Memoria Dinámica en C

Antonio Espín Herranz

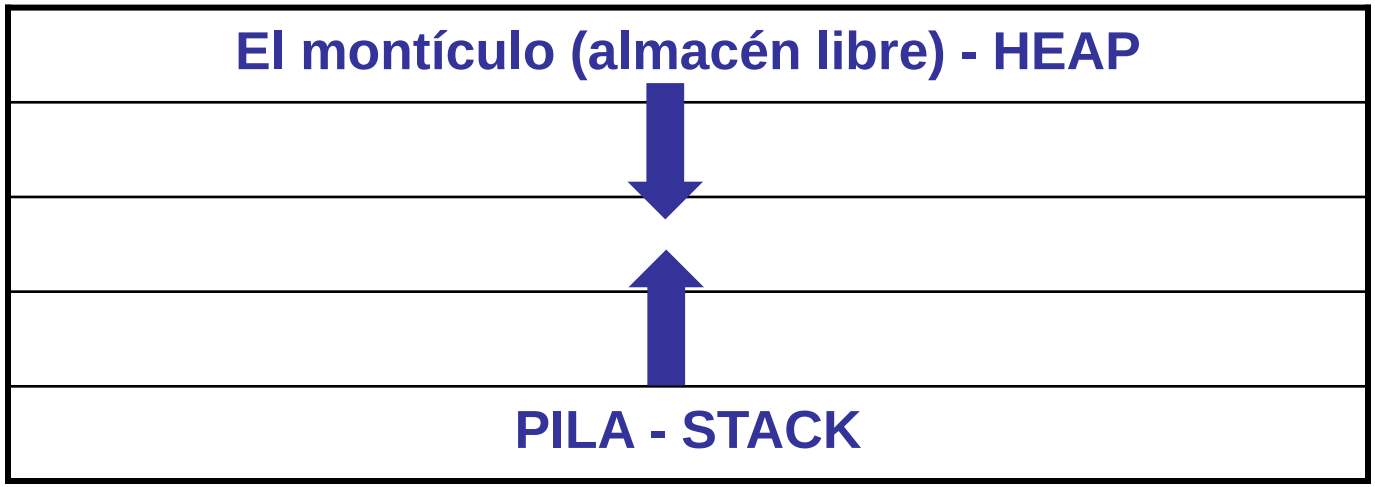
TRABAJAR CON MEMORIA DINÁMICA

- Memoria dinámica.
- Uso de la memoria dinámica.
- Concepto de memoria dinámica.
- Reserva y liberación de memoria dinámica en tiempo de ejecución.
- Utilización de matrices dinámicas.
- Almacenaje de datos dinámicos en archivos.
- Estructuras de Almacenamiento.

Memoria dinámica

- Se crea en tiempo de ejecución.
- Las variables hasta ahora definidas se crean en tiempo de compilación.
- Un programa puede crear y liberar memoria en cualquier momento de la ejecución.
- Es responsabilidad del programador liberar la memoria dinámica utilizada.

Esquema del mapa de la memoria



En la pila:

- Se almacenan la variables locales, globales.
- Se copia la información necesaria cuando se llama a una función.

En el montículo:

- Se alojan los bloques de memoria dinámica.

Uso de la memoria dinámica

- En C las funciones **malloc**(), **calloc**(), **realloc**() y **free**() asignan y liberan memoria de un bloque denominado el montículo del sistema.
- Definidas en **stdlib.h.**
- Las funciones malloc(), calloc(), realloc() asignan memoria utilizando asignación dinámica debido a que puede gestionar la memoria durante la ejecución de un programa.
- Estas funciones requieren generalmente un moldeado (**conversión de tipos**).

Concepto de memoria dinámica

- La memoria dinámica es un espacio de almacenamiento que se solicita en tiempo de ejecución.

- A medida que el proceso va necesitando espacio para más datos, va solicitando más memoria al sistema operativo para guardarlos.

- El medio para manejar la memoria que otorga el sistema operativo, es el puntero, puesto que no podemos saber en tiempo de compilación dónde nos dará huecos el sistema operativo (en la memoria de nuestro PC).

