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

Laboratorios de computación

salas A y B

|  |  |
| --- | --- |
| *Profesor:* | Edgar Tista García |
| *Asignatura:* | Estructuras de Datos y Algoritmos I |
| *Grupo:* | 1 |
| *No. de Práctica(s):* | 4 |
| *Integrante(s):* | Ugalde Velasco Armando |
| *No. de Equipo de cómputo empleado:* | 41 |
| *No. de Lista:* | 38 |
| *Semestre:* | 2020-2 |
| *Fecha de entrega:* | 7 de marzo de 2020 |
| *Observaciones:* |  |
|  |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_\_

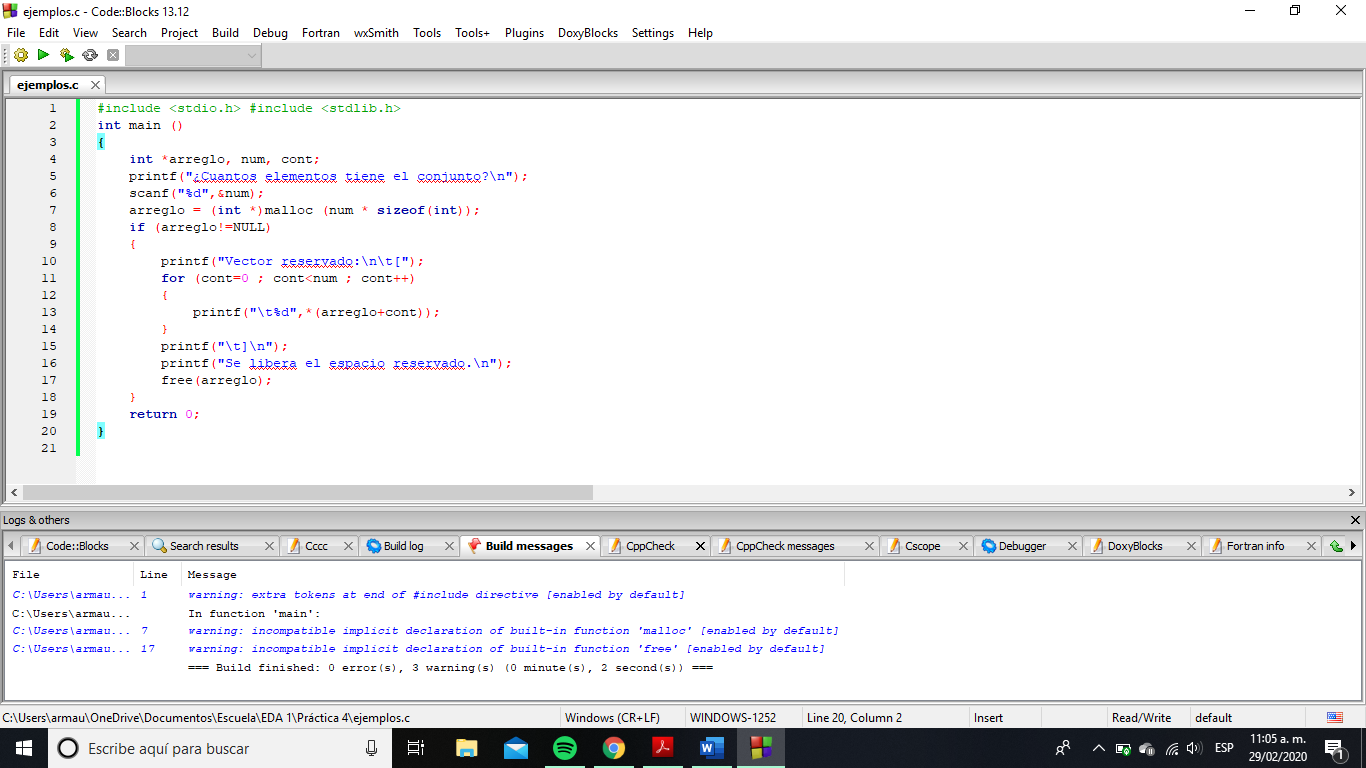
**Objetivo**

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

**Ejemplos de la guía de laboratorio**

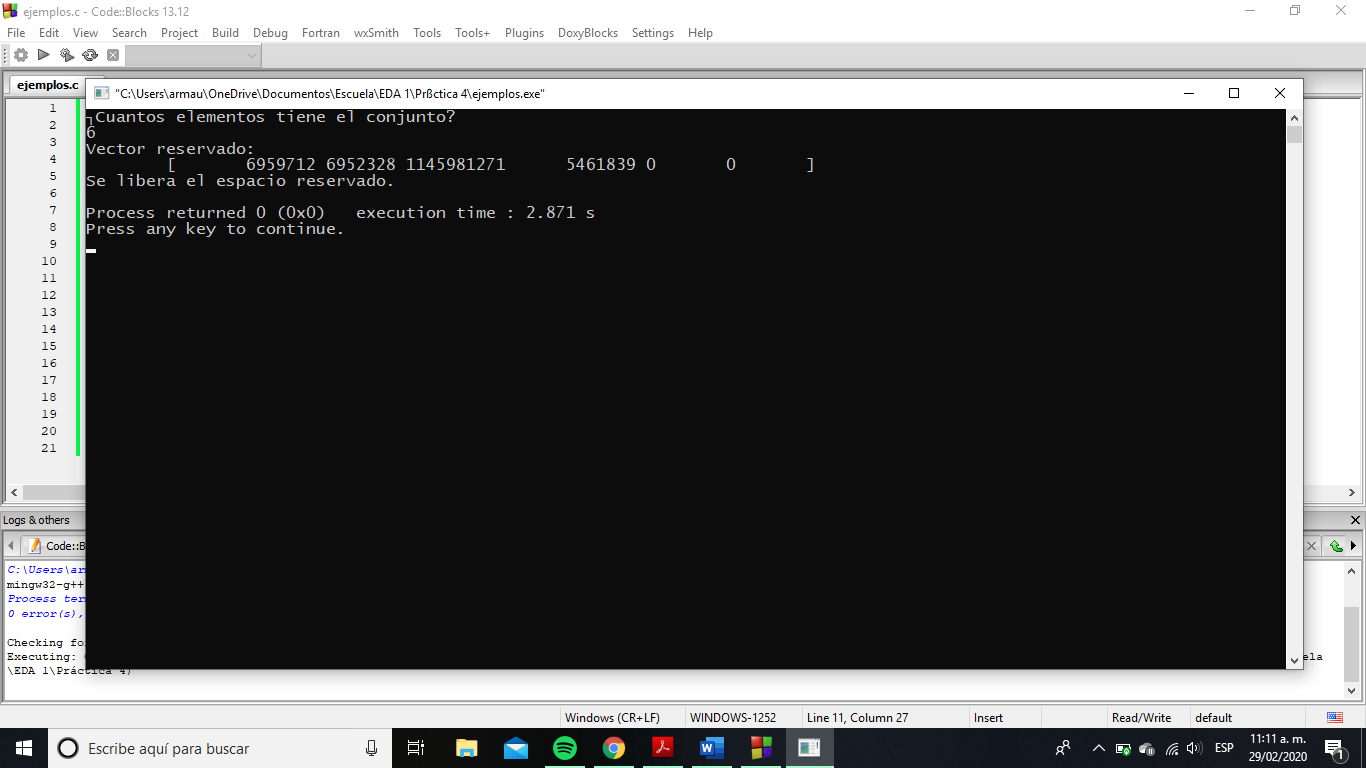
**Malloc**

El objetivo del programa fue asignar la cantidad de memoria necesaria para almacenar el número de elementos especificado por el usuario. En la siguiente línea se observa la asignación de memoria dinámica realizada para el arreglo de enteros (variable arreglo), que en realidad se trata de un puntero a entero. El argumento de la función **malloc** fue el tamaño en bytes del total de elementos a almacenar, es decir, el número de elementos (almacenado en variable **num**) multiplicado por el tamaño de un **int**.



***Asignación dinámica de memoria***

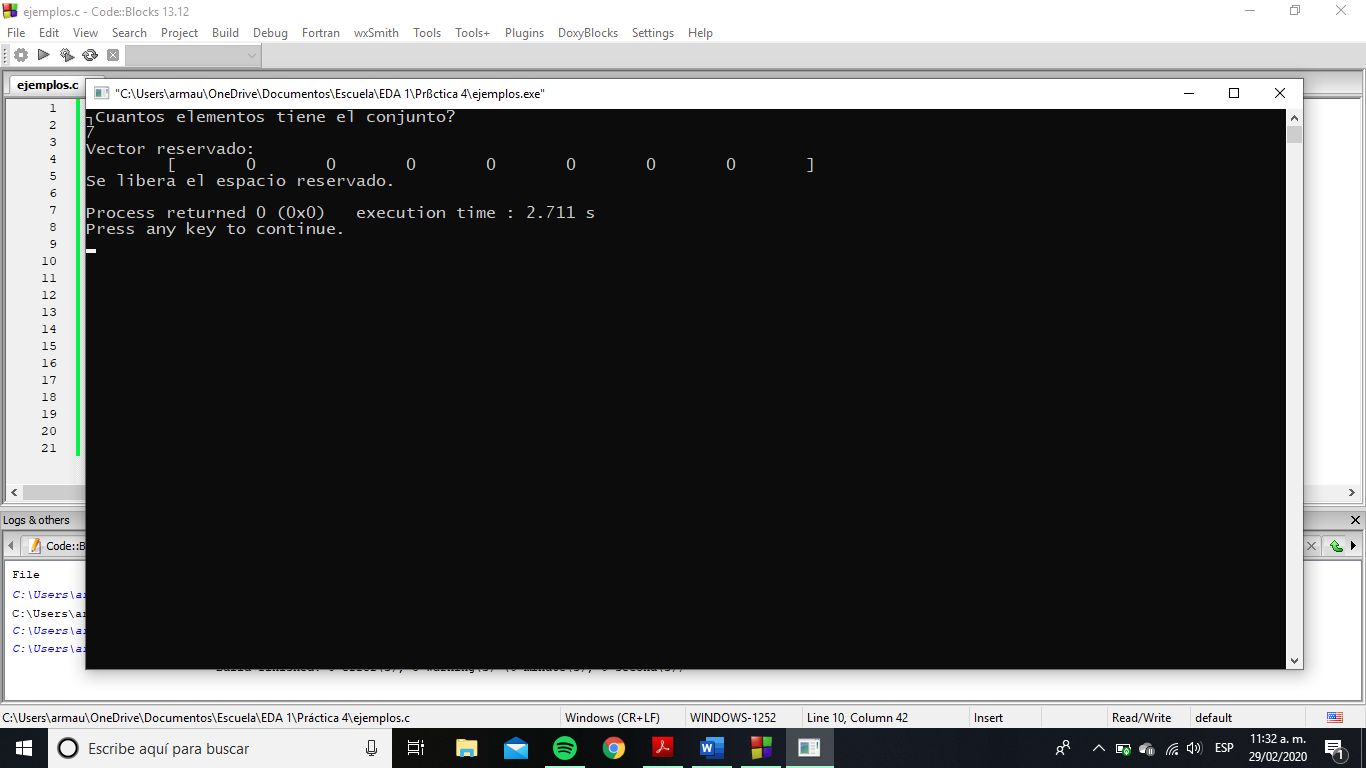
Después, se imprimía el contenido de cada elemento del arreglo. Sin embargo, los valores arrojados eran “valores basura”, o aleatorios, ya que la función **malloc** no inicializa los elementos a cero. A continuación, se muestra la salida del programa:



***Valores “basura” en la memoria asignada por malloc.***

**Calloc**

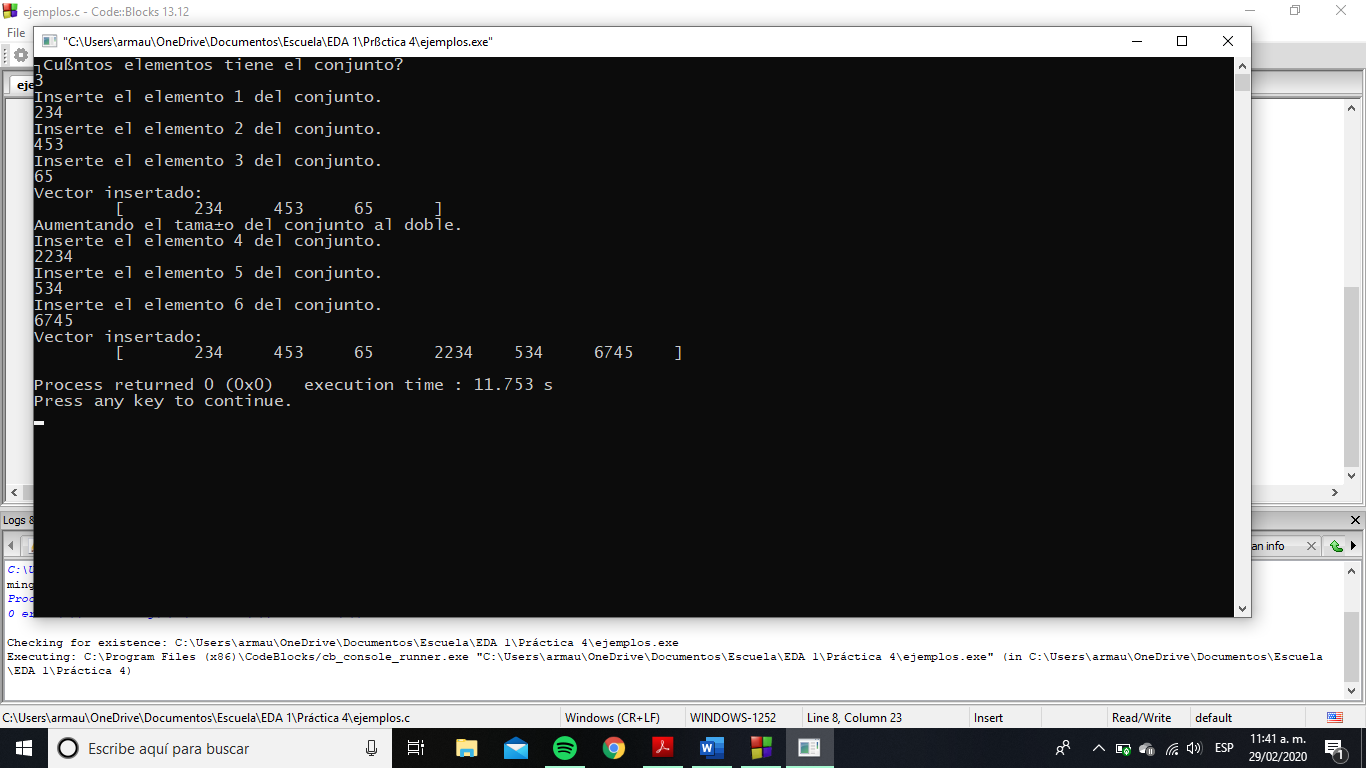
Este programa realizaba la misma tarea que el anterior, pero utilizaba la función **calloc,** en lugar de **malloc**. Como sabemos, **calloc** inicializa los valores de cada elemento de la porción de memoria solicitada a 0, a diferencia de **malloc**. A continuación, se muestra la salida del programa:



***Valores inicializados a 0 en la memoria asignada por calloc.***

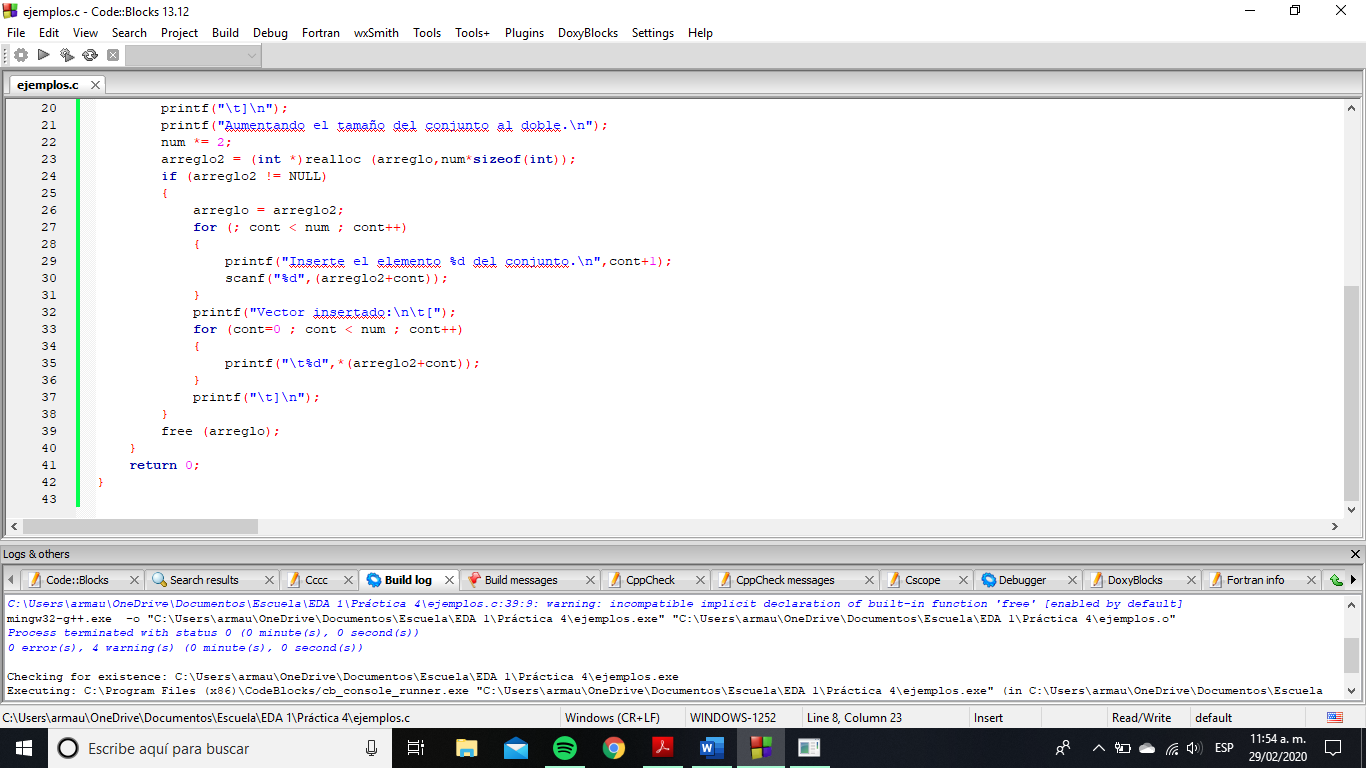
**Realloc**

El objetivo de este programa fue, al igual que en los dos programas anteriores, la asignación dinámica de memoria. Primero, se creó un arreglo de enteros almacenado de forma dinámica, utilizando la función **malloc** y el tamaño proporcionado por el usuario. Después, se inicializó el valor de cada elemento, con los datos que se solicitaron al usuario:



