**Objetivo.**

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

**Introducción.**

Cuando creamos una variable de cualquier tipo, ya sea un entero, un arreglo o una estructura, se le esta reservando una determinada cantidad de memoria en la que se van a almacenar los valores de dichas variables. La cantidad de memoria que se reserva depende de en un principio de dos factores, el tipo de dato que guarda la nueva variable, y la cantidad de bytes que el procesador le da a cada tipo. Es por esto, al ejecutarse grandes programas en los que se maneja muchas variables al mismo tiempo, podamos quedarnos sin memoria para seguir trabajando o creando más variables.

Esta es la razón por la que existe la asignación dinámica de memoria. La asignación de esta memoria se lleva a cabo mediante apuntadores, que me dan información de donde se encuentra la memoria que he reservado.

La memoria dinámica se refiere al espacio de almacenamiento que se reserva en tiempo de ejecución, debido a que su tamaño puede variar durante la ejecución del programa. Su principal ventaja frente a la estática, es que su tamaño puede variar durante la ejecución del programa.

El uso de memoria dinámica es necesario cuando a priori no se conoce el número de datos y/o elementos que se van a manejar.

***Memoria dinámica.***

Dentro de la memoria RAM, la memoria reservada de forma dinámica está alojada en el heap o almacenamiento libre y la memoria estática (como los arreglos o las variables primitivas) en el stack o pila.

La pila generalmente es una zona muy limitada. El heap, en cambio, en principio podría estar limitado por la cantidad de memoria disponible durante la ejecución del programa y el máximo de memoria que el sistema operativo permita direccionar a un proceso.

El montículo o heap (también llamado segmento de datos) es un medio de almacenamiento con más capacidad que la pila y que puede almacenar datos durante toda la ejecución de las funciones. Las variables globales y estáticas viven en el heap mientras la aplicación se esté ejecutando.

Para acceder a cualquier dato almacenado dentro del heap se debe tener una referencia o apuntador en la pila.

Para acceder a cualquier dato almacenado dentro del heap se debe tener una referencia o apuntador en la pila.

El almacenamiento dinámico puede afectar el rendimiento de una aplicación debido a que se llevan a cabo arduas tareas en tiempo de ejecución: buscar un bloque de memoria libre y almacenar el tamaño de la memoria asignada. Lenguaje C permite el almacenamiento de memoria en tiempo de ejecución a través de tres funciones: malloc, calloc y realloc, en lo cual en la librería stdlib.h encontramos las funciones

Desarrollo.

**Malloc**

La función malloc permite reservar un bloque de memoria de manera dinámica y devuelve un apuntador tipo void. Su sintaxis es la siguiente:

void \*malloc(size\_t size);

La función recibe como parámetro el número de bytes que se desean reservar. En caso de que no sea posible reservar el espacio en memoria, se devuelve el valor nulo (NULL).

**Free**

El almacenamiento en tiempo de ejecución se debe realizar de manera explícita, es decir, desde la aplicación se debe enviar la orden para reservar memoria. Así mismo, cuando la memoria ya no se utilice o cuando se termine el programa se debe liberar la memoria reservada. La función free permite liberar memoria que se reservó de manera dinámica. Su sintaxis es la siguiente:

void free(void \*ptr);

El parámetro ptr es el apuntador al inicio de la memoria que se desea liberar. Si el apuntador es nulo la función no hace nada.

*Código (malloc)*

#include<stdio.h>

#include<stdlib.h>

int main()

{

int \*arreglo, num, cont;

printf("¿Cuantos elementos tiene el conjunto?\n");

scanf("%d", &num);

arreglo=(int \*)malloc (num \* sizeof(int));

if(arreglo!=NULL)

{

printf("Vector reservado:\n\t[");

for (cont=0; cont<num; cont++)

printf("\t%d",\*(arreglo+cont));

printf("\t]");

