Resumen Segunda EMT CPLP

Para esta EMT se evalúan los contenidos de las clases 4, 5 y 6, también resumo la clase 7 hasta Tipos de Datos Abstractos

Clase 4

Unidades o Rutinas

- Los lenguajes de programación permiten que un programa esté compuesto por **unidades** que realizan **acciones abstractas**.
- Estas unidades o rutinas se pueden dividir en 2:
    - **Procedimientos**: No retornan ningún valor.
    - **Funciones**: Estas retornan un valor.

- Existen lenguajes que solo tienen **funciones** y “simulan” los **procedimientos** con **funciones** que retornan **Void** (C, C++, Python, etc).

Atributos de una Unidad o Rutina

Nombre

- **String de caracteres que se usa para invocar a la rutina. (identificador).**
- Este **nombre** se introduce en su **declaración**.
- Toda **llamada** a una **rutina** debe de estar **dentro** del **alcance** del **nombre** de la misma.

Alcance

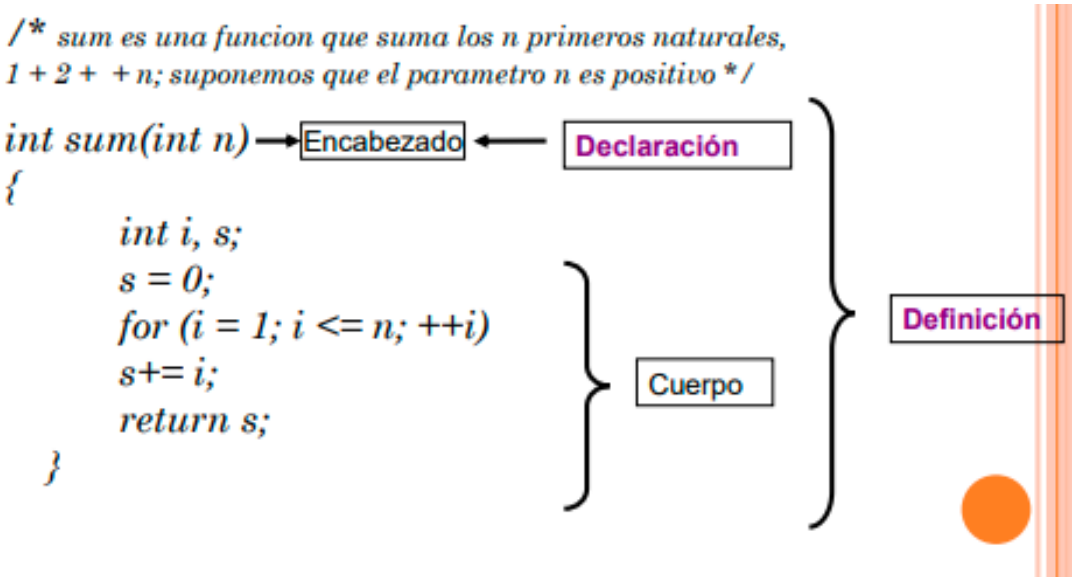
- **Rango de instrucciones donde se conoce el nombre de la rutina.**
- El **alcance** se extiende desde el punto de su **declaración** hasta algún **constructor de cierre**.
- Según el lenguaje puede ser **estático** o **dinámico**.

Activación

- La **llamada** puede estar solo **dentro del alcance** de la **rutina.**

Definición vs Declaración

- Algunos lenguajes (C, C++, Ada, etc) hacen distinción entre Definición y Declaración de las rutinas.



- Si el lenguaje distingue entre la **declaración** y la **definición** de una **rutina permite manejar esquemas de rutinas mutuamente recursivas.**
- Según la **implementación** de cada lenguaje, si no se encuentra la **declaración** de la **función o prototipo** y se hace un **llamado** a la **misma**, en **C** se asume que el **prototipo** devuelve **enteros**, en **C++ no se asume** ningún **prototipo**, en **Pascal** se hace uso de **forward**, etc.

Tipo

- El **encabezado** de la rutina define el **tipo** de los **parámetros** y el **tipo** del **valor de retorno (si lo hay).**
- Un **llamado** a una **rutina** es **correcto si está de acuerdo** al **tipo** de la **rutina**.
- La **conformidad** requiere la **correspondencia** de **tipos** entre **parámetros formales y reales.**

Signatura

- **Permite especificar el tipo de una rutina.**
- Una **rutina** "**fun**" que tiene como **entrada parámetros** de tipo **T1, T2, Tn** y **devuelve** un **valor** de tipo **R**, puede especificarse con la siguiente **signatura *"fun: T1xT2x....Tn -> R".***

L-Valor

- Es el lugar de **memoria** en el que se **almacena** el **cuerpo** de la **rutina**.

R-Valor

- La **llamada** a la **rutina** causa la **ejecución** de su **código**, **eso constituye su r-valor**.
    - **Estático**: El caso más usual.
    - **Dinámico**: **Variables** de tipo **Rutina**. Se **implementan** a través de **punteros a rutinas** que permiten una política dinámica de invocación de las mismas**.**

Comunicación entre Rutinas

- **Ambiente no local.**
- **Parámetros**:
    - Diferentes datos en cada llamado a una rutina.
    - Dan mayor legibilidad y modificabilidad.

Parámetros Formales

- Son los que aparecen en la **definición** de la **rutina**.

Parámetros Reales

- Son los que aparecen en la **invocación** de la **rutina**, ya sean **datos** u otras **rutinas**.

