ESAT
INNOVATION SCHOOL
ESCUELA SUPERIOR DE ARTE Y TECNOLOGÍA

IA

Subject:
IA: Teacher: Iván Sancho Year:
2020/2021
Author: Adam Álvarez Enfedal Saúl Vicente Camacho Díaz

Index

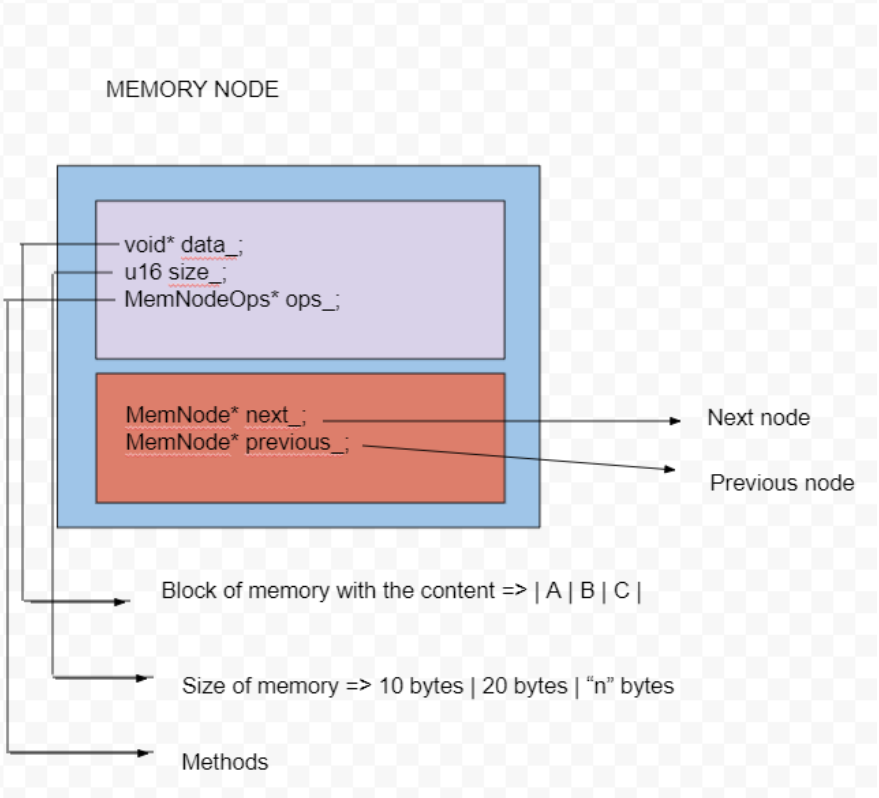
1.- Adts

Memory Node:

El memory node es el paquete básico que utilizan todos los Adts. Está formado por un puntero

tipo void que almacena cualquier tipo de dato, un puntero size que nos indica el tamaño del nodo, y dos punteros que necesitaremos en la lista.

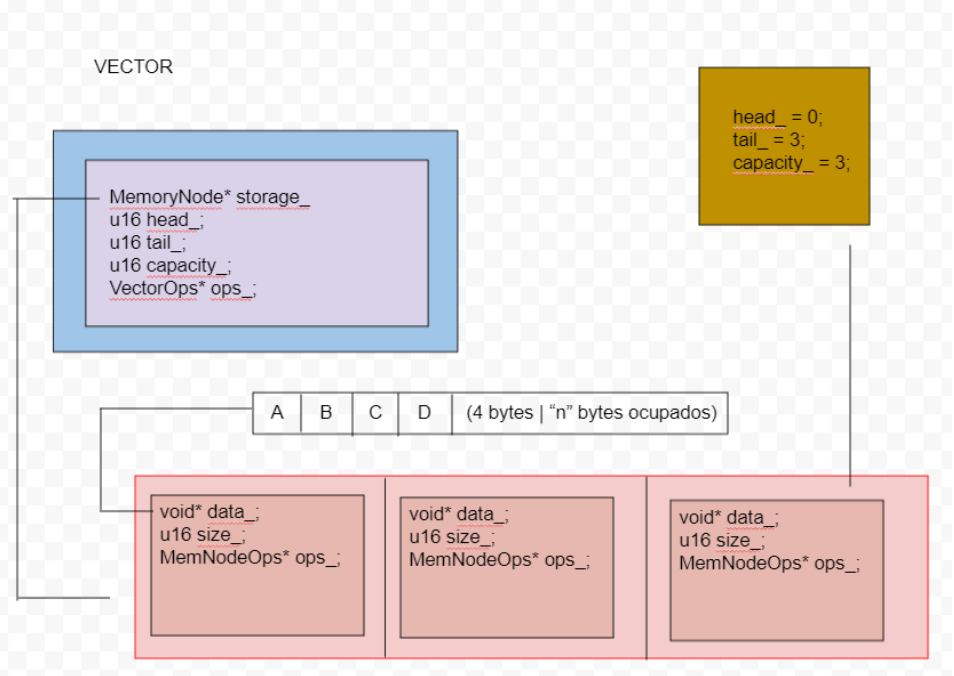
También tenemos un puntero que usaremos para hacer callback de las funciones de cada uno de los Adts.



