Clase 4

Unidades de programa

Los lenguajes de programación permiten que un programa esté compuesto por rutinas que pueden ser *procedimientos* o *funciones.*

Es una 5-tupla **<NOMBRE, ALCANCE, TIPO, L-VALUE, R-VALUE>**

- **Nombre**: cadena usada para invocar a la rutina, introducido en su declaración y usado para invocarlas.
- **Alcance**: visibilidad de la rutina en diferentes partes del programa. Por lo general, se definen en un módulo separado del resto del programa y el alcance determina en qué partes del programa se puede llamar.

  En algunos lenguajes como C, Pascal o Ada hay una diferencia entre definición y declaración. La **declaración** especifica el tipo de datos que devuelve y los parámetros de entrada mientras que la **definición** es el cuerpo de la rutina.

  En general, la declaración de una rutina puede aparecer en cualquier parte en la que se necesite llamarla, pero la definición debe aparecer una vez. Se debe a que la definición proporciona el cuerpo real de la rutina y puede ocupar espacio en la memoria del programa. Al permitir que la definición se coloque en un archivo separado, el compilador puede crear un archivo de objeto separado para la rutina y vincularlo con el resto del programa solo cuando sea necesario.

  Si el lenguaje distingue entre declaración y definición, permite manejar esquemas recursivos.

  La representación de una rutina durante su ejecución se llama instancia de rutina el cual está compuesto por:

  - Segmento del código: instrucciones.
  - Registro de activación: son contenidos cambiantes, tiene toda la información necesaria para ejecutar la rutina y otras cosas como datos asociados a las variables locales.

  La **cabecera de una rutina** define el nombre de la rutina, los tipos de su rutina y el tipo de valor de retorno que tiene.
- **Tipo**: el encabezado de la rutina define el tipo de los parámetros y el tipo de valor de retorno, si hay.

  La signatura de una rutina es la especificación de su tipo, incluyendo los parámetros de entrada y valor de retorno. 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-Value**: lugar de memoria donde se almacena el cuerpo de la rutina.
- **R-Value**: la llamada a la rutina causa la ejecución su código, eso constituye su r-valor. Puede ser estático (el caso más usual) o dinámico (variables de tipo rutina)
    - Las **rutinas estáticas** se implementan usando una definición estática de la rutina del código fuente del programa. Cuando se llama a la rutina estática, se ejecuta el código correspondiente y se devuelve un valor que se puede usar en el programa.
    - Las **rutinas dinámicas** son variables que contienen una referencia a una rutina, en lugar de contener directamente el código de la rutina. Se implementan a través de punteros a rutinas, o sea que la variable de la rutina dinámica contiene la dirección de memoria donde se encuentra el código de la rutina en tiempo de ejecución. Cuando se llama a una rutina dinámica, se sigue el puntero a la dirección de memoria de la rutina y se ejecuta el código correspondiente. Por ejemplo C implementa este tipo de rutinas.
    - **Comunicación entre rutinas** se puede dar en el ambiente no local o mediante parámetros, los cuales son diferentes datos en cada llamado y brindan legibilidad y modificabilidad.

      Los parámetros se clasifican en

        - Formales: aparecen en la definición de la rutina.
        - Reales: aparecen en la invocación de la rutina.

      El binding entre parámetros formales y reales es el proceso de asociar los argumentos formales y reales. Hay distintos métodos para llevar a cabo este binding

        - Método posicional: los parámetros se ligan uno a uno en orden de posición. O sea que el primer parámetro real se liga al primer parámetro formal y así sucesivamente. En este método es importante conocer las posiciones de los parámetros.
        - Combinación método posicional + asignación por valor por defecto, en el ejemplo dado de C++, la función "distancia" tiene dos parámetros formales, pero se ha definido que si no se proporcionan valores para estos parámetros, se asignarán por defecto los valores cero. Entonces, cuando se llama a la función sin argumentos, se ligarán los valores por defecto a los parámetros formales, mientras que si se proporciona un argumento, se ligará al primer parámetro formal y el segundo se ligará al valor por defecto.

