Sesión 05: *Arrays* y *Strings*

Programación para Sistemas

Ángel Herranz

2021-2022

Universidad Politécnica de Madrid

Recordatorio *módulos*

generador_lcg.h

```c
#define A 7
#define C 1
#define M 13

extern int generar_aleatorio();
```

generador_lcg.c

```c
#include "generador_lcg.h"
int x = 0;
int generar_aleatorio() {
  int anterior = x;
  x = (A * x + C) % M;
  return anterior;
}
```

lcg2.c

```c
#include <stdio.h>
#include "generador_lcg.h"
int main() {
  int i;
  for (i = 0; i < M; i++) {
    printf(
      "%i -> %i\n",
      i,
      generar_aleatorio());
  }
  return 0;
}
```

Convención (evitará *dobles inclusiones*)

generador_lcg.h

```c
#ifndef GENERADOR_LCG_H
#define GENERADOR_LCG_H

#define A 7
#define C 1
#define M 13

extern int generar_aleatorio();

#endif
```

generador_lcg.c

```c
#include "generador_lcg.h"
int x = 0;
int generar_aleatorio() {
  int anterior = x;
  x = (A * x + C) % M;
  return anterior;
}
```

lcg2.c

```c
#include <stdio.h>
#include "generador_lcg.h"
int main() {
  int i;
  for (i = 0; i < M; i++) {
    printf(
      "%i -> %i\n",
      i,
      generar_aleatorio());
  }
  return 0;
}
```

Convención (evitará *dobles inclusiones*)

generador_lcg.h

```c
#ifndef GENERADOR_LCG_H
#define GENERADOR_LCG_H

#define A 7
#define C 1
#define M 13

extern int generar_aleatorio();

#endif
```

generador_lcg.c

```c
#include "generador_lcg.h"
int x = 0;
int generar_aleatorio() {
  int anterior = x;
  x = (A * x + C) % M;
  return anterior;
}
```

lcg2.c

```c
#include <stdio.h>
#include "generador_lcg.h"
int main() {
  int i;
  for (i = 0; i < M; i++) {
    printf(
      "%i -> %i\n",
      i,
      generar_aleatorio());
  }
  return 0;
}
```

#ifndef GENERADOR_LCG_H: Si no está definida la macro GENERADOR_LCG_H entonces se procesa todo hasta **#endif** (en otro caso no se procesa)

Busca *headers* de la biblioteca estándar como stdio.h o limits.h y mira cómo siguen la convención.

En el capítulo de hoy...

- Vectores (*Arrays*)
- Cadenas de caracteres (*Strings*) = arrays de caracteres

En el capítulo de hoy...

- Vectores (*Arrays*)
- Cadenas de caracteres (*Strings*) = arrays de caracteres

⚠ *Íntima* relación entre punteros y *arrays*

Variables de tipo *array* i (longitud fija)

- Sintaxis i:

$$T\ \ a[N];$$

- Esa definición crea un espacio de memoria contigua para almacenar N elementos de tipo T,

💬 tan grande en bytes como lo que indican N y `sizeof(`T`)`,

Variables de tipo *array* i (longitud fija)

- Sintaxis i:

$$T\ \ a[N];$$

- Esa definición crea un espacio de memoria contigua para almacenar N elementos de tipo T,

💬 tan grande en bytes como lo que indican N y `sizeof(`T`)`,

- elementos accesibles usando la expresión

$$a[i]$$

- donde i deberá estar entre 0 y $N-1$

lcg3.c: modificar el programa lcg2.c

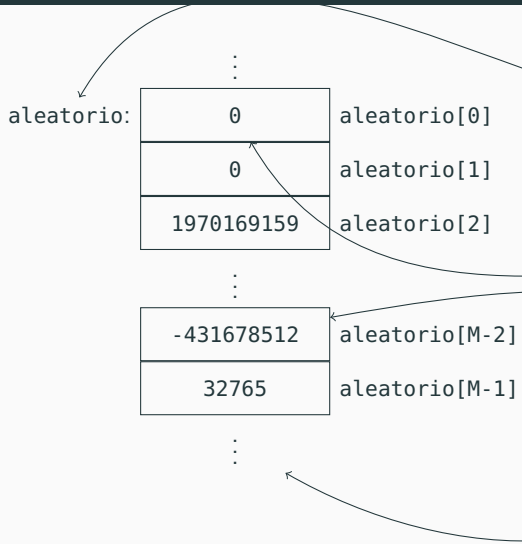
- Almacenar M[1] datos en un array y luego imprimirlos.

```c
#include <stdio.h>
#include "generador_lcg.h"
int main() {
  int i;
  int aleatorio[M];
  for (i = 0; i < M; i++) {
    aleatorio[i] =
      generar_aleatorio());
  }
  for (i = 0; i < M; i++) {
    printf(
      "%i -> %i\n",
      i,
      aleatorio[i]);
  }
  return 0;
}
```

5'

