

Portada

Mapa de memoria

Stack Overflo

Fragmentació

rragmentacio

arrays

Forma 1

Forma 2

liberación

valgrind Ejemplo

Eiercicios

Curso de programación en C moderno (II Edición)

Neira Ayuso, Pablo Falgueras García, Carlos

Tema 6

Reserva dinámica de memoria



Índice

Portada

Mapa de memoria Ejemplo

Stack Overflo

Fragmentació

arrays multiD

Forma 1

liberació

valgrind Ejemplo

Ejercicio

- Mapa de memoria
 - Ejemplo
- 2 Stack Overflow
- 3 Fragmentación
- 4 Reserva de arrays multidimensionales
 - Forma 1 (la mala)
 - Forma 2 (la buena)
- 5 Uso tras liberación
- 6 Depuración con valgrind
 - Ejemplo
- 7 Ejercicios: Reserva dinámica de memoria



Mapa de memoria

Portada

Mapa de memoria Ejemplo

Stack Overflo

Fragmentació

arrays multiD

Forma 1

Uso tra liberaci

valgrind Ejemplo

Ejercicio

- Heap: Se almacena la memoria reservada dinámicamente con malloc
- Stack: Se almacenan las variables locales de cada llamada a función
- Globales: Todas las variables globales
- Constantes: Todas las constantes (números, cadenas, etc)
- Código: El programa en sí

Direcciones altas de memoria STACK **HEAP** Globales Constantes Código



```
Portada
```

```
3
                4
                5
                6
Eiemplo
                8
               10
               11
Fragmentación 12
               13
arravs
               14
               15
Forma 1
               16
Forma 2
               17
               18
               19
               20
               21
               22
Ejemplo
               23
               24
               25
               26
               27
```

```
#include <stdio.h>
  #include <stdlib.h>
   int *array alloc(int n)
     int *p = (int *) malloc(n * sizeof(int))
     return p:
   static const int tam = 3;
   int main()
     int i:
     int *array;
     array = array alloc(tam);
     /* Inicializamos */
     for (i = 0; i < tam; i++)
       arrav[i] = i:
     /* Imprimimos */
     for (i = 0; i < tam; i++)
       printf("%d\n", array[i]);
     free (array);
28
29
     return 0;
30
```

Direcciones altas de memoria

int i STACK int *array Globales const int i Constantes "%d\n" Código



#include <stdio.h>

Portada

```
Eiemplo
```

arravs

```
Forma 1
Forma 2
```

Ejemplo

```
#include <stdlib.h>
            3
              int *array alloc(int n)
            4
            5
            6
                 int *p = (int *) malloc(n * sizeof(int))
                 return p:
            8
           10
               static const int tam = 3;
           11
Fragmentación 12
              int main()
           13
           14
                 int i:
           15
                 int *array;
           16
           17
                 array = array alloc(tam);
           18
           19
                 /* Inicializamos */
                 for (i = 0; i < tam; i++)
           20
           21
                   arrav[i] = i:
           22
           23
                 /* Imprimimos */
                 for (i = 0; i < tam; i++)
           24
                   printf("%d\n", array[i]);
           25
           26
           27
                 free (array);
           28
           29
                 return 0;
           30
```

Direcciones altas de memoria

STACK



Globales Constantes



Código



Portada

```
Eiemplo
Fragmentación 12
arravs
Forma 1
Forma 2
```

Ejemplo

```
#include <stdio.h>
  #include <stdlib.h>
3
   int *array alloc(int n)
4
5
6
     int *p = (int *) malloc(n * sizeof(int))
     return p:
8
10
   static const int tam = 3;
11
   int main()
13
14
     int i:
15
     int *array;
16
17
     array = array alloc(tam);
18
19
     /* Inicializamos */
     for (i = 0; i < tam; i++)
20
21
       arrav[i] = i:
22
23
     /* Imprimimos */
     for (i = 0; i < tam; i++)
24
       printf("%d\n", array[i]);
25
26
27
     free (array);
28
29
     return 0;
30
                                                4□ → 4周 → 4 = → 4 = → 9 Q P
```

```
Direcciones altas de memoria
                       int i
      STACK
                      int *array
                       int n
                       int *p -
       HEAP
                   bloque de 12 bytes
   Globales
                     const int i
Constantes
                       "%d\n"
                    Código
```



```
Portada
```

```
#include <stdlib.h>
            3
            4
               int *array alloc(int n)
            5
            6
                 int *p = (int *) malloc(n * sizeof(int))
                 return p:
Eiemplo
            8
           10
               static const int tam = 3;
           11
Fragmentación 12
               int main()
            13
arravs
           14
                 int i:
           15
                 int *array;
Forma 1
           16
Forma 2
           17
                 array = array alloc(tam);
           18
            19
                 /* Inicializamos */
                 for (i = 0; i < tam; i++)
            20
            21
                    arrav[i] = i:
            22
Ejemplo
            23
                 /* Imprimimos */
                 for (i = 0; i < tam; i++)
           24
                    printf("%d\n", array[i]);
           25
            26
            27
                 free (array);
           28
           29
                 return 0;
```

