Introducción a la programación en Turbo C ++

Clase 1: Introducción:

El lenguaje C fue inventado por Dennis Ritchie en 1972 cuando trabajaba, junto con Ken Thompson, en el diseño del sistema operativo UNIX.

Compiladores del lenguaje C:

- ***Borland***: C++: Ejecutar: ***Bcc32*** ***Turbo C++:*** Viene con IDE integrado: Ejecutar ***Tcc*** Vers obsoleta gratuita en: Embarcadero.com
- G++ para Linux: Ejecutar: ***gcc***
- ***MS Visual C ++***: Crear el proyecto (File – New – Project - Win32 Console aplication) Crear New Works pace y luego crear C++ Source file. Microsoft Visual C++ 6: New: C++ Source code
- ***Dev C++:*** Compilador de software libre con IDE de www.bloodshed.net: File - New Source File La función main debe ser int.

Características:

Es un lenguaje de propósito genera, de medio nivel. Permite programar a alto nivel y a bajo nivel.
Es un lenguaje portátil. Los programas escritos en C son fácilmente transportables a otros sistemas.
Pasos para crear y ejecutar un programa en C:

1º. Escribirlo en un editor: Cualquier editor que genere ficheros de texto: serán los ficheros fuentes.
2º. Compilarlo en un compilador: El compilador produce ficheros objetos a partir de los ficheros fuentes. Los ficheros objetos contienen código máquina.
3º. Enlazarlo("Linkar"): El enlazador produce un fichero ejecutable a partir de los ficheros objetos.
4º. Ejecutarlo.

Clase 2: Ejercicio introducción al lenguaje C

```
// --PRIMER PROGRAMA EN C --
 #include <stdio.h>

 main ()
 {
   printf ("Hola mundo");
 }
```

```
//-SEGUNDO PROGRAMA-
#include <stdio.h>
#include <conio.h>

main()
{
  clscr() //en libreria conio - limpia pantalla
  printf("hola mundo");
  getch(); //en libreria stdio - pausa
}
```

Getchar

- getchar () es una función que espera la pulsación de la tecla return. se encuentra en la librería stdio.h, por lo tanto, siempre que la utilicemos es necesario incluir la línea: #include <stdio.h>

Los programas C están compuestos de unidades de programa llamadas funciones, las cuales son los módulos básicos del programa.
Un programa en C contiene al menos una función, la función ***main***

```
//---- TERCER PROGRAMA C ----
#include <stdio.h>                    // Son como los diccionarios que va a emplear

main ()
{
  printf ("_Tercer programa en C\n");
  printf ("Pulsa la tecla RETURN para terminar.");
  getchar ();
}
```

Secuencias de escape:

- Los caracteres empezados por \ se les llaman secuencias de escape. Representan caracteres no imprimibles. Hemos visto \n para salto de línea.
- Otras secuencias de escape:
  \r para el retorno de carro, \t para el tabulador, \b para retroceso, \" para la comilla, \' para el apóstrofe y \\ para la barra diagonal invertida.


```
#include <stdio.h>
main ( )    /* Tercer ejemplo */
{
  int horas, minutos;       //se declaran las variables
  horas = 3;                 //se asignan las variables
  minutos = 60 * horas;
  printf ("Hay %d minutos en %d horas.", minutos, horas);
  getchar ();
}
```

int horas, minutos; -> es una sentencia de declaración y las variables son de tipo entero (integer)

/*Comentarios: todo aquello que vaya entre un /* y un */ es ignorado En TC++ también se usa: //

Clase 3: Variables:

int – Valor número entero.
flotante - valor de punto flotante, con una parte fraccional.
doble - un valor de punto flotante de doble precisión.
char - un solo carácter.
vacío - Tipo de valor para fines especiales.

Declaración: La declaración de variables deben ir al principio de la función,
signación: Una vez declarada la variable ya puede usarla almacenando algo en ella.

El símbolo % avisa a la función printf que se va a imprimir una variable en esta posición;
la letra **d** informa que la variable a imprimir es entera (digit). /*

Clase 4: Tipos de datos

Existen cinco tipos de datos básicos en C:

Tipo	Descripción	Long.en bytes	Rango
char	carácter	1	0 a 255
int	entero	2	-32768 a 32767
float	coma flotante	4	aprox. 6 dígitos de precisión
double	coma flotante doble precisión	8	aprox. 12 dígitos de precisión
void	sin valor	0	sin valor

Secuencias de escape:

Código	Significado
\b	retroceso
\f	salto de página
\n	***nueva línea***
\r	retorno de carro
\t	tabulación horizontal
\"	comillas (")
\'	apóstrofo (')
\0	carácter nulo
\\	barra invertida (\)
\v	tabulación vertical
\a	alerta (bell, campanilla)
\ddd	constante octal (ddd son tres dígitos como máximo)
\xddd	constante hexadecimal (ddd son tres dígitos como máximo)

Tipo entero:

Es un número sin parte fraccionaria.
Las constantes enteras se pueden escribir de uno de los tres modos siguientes:

- Decimal: escribiendo el número sin empezar por 0 (excepción sel propio 0). Ej: 1, 0, -2.
- Hexadecimal: empezando el número por 0x. Ejemplos: 0xE, 0x1d, 0x8.
- Octal: empezando el número por 0. Ejemplos: 02, 010.

Tipos float y double

Tienen parte real y parte fraccionaria (num. decimal). Los dos tipos son casi iguales pero
El tipo double tiene doble precisión que el tipo float.
La sintaxis correcta de las constantes de estos dos tipos es:

[signo] [dígitos] [.] [dígitos] [exponente [signo] dígitos]

donde:

- signo → es + o -;
- dígitos → es una secuencia de dígitos;
- **.** → es el punto decimal;
- exponente → es E o e.

Los elementos que están entre [] son opcionales, pero el número no puede empezar por e o E, ya que el compilador lo interpretaría en este caso como un identificador y no como un número.
Algunos ejemplos de constantes de este tipo: 1.0e9, -3E-8, -10.1.

Tipo void

Significa sin valor, sin tipo.
Uno de los usos de void se puede observar al comparar estos dos programas:

```
#include <stdio.h>
main ()
{
  printf ("Versión 1.");
  getchar ();
}
```

```
#include <stdio.h>
void main (void)
{
printf ("Versión 2.");
getchar ();
}
```

Al poner ***void*** entre los paréntesis de la definición de una función, se define a ésta como función que no tiene argumentos. Si se pone delante indica que la función no retorna nada (procedure).

PROGRAMA EJEMPLO

```
#include <stdio.h>

void main (void)
{
  int i = 1;
  char c = 'c';
  float f = 1.0;
  double d = 1e-1;
  printf ("  i = %d\n  c = %c\n  f = %f\n  d = %lf\n", i, c, f, d);
  getchar ();
}
```

Comprueba que la salida de este programa es:

```
i = 1 (entero)
c = c (chart)
f = 1.000000 (flotante)
d = 0.100000 (doble prec.)
```

Clase 5: Clase 6: MODIFICADORES DE TIPO

Un modificador de tipo se usa para alterar el significado del tipo base para que se ajuste más precisamente a las necesidades de cada momento.

Modificador	Descripción	Tipos a los se les puede aplicar
signed	con signo	int, char
unsigned	sin signo	int, char
long	largo	int, char, double
short	corto	int, char

Preprocesador: Directivas #include y #define

#**include**: Al encontrar una línea de #include seguido de un nombre de archivo, el preprocesador la sustituye por el archivo.
#**define**: Crea una macro que luego puede ser sustituida en el código fuente
#if, #else, #endif: Compilación de una parte del código condicional

Clase 6: FUNCIONES DE E/S: Printf(), scanf() y Getchar()

Scanf. Lee datos de la entrada estándar y almacena en una variable:

- Sintaxis: **printf** ("cadena de control", lista de argumentos); Ej: **Scanf ("% d", &a);**

Printf: Para mostrar el valor almacenado en una variable:

- Sintaxis: **scanf** ("cadena de control", lista de argumentos); **printf ("El valor a% d", a);**

Scanf puede obtener varias variables a la vez y de distintos tipos. ejemplo:

scanf ("%**d** %**s** %**f**", &var_entera, var_caracteres, &var_float);

Especificadotes o argumentos

Codigo	Formato o variable
%c	Char – 1 caracter simple
%d o %i	Entero decimal
%e	Flotante o doble notación científica
%f	Flotante o doble decimal con signo
%g	Usa el mas corto de los anteriores
%o	Entero octal sin signo
%s	Cadena de varios caracteres
%u	Entero decimal sin signo
%x	Entero hexadecimal sin signo
%	Signo de tanto por ciento: %
%p	Pointer, puntero de dirección
%n	puntero a entero con nº caracteres