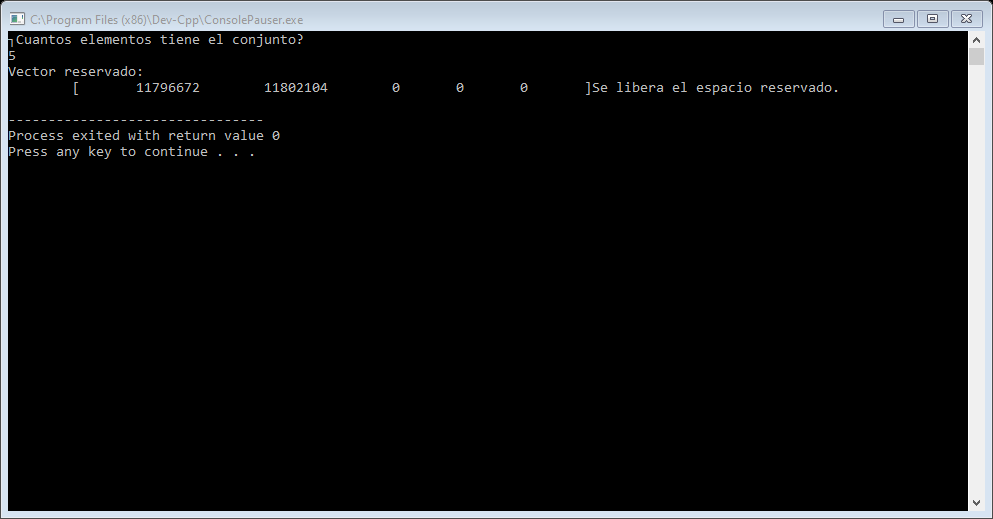
printf("Se libera el espacio reservado.\n");

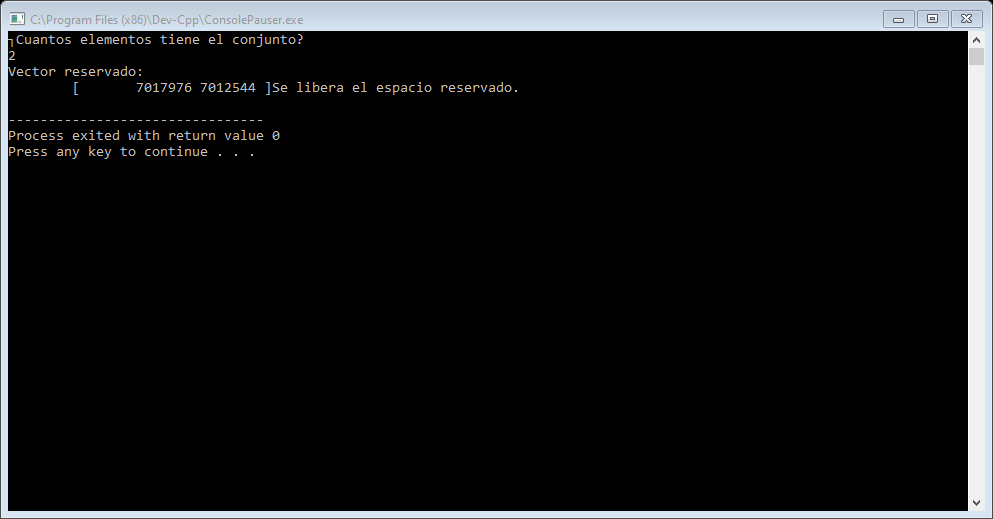
free(arreglo);

}

return 0;

}





Nos muestra el espacio reservado a partir de los elementos que tenga el conjunto, eso por el apuntador que se ocupó para asignarle los espacios solicitados por el usuario, y a su vez devuelve el espacio de memoria de la zona reservada de la memoria.

Y por último con la función free, libera la memoria que se reservó utilizado previamente.

**Calloc**

La función calloc funciona de manera similar a la función malloc pero, además de reservar memoria en tiempo real, inicializa la memoria reservada con 0. Su sintaxis es la siguiente:

void \*calloc (size\_t nelem, size\_t size);

El parámetro nelem indica el número de elementos que se van a reservar y size indica el tamaño de cada elemento.

*Código (calloc)*

#include<stdio.h>

#include<stdlib.h>

int main()

{

int \*arreglo, num, cont;

printf("¿Cuantos elementos tiene el conjunto?\n");

scanf("%d", &num);

arreglo=(int \*)calloc (num, sizeof(int));

if(arreglo!=NULL)

{

printf("Vector reservado:\n\t[");

for (cont=0; cont<num; cont++)

printf("\t%d",\*(arreglo+cont));

printf("\t]");