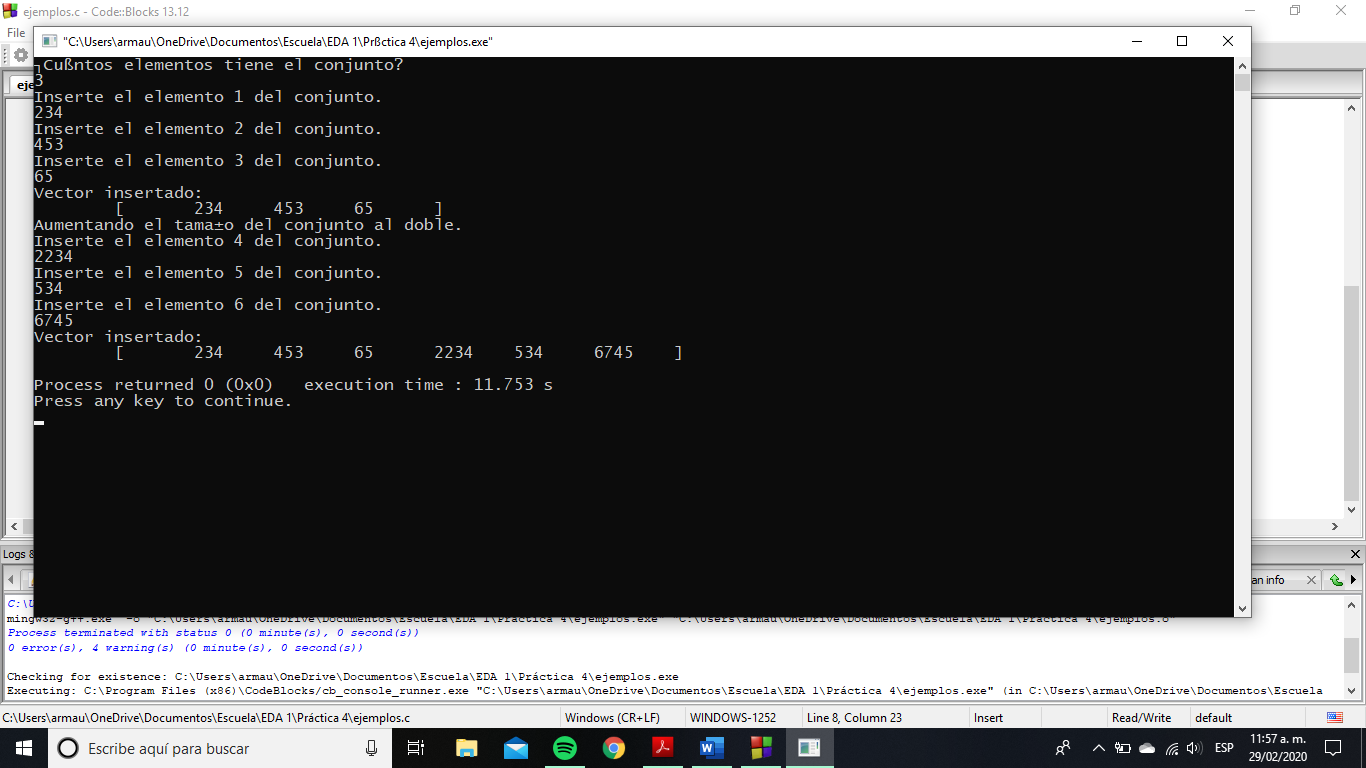
***Creación e inicialización de arreglo almacenado dinámicamente.***

Luego, se utilizó la función **realloc** para aumentar el tamaño del arreglo al doble. Dicha tarea se llevó a cabo en la siguiente línea:



***Utilización de la función realloc para aumentar el tamaño del arreglo al doble.***

Finalmente, se solicitaron los datos restantes al usuario para llenar el arreglo, y se imprimieron para corroborar su correcto almacenamiento. Se liberó la memoria asignada utilizando la función **free**.



***Utilización de realloc para aumentar el tamaño del arreglo al doble. Se solicitaron los datos restantes al usuario.***

**Ejercicio 1**

Codifica, compila y ejecuta el código “ejercicio1”, posteriormente, responde las preguntas.

**a) Captura la pantalla con la ejecución del programa y explica de manera detallada los resultados que se observan.**

Se logró compilar y ejecutar el programa con éxito. Cabe mencionar que se mostraron algunas advertencias al momento de compilar el programa. A continuación, se muestra la captura de pantalla de la salida del programa.



***Ejecución del programa***

En cada fila, podemos observar que se imprimen dos valores asociados a la posición de un arreglo.

En el programa, se encuentra declarada e inicializada la variable arreglo, que contiene 5 elementos de tipo entero.



***Declaración e inicialización de variable arreglo.***

Después, encontramos el siguiente ciclo:



***Ciclo donde se imprime la dirección y el valor de cada elemento en el arreglo.***

Podemos notar que en cada iteración se imprime la dirección del elemento del arreglo correspondiente, utilizando el operador de dirección (**&**), y enseguida se imprime su valor. Sin embargo, el arreglo únicamente tiene 5 elementos inicializados. Por lo tanto, en la salida del programa se observan valores inesperados a partir del índice 5, es decir, del sexto elemento. Dicho comportamiento ocurre debido a que se accede a localidades de memoria que no pertenecen al arreglo, lo cual se considera **comportamiento indefinido**. Es importante notar que las direcciones de memoria mostradas son las esperadas, ya que son contiguas, tomando en cuenta que el tamaño de cada elemento es de 4 bytes.

Después, se imprime una lista de direcciones y valores. Dicha tarea se lleva a cabo en el siguiente ciclo:

****

***Ciclo que imprime la dirección y el valor de cada elemento del espacio asignado a la variable ptr.***

Como podemos observar, la tarea de este ciclo es recorrer el espacio de memoria asignado a la variable **ptr**, e imprimir sus respectivas direcciones y valores. Dicha variable se encuentra definida al inicio del programa de la siguiente forma:

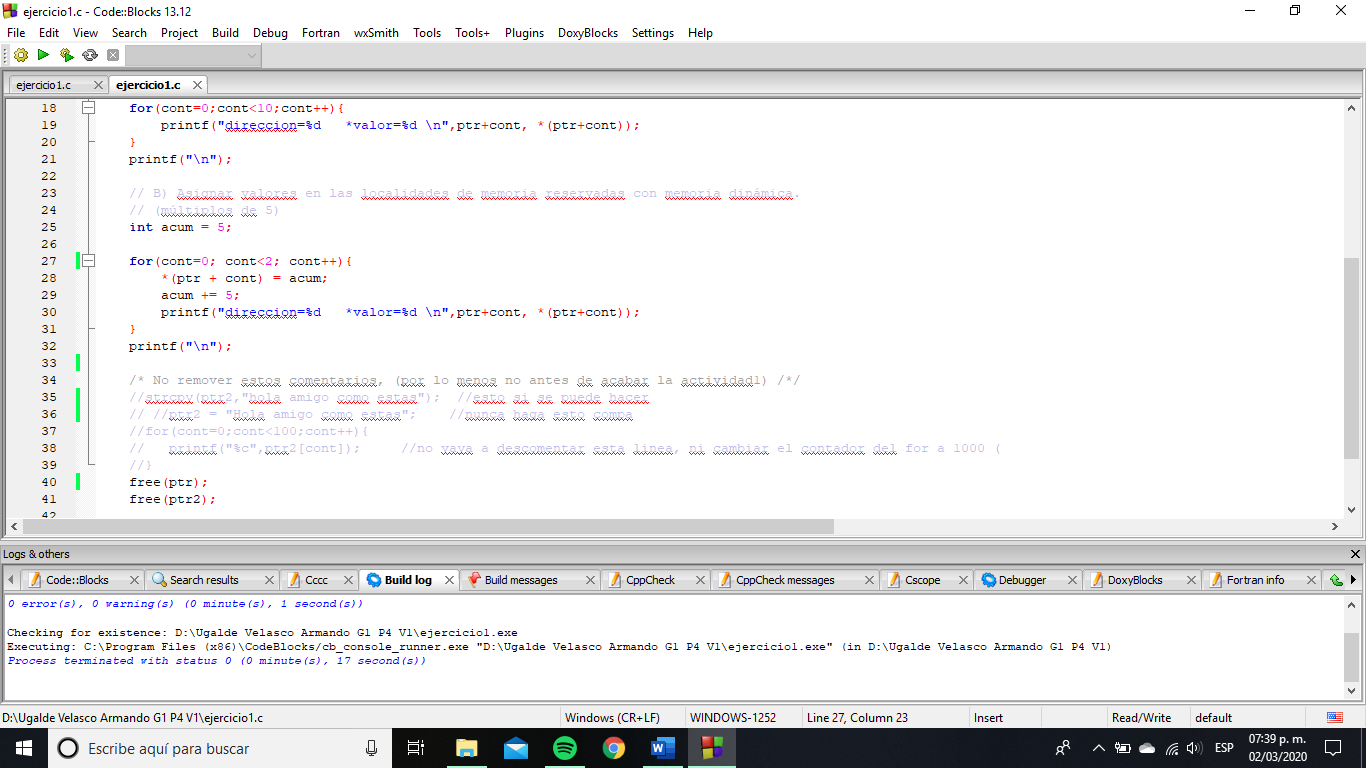


***Asignación de memoria dinámica a la variable ptr.***

Se le asignan 10 bytes de memoria dinámicamente a la variable **ptr** mediante la función **malloc**.

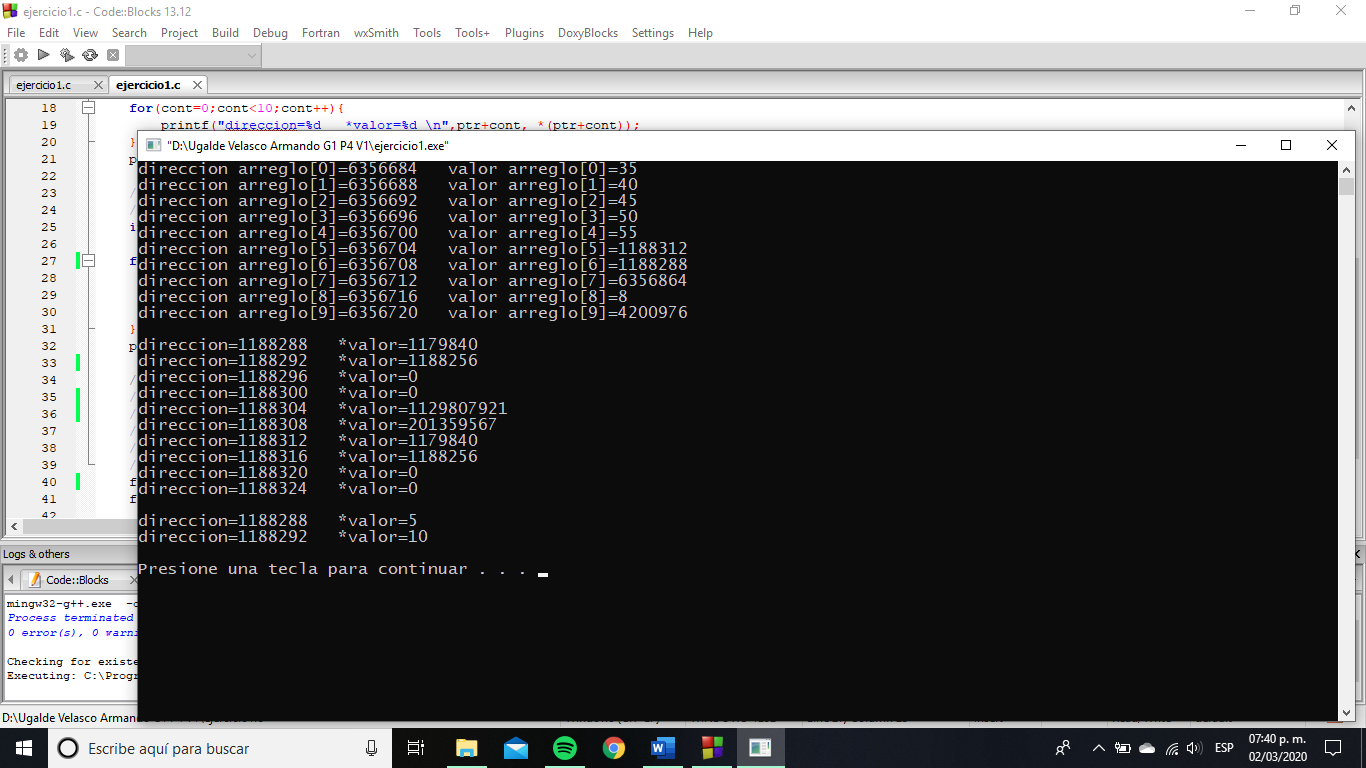
En el ciclo, se recorren los elementos de la memoria asignada, y se imprime su dirección y su valor. La salida del programa es la esperada: las direcciones de memoria son contiguas, tomando en cuenta que el tamaño de cada entero es de 4 bytes, y, además, los valores impresos no tienen un comportamiento en específico, ya que la función malloc no inicializa los elementos a 0.

**b) Agrega instrucciones para asignar valores en las localidades de memoria reservadas con memoria dinámica (múltiplos de 5). Se deberán mostrar en pantalla los valores asignados y la dirección de éstos.**



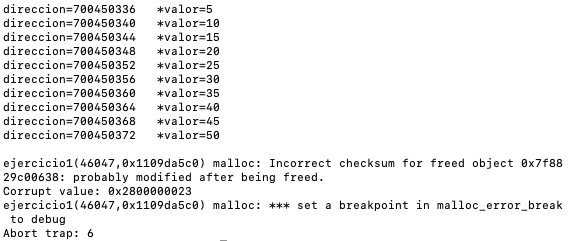
***Ciclo utilizado para asignar valores a cada elemento e imprimir su valor.***

Se utilizó el ciclo mostrado para asignar valores a cada elemento del arreglo almacenado dinámicamente en la variable **ptr**. Como únicamente se asignaron 10 bytes, la cantidad de enteros posibles a asignar es 2, ya que 3 elementos ocuparían 12 bytes, una cantidad que sobrepasa la memoria asignada.



***Salida del programa***

Sin embargo, si bien es posible realizar la asignación de más elementos, la ejecución de dicha tarea se considera **comportamiento indefinido**, por lo que se considera incorrecto realizarlo. A continuación, se muestra la salida del programa al asignar valores a 10 elementos:

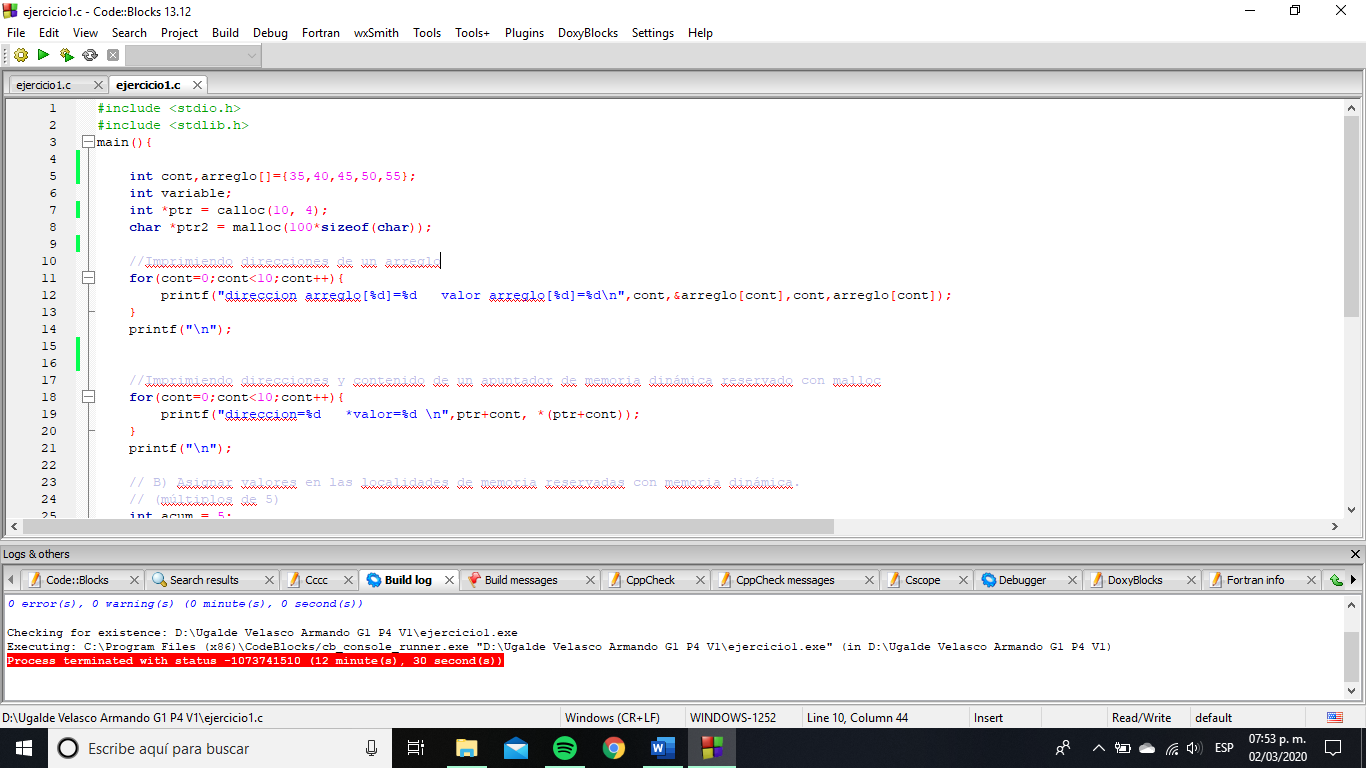
****

***Asignación de 10 valores a los elementos del arreglo asignado dinámicamente.***

Los valores impresos son los esperados. Se lograron realizar las asignaciones exitosamente, a pesar de que la memoria asignada no era del tamaño adecuado. Podemos concluir que se accedió a localidades de memoria que no pertenecían al arreglo, lo cual se considera comportamiento indefinido. Al final del programa se muestra un error relacionado al problema anterior.

