UNIDAD I

Manejo de memoria estática y dinámica

La diferencia entre estructuras estáticas y dinámicas es el manejo de memoria:

Estática

- Durante la ejecución del programa el tamaño de la estructura no cambia.
- Se define el tamaño en tiempo de compilación.

Dinámica

- Durante la ejecución del programa el tamaño de la estructura puede cambiar.
- Se define el tamaño en tiempo de ejecución.

Asignación de memoria

- consiste en el proceso de asignar memoria para propósitos específicos, ya sea en tiempo de compilación o de ejecución.
- Si es en tiempo de compilación es *estática*
- Si es en tiempo de ejecución es *dinámica*
- Si son variables locales a un grupo de sentencias se denomina *automática*.

Asignación estática de memoria$_1$

- consiste en el proceso de asignar memoria en tiempo de compilación antes de que el programa asociado sea ejecutado, a diferencia de la asignación dinámica o la automática donde la memoria se asigna a medida que se necesita en tiempo de ejecución.
- un módulo de programa (por ejemplo función o subrutina) declara datos estáticos de forma local, de forma que estos datos son inaccesibles desde otros módulos a menos que se les pasen referenciados como parámetros o que les sean devueltos por la función.

Asignación estática de memoria$_2$

- Se mantiene una copia simple de los datos estáticos, accesible a través de llamadas a la función en la cual han sido declarados.
- El uso de variables estáticas dentro de una clase en la programación orientada a objetos permite que una copia individual de tales datos se comparta entre todos los objetos de esa clase.
- Las constantes conocidas en tiempo de compilación, como literales de tipo cadena, se asignan normalmente de forma estática.
- En programación orientada a objetos, el método usual para las tablas de clases también es la asignación estática de memoria.

Asignación dinámica de memoria$_1$

- Es la asignación de almacenamiento de memoria para utilización por parte de un programa de computador durante el tiempo de ejecución de ese programa.
- Es una manera de distribuir la propiedad de recursos de memoria limitada entre muchas piezas de código y datos.
- Un objeto asignado dinámicamente permanece asignado hasta que es desasignado explícitamente, o por el programador o por un recolector de basura; esto es notablemente diferente de la asignación automática de memoria y de la asignación estática de memoria (la de las variables estáticas). Se dice que tal objeto tiene tiempo de vida dinámico.

Asignaciones automáticas de memoria (locales estáticas)

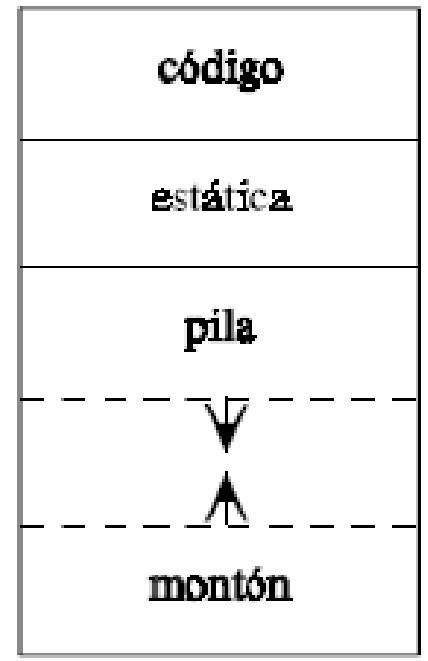
- Las variables automáticas son variables locales a un bloque de sentencias.
- Pueden ser asignadas automáticamente en la pila de datos cuando se entra en el bloque de código.
- Cuando se sale del bloque, las variables son automáticamente desasignadas.
- Las variables automáticas tendrán un valor sin definir cuando son declaradas, por tanto es buena práctica de programación inicializarlas con un valor válido antes de usarlas.

Proceso de carga de un programa en Memoria

- Cuando un programa se ejecuta sobre un sistema operativo existe un proceso previo llamado cargador que suministra al programa un bloque contiguo de memoria sobre el cual ha de ejecutarse.
- El programa resultante de la compilación debe organizarse de forma que haga uso de este bloque.
- Para ello el compilador incorpora al programa objeto el código necesario.