          ```
          int distancia (int a = 0, int b = 0) parámetros formales


          distancia() --> distancia (0,0)
          distancia(10) ..> distancia (10,0)
          ```

- Método por nombres: los parámetros formales y reales se ligan por su nombre y no por su posición en la lista. En este método hay que conocer los nombres de los parámetros formales para ligar correctamente los reales.

```
Si X, Y y Z son de tipo T1, T2 y T3 dado procedure Ejem (A:T1; B: T2:= W; C:T3);
Ejem(X,Y,Z) hace una asociación posicional
Ejem(X,C => Z) X se liga a A por posición, B toma el valor por defecto de W y
                C se liga a Z por nombre
Ejem(C=>Z,A=>X,B=>Y) se ligan todos por nombre
```

En general los parámetros pasados a una rutina serán datos de entrada y en otros casos pueden ser rutinas pero esto es solo permitido por lenguajes que manipula a las rutinas como objetos de primera clase y que pueden ser
asignados a variables como C

Rutinas genéricas

Las rutinas genéricas se refieren a funciones o métodos que pueden ser utilizados con diferentes tipos de datos. Ada permite rutinas genéricas, en las rutinas generéricas el binding entre parámetros actuales y formales se hace en la compilación y para cada tipo de parámetro, se generará una nueva instancia de rutina.

Rutinas recursivas

Las rutinas pueden ser activadas recursivamente, o sea que una unidad puede llamarse a ella misma. En este caso todas las unidades estarían compuestas por el mismo segmento de código pero distinto registor de activación
Ante la presencia de recursión el binding entre el registro de activación y el segmento del código es dinámico

Representación en ejecución

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

Instancia de la unidad: representación de la rutina en ejecución.

- Segmento de código: instrucciones de la unidad se almacenan en la memoria de instrucción, es un contenido fijo.
- Registro de activación: datos locales de la unidad se almacena en la memoria de datos, es un contenido cambiante.

Procesador abstracto, SIMPLESEM

- **IP (instruction pointer)**: registro que apunta a la dirección de la próxima instrucción a ejecutarse.
- **Memoria de código**: lugar donde se almacenan las instrucciones del programa.

- **Memoria de datos**: lugar donde se almacenan los datos usados en el programa.
- **Segmento de código**: parte de la memoria del código que contiene las instrucciones que se están ejecutando actualmente.
- **Registro de activación**: estructura de datos que se usa para mantener información sobre la función o procedimiento en ejecución actualmente.

SIMPLESEN permite comprender qué efecto causan las instrucciones del lenguaje al ejecutarse el cual tiene una semántica intuitiva. Describe la semántica del lenguaje de programación con reglas de cada constructor del lenguaje traduciéndolo en una secuencia de caracteres equivalentes del procesador abstracto.

- Memoria de código: C(y) valor almacenado en la y-ésima celda de la memoria de código. Comienza en 0.
- Memoria de datos: D(y) valor almacenado en la y-ésima celda de la memoria de datos. Empieza en cero y representa el l-valor, D(y) o C(y) su r-valor.
- IP (instruction pointer): puntero a la instrucción ejecutándose. Se inicializa en 0 en cada ejecución y se actualiza cuando se ejecuta cada acción. IP indica las direcciones de C.

El proceso de ejecución entonces es

1. Obtener la instrucción actual para ser ejecutada C[IP].
2. Incrementar IP.
3. Ejecutar la instrucción actual.

INSTRUCCIONES

- SET setea valores en la memoria de datos, *set target, source.* Copia el valor representado por el source en la dirección representada por target.

  set 10,D[20] copia el valor almacenado en la posición 20 en la posición 10.

- E/S: read y write permiten la comunicación con el exterior.

  set 15,read el valor leído se almacenará en la dirección 15.

  set write, D[50] se transfiere el valor almacenado en la posición 50.

- Combinación de expresiones:

  set 99, D[15] + D[33]*D[4] copia en la posición 99 el resultado de multiplicar el contenido de la dirección 4 con el contenido de la dirección 33 y sumarle el contenido de la dirección 15.

Algunas instrucciones pueden llegar a modificar este orden secuencial por lo tanto esto SIMPLESEM lo tiene que manipular con dos instrucciones JUMP y JUMPT

- JUMP: bifurcación condicional

jump 47 la próxima instrucción a ejecutarse será la que esté almacenada en la dirección C[47]

- JUMPT: bifurcación condicional, bifurca si la expresión se evalúa como verdadera

  jumpt 47, D[13] > D[8] bifurca si el valor almacenado en la celda 13 es mayor que el almacenado en la celda 8.