Vector:

El Vector es un bloque de memoria contigua formada por nodos de memoryNode. Está compuesto por un puntero head, que nos marca el primer nodo del vector, un puntero tail que nos marca la última posición ocupada del vector, un puntero capacity que nos indica la capacidad máxima del mismo vector.

Tambíen tenemos un puntero que usaremos para hacer callback de las funciones de este vector.

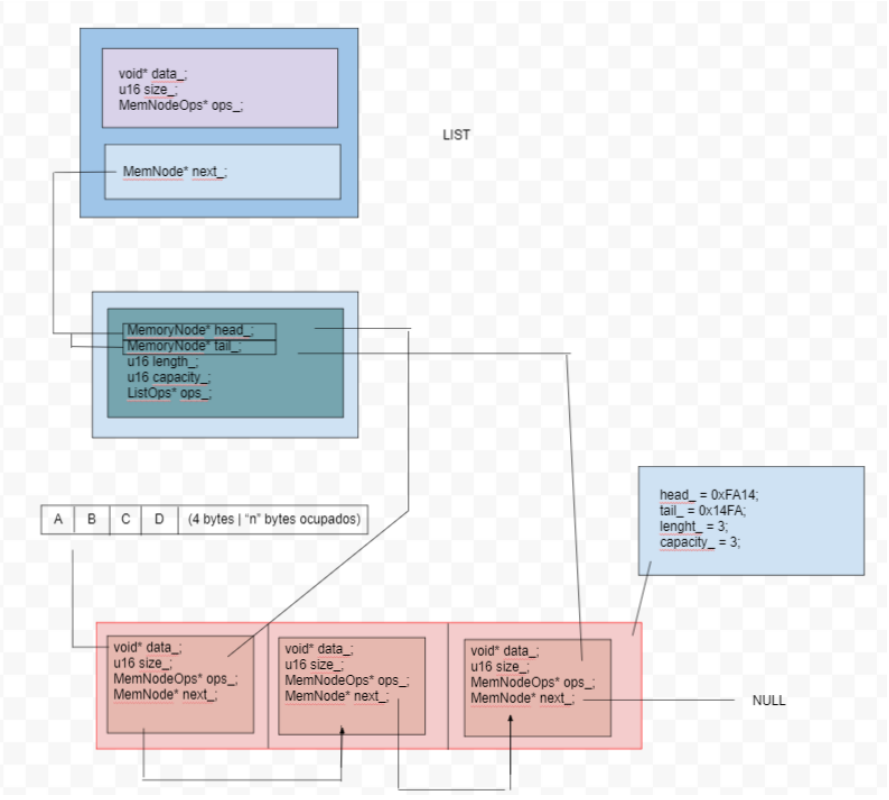


List:

Estamos ante una lista la cual es un conjunto de nodos que, a diferencia del vector, no es memoria contigua si no que se van metiendo en la misma una vez los hayamos insertado.

Tenemos un puntero head, que nos marca el primer elemento de la lista, tail, que nos marca el último elemento ocupado de la misma, capacity que nos indica la capacidad total de la lista, length que nos indica el tamaño de la misma y next que nos permite apuntar al nodo siguiente al cual nos encontramos.

Tambíen tenemos un puntero que usaremos para hacer callback de las funciones de esta lista.