Ligadura entre Parámetros Formales y Reales

Método posicional

- Se ligan uno a uno:
    - ***routine S (F1,F2,……,Fn) Definición.***
    - ***call S (A1, A2,….. An) Llamado.***

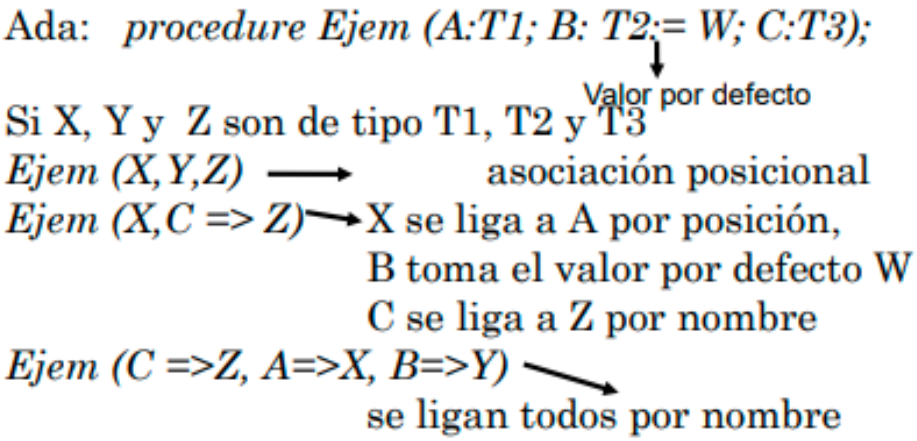
- ***Fi*** se liga a ***Ai*** para "i" de 1 a n.
- **Deben conocerse las posiciones.**

Variante del Posicional

- Combinación con valores por defecto (C++):
  - ***int distancia (int a = 0, int b = 0) Definición.***
  - ***distancia() -> distancia (0, 0) Llamado.***
  - ***distancia(10) -> distancia (10, 0) Llamado.***

Método por Nombre

- Se ligan por el nombre (Ada):



- **Deben conocerse los nombres de los formales.**

Representación en Ejecución

- La **definición** de la **rutina** especifica un **proceso de cómputo**.
- Cuando se **invoca** una **rutina** se **ejecuta** una **instancia** del **proceso** con los particulares **valores** de los **parámetros**.

Instancia de la Unidad

- **Es la representación de la rutina en ejecución.**
  - **Segmento de Código**: Las instrucciones de la unidad se almacenan en la **memoria de instrucción C**. **Tienen Contenido Fijo.**
  - **Registro de activación**: Los datos locales de la unidad se almacenan en la **memoria de datos D**. **Tienen Contenido Cambiante.**

Procesador Abstracto

- Sirve para comprender qué efecto causan las instrucciones del lenguaje al ser ejecutadas.
- Se describe una semántica intuitiva del lenguaje de programación a través de reglas de cada constructor del lenguaje traduciéndose en una secuencia de instrucciones equivalentes del procesador abstracto.

Memoria de Código

- **C(y)** valor **almacenado** en la **y-ésima celda** de la **memoria de código**.
- Comienza en cero.

Memoria de Datos

- **D(y)** valor **almacenado** en la **y-ésima celda** de la **memoria de datos**.
- Comienza en cero y representa el L-Valor, D(y) o C(y) su R-Valor.

Ip

- **Puntero a la instrucción que se está ejecutando.**
- Se **inicializa** en **cero** en **cada ejecución** y se **actualiza** cuando se **ejecuta** cada **instrucción**.

Ejecución:

- obtener la instrucción actual para ser ejecutada (C[ip])
- incrementar ip
- ejecutar la instrucción actual

Instrucciones

- **SET**: **Setea valores en la memoria de datos**. Tiene la forma: ***set target,source*** -> **Copia** el **valor** representado por "**source**" en la **dirección** representada por "**target**". Ejemplo: ***set 10,D[20].***

- **E/S: Read y Write permiten la comunicación con el exterior.** Ejemplos: ***set 15,read*** -> El **valor leído** se **almacenará** en la posición 15; ***set write,D[50]*** -> Se **transfiere** el **valor almacenado** en la posición 50.

- **JUMP: Bifurcación incondicional.** Ejemplo: ***jump 47*** -> La **próxima instrucción** a **ejecutarse** será la que esté **almacenada** en la **dirección 47 de C.**

- **JUMPT: Bifurcación condicional, bifurca si la expresión es verdadera.** Ejemplo: ***jumpt 47,D[13]>D[8]*** -> **Bifurca** si el **valor** almacenado en la celda 13 es **mayor** que el almacenado en la celda 8.

Punto de Retorno

- Es una pieza cambiante de información que debe ser salvada en el registro de activación de la unidad llamada.

Ambiente de Referencia

- **Ambiente local:** variables locales, ligadas a los objetos almacenados en su registro de activación.

- **Ambiente no local:** variables no locales, ligadas a objetos almacenados en los registros de activación de otras unidades.

Estructura de Ejecución de los Lenguajes de Programación

Esquema Estático - Espacio Fijo

- El espacio necesario para la ejecución se deduce del código.
- Todos los requerimientos de memoria ya sea de código o de datos necesarios se conocen antes de la ejecución, es decir, **se carga todo cuando se empieza a ejecutar el programa.**
- Las unidades en la memoria **perduran durante toda la ejecución del programa.**
- **Toda variable que se use en este esquema será estática según su tiempo de vida.**
- La alocación puede hacerse estáticamente.
- No puede haber recursión.
- **Utiliza C2.**

C1

- Programa sencillo sin bloques internos, solo código programa principal.
- En zona de datos (Registro de Activación): SOLO **datos locales.**

C2 - C1 + Rutinas Internas

- Programa con bloques internos (procedimientos) SIN anidamientos.
- En zona de datos (Registro de Activación): **datos locales + pto. de retorno.**
- Datos Globales.
- Puede ligar cada variable con su desplazamiento dentro del correspondiente registro de activación. El desplazamiento es estático

