Esquema de un programa en C: bloques básicos

```
// Incluimos los ficheros de cabecera que necesitemos.
// Esto es un comentario
#include <stdio.h>
#include <stdlib.h>
```

```
// Si queremos definir alguna constante, podemos hacerlo
#define TAMANYO_BUFFER 128
#define MAX_PROCESOS 10
```

```
// A continuación definimos las variables globales de
// nuestro programa. Estas variables se pueden usar en
// cualquier función del programa.
// a,b, y c son enteros. d es un carácter. procs es un  vector de enteros y buffer un vector de
// caracteres. En C  no hay booleanos, podemos usar el tipo int.
// Un valor igual a 0 es falso, cualquier otra cosa es cierto.
//Tampoco existe el tipo string, se define como una cadena de caracteres, acabada en el carácter '\0'.
int a,b,c;
char d;
int procs[MAX_PROCESOS];
char buffer[TAMANYO_BUFFER];
```

```
// A continuación el código, encapsulado en funciones. Lo correcto es definir primero las variables
// locales y poner el código a continuación
 void funcion1(void)
{
        // esta función no devuelve nada y no recibe
        // parametros
}
int funcion2(int a)
{
        // esta función devuelve un entero y recibe un entero
        return 1;
}
int funcion3(int a, char b, char *c)
{
        // d es una variable local, solo está definida en el contexto de esta función
        int d;
        // esta función devuelve un entero y recibe un entero,
        // un char y un puntero a char
        return 0;
}
// PUNTO ENTRADA DEL PROGRAMA. La función principal tiene un prototipo fijo, se llama main
int main(int argc,char *argv[])
{
        // devuelve un int y recibe dos parámetros
        // argc es el numero de argumentos del programa
        // argv es la lista de argumentos
}
```

Ejemplo de invocación de un programa en C desde línea de comandos

#programa  → Este programa tendrá, en el main, argc=1, argv[0]="programa"

#programa A B C  → → Este programa tendrá, en el main, argc=4, argv[0]="programa", argv[1]="A", argv[2]="B", argv[3]="C"

#programa 1 2 3 4 5  → → Este programa tendrá, en el main, argc=6, argv[0]="programa", argv[1]="1", argv[2]="2", argv[3]="4", argv[4],"5" y argv[5]="6", siempre se recibe como cadena de caracteres. En

Variables

- Definición variables simples

```
Tipo nombre_variable;
```

- Definición vectores

```
Tipo nombre_variable[dimensión_vector];
```

En el caso de la dimensión del vector, es conveniente utilizar o bien constantes o bien números, no variables. La utilización de variables no siempre funciona como esperáis aunque el compilador la acepte.

Utilizaremos básicamente tres tipos de datos: enteros (int), caracteres (char), y punteros (sean variables simples o vectores). Un int se puede utilizar como ***booleano***, sabiendo que si vale 0 es falso y si vale diferente de 0 es cierto. Una cadena de caracteres se puede usar como ***string***, sabiendo que un "string" (que en C no existe como tipo) debe acabar en '\0', sino es simplemente una vector de caracteres. Los parámetros que recibimos en argv los recibimos como ***string (acabados en '\0')***. Un puntero puede ser a cualquier cosa, puede ser un puntero a entero (int *), puntero a char (char *) o puntero "genérico" (void *).

- Punteros

Un puntero es una dirección de memoria, por lo tanto, serán o 32 o 64 bits dependiendo de la arquitectura. Cuando definimos una variable tipo puntero en C, debemos asignarle un valor para poder utilizarla, o estaremos accediendo a una dirección indeterminada. En C, cuando utilizamos el nombre de un vector, el compilador lo traduce por la dirección del primer elemento. En el siguiente ejemplo la función escribe_mesaje recibe la dirección de buffer.

```
void escribe_mensaje(char *msj){
        write(1,msj,strlen(msj));
}
int main(int argc,char *argv[])
{
        char buffer[64];
        strcpy(buffer,"esto es una prueba");
        escribe_mensaje(buffer); // igual que si ponemos escribe_messaje(&buffer[0])
}
```

- Asignación (=)

```
int x=4;
char a='A';
int *p=&x;          // p es un puntero a entero, inicializado con la dirección de x
int v[10]; // los vectores en C van de 0 a (n-1), en este caso de 0..9
v[0]=1;v[1]=2;v[9]=19;
```