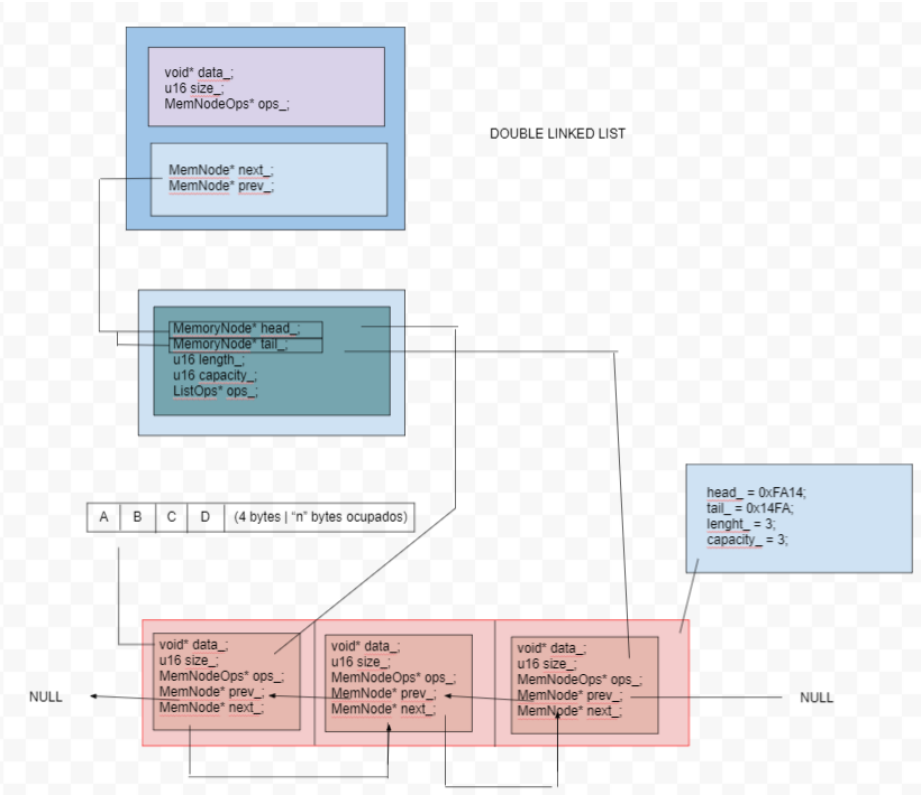


Double List:

Estamos ante una lista la cual es un conjunto de nodos que, a diferencia del vector, no es memoria contigua si no que se van metiendo en la misma una vez los hayamos insertado. a diferencia de la lista simple, esta lista se puede recorrer tanto hacia delante como hacia atrás debido a que tenemos un puntero prev que nos permite ir hacia atrás sin necesidad de hacer el recorrido completo de la misma.

Tenemos un puntero head, que nos marca el primer elemento de la lista, tail, que nos marca el último elemento ocupado de la misma, capacity que nos indica la capacidad total de la lista, length que nos indica el tamaño de la misma y next que nos permite apuntar al nodo siguiente al cual nos encontramos. , y un puntero prev que apunta al nodo anterior al que nos encontramos.

Tambíen tenemos un puntero que usaremos para hacer callback de las funciones de esta lista.



Stack:

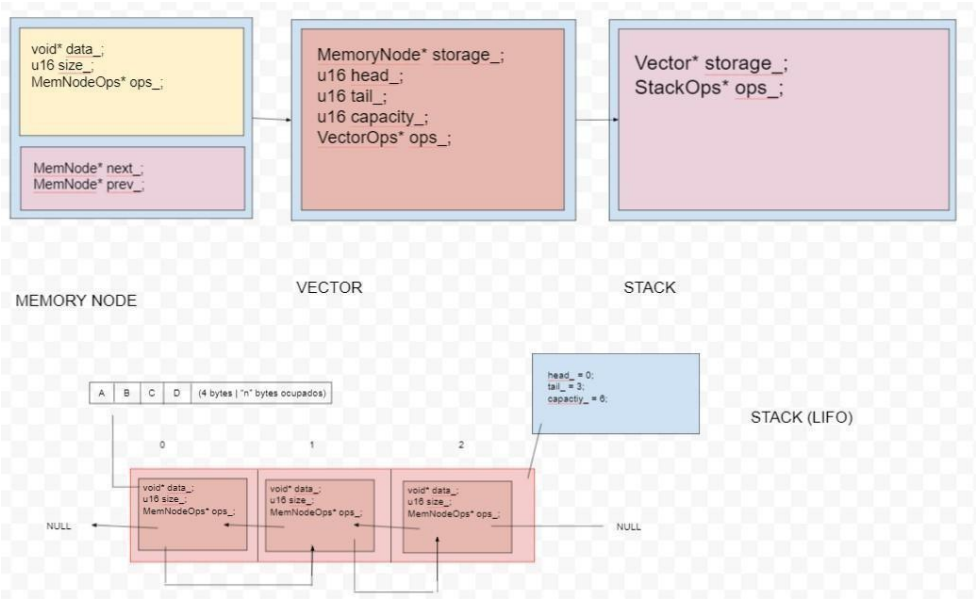
El stack usa como base el vector. Encontramos ciertas diferencias con el vector dado que este Adt va haciendo operaciones descendentes.