Esquema Basado en Pila - Espacio Predecible

- El espacio se deduce del código.
- La memoria a utilizarse es predecible y sigue una disciplina last-in-first-out.
- Las variables se alocan automáticamente y se desalocan cuando el alcance se termina.
- Se utiliza una estructura de pila para modelizarlo, esta no es parte de la semántica del lenguaje.
- **Utiliza C5.**

Esquema Dinámico - Espacio Impredecible

- Los datos son alocados dinámicamente sólo cuando se los necesita durante la ejecución.
- No pueden modelizarse con una pila, el programador puede crear objetos de datos en cualquier punto arbitrario durante la ejecución del programa.
- Los datos se alocan en la zona de memoria heap.
- **Utiliza C6.**

Clase 5

Continuación Esquema Basado en Pila

C3 - C2 + Recursión, Valor de Retorno, Link Dinámico, Current y Free

- Las Rutinas tienen capacidad de llamarse a sí mismas **(recursión directa)** o de llamar a otra rutina en forma recursiva **(recursión indirecta)**.
- Las Rutinas tienen capacidad de devolver valores, es decir, **funciones**.

Funcionamiento

- El registro de activación de cada unidad será de tamaño fijo y conocido, pero no se sabrá cuántas instancias de cada unidad se necesitarán durante la ejecución.
- Puede ligar cada variable con su desplazamiento dentro del correspondiente registro de activación. El desplazamiento es estático.
- La dirección donde se cargará el registro de activación, es dinámica, por lo tanto, la ligadura con la direcciones absolutas en la zona de Datos de la memoria, solo puede hacerse en ejecución.
- Cada nueva invocación aloca un nuevo registro de activación y se establecen las nuevas ligaduras entre el segmento de código y el nuevo registro de activación.
- Las unidades pueden devolver valores (funciones) y esos valores NO deberían perderse cuando se desactive la unidad **(NECESIDAD DEL VALOR DE RETORNO)**.
- Cuando la instancia actual de la unidad termine de ejecutarse, su registro de activación no se necesitará más, por lo tanto se puede liberar el espacio ocupado por su registro de activación y dejar el espacio disponible para nuevos registros de activación **(NECESIDAD DEL LINK DINÁMICO, CURRENT Y FREE).**

Datos Necesarios

- **Valor de Retorno:** Al terminar una rutina se desaloca su RA, por lo tanto la rutina llamante debe guardar en su RA el valor de retorno de la rutina llamada.
- **Link Dinámico:** Contiene un puntero a la dirección base del registro de activación de la rutina llamadora.
- **Head - Current:** Dirección base del registro de activación de la unidad que se esté ejecutando actualmente.
- **Head - Free:** Próxima dirección libre en la pila.

Cadena Dinámica

- Cadena de links dinámicos originada en la secuencia de registros de activación activos.
- Representa la secuencia dinámica de unidades activadas.

C4 - Estructura de Bloque

Estructura de Bloque

- Controla el alcance de las variables.
- Define el tiempo de vida de las variables.
- Divide el programa en unidades más pequeñas.
- Pueden ser:
    - **Disjuntos:** No tienen porciones en común.
    - **Anidados:** Un bloque está completamente contenido en otro.

C4' - Anidamiento vía Sentencias Compuestas

- Permite que dentro de las sentencias compuestas aparezcan declaraciones locales.
- Un bloque tiene forma de una sentencia compuesta: ***{<lista de declaraciones>;<lista de sentencias>}***.
- **Las variables tienen alcance local:** son visibles dentro de la sentencia compuesta, incluyendo cualquier sentencia compuesta anidada en ella.
- Si en el anidamiento, hay una nueva declaración de un nombre, la declaración interna enmascara la externa del mismo nombre.
- **A partir de estas sentencias compuestas los RA correspondientes se pueden crear de 2 formas:**
    - **Estático**: Incluir todas las necesidades dentro del registro de activación de la unidad a la que pertenece, reservando espacio para todo. Esta forma es simple y eficiente.
    - **Dinámico**: Alocar el espacio dinámicamente cuando se ejecutan las sentencias. Eficiente en cuanto a espacio utilizado.

C4" - Anidamiento de Rutinas

- Permite la definición de una rutina dentro de otras rutinas. (anidamiento de rutinas).
- Se genera la **necesidad del link estático** para poder localizar referencias no locales a la rutina.

Datos Necesarios

- **Link Estático:** Apunta al registro de Activación de la unidad que estáticamente la contiene.

Cadena Estática

- Secuencia de links estáticos.
- Permite localizar esas referencias no locales, en lenguajes que siguen la cadena estática.

C5 - Datos más Dinámicos

C5’ - Datos Semidinámicos

- Registro de activación cuyo tamaño se conoce cuando se activa la unidad.
- Variables cuyo tamaño se conoce en compilación.
- Un ejemplo de estas variables son los **Arreglos Dinámicos.**

Datos Semidinámicos en Compilación

- **VIendo el caso de los Arreglos Dinámicos**:
    - En **Compilación** se reserva lugar en el registro de activación para los descriptores de los mismos y todos los accesos al arreglo serán traducidos como referencias indirectas a través del puntero en el descriptor, cuyo desplazamiento se determina estáticamente.

