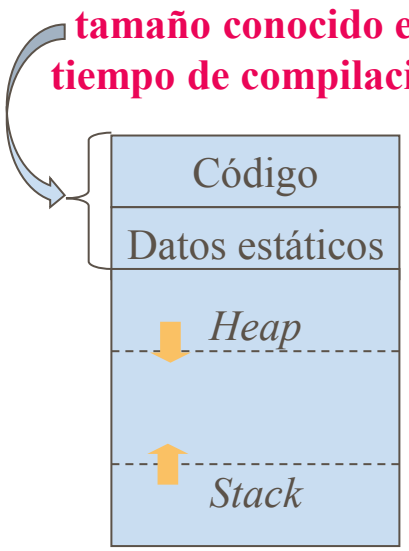
Organización de la memoria en tiempo de ejecución

Eva Lucrecia Gibaja Galindo
Dpto. Informática y Análisis Numérico

Administración de la memoria en tiempo de ejecución

- Al ejecutar un programa, la memoria se divide (de manera lógica, no física) en:
  - **Código ejecutable**
    - Código máquina, constantes y literales (sólo lectura)
    - Se puede situar en una zona de memoria estática (permanente durante toda la ejecución)
  - **Datos**
    - Variables globales y estáticas (lectura/escritura)
  - La **pila o *stack*** de control que controla las ejecuciones de los procedimientos
    - Cuando se produce una llamada, se interrumpe la ejecución de una activación y se guarda en la pila
      - Variables locales y resultados intermedios
      - Información sobre el estado de la máquina
  - **Montículo o *heap***
    - Guarda el resto de la información generada en tiempo de ejecución
    - Bloques de memoria direccionados por punteros



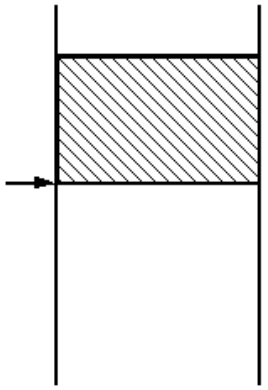
El tamaño del *stack* y el *heap* varia durante la ejecución del programa. Como la memoria reservada para ambos es fija, si para una se necesita mucho espacio, se reduce el espacio para la otra y viceversa

Estrategias de asignación de memoria

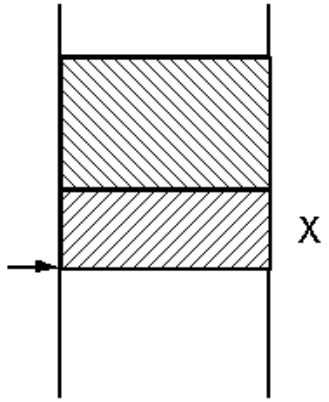
- **Asignación estática**. Dispone la memoria para todos los datos durante la compilación
- **Asignación por medio de una pila**. Trata la memoria en ejecución como una pila
- **Asignación por medio de montículo** y desasigna la memoria conforme se necesita, durante la ejecución

Asignación estática de memoria

- El tamaño la memoria que va a ser reservada es conocido en tiempo de compilación
- Los objetos están vigentes desde que comienza la ejecución del programa hasta que termina
- La memoria se reserva de forma **consecutiva**
- La dirección de memoria de los objetos se puede conocer en tiempo de compilación
  - Si la primera variable está en la dirección x y ocupa n_1 unidades de memoria, la segunda variable estará en la dirección $x+n_1$, la siguiente en $x+n_1+n_2$ etc.
- Se suele utilizar en asignaciones estáticas para datos globales



Alojamiento en memoria de un objeto x

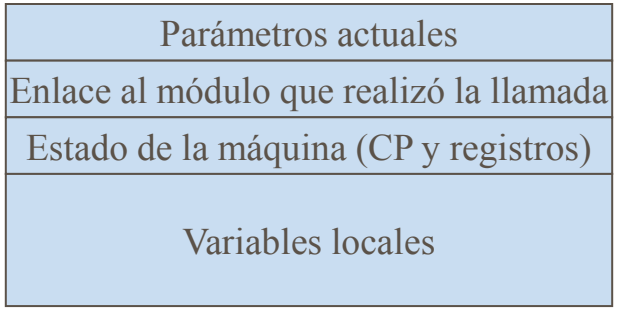


Asignación estática de memoria

- **Ventajas**
  - La implementación de esta estrategia es simple
  - Conocer las direcciones de las variables al generar el código objeto mejora el tiempo de ejecución de los programas
- **Inconvenientes**
  - El tamaño del objeto ha de ser conocido en tiempo de compilación
  - Al no asignar memoria en tiempo de ejecución:
    - No permite procedimientos recursivos puesto que todas las llamadas recursivas utilizarían las mismas posiciones de memoria, guardando solo un entorno de la función recursiva
    - Las estructuras de datos no se pueden crear dinámicamente (listas, árboles)

Asignación mediante pila (*stack*)

- **Registro de activación *(stack frame)***
  - Información que se transmite a una función cuando es llamada
  - La magnitud de estos campos se puede determinar en tiempo de compilación



- **Función recursiva**
  - El número de llamadas recursivas ($\rightarrow$ número de registros de activación) no es conocido en tiempo de compilación
  - Para implementar la recursividad se usa una estructura de pila (*stack*) de forma que:
    - Al llamar a una función se coloca un nuevo registro de activación en la pila
    - Al finalizar la función se extrae
    - Entre la activación de un bloque y su terminación pueden ser invocados otros bloques

Asignación mediante montículo

- La estrategia anterior no es suficiente para manejar estructuras de datos cuya magnitud puede cambiar durante la ejecución de un programa y no es conocida en tiempo de compilación sino en tiempo de ejecución
- Para gestionar esta memoria se reserva un gran bloque contiguo de memoria llamado *montículo*, *montón* o *heap*
- Cuando se realiza una petición adicional de memoria, un controlador de memoria localiza espacio en el *heap* en tiempo de ejecución
- Cuando un especio ha dejado de utilizarse, esta memoria ha de ser liberada. Existen tres técnicas de liberar la memoria:
  - **No liberar**. Cuando se acaba la memoria se para la ejecución del programa
  - **Liberación explícita**. Con lo cual se deja al programador la responsabilidad de liberar la memoria
  - **Liberación implícita**. Recuperación automática del espació en el *heap* sin utilizar. Este proceso se denomina a menudo **recolección de basura**

Asignación mediante montículo

- Estrategias de desasignación **implícita:**
  - **Desocupación sobre la marcha** (*free-as-you-go*)
    - Desalojar (implícitamente) los bloques del montón tan pronto como se detecta que no están referenciados por ningún puntero
    - Es necesario que el sistema lleve internamente un contador del número de punteros que se dirigen hacia cada uno de los bloques del montón
  - **Método de *Marcar y Barrer*** (*mark and sweep*)
    - Activar un procedimiento especial cuando se detectan ciertas condiciones en el uso de la memoria (ej. cuando queda poca memoria libre, finaliza un módulo, etc.)