Introducción a la Programación en C
–Elementos Básicos–

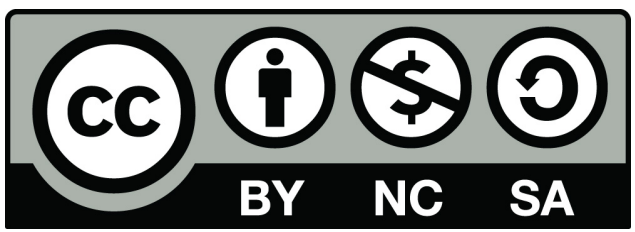
Christopher Expósito-Izquierdo
cexposit@ull.edu.es

Airam Expósito-Márquez
aexposim@ull.edu.es

Israel López-Plata
ilopezpl@ull.edu.es

Belén Melián-Batista
mbmelian@ull.edu.es

José Marcos Moreno-Vega
jmmoreno@ull.edu.es



Contenidos

Palabras Reservadas:

continue	float	signed	auto
default	for	sizeof	typedef
break	static	union	case
do	goto	struct	unsigned
double	if	switch	char
else	register	void	enum
int	return	volatile	const
extern	long	short	while

Palabras Reservadas: Identificadores

- Un identificador es un nombre con el que se hace referencia a una función o al contenido de una variable o de una constante
- Reglas:
  - Formado por secuencia de letras y dígitos
  - El carácter subrayado o underscore (_) es una letra más
  - No pueden contener: espacios en blanco, ni *, /,+,-,:, etc.
  - El primer carácter debe ser una letra o _
  - Se distingue entre mayúsculas y minúsculas
  - Conviene usar nombres de funciones, constantes y variables que indiquen qué representan o su utilidad

Palabras Reservadas: Comentarios

- El objetivo al introducir comentarios en el código fuente es explicar o aclarar cómo está hecho el programa
- Los comentarios son ignorados por completo por el compilador
- /* */ Delimitan los comentarios que pueden ocupar más de una línea
- // Especifica el comienzo de un comentario que acaba con el final de dicha línea
- Los comentarios apropiados dan calidad al código
- Los espacios en blanco (horizontales y verticales) dan estructura al código que mejoran su visualización

Tipos de Datos Básicos:

- Enteros:
  - char
  - signed char
  - unsigned char
  - signed short int (short)
  - signed int (int)
  - signed long int (long)
  - unsigned short int (unsigned short)
  - unsigned int (unsigned)
  - unsigned long int (unsigned long)
- Reales:
  - float
  - double
  - long double

Tipos de Datos Básicos:

Enteros:

Tipos:

- short: 2 bytes (-32766, 32766)
- int: 2 bytes (-32766, 32766) 4 bytes (-2147483647, 2147483647)
- long: 4 bytes (-2147483647, 2147483647)
- unsigned int: 4 bytes (0, 429467295)

Operadores:

- Aritméticos: + - * / % (operador módulo o resto)
- Relacionales: == != $\geq$ $\leq$

Tipos de Datos Básicos:

Reales:

Tipos:

- float: 4 byte (1.2E-38, 3.4E+38) 6 decimales
- double: 8 byte (2.3E-308,1.7E+308) 15 decimales
- long double: 10 byte (3.4E-4932, 1.1E+4932) 19 decimales

Operaciones:

- Operadores aritméticos: + - * /
- Relacionales: >, <, >=, <=, ==, ! = (No deben usarse)

Ejemplos: double x = 1.34, y = 0.0, 5.0 / (x+y)

Caracteres:

- char: 1 byte
  - Normales:
    - Letras (minúsculas) 'a', 'b', 'c', . . .
    - Letras (mayúsculas) 'A', 'B', 'C', . . .
    - Dígitos '0', '1', . . . , '9'
    - Otros '+', . . .
  - Especiales: '\n', '\t', ...
- Basados en el código ASCII: American Standard Code for Information Interchange (código estándar americano para intercambio de información)
- No incluye la ñ ni las vocales con tilde: á, é, í, ó, ú

Caracteres:

Constante carácter:

- Formada por un carácter entre apóstrofes 'e', 'J'
- El valor de la constante es el valor numérico asignado a dicho carácter de acuerdo con el código ASCII En realidad son constantes enteras (0 a 127)
- Ciertos caracteres se representan con las secuencias de escape

Strings:

String

Es una secuencia finita de caracteres consecutivos

- Un *string* o array de caracteres se puede inicializar en el momento de su declaración con una cadena literal. Dicha cadena es una secuencia de caracteres encerrada entre dobles comillas

- Las cadenas literales acaban con el carácter nulo `'\0'` Por tanto, la cadena `"Primero"` tiene longitud 8

```
char curso[8] = "Primero";
```

  y entonces en `curso[0]` está `'P'`, en `curso[1]` está `'r'`,... en `curso[6]` está `'o'` y en `curso[7]` está `'\0'`

Strings:

Caracteres especiales:

Los strings o arrays de caracteres pueden contener unos caracteres especiales (secuencias de escape) que tienen un efecto determinado:

- \n
- \t
- \v (tabulación vertical)
- \b (retrocede un espacio)
- \r (va al comienzo de la línea)
- \a (sonido de alerta del sistema)
- \'
- \"
- \\
- \? (para símbolos especiales)

Tipos Enumerados:

Enumerado

Permite definir un conjunto de datos a través de constantes simbólicas con valores enteros

- Definición:

```
enum nombre{valor1, valor2,..., valorN}
```

- Declaración:

```
enum nombre identificador;
```

Ejemplos:

- `enum diasSemana{lunes, martes, miercoles, jueves, viernes, sabado, domingo};`
- `enum diasSemana dia;`
- `enum boolean{false, true};`
- `...`

Punteros:

- Cada vez que se define una variable, el compilador le asigna el espacio necesario en la memoria del ordenador para poder almacenar la información correspondiente

- El espacio asignado a una variable se encuentra en una determinada dirección de memoria

Punteros:

Puntero

Es un tipo de dato que a diferencia del resto **NO** almacena datos sino direcciones de memoria

- Los punteros almacenan direcciones de memoria Por tanto, referencian a zonas de memoria donde se encuentran los verdaderos datos
- Para poder hacer referencia correctamente a un dato de memoria se debe (i) conocer su dirección inicial de memoria y (ii) su tamaño
- El tamaño en bytes de una variable de tipo puntero viene dado por el tamaño de dirección que utiliza el ordenador

Punteros:

Un puntero se define tal como sigue:

```
tipoBase * identificadorPuntero
```

- `tipoBase`: puede ser cualquier tipo predefinido en C o cualquiera definido por el programador
- `identificadorPuntero`: es el identificador utilizado para la variable de tipo puntero

Ejemplo:

```
long * punteroDato;
```

Los datos que hay almacenados en la dirección guardada en `punteroDato` deben ser de tipo `long`

Punteros:

- Al declarar una variable se reserva una zona de memoria para almacenar un valor concreto perteneciente a un tipo de datos

- Para acceder a una variable almacenada en memoria, el compilador necesita disponer de:
  - Número de bytes que componen la variable, dado por el tipo de datos
  - Dirección de memoria del byte inicial de dicha variable, representada por el nombre de la variable

Punteros:

```
#include <stdio.h>

int main(void) {
        long dato = 0;
        long * punteroDato;
        punteroDato = &dato;
        printf("La direccion de la variable dato es %p\n", &dato);
        printf("El valor de la variable punteroDato es %p\n", punteroDato);
        return 0;
}
```

Con `punteroDato = &dato` se asigna a la variable `punteroDato` la dirección de memoria de la variable `dato`

Punteros:

Operadores para Punteros:

- **Operador dirección (&)**. Permite obtener la dirección de memoria en la que se almacena una variable

  long dato;
  printf(" %lu", &dato);
  printf(" %p", &dato); /* *hexadecimal*/
  int *p;
  p = &k;

- **Operador de indirección (*)**. Permite acceder al contenido de una dirección de memoria. Sólo se puede usar con variables de tipo puntero

```
int * puntero;
*p = 3;
```

