Anexos del tema 2.

Introducción

En estos apartados vamos a incluir determinados conceptos que, pese a no pertenecer de forma directa a los contenidos del tema 2, es necesario adelantarlos para poder comenzar la parte práctica de la asignatura. La mayoría de estos conceptos serán ampliados en temas posteriores.

Comenzaremos explicando la estructura más básica de un programa en C. Continuaremos hablando de un tipo de datos estructurado: las cadenas de texto. Estudiaremos también algunas operaciones básicas de entrada/salida en C, y finalizaremos comentando las contracciones que permite realizar el lenguaje C en algunas expresiones aritméticas.

Anexo 1: Estructura de un programa elemental en C

Hasta ahora hemos visto cómo introducir algunas instrucciones, en general operaciones de cálculo en C. Un programa no es una mera lista de instrucciones, sigue una determinada estructura que debemos respetar.

El aspecto de un programa básico en C es el siguiente:

```
/*
        Estructura de un programa elemental en C
            Fundamentos de la Programación
                Grado en Ingeniería en
                  Sonido e Imagen
*/


/* Inclusión de las librerías que se van a usar */
#include <stdio.h>
#include <string.h>
// etc…
```

```
/* Definición de constantes */
#define PI 3.141516
#define DIAS_SEMANA 7
// etc...

/* Función principal */
void main(void)
{
    /* Definición de las variables que se van a usar */
    int a, b=5, c=10;
    float mak = 5.4;
    char miVar = 'c', otraVar=5;
    // etc...

    /* Instrucciones del programa */
    a = b*c+11*DIAS_SEMANA;
    mak = mak * 2.5 / PI;
    miVar = miVar – otraVar;
    // etc...

    /* fin del programa */
    return;
}
```

Como vemos, al principio del programa aparecen una serie de comentarios. Los comentarios sirven para documentar el propio texto del programa, para que sea más comprensible tanto para nosotros como para cualquiera que lo tenga que utilizar. No afectan en ningún caso al funcionamiento del programa, se trata de textos simplemente aclarativos.

Existen dos formas de insertar comentarios:

Comentarios que pueden contener una o varias líneas: Se colocan entre los símbolos especiales **/*** (inicio de comentario) y ***/** (fin de comentario). Cualquier cadena colocada entre dichos símbolos será obviada por el compilador. El comentario puede ocupar varias líneas.

Comentarios de una sola línea: Se colocan a la derecha del símbolo especial //. Afectan a toda la línea colocada a la derecha de éste hasta que la línea finalice (retorno de carro).

Las primeras instrucciones significativas del programa son las inclusiones de las librerías que se van a usar. En este sentido, mediante la directiva **#include <librería.h>** especificamos al compilador las librerías estándar que se van a usar a lo largo del programa. Una librería es una colección de funciones de soporte que no se encuentran por defecto en el lenguaje. Existen muchas librerías de uso común, como por ejemplo:

stdio.h (funciones estándar de entrada/salida), **string.h** (funciones de manejo de cadenas) o **math.h** (funciones matemáticas). Una vez incluidas en nuestro programa podremos utilizar todas las funciones asociadas con las mismas.

A continuación aparecen las constantes definidas por el usuario. Una vez declaradas podremos referirnos a ellas a lo largo de todo el programa.

Seguidamente aparece la función principal o **main**. Se trata de una función especial (las funciones serán tratadas en temas posteriores) que se caracteriza por ser la primera función que se ejecuta del programa. Todo programa ha de contener una y solo una función **main**. Toda función (incluida la función main) tiene el siguiente prototipo:

```
Tipo NombreFunción(parámetros)
{
    [Instrucciones]
}
```

Como vemos, es necesario especificar el tipo de datos de salida de la función, así como los parámetros que va a manejar. Por ahora, no utilizaremos ni el tipo ni los parámetros de la función **main** (asignamos el tipo vacío **void** a ambos). En temas posteriores ampliaremos estos conceptos. Lo que sí es importante señalar es que las instrucciones asociadas a una función han de estar enmarcadas entre los símbolos **{** (inicio de bloque) y **}** (fin de bloque). Estos símbolos determinan el ámbito de la función. La instrucción especial **return** finaliza la ejecución de la función, y en el caso de la función principal, también del programa.