Datos Semidinámicos en Ejecución

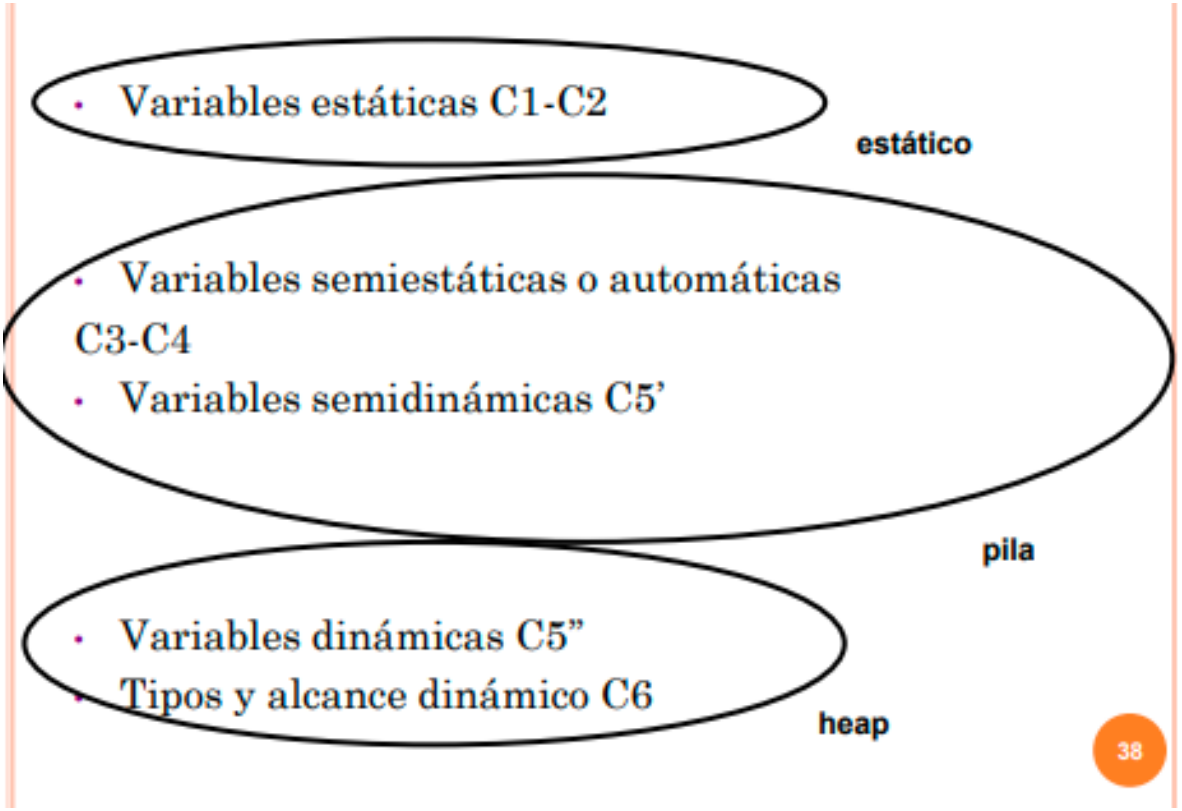
- **El RA se aloca en varios pasos:**
    1. Se aloca el almacenamiento para los datos de tamaño conocido estáticamente y para los descriptores de los arreglos dinámicos.
    2. Con la declaración se calculan las dimensiones en los descriptores y se extiende el registro de activación para incluir el espacio para la variable dinámica.
    3. Se fija el puntero del descriptor con la dirección del área alocada

C5” - Datos Dinámicos

- Se alocan explícitamente durante la ejecución en la heap mediante instrucciones de alocación.
- El tiempo de vida de estos datos no depende de la sentencia de alocación, vivirán mientras estén apuntados.

C6 - Lenguajes Dinámicos

- Se trata de aquellos lenguajes que adoptan más reglas dinámicas que estáticas.
- Usan **tipado** dinámico y reglas de **alcance** dinámicas.
- Se podrían tener reglas de tipado dinámicas y de alcance estático, pero en la práctica las propiedades dinámicas se adoptan juntas.
- Una propiedad dinámica significa que las ligaduras correspondientes se llevan a cabo en ejecución y no en compilación.



Clase 6

Rutinas

- También llamadas **Subprogramas**. Son una **Unidad de Programa *(función, procedimiento)*.**
- Están formadas por un conjunto de sentencias que representan una acción abstracta.
- Permiten ampliar a los lenguajes, dan modularidad, claridad y buen diseño.
- Se **lanzan** con una **llamada explícita (se invocan por su nombre)** y luego **retornan** a **algún punto de la ejecución (responden al esquema call/return).**
- Permiten al programador definir una nueva operación a semejanza de las operaciones primarias ya integradas en el lenguaje.

Procedimientos

- **Construcción** que permite **dar nombre a un conjunto de sentencias y declaraciones asociadas** que se usarán para resolver un subproblema dado.
- Brindan una **solución** de **código** más **corta**, **comprensible** y **fácilmente modificable**.
- Permiten al **programador definir** y **crear** nuevas **acciones/sentencias**
- El **programador** las **invocará**.
- Los **resultados** los produce en **variables no locales** o en **parámetros** que cambian su valor.

- Se definen una única vez mientras que la cantidad de activaciones pueden ser las que el programador quiera.

Funciones

- Mientras que **un procedimiento ejecuta un grupo de sentencias**, una **función** además **devuelve un valor al punto donde se llamó.**
- Permite al programador crear **nuevas operaciones.**
- **Se las invoca dentro de expresiones y lo que calcula reemplaza a la invocación dentro de la expresión.**
- **Siempre** deben retornar un valor.
- Se definen una única vez mientras que la cantidad de activaciones pueden ser las que el programador quiera.
- Similar a las funciones matemáticas ya que hacen algo y luego devuelven un valor y no producen efectos colaterales.

Formas de compartir datos entre Unidades

- Si las **unidades programa** utilizan **variables locales no hay problema.**
- Si las **unidades programa** utilizan **variables que no son locales** hay 2 formas: A través del acceso al ambiente no local ó A través del uso de parámetros.