**c) Modifica la asignación inicial de “\*ptr” de tal modo que ahora se utilice la función calloc. Compila, ejecuta y explica la diferencia con la ejecución previa.**

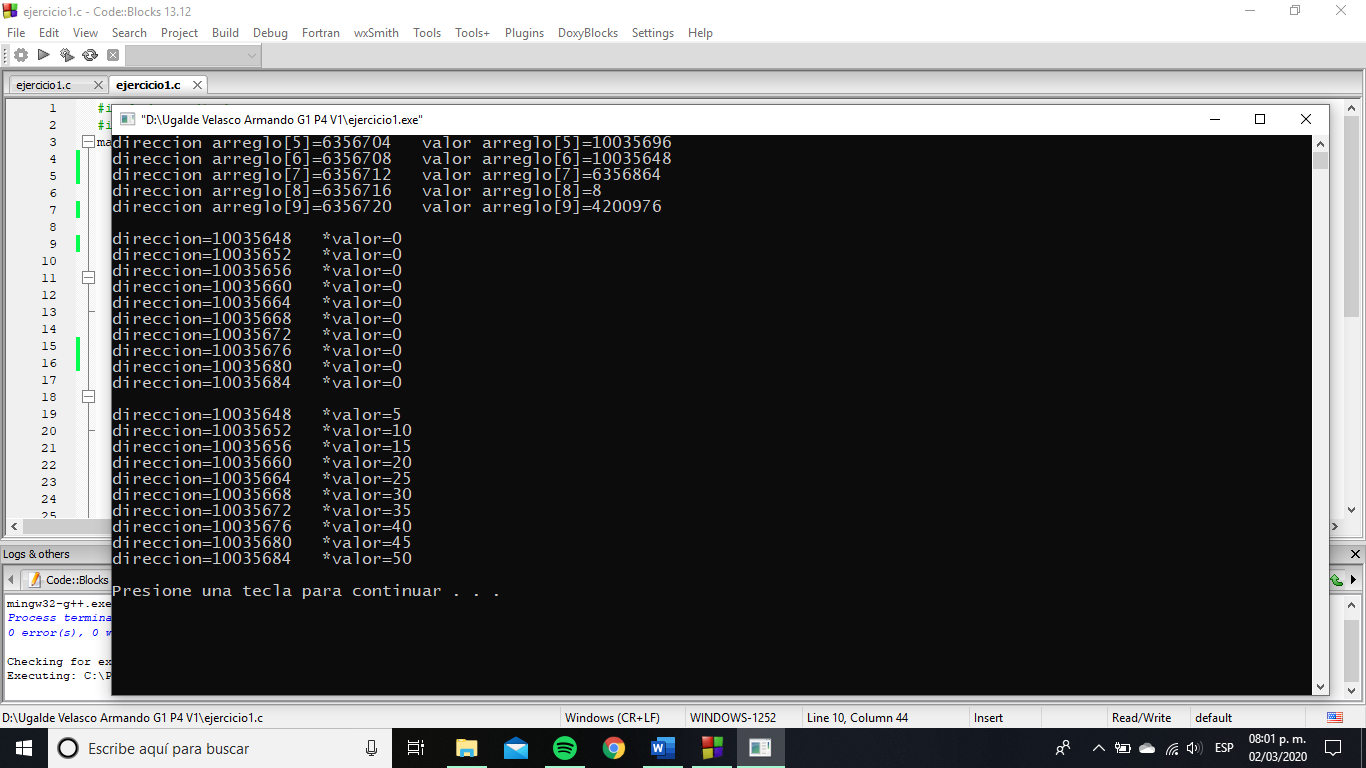
Se reemplazó la función **malloc** por **calloc**. El primer argumento de la función fue 10, ya que se decidió almacenar esta cantidad de elementos. El segundo argumento fue el número 4, ya que éste es el tamaño en bytes de un entero. A continuación, se observa la línea de código mencionada:



***Función calloc***

Además, se modificó el índice del ciclo creado en el inciso anterior (*de 2 a 10*), para que éste recorriera los 10 elementos del arreglo.

Como sabemos, la diferencia más notable de **calloc** con la función **malloc**, es que al ejecutarse **calloc**, inicializa los valores de todos los elementos asignados dinámicamente a 0. En la primera salida del programa, observamos valores aleatorios en los elementos del arreglo. Sin embargo, a continuación, se muestra que el comportamiento esperado del programa modificado se cumplió: el valor de los elementos del arreglo fue 0, inicialmente.



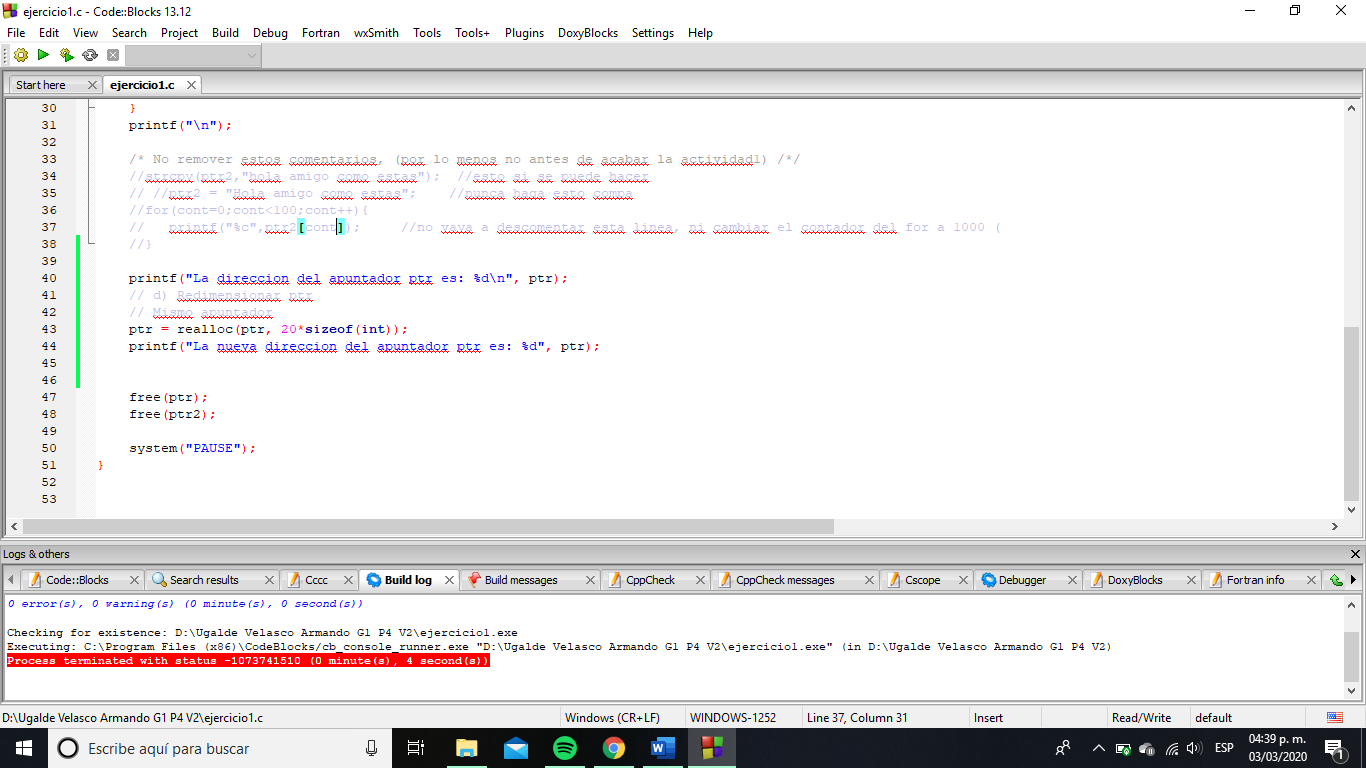
***Salida del programa***

Después, se ejecutó el ciclo creado en el inciso anterior, el cual inicializa los valores del arreglo a múltiplos de 5. Lo anterior coincide con la salida del programa.

**d) Agrega instrucciones para utilizar la función realloc para redimensionar “ptr” con un nuevo tamaño de 20 espacios.**

* **Utiliza realloc para asignar el nuevo tamaño en el mismo apuntador ptr.**

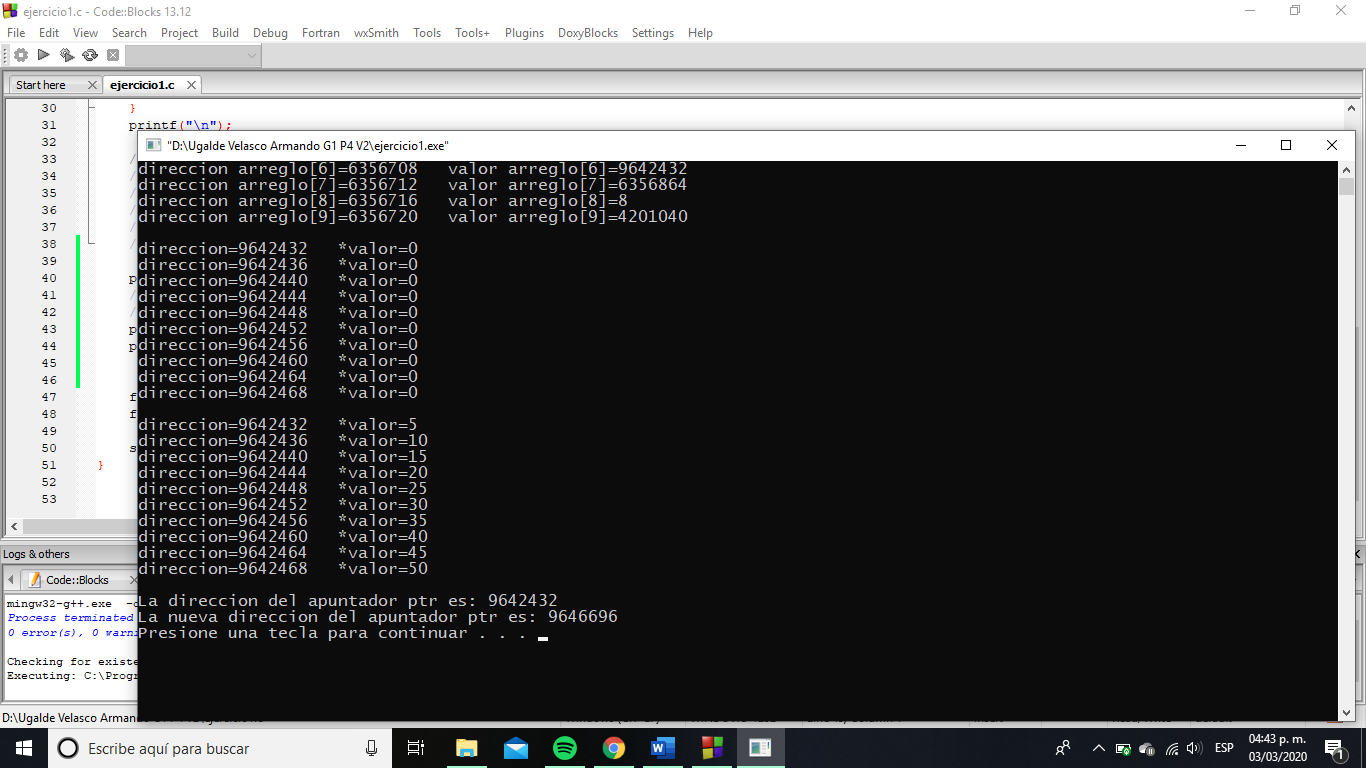
Se añadieron las siguientes líneas de código para cumplir con el objetivo:



***Reasignación de memoria utilizando realloc.***

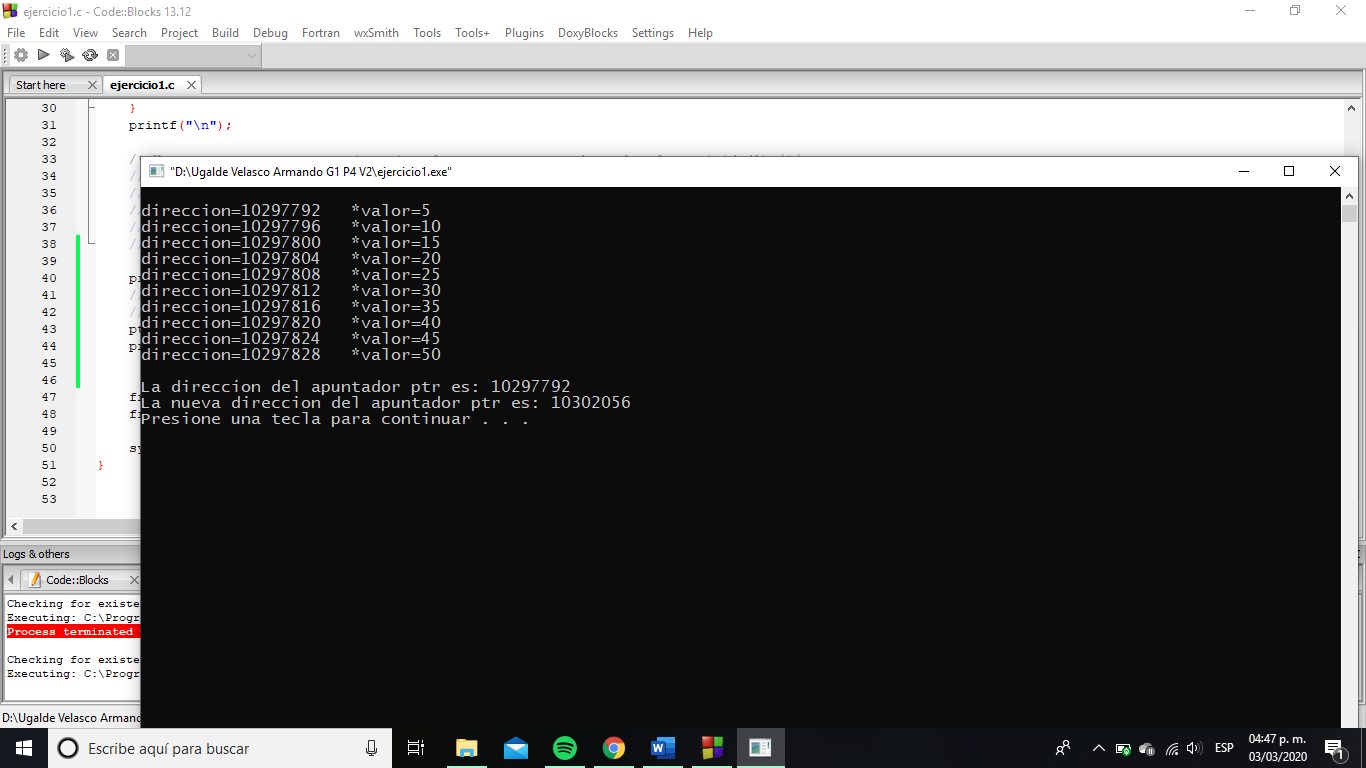
Primero, se imprimió la dirección original del apuntador, para percatarnos fácilmente del posible cambio que se presentaría. Después, se realizó la reasignación correspondiente, a 20 espacios para entero. Finalmente, se imprimió la nueva dirección del apuntador.