Dentro de la función principal distinguimos entre dos partes: la zona de declaración de las variables a utilizar y la zona de las instrucciones del programa. Recordemos que para utilizar una variable es necesario declararla antes.

En algunos compiladores, es necesario que **main** devuelva algún valor de retorno, que normalmente es obviado por el sistema operativo. En este caso, devolveremos cero, lo cual significa que nuestro programa ha finalizado correctamente:

```
int main(void)
{
    //instrucciones del programa
    …
    return(0);
}
```

Anexo 2: Cadenas de caracteres

Las cadenas de caracteres son un tipo de datos estructurados. Se trata de un tipo especial de vectores o arrays para el manejo de textos que poseen sus propias funciones. Aunque veremos más adelante las definiciones completas sobre funciones y arrays, vamos a introducir este tipo debido a su enorme utilización.

Hasta ahora hemos definido el tipo carácter (char), que como vimos puede almacenar cualquier carácter perteneciente al código ASCII. Este tipo, por sí solo, tiene poca utilidad. Lo realmente interesante es poder utilizar grupos de caracteres para almacenar palabras, frases, etc. Para ello, surgen las cadenas de caracteres. Una cadena de caracteres es una lista cuyos elementos básicos son variables de tipo carácter (char) con una determinada longitud. Para definir una variable de tipo cadena de caracteres hacemos:

```
char nombre_cadena[tamaño];
```

Donde el tamaño es un número entero fijo que determina la longitud máxima de la cadena. Adicionalmente se puede especificar el contenido inicial de la cadena de la forma:

```
char nombre_cadena[tamaño] = “contenido inicial”;
```

Para referirnos a un carácter de la cadena, especificaremos el ordinal correspondiente a su posición entre corchetes. Hemos de tener en cuenta que el primer carácter está en la posición 0 y el último el que está en la posición tamaño-1.

Ejemplo: Vamos a crear una cadena llamada Mensaje con el contenido inicial “Hola Mundo.”. Después cambiaremos el tercer elemento de la cadena por la letra ‘K’ y el octavo por la letra ‘M’:

```
char Mensaje[100] = “Hola Mundo.”;
Mensaje[2] = ‘K’;
Mensaje[7] = ‘M’;
```

Después de estas operaciones, Mensaje contendría la cadena “HoKa MuMdo.”. En ningún caso podemos referenciar una posición que esté fuera del tamaño reservado para ésta. En ese caso se produciría un error de desbordamiento.

Como podemos observar, la longitud de la cadena puede ser mucho mayor a su contenido. Para controlar el final de la cadena se coloca al final de ésta un carácter especial, el carácter de fin de cadena **‘\0’**. Al crear una cadena y asignarle contenido directamente, el compilador ya coloca dicho símbolo. Si rellenamos cada posición manualmente, tendríamos que colocarlo nosotros mismos:

```
char Mensaje[100];
Mensaje[0] = ‘H’;  Mensaje[1] = ‘o’;   Mensaje[2] = ‘l’;
Mensaje[3] = ‘a’;  Mensaje[4] = ‘ ’;   Mensaje[5] = ‘M’;
Mensaje[6] = ‘u’;  Mensaje[7] = ‘n’;   Mensaje[8] = ‘d’;
Mensaje[9] = ‘o’;  Mensaje[10] = ‘.’;  Mensaje[11] = ’\0’;
```

Como vemos, a la hora de reservar tamaño para una cadena, tendremos que tener en cuenta que contiene un carácter más. El tamaño mínimo para la cadena anterior sería de 12 unidades, 11 para la cadena que queremos guardar y 1 para el carácter de fin de cadena.