- Direccionamiento indirecto:

  set D[10],D[20]

  jump D[30]

  IP= 5 posición 5 en C

ELEMENTOS EN EJECUCIÓN

- **Punto de retorno**: indicar la ubicación en la que se debe reanudar la ejecución de un programa después de haberse detenido temporalmente para ejecutar otra tarea.
- Ambiente de referencia: se refiere al conjunto de variables y objetos que están disponibles para una unidad de programa, como una función o método, en un momento dado.
  - **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

Los lenguajes pueden clasificarse según su estructura a la hora de ejecutar rutinas

- Estático (espacio fijo): estos lenguajes garantizan que la memoria requerida para cualquier programa puede ser evaluada antes de que el programa se ejecute. Toda la memoria necesitada será alocada antes que se ejecute el programa. Consecuentemente, los lenguajes estáticos no permitirán la recursividad porque entonces requerirían un número arbitrario de unidades.
- Basado en pila: este tipo de lenguajes son más demandantes en términos de memoria los cuales no pueden ser calculados en la compilación. Sin embargo el uso de memoria es predecible y sigue la disciplina LIFO (last in first out)
  o sea que el último registro alocado será el primero en desalcoarse. No necesitan una pila literalmente si no que se usa si no que es una representación. Se seguirán usando mismos espacios de memoria
- Dinámico: dedicado a lenguajes con impredecible uso de memoria. Los datos son alocados dinámicamente solo cuando se los necesita durante la ejecución y no pueden modelizarse con una pila, el programador puede crear objetos de datos en

cualquier punto arbitrario durante la ejecución y los datos se alocan en la zona de memoria *heap.*

C1

Lenguaje de programación que tiene un conjunto limitado de características. Se compone de sentencias simples que se pueden usar para definir las instrucciones del programa. Incluye tipos simples para definir variables y constantes.

No incluye funciones entonces todas las operaciones se tienen que hacer en la rutina principal main(). Tampoco admite datos dinámicos.

Admite la declaración y uso de variables de diferentes tipos como enteros, reales, arreglos, estructuras e incluye operaciones de E/S usando las funciones get y print.

C2: C1 + rutinas internas

Lenguaje basado en C1 y agrega la capacidad de definir rutinas internas dentro de la rutina principal. En C2 un programa se compone de datos globales, declaraciones de rutinas y la rutina principal.

Ademas de los tipos admitidos por C1, C2 también admite la definición de tipos de datos personalizados y estructuras más complejas.

- Rutinas internas: subrutinas definidas en la rutina principal del programa pero tienen algunas limitaciones a comparación de la rutina principal. Las rutinas internas en C2 deben ser disjuntas, lo que significa que no pueden estar anidadas una dentro de la otra y no pueden ser recursivas.
- Además la zona de datos tendrá datos locales y punto de retorno.

C2’

En C2’ la forma en que se gestionan las variables locales y globales es diferente a la de C2. En C2 el compilador asigna direcciones absoluta a las variables locales y globales durante la ejecución, o sea que la memoria de estas variables se reserva de antemano y no puede ser modificada en la ejecución.

En cambio en C2’, el compilador no puede ligar las variables a direcciones absolutas durante la compilación. En su lugar, el proceso de vinculación se realiza por un programa llamado “linkeditor”, que se encarga de combinar diferentes módulos que forman un programa final y de vincular la información faltante, como las direcciones de las variables locales y globales, permitiendo más flexibilidad en la gestión de memoria y recursos.

En cuanto a la diferencia semántica entre C2 y C2', se puede decir que son muy similares y comparten muchas características. La diferencia principal es la forma en que se gestionan las variables locales y globales.

C3: C2 + recursión y valor de retorno

C3 (esquema basado en una pila) permite la recursión (directa o indirecta) y devolver valores de funciones. El compilador asigna un lugar para cada variable y existirán los registros de activación (RA) que son estructuras de datos usados para almacenar temporalmente información sobre una llamada a una rutina, como variables locales, parámetros o valores de retorno.

El tamaño del RA es fijo y conocido pero no se sabe cuántas instancias se van a necesitar a la hora de ejecutarlo. Por ejemplo si A llama a B el cual llama a C, el RA para funciones están alocados en el orden A, B, y C. Cuando C vuelve a B, el RA de C es descartado, luego se descarta el RA de B cuando vuelve a A.