Ejemplos:

Sentencia printf ()	Salida
(":%f:", 123.456)	:123.456001:
(":%e:", 123.456)	:1.234560e+02:
(":%g:", 123.456)	:123.456:
(":%-2.5f:", 123.456)	:123.45600:
(":%-5.2f:", 123.456)	:123.46:
(":%5.5f:", 123.456)	:123.45600:
(":%10s:", "hola")	:hola:
(":%-10s:", "hola")	:hola :
(":%2.3s:", "hola")	:hol:
(":%x:", 15)	:f:
(":%o:", 15)	:17:
(":%05d:", 15)	:00015:
(":abc:%n", &var_int)	:abc:

Caracteres acentuados: Alt+160 =á - Alt+130 =é - Alt+161 =í - Alt+162 =ó Alt+163 =ú

Programas ejemplos:

/* Programa que lee números enteros de teclado hasta que se introduce un 0.
El programa no muy útil pero sí instructivo

```
#include <stdio.h> /* para scanf */
int main (void)
{
  int x;
  do
    {
      scanf ("%d", &x);
    } while (x != 0);
}
```

/* para las funciones printf() y getchar() */
/*Mientras el caracter sea distinto de INTRO imprimelo */

```
#include <stdio.h>
int main(void)
{
   int c;

   while ((c = getchar()) != '\n')
            printf("%c", c);
   return 0;
}
```

Operadores

Aritméticos: + mas - menos * por / división % resto
Relacionales: **>=** mayor o igual **!=** diferente de… **= =** igual que…
Lógicos: NO negación: **!**, Y AND lógico: **&&** O OR lógico: **||**

Incrementales:

++ Incremento: i=i+1 → i++
-- Decremento: i=i-1 → i -
+ = - = /= Asignación compleja
¬ SUMA → SUMA+NUM → Primero se suma y luego se asigna.

Otros operadores:

& Dirección a un objeto * Direccionamiento indirecto [ ] Direccionamiento indexado
Sizeof( ) Tamaño de la variable , reunir varias expresiones en una función

EJERCICIOS

```
#include <stdio.h>              //ok edu
main ()
{
 int nume1,nume2,suma=0;
  printf ("Suma en C.\n");
  printf ("Escriba el primer numero:");
  scanf("%d",&nume1);
  printf ("Escriba el segundo numero:");
  scanf("%d",&nume2);
  suma=nume1+nume2;
  printf ("%d mas %d son %d",nume1,
            nume2,suma);
  printf("\n");
  printf ("Pulsa RETURN para terminar.");
  scanf("%d"); //o getchar()
}
```

```
#include <stdio.h>              //ok edu

main ()   //crear un menu
{
int x = 0;
while (x! = 3)
{
  printf ("1.-Opcion menu1\n");
  printf ("2.-Opcion menu2\n");
  printf ("3.-Salir\n\n");
  printf ("Escoja opcion: ");
  scanf("%d",&x);
  switch (x)
 {
  case 1:printf ("Elegido opcion 1\n\n");
  case 2:printf ("Elegido Opcion 2\n\n");
}
}
}
```

Clase 7: OPERACIONES ARIMETICAS

Condiciones o comparaciones lógicas:

Puede ser cualquier prueba de un valor contra otro. Por ejemplo:

- a > 0 -> Verdadero si contiene un valor mayor que cero;
- b < 0 -> Verdadero si B contiene un valor inferior a cero.
- == 0 -> Prueba de igualdad

Para probar si algo es igual a otra cosa se utiliza la secuencia de caracteres == y no =.

El siguiente programa pide dos números flotantes en formato "%f" *flotante*. Luego calcula la suma, diferencia, producto y cociente y los valores los imprime en formato "%g" *flotante corto* y espera a que el usuario pulse ↵

```
#include <stdio.h>        /* para utilizar: printf (), getchar () */
void main (void)          // ->ok en Borland TurboC++ Unit1   - en C++ funciona con main (void)
{
  float numero1, numero2;                   //variables decimal flotantes float
  float suma, diferencia, producto, cociente;
  printf("Escribe primer numero:");
    scanf("%f",&numero1);                   //pide en formato %f el numero 1
  printf("Escribe segundo numero:");
    scanf("%f",&numero2);                   //pide en formato %f el numero 2
      suma = numero1 + numero2;
      diferencia = numero1 - numero2;
      producto = numero1 * numero2;
      cociente = numero1 / numero2;
  printf ("OPERACIONES ARITMETICAS CON DOS NUMEROS:");
  printf ("\n\nNumero 1: %g", numero1);       //muestra numero 1 en formato %g flotante corto
  printf ("\nNumero 2: %g", numero2);         //muestra numero 2 en formato %g flotante corto
  printf ("\n\nSuma: %g", suma);
  printf ("\nDiferencia: %g", diferencia);
  printf ("\nProducto: %g", producto);
  printf ("\nCociente: %g", cociente);
  printf ("\n\nPulsa la tecla RETURN para terminar programa.");
  getchar ();
}
```


CALCULO DE LA LONGITUD Y AREA DEL CIRCULO

```
#include <stdio.h> // printf (), getchar ()

main (void)
{
  const float pi = 3.141592;        // declara y asigna constante flotante Pi
  float radio, longitud, area;      // declara variables flotantes

  printf("Escribe el radio: ");
  scanf("%f",&radio);               // pide el radio

  longitud = 2 * pi * radio;
  area = pi * radio * radio;

  printf ("Calculo de la longitud y area de un circulo de radio %g:\n\n", radio);
  printf ("LONGITUD: %f\n", longitud);
  printf ("AREA    : %f\n", area);
  getchar ();
}
```

El siguiente programa pide el radio del círculo y calcula su longitud y área .

Para ello declara una **constante** Pi y tres variables del tipo float

Clase 8: TAMAÑO DE LOS ESPECIFICADORES

Este programa muestra el tamaño de los tipos más usados en C en el sistema en el cual se ejecuta.

```
#include <stdio.h> /* printf(), getchar() */

void main (void)
{
 printf ("TAMAÑO EN BYTES DE ALGUNOS TIPOS \n");
 printf ("  Tipo          Tamaño (en bytes)\n");
 printf ("  ------------  ----------------\n");
 printf ("  char             %d       \n", sizeof (char));
 printf ("  int              %d       \n", sizeof (int));
 printf ("  short int        %d       \n", sizeof (short int));
 printf ("  long int         %d       \n", sizeof (long int));
 printf ("  float            %d       \n", sizeof (float));
 printf ("  double           %d       \n", sizeof (double));
 printf ("  long double      %d       \n", sizeof (long double));
 printf ("\nPulsa la tecla RETURN para salir.\n");
 getchar ();
```

Funciones comunes:

Abs	Retorna el valor absoluto	Pow	Eleva a la potencia
Asin	Retorna el arco seno	Rand	Retorna valor aleatorio
Cos	Retorna del coseno	Srand	Pone punto de inicio para Rand
Exit	Cierra archivos y termina el programa	Sqrt	Raíz cuadrada
Exp	Eleva a la potencia d	Strcat	Concatena cadenas de texto
Fabs	Retorna el valor absoluto	Strcmp	Compara cadenas de texto
Fclose	Cierra el archivo	Strcomp	Compara cadenas de texto
Feof	Indica si ha llegado al final del archivo	StrCpy	Copia cadenas de texto
Fgetc/getc	Lee un carácter del archivo	Strlen	Cuenta caracteres del texto
Floor	Redondea al valor entero	System	Pasa la orden al sistema Op.
Fmod	Retorna el resto	Tan	Retorna la tangente
Fopen	Abre el archivo	Time	Retorna segundos transcurridos
Fputc	Escribe un carácter en el archivo	Toupper	Convierte letra a mayúsculas
Log	Retorna logaritmo natural	Fabs	Retorna valor absoluto

Funciones para cadenas de caracteres:

Están en el fichero cabecera: string.h

- STRLEN(): Cuenta al num de caracteres de la cadena
- STRCAT(): Concatena 2 cadenas
- STRCMP(): Compara 2 cadenas
- STRCPY(): Copia una cadena en otra
- STRCHR() y STRSTR(): Busca un carácter o un string.
- Toupper():Convierte letra a mayúsculas

```
#include <stdio.h>
#include <string.h>
int main(void)
{
   char *string = "OFIMEGA";
   printf("%d\n", strlen(string));
   return 0;
}
```

```
#include <string.h>
#include <stdio.h>
int main(void)
{
   char destination[25];
   char *source = " States";
   strcpy(destination, "United");
   strncat(destination, source, 7);
   printf("%s\n", destination);
   return 0;
}
```

Streams – Cin - Cout:

