El lenguaje de programación C

1. Introducción

El lenguaje C es un lenguaje orientado a la implementación de Sistemas Operativos, concretamente Unix. C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones. Se trata de un lenguaje de tipos de datos estáticos, débilmente tipificado, de medio nivel pero con muchas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos. (Puedes ampliar la información en el siguiente enlace: https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n).

```
/* Ejemplo 1 */

#include <stdio.h>

int main ()
{
        printf (“Hola mundo.\n”);
        return 0;
}
```

Ejemplo de programa en C

2. Tipos de datos primitivos

Los tipos de datos primitivos (predefinidos o simples) de un lenguaje de programación son los tipos de datos que se encuentran disponibles directamente. En C existen los siguientes tipos de datos primitivos:

Dato	Tamaño	Rango
int	2 bytes	-32.768 a 32.768
short	2 bytes	-32.768 a 32.768
long	4 bytes	-2.147.483.648 a 2.147.483.647
unsigned	2 bytes	0 a 65.535
unsigned short	2 bytes	0 a 65.535
unsigned long	4 bytes	0 a 4.294.967.295
char	1 byte	-128 a 127
unsigned char	1 byte	0 a 255

Dato	Tamaño	Rango
float	4 bytes	3.4×10^{-38} a 3.4×10^{38}
double	8 bytes	1.7×10^{-308} a 1.7×10^{308}
long double	10 bytes	$3.4 \times 10^{-4.932}$ a 3.4×10^{4932}

3. Operadores

Los lenguajes de programación poseen diferentes tipos de operadores que son utilizados para construir expresiones por combinación de otras más sencillas En nuestro caso, se pueden distinguir los siguientes tipos de operadores:

- Operadores aritméticos.
- Operadores relacionales.
- Operadores lógicos.
- Operadores de asignación.
- Operadores de acceso.

3.1. Operadores aritméticos

Operador	Operación
+	Suma
-	Resta
*	Mutiplicación
/	División
%	Resto de división
- (monario)	Cambio de signo
++	Incremento
--	Decremento

3.2. Operadores relacionales

Operador	Operación
>= , <=	Mayor o igual, menor o igual
> , <	Mayor, menor
==	Igual
!=	Distinto

3.3. Operadores lógicos

Operador	Operación
>= , <=	Mayor o igual, menor o igual
> , <	Mayor, menor
=	Igual
=	Distinto

3.4. Operadores de asignación

Operador	Operación
=	Asignación

3.5. Precedencia y asociatividad de los operadores

A la hora de evaluar las expresiones, hay que tener en cuenta la precedencia y asociatividad de los operadores. En la tabla siguiente se muestra la precedencia y asociatividad de los diferentes operadores del lenguaje:

Prioridad	Operador	Asociatividad
1	., ()	Izquierda a derecha
2	[]	Derecha a izquierda
3	++, --, - (monario)	Derecha a izquierda
4	*, /,%	Izquierda a derecha
5	+, -	Izquierda a derecha
6	<, >	Izquierda a derecha
7	==, !=	Izquierda a derecha
8	&&	Izquierda a derecha
9	||	Izquierda a derecha
10	=	Derecha a izquierda

Para alterar la precedencia y asociatividad de los operadores se utilizan los paréntesis. En este sentido, los paréntesis son los operadores de mayor precedencia.

4. Comentarios

Un comentario es una secuencia de caracteres encerrada entre los delimitadores "/*" y "*/". Todos los caracteres entre esos delimitadores son ignorados por el compilador. Los

comentarios pueden abarcar varias líneas, pero, no se pueden anidar. Se utilizan para documentar los programas.

Ejemplos:

/* Este es un comentario de varias líneas. Esta es la primera línea del comentario
Esta es la segunda línea del comentario
Esta es la tercera línea del comentario */

/* Este es un comentario de una línea */

// Este es un comentario de varias líneas. Esta es la primera línea del comentario
// Esta es la segunda línea del comentario
// Esta es la tercera línea del comentario

//Este es un comentario de una línea

```
/* Ejemplo 2b */
/* Ejemplo de programa que calcula el producto de dos números */

#include <stdio.h>

int main()
{
        /* Se declaran las variables multiplicando y multiplicador
        y se les asignan los valores 1000 y 2 respectivamente */
        int multiplicando = 1000 , multiplicador = 2 ;
        /* Se calcula el producto y se muestra por pantalla */
        printf("Resultado = %d\n", multiplicando * multiplicador);
        return 0;
}
```

Ejemplo de programa en C

5. Identificadores

Un identificador consiste en una secuencia de caracteres, dígitos o caracteres de subrayado que comienzan por una letra o un carácter de subrayado. Los identificadores se usan para nombrar entidades del programa (constantes, tipos definidos por el usuario, variables, funciones, etc.).

