

Sesión 06: *Structs* y memoria dinámica

Programación para Sistemas

Ángel Herranz

Otoño 2018

Universidad Politécnica de Madrid

En capítulos anteriores. . .

Sesión 0: Presentación

Sesión 1: Contacto C

Sesión 2: Ejecutando C

Sesión 3: Tipos básicos

Sesión 4: Módulos

Sesión 5: Arrays y Strings

Sesión 6: Punteros

En el capítulo de hoy...

- *Structs*
- Memoria dinámica

Structs

A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling. (Structures are called records in some languages, notably Pascal.) [...]

Capítulo 6, K&R.


struct ii

- Empezamos creando una variable para representar un punto en coordenadas cartesianas enteras

```
struct {  
    int x;  
    int y;  
} a;
```

- El código anterior **declara la variable a**,
- como un **registro (*struct*)**,
- con **dos atributos (*members*) x e y** de tipo entero,
- accesibles con la sintaxis **a.x** y **a.y**

Sintaxis *popular*

 Escribe un programa con dos structs a y b

```
struct {  
    int x;  
    int y;  
} a, b;
```

y explora la sintaxis de **struct**

- Ideas:

```
a.x = 1;  
printf("x == %i\n", a.x);  
sizeof(a)  
b = a;
```

struct iii

- Si observas con detalle las declaraciones anteriores, la frase

```
struct {int x; int y;}
```

se puede considerar como **un nuevo tipo** que se puede declarar con una *etiqueta (tag)* de esta forma

```
struct point {  
    int x;  
    int y;  
};
```

- Ahora la **etiqueta point** nos permite declarar variables así:
struct point a, b;

struct iv

- Por supuesto, es posible declarar **structs de structs** y **arrays de structs**

```
struct rectangulo {  
    struct point so;  
    struct point ne;  
};
```

```
struct rectangulo r;  
struct point h[6];
```

Memoria dinámica



Ordenar enteros

- Escribe un programar que ordene enteros de menor a mayor
- La entrada estándar tiene
 - Un entero positivo n en la primera línea
 - n enteros en las n siguientes líneas
- La salida de tu programar tiene los n enteros después de la primera línea ordenados de menor a mayor

Ordenar enteros

- Escribe un programar que ordene enteros de menor a mayor
- La entrada estándar tiene
 - Un entero positivo n en la primera línea
 - n enteros en las n siguientes líneas
- La salida de tu programar tiene los n enteros después de la primera línea ordenados de menor a mayor

Evita consumir más memoria de la necesaria

`scanf = printf-1`

- `scanf` ya visto en clase y en ejercicios

```
int i, n, *datos;
```

```
...
```

```
scanf("%d", &n);
```

`scanf` = `printf`⁻¹

- `scanf` ya visto en clase y en ejercicios

```
int i, n, *datos;
```

```
...
```

```
scanf("%d", &n);
```

- El operador `&` se puede aplicar a cualquier *lvalue*

```
for (i = 0; i < n; i++)
```

```
    scanf("%d", &datos[i]);
```

`scanf` = `printf`⁻¹

- `scanf` ya visto en clase y en ejercicios

```
int i, n, *datos;  
...  
scanf("%d", &n);
```

- El operador `&` se puede aplicar a cualquier *lvalue*

```
for (i = 0; i < n; i++)  
    scanf("%d", &datos[i]);
```

- Pero siempre podemos usar *aritmética de punteros*

```
for (i = 0; i < n; i++)  
    scanf("%d", datos+i);
```

Bubble

```
for (i = 0 ; i < n - 1; i++)  
    for (j = 0 ; j < n - i - 1; j)  
        if (datos[j] > array[j + 1])  
            intercambiar(&datos[j], &datos[j+1]);
```


¿Memoria suficiente?

- Hasta ahora sólo podíamos hacer esto

```
#define MAX 1000000
```

```
...
```

```
int datos[MAX]
```

- Pero... si hay **menos** de 1000000, **desperdiciamos memoria**
- Y si hay **más** de 1000000, **tenemos un problema**

Solicitud de memoria en tiempo de ejecución

- En lugar de establecer la memoria en tiempo de compilación debemos hacerlo en tiempo de ejecución
- Nada de automatismo en C: solicitud al sistema operativo
- En la biblioteca estándar¹ (**#include** <stdlib.h>)

void *malloc(size_t size);

- *The malloc() function allocates size bytes and returns a pointer to the allocated memory. The memory is not initialized. On error, these functions return NULL*

¹man 3 malloc

¿Cuánta memoria hay que pedir en bytes?

```
int n;
```

```
int *datos;
```

```
scanf("%d", &n);
```

```
datos = malloc(      ?      );
```

```
/* datos es un puntero a un bloque de  
   memoria en el que caben n enteros,  
   manejable como un array */
```

