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

Laboratorios de computación

salas A y B

|  |  |
| --- | --- |
| *Profesor:* | M.I. Marco Antonio Martinez Quintana |
| *Asignatura:* | estructuras de datos y algoritmos |
| *Grupo:* | 17 |
| *No de Práctica(s):* | 3 |
| *Integrante(s):* | Villanueva Corona Miguel Angel |
| *No. de Equipo de cómputo empleado:* | 36 |
| *No. de Lista o Brigada:* |  |
| *Semestre:* | 2020-2 |
| *Fecha de entrega:* | 1 de marzo del 2020 |
| *Observaciones:* |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_\_

* **Objetivos**

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

* **Introducción**

**Memoria dinámica**

**Malloc**

Por medio de punteros se puede reservar o liberar memoria dinámicamente, es decir, según se necesite. Para ello existen varias funciones estándares, de la biblioteca **<stdlib.h>**

La función **malloc** sirve para solicitar un bloque de memoria del tamaño suministrado como parámetro. Devuelve un puntero a la zona de memoria concedida:

**void\* malloc ( unsigned numero\_de\_bytes );**

El tamaño se especifica en bytes. Se garantiza que la zona de memoria concedida no está ocupada por ninguna otra variable ni otra zona devuelta por **malloc**.

Si **malloc** es incapaz de conceder el bloque, devuelve un puntero nulo.

**Free**

Cuando una zona de memoria reservada con **malloc** ya no se necesita, puede ser *liberada*mediante la función **free**.

**void free (void\* ptr);**

**ptr** es un puntero de cualquier tipo que apunta a un área de memoria reservada previamente con **malloc**.

Si **ptr** apunta a una zona de memoria indebida, los efectos pueden ser desastrosos, igual que si se libera dos veces la misma zona.

**Calloc**

Es similar a la función **malloc** con la diferencia de que:

* Inicializa la memoria reservada a 0 (cero)
* La sintaxis de la función es distinta, en lugar de recibir la cantidad de bytes recibe la cantidad de "slots" y el tamaño del tipo de dato.

Su sintaxis es:

Void\*calloc(size\_t nmemb, size\_t size);

**Realloc**

Esta función se utiliza para, dado un espacio de memoria, del cual tenemos un puntero a él, redimensionarlo, ya sea para hacerlo más chico o más grande.

La sintaxis de la función es:

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

* **Desarrollo**

**Codigos**

**Código 1**

#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]\n");

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

free(arreglo);

}

return 0;

}

**Código 2**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int \*arreglo, num, cont;

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

scanf("%d",&num);

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

if(arreglo!=NULL)

{

printf("vector reservado");

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

{

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

}

printf("\t[\n");

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

free(arreglo);

}

return 0;

}

**Código 3**

#include<stdio.h>

#include<stdlib.h>

//Programa elaborado por

int main(){

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

printf("%cCu%cntos elementos tiene el conjunto%c\n",168,160,63);

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",cont+1);

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

}

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

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

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

}

printf("\t]\n");

printf("Aumentando el tama%co del conjunto doble.\n",164);

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]\n");

}

free (arreglo);

}

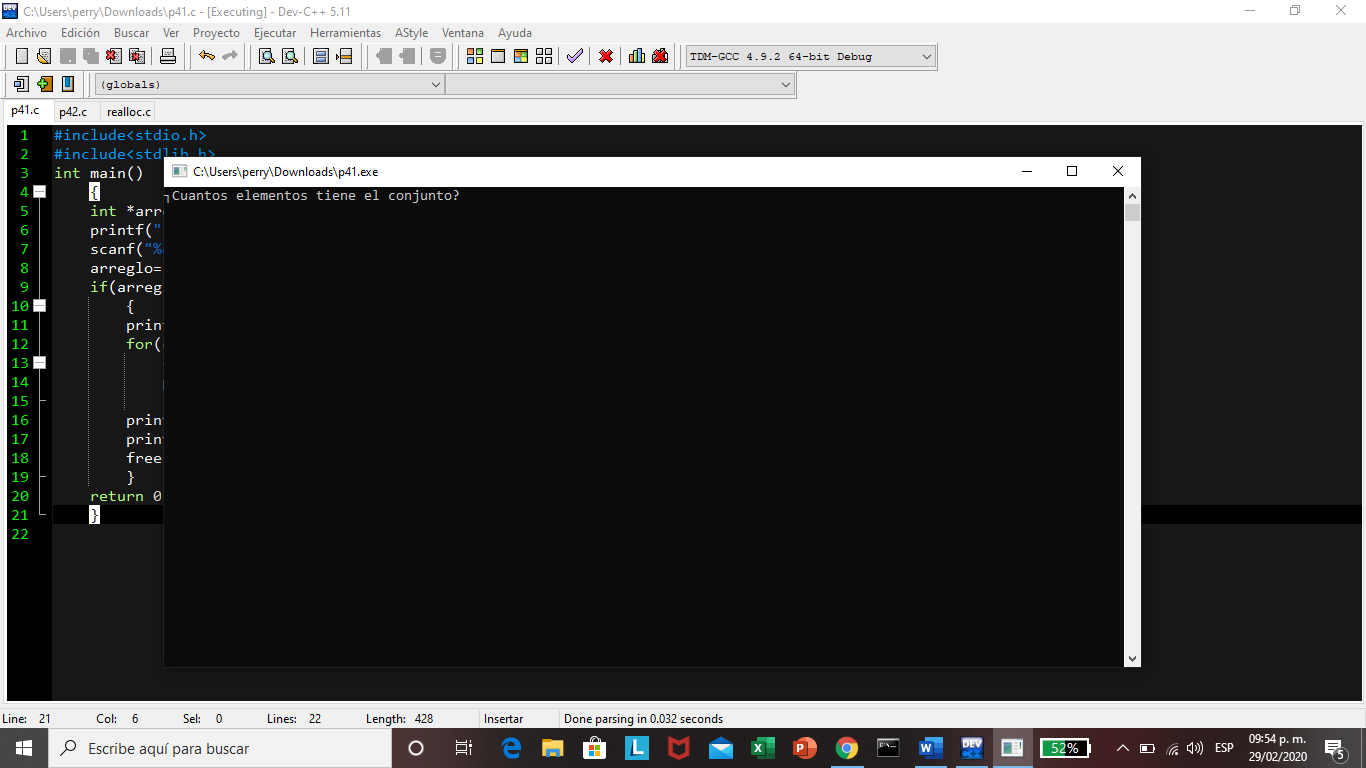
return 0;