Para manejar la recursividad, se debe guardar en el RA

- Return point: dirección de la siguiente instrucción que debe ejecutarse después de que se complete la llamada a la función actual y se desaloje su RA.
- Dirección base del RA: cuando se realiza una llamada recursiva, se crea una nueva instancia de la rutina con su propio registro de activación. Este registro de activación debe estar ubicado en la pila, por lo que es necesario saber la dirección base del registro de activación de la instancia actual para poder acceder a las variables locales y los parámetros de la instancia actual.

  Al guardar la dirección base del registro de activación en el registro de activación de la instancia actual, se puede acceder a la información de la instancia actual y, al mismo tiempo, permitir la creación de nuevas instancias de la rutina sin perder la información de las instancias anteriores.
- Valor de retorno (en caso de ser necesario).

Por otra parte, la asignación dinámica se usa cuando el tamaño de los registros de activación no se pueden determinar estáticamente en la compilación, en este caso será necesario para almacenar RA a medida que se instancien nuevas rutinas. También se usa para almacenar los valores de retorno de las funciones.

Cuando una rutina se llama a sí misma o a otra rutina, se agrega un nuevo RA a la pila. Cuando se completa la instancia actual de la rutina, se desaloca su RA para liberar espacio en la pila. Si la rutina tiene valor de retorno, ese valor debe guardarse en el RA de la rutina llamante antes de desalojar la rutina llamada.

Para manejar la alocación dinámica se necesitan nuevos elementos

- Valor de retorno: se almacena en el RA de la rutina llamante antes de que se desaloque el RA de la rutina llamada.
- Link dinámico: un puntero que apunta a la dirección base del RA de la rutina llamante. Se usa para acceder a los valores de variables locales de la rutina llamante desde la rutina llamada.
- Current: la dirección base del RA de la unidad que se esté ejecutando actualmente. Para hacer posible la recursividad, se usa la celda de la dirección 0 de D[X] para

almacenar esta dirección base.

- Free: próxima dirección libre en la pila. Se actualiza cada vez que se agrega o desaloja un RA de la pila.

Cadena dinámica: cadena de link dinámicos originada en la secuencia de registros de activación activos.

C4: estructura de bloque

La estructura de bloque se refiere a un conjunto de declaraciones de variables y/o instrucciones que están encerradas dentro de { }.

Es usado para controlar el alcance de las variables. Las que son declaradas dentro de un bloque, solo pueden ser accedidas desde dentro del mismo bloque y los anidados a él. Al salir del bloque, las variables declaradas dentro de él se destruyen automáticamente entonces su tiempo de vida se limita a la duración del bloque.

La estructura de bloque también se utiliza para dividir un programa en unidades más pequeñas y manejables. Al agrupar declaraciones de variables y/o instrucciones relacionadas en bloques, se puede aumentar la legibilidad del código y facilitar el mantenimiento y la depuración del programa.

C4’

La novedad es la capacidad de declarar variables locales dentro de bloques de código, permitiendo mayor modularidad. A diferencia de C4 donde las variables locales deben declararse al principio de una función.

Un bloque tiene forma de una sentencia compuesta: {<lista de declaraciones>;<lista de sentencias>}

```
{
### <lista de declaraciones>
    int a = 5;
    float b = 3.14;
    char c = 'c';
## <lista de sentencias>
    printf("El valor de a es: %d\n", a);
    printf("El valor de b es: %f\n", b);
    printf("El valor de c es: %c\n", c);
## bloque 2
    {
        int a = 10;
        float b = 2.71;

        printf("El valor de a dentro del bloque interno es: %d\n", a);
        printf("El valor de b dentro del bloque interno es: %f\n", b);
    }

    printf("El valor de a fuera del bloque interno es: %d\n", a);
    printf("El valor de b fuera del bloque interno es: %f\n", b);
}
# si fuera C4 todas las declaraciones tendrían que estar antes del bloque 1
```

Si se declara una nueva variable con el mismo nombre en un bloque anidado, la nueva declaración "enmascara" la declaración externa del mismo nombre.

C4”

La principal novedad de C4’’ es la posibilidad de definir rutinas dentro de otras rutinas, conocido como anidamiento de rutinas, permitiendo que el código sea más modular.