A continuación veremos las funciones fundamentales de trabajo con cadenas. Estas funciones se encuentran dentro de la librería **string.h**, por lo que tendremos que

incluirla en nuestro programa para poder utilizarlas. Es necesario que tengamos en cuenta que cada cadena que usemos tendrá su carácter de fin de cadena. En caso contrario estas funciones provocarán un error (el compilador no sabría que existen):

int strlen(cadena) : Devuelve el número de caracteres que hay antes del símbolo '\0' en la cadena especificada (sin contar éste).

int strcmp(cadena_1, cadena_2) : Devuelve un entero con la comparación de las dos cadenas. Dicho entero valdrá 0 si las dos cadenas son iguales, será >0 si la primera es mayor que la segunda en orden alfabético y <0 si la segunda es mayor en orden alfabético que la primera.

strcpy(cadena_destino, cadena_origen) : Copia el contenido de la cadena_origen sobre la cadena_destino. La longitud de la cadena_destino ha de ser igual o mayor al contenido de la cadena de origen. En caso contrario se produciría un desbordamiento.

strcat(cadena_1, cadena_2) : Concatena (une) ambas cadenas en una sola cadena poniendo cadena_1 al principio seguida de cadena_2 y vuelca el resultado sobre cadena_1. Para que esto sea posible, la longitud reservada de cadena_1 ha de ser mayor o igual a la suma de las longitudes de las cadenas que contienen ambas. En caso contrario se produciría un desbordamiento.

Ejemplo: En el siguiente ejemplo vamos a declarar tres cadenas: cad1, cad2 y cad3 y vamos a combinarlas para obtener la cadena "Hola, prueba inicial.":

```
char cad1[50]="Ho", cad2[50], cad3[50]="inicial.";
int ltotal,comparacion;
strcpy(cad2,"prueba ");
strcat(cad1,"la, ");
strcat(cad1,cad2);
strcat(cad1,cad3);
comparacion = strcmp(cad2,cad3);
ltotal = strlen(cad1);
```

En el ejemplo, hemos declarado las tres cadenas asignando a la primera "Ho" y a la tercera "inicial." y dejando la segunda vacía. A continuación copiamos "prueba " sobre la segunda cadena. En la siguiente línea concatenamos al contenido de la cadena 1, la cadena fija "la, " quedando el contenido de las tres cadenas: "Hola, ", "prueba " e "inicial." respectivamente. A continuación unimos la cadena 1 con la cadena 2, quedando la cadena 1 con el contenido "Hola, prueba ". Seguidamente le añadimos a la misma cadena la cadena 3, quedando de la forma "Hola, prueba inicial.". Finalmente comparamos la cadena 2 con la 3 y almacenamos el resultado en *comparacion* y calculamos la longitud total de la cadena 1 almacenándola en *ltotal*. La comparación será un valor entero inferior a 0 puesto que alfabéticamente el contenido de la cadena 2 es anterior al de la cadena 1. La longitud final será de 21 unidades.

Anexo 3: Salida por pantalla y entrada por teclado

Existen dos funciones fundamentales relacionadas con la entrada/salida de información formateada. Se trata de **printf** , que se usa para enviar texto por pantalla y **scanf** que se usa para introducir datos por el teclado. Estas funciones se encuentran dentro de la librería **stdio.h**, por lo que tendremos que incluirla en nuestro programa para poder utilizarlas. Veamos cada una de ellas:

Salida formateada por pantalla. La función printf.

La función **printf** sigue la siguiente sintaxis:

```
printf(cadena_de_formato,argumento1,argumento2, ...,argumentoN);
```

Donde la cadena de formato contiene la información sobre el formato de salida de los datos. Se trata de una cadena de caracteres donde se indica, mediante el símbolo **%** y un código de tipo, el tipo de los datos y el formato de visualización. Cada tipo se asocia ordenadamente a cada dato especificado en la lista de argumentos. Los posibles códigos de tipo son los siguientes:

Códigos de tipo	
%c	Carácter (char)
%d	Número entero (int)
%f	Flotante simple precisión (float)
%lf	Flotante doble precisión (double)
%s	Cadena de caracteres

Todo lo especificado en la cadena de formato que no sean códigos de tipo serán caracteres que se imprimirán por pantalla directamente. Dichos caracteres pueden ser los normales (letras, números y símbolos) o alguno de los caracteres especiales, como los de tabulación o retorno de carro.

