	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I	Código:	MADO-19
		Versión:	01
		Página	33/151
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 04. Almacenamiento en tiempo de ejecución




Elaborado por:

M.C. Edgar E. García Cano
Ing. Jorge A. Solano Gálvez

Autorizado por:

M.C. Alejandro Velázquez Mena

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I	Código:	MADO-19
		Versión:	01
		Página	34/151
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 04. Almacenamiento en tiempo de ejecución.

Objetivo:

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

Actividades:

- Utilizar funciones para reservar memoria dinámica en lenguaje C.
- Almacenar información en los elementos reservados con memoria dinámica.

Introducción

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.

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.


Licencia GPL de GNU

El software presente en esta guía práctica es libre bajo la licencia GPL de GNU, es decir, se puede modificar y distribuir mientras se mantenga la licencia GPL.

```

/*
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

```

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I	Código:	MADO-19
		Versión:	01
		Página	35/151
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```

* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program. If not, see <http://www.gnu.org/licenses/>.
*
* Author: Jorge A. Solano
*
*/

```

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.

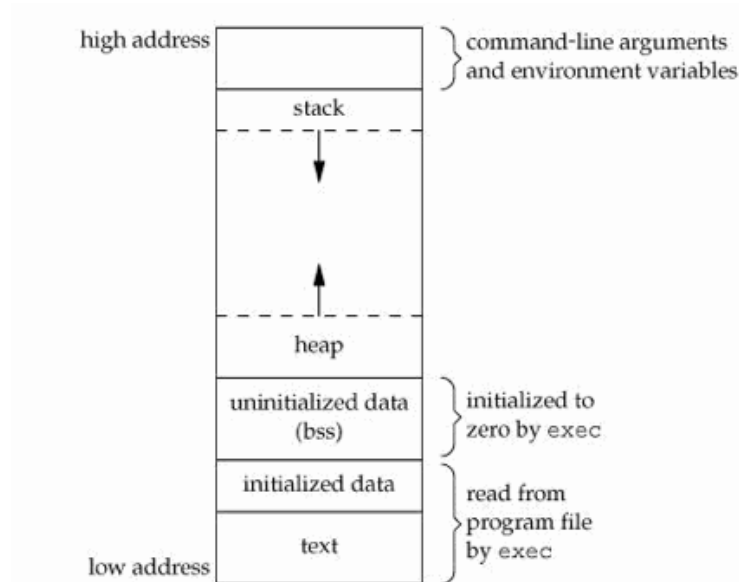



Figura 1. Regiones de la memoria RAM

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I	Código:	MADO-19
		Versión:	01
		Página	36/151
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

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.

La memoria que se define de manera explícita (estática) tiene una duración fija, que se reserva (al iniciar el programa) y libera (al terminar la ejecución) de forma automática. La memoria dinámica se reserva y libera de forma explícita.

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.

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).

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I	Código:	MADO-19
		Versión:	01
		Página	37/151
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

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("¿Cuántos 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;
}
```

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I	Código:	MADO-19
		Versión:	01
		Página	38/151
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

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("¿Cuántos 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]\n");
        printf("Se libera el espacio reservado.\n");
        free(arreglo);
    }
    return 0;
}
```

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I	Código:	MADO-19
		Versión:	01
		Página	39/151
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

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("¿Cuántos 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",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 tamaño del conjunto al doble.\n");
        num *= 2;
        arreglo2 = (int *)realloc (arreglo,num*sizeof(int));
        if (arreglo2 != NULL) {
```

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I	Código:	MADO-19
		Versión:	01
		Página	40/151
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```

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;
}

```


	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I	Código:	MADO-19
		Versión:	01
		Página	41/151
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Bibliografía



El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.

Ariel Rodríguez (2010). How knowing C and C++ can help you write better iPhone apps, part 1. [Figura 1]. Consulta: Enero de 2016. Disponible en: <http://akosma.com/2010/10/11/how-knowing-c-and-c-can-help-you-write-better-iphone-apps-part-1/>