Además, la estructura de bloque de C4’’ sigue controlando el alcance y tiempo de vida de las variables.

Los bloques podrán ser

- Disjuntos: aquellos que no tienen porciones comúnes, o sea que están separados por otras sentencias que no pertenecen al bloque.

```
int a = 5;
{
  printf("%d", a);
}
printf("%d", a + 10);
```

- Anidados: están completamente contenidos en otro bloque

```
{
  int a = 5;
  {
    int b = 10;
    printf("%d", a + b);
  }
}
```

C5: datos más dinámicos

C5 permite la creación de estructuras más dinámicas donde el almacenamiento de datos no es conocido en tiempo de compilación si no en tiempo de ejecución. Lográndose mediante el uso de punteros.

Gracias a los punteros se pueden implementar estructuras de datos dinámicas como listas enlazadas, árboles o grafos, los cuales se construyen en tiempo de ejecución.

Además C5 permite el uso de memoria dinámica mediante las funciones de reserva y liberación de memoria.

C5’

Agrega soporte para el manejo de arreglos dinámicos (aquellos cuyo tamaño no se conoce en tiempo de compilación).

Usa un descriptor de arreglo que es una estructura que tiene una celda para almacenar un puntero al área de almacenamiento para el arreglo y celdas para indicar el mínimo y máximo de la dimensión física del arreglo.

Para manejar el arreglo dinámico en tiempo de ejecución, el registro de activación se asigna en varios pasos.

1. Se reserva el almacenamiento para los datos de tamaño conocido en tiempo de compilación y par los descriptores de arreglos dinámicos.
2. Cuando se declara la variable dinámica, se calculan las dimensiones en los descriptores para incluir el espacio necesario para la variable dinámica.
3. Se encuentra un puntero del descriptor que apunta a la dirección del área de almacenamiento asignada para el arreglo dinámico. De esta manera, se puede acceder a los elementos del arreglo dinámico durante la ejecución del programa.

C5”

Se permite la asignación dinámica en tiempo de ejecución mediante la instrucción de alocación explícita.

La alocación de memoria se realiza en un área llamada "heap" y se puede usar para almacenar variables cuyo tamaño no es conocido en tiempo de compilación, como los arreglos dinámicos. Para manejar el almacenamiento dinámico se utilizan punteros que apuntan a la dirección de inicio del bloque de memoria alocado.

La principal diferencia entre C5' y C5'' es que en C5'' se permite la asignación de memoria explícita durante la ejecución del programa para cualquier tipo de variable, mientras que en C5' solo se permitía para arreglos dinámicos.

En C5', el tamaño de los datos dinámicos como los arreglos dinámicos, se determina en tiempo de compilación y se maneja a través de descriptores. En C5'', se pueden asignar dinámicamente los datos en tiempo de ejecución a través de instrucciones de alocación, lo que permite manejar cualquier tipo de variable con tamaño dinámico.

C6: lenguajes dinámicos

C6 se refiere a los lenguajes de programación dinámicos, que son aquellos que tienen más reglas dinámicas que estáticas. Por lo general, estos lenguajes utilizan un tipado dinámico y reglas de alcance dinámicas, lo que significa que las ligaduras correspondientes se llevan a cabo en tiempo de ejecución y no en tiempo de compilación.

En un lenguaje dinámico, el tipo de una variable se determina en tiempo de ejecución en lugar de tiempo de compilación. Esto significa que las variables pueden cambiar de tipo durante la ejecución del programa, lo que hace que estos lenguajes sean más flexibles pero también más propensos a errores.

Las reglas de alcance dinámicas significan que las variables pueden ser accesibles en diferentes partes del programa en diferentes momentos. Esto permite una mayor

flexibilidad en la programación pero también puede conducir a errores si no se manejan adecuadamente.

Variable	Tipo de variable	Alcance
C1	Estática	Estático
C2	Estática	Estático
C3	Semi estática o Automática	Alcance limitado
C4	Semi estática o Automática	Alcance limitado
C5’	Semi dinámica o Dinámica	Dinámico
C6	Dinámica	Dinámico

TIPO DE VARIABLE

- Estática: valor asignado una sola vez y permanece constante durante toda la ejecución..
- Semi estática/ Automática: valor puede cambiar en la ejecución del programa pero el alcance el limitado a una función, bloque de código o un módulo. Se inicializan una sola vez y mantienen su valor entre llamadas sucesivas a una función.
- Semi dinámica/ Dinámica: valor puede cambiar durante la ejecución del programa y el alcance puede ser global o local. Pueden ser inicializadas varias veces durante la ejecución del programa.

