|  |  |  |
| --- | --- | --- |
|  | **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación

salas A y B

|  |  |
| --- | --- |
| *Profesor:* | M. I. Marco Antonio Martínez Quintana |
| *Asignatura:* | Estructuras de Datos y Algoritmos de programación I |
| *Grupo:* | 17 |
| *No de Práctica(s):* | 4 |
| *Integrante(s):* | Marcos Vega Alvarez |
| *No. de Equipo de cómputo empleado:* | 25 |
| *No. de Lista o Brigada:* |  |
| *Semestre:* | 2020-2 |
| *Fecha de entrega:* | 01 de Marzo de 2020 |
| *Observaciones:* |  |
|  |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_\_

**Objetivo:**

Utilizarás funciones en lenguaje *C* que permiten reservar y almacenar información de manera dinámica (en tiempo de ejecución).

## Introducción:

## Tipos de Memoria

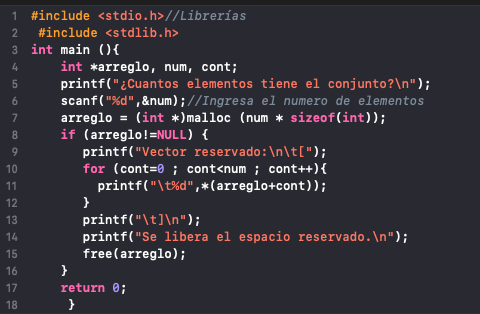
**La memoria Estética**

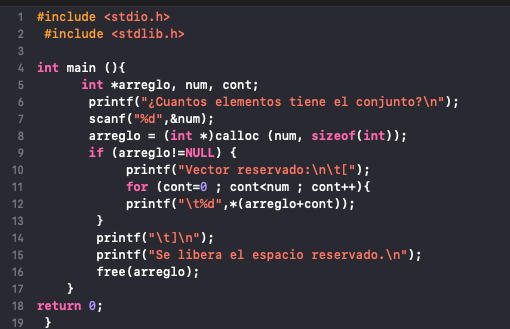
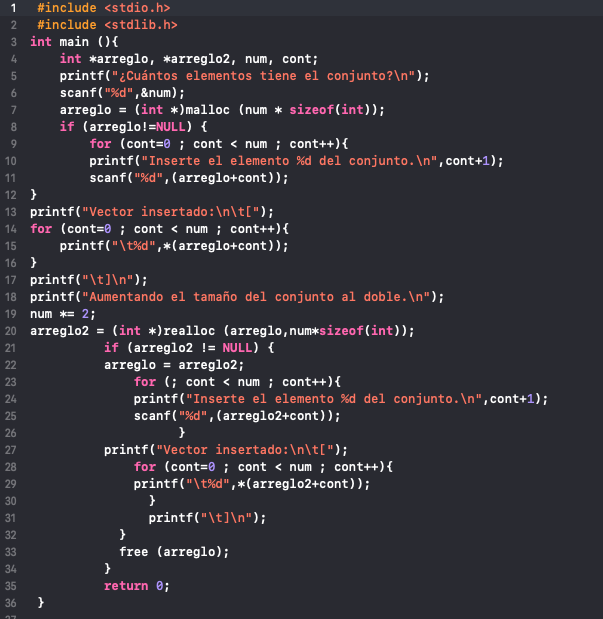
* Sin importa nada, ocupa todo el espacio de las variables que declaramos, no le importa si las usamos.

**Memoria Dinámica**

* Puede hacerse más más compleja y dar muchos errores.
* Da la capacidad de modificar tamaño de variables.
* SIEMPRE se usan punteros.
* Es mucho más lento un programa que la implemente pues ocupa más instrucciones.