Bloque de memoria continua que se insertan y se extraen siempre por la cola.
Los punteros del stack son:

- El Storage_: Es el vector del stack
- Callback: Nos permite usar la función en el propio stack.

El stack, además, consta de tres funciones especiales que lo diferencian del resto:

.Pop -> Es un método que llama, por debajo, al extractLast del Vector.
.Top -> Es un método que llama, por debajo, al last del Vector.
.Push-> Es un método que llama, por debajo, al inserLast del Vector.



2.- Comparativas

A medida que íbamos haciendo los Adts sabíamos qué y porqué x Adt sería más rápido que otro, pero hasta que no pudimos ver los tiempos de esto no nos dimos cuenta de la aplicación que le podríamos encontrar en un futuro.

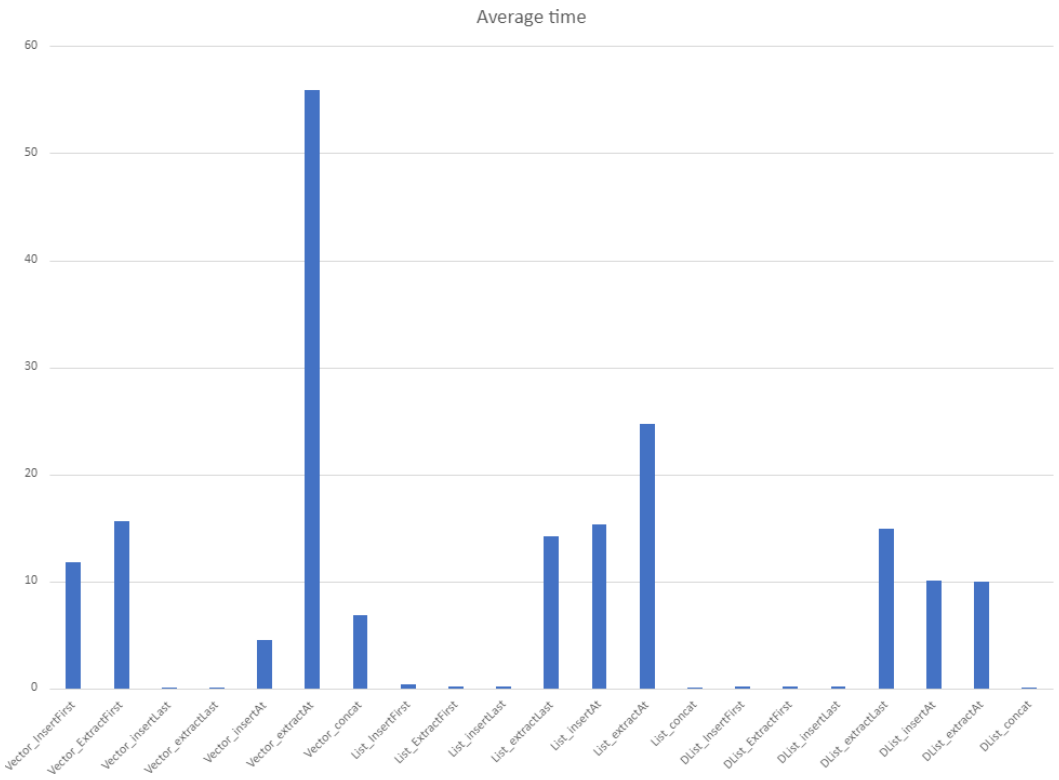
Nuestra hipótesis es que la lista doble sería casi siempre más rápida que el vector o la lista, pero esto no es del todo así y depende como trates el vector y su memoria para ver ciertos cambios en las comparativas.

Los resultados muestran como el vector es tremendamente rápido haciendo un Insert Last dado que este no necesita recorrer todo el vector, llegar al último nodo e insertar. En el Vector podemos ir directamente al nodo requerido por lo que ir al último e insertar información es tremendamente rápido en todos los sentidos, superando a la lista y a la lista doble. Por ende, podemos apreciar que el extract last también es tremendamente rápido en el vector debido a que no necesita recorrer nada, si no que vamos directamente al último nodo.