30

#include <stdio.h>

```
Direcciones altas de memoria
                        int i
      STACK
                       int *array_
                    bloque de 12 bytes
        HEAP
   Globales
                      const int i
Constantes
                        "%d\n"
                     Código
               Direcciones baias de memoria
```

4□ → 4周 → 4 = → 4 = → 9 Q P



```
Portada
```

```
#include <stdio.h>
              #include <stdlib.h>
            3
            4
               int *array alloc(int n)
            5
                 int *p = (int *) malloc(n * sizeof(int))
            6
                 return p:
Eiemplo
            8
           10
               static const int tam = 3;
           11
Fragmentación 12
               int main()
            13
arravs
           14
                 int i:
           15
                 int *array;
Forma 1
           16
Forma 2
           17
                 array = array alloc(tam);
           18
            19
                 /* Inicializamos */
                 for (i = 0; i < tam; i++)
            20
            21
                    arrav[i] = i:
            22
Ejemplo
            23
                 /* Imprimimos */
                 for (i = 0; i < tam; i++)
           24
                    printf("%d\n", array[i]);
           25
            26
            27
                 free (array);
           28
           29
```

return 0;

30

```
Direcciones altas de memoria
                       int i
      STACK
                     int *array
   Globales
                    const int i
Constantes
                       "%d\n"
                    Código
```

Direcciones bajas de memoria

4□ → 4周 → 4 = → 4 = → 9 Q P



Portada

El uso de funciones recursivas puede ocasionar un agotamiento de la pila

```
Eiemplo
```

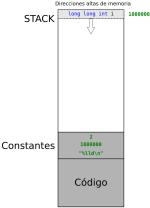
Stack Overflow

arravs

Forma 1 Forma 2

Ejemplo

```
#include <stdio.h>
   long long int sum all(long long int i)
5
     if (i >= 2)
       return i + sum_all(i - 1);
       return i:
8
9
10
11
   int main()
12
     printf("%||d\n", sum a||(1000000));
13
14
15
     return 0;
16
```





Portada

Mapa de memoria

Ejemplo Stack

Overflow

Fragmentació

arrays

Forma 1 Forma 2

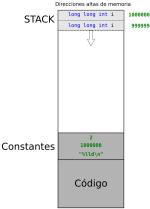
Uso tras liberación

valgrind Ejemplo

```
Ejercicio
```

El uso de funciones recursivas puede ocasionar un agotamiento de la pila

```
#include <stdio.h>
   long long int sum all(long long int i)
5
     if (i >= 2)
       return i + sum all(i - 1);
     else
8
       return i:
9
10
11
   int main()
12
     printf("%||d\n", sum a||(1000000));
13
14
15
     return 0;
16
```





Portada

El uso de funciones recursivas puede ocasionar un agotamiento de la pila

```
Eiemplo
```

Stack Overflow

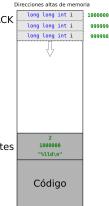
arravs

Forma 1 Forma 2

Ejemplo

```
4
 5
 8
 9
10
12
13
14
15
16 }
```

```
STACK
  #include <stdio.h>
2
  long long int sum all(long long int i)
    if (i \ge 2)
      return i + sum all(i - 1);
    else
      return i:
  int main()
                                                 Constantes
                                                                 1000000
    printf("%lld \n", sum all(1000000));
                                                                 "%lld\n"
    return 0:
                                                                Código
```





Portada

El uso de funciones recursivas puede ocasionar un agotamiento de la pila

```
memoria
Ejemplo
```

Overflow

Fragmentació

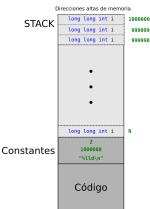
arrays multiD

Forma 1 Forma 2

liberación

Ejemplo

```
#include <stdio.h>
2
   long long int sum all(long long int i)
4
5
     if (i \ge 2)
       return i + sum all(i - 1);
     else
8
       return i:
9
10
   int main()
12
13
     printf("%lld \n", sum all(1000000));
14
15
     return 0:
16 }
```