Punteros:

```
#include <stdio.h>

int main() {
        int x = 4, y = 7, temp;
        int *px = NULL, *py = NULL;
        printf("x = %d, y = %d ", x, y);
        px = &x;
        py = &y;
        printf("\npx tiene la direccion %p, py tiene la direccion %p ", px, py);
        printf("\nLa direccion de la variable x es %p, la direccion de la variable y es %p", &x , &y) ;
        *px = *px + 10;
        *py = *py + 10;
        printf("\nx es %d y es %d", x, y) ;
        temp = *px ;
        *px = *py;
        *py = temp;
        printf("\npx contiene %d, py contiene %d", *px, *py) ;
        printf("\nx es %d y es %d\n",x,y);
        return 0 ;
}
```

Punteros:

Precauciones:

- Consultar el contenido de una dirección de memoria a través de un puntero (*p) que no ha sido inicializado puede producir errores en tiempo de ejecución realmente graves
- Para evitarlo se debe inicializar todo puntero de la siguiente manera:

```
tipoDato * identificadorPuntero = 0;
```

  o bien

```
tipoDato * identificadorPuntero = NULL;
```

- De esta manera, si se intenta acceder a esta dirección de memoria antes de asignar otro valor se producirá un error fácil de identificar en tiempo de ejecución (violación de segmento)

Punteros:

Punteros y Arrays:

- Cuando un puntero es incrementado en una unidad pasa a apuntar al siguiente elemento del array
- La diferencia entre un array y un puntero es que cuando se define un array estático, `tipoDato array[tamaño]`, el valor de la variable `array` siempre representa la dirección de memoria del primer elemento y dicha dirección no se puede modificar

Punteros:

Un string es el caso particular de array formado por una secuencia de caracteres más el carácter nulo ():

```
char string[númeroElementos];
```

Notas:

- Un array de caracteres sin el carácter nulo al final no es una cadena de caracteres

```
char cadena[5] = "Hola";
```

- Una cadena de longitud 1 no es lo mismo que un solo eslabón

`"a"(cadena de longitud 1)` $\neq$ `'a'(carácter individual)`

Punteros:

Lectura y Escritura de Strings:

- Con una cadena de caracteres se debe usar `%s` para su lectura y su escritura
- `scanf()` deja de buscar elementos cuando encuentra un espacio en blanco
- Debemos usar la función `fgets()`. Así:
  - `char* fgets(char* s, int n, stdin);` Copia caracteres de la entrada estándar (`stdin`) en la cadena `s`, termina cuando ha copiado `n-1` caracteres, cuando encuentra el caracter de final de línea o cuando detecta un error. Esta función devuelve el puntero a la cadena o `NULL` en caso de que encuentre un error
  - `char* strcpy(char* s, const char* ct);` Copia la cadena `ct` en `s` incluyendo el indicador de final de cadena y devuelve la dirección de `s`
  - `int strcmp(const char* cs, const char* ct);` Compara `cs` con `ct` en orden alfabético devolviendo un valor negativo si `cs<ct`, cero si `cs==ct`, y uno positivo si `cs>ct`
  - `size_tstrlen (char* sz);` devuelve la longitud de la cadena que se le pasa como parámetro

Punteros:

Punteros a Estructuras:

Se puede declarar un puntero que apunte a una estructura

```
struct nombreEst *nombrePunt;
struct coordenada {
        float x;
        float y;
};
struct coordenada *p;
```

Para acceder a los elementos de una estructura apuntada varía la sintaxis. En lugar de:

```
*p.x = 1.3;
scanf("%f", &(*p.x));
```

se usa:

```
p->x = 1.3;
scanf("%f", &(p->x));
```

Introducción a la Programación en C
–Elementos Básicos–

Christopher Expósito-Izquierdo
cexposit@ull.edu.es

Airam Expósito-Márquez
aexposim@ull.edu.es

Israel López-Plata
ilopezpl@ull.edu.es

Belén Melián-Batista
mbmelian@ull.edu.es

José Marcos Moreno-Vega
jmmoreno@ull.edu.es