Un identificador en C ha de cumplir las reglas siguientes:

- Puede estar formado por letras, números y símbolos de subrayado _.
- Ha de empezar por una letra o el símbolo de subrayado _.
- Se distingue entre mayúsculas y minúsculas.

El lenguaje C es sensible a las mayúsculas (case sensitive), lo que significa que dos identificadores formados por los mismos caracteres y que difieran únicamente en el uso de mayúsculas o minúsculas se consideran diferentes. Por ejemplo nombre, Nombre y NOMBRE son identificadores diferentes. No existe limitación en cuanto a la longitud de los identificadores.

Nota: Las palabras reservadas son identificadores que tienen un significado especial para el lenguaje y por lo tanto no pueden ser utilizadas para nombrar otras entidades. Entre las palabras reservadas se pueden citar: main, void, return, int, if, else, putw, puts, while, switch, case, break, default, char, struct,etc.

6. Declaración de constantes

Una constante simbólica es la representación nombrada un dato constante; es decir, un dato cuyo valor va a permanecer inalterado a lo largo de toda la ejecución del programa.

Las constantes simbólicas se han de declarar al inicio del programa y la sintaxis para su declaración es la siguiente:

#define nombre valor;

Para cada constante declarada es necesario utilizar esta cláusula, terminando obligatoriamente en ";". Aunque no es obligatorio, por claridad del código, las constantes suelen expresarse en mayúsculas.

Ejemplos:

```
#define PI 3.1416;
#define TRUE 1;
#define FALSE 0;
#define MAXIMO 100;
```

```
/* Ejemplo */
/* Ejemplo de programa que utiliza constantes */

#include <stdio.h>

#define HORAS_DIA 24;

int main()
{

        /* Se calculan los minutos que tiene un día y se muestra por pantalla */
        printf ("Un día tiene %d minutos.\n", 60 * HORAS_DIA);
        return 0;

}
```

Ejemplo de programa en C

7. Declaración de variables

Las variables hay que declararlas antes de ser utilizadas para asignarles un tipo y reservar el espacio necesario para almacenarlas en la memoria.

Para declarar variables se utiliza la siguiente sintaxis, dentro de las áreas de declaración de variables de un programa:

nombre-tipo nombre1 [=valor1], nombre2 [=valor2],…

Donde nombre-tipo es el nombre de un tipo de dato (un tipo primitivo del lenguaje o un tipo definido por el usuario), nombre1, nombre2,... son los identificadores de las variables declaradas y valor1, valor2,... son los valores asignados a esas variables (la asignación de valores es opcional).

Ejemplos:

```
int sumando1, sumando2, resultado;
int a, b, c;
int a = 0, b = 0, c = 0;
float a, b;
float a = 5.0, b = 3.0;
double num1, num2;
```

char c;
char c[20];

```
/* Ejemplo 2b */
/* Ejemplo de programa que calcula el producto de dos números */

#include <stdio.h>

int main()
{
        /* Se declaran las variables multiplicando y multiplicador
        y se les asignan los valores 1000 y 2 respectivamente */
        int multiplicando = 1000 , multiplicador = 2 ;
        /* Se calcula el producto y se muestra por pantalla */
        printf("Resultado = %d\n", multiplicando * multiplicador);
        return 0;
}
```

Ejemplo de programa en C

Las variables pueden ser globales (se declaran en la sección de declaraciones globales y se puede acceder a ellas desde cualquier parte del programa) y locales (se declaran dentro de una función o bloque de sentencias y sólo se puede acceder a ellas dentro de su ámbito de declaración).

8. Sentencias

El cuerpo de un programa o subprograma está formado por sentencias. Las sentencias pueden ser de diferentes tipos:

8.1. Expresiones

Una expresión es una construcción del lenguaje que devuelve un valor de retorno al contexto sintáctico del programa donde se evaluó la expresión. Las expresiones no deben aparecer de forma aislada en el código; es decir, han de estar incluidas como parte de una sentencia allá donde se espere una expresión. Las expresiones se pueden clasificar en:

- Expresiones aritméticas.
- Expresiones lógicas.
- Expresiones de asignación.
- Llamadas a funciones.

8.1.1. Expresiones aritméticas

Las expresiones aritméticas son aquellas cuya evaluación devuelve un valor de tipo numérico al contexto del programa donde se evalúan.

Ejemplo:

```
resultado = sumando_1 + sumando_2;
a = b + c * d;
```

8.1.2. Expresiones lógicas

Las expresiones lógicas son aquellas cuya evaluación devuelve un valor lógico (verdadero o falso) al contexto del programa donde se evalúan.

Ejemplo:

```
a >= 5;
a && b;
```

8.1.3. Expresiones de asignación

Las expresiones de asignación sirven para asignar un valor a una variable, elemento de una matriz o campo de un registro. Para ello se escribe primero una referencia a alguno de estos elementos seguido del operador de asignación "=" y a su derecha una expresión. El compilador deberá comprobar en primer lugar la compatibilidad entre el tipo de la expresión a la derecha del operador de asignación con el de la referencia a la izquierda. La sintaxis de la expresión de asignación es:

```
referencia = expresión
```

Donde referencia es una referencia a una posición de memoria (variable, parámetro, campo de registro o elemento de un vector) y expresión una expresión que le da valor.

