C
Programación II (R-123)
Licenciatura en Ciencias de la Computación

Iker M. Canut

2020

1. Introducción

C es un lenguaje procedural, de estilo estructurado, con una sintaxis estricta, compilado (se genera un archivo ejecutable que depende del O.S. y arquitectura). *#include<stdio.h>* contiene las funciones de entrada y salida que vamos a precisar para imprimir en pantalla. La función main siempre tiene que estar y retorna un entero, 0 representa una terminación correcta. Cada sentencia lleva un ; al final y los bloques son delimitados por { }.

TIPO	RANGO
char	-128 a 127
short	-32768 a 32767
int	-2,147,483,648 a 2,147,483,647
long	-9223372036854775808 a 9223372036854775807
float	3.4E-38 a 3.4E+38
double	1.7E-308 a 1.7E+308

También están los unsigned char, unsigned int, short int, unsigned short int, long int, unsigned long int, unsigned long y long double. Unsigned permite extender el rango superior.

La palabra clave **const** se ysa para declarar una constante, las cuales no se pueden modificar. Toda asignación retorna el valor, luego se puede hacer *int a = b = c = 5.*

Para formatear el texto se usan **%d** (base 10), **%u** (base 10 sin signo), **%o** (base 8 sin signo), **%x** (entero en base 16), **%f** (coma flotante precisión simple), **%lf** (coma flotante precisión doble), **%ld** (long), **%lu** (unsigned long), **%e** (notación científica), **%c** (caracter), **%s** (cadena de caracteres).

Para tener el resultado exacto de una división de enteros, hay que castearlo: *double d = (double) a/b;*

Los operadores logicos son **!** (not), **&&** (and) y || (or). 0 es false y 1 es true.

Para guardar información, se usa el scanf, con & si es un char o un numero. Si es un string va sin &. Se pueden guardar varias variables en un mismo scanf.

Para iterar tenemos el **while** *(condicion)* { *// Sentencias* }
Declarar funciones: *tipo_retorno mi_funcion(tipo1 param1, tipo2 param2)* { *// sentencias* }.
Si se declara con void, no hay valor de retorno. Para declarar nuevas funciones, tienen que estar definidas antes de ser usadas, o al menos el prototipo (tipo de retorno, nombre y los tipos de argumentos).

switch *(variable)* {
case constante1:
    // sentencias1
    break;
case constante2
    // sentencias2
    break;
default:
    // sentencias default
}

La sentencia switch compara el valor de una variable (tipo entero o caracter), al encontrara una coincidencia empieza a ejecutar las sentencias hasta encontrar un break. Si no coincide ninguna se ejecuta el default.

2. Arrays

Al crear un array hay que especificar el tamaño, que no puede modificarse. Todos los elementos de un array son del mismo tipo. No se puede recuperar el tamaño de un array (solo de chars con **strlen()**).

La sintaxis del for es: **for** *(init; cond; incr)* { *// sentencias* } . Un bloque sin cond es un bucle infinito. La unica limitación en el init son las declaraciones, porque las comas tienen otro significado.

3. Memoria

Al ejecutar un programa, el mismo se carga en la memoria física de nuestra computadora. El sistema operativo nos asigna un fragmento de dicha memoria para que podamos usarlo en nuestro programa. Una variable que almacena direcciones de memoria se la llama **puntero**. El operador **&** nos devuelve la dirección de una variable y con ***** accedemos al valor (desreferencia). El valor de un array es la dirección del primer elemento del array.

En C todo argumento es pasado por valor, es decir, al llamar a una función, se hace una copia de los parametros en otro lugar de la memoria. Se manipula ese espacio dentro de la función, y al finalizar se libera. Ergo, no podemos retornar la dirección de una variable creada dentro de una función.

La memoria estática hace referencia a la memoria que no va a ser liberada hasta que finalice el programa. Las funciones, variables y definiciones globales se alojan en esta memoria. La memoria dinámica se denomina así porque se asigna en tiempo de ejecución y, se va liberando cuando ya no se precisa. Para solicitar memoria dinámica se usa **malloc** (se le indica el tamaño del bloque y retorna la dirección de memoria donde inicia ese bloque), que se usa en conjunto con la función **sizeof**. Finalmente, **free** se usa para liberar esa memoria. Para acceder a dicha memoria podemos usar el operador [] o aritmética de punteros. Escribir **ptr[i]** es lo mismo que ***(ptr+i)** y escribir **&ptr[i]** es lo mismo que **ptr+i**.

4. Estructuras

Crear una estructura permite definir un nuevo tipo de dato. Se usa la palabra clave **struct** y para acceder a los elementos se usa el **.**. Ahora, si usamos un puntero para referenciar a una estructura:
struct *Persona* p1;*
p1 = malloc(sizeof(struct Persona))
Para acceder a los campos hay que hacer **(*p1).nombre**, o bien podemos escribir **p1->nombre**.

La palabra clave typedef permite declarar un tipo de dato. Se puede usar para evitar repetir continuamente la palabra struct.

5. Archivos

El prototipo es: *FILE *fopen(const char * filename, const char * mode);*
El modo **"r"** abre un archivo existente para su lectura, **"w"** para escritura (crea si no existe), **"a"** abre en append, **"r+"** abre en lectura y escritura, **"w+"** lectura y escritura (pisando todo si existe), **"a+"** es lectura desde el comienzo, y escritura al final.
Para cerrar usamos *int fclose (FILE * fp);*. Devuelve 0 si se cerró bien.
Para escribir usamos *fputc(int c, FILE * fp)* (que devuelve el entero que corresponde al char o EOF), *int fputs(const char * s, FILE * fp)* (devuelve un negativo si salió todo bien o EOF si falló), o sino *int fprintf(FILE * fp, const char * format, ...)*.
Para leer tenemos *int getc(FILE * fp)* (lee de a un caracter), *char* gets(char *buf, int n, FILE * fp)* (lee n-1 caracteres y agrega un '\0'), o por último *int fscanf(FILE * fp, const char *format, ...)*, que lee hasta encontrar un separador (espacio en blanco, tabulación o el EOF).