}

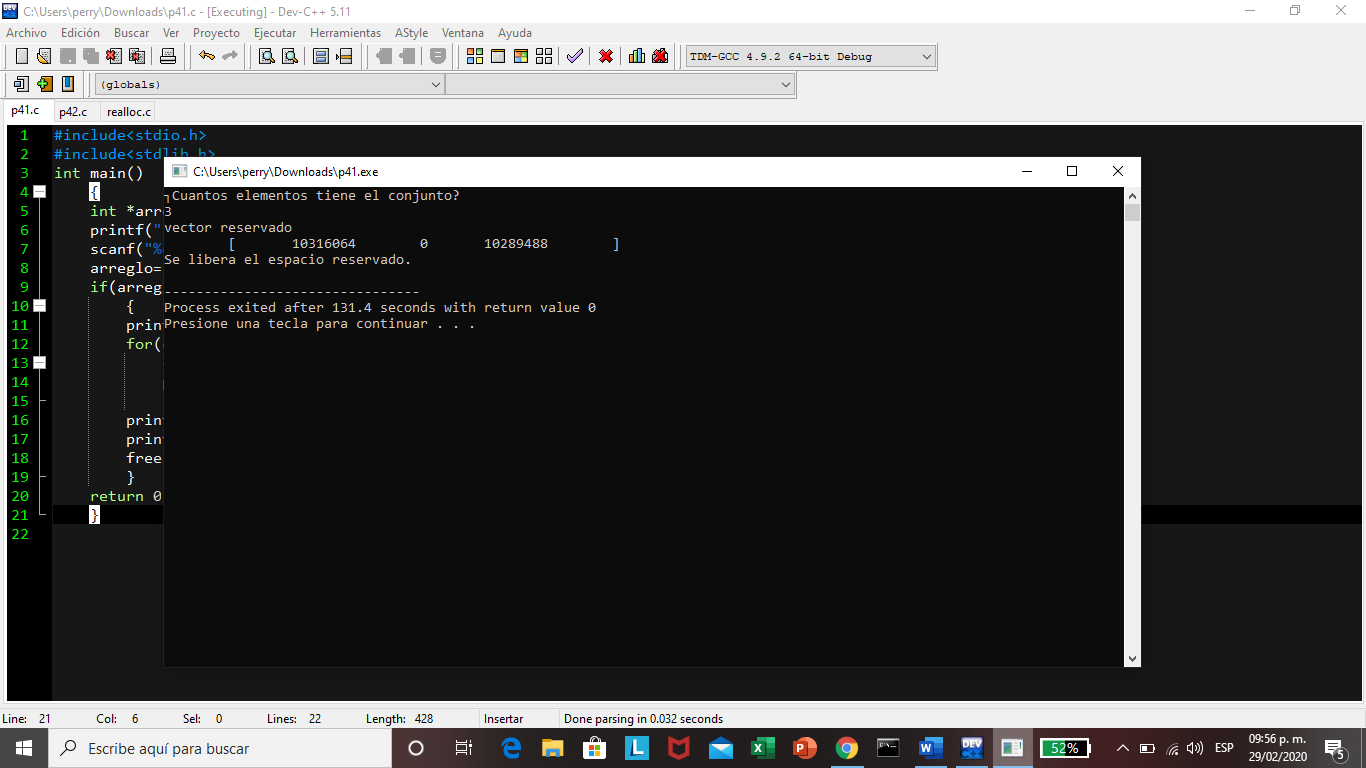
**Explicación de los códigos**

**Código 1**

Al ejecutar el código te pide los elementos que tiene el conjunto



Al ingresar el numero de elementos el malloc reserva un espacio en la memoria y lo muestra en un vector