Compartir Datos A Través del acceso al ambiente no local

- Se basa en **compartir variables** que **son de otra unidad** con un acceso no local.
- Es **menos claro** y puede llevar a más **errores**.
- Se usa **cadena estática o dinámica.**

Ambiente no Local implícito

- Es **automático**.
- Utiliza alguna de las 2 reglas:
    - **regla de alcance dinámico - aplica cadena dinámica** ***(quién me llamó*** *y buscó el identificador)*
    - **regla de alcance estático - aplica cadena estática** ***(dónde está contenido*** *y busco el identificador)*

Ambiente no Local Explícito

- **Permite definir áreas comunes de código.**
- El **programador** debe **especificar** que es lo **comparte.**
- Cada lenguaje tiene su forma de realizarlo, Ejemplos:
    - **ADA**: Uso de paquetes con cláusula **PACKAGE** (para Tipos Abstractos de Datos - TAD).
    - **PL/1**: Variables externas con cláusula **DECLARE**.
    - **FORTRAN**: Área común con cláusula **COMMON.**

Compartir Datos A Través del Pasaje de Parámetros

- El **pasaje de parámetros es mejor**, ya que el **uso intensivo de accesos al ambiente no local** puede provocar alguna **pérdida de control**, y provocar que **las variables terminen siendo visibles donde no es necesario y llevar a errores.**

Ventajas

- Permite **enviar distintos parámetros en distintas invocaciones** a las rutinas.
- Brinda distintas **posibilidades de compartir cosas.**
- **Más flexibilidad**, se pueden transferir **más datos y de diferente tipo en cada llamada.**
- Permite **compartir en forma más abstracta**, sólo especificamos el **nombre a argumentos y parámetros**, y el **tipo** de cada cosa que se comparta.
- Brindan **modificabilidad** ya que si hay errores, uno se focaliza en qué cosas estoy compartiendo, qué argumentos y parámetros estoy utilizando y su tipo.
- **Ventajas en protección:**
    - El uso intensivo de accesos al ambiente no local decrementa la seguridad de las soluciones ya que las variables terminan siendo visibles aun donde no es necesario o donde no debería.
- **Ventajas en legibilidad:**
    - Permite al programador encontrar más fácilmente los errores.

Parámetro Real (Argumento)

- Es un valor, entidad, expresión u otro, que puede ser local, no local o global, que se pasa a un procedimiento o función.
- Están colocados en la parte de la invocación de la rutina.

Parámetro Formal (Parámetro)

- Es una variable utilizada para recibir valores de entrada en una rutina, subrutina etc.
- Están colocados en la parte de la declaración de la rutina.
- Son variables locales a su entorno.

Vinculación de los Parámetros (Evaluación - Ligadura)

- El **momento de vinculación de los Parámetros Formales y los Reales** comprende la **evaluación** de los **parámetros reales** y la **ligadura** con los **parámetros formales.**
- ***Evaluación:***
    - En general antes de la invocación primero se evalúan los parámetros reales, y luego se hace la ligadura.
    - Se verifica que todo esté bien antes de transferir el control a la unidad llamada.

- ***Ligadura:***
    - **Por posición:** Se corresponden con la posición que ocupan en la lista de parámetros. Van en el mismo orden.

    - **Por Nombre o palabra clave**: Se corresponden con el nombre por lo tanto pueden estar colocados en distinto orden en la lista de parámetros. La

desventaja que tiene es que cuando invocas hay que conocer/recordar el nombre de los parámetros formales para poder hacer la asignación. Esto puede llevar a cometer errores.

Tipos de Parámetros

Datos

- Desde el punto de vista semántico de los parámetros formales pueden ser: Modo IN, Modo OUT y Modo IN/OUT.

Modo IN

- El parámetro formal recibe el dato desde el parámetro real. La conexión es al inicio cuando se invoca la función, se copia y se corta la vinculación.

Modo IN por Valor

- **El valor del parámetro real se usa para inicializar el correspondiente parámetro formal al invocar la unidad.**
- Se **transfiere** el **dato real** y **se copia** en una nueva variable.
- En este caso **el parámetro formal actúa como una variable local de la unidad llamada, y crea otra variable.**
- La **conexión es al inicio** para pasar el valor y **se corta la vinculación.**
- Es el mecanismo por **default** y el **más usado**.
- ***Ventajas:***
    - El parámetro real se protege, ya que se hace una copia, por lo que el "verdadero" no se modifica. No hay efectos colaterales.
- ***Desventajas:***
    - Se consume tiempo en hacer las copias de los parámetros y almacenamiento para duplicar cada dato.

Modo IN por Valor Constante

- Se **envía un valor**, pero **la rutina receptora no puede modificarlo**, es decir queda con un valor fijo que no se puede cambiar.
- En Ada se especifican usando “in” en la declaración, y en C/C++ usando “const”.
- No indica si se realiza o no la copia. (dependerá del lenguaje).
- **La implementación debe verificar que el parámetro real no sea modificado.**
- No todos los lenguajes permiten este modo.
- ***Ventajas:***
    - Protege los datos de la unidad llamadora, el parámetro real no se modifica.
- ***Desventajas:***
    - Requiere realizar más trabajo para implementar los controles.

Modo OUT

- El valor del parámetro formal se copia al parámetro real al terminar de ejecutarse la unidad que fue llamada (rutina).

Modo OUT por Resultado

- El valor del **parámetro formal** de la rutina **se copia al parámetro real al terminar de ejecutarse la unidad que fue llamada.**
- El **parámetro formal es una variable local del entorno de la rutina.**
- El **parámetro formal** es una **variable sin valor inicial** porque **no recibe nada,** entonces, se debe inicializar de alguna forma si el lenguaje no lo hace por defecto.
- ***Ventajas:***
    - Protege los datos de la unidad llamadora, el parámetro real no se modifica en la ejecución de la unidad llamada
- ***Desventajas:***
    - Consume tiempo y espacio ya que lo que tenga el parámetro formal lo va a copiar en el entorno del parámetro real al final de la ejecución.
    - Debemos inicializar la variable en la unidad llamada de alguna forma (si el lenguaje no lo hace por defecto).
    - Si se repiten los parámetros reales los resultados pueden ser diferentes.
    - Se debe tener en cuenta el momento en que se evalúa el parámetro real.