En C++ a diferencia de C las entradas y salidas se leen desde ***streams*** (*canal, flujo o corriente*)
Para su uso hay que incluir: **#include <iostream.h>** entonces de pueden introducir y sacar datos con
Cin: Lee datos (por defecto del teclado)
Cout: Imprime el texto entre comillas tal y como está escrito.
>> : operador de inserción acompaña a cin
<< : operador de extracción acompaña a cout

```
Int i=1;
Double tiempo=2;
Float acel=9.8;
Const float Vo=5;
Cout <<"Resultado numero:"<<i<<" en el instante" << tiempo << " la velocidad vale" <<
        v0+acel*tiempo <<"\n";
```

```
#include <stdio.h>                                          // (Comprobado con Dev-C++)
#include <iostream>
/* Se pueden cambiar las dos líneas anteriores de cabecera por:  using namespace std; */

int main ()
{
   char salir;

   salir = '\0';
   while (salir != 'q')
   {
      cout << "Hola, Esta es una aplicación con bucle while" << endl;
      cout << "Muestra mensaje con cout y pide con cin  " << endl;
      cout << "Con cin pide y guarda la variable salir" << endl;
      cout << "mietras la variable salir no sea la letra q" << endl;
      cout << "Pulse q para salir " << endl;
      cin >> salir;
   }
   return 0;
}
```


SENTENCIAS QUE CAMBIAN EL FLUJO DEL PROGRAMA:

SENTENCIAS CONDICIONALES:

Sentencia if

```
if (contador < 50)
  contador++;
```

```
if (x < y)
  z = x;
  else
  z = y;
  }
```

```
if (ch == '\n')
{
numero_de_lineas++;
numero_de_caracteres++;
```

```
#include <stdio.h>
main ()                          /* Division */
 {
   int num1, num2;
   printf("\nPrimer número");
   scanf("%d",&num1);
   printf("\nSegundo número ");
   scanf("%d",&num2);
   if (num2 ==0) printf("\n\nNodivisible por cero\n\n");
   else          printf("\n\nRerspuesta:
%d\n\n",num1/num2);
 }
```

En este ejemplo se evalúa si el denominador es cero antes de dividir

El operador **?** se puede utilizar para reemplazar las sentencias if-else.

Ejemplo 1 con if-else */ /* Ejemplo 1 con ?: */

```
if (x < = y)
  z = x;
else
  z = y;
```

```
z = x <= y ? x : y;
```

Ejemplo 2 con if-else */ /* Ejemplo 2 con ?: */

```
if (n == 1)
  printf ("Mensaje 1");
else
  printf ("Mensaje 2");
```

```
n == 1 ?
printf ("Mensaje 1"):
printf ("Mensaje 2");
```

Sentencia switch

Es un If con múltiples ramificaciones (equivalente al ***case*** en pascal)

Forma general:

```
switch (expresión)
 {
   case expresión_constante_1:
      sentencias_1
      break;    //      -> para que pare de leer
   case expresión_constante_2:
      sentencias_2
      break;
   case expresión_constante_3:
      sentencias_3
      break;
    .
    .
   default:
      sentencias_n
 }
```

Ejemplo:

```
switch (operando)
 {
   case 1:
      x *= y;
      break;
   case 2:
      x /= y;
      break;
   case 3:
      x += y;
      break;
   case 4:
      x -= y;
      break;
   default:
     printf ("¡ERROR!");
 }
```

SECUENCIAS DE BUCLE:

Sentencia while:

Evalúa y ejecuta mientras es cierta.

Ejemplo:

imprime los cinco primeros números naturales:

```
#include <stdio.h>

int main (void)
{
    int i;
    i = 1;
 while (i <= 5)
   {
     printf ("\ni = %d", i);
     i++;
   }
getchar ();
}
```

Sentencia for

Sintaxis general:

for (expresión_1; expresión_2; expresión_3)
sentencia

En la sentencia for, a expresión_1 se le llama inicialización, a expresión_2 se le llama condición, y a expresión_3 se le llama incremento.

```
#include <stdio.h>

main()
{
   int fahr;

   for ( fahr = 0 ; fahr <= 300 ; fahr = fahr + 20)
      printf("%4d %6.1f\n" , fahr , (5.0/9.0)*(fahr-32));
}
```

Ejemplo. Contar hasta 100:

```
#include <stdio.h>
main (void)
{
  printf ("Lista de los 100 primeros"
          "números naturales:\n");
  int x;
  for (x = 1; x <= 100; x++)
    printf ("%d ", x);
    getchar();
}
```

Sentencia do-while:

Primero ejecuta y después evalúa

Sintaxis general:

do
sentencia
while (expresión);

Sentencia: Continue: Salta el resto del bucle

```
/*   imprime los 50 primeros
   números naturales*/

#include <stdio.h>
int main (void)
{
  int i = 1;
  do
    {
      printf ("%d ", i++);
    } while (i <= 50);
getchar();
}
```

Ejercicio secuencia aleatoria

Generación de 0 y 1 sin que se repita la misma secuencia

```
#include <time.h>   /* randomize () */
#include <stdlib.h> /* randomize (), random () */
#include <stdio.h>  /* printf (), getchar () */

void main (void)
{
  const int numero_de_valores_a_generar = 80 * 24, valor_maximo = 2;
  int i;

  randomize ();
  for (i = 1; i <= numero_de_valores_a_generar; i++)
    printf ("%d", random (valor_maximo));
  getchar ();
}
```


SWITCH: TRANSFORMACION DE NOTA NUMERICA A ALFABETICA

```
/*  Este programa lee una calificación entre 0 y 10 y la transforma en nota
 alfabética según la siguiente tabla:

      Nota numérica        Nota alfabética
    0 <= NOTA < 3          MUY DEFICIENTE
    3 <= NOTA < 5          INSUFICIENTE
    5 <= NOTA < 6          SUFICIENTE
    7 <= NOTA < 9          NOTABLE
    9 <= NOTA <= 10        SOBRESALIENTE
Imprimiendo el resultado.  */
```

```
#include <stdio.h> // printf (), scanf (), getchar ()

main (void)
{
  int nota;

  printf ("Introduce una calificación (0-10): ");
  scanf ("%d", &nota);

  printf ("\nLa notación numérica %d equivale a la alfabética ", nota);
  switch (nota)
   {
     case 0:
     case 1:
     case 2:
       printf ("MUY DEFICIENTE.");
       break;
     case 3:
     case 4:
       printf ("INSUFICIENTE.");
       break;
     case 5:
       printf ("SUFICIENTE.");
       break;
     case 6:
       printf ("BIEN.");
       break;
     case 7:
     case 8:
       printf ("NOTABLE.");
       break;
     case 9:
     case 10:
       printf ("SOBRESALIENTE.");
       break;
     default:
       printf ("NOTA INCORRECTA.");
   }

  printf ("\n\nPulsa la tecla RETURN para finalizar.");
  getchar (); /* esta función espera a que el usuario pulse RETURN */
}
```

CALCULO DEL PROMEDIO O LA MEDIA

```
#include <stdio.h> // printf (), scanf ()
#include <conio.h> // getch ()

main (void)
{
  int nota, numero_de_notas, suma_de_las_notas;

  printf ("\nPara terminar de introducir notas, escribir nota negativa.\n\n");
  printf ("Introduce nota 1: ");
  scanf ("%d", &nota);
  numero_de_notas = suma_de_las_notas = 0;
  while (nota >= 0)
    {
      numero_de_notas++;
      suma_de_las_notas += nota;
      printf ("Introduce nota %d: ", numero_de_notas + 1);
      scanf ("%d", &nota);
    }
  if (numero_de_notas == 0)
    printf ("\n\nNo se ha introducido ninguna nota.");
  else
    printf ("\n\nMedia de las notas introducidas: %g",
          (float) suma_de_las_notas / numero_de_notas);
  printf ("\n\n\nPulsa cualquier tecla para finalizar.");
  getch ();
}
```

/* Este programa lee por el teclado las notas de los alumnos de una clase y calcula el número de ellas y su media. Para terminar la introducción de notas se introduce una nota negativa. */

Constantes:

Información que no cambia. Numérica o de carácter. **Const Int size=5**
Constantes de enumeración: **enum dia {lunes=1, martes, miércoles, jueves, viernes}**
Asocia el valor 1 y continúa con el resto.

Clase 9: FUNCIONES Y VARIABLES

VARIABLES LOCALES: Son aquéllas que se declaran dentro de una función.
PARAMETROS FORMALES: Si una función va a usar argumentos, entonces debe declarar las variables que van a aceptar los valores de esos argumentos. Estas variables son las
VARIABLES GLOBALES: Válidas en todo el programa, se crean al declararlas en cualquier parte fuera de una función.
EXTERN: Cuando un programa está compuesto de varios ficheros y tenemos una variable global a varios de ellos.
STATIC

- VARIABLES ESTATICAS LOCALES: Vuelven a tener el valor anterior al volver a la función.
- VARIABLES ESTATICAS GLOBALES: No se reconocen ni alteran por otros ficheros.