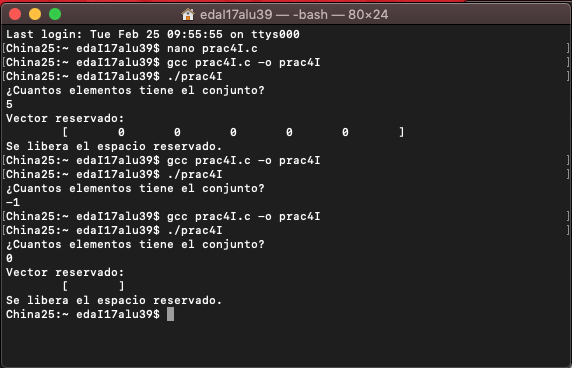
Desarrollo:



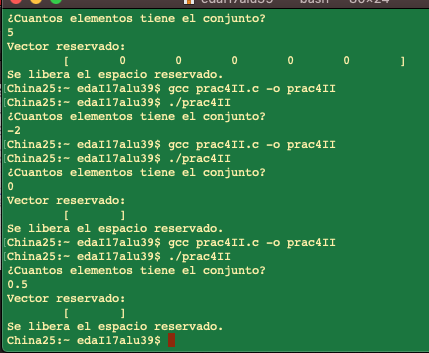


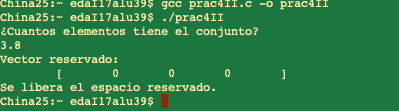
Resultados:

Código 1:

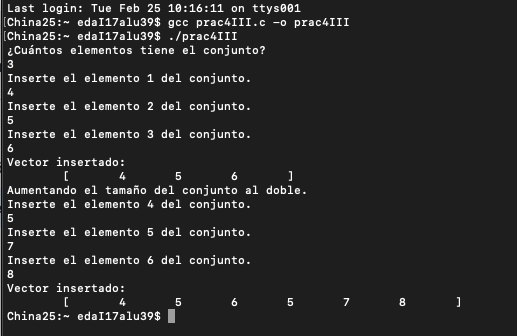


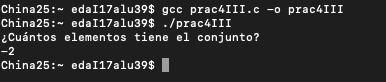
Código 2:

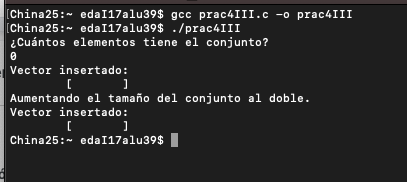


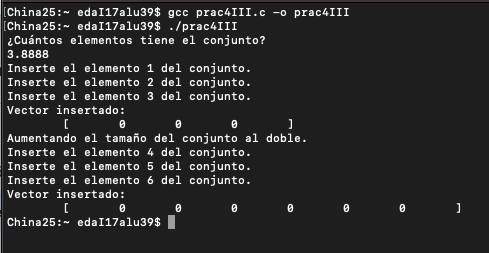


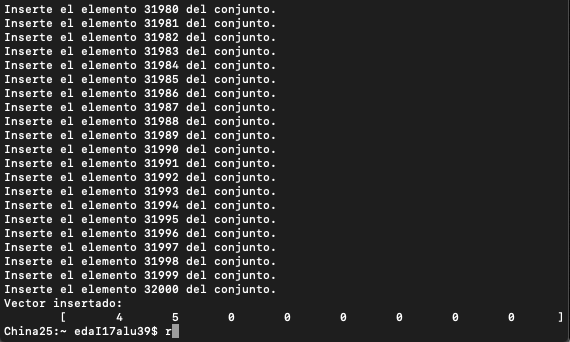
Código 3:











Código 1

Se declaran las dos bibliotecas a usar, usamos variables tradicionales, apuntador y dos enteros con un sacn se guardan los elementos del conjunto se le asigna al arreglo entero el tipo del dato que necesita (pues se castea ode void a int) Seguido de malloc y en los parametros num y sizeo (que regresa el tamaño del entero. Recorre con arreglo el lugar de memoria que se le asigna

Cuando no se puede reservar en memoria regresa a vacío.

En el primer caso nos arroja datos basura o lo qu e sea que se tenga en ese momento en el espacio de memoria.

En el segundo caso  no existe un espacio negativo y al ingresar cero nos regresa un vacío.

Código 2:

Calloc inicializa en 0 cambia su sintaxis y los valores los cambio con un cero.