Operador	Operación
=	Asignación

Ejemplos:

```
a = 5;
resultado = 3 + 4;
resultado = 3 * a + 5;
a = 6 >= 5
a == b
e = a % b
```

```
/* Ejemplo asignaciones */
/* Ejemplo de programa que calcula el producto de dos números */

#include <stdio.h>

int main()
{
        /* Se declaran las variables a, b y c
        A las variables a y b se les asignan los valores 5 y 3 respectivamente */
        int a = 5 , b = 3;
        /* Se calcula la suma de a + b y se asigna a la variable c*/
        c = a + b + 4;
        /* se muestra el resultado por pantalla */
        printf ("La suma de %d y %d es igual a %d\n", &a, &b, &c);
        return 0;
}
```

Ejemplo de programa en C

Actividades

A1. Escribe un programa que pida al usuario dos números enteros y muestre el resultado de la suma, la resta, el producto y la división por pantalla. El resultado de cada operación debe mostrarse en una línea diferente.

8.2. Sentencias de entrada y salida

Las sentencias de entrada y salida permiten mostrar mensajes por la salida estándar (pantalla) y leer información introducida a través del teclado.

Código C	Ejemplo
printf (mensaje)	printf (“Hola”)

scanf (“%tipoDeDato”,&variable)	scanf (“%d”,&numero)

La sentencia printf se utiliza para mostrar información por la salida estándar. La sentencia scanf se utiliza para leer información desde la entrada estándar.

Ejemplos:

```
printf ("Hola mundo!\n");
printf ("InIntroduzca el valor deseado: ");
printf ("EI resultado de sumar %d y %d es %d", sumando 1 , sumando2, resultado);
printf ("EI resultado de sumar %f y %f es %r", sumando 1 , sumando2, resultado);
scanf("%d",valor);
scanf("%d %d %d",valor1, valor2, valor3);
scanf("%f”, valor);
scanf("%c”, caracter);
scanf("%c”, cadena);
```

```
/* Ejemplo 5 */

#include <stdio.h>

int main()
{

        char nombre[20]; /* se define nombre como una cadena de caracteres */
        printf("\n Como te llamas? ");
        scanf("%s",nombre); /* se asigna a nombre la cadena de caracteres introducida por
el teclado */
        printf("\n Hola %s!",nombre); /* se muestra el saludo */
        return 0;

}
```

Ejemplo de programa en C

Actividades

A1. Escribe un programa que muestre un mensaje de tres líneas por pantalla.
A2. Escribe un programa que pida un número entero y lo muestre por pantalla.

8.2.1. Secuencias de escape

Como se ha visto, las constantes de cadena, que se encierran entre comillas dobles (por ejemplo "hola"). Las constantes de cadena pueden contener cualquier carácter ASCII. El carácter de backslash introduce códigos que representan caracteres de escape, por ejemplo '\n' representa carácter un salto de línea.

Secuencia de escape	Carácter	Significado
\n	NL	Nueva línea
\t	HT	Tab horizontal
\v	VT	Tab vertical
\b	BS	Retroceso
\r	CR	Regreso de carro
\f	FF	Avance de forma
\a	BEL	Señal audible
\\	\	Diagonal inversa
\?	¿	Interrogación
\’	‘	Apóstrofo
\”	“	Comillas
\ooo	Ooo	Número octal
\xhh	hh	Número hexadecimal

8.3. Sentencias de control: sentencias condicionales o de selección.

8.3.1. Sentencia if

Esta sentencia permite alterar el flujo normal de ejecución de un programa en virtud del resultado de la evaluación de una determinada expresión lógica. Sintácticamente esta sentencia puede presentarse de esta forma:

```
if (expresión lógica)
        sentencia o bloque de sentencias 1
[else
        sentencia o bloque de sentencias 2]
```

Los corchetes en este caso indican que la parte else es opcional; es decir, puede aparecer o no. La expresión lógica debe ir siempre entre paréntesis.

Código C	Ejemplo

if (condición) {
sentencias
}
	if (edad>=18) {
	printf (“Soy mayor de edad.”);
	}

if (condición) {
sentencias
}
else {
sentencias
}
	if (edad>=18) {
	printf (“Soy mayor de edad.”);
	}
	else {
	printf (“Soy menor de edad.”);
	}

Ejemplos:

```
if (a>b)
        a=b;

if (a>b) {
        a=b;
}

if (a>b)
        a=b;
else
        b=a;

if (a>b)
        a=b;
else {
        b=a;
        a=2;
        c=b;
}
```

Este tipo de construcciones pueden anidarse con otras construcciones de tipo if-else o con otros tipos de sentencias de control de flujo que estudiaremos a continuación.