REGISTER: Mantiene el valor en los registros de la CPU en lugar de en memoria. Su acceso es más rápido.
AUTO: Las variables auto (automáticas) son todas aquellas variables locales que no son estáticas. No se usa.
VALORES DEVUELTOS:
Todas las funciones, excepto aquéllas del tipo ***void***, devuelven un valor.
Este valor se indica en la sentencia ***return*** y si no existe ésta, el valor es 0.
En las funciones de tipo void se puede hacer: *return*; saliendo sin devolver valor.

Ejemplo de llamadas a funciones:
Máximo y potencia son dos funciones llamadas desde la función Main.
A la función máximo se le pasan dos valores enteros y devuelve con *Return* una valor entero

```
#include <stdio.h>

int maximo (int, int);        // ->Declaramos la función maximo
long potencia (int, int);     // -> Declaramos la función potencia

void main (void)
{
  int a = 2, b = 3, c = 4, d = 5;
  printf ("\nEl máximo entre %d y %d es %d.", a, b, maximo (a, b));
  printf ("\n%d elevado a %d es %d.\n", c, d, potencia (c, d));
}

int maximo (int ma, int mb)   // ->Función máximo tipo entera
{
  return ma >= mb ? ma : mb; //-> ? equivale a if
}
long potencia (int pa, int pb)   // ->Función potencia tipo long
{
  int i;
  long pot = 1;
  for (i = 1; i <= pb; i++)
    pot *= pa;
  return pot;
}
```

La salida de este programa es:

El máximo de 2 y 3 es 3.

4 elevado a 5 es 1024

Clase 10: Funciones principales

FUNCIONES DE CONTROL DE PROGRAMA

FUNCION exit()

exit ()-> Se encuentra declarada en la biblioteca estándar (***stdlib.h***), da lugar a la terminación automática de un programa.
Provoca la terminación inmediata del programa sin realizar el vaciado de los buffers ni cerrar los ficheros.
exit (0)->exit (EXIT_SUCCESS); - salida 0 normal
exit (1)->exit (EXIT_FAILURE); - salida 1 anormal

FUNCION abort ()

Aborta el programa. Es muy similar a la función exit.
La diferencia es que no acepta parámetros, no vacía los buffers ni cierra ningún fichero.
Ejemplo:

```
#include <stdlib.h> /* para poder utilizar la función exit () */

/* las funciones tarjeta_color () y jugar () han de estar definidas en
   algún lado */

void main (void)
{
  /* tarjeta_color () es una función que devuelve 0 (falso) si la tarjeta
     del sistema no es color y 1 (cierto) si lo es */
  if (tarjeta_color ())
    exit (1); /* terminación anormal: la tarjeta no es color */
  jugar (); /* llamada a función para jugar */
  exit (0); /* terminación normal, esta sentencia no es necesaria */
}
```

FUNCION assert ()

Esta función testea la expresión dada; si la expresión es cierta no hace nada; si la expresión es falsa escribe un mensaje en la salida de error estándar y termina la ejecución del programa.

Sintaxis:

Assertion failed: <expresión>, file <fichero>, line <num_línea>
Abnormal program termination

Ejemplo:

```
#include <assert.h> /* para poder utilizar la función assert ()

void main (void)
{
  int x, y;
  x = y = 1;
  assert (x < y); /* la expresión x < y es falsa y el programa termina */
  x++; y++; /* estas dos sentencias nunca se ejecutan */
}
```

FUNCIONES putchar () y getchar ()

La función putchar escribe un carácter en la salida estándar.

La función getchar escribe un carácter en la entrada estádar.

Ejemplo:
Se ejecuta el programa de la siguiente forma:

```
/* fichero ejemplo.c */

#include <stdio.h> /* para: getchar (), putchar (), EOF */

void main (void)
{
  int ch;
  while ((ch = getchar ()) != EOF)
    putchar (ch);
}
```

Se leen caracteres de teclado y se escriben en pantalla. Se leen caracteres hasta que se encuentra el carácter de marca de fin de fichero que en el DOS se escribe con: ***CONTROL-Z***.
Si se ejecuta el programa con redirección: ejemplo: < fichero_fuente > fichero_destino se produce una copia del fichero_fuente al fichero_destino. Ejemplo: >> fichero_destino en este caso se leen caracteres de teclado y se añaden al *fichero_destino*.

FUNCIONES puts () y gets ()

La función puts escribe una cadena de caracteres y un carácter de nueva línea al final de la cadena en la salida estándar.
La función gets lee una cadena de caracteres de la entrada estándar hasta que se encuentra el carácter '\n', aunque este carácter no es añadido a la cadena.

```
char cadena [100];
gets (cadena);
```

es similar a:

```
char cadena [100];
scanf ("%100s", cadena);
```

Ejemplo:

```
#include <stdio.h> /* printf(), scanf()*/  -> OK
#include <conio.h> /* getch(), putch() */
main (void)
{
  float f;
  int i;
  printf ("Introduce un número entero\n"
          "comprendido entre 0 y 5: ");
  while ((i = getch ()) < '0' || i > '5')
    ;
  putch (i);
  printf ("\nNúmero introducido: %d",
          i -= '0');
  printf ("\nIntroduce un número en\n"
          "coma flotante: ");
  scanf ("%f", &f);
  printf ("\nFormato %%f: %f\nFormato "
    "%%e: %e\nFormato %%g: %g", f, f, f);
getchar ();
}
```


Clase 11: DIBUJO DE UN GRÁFICO

Este programa dibuja un gráfico de una forma recursiva. Está implementado En Turbo C.
La atractiva forma gráfica que muestra en la ejecución del programa está formada por la superposición de cinco curvas. Estas siguen un esquema regular y sugieren que deben poder dibujarse por un «plotter» controlado por computador. El objetivo a conseguir es descubrir el esquema recursivo con el que debe construirse el programa que haga el dibujo.
Por inspección
se observa que tres de las curvas superpuestas tienen la forma indicada en los dibujos siguientes; se designan por H1, H2 y H3.
Las figuras muestran que H(i+1) se obtiene por composición de cuatro curvas de tipo H(i) de tamaño mitad, giradas apropiadamente, y unidas por tres líneas. Obsérvese que H(1) puede considerarse formada por cuatro curvas de un tipo vacío H(0) conectadas por tres rectas. H(i) se denomina curva de Hilbert de orden i, en honor a su inventor D. Hilbert (1891).

```
Curva de Hilbert de orden 1:
   +-------
   ¦
   ¦
   +-------
Curva de Hilbert de orden 2:
   +-------+
   ¦       ¦   ¦
   +---+   +---+
       ¦
   +---+   +---+
   ¦       ¦   ¦
   +-------+
Curva de Hilbert de orden 3:
   +---+ +---+ +-
   +-+ +-+ +-+ +-+
   +-+ +-+ ¦ +-+ ¦
   +---+ ¦ +-+ +-+
   +---+ ¦ +-+ +-+
   +-+ +-+ ¦ +-+ ¦
   +-+ +-+ +-+ +-+
   +---+ +---+ +-
```

Supóngase que las herramientas básicas de que se dispone para dibujar son dos variables de coordenadas x e y, un procedimiento setplot (situar la pluma del «plotter» en las coordenadas x e y) y un procedimiento plot (que mueve la pluma de dibujo desde la situación actual a la posición indicada
por x e y).

Como cada curva H(i) está formada por cuatro copias de tamaño mitad de la curva H(i-1) es lógico expresar el procedimiento de dibujar H(i) como compuesto de cuatro partes, cada una dibujando una H(i-1) del tamaño apropiado, convenientemente girada. Si se denomina cada parte, respectivamente, por A, B, C y D, y las rutinas que dibujan las correspondientes líneas de interconexión se representan por flechas apuntando en la dirección correspondiente aparece el siguiente esquema recursivo:

```
+-
+->  A:  D <- A _  A -> B

+-+
¦ _  B:  C _  B -> B _  A

<-+
-+  C:  B -> C _  C <- D

_ ¦
+-+  D:  A _  D <- D _  C
```

Si se designa la línea unitaria por h, el procedimiento correspondiente al esquema A se expresa inmediatamente utilizando activaciones recursivas de los procedimientos designados análogamente por B y D y del propio A.

```
procedimiento A (i es entero)
  si i > 0 entonces
    D (i-1); x = x - h; plot;
    A (i-1); y = y - h; plot;
    A (i-1); x = x + h; plot;
    B (i-1);
  finsi
finprocedimiento
```

Este procedimiento se inicia por el programa principal una vez por cada curva de Hilbert a superponer. El programa principal determina el punto inicial de la curva, o sea, los valores iniciales de x e y, y el incremento unitario h. S llama h0 al ancho total de la página, que debe ser h0 = (2 elevado a k) para algún k _ n. El programa completo dibuja las n curvas de Hilbert H1, ... Hn.