ALCANCE

- Estático: definido en tiempo de compilación y son visibles en todo el bloque donde se declara y sus sub-bloques.
- Limitado: la variable es solo visible dentro del bloque declarado y no es accesible por fuera ni por sus sub-bloques.
- Dinámico: definido en tiempo de ejecución y la variable es visible solo dentro el bloque donde se declara y en cualquier sub-bloque dentro del mismo bloque.

Clase 5

Esquema de ejecución: forma en la que se organiza la memoria y los procesos durante la ejecución

- Estático: dividido en C1 y C2.

  C1 no tienen bloques internos (como rutinas) y solo tienen un programa principal y en la zona de datos solo hay datos locales.

  C2 tienen bloques internos pero sin anidamientos o sea que hay rutinas pero adentro no pueden tener otras rutinas. La compilación puede ser junta o separada (se va compilando de a partes). Los datos son locales y también tiene un punto de retorno que indica la ubicación a dónde debe volver el programa luego de una rutina.

  Por cada unidad de programa el registro de activación debe tener: datos locales, punto de retorno y referencia a datos no locales.

- Basado en pila: hay una estructura de datos llamada “pila” que se usa para almacenar información temporalmente mientras de ejecuta un programa.

  C3= C2 + recursión y valor de retorno

  Las rutinas tienen la capacidad de llamarse a sí misma durante la ejecución gracias a la recursión directa (la rutina se llama a sí misma directamente) e indirecta (la función llama a otra función que a su vez puede llamar a la función llamadora).

  Además las rutinas tienen la capacidad de devolver valores que se podrán usar posteriormente.

  Funcionamiento de C3

  El tamaño del registro de activación de cada unidad es fijo y conocido. Significa que el espacio de memoria necesario para almacenar variables locales y otros datos durante la ejecución de una unidad es constante y no varía. Pero no se sabe cuántas unidades se necesitarán durante la ejecución del programa.

  El compilador puede asignar un espacio de memoria específico para cada variable local y acceder a ella mediante su desplazamiento.

  Además se establece que la dirección donde se cargará el registro de activación es estática. Esto significa que la dirección de memoria donde se almacenará el registro de activación se determina en tiempo de compilación y no cambia durante la ejecución del programa.

  Las rutinas pueden devolver valores que no deben perderse al desactivar la unidad porque una vez terminada una rutina, su registro de activación es eliminada. Una solución a esta cuestión es la asignación dinámica de memoria y la liberación.

  Asignación dinámica: asignación de memoria en tiempo de ejecución.

Link dinámico: necesario para la modularidad y llamada a otras rutinas.

Link estático: apunta al registro de activación de la unidad que estáticamente lo contiene, su secuencia se conoce como cadena estática

Algunos elementos serán necesarios para manejar la asignación dinámica, además del valor de retorno y link dinámico:

- Current: dirección base del RA que se está ejecutando actualmente.
- Free: próxima dirección libre en la pila.
- Cadena dinámica: cadena de links dinámicos originada en la secuencia de registros de activación activos.

C4

C4' permite que dentro de las sentencias compuestas aparezcan declaraciones locales.

C4'' permite la definición de una rutina dentro de otras rutinas.

C5

C5' el tamaño de los registros de activación se conoce cuando se activa la unidad y son datos semi dinámicos. Cuando se implementan los arreglos dinámicos en el momento de compilación se reserva lugar en el registro para los descriptores de los arreglos.

C5'' los datos pueden alocarse durante la ejecución y son datos dinámicos, los cuales se alocan explícitamente durante la ejecución mediante instrucciones de alocación.

C6

Estos lenguajes implementan:

- Tipado dinámico: los tipos de datos se establecen en la ejecución, permitiendo mayor flexibilidad.
- Reglas de alcance dinámicas: determinan cómo se accede a las variables en diferentes programas.

Estructura de bloques: delimitación de un conjunto de instrucciones mediante llaves para definir el alcance y tiempo de vida de las variables declaradas.