Fragmentación

Portada

Fragmentos pequeños de memoria libre entre bloques de memoria reservada.

```
Eiemplo
Fragmentación
```

arravs Forma 1 Forma 2

Ejemplo

```
#include <stdio.h>
   #include <stdlib.h>
   int main()
     void *p1 = malloc(2 * 1024);
     void *p2 = malloc(1 * 1024);
     void *p3 = malloc(2 * 1024):
10
     void *p;
11
12
     printf("%|d\n", (long int)p1 / 1024);
13
     printf("%|d\n", (long int)p2 / 1024);
14
     printf("%|d\n", (long int)p3 / 1024):
15
16
     free (p2);
17
     p = malloc(2 * 1024);
18
19
     printf("\n%ld\n", (long int)p / 1024);
20
21
     return 0:
22
```

2 KiB

1 KiB

2 KiB



Fragmentación

Portada

Fragmentos pequeños de memoria libre entre bloques de memoria reservada.

```
#include <stdio.h>
              #include <stdlib.h>
Eiemplo
              int main()
            5
                 void *p1 = malloc(2 * 1024);
Fragmentación
                 void *p2 = malloc(1 * 1024);
            7
                 void *p3 = malloc(2 * 1024);
            8
arravs
           10
                 void *p;
Forma 1
           11
Forma 2
                 printf("%|d\n", (long int)p1 / 1024);
           12
                 printf("%|d\n", (long int)p2 / 1024);
           13
                 printf("%|d\n", (long int)p3 / 1024);
           14
           15
           16
                 free (p2);
           17
                 p = malloc(2 * 1024):
Ejemplo
           18
           19
                 printf("\n\%ld\n", (long int)p / 1024);
           20
           21
                 return 0:
           22
```

2 KiB

2 KiB



Fragmentación

Portada

Fragmentos pequeños de memoria libre entre bloques de memoria reservada.

```
#include <stdio.h>
              #include <stdlib.h>
Eiemplo
              int main()
            5
                 void *p1 = malloc(2 * 1024);
Fragmentación
                 void *p2 = malloc(1 * 1024);
                 void *p3 = malloc(2 * 1024);
            8
arravs
           10
                 void *p;
Forma 1
           11
Forma 2
                 printf("%|d\n", (long int)p1 / 1024);
           12
                 printf("%|d\n", (long int)p2 / 1024);
           13
                 printf("%|d\n", (long int)p3 / 1024);
           14
           15
           16
                 free (p2);
           17
                 p = malloc(2 * 1024):
Ejemplo
           18
           19
                 printf("\n\%ld\n", (long int)p / 1024);
           20
           21
                 return 0:
```

22

```
2 KiB
2 KiB
2 KiB
```



Forma 1 (la mala)

Portada

Eiemplo

arravs

Forma 1

Forma 2

Ejemplo

Un malloc por cada fila del array (array de punteros)

```
int main()
3
     int i. i:
4
     int **arrav:
5
6
     /* Reservamos */
     array = (int **) malloc(ROWS * sizeof(int *));
     for (i = 0; i < ROWS; i++)
       array[i] = (int *)malloc(COLS * sizeof(int)); // Falta comprobar
10
     /* Inicializamos */
11
     for (i = 0; i < ROWS; i++)
12
13
       for (j = 0; j < ROWS; j++)
14
         arrav[i][i] = i * 10 + i:
15
     /* Imprimimos */
16
17
     for (i = 0: i < ROWS: i++) {
18
       for (i = 0; i < ROWS; i++) {
         printf("%02d ", array[i][i]);
19
20
21
       printf("\n");
22
23
     /* Liberamos */
24
25
     for (i = 0; i < ROWS; i++)
26
       free (array[i]);
27
28
     return 0;
                                               4□ > 4同 > 4 = > 4 = > ■ 90○
29
```



Forma 1 (la mala)