Descripción de las características gráficas utilizadas de Turbo C (vers. Borland C++ 2.0).
Para trabajar con gráfico es necesario incluir el fichero: **<graphics.h>**.

En Turbo C, siempre que se vaya a utilizar el modo gráfico, es necesario inicializarlo y cerrarlo. La inicialización se hace con la función: **initgraph()**.

El prototipo de esta función es:
*void **initgraph** (int *graphdriver, int *graphmode, char *pathtodriver);*
donde: ***graphdriver*** debe contener el dispositivo gráfico;

los valores que puede tomar son:

CGA	MCGA
EGA	EGA64
EGAMONO	IBM8514
HERCMONO	ATT400
VGA	PC3270
DETECT (autodetección)	

En **graphmode** se guarda el modo gráfico para cada dispositivo gráfico; los valores que puede tomar ***entre otros*** son son:

Tipo	Resolución	Colores	Tipo	Resolución	Colores
CGAC0	320 x 200	paleta 0	HERCMONOHI	720 x 348	2 colores
CGAC1	320 x 200	paleta 1	ATT400C0	320 x 200	paleta 0
CGAC2	320 x 200	paleta 2	ATT400C1	320 x 200	paleta 1
EGALO	640 x 200	16 colores	ATT400C2	320 x 200	paleta 2
EGAHI	640 x 350	16 colores	VGALO	640 x 200	16 colores
EGA64LO	640 x 200	16 colores	VGAMED	640 x 350	16 colores
IBM8514LO	640 x 480	256 colores	IBM8514HI	1024 x 768	256 colores

- En ***pathtodriver*** debe estar el path del directorio donde están los ficheros con extensión .BGI. Su sintaxis es: "..\\bgi\\drivers"
- El modo gráfico se cierra con la función: **closegraph**() cuyo prototipo es:
- void closegraph (void);
- La esquina izquierda superior de la pantalla gráfica tiene coordenadas: 0, 0

Funciones gráficas utilizadas en este programa:
1) void lineto (int x, int y): Dibuja una línea desde la posición actual del cursor hasta (x,y).
2) void moveto (int x, int y): Mueve el cursor gráfico a (x,y).
3) int graphresult (void): Devuelve un código de error para la última operación gráfica no exitosa o grOk si la última operación gráfica ha tenido éxito.
4) char *grapherrormsg (int errorcode);
Devuelve un puntero a un string con el mensaje de error.
5) void setviewport (int izq, int ar, int der, int ab, int clip): Crea un viewport para la salida gráfica. Un viewport es una ventana de trabajo en la pantalla gráfica de tal modo que la coordenada (0,0) corresponde a la esquina izquierda superior del viewport y no de la pantalla.
El parámetro clip especifica si los valores fuera del viewport son recortados.

Los códigos de error devueltos por graphresult()

grOk	grNoInitGraph
grNotDetected	grFileNotFound
grInvalidDriver	grNoLoadMem
grNoScanMem	grNoFloodMem
grFontNotFound	grNoFontMem
grInvalidMode	grError
grIOerror	grInvalidFont
grInvalidFontNum	grInvalidDeviceNum
grInvalidVersion	

Las características gráficas de Turbo C se estudiará en lecciones posteriores
Aquí sólo hemos dado los conceptos básicos sobre ellas para comprender cómo está hecho el programa. */


```
#include <graphics.h> /* lineto (), moveto (), setviewport (), initgrah (), graphresult (), grOk, getmaxx (), getmaxy ()      */
#include <stdio.h>    /* printf () */
#include <conio.h>    /* geth () */
#include <stdlib.h>   /* exit (),EXIT_FAILURE/

int h0, h, x, y;

void a (int), b (int), c (int), d (int);
void inicializar_grafico (void);

#define empezargraf  inicializar_grafico ();
#define plot         lineto (x, y)
#define setplot      moveto (x, y);
#define terminargraf closegraph ();

void main (void)
{
  const int n = 4;
  int i, x0, y0;
  empezargraf;
  i = 0;
  h = h0;
  x0 = h / 2;
  y0 = x0;
  do
    {
      i++;
      h /= 2;
      x0 += (h/2);
      y0 += (h/2);
      x = x0;
      y = y0;
      setplot;
      a (i);
    } while (i != n);
   getch ();
   terminargraf;
}

void inicializar_grafico (void)
{
  int gdriver = DETECT, gmode;
  int vpx1, vpy1, vpx2, vpy2;
  int errorcode;
  initgraph (&gdriver, &gmode, "");
  errorcode = graphresult ();
  if (errorcode != grOk)
    {
      printf ("Error Gráfico: %s\n", grapherrormsg (errorcode));
      getch ();
      exit (EXIT_FAILURE);
    }
  h0 = getmaxx () + 1 >= 640 && getmaxy () + 1 >= 480 ? 512 : 128;
  /* calcula las coordenadas de viewport para crear un viewport en el centro de la pantalla de altura y anchura h0 si puede; esto se hace para que el gráfico salga centrado en la pantalla */
  vpx1 = (getmaxx () + 1 - h0) / 2 + 1;
  vpy1 = getmaxy () + 1 <= h0 ? 0 : (getmaxy () + 1 - h0) / 2 + 1;
  vpx2 = vpx1 + h0 - 1;
  vpy2 = getmaxy () + 1 <= h0 ? getmaxy () : vpy1 + h0 - 1;

  setviewport (vpx1, vpy1, vpx2, vpy2, 0);
}

void a (int i)
{
  if (i > 0)
    {
      d (i-1); x -= h; plot;
      a (i-1); y -= h; plot;
      a (i-1); x += h; plot;
      b (i-1);
    }
}

void b (int i)
{
  if (i > 0)
    {
      c (i-1); y += h; plot;
      b (i-1); x += h; plot;
      b (i-1); y -= h; plot;
      a (i-1);
    }
}

void c (int i)
{
  if (i > 0)
    {
      b (i-1); x += h; plot;
      c (i-1); y += h; plot;
      c (i-1); x -= h; plot;
      d (i-1);
    }
}

void d (int i)
{
  if (i > 0)
    {
      a (i-1); y -= h; plot;
      d (i-1); x -= h; plot;
      d (i-1); y += h; plot;
      c (i-1);
    }
}
```

Clase 12: ARRAYS

ARRAYS UNIDIMENSIONALES

La forma general de declaración de un array unidimensional es:

especificador_de_tipo nombre_variable [tamaño];

donde especificador_de_tipo es el tipo base, es decir, el tipo de cada elemento y tamaño es el número de elementos del array.

La forma general de acceder a un elemento del array es:
nombre_variable [indice]
La longitud en bytes de un array se calcula mediante la fórmula:
total_de_bytes = sizeof (tipo) * numero_de_elementos

ARRAYS BIDIMENSIONALES

Un array bidimensional es, en realidad, un array unidimensional donde cada elemento es otro array unidimensional. Los arrays bidimensionales son un caso particular de los arrays multidimensionales.

Así como a los arrays unidimensionales se les suele llamar vectores, a los arrays bidimensionales se les suele llamar matrices.