Organización de la memoria:

- Para lenguajes imperativos, los compiladores generan programas que tendrán en tiempo de ejecución una organización de la memoria de la siguiente manera:



Segmento de código (CS)

- Es la zona donde se almacenan las instrucciones del programa ejecutable en código máquina, y también el código. correspondiente a los procedimientos y funciones que utiliza.
- Su tamaño puede fijarse en tiempo de compilación.
- Algunos compiladores fragmentan el código del programa objeto usando “overlays”.
- Estos “overlays” son secciones de código objeto que se almacenan en archivos independientes y que se cargan en la memoria central (RAM) dinámicamente, es decir, durante la ejecución del programa. Los overlays de un programa se agrupan en zonas y módulos, cada uno de los cuales contiene un conjunto de funciones o procedimientos.

Memoria estática (Segmento de Datos - DS)

❖ La forma más fácil de almacenar el contenido de una variable en memoria en tiempo de ejecución es en memoria estática o permanente a lo largo de toda la ejecución del programa.

❖ No todos los objetos (variables) pueden ser almacenados estáticamente.

❖ Para que un objeto pueda ser almacenado en memoria estática su tamaño (número de bytes necesarios para su almacenamiento) ha de ser conocido en tiempo de compilación.

Memoria estática (Segmento de Datos - DS)

Como consecuencia de esta condición no podrán almacenarse en memoria estática:

❖ Los objetos correspondientes a procedimientos o funciones recursivas, ya que en tiempo de compilación no se sabe el número de variables que serán necesarias.

❖ Las estructuras dinámicas de datos tales como listas, árboles, etc. ya que el número de elementos que la forman no es conocido hasta que el programa se ejecuta.

Asignación de memoria estática

Las técnicas de asignación de memoria estática son sencillas:

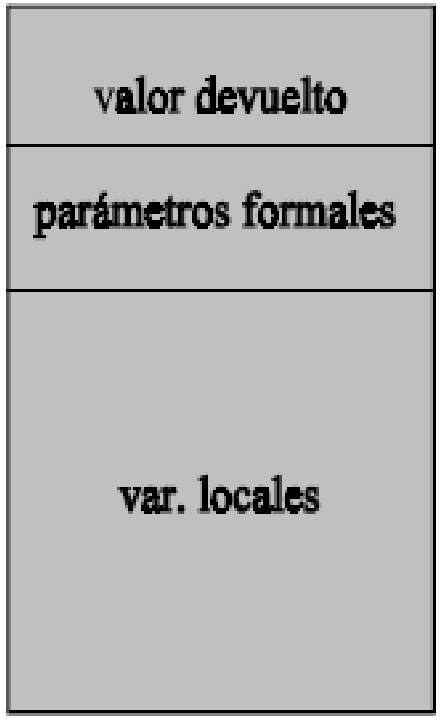
❖ A partir de una posición señalada por un puntero de referencia se aloja el objeto X, y se avanza el puntero tantos bytes como sean necesarios para almacenar el objeto X.