8.3.2. Otras sentencias condicionales o de selección

Código C	Ejemplo
if (condición 1) {	if (a>b) {

```
        sentencias
}
else if (condición 2) {
        sentencias
}
…
}
else if (condición i) {
        sentencias
}
…
else {
        sentencias
}
```

```
   printf (“a es mayor que b.”);
}
else if (a=b) {
   printf (“a es igual a b.”);
}
else {
   printf (“a es menor que b.”);
}
```

Código C	Ejemplo

switch (expresión) {
  case exp_1:
  sentencias
  case exp_2:
  sentencias
  …
  case exp_i:
  sentencias
  …
  case default:
  sentencias
}
	switch (opcion) {
	  case '1':
	  printf (“Opción 1.”);
	  break;
	  case '2':
	  printf (“Opción 1.”);
	  break;
	  case '3':
	  printf (“Opción 1.”);
	  break;
	  case default:
	  printf (“Otra opción.”);
	}

```
/* Ejemplo 9 */

#include <stdio.h>

int main() {

        char c;

        printf("\nIntroduzca una opcion (S/R/M/D/F): ");
        printf("\n>>");
        scanf("%s",&c); /* se asigna valor a la variable c */

        while (c != 'F') {
```

```
                switch (c) {
                        case 'S':
                                suma();
                                break;
                        case 'R':
                                resta();
                                break;
                        case 'M':
                                mutiplicacion();
                                break;
                        case 'D':
                                division();
                                break;
                        default:
                                printf("Opcion incorrecta");
                                break;
                }

                printf("\nIntroduzca una opcion (S/R/M/D/F): ");
                printf("\n>>");
                scanf("%s",&c); /* se asigna valor a la variable c */

        }

        return 0;
}
```

Ejemplo de programa en C

Actividades

A1. Escribe un programa que pida la edad al usuario y muestre por pantalla si es mayor o menor de edad.

8.4. Sentencias de control: sentencias repetitivas o bucles

8.4.1. Sentencias while y do-while

Esta sentencia se utiliza para realizar iteraciones sobre un bloque de sentencias alterando así el flujo normal de ejecución del programa. Antes de ejecutar en cada iteración el bloque de sentencias el compilador evalúa una determinada expresión lógica para determinar si debe seguir iterando el bloque o continuar con la siguiente sentencia a la estructura while mientras se cumpla una determinada condición, determinada por una expresión. Su estructura es:

```
while (expresionLogica)
        sentencia o bloque de sentencias
```

Código C	Ejemplo

while (condición) {
sentencias
}
	int i=0;
	while (i<10) {
	printf("Me portaré bien.\n");
	i++;
	}

do {
sentencias
} while (condición);
	int i=0;
	do {
	printf("Me portaré bien.\n");
	i++;
	} while (i<10);

8.4.2. Sentencia for

Su estructura es:

```
for (inicialización; condición; incremento)
        sentencia o bloque de sentencias
```

Código C	Ejemplo

for (exp_1; exp_2; exp_3) {
sentencias
}
	for (int i=0; i<10; i++) {
	printf("Me portaré bien.\n");
	}

```
/* Ejemplo 4a */

#include <stdio.h>
```

```
int main()
{
        /* Repetir 10 veces */
        for (int i=0; i<10; i++) {
                printf("Estaré muy atento en clase.\n");
        }

        return 0;
}
```

Ejemplo de programa en C

Actividades

A1. Escribe un programa que pida números al usuario hasta que se introduzca un cero y muestre por pantalla la suma de esos números.
A2. Escribe un programa que muestre por pantalla los diez primeros números naturales.

9. Funciones

Las funciones se utilizan (se declaran) para organizar modularmente el código. Una función es una secuencia de instrucciones encapsuladas bajo un nombre con un conjunto ordenado de parámetros tipificados (opcionalmente ninguno) y un valor de retorno también tipificado. Para llamar a una función, ésta debe haber sido declarada con anterioridad en el programa. La definición se hace indicando el tipo de retorno, el nombre de la función y después, entre paréntesis, los argumentos. Por último se añaden las sentencias. Debe existir una sentencia return que devuelva el valor de retorno de la función.

```
/* Ejemplo de función que devuelve la suma de dos números que se pasan como parámetros*/
int suma(int sumando_1, sumando_2)
{
        int resultado ;
        resultado = sumando_1 +  sumando_2;
        return resultado.
```

```
}
```

Ejemplo de programa en C

Ejemplo sencillo del uso de funciones

```
/* Ejemplo 06a */

#include <stdio.h>

int main() {

        printf("Buenos dias.\n");

        printf("\n”);
        printf("Un programa puede descomponerse en funciones.\n");
        printf("Cada función se encarga de resolver una parte del problema.\n");
        printf("Esto permite construir una aplicación de forma modular.\n");
        printf("\n”);

        printf("Adios. \n");

        return 0;
}
```