¿Cuánta memoria hay que pedir en bytes?

```
int n;
```

```
int *datos;
```

```
scanf("%d", &n);
```

```
datos =      ?   malloc(n * sizeof(int));
```

```
/* datos es un puntero a un bloque de  
   memoria en el que caben n enteros,  
   manejable como un array */
```

¿Cuánta memoria hay que pedir en bytes?

```
int n;
```

```
int *datos;
```

```
scanf("%d", &n);
```

```
datos = (int *) malloc(n * sizeof(int));
```

```
/* datos es un puntero a un bloque de  
   memoria en el que caben n enteros,  
   manejable como un array */
```

Ordenar enteros

- Escribe un programar que ordene enteros de menor a mayor
- La entrada estándar tiene
 - Un entero positivo n en la primera línea
 - n enteros en las n siguientes líneas
- La salida de tu programar tiene los n enteros después de la primera línea ordenados de menor a mayor

Ordenar enteros

- Escribe un programar que ordene enteros de menor a mayor
- La entrada estándar tiene
 - Un entero positivo n en la primera línea
 - n enteros en las n siguientes líneas
- La salida de tu programar tiene los n enteros después de la primera línea ordenados de menor a mayor

Evita consumir más memoria de la necesaria

while (n) Ordenar

2
7
1
3
1
4
2
0

Entrada

1
7

1
2
4

Salida

¿Problema?

liberar la memoria solicitada una vez usada

- En la biblioteca estándar² (**#include** <stdlib.h>)

void free(void *ptr);

- The *free()* function *frees the memory space* pointed to by *ptr*, which must have been returned by a *previous call to malloc()*. Otherwise, or if *free(ptr)* has already been called before, *undefined behavior occurs*. If *ptr* is *NULL*, *no operation* is performed.

²man 3 malloc

Liberar despues de cada ordenación

```
while(n) {  
    /* Solicitar memoria */  
    datos = (int *) malloc(n * sizeof(int));  
    /* Leer enteros y ordenarlos */  
    ...  
    /* Imprimir el array ya ordenado */  
    ...  
    /* Liberar memoria */  
    free(datos);  
    /* Leer siguiente n */  
    scanf("%d", &n);  
}
```



Memory leaks

Cuando se nos olvida liberar memoria



Memory leaks

Cuando se nos olvida liberar memoria

Segmentation fault

Cuando se nos olvida solicitar memoria



Memory leaks

Cuando se nos olvida liberar memoria

Segmentation fault

Cuando se nos olvida solicitar memoria
o usamos más allá de la solicitada



Memory leaks

Cuando se nos olvida liberar memoria


Segmentation fault

Cuando se nos olvida solicitar memoria
o usamos más allá de la solicitada


Comportamiento indefinido

Cuando liberamos memoria no solicitada

¿Cuánta memoria puedes pedir?

-  Escribe un programa que escriba el tamaño de memoria máximo que puedes solicitar (en bytes).

¿Cuánta memoria puedes pedir?

 Escribe un programa que escriba el tamaño de memoria máximo que puedes solicitar (en bytes).

- Idea:

1, 2, 4, 8, 16, 32, 64, 128, 256,
192, 224,
208,
200, 204
¡Bingo!

malloc es aplicable a cualquier tipo

```
char *s = (char *) malloc(n * sizeof(char));  
double *reales = (double *) malloc(n * sizeof(double));  
struct rectangulo *rects =  
    (struct rectangulo *)  
    malloc(n * sizeof(struct rectangulo));
```

³Relax: espero que podamos entender la sintaxis para declarar la variable vectores al final de la asignatura

malloc es aplicable a cualquier tipo

```
char *s = (char *) malloc(n * sizeof(char));  
double *reales = (double *) malloc(n * sizeof(double));  
struct rectangulo *rects =  
    (struct rectangulo *)  
    malloc(n * sizeof(struct rectangulo));
```

Incluso³

```
char **cadenas =  
    (char **) malloc(n * sizeof(char **));  
int (*vectores)[10] =  
    (int (*)[]) malloc(N * sizeof(int (*)[]));
```

³Relax: espero que podamos entender la sintaxis para declarar la variable vectores al final de la asignatura

Cadenas enlazadas

Cadenas enlazadas

- 💬 ¿Cómo podemos implementar cadenas enlazadas en C?
- 💬 Dibujamos primero.
- 💻 Lo intentamos empezando por aquí

Cadenas enlazadas

💬 ¿Cómo podemos implementar cadenas enlazadas en C?

💬 Dibujamos primero.

📖 Lo intentamos empezando por aquí

```
struct nodo {  
    int dato;  
    struct nodo *siguiente;  
};
```

⚠️ Restricciones sintácticas y nueva sintaxis (->)

- E implementamos cada una de las operaciones típicas de las cadenas enlazadas: **crear vacía**, **primero**, **último**, **es vacía**, **añadir al principio**, **añadir al final**, etc.