La forma general de declaración es:
especificador_de_tipo nombre_variable [tamaño_1] [tamaño_2];
y se accede a los elementos del array:
nombre_variable [indice_1] [indice_2]

Inicializar arrays multidimensionales: 2 formas:

```
int m [3] [4]
  {
    1, 2, 3, 4,
    5, 6, 7, 8,
    9, 10, 11, 12
  };
```

```
int m [3] [4]
{
  { 1, 2, 3, 4 },
  { 5, 6, 7, 8 },
  { 9, 10, 11, 12 }
};
```

EJEMPLO DE ARRAY BIDIMENSIONAL

```
#include <stdio.h>
void main (void)
{
  #define num_filas    4
  #define num_columnas 7
  int i, j, matriz [num_filas] [num_columnas];
  for (i = 0; i < num_filas; i++)
    for (j = 0; j < num_columnas; j++)
      matriz[i][j] = i + j;
  for (i = 0; i < num_filas; i++)
    {
      for (j = 0; j < num_columnas; j++)
        printf ("%2d ", matriz[i][j]);
      putchar ('\n');
    }
}
```


MISMO EJEMPLO ANTERIOR PERO CON ARRAYS COMO ARGUMENTOS

```
#include <stdio.h>

/* declaramos  las dos funciones fuera del main*/
void rellenar_matriz (int matriz[][4]);
void imprimir_matriz (int matriz[][4]);

int i, j;

void main (void)
{
        int matriz [4][4];   //  -> declaramos variable matriz entera de 4x4:

        rellenar_matriz (matriz);    //  llamamos a función rellenar y pasa parámetro
        imprimir_matriz (matriz);   //  llamamos a función imprimir y pasa parámetro
}                                    //->aquí termina el programa principal


 void rellenar_matriz (int matriz[][4])   //  -> función rellenar
 {
   for (i = 0; i < 4; i++)
     for (j = 0; j < 4; j++)
         matriz[i][j] = i+j;
 }
 void imprimir_matriz (int matriz[][4])  //  -> función imprimir
 {
   for (i = 0; i < 4; i++)
     {
       for (j = 0; j < 4; j++)
           printf ("%2d ", matriz[i][j]);
           putchar ('\n');
     }
scanf("%d");
 }
```

Ejemplo de inicialización de un vector: int v[5] = { 1, 2, 3, 4, 5 };

La inicialización de cadenas; se puede hacer de dos formas:
char cadena[4] = "abc"; o char cadena[4] = { 'a', 'b', 'c', '\0' };

EJEMPLO:

```
#include <stdio.h>
void main (void)
{
  /*    si da problema las dos inicializaciones de arrays solucionar de dos
formas: o bien haces estos dos arrays globales o los hace locales estáticos  */
  char meses[12][11] = {"Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio",
"Julio", "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"};
  int dias[12] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
  register short i;
  for (i = 0; i < 12; i++)
    printf ("El mes de %-10s tiene %d días.\n", meses[i], dias[i]);
}
```

Clase 13: TRES EN RAYA

/* Programa que permite jugar al usuario a las tres en raya con la computadora.

El programa representa la matriz de tres en raya como un array de caracteres de 3 por 3. El jugador siempre es X y la computadora O.

Cuando mueve el jugador, el carácter X se coloca en la posición especificada de la matriz. Cuando le toca mover a la computadora, recorre la matriz y pone O en la primera posición vacía de la matriz. Si no encuentra una posición vacía, lo indica y sale.
*/

```
#include <stdio.h>  /* printf (), scanf () */
#include <stdlib.h> /* exit ()             */
#include <conio.h>  /* getch ()            */

#define ESPACIO ' '

char matriz[3][3] =
  {
    ESPACIO, ESPACIO, ESPACIO,
    ESPACIO, ESPACIO, ESPACIO,
    ESPACIO, ESPACIO, ESPACIO
  };

void obtener_movimiento_de_jugador (void);
void obtener_movimiento_de_computadora (void);
void mostrar_matriz (void);
char comprobar (void);

void main (void)
{
  char hecho = ESPACIO;

  printf ("JUEGO DE LAS TRES EN RAYA.\n\n");

  do
    {
      mostrar_matriz ();
      obtener_movimiento_de_jugador ();
      hecho = comprobar (); /* ver si gana el jugador */
      if (hecho != ESPACIO)
        break;
      obtener_movimiento_de_computadora ();
      hecho = comprobar (); /* ver si gana la computadora */
    } while (hecho == ESPACIO);

  if (hecho == 'X')
    printf ("\n¡HAS GANADO!\n");
  else
    printf ("\n¡YO GANO!\n");

  mostrar_matriz (); /* mostrar las posiciones finales */
  printf ("\n\nPulsa cualquier tecla para finalizar. ");
  getch ();
}

void obtener_movimiento_de_jugador (void)
{
```



```
  int x, y;

  printf ("\nIntroduzca sus coordenadas de la X (1¾fila¾3, 1¾columna¾3): ");
  scanf ("%d%d", &x, &y);
  x--;
  y--;
  if (matriz[x][y] != ESPACIO)
    {
      printf ("Movimiento inválido, prueba de nuevo.\n");
      obtener_movimiento_de_jugador ();
    }
  else
    matriz[x][y] = 'X';
}

void obtener_movimiento_de_computadora (void)
{
  int encontrado = 0;
  register int i, j;

  for (i = 0; ! encontrado && i < 3; i++)
    for (j = 0; ! encontrado && j < 3; j++)
      if (matriz[i][j] == ESPACIO)
        encontrado = 1;
  if (encontrado)
    matriz[i-1][j-1] = 'O';
  else
    {
      printf ("Tablero completo.\n");
      exit (0);
    }
}

void mostrar_matriz (void)
{
  register int i;

  printf ("\n    1   2   3");
  printf ("\n  +-----------+");
  for (i = 0; i < 3; i++)
    {
      printf ("\n%d ¦ %c ¦ %c ¦ %c ¦", i+1, matriz[i][0], matriz[i][1], matriz[i][2]);

      if (i != 2)
        printf ("\n  +---+---+---¦");
    }
  printf ("\n  +-----------+");
}

char comprobar (void)
{
  register int t;

  /* comprobar filas */
  for (t = 0; t < 3; t++)
    if (matriz[t][0] == matriz[t][1] && matriz[t][1] == matriz[t][2])
      return matriz[t][0];
```

```
  /* comprobar columnas */
  for (t = 0; t < 3; t++)
    if (matriz[0][t] == matriz[1][t] && matriz[1][t] == matriz[2][t])
      return matriz[0][t];

  /* comprobar diagonal principal */
  if (matriz[0][0] == matriz[1][1] && matriz[1][1] == matriz[2][2])
    return matriz[0][0];

  /* comprobar diagonal inversa */
  if (matriz[0][2] == matriz[1][1] && matriz[1][1] == matriz[2][0])
    return matriz[0][2];

  return ESPACIO;
}
```

Operadores especiales:

- De resolución de visibilidad (**::**) Permite acceder a una variable global oculta en una variable local del mismo nombre.
- De dirección e indirección:
  ***expresión**: devuelve la dirección de memoria donde está almacenada
  &variable: devuelve el dato almacenado en la dirección de memoria

Clase 14: PUNTEROS

Permite acceso directo al mapa de memoria en lugar de utilizar variables.
Un puntero contiene una dirección de memoria. Cuando una variable contiene la dirección de otra variable se dice que la primera variable apunta a la segunda.
La forma general para declarar una variable puntero es:

tipo *nombre;

donde tipo es cualquier tipo válido de C (también llamado tipo base) y nombre es el nombre de la variable puntero.
Existen dos operadores especiales de punteros: & y *.
Estos dos operados son monarios y no tienen nada que ver con la multiplicación (*) y el and bits (&).

OPERADOR &
& es un operador monario que devuelve la dirección de memoria de su operando.

Ejemplo:

```
#include <stdio.h>
 void main (void)
 {
   int x = 10;
   printf (" x = %d\n &x = %p\n",  x, &x) ;
 }
```

Salida del ejemplo:

```
x = 10
&x = 8FBC:0FFE
```

OPERADOR *
El operador * es el complemento de &.
Es un operador monario que devuelve el valor de la variable localizada en la dirección que le sigue.

Notación numérica: Recuerda que para valores en hexadecimal (hex) se marcan con el prefijo: "**0x**" y los valores octales como prefijo el 0.

Ejemplo 1:

```
#include <stdio.h>
void main (void)
{
  int x = 10;
  printf ("  x = %d\n", x);
  printf ("  *&x = %d", *&x);
}
```

Salida de ejemplo 1:

```
x = 10
*&x = 10
```


VISUALIZADOR DE MEMORIA

Estos programas visualizan 256 bytes de memoria a partir de la dirección de comienzo especificada.

```
#include <stdio.h>          /* printf (), scanf () */
#include <conio.h>          /* getch () */

main (void)
{
 register int i;
 unsigned char ch, *p;

 printf ("VISUALIZAR MEMORIA.\n\n");

 printf ("Dirección de comienzo (en hex): ");
 scanf ("%p%**c**", **&p**);

 printf ("\n%p: ", p); /* imprime dirección */
 for (i = 1; i <= 256; i++)
  {
    ch = *p;
    printf ("%02x ", ch); // visualiza en hexadec.
    p++;
    if (! (i % 16)) // cada 16 bytes salta de línea
     {
       printf ("\n");
       if (i != 256)
         printf ("%p: ", p); /* imprime dirección */
     }
  }
 getch ();
}
```

```
#include <stdio.h>          OK
#include <iostream.h> /para cout

int main(void)
{
cout <<"Uso de punteros\n";
char str1[]="programando";
//puntero del string:
char *pstr1=str1;

cout << str1 <<endl;
while (*pstr1)
{
cout <<" *pstr1="<< *pstr1++
<< endl;
}

getchar();
}
```

Clase 15: CLASE 17: ESTRUCTURAS y CLASES

Una estructura se declara con la palabra ***struct*** seguida de una lista de declaraciones encerradas entre llaves.
La forma general de definición de un estructura es >
donde tanto los nombres *nombre_tipo_estructura* como *nombres_variables_estructura* pueden omitirse.

```
struct nombre_tipo_estructura
 {
   tipo_1 nombre_variable_1;
   tipo_2 nombre_variable_2;
   ...
   tipo_n nombre_variable_n;
 } nombres_variables_estructura;
```

Ejemplo:

```
struct emplados
{
  char nombre[25];
  int edad;
  int sueldo;
};
```

```
struct emplados
{
  char nombre[25];
  int edad;
  int sueldo;
} empleado;
```

En el segundo caso, se define la estructura empleados y se declara una variable llamada empleado escrita al final antes del ;
Mejor usar: ***struct emplados empleado*** donde empleado es una variable del tipo empleados.
Luego se podrá asignar valores como: empleado.edad=32 para un valor entero;
o empleado.nombre [2] = 'X'; para un array.

