Rutinas

1. ¿Qué es una rutina?

Conjunto de sentencias que representan una acción abstracta. Representan un tipo unidad de programa. Amplían el lenguaje. A nivel de diseño, permite definir una operación creada por el usuario a semejanza de las operaciones primarias integradas en el lenguaje. El ejemplo más utilizado por excelencia son los subprogramas. Están basadas en un esquema de call/return. Para ejecutarlas hay que invocarlas (call) al finalizar (return) devuelven el control a quién las llamó.

2. ¿Qué es un subprograma?

Es un tipo de rutina. Su definición está compuesta por un encabezado y un cuerpo. El encabezado contiene el nombre y la lista de parámetros. El cuerpo, en cambio, contiene toda la implementación, es decir, la definición de las variables locales y las sentencias que forman su código. Su uso radica en poder definir bloques de código con comportamiento específico(en esta etapa importa el cómo lo hace y no el qué hace) que luego se invocarán por su nombre y generarán una instancia de dicha unidad(aca pasa a importar el qué hace y no cómo lo hace). Con una definición pueden crearse muchas activaciones: la definición no es más que un molde. Un subprograma es la implementación de una acción abstracta y su invocación representa el uso de dicha abstracción por lo que codificar un subprograma es como si hubiéramos incorporado una nueva sentencia a nuestro lenguaje. Hay dos tipos:

- **Procedimientos**: definen nuevas sentencias creadas por el usuario. Los resultados quedan en variables no locales o en parámetros que cambian su valor. En FORTRAN se los llama subrutinas, pero en general son conocidos como procedimientos.
- **Funciones**: definen un nuevo operador. Son similares a las funciones matemáticas ya que solo producen un valor y deberían tener o producir efectos colaterales. Solo en los lenguajes funcionales las funciones no tienen efectos colaterales(transparencia referencial). Se las invoca dentro de expresiones con su lista de parámetros reales. El valor que produce reemplaza a la invocación dentro de la expresión.

3. ¿Qué es un parámetro?

Es una forma de compartir datos entre diferentes unidades. Esto puede darse a través de dos formas:

- **Acceso al ambiente no local**: puede ser a través del ambiente común explícito(el COMMON de FORTRAN) o a través de un ambiente común no explícito(utilizando reglas de alcance estático o dinámico).
- **Uso de parámetros(propiamente dicho)**: el pasaje de parámetros es la forma más flexible y permite la transferencia de diferentes datos en cada llamada. Proporciona ventajas en legibilidad y modificabilidad. Nos permiten compartir los datos en forma abstracta ya que indican con precisión qué es exactamente lo que se comparte.

4. Parámetro real vs parámetro formal

- **Parámetros reales**: parámetros que se codifican en la invocación del subprograma. Puede ser local a la unidad llamadora; ser a su vez un parámetro formal de ella; un dato no local pero visible en dicha unidad; o una expresión.
- **Parámetros formales**: parámetros declarados en la especificación del subprograma. Contiene los nombres y los tipos de los datos compartidos. En general son similares a variables locales.

Por si acaso no hubiera quedado claro: el parámetro real es ese que vos mandás cuando invocás la función; el parámetro formal es ese que está en la declaración de la función.

5. Evaluación de los parámetros reales y ligadura con los parámetros formales

- **Evaluación**: en general, en el momento de la invocación, primero se evalúan los parámetros reales y luego se hace la ligadura antes de transferir el control a la unidad llamada.
- **Ligadura**: puede ser posicional si se correspondieran con la posición que ocupan en la lista; o por palabra clave o nombre si se correspondieran con el nombre, por lo que, consecuentemente, pudieran estar colocados indistintamente en la lista.