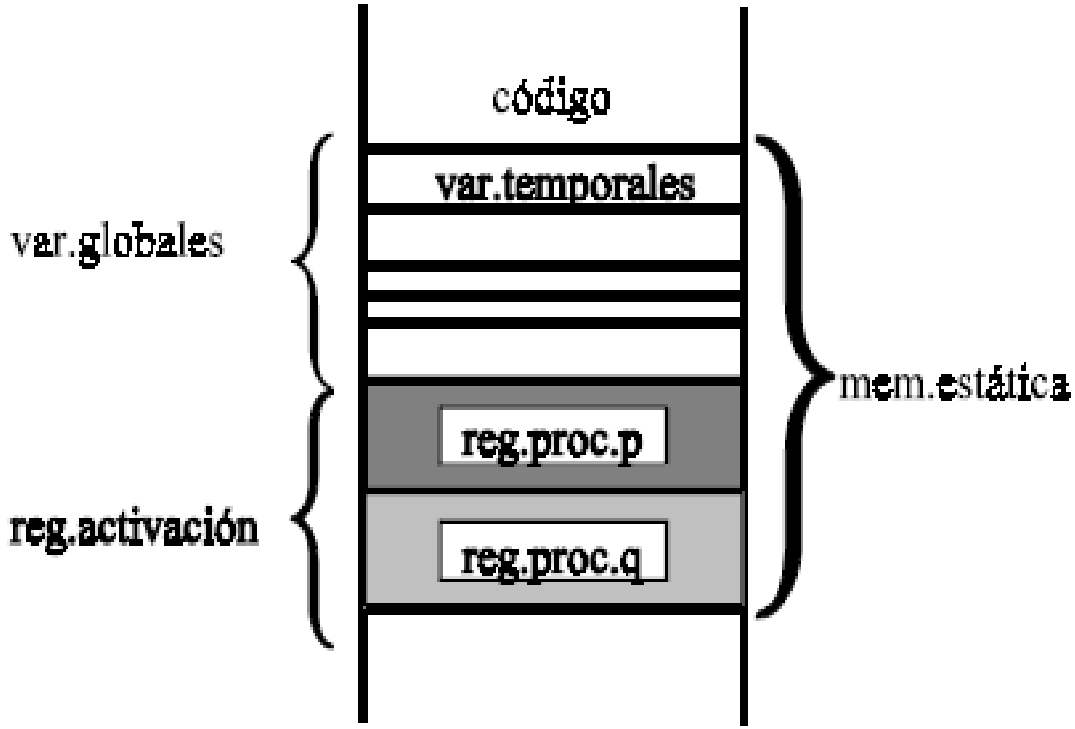
❖ La asignación de memoria puede hacerse en tiempo de compilación y los objetos están vigentes desde que comienza la ejecución del programa hasta que termina.

❖ En los lenguajes que permiten la existencia de subprogramas, y siempre que todos los objetos de estos subprogramas puedan almacenarse estáticamente se aloja en la memoria estática un registro de activación correspondiente a cada uno de los subprogramas.

Estructura de Registros de Activación en memoria estática

Estos registros de activación contendrán las variables locales, parámetros formales y valor devuelto por la función, tal como se indica:





Asignación estática para subprogramas

1. Dentro de cada registro de activación las variables locales se organizan secuencialmente.

2. Existe un solo registro de activación para cada procedimiento y por tanto no están permitidas las llamadas recursivas.

3. Dado que las variables están permanentemente en memoria es fácil implementar la propiedad de que conserven o no su contenido para cada nueva llamada.

Asignación estática para subprogramas

El proceso que se sigue cuando un procedimiento p llama a otro q es el siguiente:

• p evalúa los parámetros de llamada, en caso de que se trate de expresiones complejas, usando para ello una zona de memoria temporal para el almacenamiento intermedio.

• Por ejemplo, si la llamada a q es q((3*5)+(2*2),7) las operaciones previas a la llamada propiamente dicha en código máquina han de realizarse sobre alguna zona de memoria temporal. (En algún momento debe haber una zona de memoria que contenga el valor intermedio 15, y el valor intermedio 4 para sumarlos a continuación).

• En caso de utilización de memoria estática ésta zona de temporales puede ser común a todo el programa, ya que su tamaño puede deducirse en tiempo de compilación.

• q inicializa sus variables y comienza su ejecución.

Segmento de pila (SS)

❖La aparición de lenguajes con estructura de bloque trajo consigo la necesidad de técnicas de alojamiento en memoria más flexibles, que pudieran adaptarse a las demandas de memoria durante la ejecución del programa.