Reserva y liberación de memoria dinámica en tiempo de ejecución

- Función malloc(): Reserva espacio en la memoria dinámica.
- Prototipo: void *malloc(size_t n);
- Devuelve un puntero a void, tenemos que utilizar un casting o moldeo al tipo de datos deseado.
- Uso:

  tipo *puntero;

  puntero = (tipo *)malloc(tamaño en bytes);
- En caso de que no haya suficiente memoria, malloc devuelve NULL.

Reserva y liberación de memoria dinámica en tiempo de ejecución

- Se aconseja el uso de sizeof() para reservar memoria.
- Ejemplo:

```
float *bloqueMEM;
bloqueMEM = (float *)malloc(100*sizeof(float));
```

- En caso de reservar memoria para cadenas de caracteres, reservar 1 char de mas para contener el ‘\0’.

Ejemplo con char *

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void main(){
    char cad[121], *ptr;
    int lon;
    puts(“Introduce una línea de texto:”);
    gets(cad);
    lon=strlen(cad);
    // Reserve memoria para la longitud de la cadena+1.
    ptr = (char *)malloc((lon+1)*sizeof(char));
    strcpy(ptr, cad);
    printf(“ptr = %s“, ptr);
    free(ptr);
}
```

Utilización de matrices dinámicas

- Para definir matrices dinámicas, tenemos que utilizar punteros del tipo que queramos los elementos.

```
int *numeros, n, i;

printf("Introduzca el número de elementos:");
scanf("%d",&n);
numeros = (int *)malloc(n * sizeof(int));

for (i = 0 ; i < n ; i++){
        printf("Introduzca número %d: ", i);
        scanf("%d", &numeros[i]);
}
```

Uso de malloc() para array de mas dimensiones

```
// Reservar memoria para una tabla de 5 x 5 elementos.

int **tabla, i;
// Sitio para las 5 filas:
tabla = (int **)malloc(5 * sizeof(int *));

// Por cada fila reservamos para las 5 columnas.
for (i = 0 ; i < 5 ; i++ ){
    tabla[i] = (int *)malloc(5 * sizeof(int));
}
```

Liberación de la memoria

- Para liberar la memoria ocupada por malloc() utilizaremos la función free().

- Prototipo:
  - void free(void *);

- Ejemplo:

```
int *ad; // Reserva para un entero.
ad = (int *)malloc(sizeof(int));
char *s; // Reserva para 100 char.
s = (char *)malloc(100 * sizeof(char));

// Liberar ambos punteros:
free(ad);
free(s);
```

Función calloc()

- Reserva memoria dinámica para un número de elementos.
- Devuelve un puntero a void que tenemos que convertir.
- A diferencia de malloc inicializa la memoria.
- Si no hay suficiente memoria devuelve NULL.
- La forma de utilizarla:
  - puntero = calloc(número elementos, tamaño)
- Prototipo:
  - void *calloc(size_t n, size_t t);

Ejemplo de calloc()

```
#define N 5
double *p;
p = (double *)calloc(N, sizeof(double));

char *c, B[121];
puts(“Introduce una cadena:”);
gets(B);
c = (char *)calloc(strlen(B)+1, sizeof(char));
strcpy(c, B);
```

Función realloc()

- Al igual que las funciones malloc() y realloc() permite reservar memoria dinámica.
- realloc() permite ampliar el bloque de memoria inicialmente reservado.
- Devuelve un void *.
- Uso:
  - tipo *puntero;
  - puntero=(tipo *)realloc(puntero a bloque, tamaño total del nuevo bloque);

Ejemplo realloc()

- Reserva memoria para una cadena y a continuación reserva memoria para otra mas larga.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){
    char *cadena;
    int tam;
    tam = (strlen(“Primavera”)+1)*sizeof(char);
    cadena = (char *)malloc(tam);
    strcpy(cadena, “Primavera”);
    puts(cadena);

    tam+=(strlen(“ en Lupiana\n”)+1)*sizeof(char);
    cadena = (char *)realloc(cadena, tam);
    strcat(cadena, “ en Lupiana\n”);
    puts(cadena);
    free(cadena);
    return 0;
}
```