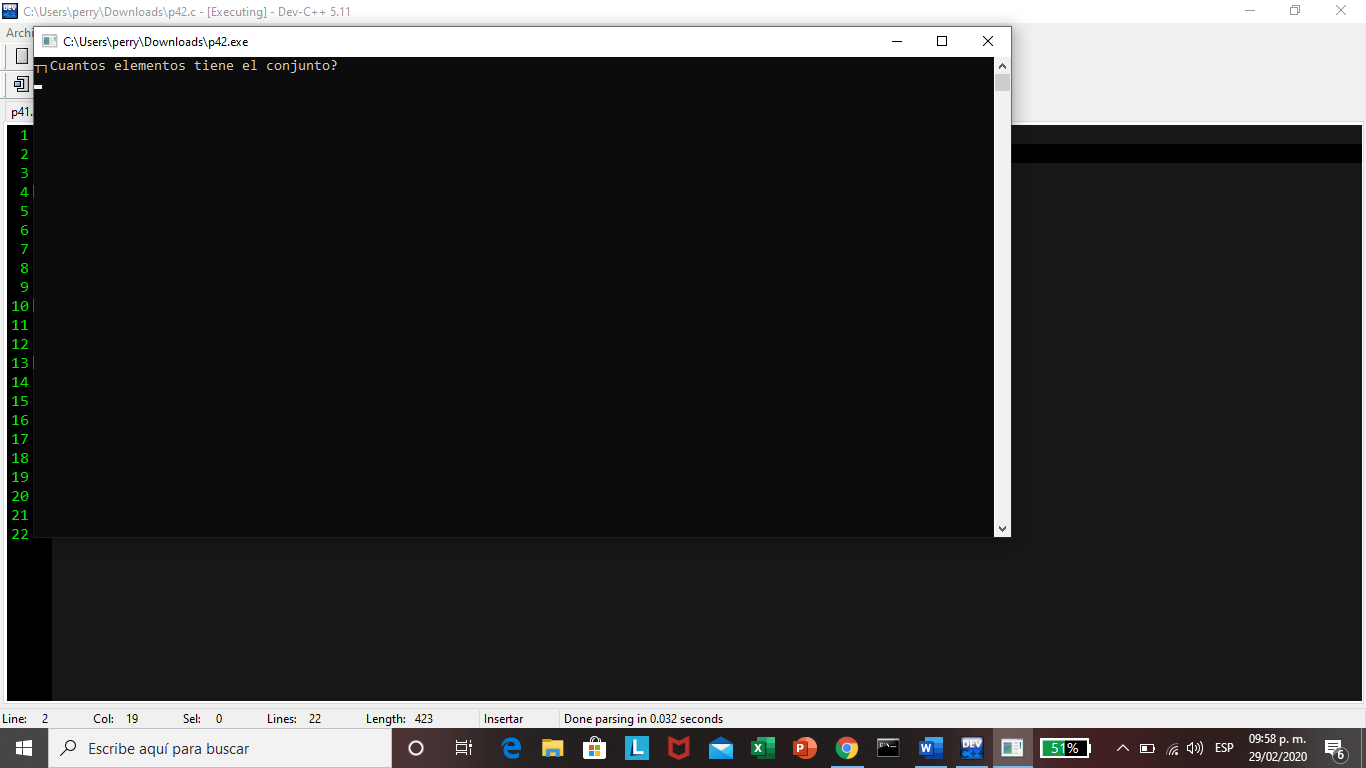


Por ultimo el free libera el espacio almacenado por el malloc

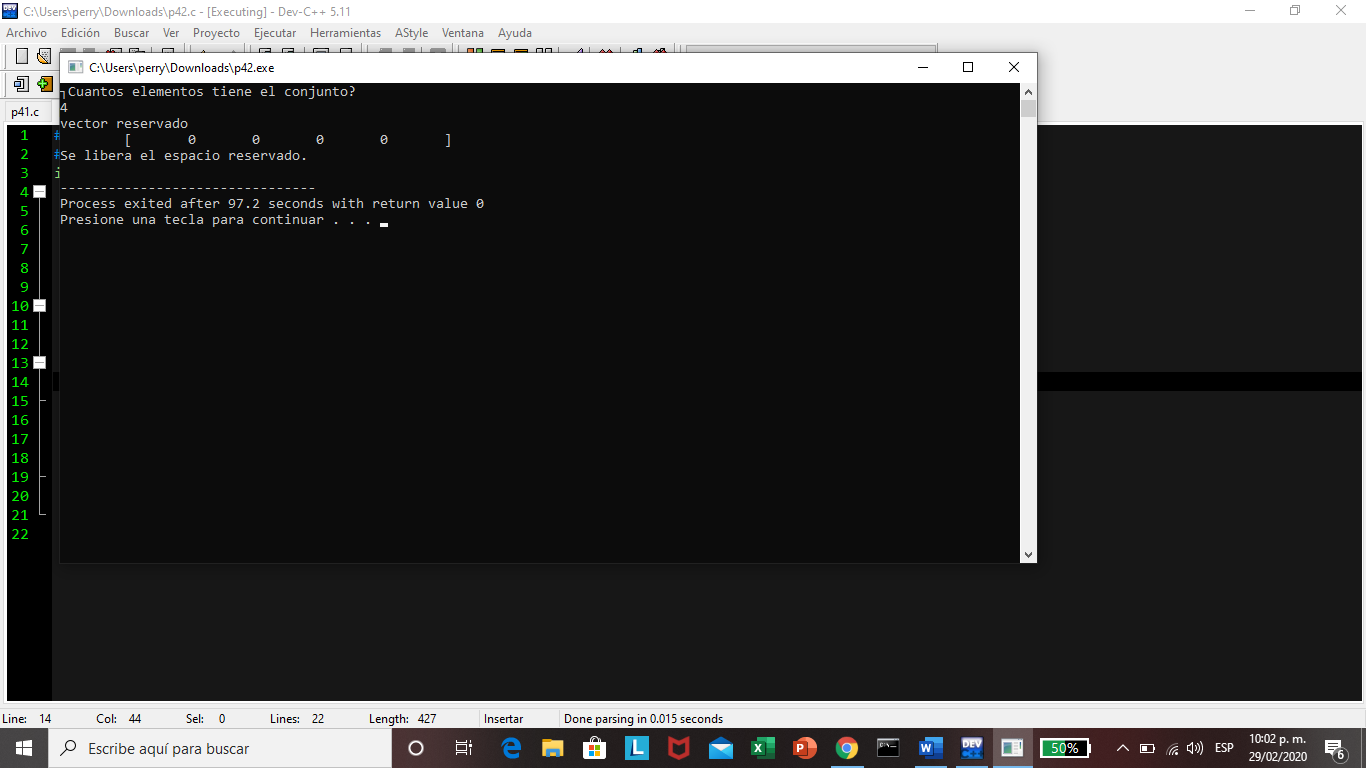
**Código 2**

Este código es similar al pasado ya que tanto malloc como calloc realizan un almacenamiento en la memoria a diferencia que este inicializa desde cero.

El código le solicita al usuario los elementos del conjunto a reservaren la memoria



Después el código lo almacena en la memoria con calloc y lo mustra en un vector con el numero de elementos seleccionados los cuales se inicializan en cero siempre.