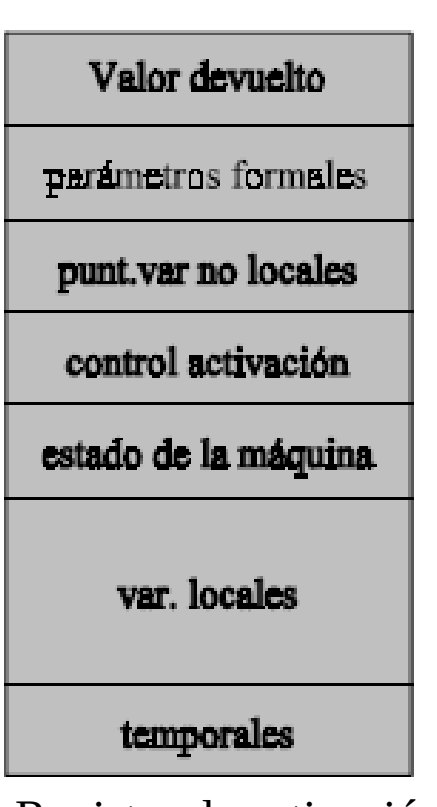
❖En estos lenguajes, cada vez que comienza la ejecución de un procedimiento se crea un registro de activación para contener los objetos necesarios para su ejecución, eliminándolo una vez terminada ésta.

❖Dado que los bloques o procedimientos están organizados jerárquicamente, los distintos registros de activación asociados a cada bloque deberán colocarse en una pila en la que entrarán cuando comience la ejecución del bloque y saldrán al terminar el mismo.

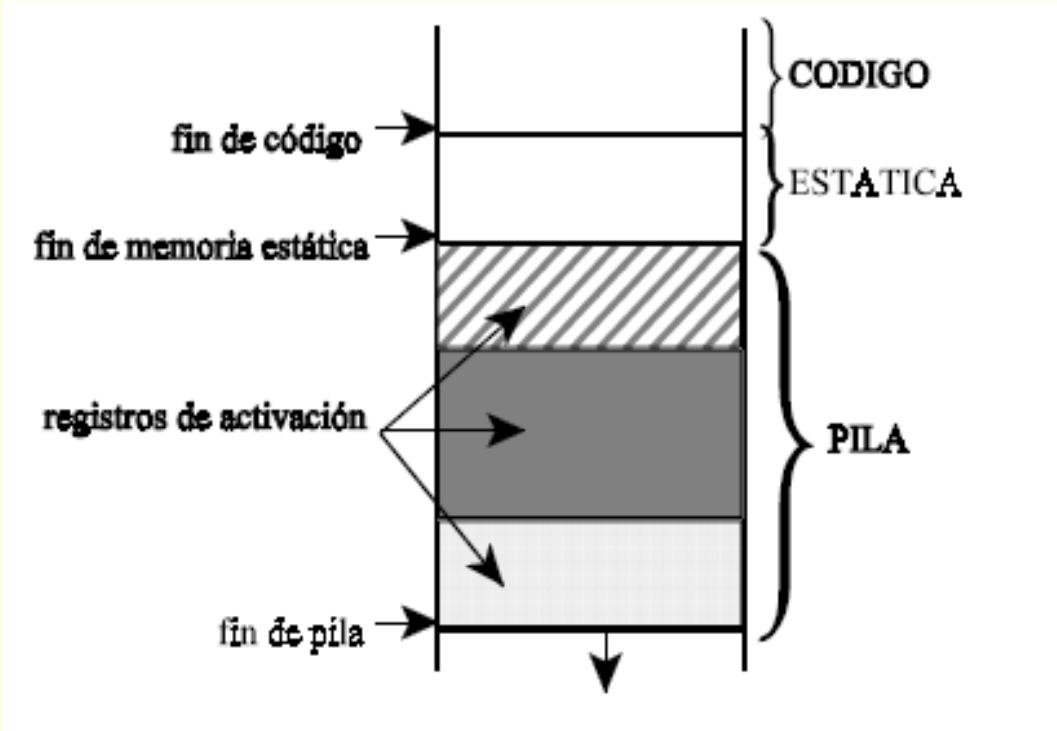
❖La estructura de los registros de activación varía de unos lenguajes a otros, e incluso de unos compiladores a otros. Este es uno de los problemas por los que a veces resulta difícil enlazar los códigos generados por dos compiladores diferentes.

