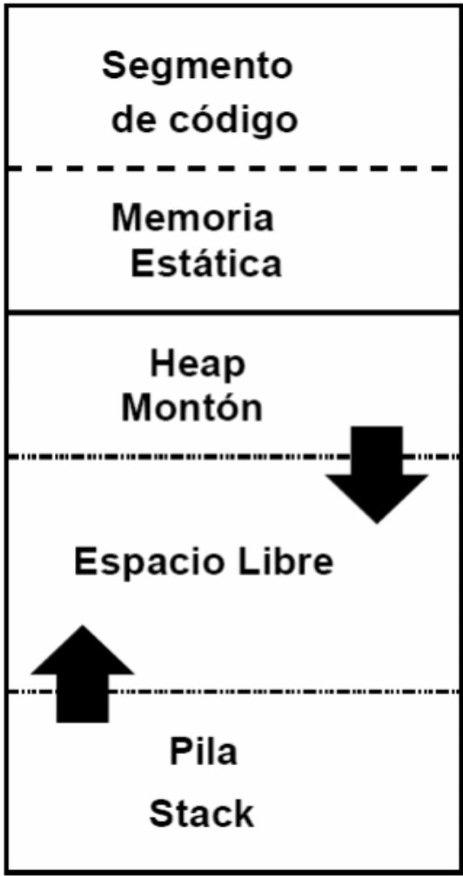
1. Memoria Dinámica
   1. Asignación dinámica de memoria

|  |
| --- |
| Las funciones que realizan un manejo dinámico de la memoria del sistema requieren todas ellas para su correcto funcionamiento la inclusión, mediante la directiva del prepocesador #include del archivo de cabecera <stdlib.h>. |

* + 1. Segmentos de memoria

|  |
| --- |
| Cada vez que se ejecuta cualquier programa, el mismo deberá pasar a memoria.  Los programas en memoria tienen varios segmentos, los cuales sirven para administrar el manejo de la memoria.   * **Segmento de Código:** En este segmento se guardan las instrucciones, en lenguaje máquina, de nuestro programa. * **Segmento de Memoria estática:** En este Segmento se guardan las variables globales del programa. * **Segmento del Stack:** Cada vez que se llama a una función entra en este segmento con toda su información y allí se guardan: * Los llamados a las funciones. * Los parámetros de las funciones llamadas. * Las variables locales. * Otra información necesaria para el funcionamiento del programa. * **Segmento del Heap:** En este segmento se guardan las variables que han sido creadas dinámicamente en tiempo de ejecución. |
|  |



* + 1. Reserva dinámica de memoria

|  |
| --- |
| En el lenguaje C, la reserva dinámica de memoria se realiza mediante el uso de funciones, explicaremos el uso de **malloc()** y **calloc()** :  La función **malloc()** tiene la forma:  **void*\**** malloc***(* unsigned int** numBytes ***);***  Siendo **'numBytes'** el número de bytes que se desean reservar.  **malloc()** devuelve un puntero al tipo de datos **'void'** *(sin tipo).*  Dicho puntero puede ser asignado a una variable puntero del tipo que definamos mediante una conversión forzada llamada“casting”.  Veamos un ejemplo:  **int*\**** a***;***    a ***= (*int*\*)*** malloc***(* sizeof*(*int*) );***  Y ahora podríamos realizar la siguiente asignación:  ***\****a ***=*** 3***;***  La función **calloc()** tiene la forma:  **void*\**** calloc***(* unsigned int** numElementos***,* unsigned int** numBytesElemento ***);***  A diferencia de **malloc()** la función **calloc()** inicializa a 0 el contenido de cada elemento de un array.  Podemos apreciar como en la definición de **calloc()** el primer parámetro es el numero de elementos y a continuación el tamaño del elemento.  Veamos un ejemplo:  **int*\**** a***;***    a ***= (*int*\*)*** calloc***(*** 20***,* sizeof*(*int*) );***  Aquí ya tenemos un array de veinte elementos en el cual cada uno de los elementos fue inicializado en cero.  Tanto la función **malloc()** como **calloc()**, devuelven un puntero nulo (**NULL**) si la reserva de memoria no puede realizarse, generalmente por falta de memoria disponible.  Por ello, antes de usar un puntero devuelto por la función **malloc()** o por cualquier otra función de reserva dinámica de memoria es imprescindible, con el fin de evitar posibles fallos en la ejecución del programa, comprobar que dicho puntero no es nulo (**NULL**).  **int*\**** a***;***  a ***= (*int*\*)*** malloc***(* sizeof*(*int*) );***    **if *(***a ***==*** NULL***)***  ***{***  printf***(***"NO QUEDA MEMORIA"***);***  ***}***  Veamos algunos ejemplos de reserva dinámica de memoria: |