Primero el tamaño y luego el tamaño de cada espacio, hace lo mismo pero imprime el apuntador mas el contador.

En el primer caso de los niumeros enteros positivos nos rgresan puros ceros, con negativos no se puede y con cero nos regresa un espacio vacío y con decimales redondea al numero que está antes del punto.

Código 3:

Realloc con él se redimensiona el tamaño de esa memoria dinámica que reservamos lleva un apuntador void y el segundo es el nuevo tamaño que le vamos a dar, sólo se ocupa después de reservar memoria con  malloc y calloc, ya que si aún no se han reservado memoria no se puede.

Dos variables enteros apuntador, pedimos los elementos y se guardan usando malloc usando el tamaño del usuario, con el for se insertan los valores dentro de ese espacio de memoria, se considera arreglo porque entra a cada una de las secciones.

Con realloc se le asigna lo que ya existe más el nuevo, no borra el anterior free (arreglo) libera memoria.

Conclusiones:

La **asignación dinámica de memoria en el**[**Lenguaje de programación C**](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)), se realiza a través de un grupo de funciones en la [biblioteca estándar de C](https://es.wikipedia.org/wiki/Biblioteca_est%C3%A1ndar_de_C) , es decir, malloc , realloc , calloc y free . En [C++](https://es.wikipedia.org/wiki/C%2B%2B), se incluyen estas funciones por [retrocompatibilidad](https://es.wikipedia.org/wiki/Retrocompatibilidad), pero han sido sustituidas en gran parte por los operadores new y new[].

Están disponibles muchas implementaciones diferentes del mecanismo de asignación de memoria real, utilizado por malloc . Su rendimiento varía tanto en tiempo de ejecución y memoria requerida.

El [lenguaje de programación C](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)) gestiona la memoria de forma estática, automática o dinámica. Las variables estáticas se asignan en la memoria principal, por lo general junto con el código ejecutable del programa, y persisten durante toda la vida del programa; las variables automáticas se asignan sobre la pila ([stack](https://es.wikipedia.org/wiki/Pila_(inform%C3%A1tica))), comienzan cuando se invocan las funciones y acaban cuando se llama a return. Para las variables estáticas y automáticas se requiere que el tamaño de la asignación sea constante en [tiempo de compilación](https://es.wikipedia.org/wiki/Tiempo_de_compilaci%C3%B3n) (antes de [C99](https://es.wikipedia.org/wiki/C99#C99) , que permite "arrays" automáticos de longitud variable). Si el tamaño requerido no se conoce hasta el [tiempo de ejecución](https://es.wikipedia.org/wiki/Tiempo_de_ejecuci%C3%B3n) (por ejemplo, si los datos de tamaño arbitrario se están leyendo del usuario o desde un archivo de disco), la utilización de objetos de datos de tamaño fijo es insuficiente.

La vida útil de la memoria asignada es también una preocupación. Ni la memoria estática ni automática es adecuada para todas las situaciones. Los datos automáticos asignados no persisten en varias llamadas de función, mientras que los datos estáticos persisten durante toda la vida del programa, sean o no necesarios. En muchas situaciones, el programador requiere una mayor flexibilidad en la gestión de la vida útil de la memoria asignada.

Estas limitaciones se evitan mediante el uso de la [gestión de memoria](https://es.wikipedia.org/w/index.php?title=Asignaci%C3%B3n_de_memoria_din%C3%A1mica&action=edit&redlink=1) en la que la memoria es más explícitamente (y más flexiblemente) manejada, típicamente mediante la asignación desde el montón ([heap](https://es.wikipedia.org/wiki/Heap)), un área de memoria estructurada para este fin. En C, la función malloc, perteneciente a la cabecera stdlib.h, se utiliza para asignar un bloque de memoria en el montón. El programa accede a este bloque de memoria a través de un puntero que malloc regresa. Cuando ya no se necesita la memoria, se pasa el puntero a la función free, la cual libera la memoria de modo que se puede utilizar para otros fines.

Referencias:

<https://es.wikipedia.org/wiki/Malloc>

<https://compilandoconocimiento.com/2016/12/24/memoria-dinamica/>