Estructura de los registros de activación en PILA

En general, los registros de activación de los procedimientos suelen tener algunos de los campos que pueden verse a continuación:



Registro de activación



Funcionamiento de la pila(1)

- El puntero de control de activación guarda el valor que tenía el puntero de la cima de la pila antes de que entrase en ella el nuevo registro, de esta forma una vez que se desee desalojarlo puede restituirse el puntero de la pila a su posición original. Es decir, es el puntero que se usa para la implementación de la estructura de datos “Pila” del compilador.
- En la zona correspondiente al estado de la máquina se almacena el contenido que hubiera en los registros de la máquina antes de comenzar la ejecución del procedimiento. Estos valores deberán ser repuestos al finalizar la ejecución del procedimiento. El código encargado de realizar la copia del estado de la máquina es común para todos los procedimientos.
- El puntero a las variables no locales permite el acceso a las variables declaradas en otros procedimientos. Normalmente no es necesario usar este campo puesto que puede conseguirse lo mismo con el puntero de control de activación, sólo tiene especial sentido cuando se utilizan procedimientos recursivos.

Funcionamiento de la pila(2)

- Al igual que en el alojamiento estático los registros de activación contendrán el espacio correspondiente a los parámetros formales (variables que aparecen en la cabecera) y las variables locales, (las que se definen dentro del bloque o procedimiento) así como una zona para almacenar el valor devuelto por la función y una zona de valores temporales para el cálculo de expresiones.
- Para dos módulos o procedimientos diferentes, los registros de activación tendrán tamaños diferentes. Este tamaño por lo general es conocido en tiempo de compilación ya que se dispone de información suficiente sobre el tamaño de los objetos que lo componen.
- En ciertos casos esto no es así como por ejemplo ocurre en C cuando se utilizan arrays de dimensión indefinida. En estos casos el registro de activación debe incluir una zona de desbordamiento al final cuyo tamaño no se fija en tiempo de compilación sino solo cuando realmente llega a ejecutarse el procedimiento.
- Esto complica un poco la gestión de la memoria, por lo que algunos compiladores de bajo coste suprimen esta facilidad.

Asignación de memoria dentro de la pila (Invocación de subprogramas)

El procedimiento de gestión de la pila cuando un procedimiento q llama a otro procedimiento p, se desarrolla en dos fases: la primera de ellas corresponde al código que se incluye en le procedimiento q antes de transferir el control a p, y la segunda, al código que debe incluirse al principio de p para que se ejecute cuando reciba el control.

• El procedimiento autor de la llamada (q) evalúa las expresiones de la llamada, utilizando para ello su zona de variables temporales, y copia el resultado en la zona correspondiente a los parámetros formales del procedimiento que recibe la llamada.

• El autor de la llamada (q) coloca el puntero de control del procedimiento al que llama (p) de forma que apunte al final de la pila y transfiere el control al procedimiento al que llama (p).

• El receptor de la llamada (p) salva el estado de la máquina antes de comenzar su ejecución usando para ello la zona correspondiente de su registro de activación.

• El receptor de la llamada (p) inicializa sus variables y comienza su ejecución.