Ejemplo Struct y Class: Es como crear una tabla de datos estructurados. Un array equivale a un tipo de struct.
Con una estructura puedes crear un tipo de dato nuevo, en este caso, vamos a declarar una variable coo de tipo Coordenadas, la cual puede almacenar 3 valores enteros: x, y, z que son los "datos miembros" de la estructura. (ver →)
Para manipular estos datos, (asignarles un valor inicial, cargarlos, mostrarlos, etc.), se pueden escribir funciones globales en el programa.
Por Ejemplo: void Carga(void) o void Muestra(void)

```
struct Coordenadas
{
   int x;
   int y;
   int z;
}
```

Ejemplo con Struct

```
#include <iostream>            // en el C++ se escribía: #include
<iostream.h>
using namespace std ;          // <-  y no se usaba esta línea


struct Coordenadas                          //Tested Dev-C++
{
   int x,y,z;

void Cargar(void)       //Función que carga los datos.
   {
   x=8;
   y=9;
   z=10;
   }
void Mostrar(void)       //Función que muestra los datos.
   {
   cout << x <<endl;
   cout << y <<endl;
   cout << z <<endl;
   }
};

int main(void)
{
   struct Coordenadas coo;    //Define variable (coo)
                              // de tipo Coordenadas
   coo.Cargar();             //Llamadas a las funciones de coo.
   coo.Mostrar();
  getchar ();
}
```

Ejemplo con Class

```
#include <iostream>
using namespace std ;

class Coordenadas
{
   int x,y,z;
public:
   void Cargar(void)
   {
      x=8;
      y=9;
      z=10;
   }
   void Mostrar(void)
   {
      cout << x <<endl;
      cout << y <<endl;
      cout << z <<endl;
   }
 };
void main(void)
{
   Coordenadas coo;
   coo.Cargar();
   coo.Mostrar();
}
```

En el ejemplo izquierdo: La estructura, además de definir sus datos, también define las funciones para manipularlos Carga(void) o Muestra(void)

Struct es válido para C y Class para C++.
En el ejemplo derecho: En lugar de struct se pone class, luego se agrega la etiqueta public, antes de definir las funciones miembros, ya que para una estructura los datos miembros son públicos, pero en una clase, por defecto, son privados y sólo las funciones públicas pueden tener acceso a los privados.
La otra diferencia es en el momento de definir(*) la variable coo, no hace falta especificar la palabra class así como se hizo con struct.
Utilizando clases, ya no se habla de "definir" una variable de una clase en particular, sino que se crea una "instancia" o un objeto de dicha clase.
Cuando el objeto es más complejo, o preocupa la protección de los datos miembros, es mejor usar clases que estructuras.
La Programación Orientada a Objetos, consta de objetos, y una clase, define o es como la "plantilla" sobre la cual se construyen los objetos.

Clase 16: MANEJO DE FICHEROS

1. En C, cualquier cosa externa de la que podemos leer o escribir datos es un fichero.
2. El programador escribe (o lee) datos en estos ficheros a través de los flujos de cada fichero. De esta forma el programador escribe (lee) los datos de la misma forma en todos los tipos de ficheros independientemente del tipo de fichero que sea.
3. Aunque conceptualmente todos los flujos son iguales, en realidad hay dos tipos: flujos de texto y flujos binarios.
4. Hay tres flujos de texto predefinidos que se abren automática mente al principio del programa: ***stdin***, ***stdout*** y ***stderr***. Estos tres flujos se cierran automáticamente al final del programa.

RESUMEN DE LOS PASOS PARA MANIPULAR UN FICHERO

1) Declarar un puntero de fichero.
   *FILE *pf;*
2) Abrirlo el fichero.
   *if ((pf = fopen ("*nombre_fichero", "*modo_apertura")) == NULL)*
   error ();
   else
   / … */*
3) Realizar las operaciones deseadas con el fichero.
4) Cerrar el fichero.
   If (fclose (pf) != 0)
   error ();
   else
   / … */*

EJEMPLO

```
#include <stdio.h> /* funciones de E/S */
#include <stdlib.h> /* exit () */
#define NOMFICH "CON"/*nombre del fich.*/
#define error(msj,nomfich) { printf
  (msj, nomfich); exit (1); getch (); }

void main (void)
{
  FILE *pf; /*puntero a estructura FILE*/
  char *phola, *hola = "hola";

  if ((pf = fopen (NOMFICH,"r")) == NULL)
    error ("Error al intentar abrir el "
          "fichero %s.", NOMFICH);
  for (phola = hola; *phola; phola++)
    putc (*phola, pf);
  if (fclose (pf) == EOF)
    error ("Error al intentar cerrar el "
          "fichero %s.", NOMFICH);
  exit (0);
}
```

Fopen: Para abrir un fichero y asociar a él un stream, usar: File *fopen (char *fname, char *mode);

La función fopen usa la librería stdio y el *nombre_fichero* a abrir debe ser de puntero
El modo de apertura es el mostrado en la lista.
Si la aperura es válida, devuelve un puntero de archivo.

Fclose: Cierra un fichero:
int fclose(FILE *fp);
La función fclose cierra el archivo asociado a al archivo de puntero fp obtenido con fopen y devuelve 0 si fue corecto o EOF si hubo un error.

int fgetc(FILE *fp);
int fputc(int ch, FILE *fp);

getc() function reads the next byte from the file and returns its as an integer and if error occurs returns EOF

Modo apertura fopen	significado
r	Open a text file for reading
w	Create a text file for writing
a	Append to a text file
rb	Open a binary file for reading
wb	Open a binary file for writing
ab	Append to a binary file
r+	Open a text file for read/write
w+	Create a text file for read/write
a+	Append or create a text file for read/write
r+b	Open a binary file for read/write
w+b	Create a binary file for read/write
a+b	Append a binary file for read/write

FUNCIONES

Es una parte del código independiente del principal. Se pueden pasar o no datos.

Declaración de una función:

- Por *Declaración explícita*: Mediante el prototipo de la forma:
  tipodelvalorderetorno nombredelafuncion (listadeparametros)
- Por *Definición previa*: Mediante prototipos después de ***#Include*** o ***#Define***.

Llamada a una función:

Se escribe su nombre seguida entre paréntesis de la lista de argumentos separados por comas:

Void mifuncion(int i, double a) (void no retorna)

También pueden asignarse valores por defecto a los parámetros:

Double modulo (doble x[ ], int n = 3)

Puede llamarse a una function en una expresión.

Ejemplo:

- La función devuelve el valor absoluto

```
Double valor_abs (double x)
{
    If  (x<0.0 )  Return –x ;
    else return x;
}
```

- Declaración y llamada a la función:

```
Double valor_abs (double);    //  -> Declaración
Void main (void)
{
    Double z, y;
    y=-30.8
    z=valor_abs(y) + y*y        // -> Llamada en una expresión
}
```

Función main con argumentos:

Al teclear el ejecutable, se pueden pasar argumentos escribiendo palabras separadas:

Int main (argc, char *argv[])

argc contiene número de palabras a continuación y ***argv*** los punteros de caracteres

Uso del Void en las funciones

Void indica la ausencia de valor enviado o devuelto. .