Tambíen podemos apreciar que el concat del vector es infinitamente más lento que el de la lista, esto es debido a que la lista no necesita crear un vector auxiliar donde posteriormente guardar el vector con la información a concatenar y el otro con el que queremos concatenarlo.

Podemos apreciar que la lista es más rápida que el vector en funciones como el extract first, donde la lista no necesita reposicionar los nodos una vez ha sacado la información del primero, sino que con reajustar los punteros nos vale. El vector, por ende, es más lento dado que necesita reposicionar todos sus nodos una vez ha sacado el primero, y esto lleva un coste que la lista no asume.

En cambio, en la lista doblemente enlazada, funciones como el extract at o el insert at son más rápidas que en los dos anteriores debido, preferentemente, a que esta tiene un puntero prev que nos permite recorrer la lista de alante hacia atrás, y viceversa, por lo que no será necesario recorrer la lista completa en busca del nodo requerido.



3.- Memory Stack

Cada vez que un programa ejecuta una función se está generando una estructura de datos en el Stack. En este Stack se guardan todas nuestras variables locales de cualquier función, y esta va creciendo de manera descendente.

Por otro lado tenemos el Heap, que es donde guardamos todos nuestros punteros, de cualquier tipo, que hayamos usado o guardado en nuestras funciones. como por ejemplo los punteros de los Adt. Este Heap crece de manera ascendente por lo que cuando la pila y este chocan se genera overflow.

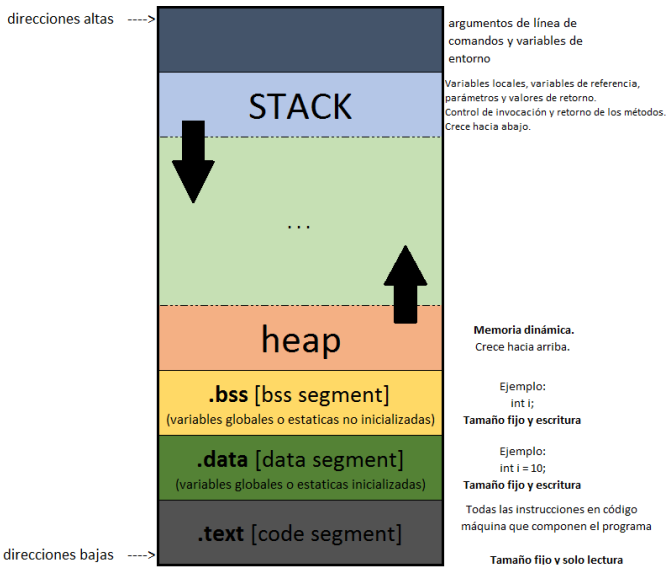
En el bss guardamos todas nuestras variables globales no inicializadas de cualquier tipo, como por ejemplo un float adt;

En el data guardamos todas las variables globales que sí estén inicializadas de manera que podríamos guardar un float adt = 10000.0f;

Por último tenemos el text que es donde se guardan todas las

direcciones de código máquina y bajo nivel del programa en cuestión.

[1] [2]



https://netting.wordpress.com/ - https://netting.wordpress.com/ - https://netting.wordpress.com/

Imagen 1.

4. Bibliografía.

Capítulo 01

Listado de imágenes:

Imagen 01 https://www.google.com/search?q=el+stack+de+un+programa&client=opera-gx&hs=Vjx&sxsrf=ALeKk001B-VsBmDsErMQ6SdgzsG6VnbKhg:1622037584819&tbm=isch&source=iu&ictx=1&fir=8I_8K7QERpPM%252CxmDAeTOFCh8-gM%252C_&vet=1&usg=AI4_-kRt0b05ORxKy80gaRqVvAVJMkSWiA&sa=X&ved=2ahUKEwjAoLjwwOfwAhWnwAIHHRb4CvQQ_h16BAgXEAE#imgrc=8I_8K7QERpPM

Bibliografía:

[1].- https://netting.wordpress.com/2016/10/01/segmentacion-de-memoria-de-un-programa-y-el-stack/#:~:text=Una%20pila%20o%20stack%20es,permite%20almacenar%20y%20recuperar%20datos.

[2].- https://es.wikipedia.org/wiki/Pila_de_llamadas