|  |
| --- |
| Ejemplo 1***:***    **float*\**** a***;***  a ***= (*float*\*)*** malloc***(* sizeof*(*float*) );***  **if *(*** a ***==*** NULL ***)***  ***{***  exit***(***0***);*** /\* Salimos del programa \*/  ***}*** |
| Ejemplo 2***:***    **unsigned long int*\**** b***;***  **if*( (***b ***= (*unsigned long int*\*)*** malloc***(* sizeof*(*unsigned long int*) )) ==*** NULL***)***  ***{***  exit***(***0***);*** /\* Salimos del programa \*/  ***}*** |
| Ejemplo 3***:***    **typedef struct**  ***{***  **unsigned int** a***;***  **float** b***;***  **int *\****c***;***  ***}***sALFA***;***  sALFA***\**** d***;***  **if*( (***d ***= (***sALFA***\*)*** malloc***(* sizeof*(***sALFA***) ) ==*** NULL ***)***  ***{***  exit***(***0***);*** /\* Salimos del programa \*/  ***}*** |

* + 1. Redimensionamiento dinámico de memoria

|  |
| --- |
| El redimensionamiento dinámico de memoria intenta cambiar el tamaño de un bloque de memoria previamente asignado. El nuevo tamaño puede ser más grande o más pequeño.  Si el bloque se hace más grande, entonces el contenido anterior permanece sin cambios y la memoria es agregada al final del bloque.  Si el bloque se hace más pequeño entonces el contenido sobrante permanece sin cambios.  La función utilizada en el leguaje C para redimensionar memoria es **realloc()**.  La función **realloc()** tiene la forma: |

|  |
| --- |
| **void*\**** realloc***(* void*\**** ptr***,* unsigned int** size ***);*** |

|  |
| --- |
| Si bloque original no puede ser redimensionado, entonces **realloc()** intentará asignar un nuevo bloque de memoria y copiará el contenido anterior.  Por lo tanto, la función devolverá un nuevo puntero, este nuevo valor será el que deberá usarse.  Si no puede ser reasignada nueva memoria la función **realloc()** devuelve **NULL**.  **Ejemplo**:  Imaginemos que tenemos el puntero **'punteroA'** el cual apunta a un espacio de memoria de 2 chars de tamaño que fue reservado con la función **malloc()**a partir de la dirección 0x01. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | DIRECCION | ESTADO |  | DIRECCION | ESTADO |
|  | 0x00 | Otros datos |  | 0x00 | Otros datos |
| **punteroA -->** | 0x01 | ASIGNADO |  | 0x01 | LIBRE |
|  | 0x02 | ASIGNADO |  | 0x02 | LIBRE |
|  | 0x03 | Otros datos |  | 0x03 | Otros datos |
| **punteroB -->** | 0x04 | ASIGNADO | **punteroB -->** | 0x04 | ASIGNADO |
|  | 0x05 | ASIGNADO |  | 0x05 | ASIGNADO |
|  | 0x06 | Otros datos |  | 0x06 | Otros datos |
|  | 0x07 | LIBRE | **punteroA -->** | 0x07 | ASIGNADO |
|  | 0x08 | LIBRE |  | 0x08 | ASIGNADO |
|  | 0x09 | LIBRE |  | 0x09 | ASIGNADO |
|  | 0x0A | LIBRE |  | 0x0A | ASIGNADO |

|  |
| --- |
| Si se quiere reasignar la memoria a cuatro **char** en lugar de los dos, se hará: |

|  |
| --- |
| punteroAuxiliar ***= (*char*\*)*** realloc***(*** punteroA***,*** 4 ***\** sizeof*(*char*) );***  **if*(***punteroAuxiliar ***!=*** NULL***)***  ***{***  punteroA ***=*** punteroAuxiliar***;***  ***}***  **else**  ***{***  printf***(***"NO QUEDA MEMORIA"***);***  ***}*** |

|  |
| --- |
| Como se observa en la tabla de la derecha, **'punteroA'** cambió su dirección inicial, ya que de otra manera no se podían conseguir los 4 bytes consecutivos, debido a que el espacio de memoria debajo de **'punteroA'** ya estaba ocupado. |