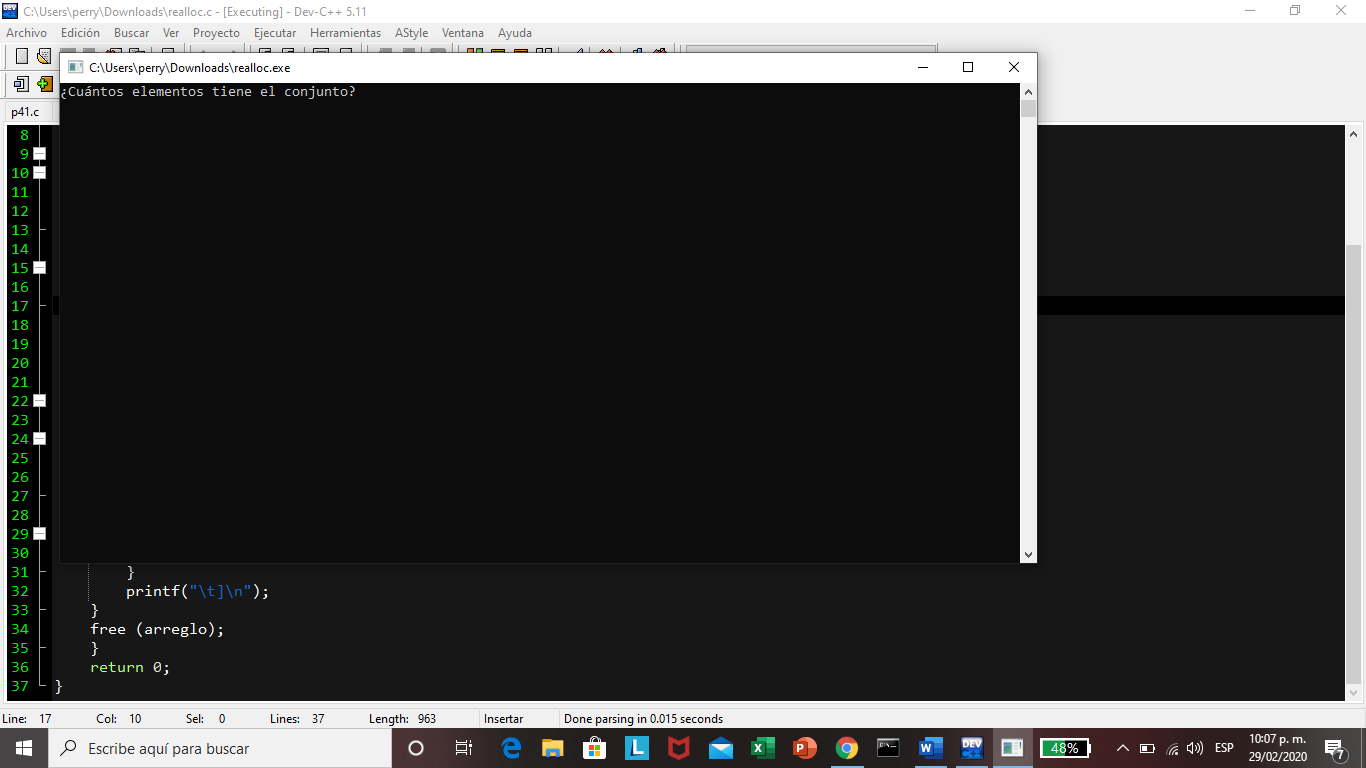


al terminar la ejecución del programa se libera el espacio reservado de la memoria con ayuda de un free