Ventajas

- Controlar alcance y tiempo de vida de variables declaradas en ese bloque.
- Dividir el programa en unidades más pequeñas.

Podrán ser disjuntos (no tienen porción común) o anidados (un bloque está contenido en otro).

Un bloque tiene forma de {<lista de declaraciones>;<lista de sentencias>}

- Dinámico

Clase 6

Las rutinas son una unidad de programa las cuales están formadas por un conjunto de sentencias que representan una acción abstracta.

Permiten definir una operación creada por el usuario e integrarla a las primarias de lenguaje, por lo tanto sirven para ampliarlo, dándole modularidad, claridad y buen diseño. Por ejemplo un subprograma.

Los subprogramas se clasifican en

- Procedimientos: construcciones que permite dar nombre a un conjunto de sentencias y declaraciones asociadas que se usarán para resolver un sub problema dado. Dan una solución de código corta, comprensible y modificable, permiten crear nuevas acciones.

  Su desventaja es que no pueden recibir ni devolver un valor si no que los resultados se pueden reflejar en los parámetros.
- Funciones: cumple misma función que los procedimientos pero estos permiten devolver un valor en el punto donde se llamó. Se las invoca dentro de expresiones y lo que produce reemplaza a la invocación dentro de la expresión y siempre retornan un valor.

Formas de conectar y conectar datos entre distintas unidades de programa

- Variables locales: no habrá problema en compartir datos porque estas variables solo son accesibles en la unidad donde se declaró.
- Variables no locales: o sea las globales o definidas en otra unidad, tendrán 2 formas de accederlas
  - Acceso al ambiente no local: las unidades acceden a variables no locales generando problemas y errores porque las variables pueden ser modificadas por distintas unidades, además de que las variables globales pueden hacer que el código sea poco legible. A su vez se puede clasificar en
    - Ambiente no local implícito: forma en la que un programa accede automáticamente a variable y código que se encuentra fuera de su alcance inmediato.
    - Ambiente común explícito: se indica explícitamente el código y variables que pueden ser accedidas.
  - Parámetros: forma más clara. Las unidades se comunican entre sí mediante parámetros, que son valores pasados entre distintas unidades. Tenemos 2 tipos de parámetros

- Parámetro real: valor pasado a la rutina durante su invocación y se usa para proporcionar la entrada.
- Parámetro formal: variable usada para recibir los valores de entrada. Se encuentra en la declaración y se usa para especificar el tipo y nombre de parámetro esperado.

Ventaja de pasaje de parámetros

- Validación de valores de entrada antes de utilizarse, previniendo errores de lógica.
- Enviar distintos parámetros en distintas invocaciones a las rutinas.
- Mayor flexibilidad.
- Adecuación a las reglas de los lenguajes.
- Protección de datos.
- Legibilidad.
- Modificabilidad.

Los parámetros formales son variables locales a su entorno en cuanto a los parámetros reales pueden ser locales, no locales o globales.

La evaluación de los parámetros reales y la ligadura con los formales son procesos hechos en la llamada a la rutina, la cual es, antes de invocar la función, obtener los valores de los parámetros reales para asegurarse que sean correctos y estén disponibles cuando se la llame.

La ligadura es el proceso de asociar el parámetro real con el formal y se puede hacer de 2 formas

- Ligadura posicional: los parámetros reales se corresponden con la posición que ocupan en la lista de parámetros.
- Ligadura por palabra clave o nombre: los parámetros reales se corresponden con el nombre de los parámetros formales. Su desventaja es que hay que conocer los nombres de los formales para asignarlo porque podría causar errores.

Tipos de parámetros

- Datos: hay diferentes formas de transmitir los parámetros hacia y desde el programa llamado.

  Los modos de los parámetros formales se refiere a cómo se transmiten los datos entre los parámetros reales y formales, puede ser de 3 formas

  - IN: el parámetro formal recibe un valor del parámetro real pero no lo modifica. Fuera de la rutina, si se hace un cambio este no se verá reflejado.

A su vez, el modo IN tiene dos formas de recibir valores

- Por valor: forma de transmitir los parámetros desde la unidad llamante hacia la unidad llamada en la que el valor del parámetro real se usa para inicializar parámetro formal. De esta forma, se transfiere el valor del parámetro real y se copia a una nueva variable que actúa como una variable local de la unidad llamada.