- Comparación: iguales (==) para comparar enteros, char's y punteros

```
int x=4;char a='A';
if (x==4){
        // Hacer algo si son iguales
}else if (a=='A'){
        // lo que sea
}
```

- Otras comparaciones: mayor (>), menor (<), mayor igual (>=), menor igual (<=), diferentes (¡=)
- Hay que tener cuidado con las preferencias, ante la duda poned paréntesis

```
// Si quereis hacer una asignación y una comparación en una línea, por ejemplo el equivalente a esto…
int fd;
fd=open("fichero",O_RDONLY);
if (fd<0) escribe_error("error al abrir el fichero\n");
// no se puede hacer asi
if (fd=open("fichero",O_RDONLY)<0) escribe_error("error al abrir el fichero\n");
// Hay que hacerlo asi… con paréntesis, porque en C primero se compara y luego se asigna
if ((fd=open("fichero",O_RDONLY))<0) escribe_error("error al abrir el fichero\n");
```

Strings

El tipo string no existe en C. Se dice que es un string cuando es un vector de caracteres acabado en '\0'.

- El carácter '\n' indica el salto de línea
- Para utilizar estas funciones, hay que consultar que ficheros de cabecera (include) hay que incluir (stdio.h, string.h, etc). Consultad en el man (man strlen, man sprintf, etc)
- Inicialización

```
char buffer[64];                                    // buffer es un vector de de char's propósito general
strcpy(buffer,"Hola, esto es un ejemplo\n"); // strcpy(destino,origen)
```

```
char *buffer="Este mensaje es mas para usos fijos, normalmente constantes\n";
```

- Calcular la longitud: int strlen(char *s)
- Generar un string a partir de varios tipos simples : int sprintf(char *s,char *formato,….)
    - Recibe un número variable de parámetros. El "formato" indica como generar el nuevo string, se substituyen caracteres especiales por la lista de variables que hay al final.(%d = int, %c= char, %s=string)

```
char buffer[64];
sprintf(buffer,"Puedo usarlo para inicializar\n");
sprintf(buffer,"Puedo poner enteros aquí %d aquí %d y aquí %d\n",1,2,3);
sprintf(buffer,"Puedo combinar otras variables, chars %c o otros strings %s\n",'x',"esto se pondra al final");
```

- Comparación: int strcmp(char *s1,char *s2) (devuelve 0 si son iguales)

```
char buffer[64],buffer1[128];
 strcpy(buffer,"Hola");
if (strcmp(buffer,"Hola")==0) {
        strcpy(buffer1,"Son iguales!\n");
        write(1,buffer,strlen(buffer1));
else{
        strcpy(buffer1,"Son diferentes!\n");
        write(1,buffer,strlen(buffer1));
}
```

Convertir de int → string y de string → int

Para convertir tipos de datos, es suficiente con dos funciones sencillas:

- sprintf → podemos usarla para pasar de int a *string*

```
int x=1000;
char buffer[64];
// para escribir un número por pantalla, necesitamos que sea ascii
sprintf(buffer,"%d",x);
write(1,buffer,strlen(buffer));
```

- int atoi(char *s) → convierte un "string" a int.
    - Por ejemplo, un programa que suma dos números recibidos como argumentos

```
int main(int argc,char *argv[])
{		char buffer[64]; // esto es un vector de caracteres, variable local
		int x,y,suma; // Esto son enteros, variable locales
		x=atoi(argv[1]);
		y=atoi(argv[2]);
		suma=x+y;
		sprintf(buffer,"La suma es %d\n",suma);
		write(1,buffer,strlen(buffer));
}
```

Código iterativo: while y for

```
while(condicion){
		// código que se repite
}
for(inicialización ; condición; modificación_por_iteracion)
{
}
// ejemplo
for (i=0;i<100;i++)
{
}
```

Código condicional: if (condicion){ codigo_cierto}else{codigo_falso}

```
if(condición){
		// caso cierto
}
if (condicion){
		// caso cierto
}else{
		// caso falso
}
if (condicion){
}else if (condicion){
}else if (condición){
}else{
}
```

Funciones

```
tipo_de_retorno nombre_funcion(tipo_var1 var1,tipo_var2 var2,…tipo_varn varn)
```

- Se utiliza void cuando no devuelve nada o cuando no tiene ningún parámetro

```
int suma(int a,int b)
{return a+b;}
void escribe_error(char *msj)
{
		write(1,msj,strlen(msj));
		exit(1);
}
```