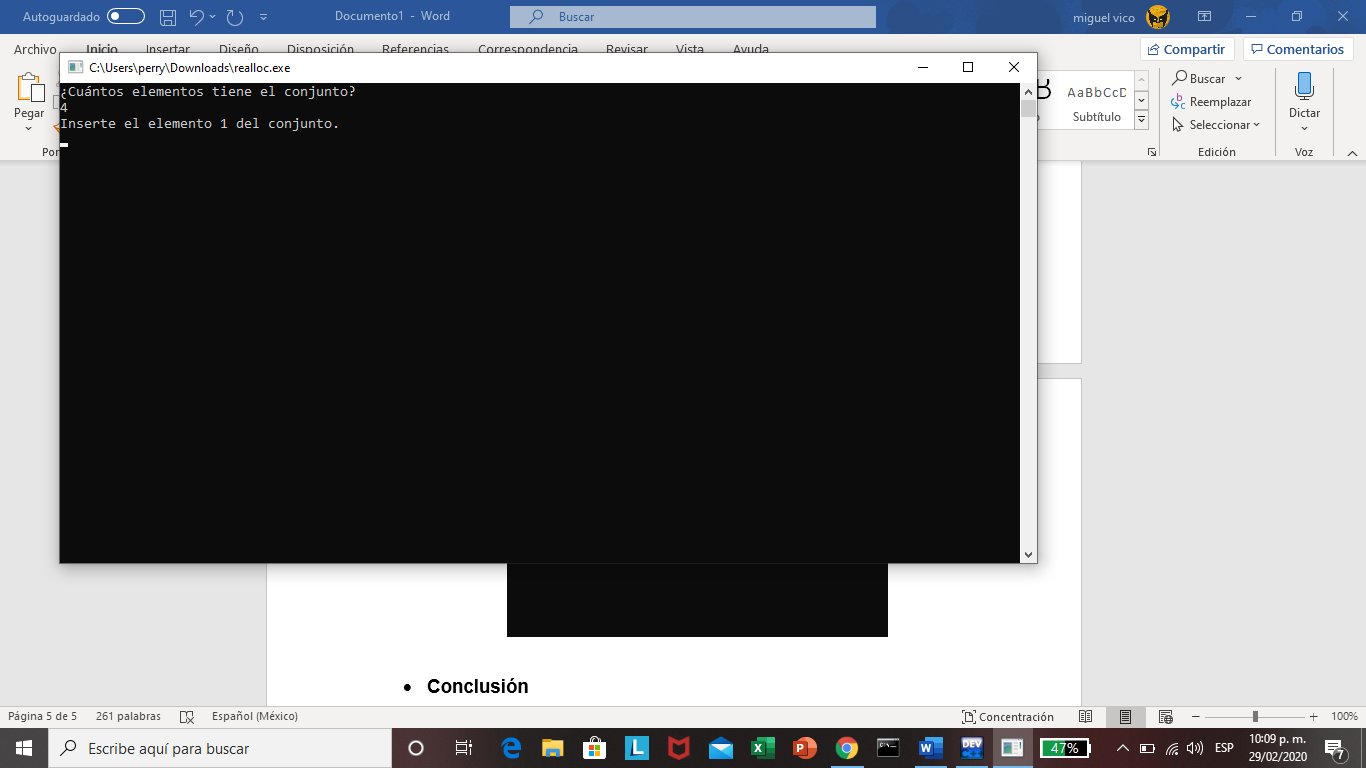
**Código 3**

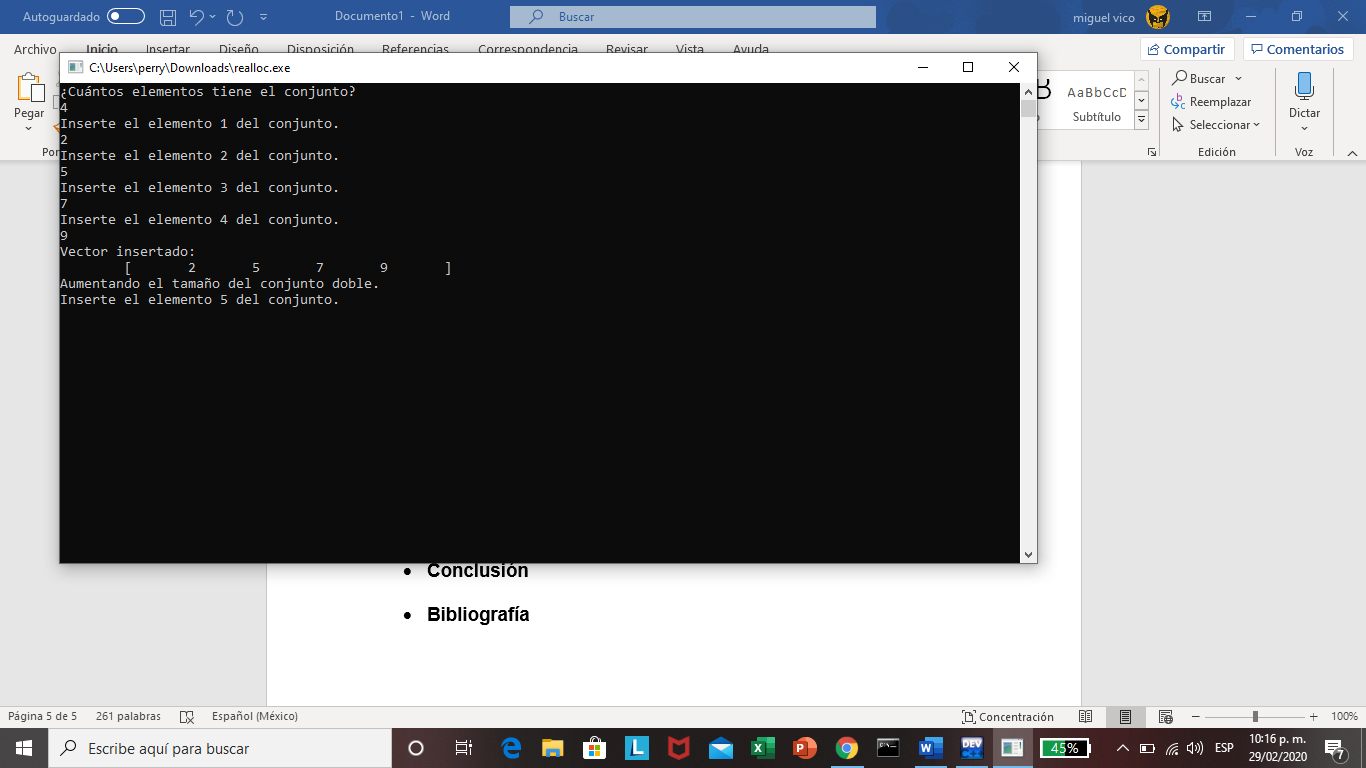
Este código es la extencion del primer código solo que se agrega la función realloc que lo que realiza esta función es la extencion o disminución de la reservación de la memoria.

Al iniciar el código pide al usuario que ingrese los elementos del conjunto que dece