Asignación de memoria dentro de la pila (Terminación de subprogramas)

Al terminar la ejecución del procedimiento llamado (p) se desaloja su registro de activación procediendo también en dos fases:

❖ La primera se implementa mediante instrucciones al final del procedimiento que acaba de terminar su ejecución (p)

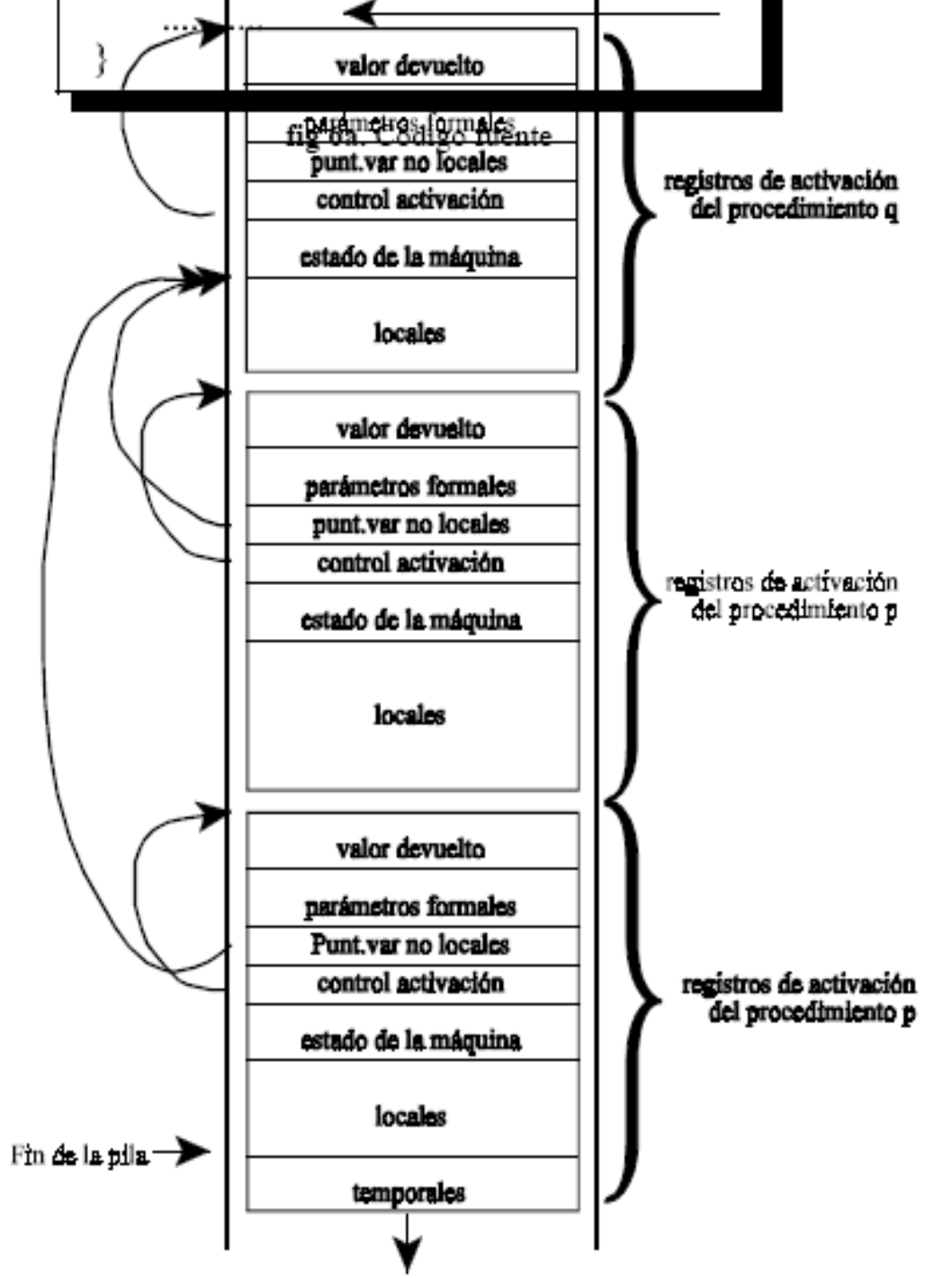
❖ La segunda en el procedimiento que hizo la llamada tras recobrar el control:

