación nombre-variable es inactiva cuando la subrutina no se está ejecutando, el objeto en sí conserva su valor.

```
void label_name (char *s) {
    static short int n;          /* C guarantees that static locals
                                    are initialized to zero */
    sprintf (s, "L%d\0", ++n);   /* "print" formatted output to s */
}
```

Subrutinas Anidadas

- La capacidad de anidar subrutinas dentro de otras fue introducida en Algol 60 y es una característica de muchos lenguajes modernos, incluyendo Pascal, Ada, Python, etc.
- Otros lenguajes, como C y sus descendientes, permiten anidar clases u otros scopes.
- Visibilidad en scopes Anidados: La regla de scope más cercano anidado establece que un nombre introducido en una declaración es conocido en el scope en el que se declara y en cada scope anidado internamente (a menos que esté oculto por otra declaración del mismo nombre en uno o más scopes anidados).
- Búsqueda de Vinculaciones: Para encontrar el objeto correspondiente a un uso dado de un nombre, buscamos una declaración con ese nombre en el scope actual, el más interno. Si no hay ninguna, buscamos en el scope inmediatamente circundante y continuamos hacia afuera hasta llegar al nivel de anidamiento externo del programa, donde se declaran los objetos globales.

Orden de Declaración

- ¿Puede una expresión E referirse a cualquier nombre declarado en el scope actual, o sólo a nombres que están declarados antes de E en el scope?
    - Varios lenguajes antiguos, como Algol 60 y Lisp, requerían que todas las declaraciones aparecieran al inicio de su scope.
    - Pascal modificó el requerimiento para establecer que los nombres deben ser declarados antes de ser utilizados, con mecanismos de casos especiales para acomodar tipos y subrutinas recursivas.
    - Sin embargo, Pascal mantuvo la noción de que el scope de una declaración es el bloque circundante completo. Estas dos reglas pueden interactuar de maneras inesperadas.
    - C++ y Java relajan aún más las reglas al prescindir del requisito de definir antes de usar en muchos casos.
        - En ambos lenguajes, los miembros de una clase (incluidos aquellos que no se definen hasta más adelante en el texto del programa) son visibles dentro de todos los métodos de la clase. En Java, las clases mismas pueden declararse en cualquier orden.

Definición vs Declaración

- Los tipos y subrutinas recursivos presentan un problema para los lenguajes que requieren que los nombres sean declarados antes de poder ser utilizados
- Una declaración introduce un nombre e indica su scope, pero puede omitir ciertos detalles de implementación.
- Una definición describe el objeto con suficiente detalle para que el compilador pueda determinar su implementación.

Módulos

- Es un desafío importante dividir el desarrollo del programa en varias partes.
- Esta modularización del esfuerzo depende críticamente de la noción de ocultamiento de información, que hace que los objetos y algoritmos sean invisibles, siempre que sea posible, a las partes del sistema que no los necesitan.
- Un código adecuadamente modularizado reduce la "carga cognitiva" del programador al minimizar la cantidad de información necesaria para comprender cualquier parte del sistema

Módulos como Abstracciones

- Un módulo permite encapsular una colección de objetos (subrutinas, variables, tipos, etc.) de tal manera que:
    - los objetos internos son visibles entre sí,
    - los objetos internos no son visibles externamente a menos que se exporten explícitamente,
    - y en muchos lenguajes, los objetos externos no son visibles internamente a menos que se importen explícitamente. Estas reglas solo afectan la visibilidad de los objetos, no su vida útil.

Orientación a Objetos

- Las clases pueden considerarse como tipos de módulos que han sido mejorados con un mecanismo de herencia.
- Objeto: Instancia de una clase.
- La herencia permite que nuevas clases se definan como extensiones o refinamientos de clases existentes, facilitando un estilo de programación en el que todas o la mayoría de las operaciones se consideran como pertenecientes a objetos, y en el que los nuevos objetos pueden heredar la mayoría de sus operaciones de objetos existentes, sin la necesidad de reescribir código.
- Cada instancia A de un tipo de clase tiene una copia separada de las variables de la misma.
    - Estas variables son visibles al ejecutar una de las operaciones de A. También pueden ser indirectamente visibles para las operaciones de alguna otra instancia B si A se pasa como parámetro a una de esas operaciones.

Clases

- Las clases pueden considerarse como tipos de módulos que han sido mejorados con un mecanismo de herencia,
- Se permite que nuevas clases se definan como extensiones o refinamientos de clases existentes, facilitando un estilo de programación en el que casi todas las operaciones se consideran como pertenecientes a objetos.
- Módulos y Clases en Aplicaciones Grandes: Muchas aplicaciones necesitan tanto abstracciones de múltiples instancias como subdivisiones funcionales.
    - En reconocimiento de este hecho, muchos lenguajes, incluyendo C++, Java, C#, Python y Ruby, proporcionan mecanismos separados para clases y módulos.