Portada

Mapa de memoria

Stack

Fragmentació

Forma 1

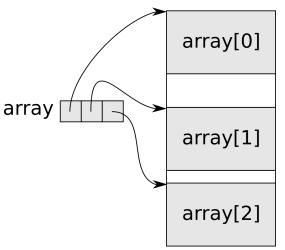
Forma 2

liberaci

valgrind Ejemplo

Eiercicios

No se garantiza continuidad entre los bloques reservados





Forma 2 (la buena)

```
Portada
              #define ROWS 3
              #define COLS 3
            3
            4
              int main()
            5
            6
                int i, j;
            7
                int *arrav:
Eiemplo
            8
            9
                /* Reservamos */
           10
                array = (int *) malloc(ROWS * COLS * sizeof(int));
           11
                if (!array) {
                   printf("Can't allocate the array\n");
           12
           13
                  return -1;
           14
arravs
           15
                /* Inicializamos */
           16
Forma 1
           17
                for (i = 0; i < ROWS; i++)
Forma 2
           18
                   for (j = 0; j < COLS; j++)
                     *(array + i * COLS + j) = i * 10 + j;
           19
           20
                /* Imprimimos */
           22
                for (i = 0; i < ROWS; i++) {
Ejemplo
           23
                   for (i = 0; j < COLS; j++) {
           24
                     printf("%02d ", *(array + i * COLS + j));
           25
           26
                   printf("\n");
           27
           28
           29
                free(array); /* Liberamos */
           30
                return 0:
                                                          ◆□→ →□→ →□→ □
                                                                                      90 Q
           31 }
```



Uso tras liberación

Portada

arravs

```
#include <stdio.h>
               #include <stdlib.h>
               #include <string.h>
             4
             5
               struct person
             6
             7
                  char name[200];
Eiemplo
                  char surname [200];
             8
             9
                  unsigned char age;
            10
               }:
            11
Fragmentación 12
               int main()
            13
            14
                  struct person *p;
            15
                  p = (struct person *) malloc(sizeof(struct person));
            16
Forma 1
            17
Forma 2
            18
                  strcpy(p->name, "Bob");
Uso tras
            19
                  strcpy(p->surname, "Smith");
liberación
            20
                  p\rightarrow age = 20;
            21
            22
                  free(p);
Ejemplo
            23
                  printf("name = %s \ n", p->name):
            24
            25
                  printf("surname = %s\n", p->surname);
            26
                  printf("age
                                    = %d n'', p \rightarrow age);
            27
            28
                  return 0:
            29
```



Depuración con valgrind

Portada

Mapa de memoria

Stack Overflo

Fragmentació

multiD

Forma 2

Uso tra

valgrind Ejemplo

Eiercicio

- Valgrind es una herramienta que nos avisa de los errores cometidos en el manejo de memoria
- Un leak o fuga de memoria es un error de programación que hace que la memoria reservada no se libere cuándo ya no se usa. Un programa que no libere correctamente la memoria reservada puede acabar consumiendo toda la memoria del sistema y dejarlo inutilizado.
- Para que valgrind nos muestre más información podemos hacer dos cosas:
 - Compilar con símbolos de depuración (gcc -g)
 - Ejecutar valgrind con el flag -leak-check=full



Portada

Mapa de memoria Ejemplo

Stack

. ruginentudio

multiD Forma 1

Forma 1 Forma 2

liberació

valgring

Ejemplo

Ejercicios

```
#include <stdio.h>
  #include <stdlib.h>
3
4
   int main()
5
6
     int *p = (int *) malloc(20);
7
8
     if (!p)
       return EXIT FAILURE;
9
10
11
     printf("La direccion de p es %p\n", p);
12
13
     /* free(p) */
14
15
     return 0;
16
```



Portada

gcc -g valgrind.c -o valg_ej valgrind --leak-check=full valg_ej

```
Memcheck, a memory error detector
Eiemplo
             Command: valg ej
              La direccion de p es 0x51d7040
            6
            7
             HEAP SUMMARY:
            8
                  in use at exit: 20 bytes in 1 blocks
                total heap usage: 1 allocs, 0 frees, 20 bytes allocated
arravs
           10
           11
              20 bytes in 1 blocks are definitely lost in loss record 1 of 1
Forma 1
           12
                 at 0x4C2ABD0: malloc (in /usr/lib /...)
Forma 2
                 by 0x4004F7: main (valgrind.c:6)
           13
           14
           15
             LEAK SUMMARY:
           16
                 definitely lost: 20 bytes in 1 blocks
           17
                 indirectly lost: 0 bytes in 0 blocks
Ejemplo
           18
                   possibly lost: 0 bytes in 0 blocks
           19
                 still reachable: 0 bytes in 0 blocks
           20
                      suppressed: 0 bytes in 0 blocks
           21
             For counts of detected and suppressed errors, rerun with: -v
             ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```



Ejercicios: Reserva dinámica de memoria

Portada

Mapa de memoria

Stack Overflo

Fragmentació

arravs

Forma 1

Forma 2

Uso tra

valgrind Ejemplo

Ejercicios