Ejemplo de programa en C

```
/* Ejemplo 06b */

#include <stdio.h>

void saludo() {
        printf("Buenos dias.\n");
}

void despedida() {
        printf("Adios. \n");
}

void mensaje() {
        printf("\n”);
        printf("Un programa puede descomponerse en funciones.\n");
        printf("Cada función se encarga de resolver una parte del problema.\n");
```

```
        printf("Esto permite construir una aplicación de forma modular.\n");
        printf("\n”);
}

int main() {
        saludo(); /* Llamada a la función saludo */
        mensaje(): /* Llamada a la función mensaje */
        despedida(); /* Llamada a la función despedida */
        return 0;
}
```

Ejemplo de programa en C

9.1. La función main

Todo programa en C debe obligatoriamente declarar una función principal llamada main. Esta función constituye el punto de arranque del programa.

```
/* Ejemplo 7: Programa que calcula la suma de dos números que se introducen por teclado
*/

#include <stdio.h>

int main() {

        float a, b, c; /* se declaran tres variables a, b y c como decimales */

        printf("\n Primer sumando? ");
        scanf("%f”,&a); /* se asigna valor a la variable a */

        printf("\n Segundo sumando? ");
        scanf("%f”,&b); /* se asigna valor a la variable b */

        c = a + b;

        printf("\n la suma de %f y %f es %f”, a, b, c); /* se muestra el resultado */

        return 0;
}
```

Ejemplo de programa en C

Actividades

Escribe una función que devuelva el mayor de dos números que se pasan como parámetros.
Escribe una función que muestre un mensaje un número determinado de veces que se pasa como parámetro.

10. Ejemplo: Programa completo para el cálculo de las raíces de una ecuación de segundo grado

```
/* Ejemplo 8: Programa para el cálculo de las raíces de una ecuación de segundo grado */

#include <stdio.h>
#include <math.h>

int main() {

	double a, b, c, x1 , x2, discriminante;

	printf("\n Valores de los coeficientes a, b y c? ");
	scanf("%lf %lf %lf",&a, &b, &c); /* se asigna valor a los coeficientes a, b y c */

	if (a==0)
		printf("\n Error: Es una ecuacion de primer grado");
	else {
		discriminante=b*b - 4*a*c;
		if (discriminante>0) {
		x1=(-b + sqrt(discriminante)) / (2.0 * a);
		x2=(-b - sqrt(discriminante)) / (2.0 * a);
		printf("\n Primera raiz: %lf", x1);
		printf("\n Segunda raiz: %lf", x2);
	}
	else if (discriminante==0) {
		x1 =(-b) / (2.0 * a);
		printf("\n Raiz doble: %lf", x1);
	}
	else
		printf("\n La ecuacion no tiene soluciones reales.");

	return 0;
```

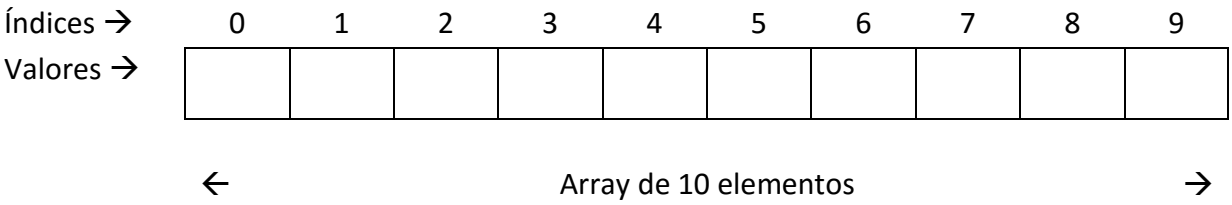
} **Ejemplo de programa en C**

11. Tipos de datos estructurados

Se llama tipo primitivo a los tipos de datos originales de un lenguaje de programación, esto es, aquellos que nos proporciona el lenguaje y con los que se pueden construir tipos de datos abstractos y estructuras de datos. Los tipos de datos primitivos definen como se almacena la información en la memoria del ordenador y los diferentes procesos que se pueden realizar sobre ella. Son abstracciones útiles que facilitan el manejo de los datos, pero en la mayoría de los casos la información que se debe procesar está estructurada de alguna manera. Por eso la mayoría de los lenguajes de programación proporcionan uno o varios mecanismos para combinar los tipos primitivos en estructuras más complejas llamadas tipos de datos estructurados. Los tipos de datos estructurados o tipos compuestos son agrupaciones de otros tipos de datos. Los tipos de datos estructurados más habituales son los arreglos, las cadenas de caracteres (String), los registros y uniones y los archivos o ficheros.

11.1. Arrays

Un array (o arreglo) es una estructura de datos que almacena una cantidad fija de elementos del mismo tipo, a los cuales se puede acceder por medio de uno (unidimensional) o varios (multidimensional) índices índice que indican su posición dentro de la estructura.