Veamos varios ejemplos:

```
printf(“Hola, esto es una prueba”);
```

Al ejecutar esta instrucción aparecería por pantalla la cadena ‘Hola, esto es una prueba’. Como vemos, puesto que no se visualiza ningún dato, la lista de argumentos está vacía.

```
printf(“\tHola, esto es una prueba\n\tde salida por pantalla”);
```

En este caso hemos utilizado el carácter especial ‘\t’, que como podemos recordar es el tabulador y el carácter especial ‘\n’ que es el retorno de carro. La salida serán dos líneas de la forma:

```
‘    Hola, esto es una prueba’
‘    de salida por pantalla’
```

Vamos a realizar un ejemplo más avanzado para imprimir datos:

```
int var1 = 50, var2 = 100;
char cadena1[100] = “Mi cadena”;
printf(“Salida de datos:\n”);
```

```
printf("\tvar1 = %d, var2 = %d y var3= %s\n",var1,var2,cadena1);
```

Como vemos, en la cadena de formato especificamos que vamos a utilizar tres variables: dos enteras y una cadena de texto y en la lista de argumentos especificamos de qué variables se trata en orden. El resultado sería:

```
'Salida de datos:'
'     var1 = 50, var2 = 100 y var3= Mi cadena'
```

Entrada formateada por teclado. La función scanf.

La función **scanf** nos sirve para introducir datos por teclado. Su sintaxis es la misma a la de printf:

```
scanf(cadena_de_formato,argumento1,argumento2, ...,argumentoN);
```

En este caso no tiene mucho sentido no especificar argumentos puesto que es justamente lo que queremos obtener. Sobre dichos argumentos tenemos que anteponer el símbolo **&** para que el valor obtenido se transfiera a cada variable (excepto en cadenas de caracteres). Los códigos de tipo y la forma de especificar la cadena de formato son los mismos que en printf.

Aunque es posible obtener por teclado varios datos a la vez, esto suele ser bastante complicado, debido a las posibles ambigüedades que pueden producirse en la entrada. Es por ello que en los ejemplos vamos a mostrar cómo introducir datos de uno en uno:

Ejemplo:

```
int var1;  float var2; char var3[255];
printf("Introduzca entero:");
scanf("%d",&var1);
printf("Introduzca flotante:");
scanf("%f",&var2);
printf("Introduzca cadena:");
scanf("%s",var3);
printf("variable1 = %d, variable2 = %f, variable3 =
%s\n",var1,var2,var3);
```

En este fragmento de código declaramos 3 variables: la primera entera, la segunda flotante y la tercera una cadena de caracteres. A continuación pedimos al usuario cada valor introduciendo con printf la pregunta (scanf no imprime) y recogiendo a continuación cada valor con scanf. Finalmente, se imprimen los contenidos de las tres variables con una sola instrucción printf. Como podemos observar en los scanf, para solicitar una variable tipo cadena no se utiliza &, pero en las demás sí.

Hemos de tener cuidado en la construcción de la cadena de formato, porque, por ejemplo la instrucción:

```
scanf("\t%d  ",&var1);
```

especificaría que estamos esperando a que el usuario pulse tabulador, introduzca un entero y a continuación deje dos espacios antes de pulsar el retorno de carro.

Esto tendría sentido solamente en algunas situaciones, como por ejemplo para introducir una fecha formateada:

```
int dia, mes, anyo;
printf("Introduzca fecha dd/mm/aa: ");
scanf("%d/%d/%d",&dia,&mes,&anyo);
```

La función scanf esperará tres enteros separados por el símbolo / antes del retorno de carro.

Anexo 4: Contracción de expresiones aritméticas en C

En el lenguaje C, existen formas de expresar algunas operaciones aritméticas de uso común de una manera más cómoda. Por ejemplo la expresión $a = a + b$, expresa un incremento de b unidades sobre la variable a. Por comodidad, puesto que se trata de una suma sobre la misma variable, el lenguaje permite expresar dicha operación de la forma contraída $a \mathrel{+}= b$.

En este sentido, las siguientes operaciones son equivalentes:

Contracción de expresiones en C	
Var = Var + 1	Var++
Var = Var - 1	Var--
Var = Var + x	Var+=x
Var = Var - x	Var-=x
Var = Var * x	Var*=x
Var = Var / x	Var/=x

Estas expresiones facilitan, en la mayoría de los casos, la comprensión y el diseño de un programa.