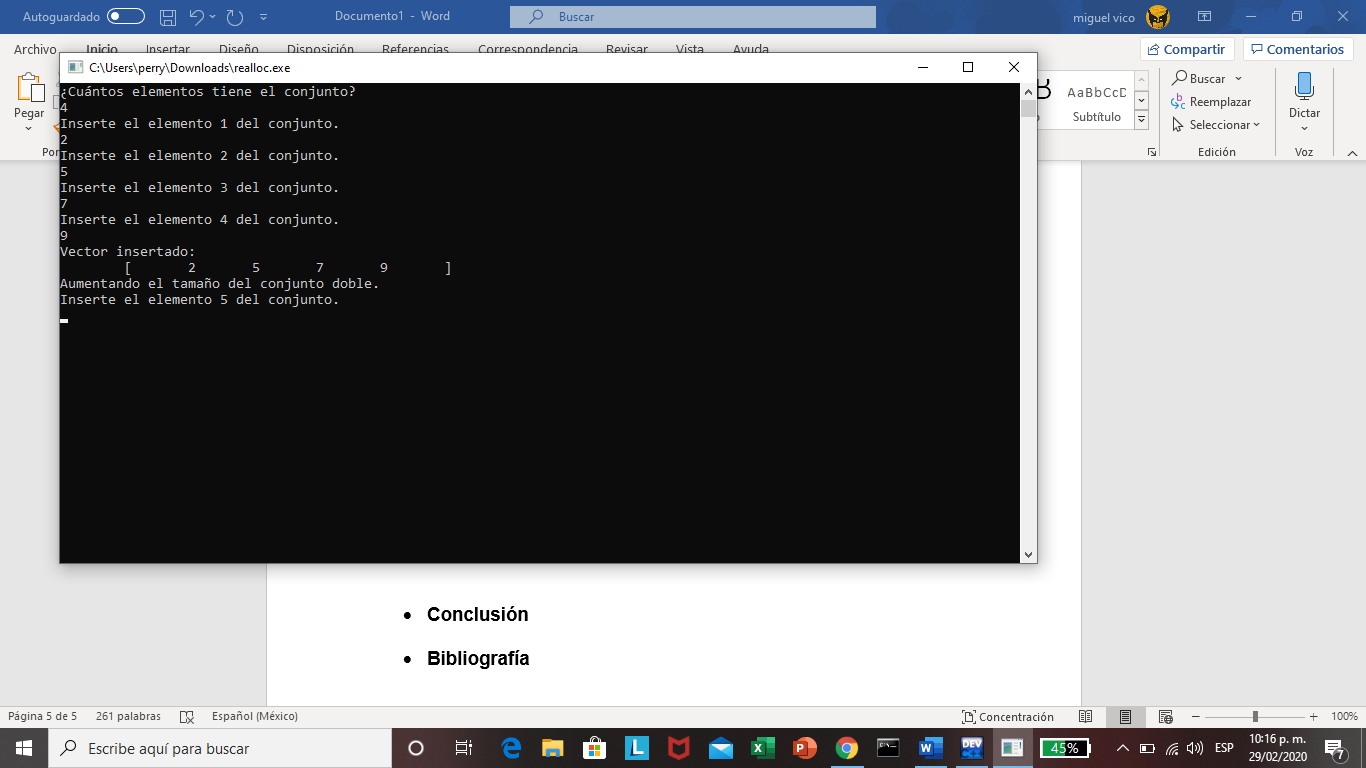


Después te pide que ingreses los elementos uno por uno

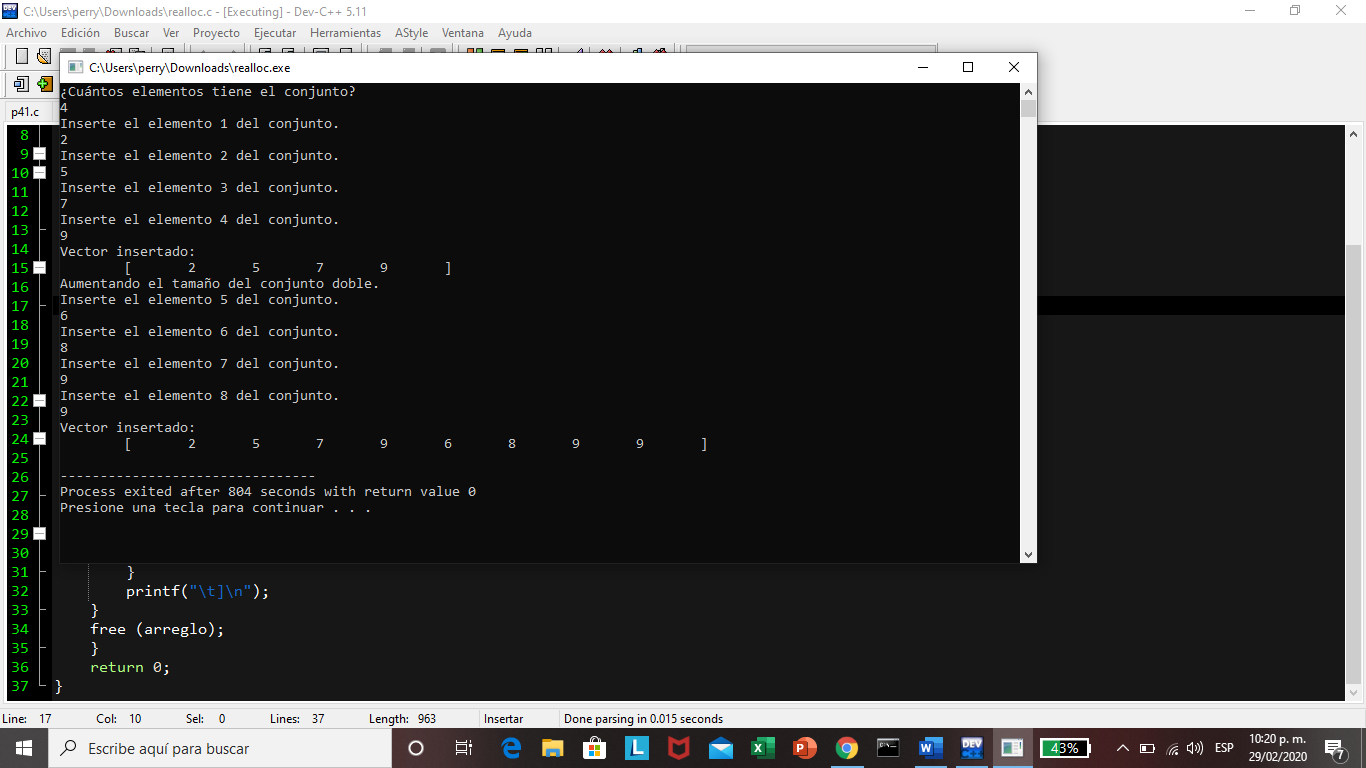




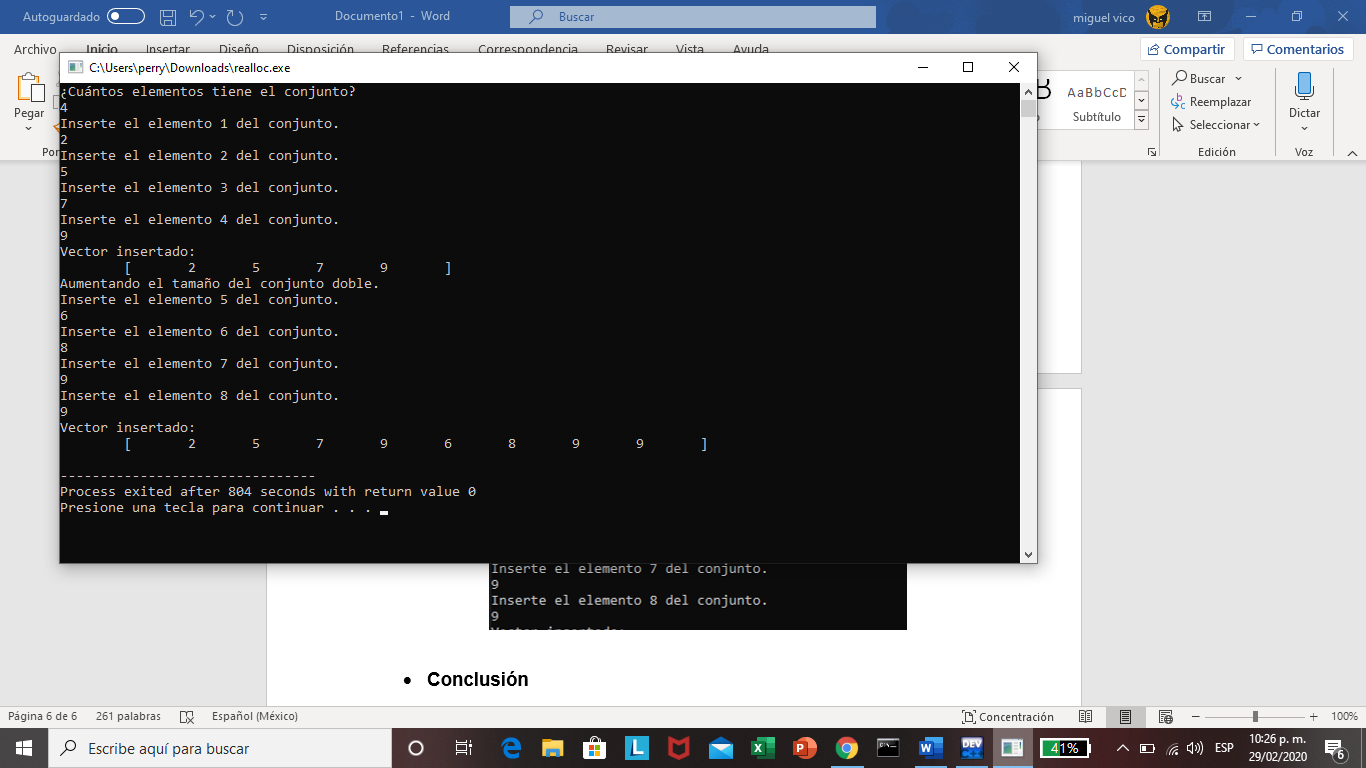
Prosiguiente a esto se almacena en la memoria y se muestran los valores asignados en un vector



Por ultimo solicita otros valores para poder hacer una extensión al vector y extender la reservación en la memoria.



Al finalizar se muestra el vector extendido con el doble de valores que se asignaron al inicio



* **Conclusión**

La memoria dinámica es muy útil para la reservación y limpieza de variables de la memoria a partir de funciones con apuntadores

* **Bibliografía**

<https://sites.google.com/site/programacioniiuno/temario/unidad-1---manejo-de-memoria-dinmica/programacin-en-c-con-memoria-dinmica>