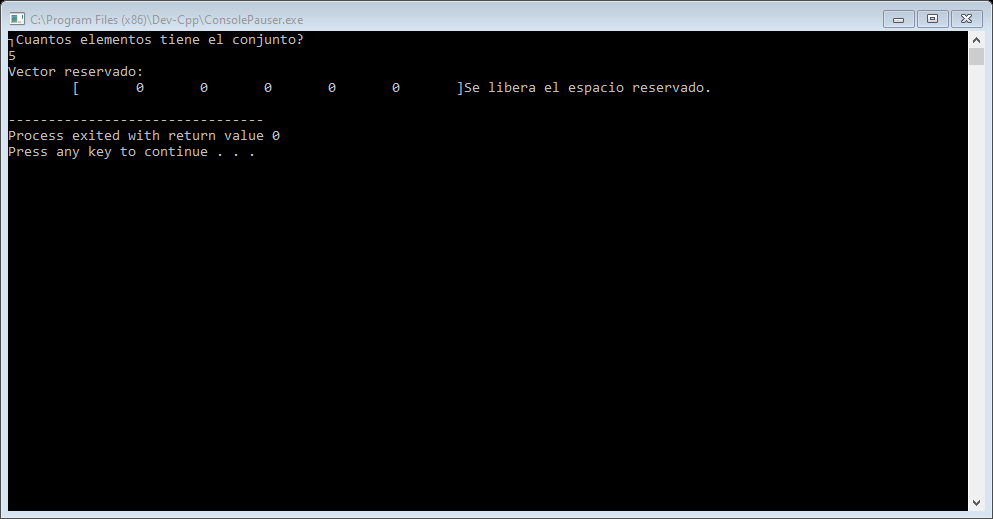
printf("Se libera el espacio reservado.\n");

free(arreglo);

}

return 0;

}



Se muestra que al momento de indicar cuantos elementos contiene el conjunto, el programa nos arroja el numero dado a partir del espacio reservado de memoria inicializando con ceros, sin importar la cantidad de elementos todos espacio reservado será cero.

Nuevamente nos encontramos con free, que libera la memoria reservada utilizada previamente.

**Realloc**

La función realloc permite redimensionar el espacio asignado previamente de forma dinámica, es decir, permite aumentar el tamaño de la memoria reservada de manera dinámica. Su sintaxis es la siguiente:

void \*realloc (void \*ptr, size\_t size);

Donde ptr es el apuntador que se va a redimensionar y size el nuevo tamaño, en bytes, que se desea aumentar al conjunto.

Si el apuntador que se desea redimensionar tiene el valor nulo, la función actúa como la función malloc. Si la reasignación no se pudo realizar, la función devuelve un apuntador a nulo, dejando intacto el apuntador que se pasa como parámetro (el espacio reservado previamente).

Código (realloc)

#include<stdio.h>

#include<stdlib.h>

int main()

{

int \*arreglo, \*arreglo2, num, cont;

printf("¿Cuantos elementos tiene el conjunto?\n");

scanf("%d", &num);

arreglo=(int \*)malloc (num \* sizeof(int));

if(arreglo!=NULL)

{

for (cont=0; cont<num; cont++)

{

printf("Inserte el elemento %d del conjunto.\n");

scanf("%d", (arreglo+cont));

}

printf("Vector insertado:\n\t[");

for (cont=0; cont<num; cont++)

printf("\t%d",\*(arreglo+cont));

printf("\t]");

printf("Aumentando el tamaño del conjunto al doble.\n");

num \*=2;

arreglo2=(int \*)realloc (arreglo,num\*sizeof(int));

if(arreglo2!=NULL)

{

arreglo=arreglo2;

for(; cont<num; cont++)

{

printf("Inserte el elemento %d del conjunto.\n", cont+1);

scanf("%d",(arreglo2+cont));

}

printf("Vector insertado:\n\t[");

for (cont=0; cont<num; cont++)

printf("\t%d", \*(arreglo2+cont));

printf("\t]");