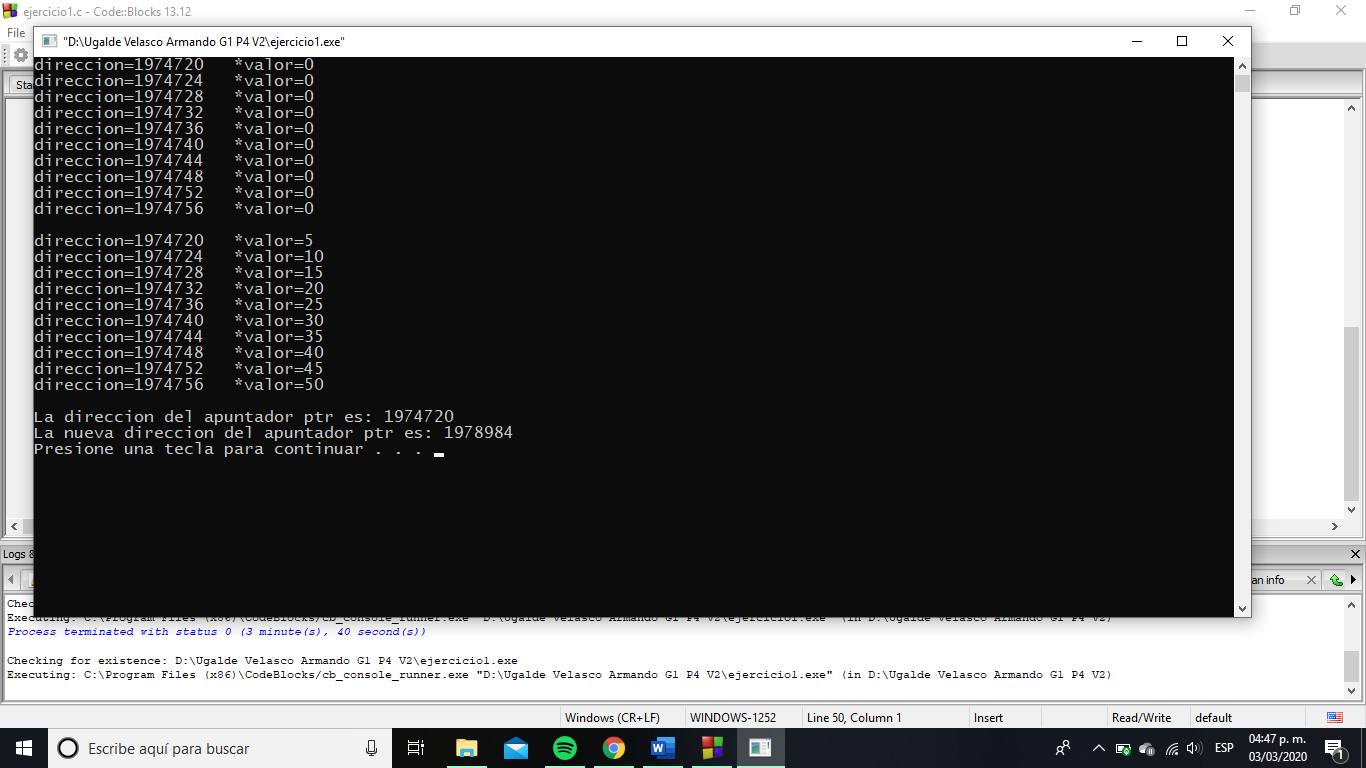
La salida del programa fue la siguiente:



***Variación de la variable ptr***

Como se puede observar, la dirección de memoria **cambió**. Se ejecutó el programa en diversas ocasiones, y ocurrió la misma situación:



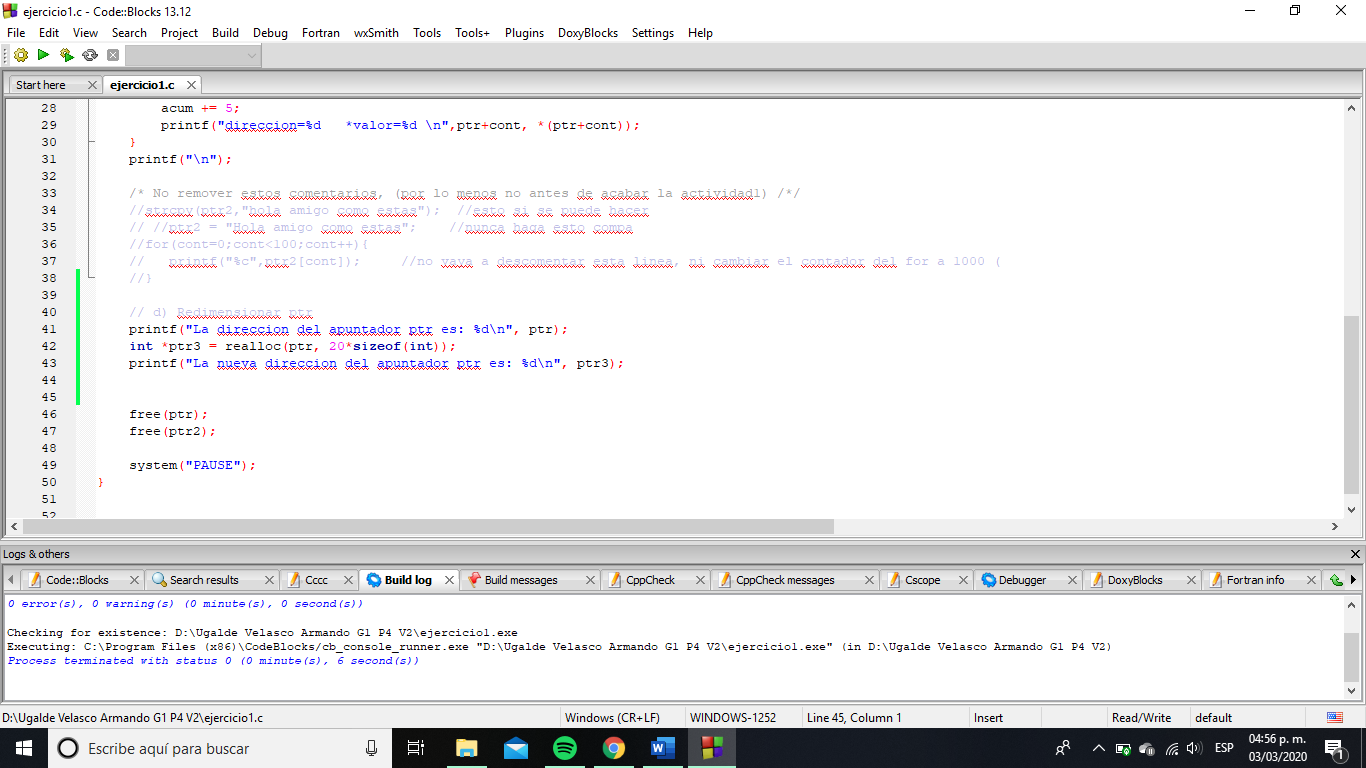


***Salida del programa en dos ejecuciones***

Sin embargo, es importante mencionar que el comportamiento de la función **realloc** no siempre es el anterior. La otra posibilidad pudo ser que el valor del puntero permaneciera igual. Sin embargo, esto **no depende** de la variable a la que asignemos el valor de retorno.

* **Utiliza realloc para asignar el nuevo tamaño en el apuntador ptr3.**

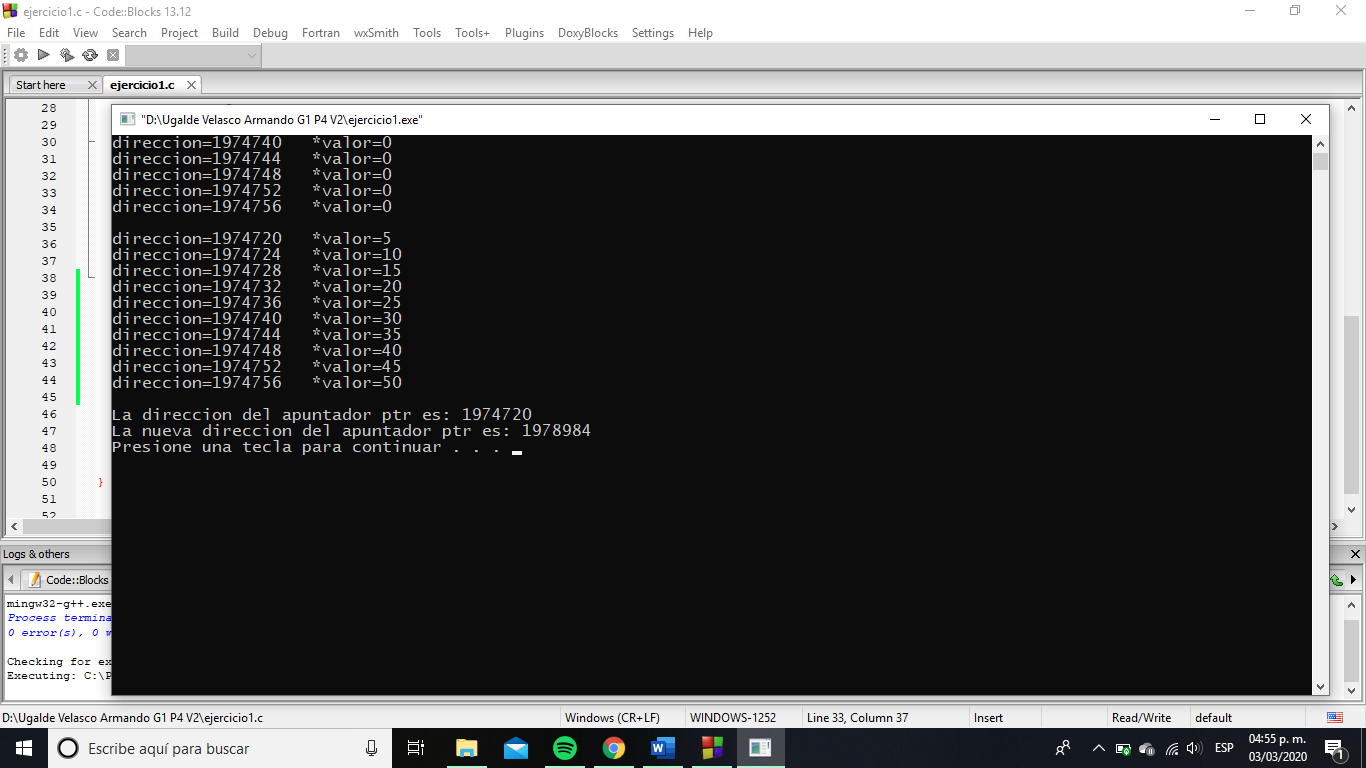
Se añadieron las siguientes líneas de código para cumplir con el objetivo:



***Reasignación de memoria utilizando realloc.***

Se realiza el mismo procedimiento que en el caso anterior. Sin embargo, en este caso asignamos el nuevo tamaño a la variable ptr3, como se especifica.

La salida del programa fue la siguiente:



***Salida del programa***

Al igual que en el caso anterior, la dirección de memoria **cambió**. Se ejecutó el programa en diversas ocasiones, y ocurrió la misma situación.

Por lo tanto, concluimos que no hay **ninguna diferencia** entre utilizar **realloc** y asignar la memoria al mismo apuntador o a otro diferente.

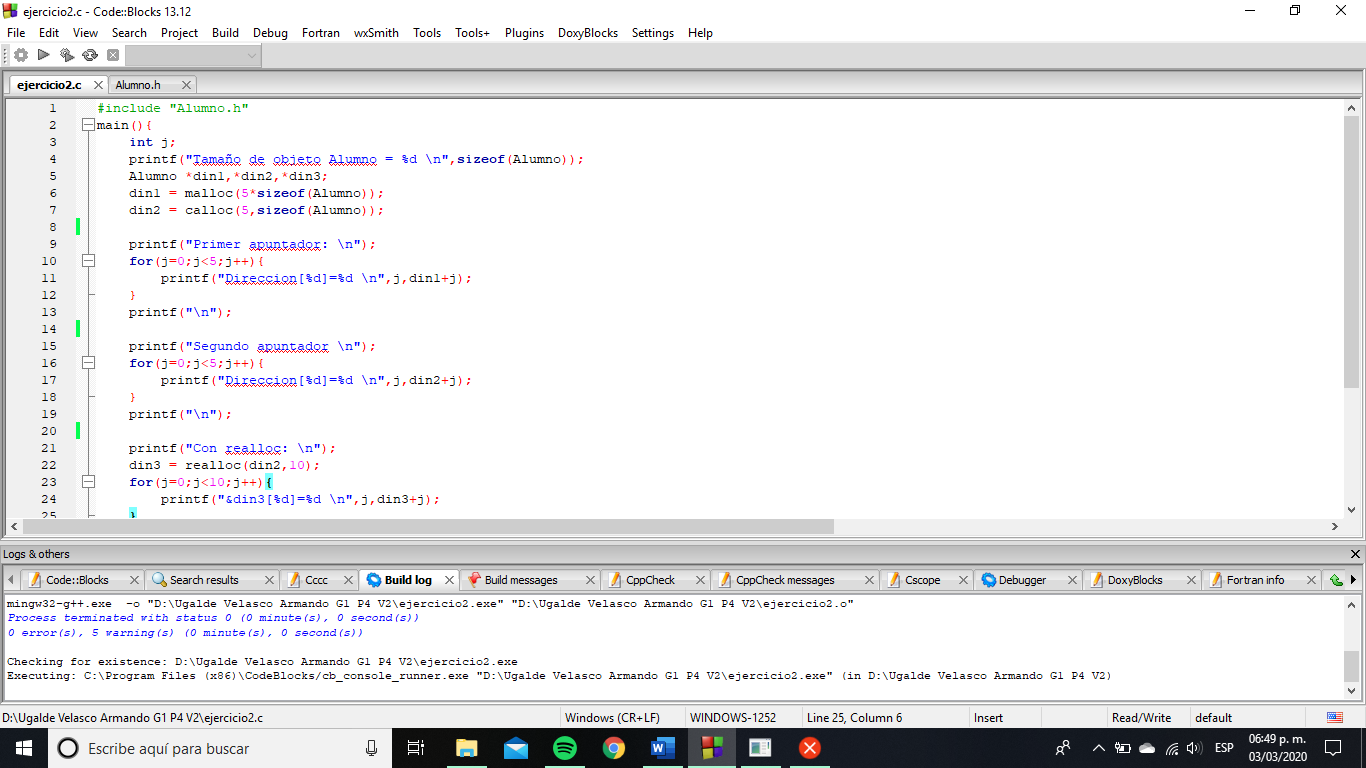
**Ejercicio 2**

Codifica, compila y ejecuta el código “ejercicio2”, posteriormente, responde las preguntas.

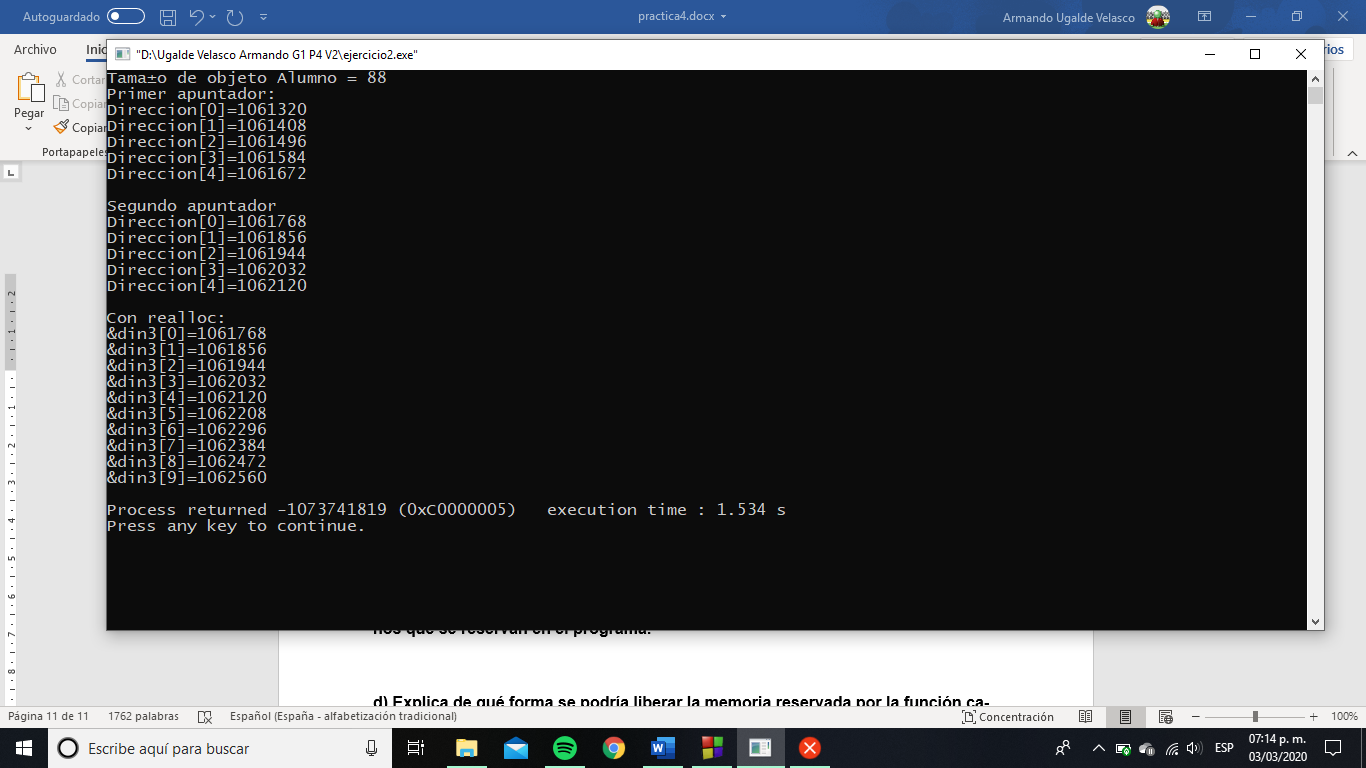
**a) Explica los resultados que se muestran en pantalla.**

El funcionamiento del programa es el siguiente:

Primero, se imprime el tamaño de la estructura alumno, declarada en el documento **Alumno.h**.