Modo OUT por Resultado de Funciones

- Es el **resultado que me devuelven las funciones.**
- **Reemplaza la invocación en la expresión que contiene el llamado.**
- Puede devolverse de distintas formas según el lenguaje:
    - ***return*** como en Python, C, etc.
    - ***nombre de la función*** (último valor asignado) que se considera como una variable local como en Pascal.

Modo IN/OUT

- El parámetro formal recibe el dato del parámetro real y el parámetro formal le envía el dato al parámetro real al finalizar la rutina. La conexión es al inicio y al final.
- La conexión no dura toda la ejecución, se evalúa al inicio, conecta ligadura, ejecuta, evalúa, conecta ligadura.

Modo IN/OUT por Valor/Resultado

- El **parámetro formal** es una **variable local** que **recibe una copia a la entrada** del contenido **del parámetro real** y **a la salida el parámetro real recibe una copia** de lo que tiene el **parámetro formal.**
- **Cuando se invoca la rutina**, el parámetro real le da valor al parámetro formal (**se genera copia**) y **se desliga** en ese momento.
- **La rutina trabaja** sobre ese parámetro formal pero **no afecta al parámetro real trabaja sobre su copia**. Cada referencia al parámetro formal es una **referencia local**.

- Una vez que termina de ejecutar el parámetro formal le **devuelve un valor al parámetro real y lo copia.**
- Se dice que hay una **ligadura y una conexión entre parámetro real y el formal** cuando **se inicia la ejecución** de la rutina y cuando se **termina**, pero **no en el medio.**
- **Posee las ventajas y desventajas de los Modos IN y OUT.**

Modo IN/OUT por Referencia

- Comparte la dirección del parámetro real al parámetro formal. Haciendo que el parámetro formal actúe como un puntero al real.
- El **PF** será una **variable local a su entorno** que contiene la **dirección** al **PR** de la unidad llamadora que estará entonces en un **ambiente no local**. Así **se extiende el alcance de la rutina (aliasing situation).**
- **Cada referencia al PF será a un ambiente no local**, entonces **cualquier cambio** que se realice en el PF dentro del cuerpo del subprograma **quedará registrado en el PR**. **El cambio será automático**.
- **El PR queda compartido por la unidad llamadora y llamada. Será bidireccional.**
- ***Ventajas:***
    - Ya que no se realizan copias de los datos será eficiente en espacio y tiempo sobre todo en grandes volúmenes de datos.
    - La indirección es de bajo costo de implementar por muchas arquitecturas.
- ***Desventajas:***
    - Se puede llegar a modificar el PR inadvertidamente. Es el peor problema. Se pierde el control y puede llevar a errores.
    - El acceso al dato es más lento por la indirección a resolver cada vez que se invoque.
    - Se pueden generar alias cuando dos variables o referencias diferentes se asignen a la misma dirección de memoria.
    - La generación de estos alias afectan la legibilidad y por lo tanto la confiabilidad, se hace muy difícil la verificación de programas y depuración de errores.

Modo IN/OUT por Nombre:

- El parámetro formal es sustituido "textualmente" por una expresión del parámetro real, más un puntero al entorno del parámetro real (expresión textual, entorno real). Se utiliza una estructura aparte que resuelve esto.
- Se establece la ligadura entre parámetro formal y parámetro real en el momento de la invocación, pero la "ligadura de valor" se difiere hasta el momento en que se lo utiliza (la dirección se resuelve en ejecución). Distinto semánticamente a por referencia.
- Para implementar este modo se usan Thunks:
    - Unidad pequeña de código (función) que encapsula y representa a una expresión que pospone su evaluación hasta que sea necesario.
    - Cada aparición del parámetro formal se reemplaza en el cuerpo de la unidad llamada por una invocación a un thunks, en el momento de la ejecución activará al procedimiento que evaluará el parámetro real en el ambiente apropiado.

- Ventajas:
    - Es un método que extiende el alcance del parámetro real, pero esto mismo puede llevar a errores.
    - Posee evaluación diferida al ejecutar.
- Desventajas:
    - Es más lento ya que debe evaluarse cada vez que se lo usa.
    - Es difícil de implementar y genera soluciones confusas para el lector y el escritor.

Pasaje de Parámetros en algunos lenguajes

C

- Por valor, (si se necesita por referencia se usan punteros).
- Permite pasaje por valor constante, agregándole const.

Pascal

- Por valor (por defecto).
- Por referencia (opcional: var).

C++

- Similar a C.
- Más pasaje por referencia.

Java

- Sólo copia de valor. Pero como las variables de tipos no primitivos son todas referencias a variables anónimas en el HEAP, el paso por valor de una de estas variables constituye en realidad un paso por referencia de la variable.

PHP

- Por valor, (predeterminado).
- Por referencia (&).

Python

- Envía objetos que pueden ser “inmutables” o “mutables” (objeto que pueden ser o no modificados). Si es inmutable actuará como por valor y, si es mutable, ejemplo: listas, no se hace una copia, sino que se trabaja sobre él

RUBY

- Por valor. Pero al igual que Python si se pasa es un objeto “mutable” (objeto que puede ser modificado), no se hace una copia sino que se trabaja sobre él.

ADA

- Por copia modo IN (por defecto).

- Por resultado modo OUT.
- IN-OUT.
- Para los tipos de datos primitivos indica que es por valor-resultado.
- Para los tipos no primitivos, y datos compuestos (arreglos, registros) se hace por referencia.
- En las funciones solo se permite el paso por copia de valor, lo cual evita parcialmente la posibilidad de efectos colaterales.