* + 1. Liberación dinámica de memoria

|  |
| --- |
| La memoria dinámica reservada es eliminada siempre al terminar la ejecución del programa por el propio sistema operativo. Sin embargo, durante la ejecución del programa puede ser interesante, e incluso necesario, proceder a liberar parte de la memoria reservada con anterioridad y que ya ha dejado de ser necesario tener reservada. Esto puede realizarse mediante la función **free().**  La función **free()** tiene la forma:  **void** free***(* void*\**** p ***);***  Donde **'p'** es la variable de tipo puntero cuya zona de memoria asignada de forma dinámica queremos liberar.  Veamos un ejemplo de liberación de memoria:  **int*\**** a***;***  **if*( (***a ***= (*int*\*)*** malloc***(* sizeof*(*int*) )) ==*** NULL ***)***  ***{***  exit***(***0***);*** /\* Salimos del programa \*  ***}***  //...  free***(*** a ***);***  Un aspecto a tener en cuenta es el hecho de que el puntero a liberar no debe apuntar a nulo (**NULL**), pues en tal caso se producirá un fallo en el programa. Es por ello que cobra aún más sentido la necesidad de comprobar al reservar memoria de forma dinámica que la reserva se ha realizado de forma correcta, tal y como se explicó anteriormente. |

* + 1. Ejemplo de asignación, redimensionamiento y liberación de memoria

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  **typedef struct**  ***{***  **char** nombre***[***50***];***  **int** edad***;***  ***}***sPersona***;***  **int** main***()***  ***{***  **int** seguirCargando***;***  **int** i***;***    **int** auxNuevaLogitud***;***  **int** longitudPersonas ***=*** 1***;***    sPersona***\**** pArrayPersona***;***  sPersona***\**** pAuxPersona***;***      // Creamos el array de personas  pArrayPersona ***=*** malloc***(* sizeof*(***sPersona***) );***  **if*(***pArrayPersona ***==*** NULL***)***  ***{***  printf***(***"No hay lugar en memoria\n"***);***  exit***(***0***);***  ***}***  **while*(***1***)***  ***{***  printf***(***"Ingrese nombre: \n"***);***  scanf***(*** "%s"***, (***pArrayPersona ***+***logitudPersonas ***-***1***)->***nombre ***);***  printf***(***"Ingrese edad: \n"***);***  scanf***(*** "%d"***, &( (***pArrayPersona ***+***logitudPersonas***-***1***)->***edad ***) );***  printf***(***"Si desea cargar otra persona ingrese (1): \n"***);***  scanf***(***"%d"***, &***seguirCargando***);***  **if*(***seguirCargando ***==*** 1***)***  ***{***  // Calculamos el nuevo tamaño del array  auxNuevaLongitud ***=* sizeof*(***sPersona***) \**** logitudPersonas***;***  // Redimencionamos la lista  pAuxPersona ***=*** realloc***(*** pArrayPersona***,*** auxNuevaLogitud ***);***  **if*(***pAuxPersona ***==*** NULL***)***  ***{***  printf***(***"No hay lugar en memoria\n"***);***  **break*;***  ***}***  logitudPersonas***++;*** //Incremento el contador de personas  pArrayPersona ***=*** pAuxPersona***;***  ***}***  **else**  ***{***  **break*;***  ***}***  ***}***    **for*(***i ***=*** 0***;*** i ***<*** logitudPersonas***;*** i***++)***  ***{***  printf***(***"Nombre: %s - "***, (***pArrayPersona ***+***i***)->***nombre***);***  printf***(***"Edad: %d \n"***, (***pArrayPersona ***+***i***)->***edad***);***  ***}***    free***(*** pArrayPersona ***);*** // Liberamos la memoria  **return** 0***;***  ***}*** |

1. Ejercicios y videos

[VIDEO PARTE 1](https://youtu.be/cNB1arCvVOQ)

**Ejercicio 1:**

1)  Construir una función llamada **"getDynamicString"** que permita al usuario ingresar un texto y

devuelva un puntero a un espacio de memoria donde esta almacenado el texto ingresado.

Se requiere el uso de memoria dinámica.