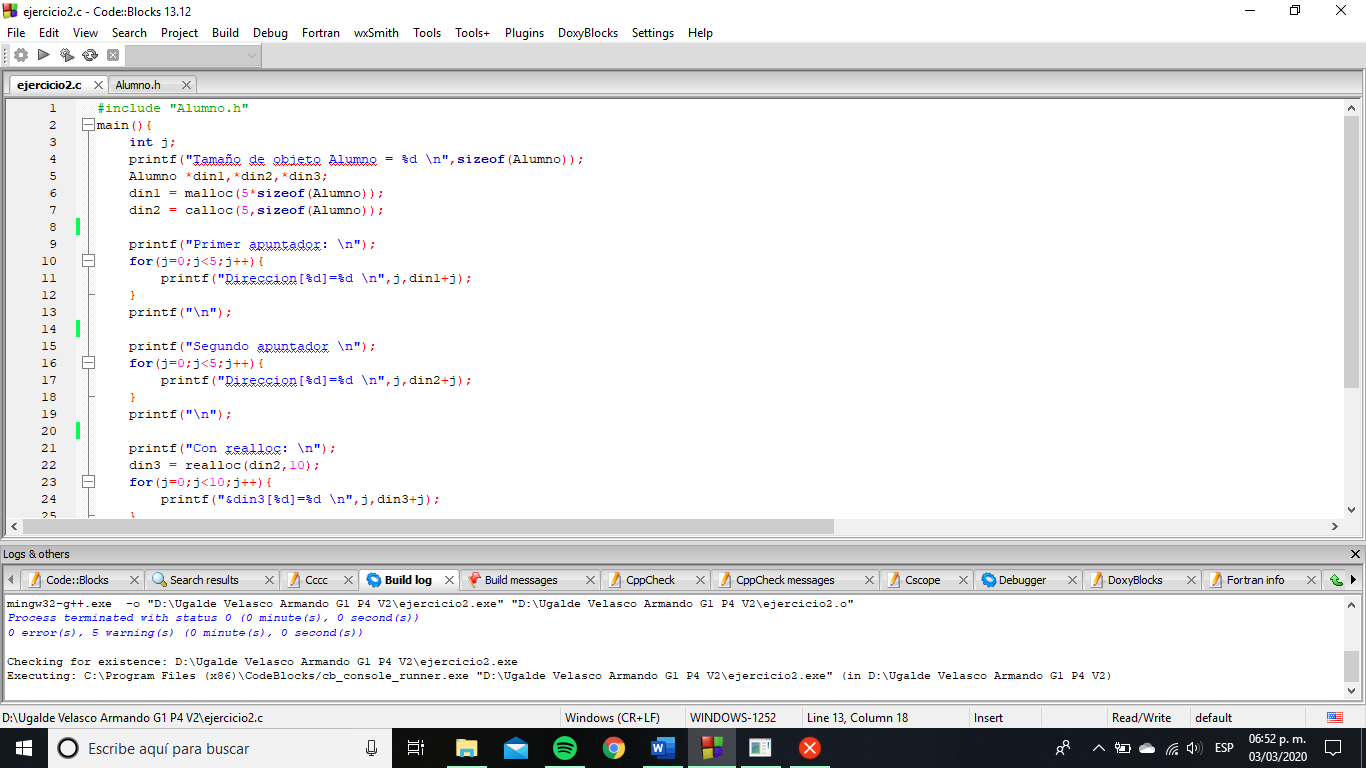


***Se imprime el tamaño de Alumno***



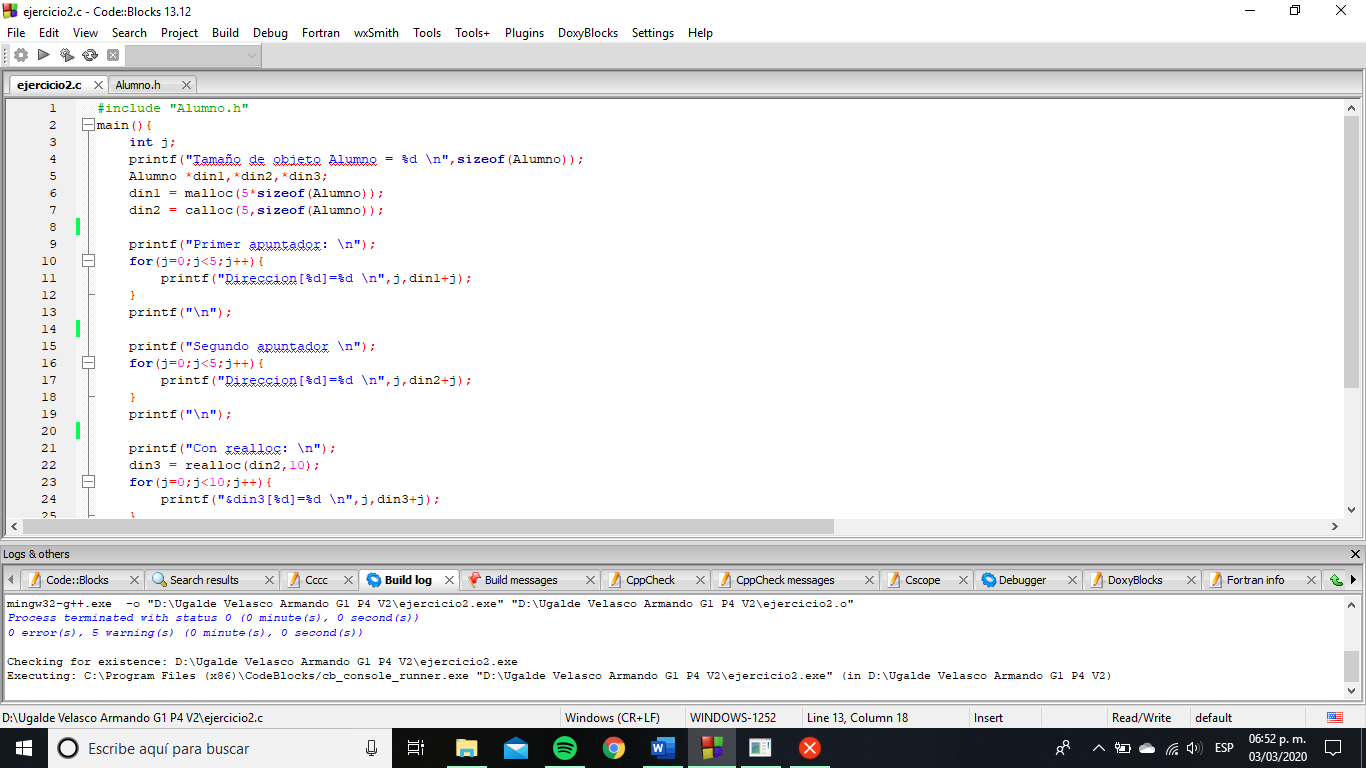
***Salida del programa***

Después, se asigna dinámicamente la memoria necesaria a las variables **din1** y **din2** para almacenar 5 estructuras Alumno. En la variable **din1** se utiliza la función **malloc,** y en la variable **din2** se utiliza **calloc**.

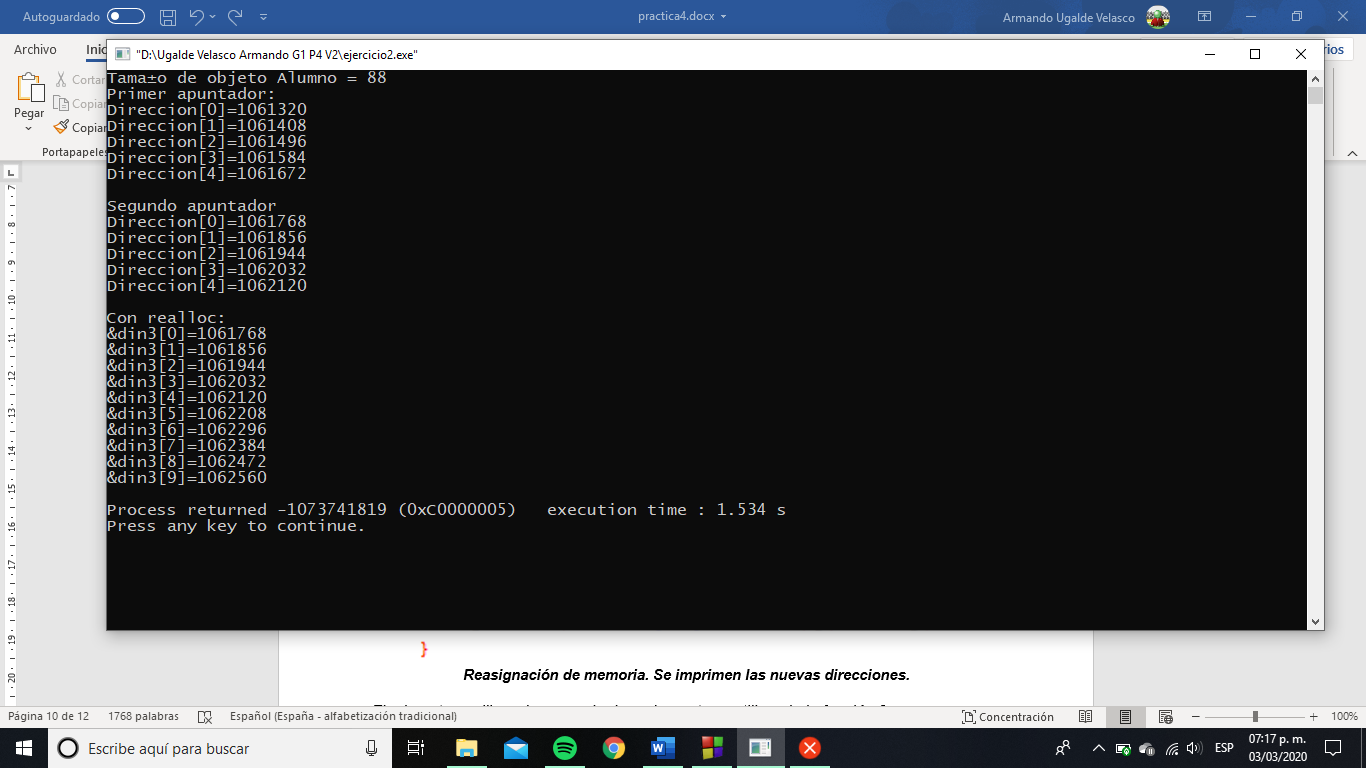


***Asignación dinámica de memoria***

Mediante la utilización de ciclos **for**, se imprimen las direcciones de los elementos de ambos arreglos:

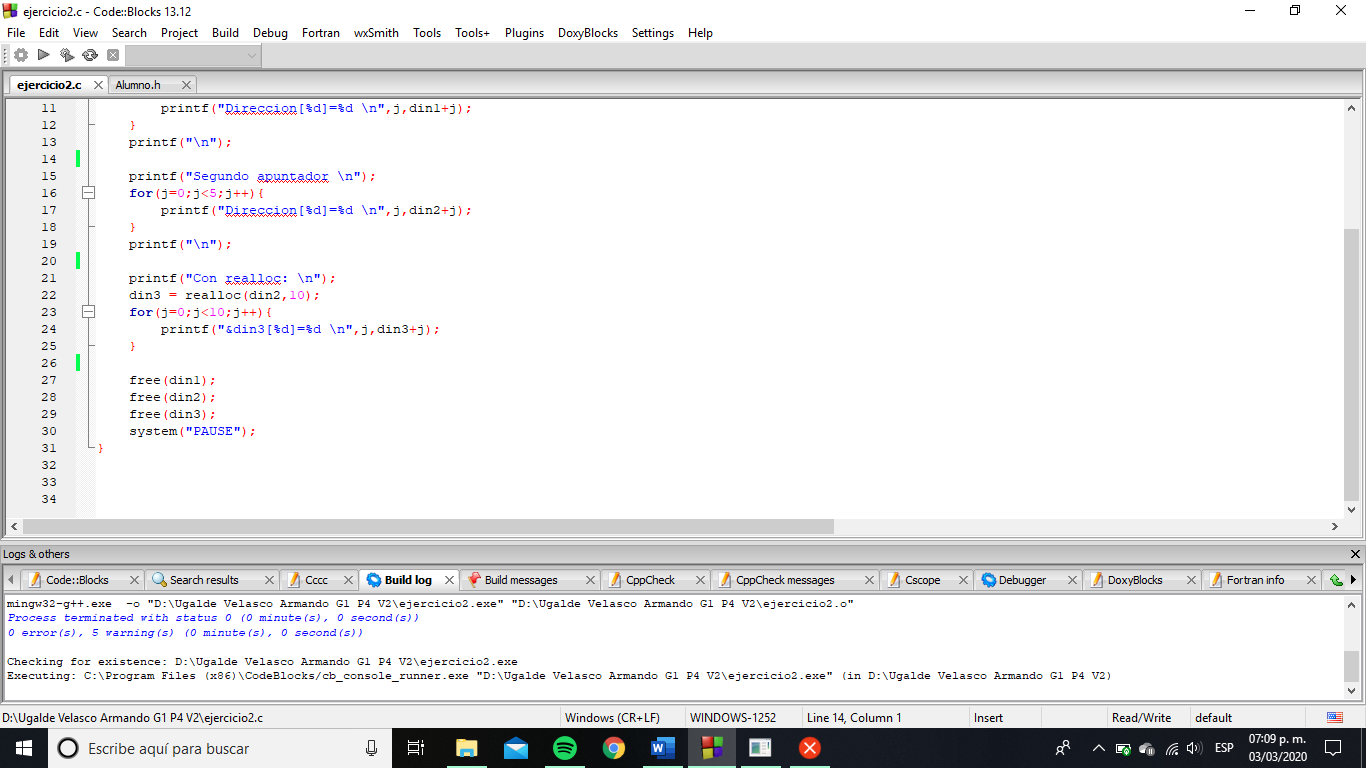


***Ciclos donde se imprimen las direcciones en los arreglos***

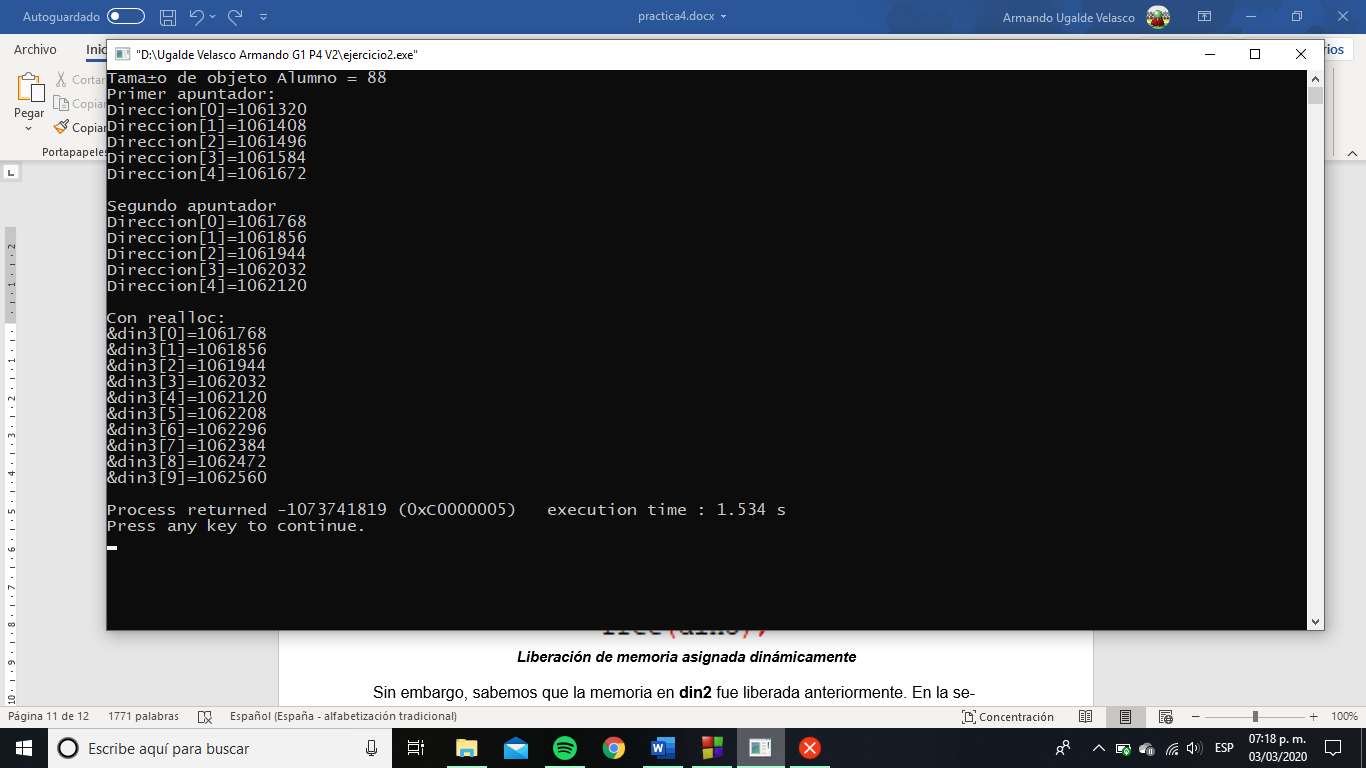


***Salida del programa***

Luego, se reasigna la memoria de **din2** en **din3**, utilizando la función **calloc**. En consecuencia, la memoria presente en **din2** es liberada y reasignada a un nuevo puntero: **din3**. Además, se imprimen las direcciones del nuevo arreglo mediante un ciclo:

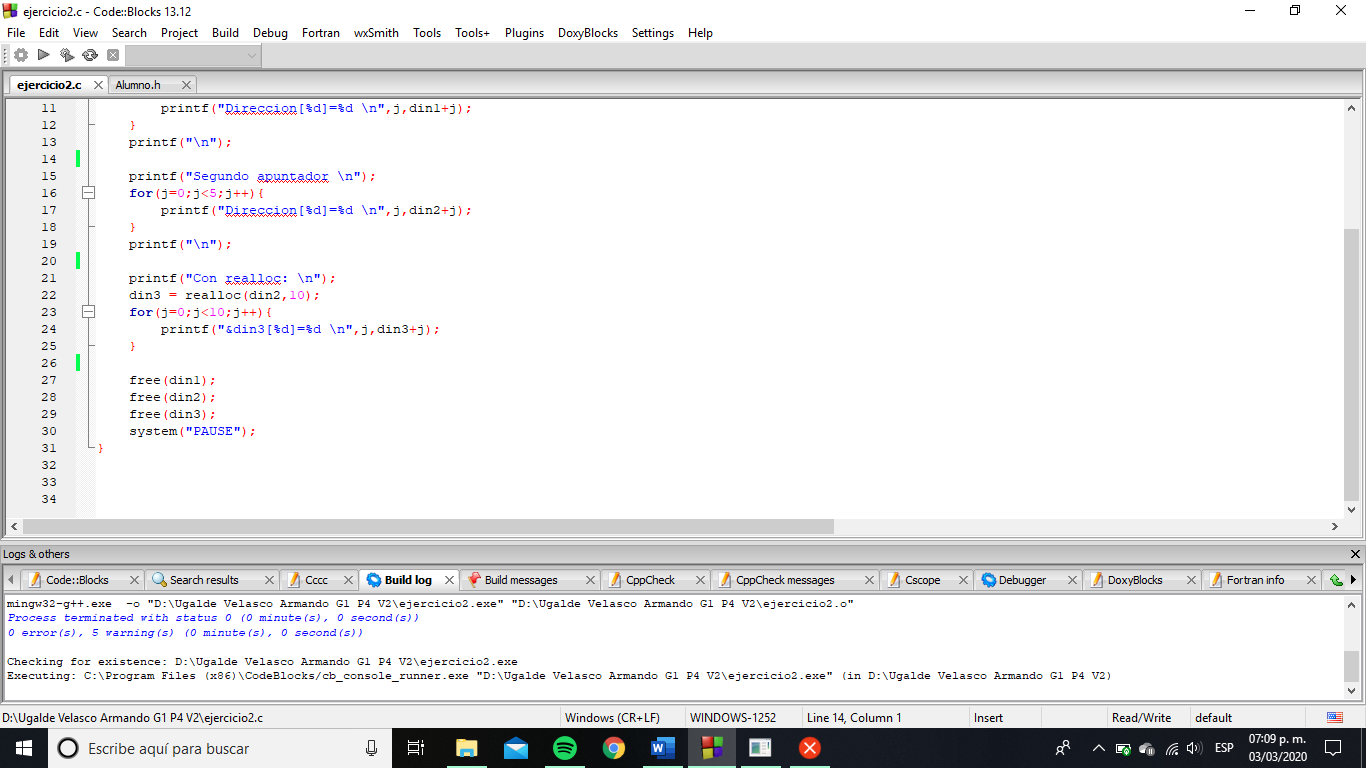


***Reasignación de memoria. Se imprimen las nuevas direcciones.***



***Salida del programa***

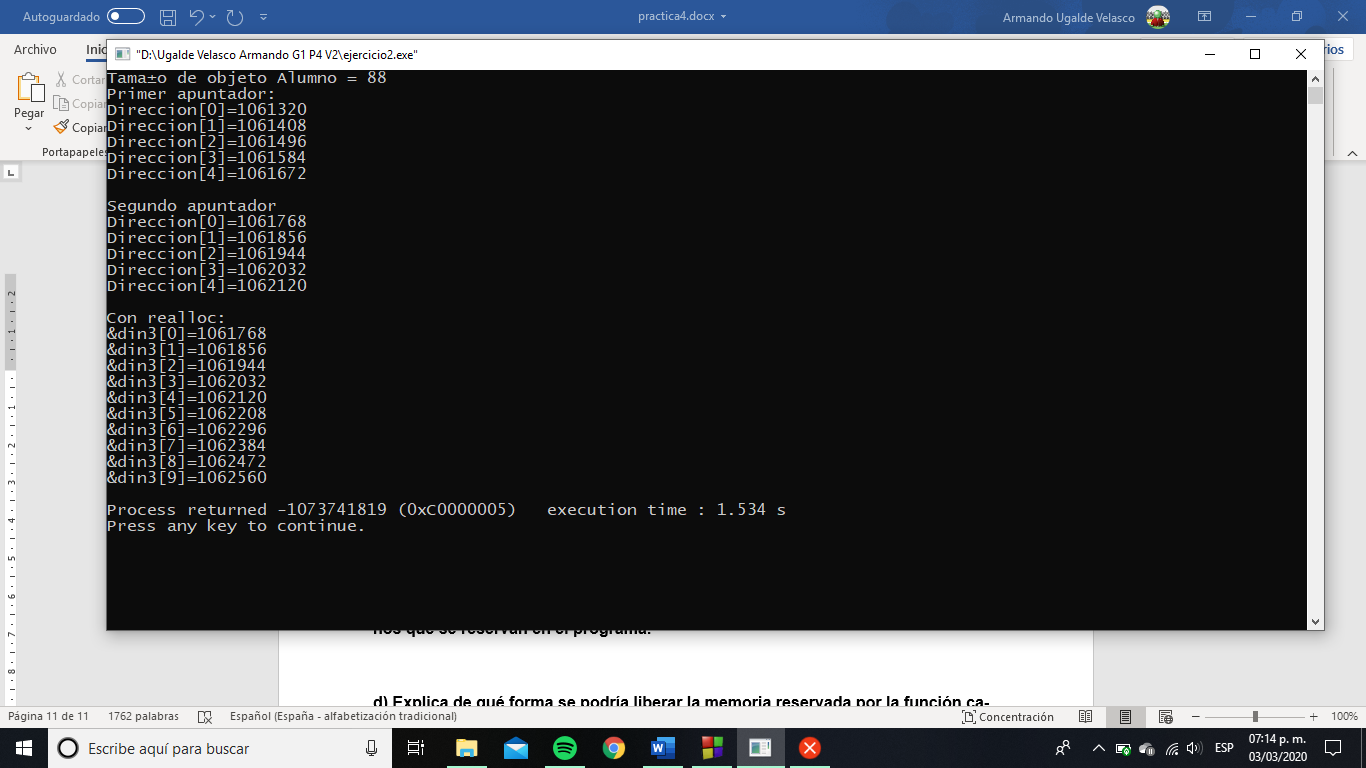
Finalmente, se libera la memoria de cada puntero, utilizando la función **free**.



***Liberación de memoria asignada dinámicamente***

Sin embargo, sabemos que la memoria en **din2** fue liberada anteriormente. En la segunda línea de la captura anterior se intenta liberar la memoria asignada a este puntero, lo que provoca que el programa se detenga y no finalice correctamente la ejecución.

A continuación, se observa la salida completa del programa:



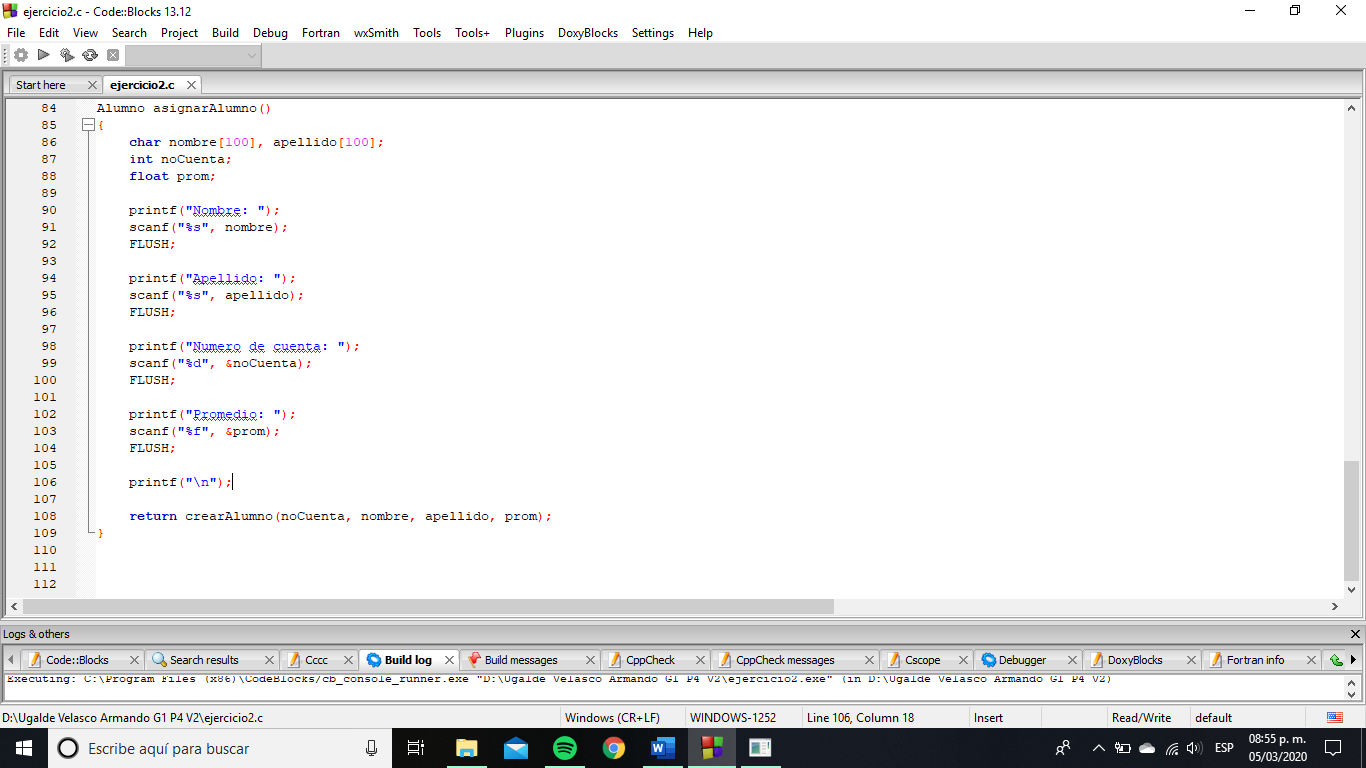
***Salida del programa***

**b) ¿Cuál es el tamaño de la estructura alumno?**

Como se logra observar en la salida del programa, su tamaño es de **88 bytes**.

**c) Agrega las instrucciones al código para pedir al usuario los datos de los alumnos que se reservan en el programa.**

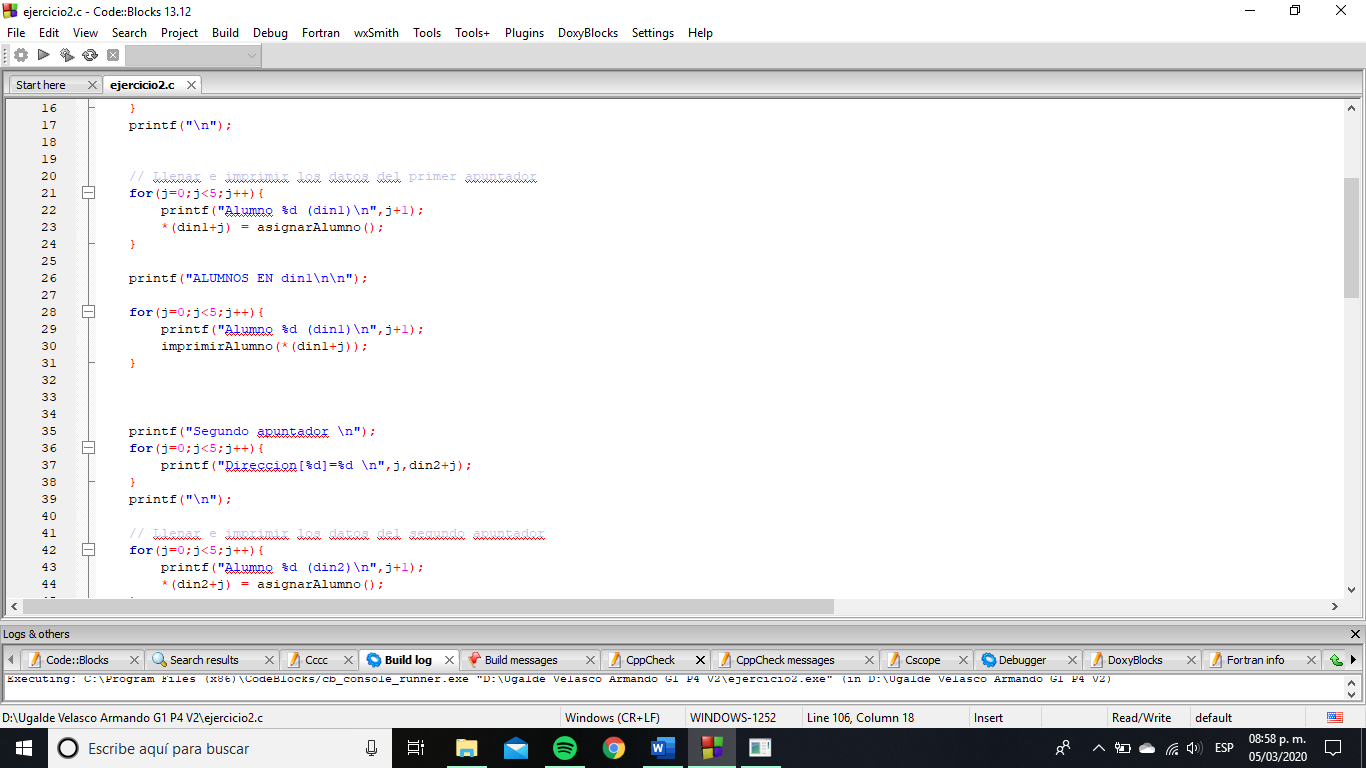
Primero, se creó la función auxiliar **asignarAlumno,** cuyo objetivo es solicitar los datos necesarios al usuario para crear un alumno, y retornar la estructura creada.



***Función asignarAlumno***

Como se puede observar, se solicitaron y almacenaron los datos en las variables correspondientes, para utilizarlos como argumentos en la función **crearAlumno** y retornar ese valor.

Para solicitar y asignar los valores correspondientes en cada arreglo, se añadieron las siguientes líneas:

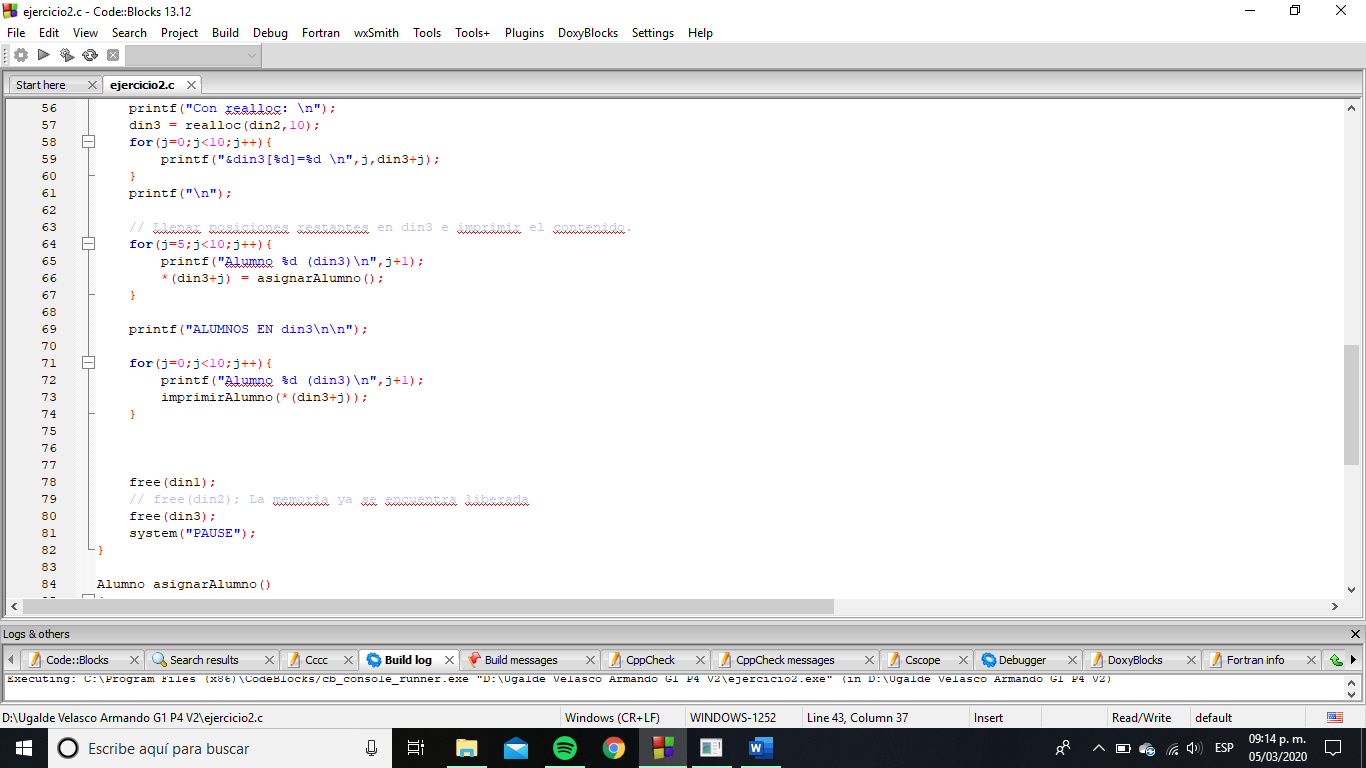


***Asignación de valores al arreglo din1***

El objetivo del primer ciclo fue solicitar los valores correspondientes para crear 5 estructuras alumno y asignarlas a los elementos del arreglo **din1**, con ayuda de la función auxiliar antes mencionada: **asignarAlumno**. Posteriormente, en el segundo ciclo se imprimieron los valores de cada alumno presente en el arreglo, para corroborar su correcto almacenamiento.