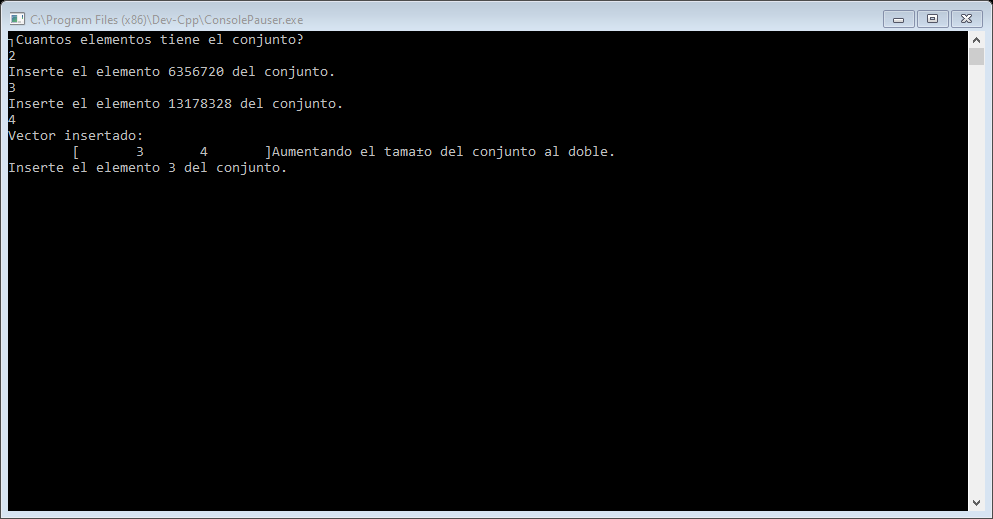
}

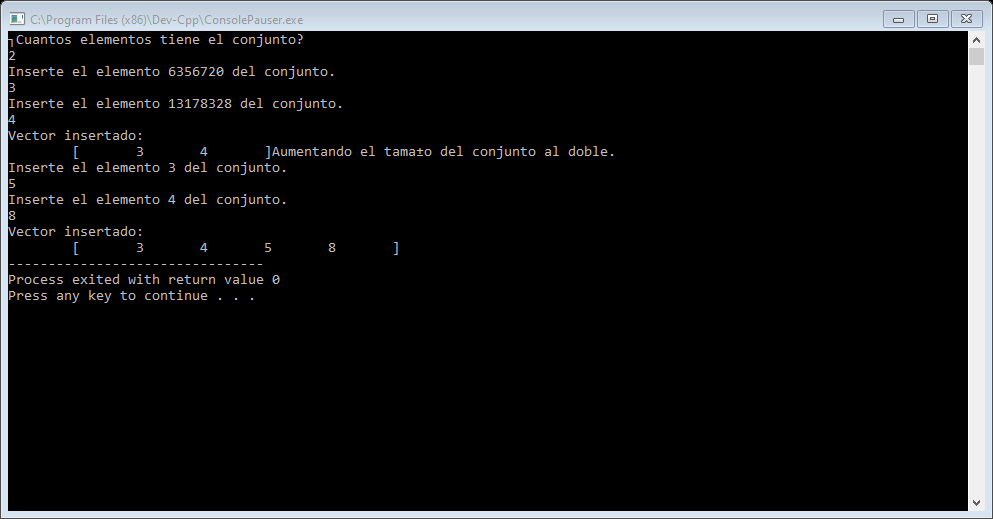
free(arreglo);

}

return 0;

}





Como se observa en el ejecutable, este código cuenta con algunas funciones mencionadas anteriormente, donde malloc nos reserva espacio de la memoria a partir del apuntador tantas veces que el usuario lo requiera, y posteriormente se incrementan los espacios al doble que el usuario índico al principio. Pero se debe tomar en cuenta que al principio se le inserta el espacio de memoria un número, y después sigue la sucesión de la cantidad de elementos.

Finalmente con free, libera el espacio reservado que se ocupó durante el ejecutable.

Conclusión.

En esta práctica se observó y se dio a conocer funciones en lo cual son de mucha utilidad para el uso de memoria en un programa, dando un espacio reservado que se va ocupar sin tener que estar ocupando todo, sino lo necesario, a su vez liberando o aumentado el espacio.

Personalmente me dio un panorama más de los usos que puedo implementar para un mejor desarrollo de programas, capaz de poder manejar una memoria dinámica utilizando lo que quiero, eso a vez dando una buena imagen en el código.

Bibliografía.

Jaime. (2015). *Memoria dinámica en C.* Febrero 29,2020, de memoria dinámica Sitio web: <https://programaciondummies.wordpress.com/2015/01/23/memoria-dinamica-en-c/>

Wikilibros. (2019*). Manejo dinámico de memoria*. Febrero 29,2020, de memoria dinámica Sitio web: <https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C/Manejo_din%C3%A1mico_de_memoria>