[1] M = 11 en las transparencias

Así están las cosas antes del primer `for`



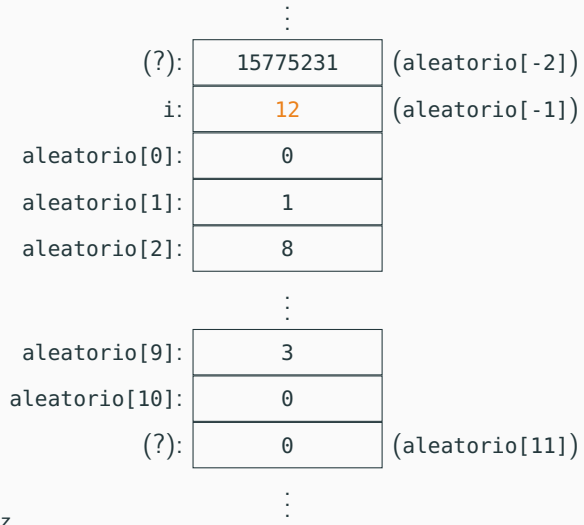
- Aquí empieza el array
- El primer elemento está justo donde empieza el array y *se ~~llama~~* `aleatorio[0]`
- Nada se inicializa en C
- Hay memoria por delante y por detrás

¿Qué pasa si *me salgo* del array?

```
for (i = -M; i <= M; i++) {
  printf(
    "%i -> %i\n",
    i,
    aleatorio[i]);
}
```

¡Explicar!

Al final de la ejecución ¿aleatorio[-1] == 12?



¿Longitud de un array?

- Imprimir la longitud de un array (usemos por ejemplo `aleatorio` asumiendo que no conocemos `M`)

`sizeof`

(tamaño en bytes de cualquier expresión)

[2] Veremos ejemplos

¿Longitud de un array?

- Imprimir la longitud de un array (usemos por ejemplo `aleatorio` asumiendo que no conocemos `M`)

`sizeof`

(tamaño en bytes de cualquier expresión)

1'

Este recurso no es válido en general[2]

[2] Veremos ejemplos

¿Inmutabilidad de las variables array?

- Intentemos estas dos asignaciones:

```
int a[10];
int b[10];
b = a;
```

- ¿Qué nos dice el compilador?

¿Inmutabilidad de las variables array?

- Intentemos estas dos asignaciones:

```
int a[10];
int b[10];
b = a;
```

¿Qué nos dice el compilador?

```
$ make
cc -Wall -g -pedantic -o arrays arrays.c
arrays.c: In function ‘main’:
arrays.c:15:5: error: assignment to expression with array type
   b = a;
     ^
```

Variables de tipo *array* ii (inicializando)

- Sintaxis ii:

$$T\ a[\,] = \{\ e_0,\ e_1,\ \ldots,\ e_{n-1}\ \};$$

  donde la inicialización es obligatoria
- Esa definición crea un espacio de memoria contigua para almacenar n elementos de tipo T,
- tan grande en bytes como lo que indican n y **sizeof**(T)
- elementos accesibles usando la expresión

$$a[i]$$

- donde i deberá estar entre 0 y $n-1$

¿Longitud de un array?

- Escribir un programa
  - Con una variable de tipo array declarada por inicialización con los números de Fibonnaci menores de 100
  - Imprimir todos los elementos
  - Imprimir la longitud

Variables de tipo *array* iii (argumentos)

- Sintaxis iii:

```
tipo_return funcion(tipo arg[]) {
  ...
}
```

- Esa definición no crea un espacio de memoria contigua,
- simplemente se pasa como argumento la dirección de memoria del primer elemento del array
- De nuevo, los elementos son accesibles usando la sintaxis

```
arg[i]
```

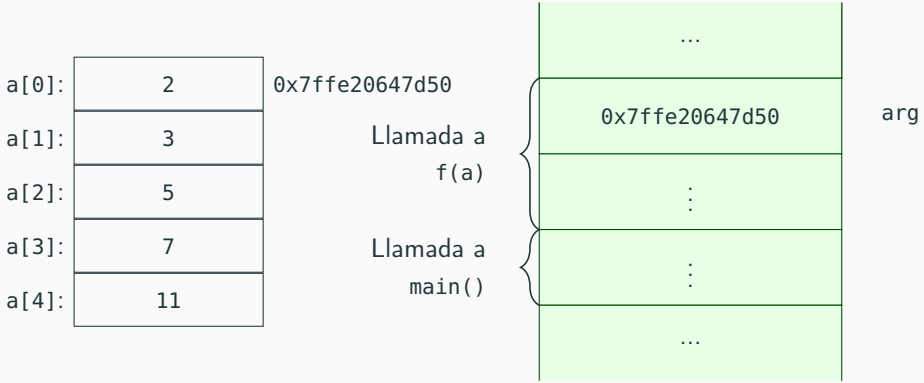
- donde *i* deberá estar entre *0* y la longitud del array - 1

¿Longitud de un array?

- Escribir una función que admita como argumento un array de enteros e imprima su longitud utilizando la técnica ya aprendida 5'
- ¡Explicar!

¿Longitud de un array?

- Escribir una función que admita como argumento un array de enteros e imprima su longitud utilizando la técnica ya aprendida 5'
- ¡Explicar!