6. Tipos de parámetros en los que se pasa un dato

- Modo IN: el parámetro formal recibe el dato desde el parámetro real. Dos tipos:
    - **Por valor**: el valor del parámetro real se usa para inicializar el correspondiente parámetro real al invocar la unidad. Se transfiere el dato real. En este caso el parámetro formal actúa como una variable local de la unidad llamada. Desventaja: consume el tiempo para hacer la copia y el almacenamiento para duplicar el dato. Ventaja: protege los datos de la unidad llamadora y el parámetro real no se modifica.
    - **Por valor constante**: no indica si se realiza o no la copia, lo que establece es que la implementación debe verificar que el parámetro real no sea modificado. Desventaja: requiere realizar mas trabajo para implementar los controles. Ventaja: idem por valor.
- Modo OUT: el parámetro formal envía el dato al parámetro real. Dos tipos:
    - **Por Resultado**: el valor del parámetro formal se copia al parámetro real al terminar de ejecutarse la unidad llamada. El parámetro formal es una variable local, sin valor inicial. Desventaja: consume tiempo, espacio y si se repiten los parámetros reales los resultados pueden ser diferentes. Además, se debe tener en cuenta el momento en que se evalúa el parámetro real. Ventaja: idem modo IN por valor.
    - **Por resultado de funciones**: El resultado de una función puede devolverse con el return o en el nombre de la función(ultimo valor asignado) que se considera como una variable local, dependiendo el lenguaje. Dicho resultado reemplaza la invocación en la expresión que contiene el llamado.
- Modo IN/OUT: el parámetro formal recibe el dato del parámetro real y el parámetro formal le envía el dato al parámetro real. Tres tipos:
    - **Valor-Resultado**: Copia a la entrada y a la salida de la activación de la unidad llamadora. 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. Cada referencia al parámetro formal es una referencia local. Tiene las desventajas y las ventajas de ambos.
    - **Por Referencia**: Se transfiere la dirección del parámetro real al parámetro formal. El parámetro formal será una variable local a la unidad llamadora que contiene la dirección en el ambiente no local. Esto significa que cualquier cambio que se realice en el parámetro formal dentro del cuerpo del subprograma quedará registrado en el parámetro real. El parámetro real es compartido por la unidad llamada. Desventajas: el acceso al dato es mas lento por la indirección, se puede modificar el parámetro real inadvertidamente, se pueden general alias(con lo que ello implica). Ventaja: eficiente en tiempo y espacio. No se realizan copias del dato.
    - **Por Nombre**: el parámetro formal es sustituido textualmente por el parámetro real. Es decir, 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. El objetivo es otorgar mayor flexibilidad a través de esta evolución de valor diferida. Si el dato a compartir fuera un único valor, se comportaría exactamente igual que el pasaje por referencia; si fuera una constante sería equivalente a por valor; si fuera un elemento de un arreglo podría cambiar el suscripto entre las distintas referencias; por último, si fuera una expresión se evaluaría cada vez. Es un método mas flexible pero mas 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.

7. Pasaje de parámetros en funciones(siempre y cuando sea solo un dato)

Las funciones no deberían producir efectos colaterales. Esto implica que no deberían alterar el valor de ningún dato(local ni no local), solo producir un resultado por lo que el tipo de parámetro debería ser siempre tipo IN. Es deseable, además, la ortogonalidad, es decir que los resultados deberían poder ser siempre de cualquier tipo.

8. Pasar un subprograma como parámetro

En algunas situaciones es conveniente poder manejar como parámetros los nombres de los subprogramas. Debe determinarse cuál es el ambiente de referencia no local correcto para un subprograma que se ha invocado y qué ha sido pasado como parámetro. Hay tres opciones:

- **Ligadura shadow**: el ambiente de referencia es el del subprograma que tiene el parámetro formal del subprograma. No es apropiada para lenguajes con estructuras de bloques ya que se puede acceder a ambientes que estáticamente no le son visibles.
- **Ligadura deep**: el ambiente es el del subprograma donde esta declarado el subprograma usado como parámetro real. Se utiliza en los lenguajes con alcance estático y estructura de bloque. Se necesita que, en el momento de la invocación a la unidad con parámetro subprograma además del puntero al código, se indique cuál será su ambiente de referencia, es decir un puntero al lugar de la cadena estática correspondiente.
- El ambiente del subprograma donde se encuentra el llamado a la unidad que tiene un parámetro subprograma. Poco natural.