- El procedimiento saliente (p) antes de finalizar su ejecución coloca el valor de retorno al principio de su registro de activación.
- Usando la información contenida en su registro el procedimiento que finaliza (p) restaura el estado de la máquina y coloca el puntero de final de pila en la posición en la que estaba originalmente.
- El procedimiento que realizó la llamada (q) copia el valor devuelto por el procedimiento al que llamó (p) dentro de su propio registro de activación (de q).

```
función q (... //param.  formales q // ...)
{
    // var. locales de q
    .....
    x = p (23 + y * z .....)
}
función p (... //param.  formales p // ...)
{
    // var. locales de p
    .....
    p (.....);
    .....
}
```

valor devuelto

fig 6a. Código fuente



Consideraciones de la pila

- Dentro de un procedimiento, las variables locales se referencian siempre como direcciones relativas al comienzo del registro de activación, o bien al comienzo de la zona de variables locales.
- Por tanto, cuando sea necesario acceder desde un procedimiento a variables definidas de otros procedimientos cuyo ámbito sea accesible, será necesario proveer dinámicamente la dirección de comienzo de las variables de ese procedimiento.
- Para ello se utiliza dentro del registro de activación el puntero de enlace a variables no locales. Este puntero, señalará al comienzo de las variables locales del procedimiento inmediatamente superior.
- El puntero de enlace es una posición fija con respecto al comienzo de sus variables locales.
- Cuando los procedimientos se llaman a sí mismos recursivamente, el ámbito de las variables impide por lo general que una activación modifique las variables locales de otra activación del mismo procedimiento, por lo que en estos casos el procedimiento inmediato superior será, a efectos de enlace, el que originó la primera activación.

Monticulo (heap)

- Cuando el tamaño de un objeto a colocar en memoria puede variar en tiempo de ejecución, no es posible su ubicación en memoria estática, ni tan siquiera en la pila.
- Son ejemplos de este tipo de objetos las listas, los árboles, las cadenas de caracteres de longitud variable, etc.
- Para manejar este tipo de objetos el compilador debe disponer de un área de memoria de tamaño variable y que no se vea afectada por la activación o desactivación de procedimientos.
- Este trozo de memoria se llama montón (traducción literal del termino ingles heap que se utiliza habitualmente en la literatura técnica).
- En aquellos lenguajes de alto nivel que requieran el uso del montón, el compilador debe incorporar al programa objeto el código correspondiente a la gestión del montón.

Operaciones Monticulo (heap)

Alojamiento: Se demanda un bloque contiguo para poder almacenar un objeto de un cierto tamaño.

Desalojo: Se indica que ya no es necesario conservar un objeto, y que por lo tanto, la memoria que ocupa debe quedar libre para ser reutilizada en caso necesario por otros objetos.

Las operaciones pueden ser explícitas o implícitas respectivamente.

- En caso de alojamiento explícito el programador incluye en el código fuente una instrucción que demanda una cierta cantidad de memoria para la ubicación de un dato (por ejemplo en PASCAL mediante la instrucción new; en C mediante malloc, etc.).
- La cantidad de memoria requerida puede ser calculada por el compilador en función del tipo correspondiente al objeto que se desea alojar, o bien puede ser especificado directamente por el programador.

Operaciones Monticulo (heap)

- El resultado de la función de alojamiento es por lo general un puntero a un trozo contiguo de memoria dentro del montón que puede usarse para almacenar el valor del objeto.
- Los lenguajes de programación imperativos utilizan por lo general alojamiento y desalojo explícitos.
- Por el contrario los lenguajes lógicos y funcionales evitan al programador la tarea de manejar directamente los punteros realizando las operaciones implícitamente.
- La gestión del montón requiere técnicas adecuadas que optimicen el espacio que se ocupa o el tiempo de acceso, o bien ambos factores.