Media del generador aleatorio i

- Escribir un programa que imprima la media del generador de números aleatorios (generar `2*M` datos) utilizando una función a la que se le pasa el array `aleatorio`

- ¿Longitud del array?

Compliant Solution

- La solución al problema de no conocer la longitud de un array en C es simple:

Añadir un argumento con la longitud del array

```
#include <stddef.h> /* Para importar size_t */
...
tipo_return funcion(tipo arg[], size_t len) {
  ...
  for (i = 0; i < len; i++) {
    ...arg[i]...
  }
}
```

size_t

- `size_t` es un tipo definido en `stddef.h` (**`#include`** `<stddef.h>`)
- Se usa para longitudes de arrays y para tamaño de datos
- Internamente es un **`unsigned`**, probablemente **`long`**, pero no importa
- Usado en las bibliotecas estándares, por ejemplo:

```
$ man 3 strlen
STRLEN(3)        Linux Programmer's Manual       STRLEN(3)
NAME
       strlen - calculate the length of a string
SYNOPSIS
       #include <string.h>
       size_t strlen(const char *s);
...
```

Media del generador aleatorio ii

- Escribir un programa que imprima la media del generador de números aleatorios (generar `2*M` datos) utilizando una función a la que se le pasa el array `aleatorio` y la longitud del array

Strings

Strings

- C no tiene tipo *String*
- Se usan arrays de caracteres (enteros que caben en 1 byte)

Transcribir el siguiente programa 2'

```
#include <stdio.h>
int main() {
  char s[] = "mundo";
  printf("El string es \"%s\"\n", s);
  printf("La longitud del array s es %lu\n",
         sizeof(s) / sizeof(s[0]));
  return 0;
}
```

¿Longitud del array?

```
El string es "mundo".
La longitud del array s es 6
```

- ¿Otra vez con problemas con la longitud?
- Modifica el programa para que imprima el código ASCII de cada elemento
- ¿Encuentras alguna explicación?

NULL terminated

Convención

las bibliotecas estándares de C asumen que los strings son NULL terminated, es decir, el string termina con el caracter `'\0'` (entero `0`)

(independientemente de la longitud del array)

¿Qué implicaciones tiene dicha convención?

NULL terminated

Convención

las bibliotecas estándares de C asumen que los strings son NULL terminated, es decir, el string termina con el caracter `'\0'` (entero `0`) (independientemente de la longitud del array)

¿Qué implicaciones tiene dicha convención?

- La longitud del string está marcada por la posición del caracter `'\0'`.
- La longitud del array tiene que tener un hueco para el caracter `'\0'`.

El tipo `char *`

- C no tiene tipo *String*, los *strings* son arrays de caracteres
- Hemos usado la sintaxis

```
char s[] = ...;
```

- Pero la sintaxis *de verdad* para declarar *strings* es

```
char *s;
```

El tipo `char *`

- C no tiene tipo *String*, los *strings* son arrays de caracteres
- Hemos usado la sintaxis

```
char s[] = ...;
```

- Pero la sintaxis *de verdad* para declarar *strings* es

```
char *s;
```

`char *` es *oficialmente* el tipo *string* ("puntero a **char**")

y por lo tanto s es un *string*

string.h i

- <string.h> es el módulo (*header*) de la biblioteca estándar de C para el manejo de strings
- Puedes ver sus funciones en el *K&R*: `strlen`, `strcpy`, etc.
- Usa también el manual, por ejemplo:

```
$ man 3 strcpy
STRCPY(3)        Linux Programmer's Manual       STRCPY(3)
NAME
       strcpy, strncpy - copy a string
SYNOPSIS
       #include <string.h>
       char *strcpy(char *dest, const char *src);
       char *strncpy(char *dest, const char *src, size_t n);
DESCRIPTION
       The  strcpy() function copies the string pointed to by
       src, including the terminating  null byte ('\0'),[...]
```

string.h ii

- Modifica el último programa para que imprima la longitud del string utilizando la función `strlen`.
- Modifica el último programa para cambiar el caracter de terminación por otro (por ejemplo `'_'`) y luego pedir a `printf` que imprima el string.

5'

- ¿Qué ocurre? ¿Puedes explicarlo? ¿Qué diferencia hay entre estos strings?

```
char s6[] = "mundo";
char s5[] = {'m', 'u', 'n', 'd', 'o'};
```

- ¿Qué pasa cuando intentas imprimirlos?

Arrays multidiménsionales

- Sintaxis:

$$T \;\; m[N][M];$$

- Esa definición crea un espacio de memoria contigua,
- tan grande como para almacenar $N \times M$ datos del tipo T,
- elementos accesibles usando la expresión

$$m[i][j];$$

- (elemento que ocupa la fila i y la columna j)
- donde i deberá estar entre 0 y $N-1$ y j entre 0 y $M-1$.

Arrays multidiménsionales: array de array

- Otra forma de ver un array multidimensional como

$$T \ \ m[N][M];$$

  es entendiendo que la expresión $m[i]$ es un array de M elementos de tipo T

- ¿Es posible pasarle cada una de las filas de una matriz a la función que calcula la media?
- Hoja de ejercicios