La sintaxis a emplear para declarar un array en C es la siguiente:

tipoDeElementosDelArray nombreDelArray [numeroDeElementos];

A la hora de declarar un array en C es necesario indicar el número de elementos del array, ya que el espacio que habrá que reservar en memoria será igual al número máximo de elementos del array por el tamaño del tipo de dato que almacena.

```
int miArrayDeEnteros [10 ];
float miArrayDeDecimales [15 ];
char miArrayDeCaracteres[30];
```

Una vez creado el array (se ha reservado en memoria el espacio necesario para el array), lo siguiente sería inicializar el array. Si no se inicializa explícitamente el array no se puede estar seguro del valor que contienen los elementos del mismo.

Para inicializar (por ejemplo a 0) los elementos de un array:

```
int [ ] miArrayDeEnteros = new int [10];
for (int i = 0; i < 10; i++) {
        miArrayDeEnteros [i] = 0;
}
```

Nota: el atributo length almacena el número de elementos del array. A la hora de trabajar con arrays hay que tener en cuenta que el subíndice no puede ser inferior a 0 ni superior a al valor (length – 1).

La declaración, creación e inicialización de un array son operaciones distintas que, como hemos visto pueden realizarse de manera independiente. No obstante, también se puede crear e inicializar el array en el momento de la declaración. Para ello se añade un signo igual y una lista de valores encerrados entre llaves y separados por comas.

```
int [ ] miArrayDeEnteros = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

El acceso a los diferentes elementos del array se hace mediante su nombre (o identificador) seguido de la posición (o índice) del elemento entre corchetes.

```
miArrayDeEnteros [0]=15;
miArrayDeEnteros [1]=2;
miArrayDeEnteros [2]=5;

int a;
a = miArrayDeEnteros [0];
int b = miArrayDeEnteros [1];
```

Los elementos de un array se comportan como cualquier variable de su tipo y pueden usarse en expresiones.

```
int suma = miArrayDeEnteros [3] + miArrayDeEnteros [5];
```

```
/* Ejemplo de uso de arrays */

#include <stdio.h>

int main ()
{

        int miArray[10];
        int i;
        /* Inicializar el array */
        for (i=0; i<10; i++) {
                miArray [i] = i;
        }

        /* Imprimir el array */
        printf ("Los valores de mi array son:\n");
        for (i=0; i<10; i++) {
                printf (“%5d”, miArray [i]);
        }

        /* Imprimir el array en orden inverso*/
        printf ("Los valores de mi array son:\n");
        for (i=9; i>=0; i--) {
                printf (“%5d”, miArray [i]);
        }

        return 0;

}
```

Ejemplo de programa en C

11.2 Arrays multidimensionales

Los arrays multidimensionales son arrays que necesitan más de un subíndice para identificar los elementos.

```
int miArrayDeEnteros [5][10];
int miArrayDeDecimales [4][3];
```

El acceso a los diferentes elementos del array se hace mediante su nombre (o identificador) seguido de los índices que indican la posición del elemento entre corchetes.

```
miArrayDeEnteros [0][0]=15;
miArrayDeEnteros [0][1]=2;
miArrayDeEnteros [2][5]=5;

int a;
a = miArrayDeEnteros [0][0];
int b = miArrayDeEnteros [0][1];

int suma = miArrayDeEnteros [3][4] + miArrayDeEnteros [3][5];
```

Para inicializar (por ejemplo a 0) los elementos de un array:

```
int [ ] miArrayDeEnteros = new int [5][10];
for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 10; j++) {
                miArrayDeEnteros [i][j] = 0;
        }
}
```

11.3. Registros o estructuras

Un registro o estructura es un tipo de datos heterogéneo compuesto por un número fijo de componentes denominados campos a los que se accede mediante un selector de campo (expresión formada por el nombre y un operador de selección como por ejemplo: “.”).

La forma de definir una estructura en C es haciendo uso de la palabra clave struct. El formato básico para la declaracion de una estructura en C es el siguiente:

```
struct mystruct
{
        int campo_1;
        double campo_2;
        char campo_3 [30];
```

} variable;

donde "variable" es una instancia de "mystruct". En realidad, no es necesario ponerla ahí. Se podría omitir de la declaración de "mystruct" y más tarde declararla de la siguiente manera:

struct mystruct variable;

Ejemplos:

struct Persona { char nombre[20]; char primerApellido[20]; char segundoApellido[20]; long dni; int edad; } persona;	struct Persona { char nombre[20]; char primerApellido[20]; char segundoApellido[20]; long dni; int edad; } ; struct Persona persona;
Ejemplo 1	**Ejemplo 2**

struct Punto {
int x, y;
} puntoInicial, puntoFinal;
	struct Punto {
	int x, y;
	};
	
	struct Punto puntoInicial, puntoFinal;

Ejemplo 3	**Ejemplo 4**

Una práctica muy común es asignarle un alias o sinónimo al nombre de la estructura, para evitar el tener que poner "struct mystruct" cada vez. En C se puede hacer esto usando la palabra clave typedef, lo que crea un alias a un tipo:

typedef struct
{
...
} Mystruct;

La estructura misma no tiene nombre (por la ausencia de nombre en la primera línea), pero tiene de alias "Mystruct". Entonces se puede usar así:

Mystruct variable;

Nota: Aunque no es obligatorio, una convención, y una buena costumbre es usar mayúscula en la primera letra de un sinónimo de tipo. De todos modos lo importante es darle algún identificador para poder hacer referencia a la estructura.