Implementación del Scope

- Estático: El compilador se basa en una abstracción de datos llamada tabla de símbolos
  - La tabla de símbolos es un diccionario, mapea nombres a la información que el compilador conoce sobre ellos.
  - Las operaciones más básicas son insertar un nuevo mapeo (una vinculación nombre-objeto) o buscar la información que ya está presente para un nombre dado.
- Dinámico: un intérprete debe realizar operaciones análogas a la inserción y búsqueda en la tabla de símbolos en tiempo de ejecución.
  - En principio, cualquier organización utilizada para una tabla de símbolos en un compilador podría usarse para realizar un seguimiento de las vinculaciones nombre-objeto en un intérprete y viceversa.

Alias

- Los alias son referencias múltiples al mismo espacio de memoria bajo diferentes nombres.
- Ejemplos simples de alias se encuentran en los bloques comunes y declaraciones de equivalencia de Fortran, y en los registros variantes y uniones de lenguajes como Pascal y C.
- También surgen de forma natural en programas que utilizan estructuras de datos basadas en punteros.

```
int a, b, *p, *q;
...
a = *p;     /* read from the variable referred to by p */
*q = 3;     /* assign to the variable referred to by q */
b = *p;     /* read from the variable referred to by p */
```

Sobrecarga

- La sobrecarga (overloading) es un concepto en la programación que se refiere a la capacidad de definir varias funciones o métodos con el mismo nombre pero con diferentes firmas o conjuntos de parámetros en un mismo scope.
- Permite que un nombre represente diferentes comportamientos dependiendo del contexto, especialmente del número y tipo de los argumentos que se pasan a la función o método.
- El compilador o el intérprete determina cuál de las funciones sobrecargadas se debe llamar en tiempo de compilación.
- La sobrecarga se utiliza en muchos lenguajes de programación modernos, como C++, Java, C# y Python, y facilita la legibilidad y la reutilización del código

Polimorfismo

- Capacidad de una función, método o objeto de operar o ser aplicado a diferentes tipos de datos.
  - El término proviene del griego "poly" (muchos) y "morph" (forma), lo que indica la habilidad de tomar múltiples formas.
- Se distinguen dos tipos
  - Polimorfismo Paramétrico: El código toma un tipo (o conjunto de tipos) como parámetro, ya sea explícita o implícitamente.
  - Polimorfismo de Subtipos: El código está diseñado para trabajar con valores de un tipo específico T, pero se pueden definir tipos adicionales como extensiones o refinamientos de T, y el código polimórfico trabajará con estos subtipos también.

Extras

- Clausura de Subrutina: técnica de implementación para funciones de primera clase en lenguajes de programación que permite a una función recordar el entorno en el que fue creada, incluso después de que ese entorno haya dejado de existir en la pila de ejecución.
    - Esto es especialmente útil para la programación funcional y para situaciones en las que una función se pasa como argumento o se devuelve desde otra función.
- Extensión Ilimitada: Si los objetos locales fueran destruidos al final de la ejecución de cada scope, entonces el entorno de referencia capturado en un cierre de larga duración podría llenarse de referencias colgantes. Para evitar este problema, la mayoría de los lenguajes funcionales especifican que los objetos locales tienen una extensión ilimitada: sus vidas útiles continúan indefinidamente y su espacio solo puede ser reclamado cuando el sistema de recolección de basura pueda probar que nunca se volverán a usar.
    - Por otro lado, los objetos locales en la mayoría de los lenguajes imperativos tienen una extensión limitada: se destruyen al final de la ejecución de su scope.

Extras

- Clausura de Objetos: Técnica de programación orientada a objetos que permite a un método recordar el entorno en el que fue creado, similar a como una función puede hacerlo en lenguajes funcionales.
    - Esto se logra encapsulando el método como un miembro de un objeto y utilizando los campos de dicho objeto para mantener el contexto necesario para el método.
- Expansión de Macros: Las macros son una característica de algunos lenguajes de programación que permite la sustitución de texto en el código fuente antes de la compilación o interpretación.
    - Originalmente, las macros proporcionaban a los programadores de lenguajes ensambladores una manera de escribir código repetitivo de manera más concisa y mantenible.
    - En C, sufren varias limitaciones. En ensambladores, se podían definir macros para automatizar la generación de secuencias de instrucciones complejas
    - Los lenguajes y compiladores modernos han abandonado en gran medida las macros en favor de constantes con nombre y subrutinas en línea

Extras

- Compilación Separada: Es una característica necesaria para lenguajes de programación que soportan el desarrollo de programas grandes, los cuales se construyen y prueban de manera incremental.
    - Dado que la compilación de un programa muy grande puede tomar varias horas, los lenguajes diseñados para soportar programas grandes deben permitir la compilación de partes individuales del programa de manera independiente.
    - Los módulos, que están diseñados para la encapsulación y proveen una interfaz estrecha, son la elección natural para las "unidades de compilación" de muchos lenguajes de programación.

Fin
Muchas Gracias