Problemas de Asignación de Memoria

- La tarea de satisfacer una petición de asignación, la cual conlleva encontrar un bloque de memoria sin usar de cierto tamaño en el heap, es un problema complicado.
- Se han propuesto una amplia variedad de soluciones, incluyendo listas de bloques libres, Paginación, y Asignación buddy de memoria.
- El problema principal para la mayoría de algoritmos de asignación de memoria dinámica es evitar la fragmentación interna y externa mientras se mantiene la eficiencia del algoritmo.
- También, la mayoría de algoritmos en uso tienen el problema de que un número grande de pequeñas asignaciones pueden causar el desaprovechamiento del espacio debido a la recolección de metadatos; así la mayoría de los programadores intentan evitar esto, a veces usando una estrategia llamada chunking.

Asignación de bloques de tamaño fijo

- Una solución es tener una lista enlazada LIFO de bloques de memoria de tamaño fijo. Esto funciona bien para sistemas empotrados simples.

Algoritmo Buddy

- En este sistema, la memoria se asigna desde un gran bloque de memoria que es tamaño potencia de dos.
- Si el bloque es más del doble de grande de lo necesario, se parte en dos. Se selecciona una de las dos mitades, y el proceso se repite (comprobando el tamaño otra vez y partiendo si se necesita) hasta que el bloque sea justamente el necesitado.
- Todos los segmentos de memoria de un tamaño particular son guardados en una lista enlazada ordenada o una estructura de datos en árbol.
- Cuando se libera un bloque, se compara con su buddy(vecino). Si los dos están libres, son combinados y colocados en la lista de bloques buddy de siguiente mayor tamaño. (Cuando un bloque es asignado, el asignador empezará con el bloque grande suficientemente pequeño para evitar romper bloques innecesariamente)

Asignación de memoria basada en Heap

- La memoria es asignada desde un gran pool de área de memoria sin usar llamada heap (también llamada almacén de libres).
- "El heap" no tiene nada que ver con la estructura de datos Heap.
- El tamaño de la asignación de memoria puede ser determinado en tiempo de ejecución, y el tiempo de vida de la asignación no es dependiente del procedimiento actual o del marco de pila.
- La región de memoria asignada es accedida indirectamente, normalmente por medio de una referencia.
- El algoritmo preciso usado para organizar la área de memoria y asignar y desasignar los trozos está oculto detrás de una interfaz abstracta y puede usar cualquiera de los métodos descritos antes.
- En contraste, la memoria de la pila de llamadas es normalmente de tamaño limitado y el tiempo de vida de la asignación depende de la duración de las funciones correspondientes.

Administración de memoria en Java

- Java utiliza un modelo de memoria conocido como "administración automática del almacenamiento" (automatic storage management), en el que el sistema en tiempo de ejecución de Java mantiene un seguimiento de los objetos.
- En el momento que no están siendo referenciados por alguien, automáticamente se libera la memoria asociada con ellos.
- Existen muchas maneras de implementar recolectores de basura, entre ellas tenemos:
- Contabilizar referencias. La máquina virtual Java asocia un contador a cada instancia de un objeto, donde se refleja el número de referencias hacia él. Cuando este contador es 0, la memoria asociada al objeto es susceptible de ser liberada. Aún cuando este algoritmo es muy sencillo y de bajo costo (en términos computacionales), presenta problemas con estructuras de datos circulares.

Administración de memoria en Java

- Marcar e intercambiar (Mark-and-Sweep). Este es el esquema más común para implementar el manejo de almacenamiento automático. Consiste en almacenar los objetos en un montículo (heap) de un tamaño considerable y marcar periódicamente (generalmente mediante un bit en un campo que se utiliza para este fin) los objetos que no tengan ninguna referencia hacia ellos.

- Adicionalmente existe un montón alterno, donde los objetos que no han sido marcados, son movidos periódicamente. Una vez en el montículo alterno, el recolector de basura se encarga de actualizar las referencias de los objetos a sus nuevas localidades. De esta manera se genera un nuevo montículo, que contiene únicamente objetos que están siendo utilizados.

- Existen muchos otros algoritmos para implementar sistemas que cuenten con recolección de basura.