  Lo bueno es que protege datos pero consume tiempo y el almacenamiento es el doble.

- Por valor constante: por valor constante es similar al por valor, pero no permite que el parámetro formal se modifique.

  Requerirá más trabajo para implementar los controles pero protege los datos.

- OUT: el parámetro formal devuelve un valor del parámetro real una vez que la rutina terminó.
  - Por resultado: el valor del parámetro formal se copia en el parámetro real después de la llamada a la rutina. El parámetro formal es una variable local y no recibe ningún valor inicial entonces es importante inicializar el parámetro formal antes del OUT.

    La ventaja es que los datos de la unidad llamadora están protegidos pero consume espacio debido a la copia al final.

  - Resultado de funciones: el valor devuelto se usa como resultado en la expresión que tiene la llamada a la función. El parámetro formal actúa como una variable local dentro de la función, y el valor devuelto se copia en el parámetro real después de la ejecución de la función.

    La ventaja de este modo es que protege los datos de la unidad llamadora, ya que el parámetro real no se modifica en la ejecución de la función. Sin embargo, puede consumir tiempo y espacio ya que se hace una copia al final de la función.

- IN/OUT: el parámetro formal se usa para recibir un valor y para devolverle otro al parámetro real.
  - Por valor-resultado: paso de parámetros en el que el parámetro formal recibe una copia del parámetro real, trabaja sobre su copia y el valor del parámetro formal se copia en el parámetro real.

  - Por referencia: el vez de copiar el valor del parámetro real al formal, se asocia la dirección de memoria del parámetro real al formal. El parámetro formal actúa como una variable que contiene la dirección del parámetro real de la unidad llamadora.

Es eficiente con respecto al tiempo y espacio pero puede haber una modificación inadvertida, afectando la legibilidad y confiabilidad.

- Por nombre: funciona de manera similar al modo por referencia, en el sentido de que se establece una conexión entre el parámetro formal y el parámetro real. Sin embargo, la "ligadura de valor" se difiere hasta el momento en que se utiliza, es decir, la dirección se resuelve en tiempo de ejecución. Dependiendo del tipo de dato que se quiera compartir, el comportamiento puede ser similar al pasaje por referencia, por valor o puede cambiar el suscripto de un arreglo en las distintas referencias.

  Para implementarlo, se utilizan procedimientos sin nombre llamados "thunks", que reemplazan cada aparición del parámetro formal en el cuerpo de la unidad llamada y activan el procedimiento que evaluará el parámetro real en el ambiente apropiado en el momento de la ejecución.

Lenguaje	Modo de pasaje de parámetros
C	Por valor o valor constante con la sentencia “const”.
Pascal	Por valor (por defecto) o por referencia (aclarando “var”).
C++	Similar a C pero con más pasaje por referencia.
Java	Solo copia de valor, pero como las variables no primitivas son referencias a variables anónimas, el paso por valor de una de estas variables es en realidad un paso por referencia de la variable. Lee estructuras de datos primitivos almacenan datos de un solo tipo y las no primitivas pueden almacenar datos de más de un tipo.
PHP	Por valor o por referencia (usando &)
Ruby	Por valor

- Subprogramas: a veces es manejar los nombre de subprogramas como parámetros aunque no es incorporado por todos los lenguajes.

  Cuando se pasa un subprograma como parámetro, hay que determinar cuál es el ambiente de referencia de las variables no locales del cuerpo de rutina. Para determinarlo, hay varias opciones

  - Ligadura superficial: las referencias no locales dentro del cuerpo del subprograma pasado como parámetro se resuelven en el subprograma que tiene el parámetro formal subprograma.
  - Ligadura profunda: las referencias no locales dentro del cuerpo del subprograma pasado como parámetro se resuelven en el subprograma donde está declarado el subprograma usado como parámetro real.

Unidades genéricas

Unidades que pueden personalizarse para funcionar con distintos tipos de datos. Al permitir diferentes instancias con diferentes subprogramas, las unidades genéricas pueden proporcionar una funcionalidad adaptable a las necesidades de un programa específico. Esto puede ayudar a hacer el código más eficiente y reutilizable, ya que una unidad genérica puede ser instanciada varias veces para diferentes tipos de datos sin necesidad de escribir un nuevo código para cada instancia.