Subprogramas

- No lo incorporan todos los lenguajes.
- Debe determinarse cuál es el ambiente de referencia no local correcto para un subprograma que se ha invocado y que ha sido pasado como parámetro
- Algunas cosas pueden ser confusas de resolver (chequeo de tipos de subprogramas, subprogramas anidados, etc.).
- Para resolver el Ambiente de referencia para las referencias no locales dentro del cuerpo del subprograma pasado como parámetro hay diversas opciones que se implementan si se utiliza cadena dinámica:

Ligadura shallow o superficial

- El ambiente de referencia, es el del subprograma que tiene declarado el parámetro formal del subprograma.

Ligadura deep o profunda

- El ambiente es el del subprograma dónde está declarado el subprograma usado como parámetro real. Se utiliza en los lenguajes con alcance estático y estructura de bloque.

Ligadura ad hoc

- El ambiente del subprograma donde se encuentra el llamado a la unidad que tiene un parámetro subprograma. Menos fiable (Poco natural).

Clase 7

Tipos de Datos

- Un tipo de dato es un conjunto de valores y un conjunto de operaciones asociadas al tipo para manejar esos valores.

<table>
<tr><th></th><th>ELEMENTALES /PRIMITIVOS</th><th>COMPUESTOS</th></tr>
<tr><td rowspan="4">PREDEFINIDOS</td><td>ENTEROS</td><td rowspan="4">STRING</td></tr>
<tr><td>REALES</td></tr>
<tr><td>CARACTERES</td></tr>
<tr><td>BOOLEANOS</td></tr>
<tr><td rowspan="4">DEFINIDOS POR EL USUARIO</td><td rowspan="4">ENUMERADOS
(RANGOS, subrangos)</td><td>ARREGLOS</td></tr>
<tr><td>REGISTROS</td></tr>
<tr><td>LISTAS</td></tr>
<tr><td>ETC.</td></tr>
<tr><td colspan="3">DOMINIO DE UN TIPO ----> VALORES POSIBLES</td></tr>
</table>

- Cualquier lenguaje de programación está “equipado” con un conjunto de tipos predefinidos o primitivos que reflejan el comportamiento del hardware.
- Estos tipos predefinidos pueden ser a su vez:
    - **Elementales/Escalares:** Son indivisibles, no se pueden descomponer a partir de otros.
    - **Compuestos:** Como en el caso de los Strings.
- Los lenguajes permiten al programador especificar agrupaciones de objetos de datos elementales, con el fin de definir lo que denominamos como “tipo de dato definido por el usuario”.
- Las definiciones de esos tipos de datos se pueden hacer de forma recursiva, o mediante agregaciones de agregados o uniones. Esto se logra mediante la prestación de una serie de constructores que permiten definir a estos tipos.

Tipos Predefinidos

- Reflejan el comportamiento del hardware subyacente y son una abstracción de él.
- ***Ventajas:***
    - Invisibilidad de la representación, se que los tipos están y los puedo usar, y no me interesa cómo están representados.
    - Tienen Verificación estática de control de tipos.
    - Desambiguar operadores sobrecargados.
    - Control de precisión, ejemplo: short int, long int, int, double, etc.

Tipos Definidos por el Usuario

- Los lenguajes de programación permiten al programador especificar agrupaciones de objetos de datos elementales y de forma recursiva, agregaciones de agregados. Esto se logra mediante constructores.

- **Separan la especificación de la implementación.**
- **Se definen los tipos que el problema necesita.**
- ***Permiten:***
    - Instanciar objetos de las agregaciones.
    - Definir nuevos tipos de dichas agregaciones.
    - Chequeo de consistencia.
- ***Ventajas:***
    - **Legibilidad**: Elección apropiada de nuevos nombres.
    - **Estructura Jerárquica de las definiciones de tipos**: Proceso de refinamiento.
    - **Factorización**: Se usan la cantidad de veces necesarias.
    - **Modificabilidad**: Solo se cambia en la definición.
    - La instanciación de los objetos en un tipo dado implica una descripción abstracta de sus valores. Los detalles de la implementación solo quedan en la definición del tipo.

Constructores

- Son los que permiten definir tipos de datos definidos por el usuario.

Producto Cartesiano

- El producto cartesiano de n conjuntos de tipos variados. Permite producir registros (Pascal) o struct (C).

Correspondencia Finita

- Es una función de un conjunto finito de valores de un tipo de dominio DT en valores de un tipo de dominio RT.
- DT -> tipo de dominio (int por ej).
- RT -> resultado del dominio (acceso a través del índice)
- Son las listas indexadas, vectores, arreglos, matrices, etc.

Unión y Unión Discriminada

- La unión/unión discriminada de uno o más tipos, es la disyunción de los tipos dados. Se trata de campos mutuamente excluyentes (uso uno o el otro), no pueden estar al mismo tiempo con valores.
- Permite manipular diferentes tipos en distintos momentos de la ejecución.
- Chequeo dinámico. No se puede asegurar en compilación qué tipo o variante adquiere una variable.
- La unión discriminada agrega un descriptor (enumerativo) que me permite saber con quien estoy trabajando y acceder correctamente a lo que tengo que acceder, ya que nos dice cual de los campos posee valor. Básicamente manipulo el elemento según el valor del discriminante, es una mejora de la unión que brinda una mayor seguridad. Este discriminante igualmente puede omitirse.