De igual forma, este proceso se llevó a cabo con la variable **din2.**

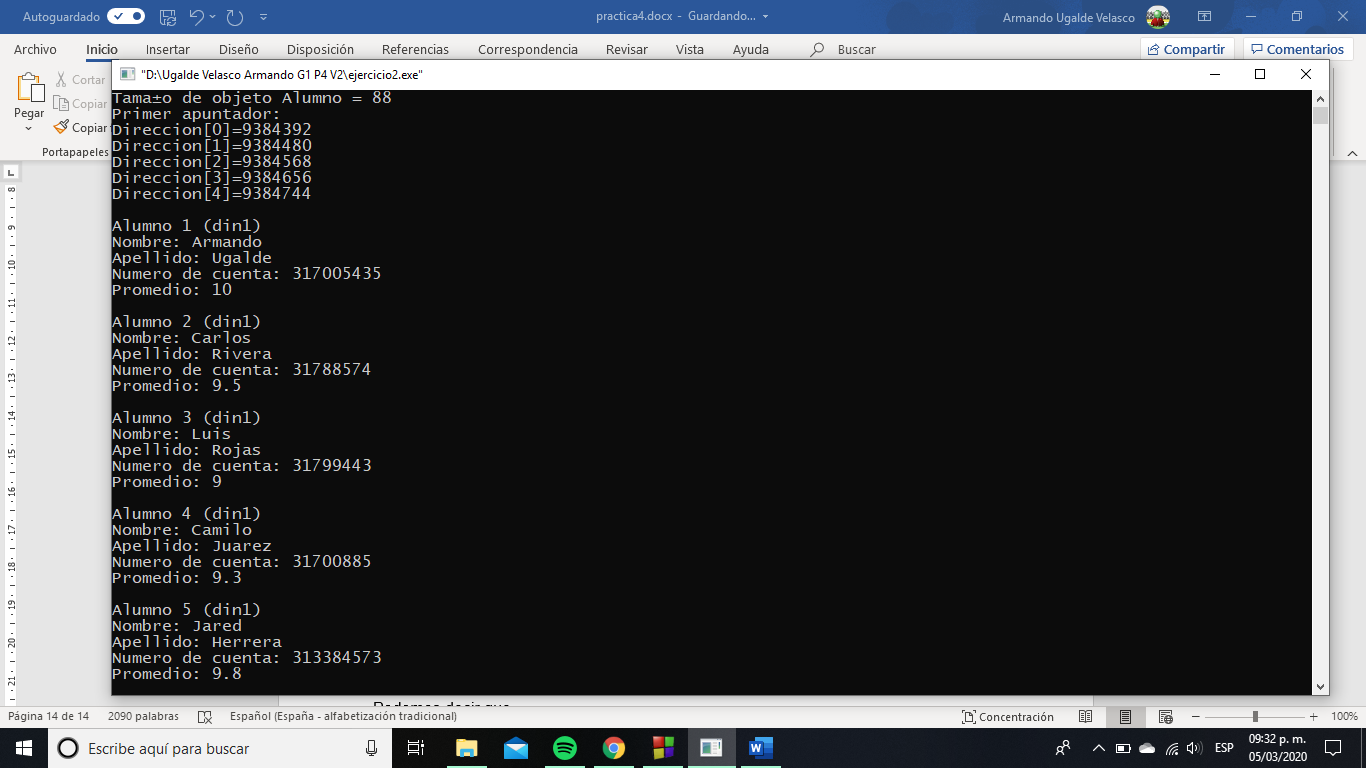
Finalmente, después de la ejecución de **realloc,** se realizó el mismo proceso para asignar valores a los 5 elementos restantes en **din3**, ya que, al reasignar la memoria en **din2** al nuevo puntero **din3** se mantuvieron intactos los primeros 5 elementos anteriormente almacenados.

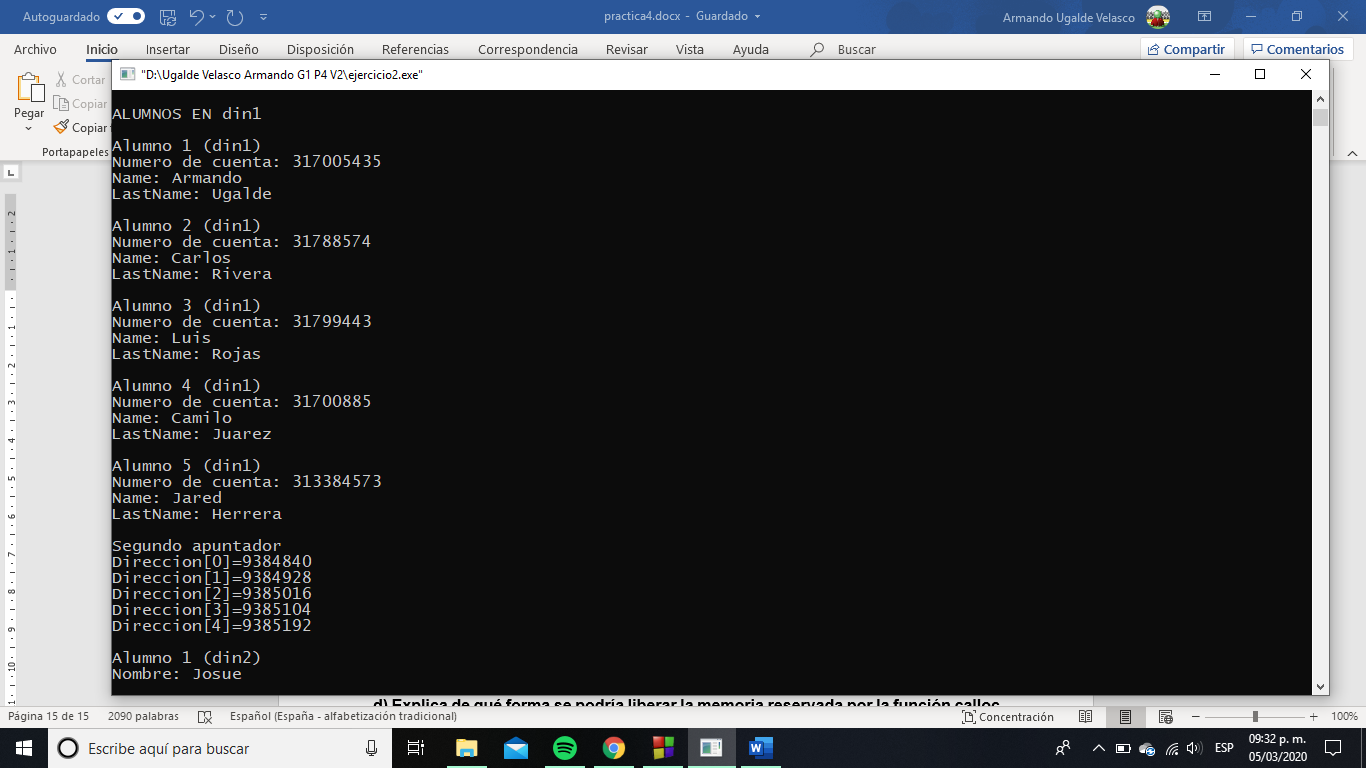


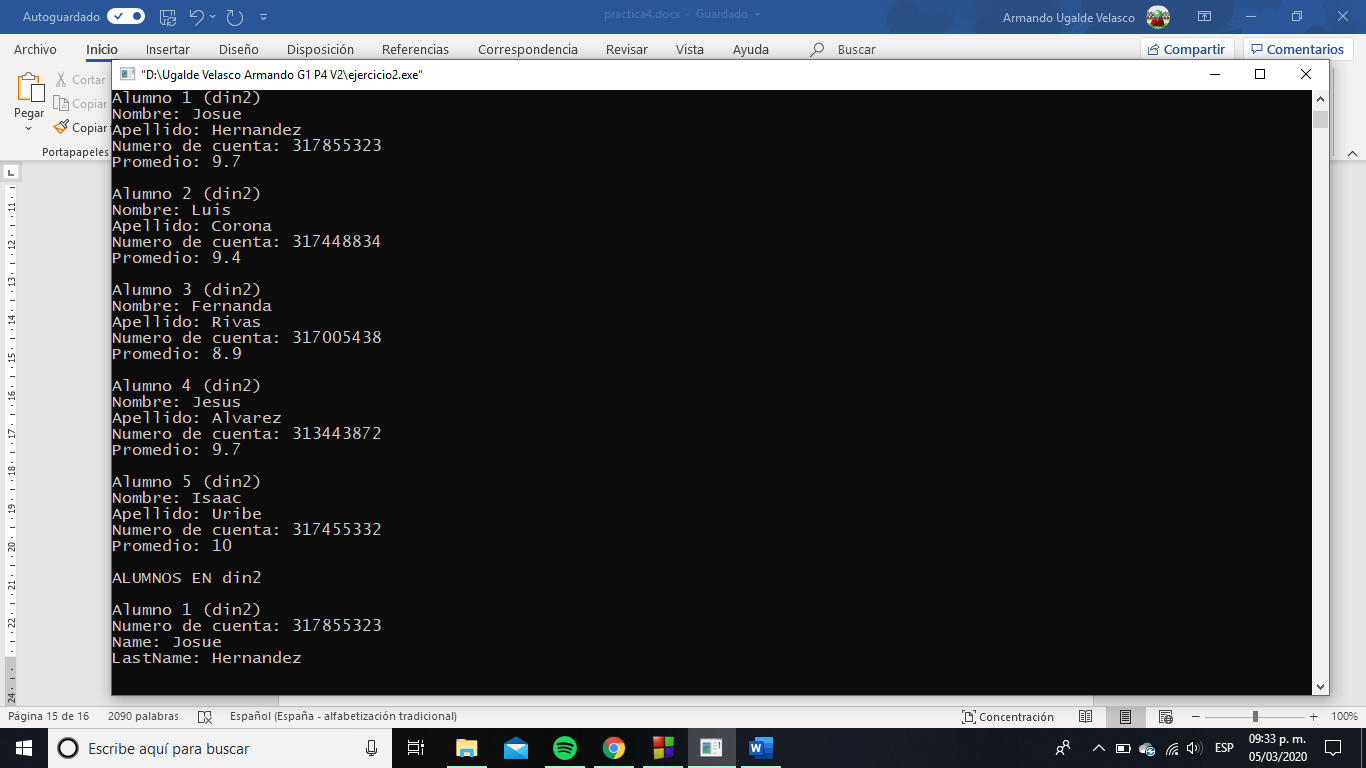
***Asignación de valores restantes al arreglo din3***

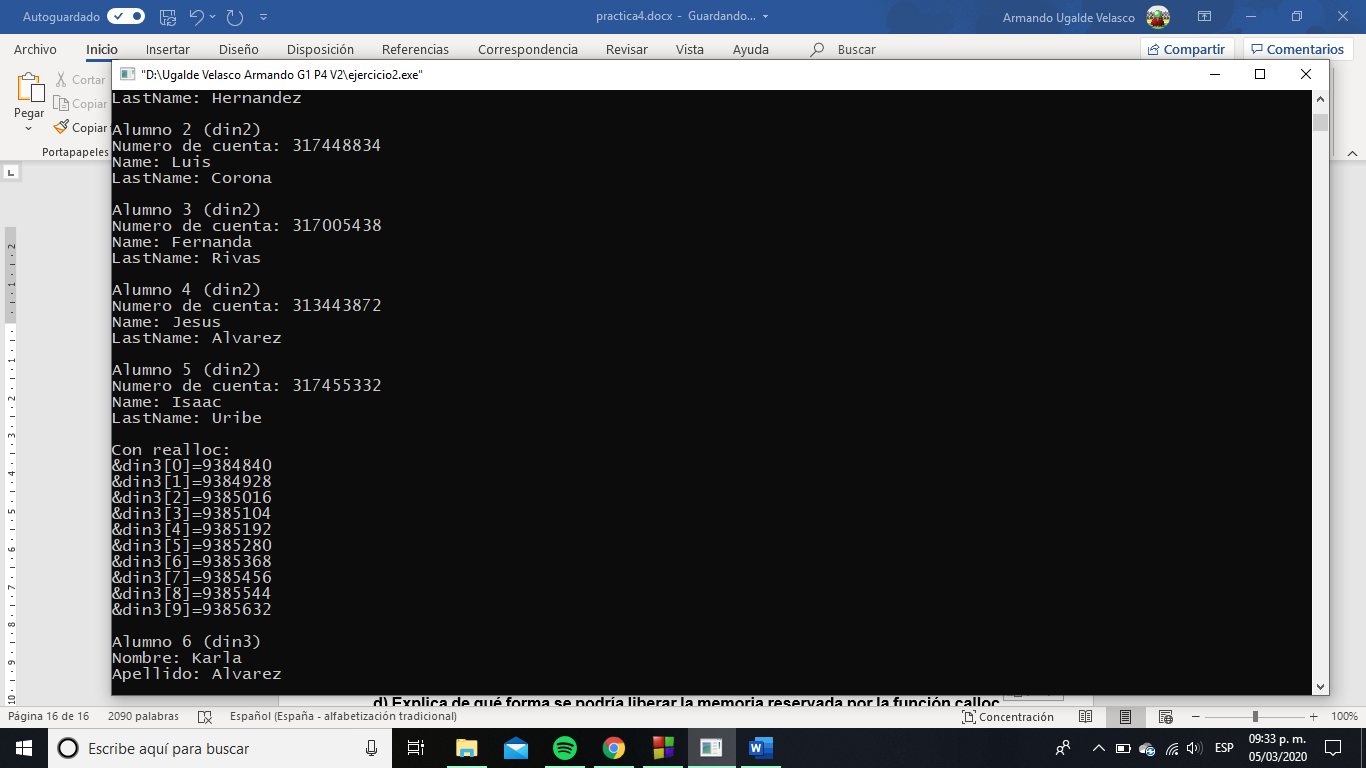
Como se puede observar, el ciclo inició en el índice 5 para asignar el valor correspondiente al sexto elemento, hasta llegar al décimo. Posteriormente se imprimieron los valores de los alumnos en el arreglo. Como se esperaba, los primeros 5 valores corresponden a los almacenados en el arreglo **din2**, que fue redimensionado.

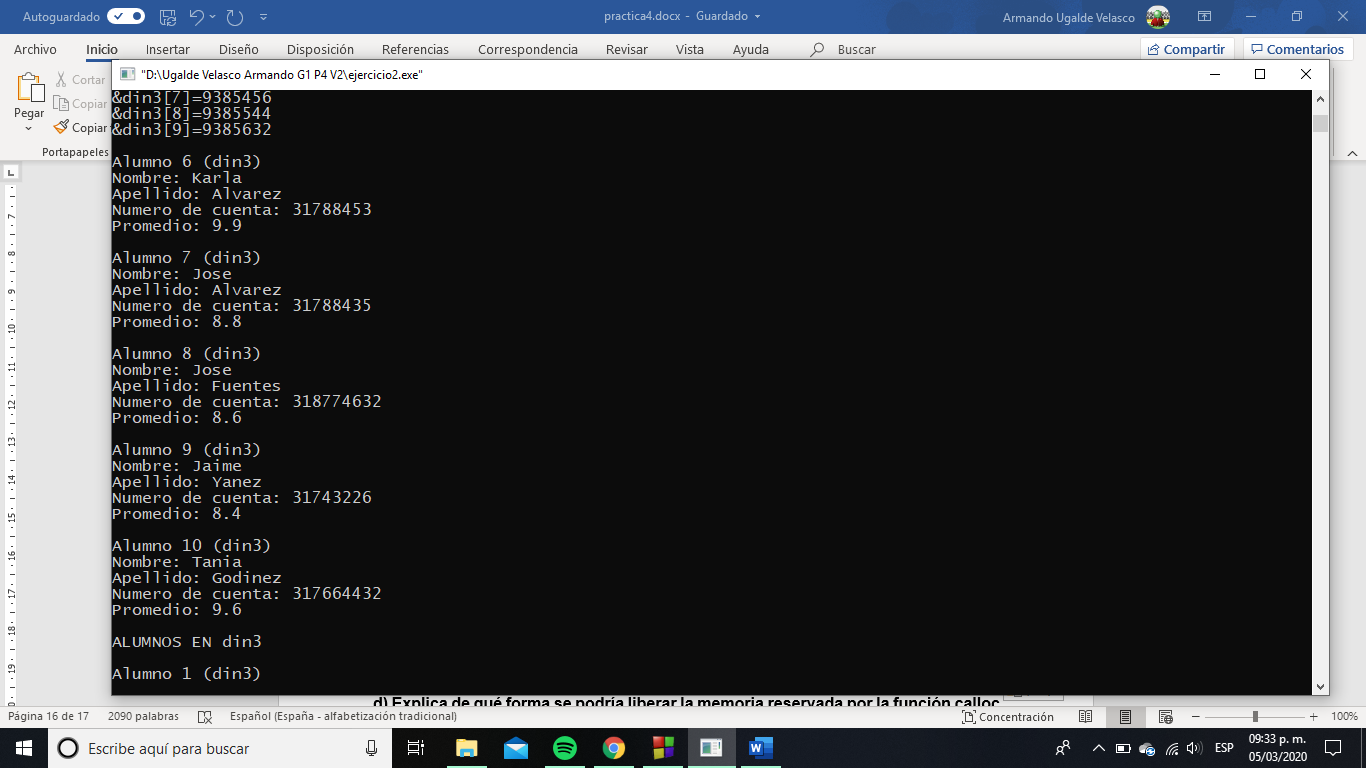
A continuación, se muestran capturas de pantalla de la ejecución del programa:

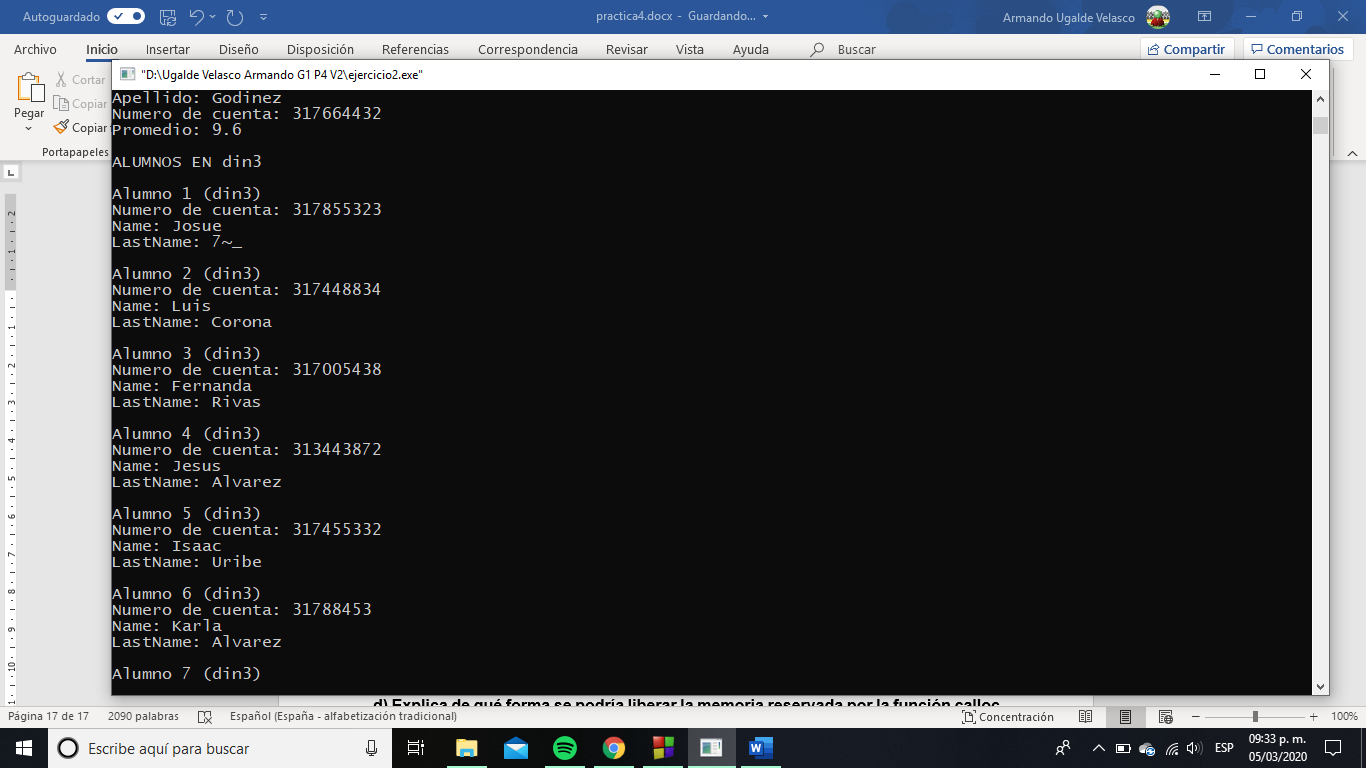


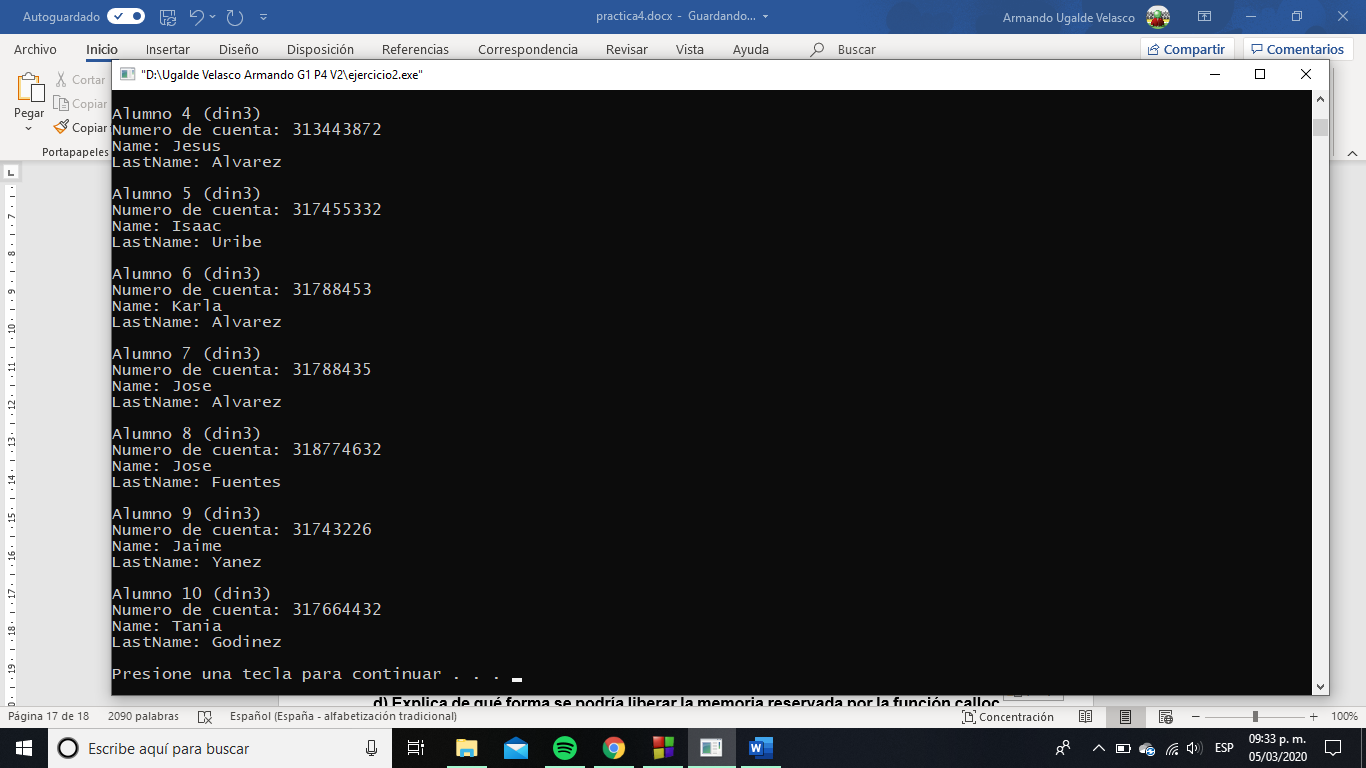












***Capturas de pantalla de la ejecución del programa***

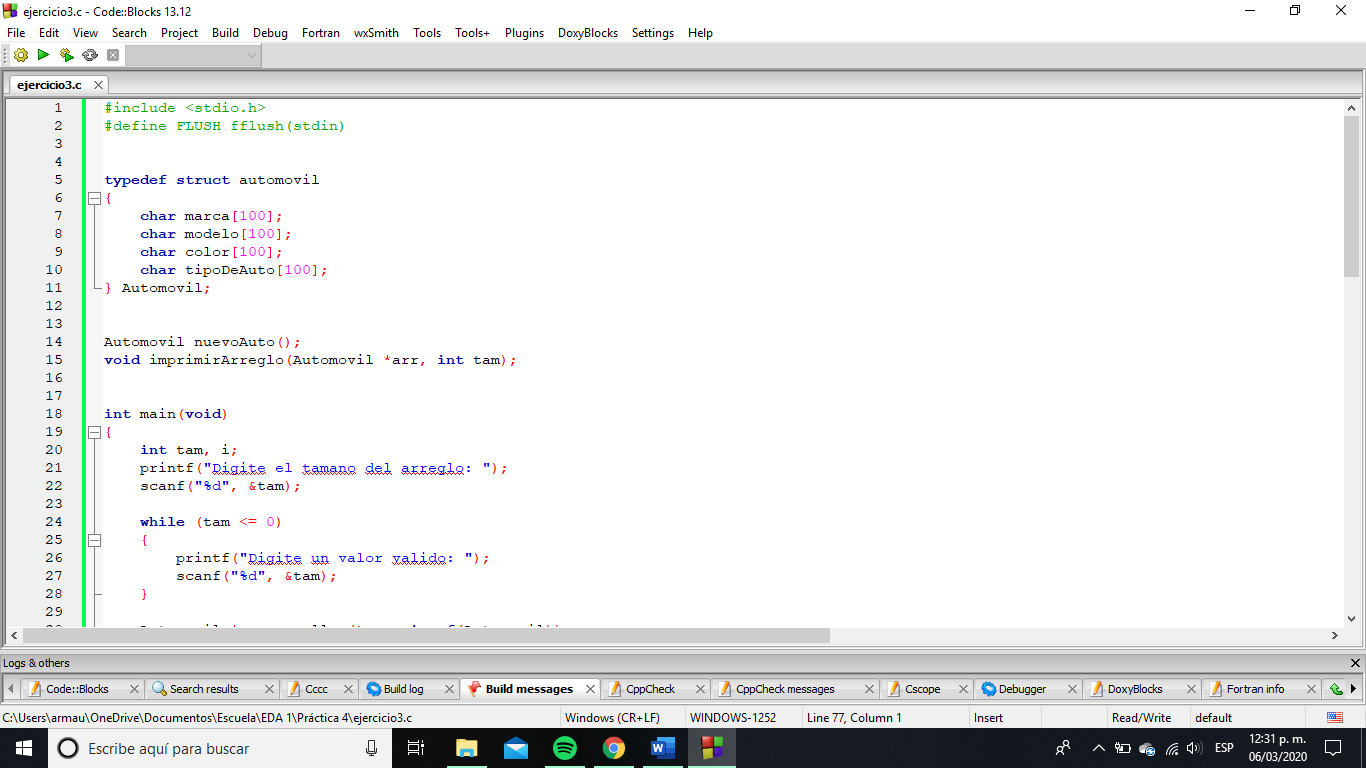
Se obtuvo la salida esperada. Como se mencionó, los primeros 5 alumnos almacenados en **din3** coincidieron con los almacenados en **din2**, corroborando el funcionamiento de la función **calloc.**

**d) Explica de qué forma se podría liberar la memoria reservada por la función calloc o malloc sin utilizar la función free.**

Podríamos utilizar la función **realloc**, tomando como argumentos el puntero cuya memoria deseamos liberar y el nuevo tamaño, es decir, 0 bytes.

**Ejercicio 3**

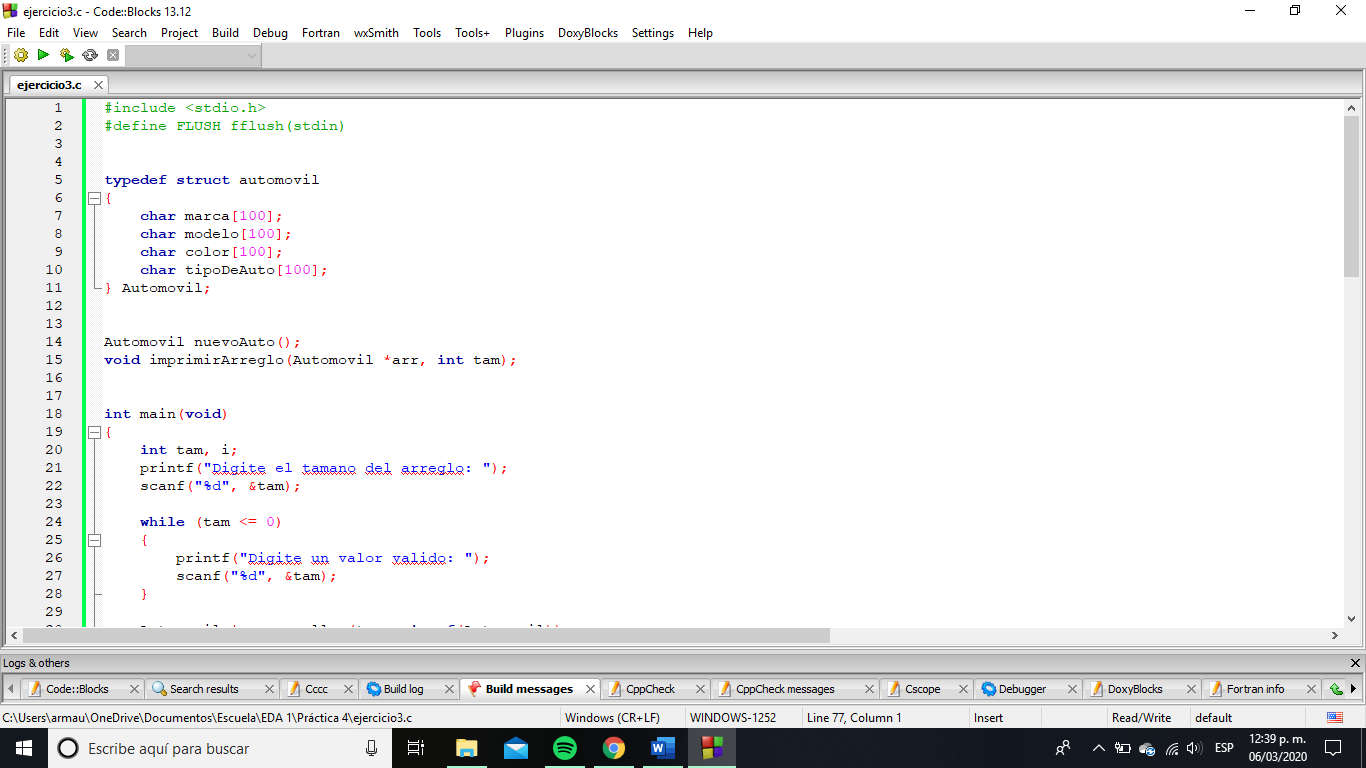
Para resolver el problema planteado, primero, se realizó la declaración de la estructura Automóvil de la siguiente forma:



***Declaración de estructura Automóvil y utilización de typedef***

Se utilizó la sentencia **typedef** para redefinir el tipo de dato de la estructura y facilitar su manipulación en el programa.

Para una manipulación eficiente de las estructuras, se realizaron las siguientes funciones auxiliares:



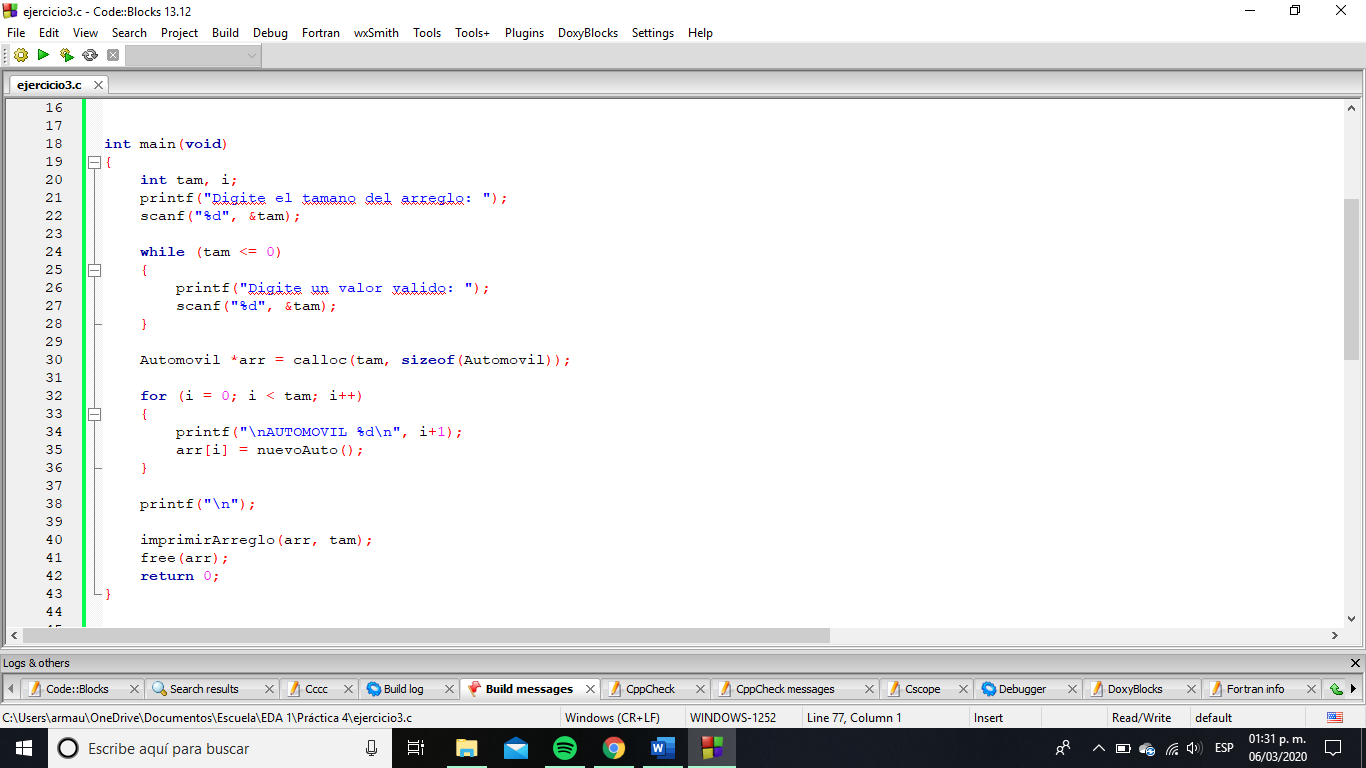
***Prototipos de funciones***

El objetivo de la función **nuevoAuto** es solicitar al usuario los datos necesarios para crear un ejemplar de estructura **Automovil,** y asignarlos a una variable de este tipo, la cual es el valor de retorno de la función.

El objetivo de la función **imprimirArreglo,** como lo indica su nombre, es imprimir todos los datos de cada elemento en un arreglo de Automóviles.

En la función **main**, primero se solicita y almacena el tamaño del arreglo.

Después, se almacena dinámicamente el arreglo con el tamaño almacenado, como se puede observar en la siguiente captura:



***Función main***

Posteriormente, se inicializa cada elemento del arreglo, utilizando la función auxiliar **nuevoAuto**.

Finalmente, se imprimen todos los elementos del arreglo, utilizando la función **imprimirArreglo**, y se libera la memoria asignada dinámicamente.

**Conclusiones**

Las funciones **malloc**, **calloc** y **realloc** nos permiten reservar y almacenar información de manera dinámica (*en tiempo de ejecución*).

Un punto que es importante recalcar es la diferencia que existe entre **malloc** y **calloc**: ambas asignan la cantidad de memoria dinámica solicitada, sin embargo, **calloc** inicializa los elementos al valor 0. Esta situación se presentó de forma clara en el ejercicio 1.

Además, durante la realización del ejercicio 2, se llegó a la conclusión de que la función **realloc** puede cumplir con el mismo objetivo que las funciones anteriores y la función **free**, utilizando los argumentos apropiados en cada caso.

En la práctica, las funciones mencionadas se utilizaron ampliamente para resolver los problemas planteados, y, además, se comprendieron sus principales implicaciones, por lo tanto, podemos decir que se cumplió el objetivo propuesto inicialmente.

Los ejercicios planteados fueron ideales para comprender el funcionamiento de las funciones. Sin embargo, en el ejercicio 1 se presentó una complicación. En la clase se mencionó que, cuando se utilizaba el “casteo” en la función **malloc**, era posible utilizar como argumento únicamente el número de enteros a almacenar, y no el tamaño total en bytes. En un inicio, se intentaron almacenar los enteros correspondientes con el argumento inicial (10). Si bien el programa se ejecutó correctamente y se almacenaron los datos correctos, al término de su ejecución se mostró un mensaje de error en la consola, cuya captura de pantalla se encuentra en el análisis del ejercicio. Por lo tanto, se procedió a cambiar el argumento por el tamaño total en bytes (40), lo cual provocó que el mensaje de error desapareciera.