

**Instituto Tecnológico de Tijuana**

**Subdirección Académica**

**Departamento de Sistemas y Computación**

**Semestre agosto-diciembre 2018**

**Carrera:** Ing. En Sistemas Computacionales Serie SC1A

**Materia:** Estructura de Datos

**Unidad 1 – Manejo de Memoria Estática y Manejo de Memoria Dinámica**

**Alumno:**15211883 - Angeles Valadez Jonathan

**Profesor:** Ray Brunett Parra Galaviz

Organización de la memoria en tiempo de ejecución Podemos distinguir tres tipos de memoria. En primer lugar, está la ocupada por el código del programa. Por otro lado, está la destinada a datos estáticos, que se gestiona en tiempo de compilación. Finalmente, tenemos la memoria que debe gestionarse en tiempo de ejecución. Hay poco que decir sobre la memoria ocupada por el programa. Su tamaño es conocido en tiempo de compilación y, en la mayoría de los lenguajes modernos, se puede decir que es “invisible”: el código no puede referirse a sí mismo ni modificarse. La memoria destinada a datos estáticos se utiliza para las variables globales y algunas otras como las variables estáticas de las subrutinas. La gestión de esta memoria es sencilla y se puede realizar en tiempo de compilación.

Al hablar de la memoria gestionada en tiempo de ejecución, podemos distinguir entre la memoria que se utiliza para albergar los objetos que se crean y destruyen con la ejecución de las subrutinas (parámetros, variables locales y algunos datos generados por el compilador) y la que se suele conocer como memoria dinámica, que se reserva explícitamente por el programador o que se necesita para almacenar objetos con tiempos de vida o tamaños desconocidos en tiempo de compilación. La memoria asociada a las subrutinas seria gestionada mediante una pila. La memoria dinámica se localizara en un heap.

**Memoria estática**

Es aquella que durante la ejecución de un programa su valor no cambia.

Por ejemplo: listas, vectores, matrices, pilas, colas y listas enlazadas.

La gestión de la memoria estática es la más sencilla. De hecho, algunos lenguajes de programación antiguos, como las primeras versiones de FORTRAN, únicamente tienen reserva estática de memoria. Las principales ventajas son la sencillez de implementación y que los requerimientos de memoria del programa son conocidos una vez compilado el mismo.

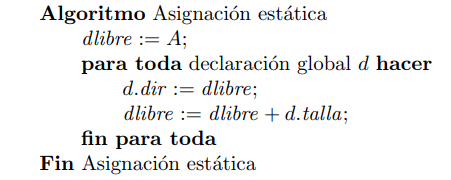
Sin embargo, existen bastantes inconvenientes: El tamaño de los objetos debe ser conocido en tiempo de compilación: no se puede trabajar con objetos de longitud variable.

Es difícil para el programador definir el tamaño de las estructuras que va a usar: si son demasiado grandes, desperdiciara memoria; si son pequeñas, no podrá utilizar el programa en todos los casos.

Solo puede haber una instancia de cada objeto: no se pueden implementar procedimientos recursivos. Por estas razones, casi todos los lenguajes de programación tienen además la posibilidad de gestionar parte de la memoria de manera dinámica.

La gestión de esta memoria se puede hacer íntegramente en tiempo de compilación. Para acceder a un objeto, el programa únicamente necesita saber en qué dirección se encuentra. Estas direcciones se pueden asignar secuencialmente. Basta con que el compilador tenga anotada la primera dirección libre del bloque de memoria estática y la vaya actualizando sumando la talla de los objetos que va reservando.

Supongamos que A es la primera dirección libre para datos globales. Las acciones que tiene que realizar el compilador para asignar las direcciones son:



Respecto a este algoritmo, hay que aclarar una serie de cosas. En primer lugar, el valor de A dependerá de la arquitectura de la máquina que vayamos a utilizar. Si el espacio de direcciones se dispone como hemos comentado al comienzo del tema, el valor inicial deberá ser la dirección inmediatamente posterior al código. Dado que la longitud de ´este no se conoce hasta el fin de la compilación, puede ser necesario generar las direcciones relativas a una etiqueta y dejar que el ensamblador o el enlazador las resuelva adecuadamente. Por otro lado, si la máquina permite que los datos estén en su propio segmento o se va a utilizar un enlazador que los pueda mover, lo habitual es comenzar por la dirección cero. En cuanto al momento de ejecutar el algoritmo, no hace falta que lo hagamos en una fase separada. Si tenemos dlibre como una variable global (o accesible en algún módulo del compilador) podemos realizar la actualización al crear el nodo correspondiente a la declaración. Es más, puede suceder que de esta manera no necesitemos crear nodos en el AST para las declaraciones, bastando con los objetos que se creen para albergar la información acerca de los identificadores.

**Memoria con reserva dinámica**

Es aquella que durante la ejecución de un programa su valor puede cambiar.

Por ejemplo: listas, vectores, matrices, pilas, colas y listas enlazadas.

La memoria que ocupan los objetos con tiempo de vida no predecible a partir del código fuente se gestiona dinámicamente: cuando se crea un objeto en tiempo de ejecución, se pide memoria para ´el, y cuando se elimina, la memoria se libera. En algunos lenguajes encontramos características que necesitan una gestión de memoria con reserva dinámica:

* Los operadores new y dispose del PASCAL
* La creación de elementos de listas en LISP y su liberación automática (garbage collection) por ausencia de referencias a la memoria
* las llamadas a funciones de biblioteca malloc y free del C
* la creación de objetos en Java y su eliminación cuando ninguna variable mantiene referencias a dichos objetos
* los operadores new y delete o las llamadas implícitas que comportan la creación y destrucción de objetos en C++
* etc.

La zona de memoria destinada a atender las demandas de memoria dinámica se denomina heap. Vamos a estudiar muy por encima el problema de la gestión del heap.

Mientras varias estrategias son usadas para colocar en un espacio procesos compitiendo por memoria, 3 de los más populares son “Best Fit”, “Worst Fit” y “First Fit”.

**Best Fit:** El asignador coloca un proceso en el bloque de memoria más pequeña disponible en el cual va a caber.

**Worst Fit:** El manejador de memoria coloca procesos en los bloques más grandes disponibles.

**First Fit:** Puede haber muchos lugares en la memoria, así que el sistema operativo para reducir la cantidad de tiempo que toma en analizar los lugares disponibles, comienza desde la memoria principal y coloca procesos en el primer lugar lo suficientemente grande para satisfacer el pedido de memoria.

**Bibliografía**

[En Línea]: <http://itslr.edu.mx/archivos2013/TPM/temas/s3u1.html>

[En Línea]: **Universitat Jaume -** **4º Ingeniería Informática II26 Procesadores de lenguaje Organización y gestión de la memoria**

<http://repositori.uji.es/xmlui/bitstream/handle/10234/5915/memoria.apun.pdf;jsessionid=9B378DBA752B81DF86F69DCD81745C9F?sequence=1>

[En Línea]: <https://courses.cs.vt.edu/csonline/OS/Lessons/MemoryAllocation/index.html>