Recursión

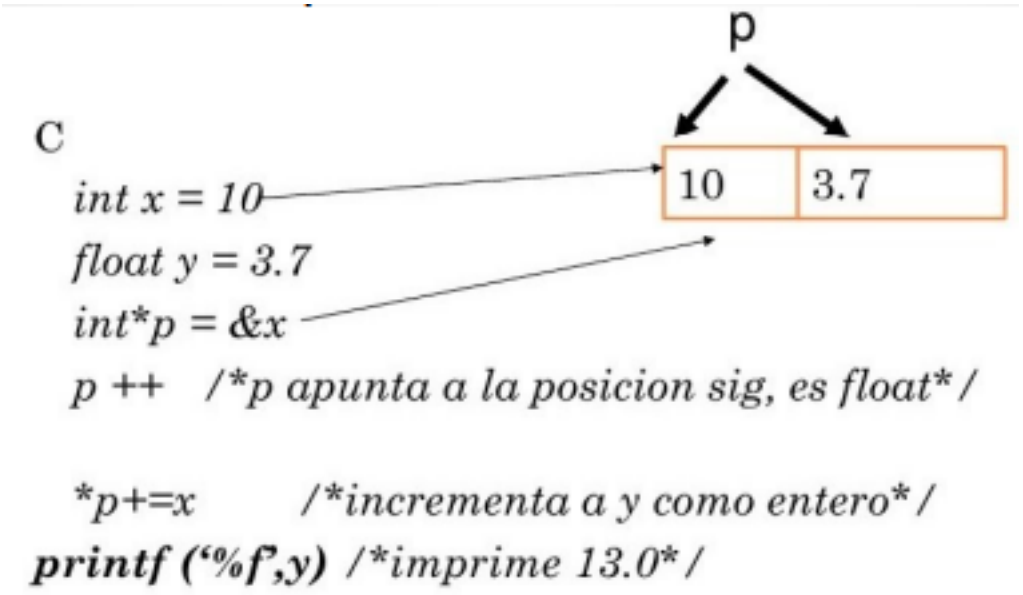
- Un tipo de dato recursivo T se define como una estructura que puede contener componentes de tipo T. Ejemplos: árboles o listas de Pascal.
- Define datos agrupados:
    - Cuyo tamaño puede crecer arbitrariamente.
    - Cuya estructura puede ser arbitrariamente compleja.
- Los lenguajes soportan la implementación de tipos de datos recursivos a través de los punteros.

Punteros

- Un puntero es una referencia a un objeto.
- Una variable puntero es una variable cuyo R-Valor es una referencia a un objeto.
- Son un mecanismo poderoso para definir estructuras recursivas pero de bajo nivel.
- Al tener un acceso a bajo nivel, los punteros están cerca de la máquina y pueden hacer inseguros a los programas.
- Permiten tener estructuras de tamaño arbitrario con un número de ítems no determinado.
- Amplían el rango de celdas de memoria y amplían los tipos de datos que pueden tener las estructuras.

Inseguridad de los Punteros

Violación de Tipos



Referencias sueltas - Referencias dangling

- Una referencia suelta o dangling es un puntero que contiene una dirección de una variable dinámica que fue desalocada, si luego se usa el puntero producirá error.

Punteros no inicializados

- Peligro de acceso descontrolado a posiciones de memoria.
- Verificación dinámica de la inicialización.
- Solución -> valor especial nulo:
    - nil en Pascal.
    - void en C/C++.
    - null en ADA, Python.

Punteros y uniones discriminadas

- Puede permitir accesos a cosas indebidas.
- Java lo soluciona eliminando la noción de puntero explícito completamente.

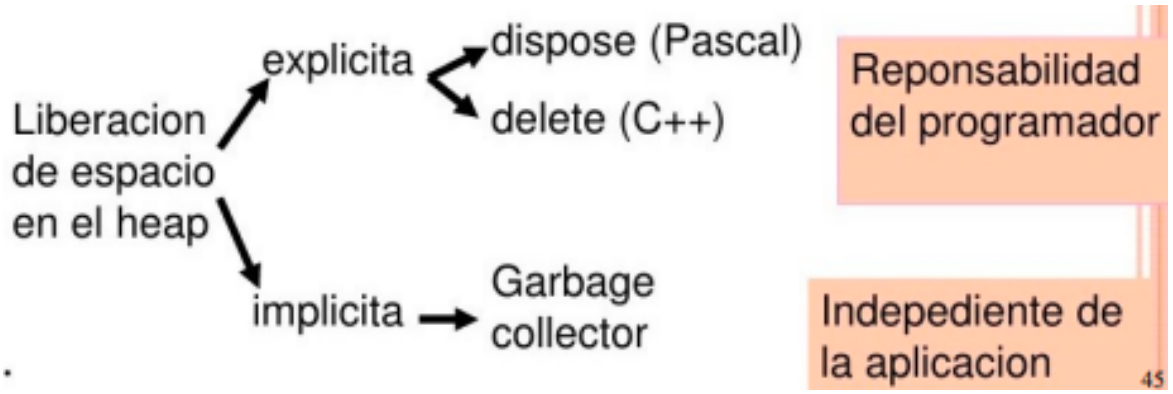
Alias

- Si 2 o más punteros comparten alias, la modificación que haga uno se verá también reflejado en los demás.

Liberación de Memoria - Objetos Perdidos

- Si los objetos en el heap dejan de ser accesibles, esa memoria podría liberarse.
- Un objeto se dice accesible si alguna variable en la pila lo apunta directa o indirectamente.
- Un objeto es basura si no es accesible .

Manejo de Memoria



Explícita

- El reconocimiento de la basura recae en el programador, quien notifica al sistema cuando un objeto ya no se usa.
- No garantiza que no haya otro puntero que apunte a esa dirección.
- Puede generar referencias sueltas.
- Ejemplos: dispose de Pascal.

Implícita

- El sistema, durante la ejecución tomará la decisión de descubrir la basura por medio de un algoritmo de recolección de basura (garbage collector).
- Importante para los lenguajes que utilicen frecuentemente variables dinámicas (LISP, Python).
- Se ejecuta durante el procesamiento de las aplicaciones.
- Debe ser muy eficiente.