Ejemplo si una función no devuelve ningún valor:
Equivale a un procedure en Delphi-Pascal

```
void hello(char *name)
{   printf("Hello, %s.",name);
}
```

Ejemplo si no se pasan parámetros a la función.

```
int init(void)
{  return 1;
}
```


Ejemplo Menú con Switch

```
#include<stdio.h>
#include<conio.h>
#include <math.h>  //para funcion pow y raiz

/*declaraciones públicas antes de la función main*/
void cuadrado(double); //-> declaro función cuadrado sin retornar nada (void)
void raiz(double);     //-> declaro función raíz sin retornnar nada (void)
float num; //-> Variable pública num por eso declaramos aquí

/*inicio de la función main (programa principal)*/
void main()
{
int op;  //-> variable para escoger opción
// clrscr(); //-> Clearscreen no va en Dev C++ sí en Borland C++
printf("BIENVENIDO AL PROGRAMA CUADRATICAS. Escribe un numero: ");
scanf("%f",&num);
printf("\n\n  Elile la funcion: ");
printf("\n\n  1. CUADRADO");
printf("\n\n  2. RAIZ\n");
scanf("%d",&op); //espera opción y la guarda en la variable op
switch(op) //-> equivale a función DO CASE en Delphi
{
  case 1:
    cuadrado(num);  //llama al procedimiento cuadrado
    break; //-> devuelve a principal
  case 2:
          raiz(num);     //llama al procedimiento raiz
    break; //-> devuelve a principal
}
getch();
}

/*inicio de las funciones cuadrado y raíz después de la principal main*/
void cuadrado(double x)
{
double cuad;
cuad =pow(x,2); //variable cuad guarda pow que eleva x a la potencia 2
printf("\n El cuadrado de %g es %g",x,cuad);
}

void raiz(double x)
{
double ra;
ra =sqrt(x); //la variable ra contendrá la raíz de x
printf("\n La raiz de %g es %g",x,ra);
//-> ¿qué pasará si el num es negativo?
}
```

Imprime La Diagonal De Una Matriz Definida Por El Usuario (Ejemplo Array)
(Con Un Límite De 5 Por 5) Utilizando La Funcion "**Gotoxy**"

```
#include"iostream.h"     //Tested by Ofimega in Borland C++
#include"stdlib.h"
#include"conio.h"
#include"stdio.h"
#define a 5

void main()
{
      int matriz[a][a],r,colu=1;
      char op;
      do
      {
      Do
      {
      clrscr();
      cout<<"IMPRIME LA DIAGONAL PRINCIAL DE UNA MATRIZ"<<endl<<endl;
      cout<<"Introduzca el tamaño de la matriz solo de 1 hasta el 5: ";
      cin>>r;
      }
      while(r>5);
      for(int i=0;i<r;i++)//inicia en 0 el indice de columna de matriz
      {
            cout<<"\n";
            for(int x=0;x<r;x++)//inicia indice fila de matriz
            {
                  cout<<"Introduzca valor["<<i<<"]"<<"["<<x<<"] = ";
                  cin>>matriz[i][x];
            }
      }
      clrscr();
      cout<<"IMPRESION DE MATRIZ: ";
      for(int i=0;i<1;i++)
      {
            for(int x=0;x<r;x++)
            {

                  gotoxy(colu,x+x+3);  //no usar en ventanas de windows
                  cout<<matriz[i+x][x]<<"  ";
                  colu+=2;
            }
      }
      cout<<endl<<endl<<endl<<"REPETIR ESTE PROCESO? S/N";
      cin>>op;
      }
      while(op=='S' || op=='s');

      getch();

}
```

EJEMPLO:

MATRIZ
1 6 11 16 21
2 7 12 17 22
3 8 13 18 23
4 9 14 19 24
5 10 15 20 25

SOLAMENTE IMPRIME LO SIGUIENTE:

1
--7
----13
-------19
----------25

ALMACENA LOS DATOS DE UN NUMERO DEFINIDO DE ALUMNOS HASTA 10 UTILIZANDO UNA ESTRUCTURA DE REGISTRO Y AL FINAL DICE CUANTOS APROBADOS Y REPROBADOS FUERON UTILIZANDO ACUMULADORES (Ejemplo Struct)

```
#include"stdio.h"     //Tested by Ofimega
#include"conio.h"
#include"iostream.h"
void main()
{
      int apro=0,rep=0;
      int cont=0,n;
      char op;
      clrscr();

      struct agenda
      {
            char nombre[40];
            char mat[10];
            int tel;
            float cal;
            float pro;
      }a[10];
      do
      {
            clrscr();
            float acal=0;
            cout<<"\t\tREGISTRO DE ALUMNOS Y PROMEDIOS"<<endl<<endl;
            cout<<"INTRODUZCA EL NOMBRE: ";
            gets(a[cont].nombre);
            cout<<endl<<"INTRODUZCA SU NO. TELEFONICO: ";
            cin>>a[cont].tel;
            cout<<endl<<"INTRODUZCA LA MATRICULA DEL ALUMNO: ";
            gets(a[cont].mat);
            for(int c=0;c<3;c++)
            {
                  cout<<endl<<"\tINTRODUZCA CALIFICACION "<<c+1<<": ";
                  cin>>a[cont].cal;
                  cout<<endl<<endl;
                  acal+=a[cont].cal;
            }
            a[cont].pro=acal/3;//Aqui se calcula el promedio conforme al acumulador de calificaciones
"acal"
            if(a[cont].pro<6)
            {
                  rep+=1;
            }
            if(a[cont].pro>=6)
            {
                  apro+=1;
            }
            cout<<endl<<endl<<"¨QUIERE ALMACENAR OTRO REGISTRO? S/N";
            cin>>op;
            if(op=='S' || op=='s')
            {
                  cont+=1;
            }
            if(op=='N' || op=='n')
            {
                  cont=cont+10;
```

```
            }
        }
        while(cont<10);
        clrscr();
        cont-=10;
        for(int i=0;i<=cont;i++)
        {
                cout<<"REGISTRO NUMERO: "<<i+1<<endl<<endl;
                cout<<"El nombre es: "<<a[i].nombre<<endl<<endl;
                cout<<"El telefono es: "<<a[i].tel<<endl<<endl;
                cout<<"No. de matricula: "<<a[i].mat<<endl<<endl;
                cout<<"Su promedio es: "<<a[i].pro<<endl<<endl<<endl<<endl;
                getch();
        }
        clrscr();
        cout<<"\tEL NUMERO DE APROBADOS ES DE---------"<<apro<<endl;
        cout<<"\tEL NUMERO DE REPROBADOS ES DE--------"<<rep;
        getch();
}
```

Funciones de texto:

Equivalencia entre punteros y vectores (arrays) para buscar cadenas:

Función que recorre para copiar la cadena t sobre la cadena s usando arrays [ ]

```
strcpy(char [s], char[t])
{
     int i = 0;
     while((s[i]=t[i])!='\0')
     i++;
}
```

Función que recorre para copiar la cadena ***t*** sobre la cadena ***s*** usando punteros *

```
strcpy(char *s, char *t)
{
   int i = 0;
   while(*s++ = *t++)!='\0'
 ;
}
```


Ejercicios básicos de programación en Dev C++

//---ejemplo de printf

```
#include <stdio.h>          //son como los diccionarios que va a emplear
int main()                  // programa principal
{
  printf("Hola mundo");
}
```

//---ejemplo de printf con salto de linea y pausa con GETCH de la librería conio

```
#include <stdio.h>                  //son como los diccionarios que va a emplear
#include <conio.h>                   //necesaria para que entienda el GETCH
int main()                           // programa principal
{
  printf("Hola mundo\n");       //Poner \n dentro de las comillas para que haga salto de línea
  getch();                           //hacer pausa
}
```

//---ejemplo de scanf

```
#include <stdio.h>            //son como los diccionarios que va a emplear
#include <conio.h>
int main()                // programa principal
{
  int edad;                        //declaramos la variable edad del tipo numerica entera
  printf("¿Cuantos años tienes?\n");
  scanf("%d",&edad); //sintaxis:%d=formato numerico y &edad=argumento que contiene a la variable
  getch(); //hacer pausa
}
```

//----ejemplo de scanf y bifurcación IF

```
#include <stdio.h>                          //es como el vocabulario
#include <conio.h>
int main()                                                  // programa principal
{
  int edad;                                 //declaramos la variable edad del tipo numerica entera
  printf("¿Cuantos años tienes?\n");
  scanf("%d",&edad); //sintaxis:%d=formato numerico y &edad=argumento que contiene a la variable
  if (edad>18)                          // si es mayor a 18…
  printf("mayor de edad\n");
  else                                  // y si no....
  printf("menor de edad\n");
  getch();                               // hacer pausa
}
```

//----ejemplo de pedir nombre

```
#include <stdio.h> //es como el vocabulario
#include <conio.h>
int main()  // programa principal
{
  int edad; //declaramos la variable edad del tipo numerica entera
  char nombre[40];
  printf("Hola, ¿Cómo te llamas?\n");
  scanf("%s",&nombre); //sintaxis:%d=formato numerico y &edad=argumento que contiene a la
variable
  printf("Pues encantado de conocerte %s\n",nombre);
  getch(); //hacer pausa
}
```