```
/* Ejemplo de programa con estructuras */

/* Este programa calcula la distancia entre dos puntos */

#include <stdio.h>
#include <math.h>

/* Se defina una estructura para los puntos */
struct Punto {
        int x, y;
};

int main(void)
{

        struct Punto puntoInicial, puntoFinal; /* Se declaran dos puntos */
        float distancia=0;

        /* Se asignan las coordenadas correspondientes a cada punto */
        puntoInicial.x = 0;
        puntoInicial.y = 0;
        puntoFinal.x = 0;
        puntoFinal.y = 5;

        /* Se calcula la distancia entre los dos puntos */
        distancia = sqrt((puntoInicial.x - puntoFinal.x) * (puntoInicial.x - puntoFinal.x) +
(puntoInicial.xy - puntoFinal.y) * (puntoInicial.y - puntoFinal.y));

        distancia = sqrt(pow(puntoInicial.x - puntoFinal.x),2) + pow(puntoInicial.y -
        puntoFinal.y),2));

        /* Se muestra el resultado por pantalla */
        printf("\n Punto inicial: %d , %d", puntoInicial.x, puntoInicial.y);
        printf("\n Punto inicial: %d , %d", puntoFinal.x, puntoFinal.y);
        printf("\n Distancia: %.2f", distancia);
```

```
        /* el .2 que está entre %f sirve para mostrar únicamente 2 decimales después del
        punto*/

        return 0;
}
```

Ejemplo de programa en C

Estructuras Anidadas

Nota: Una estructura puede estar dentro de otra estructura a esto se le conoce como anidamiento o estructuras anidadas. Ya que se trabajan con datos en estructuras si definimos un tipo de dato en una estructura y necesitamos definir ese dato dentro de otra estructura solamente se llama el dato de la estructura anterior.

11.4. Cadenas de caracteres (Strings)

En programación, una cadena de caracteres (String) es una secuencia ordenada (de longitud arbitraria, aunque finita) de elementos que pertenecen a un cierto lenguaje formal o alfabeto análogas a una fórmula o a una oración.

En general, una cadena de caracteres es una sucesión de caracteres (letras, números u otros signos o símbolos).

Algunas de las operaciones comunes con cadenas son las siguientes:

- **Asignación**: Consiste en asignar una cadena a otra.
- **Concatenación**: Consiste en unir dos cadenas o más (o una cadena con un carácter) para formar una cadena de mayor tamaño.
- **Búsqueda**: Consiste en localizar dentro de una cadena una subcadena más pequeña o un carácter.
- **Extracción**: Se trata de sacar fuera de una cadena una porción de la misma según su posición dentro de ella.
- **Comparación**: Se utiliza para comparar dos cadenas.
- **Longitud**: Número de caracteres de una cadena

Nota: Las cadenas de caracteres se escriben entre comillas dobles ("esto es una cadena de caracteres"), mientras que un carácter de esa cadena (char) se escribe entre comillas simples

('p'). Generalmente para acceder a un carácter en una posición determinada se suele usar la forma variable [posición] como cuando se accede a un vector.

Cadenas de caracteres en C

En C, las cadenas de caracteres (también llamadas cadenas o strings) son un tipo particular de vectores, concretamente, tal y como su nombre indica, son vectores de char, con la particularidad que tienen una marca (el caracter '\0') de final de cadena. Además, el lenguaje nos permite escribirlas como texto dentro de comillas dobles. Veamos unos ejemplos de su declaración:

```
char cadena[]="Hola";
char otra_cadena[]={'H','o','l','a','\0'}; // Igual al anterior
char vector[]={'H','o','l','a'}; /* Un vector de 4 elementos, con los elementos 'H','o','l' y 'a' */
char cadena[1024]="Una cadena en C";
char cadena[]=""; /* Esto sería una cadena vacía */
```

Nota: Cómo vimos anteriormente al declarar un vector se define la cantidad de elementos que puede contener, en el caso de las cadenas se debe tener en cuenta el espacio adicional necesario para el \0.

La biblioteca estándar de C a través del encabezado string.h proporciona una serie de funciones para trabajar con cadenas. Entre las funciones que provee la biblioteca estándar de C, las más importantes son:

- largo = strlen(cadena) // Para obtener el largo de una cadena
- strcpy(destino, origen) // Copia el contenido de origen en destino // destino debe ser lo suficientemente grande
- strcat(destino, origen) // Agrega el contenido de origen al final de destino // destino debe ser lo suficientemente grande
- resultado = strcmp(cadena1, cadena2) // Compara dos cadenas // devuelve un valor menor, igual o mayor que 0 según si cadena1 es menor, // igual o mayor que cadena2, respectivamente.
- posicion = strchr(cadena, caracter) // Devuelve la posición en memoria de la primera // aparición de caracter dentro de cadena
- posicion = strstr(cadena,subcadena) // Devuelve la posición en memoria de la primera // aparición de subcadena dentro de cadena

```
/* Ejemplo 5 */
```

```
#include <stdio.h>

int main()
{
        char nombre[20]; /* se define nombre como una cadena de caracteres */
        printf("\n Como te llamas? ");
        scanf("%s",nombre); /* se asigna a nombre la cadena de caracteres introducida por
el teclado */
        printf("\n Hola %s!",nombre); /* se muestra el saludo */

        return 0;
}
```

Ejemplo de programa en C

11.5. Ficheros

Un fichero es un conjunto de información relacionada, grabada en el sistema de almacenamiento secundario y a la que se hace referencia mediante un nombre. Los ficheros permiten almacenar datos en memoria secundaria para utilizarlos en el futuro.

En C existen dos tipos de ficheros: ficheros binarios (formados por secuencias de bytes) y ficheros de texto (formados por una secuencia de caracteres subdivida en registros de longitud variable llamados líneas).

Las operaciones típicas sobre ficheros son crear, borrar, abrir, cerrar, leer y escribir. Generalmente, al trabajar con ficheros, será necesario realizar un subconjunto de las operaciones anteriores. En particular, siempre que se trabaje con un fichero se deberían realizar las siguientes operaciones:

1. **Crear**, o asignar, un nombre lógico al fichero físico.
2. **Abrir** el fichero.
3. **Operar** sobre el fichero (lectura/escritura, inserción/borrado, etc.).
4. **Cerrar** el fichero.

Nota: Un fichero puede abrirse de diferentes maneras. La apertura del fichero en modo lectura, escritura, lectura/escritura dependerá del tipo de operaciones que vayan a realizarse con los datos almacenados en el fichero.

Dentro de la programación, una parte importante son las operaciones relacionadas con la entrada y salida (E/S) de datos. El estándar de C contiene varias funciones para la edición de

ficheros, éstas están definidas en la cabecera stdio.h y por lo general empiezan con la letra f, haciendo referencia a file. Adicionalmente se agrega un tipo FILE, el cual se usará como apuntador a la información del fichero. La secuencia que usaremos para realizar operaciones será la siguiente:

1. Crear un apuntador del tipo FILE *
2. Abrir el archivo utilizando la función fopen y asignándole el resultado de la llamada a nuestro apuntador.
3. Hacer las diversas operaciones (lectura, escritura, etc).
4. Cerrar el archivo utilizando la función fclose.

12. Actividades

Actividad 1

El alumno creará una aplicación que muestre por pantalla los datos personales del alumno (nombre y apellidos, lugar donde vive, centro donde estudia, etc.) y un saludo.

Actividad 2

El alumno creará una aplicación que calcule el máximo de dos números que el programa pedirá al usuario. El funcionamiento de la aplicación será el siguiente:

1. Aparecerá un mensaje de bienvenida.
2. Pedirá el primer número.
3. Pedirá el segundo número.
4. Calculará el máximo de los dos números.
5. Devolverá el resultado del cálculo con un mensaje de despedida.

Importante: Se valorará el aspecto estético de la aplicación y la interacción con el usuario.

Actividad 3

El alumno creará una aplicación que simule una calculadora. El programa calculará la suma, la diferencia, el producto y la división de dos números que el programa pedirá al usuario. El funcionamiento de la aplicación será el siguiente:

1. Aparecerá un mensaje de bienvenida.
2. Aparecerá un menú de opciones (suma, diferencia, producto, división y salir).
3. Si se selecciona una operación:

3.1. Pedirá el primer número.
3.2. Pedirá el segundo número.
3.3. Calculará el resultado de la operación.
3.4. Devolverá el resultado de la operación y pedirá al usuario si volver al menú o salir de la aplicación.
3.4.1. Si se selecciona volver al menú, se limpiará la pantalla y volverá a aparecer el menú de la aplicación.
3.4.2. Si se selecciona la opción Salir, el programa se cerrará con un mensaje de despedida.
4. Si se selecciona la opción Salir:
4.1. El programa se cerrará con un mensaje de despedida.

Importante: Se valorará el aspecto estético de la aplicación y la interacción con el usuario.