Tema 1. Fundamentos de C.

Diseño de Algoritmos.

E.U. Politécnica
Departamento Lenguajes y Ciencias de la Computación.
Universidad de Málaga
José Luis Leiva Olivencia.
Despacho: I-326D (Edificio E.U.P)/ 3.2.41 (Teatinos-E.T.S.I.I.)

1. Estructura de un programa.

```
declaraciones globales

    main( ) {
        variables locales
        bloque
    }

    funcion1( ) {
        variables locales
        bloque
```

1.1. Comentarios.

Para poner comentarios en un programa escrito en **C** usamos los símbolos **/*** y ***/**:

/* Este es un ejemplo de comentario */

/* Un comentario también puede
estar escrito en varias líneas */

El símbolo **/*** se coloca al principio del comentario y el símbolo ***/** al final.

1.2. Palabras clave.

char	**int**	**float**	**double**
else	**do**	**while**	**for**
switch	**short**	**long**	**extern**
static	**default**	**continue**	**break**
register	**sizeof**	**typedef**	**if**

1.3. Identificadores.

Un identificador es el nombre que damos a las variables y funciones. Está formado por una secuencia de letras y dígitos, aunque también acepta el caracter de subrayado **_**. Por contra no acepta los acentos ni la **ñ/Ñ**.

Válidos	**No válidos**
_num	1num
var1	número2
fecha_nac	año_nac

2. La función main.

main(int argc, char *argv[])

El parámetro argc recoge automáticamente el número de items presente en la llamada desde el DOS al programa ejecutable. Así, una llamada a 'pepe.exe' del tipo:

Pepe archivo1 archivo2

hará que argc tome 3 como valor.

El parámetro argv[] es un array de punteros de tipo char,que apunta a los items o cadenas de caracteres, recibidas desde el exterior.

En este caso, las cadenas apuntadas por cada puntero de argv[] son:

argv[0]	"pepe"
argv[1]	"archivo1"
argv[2]	"archivo2"
argv[3]	puntero NULL

3. Tipos de datos.

TIPO	Tamaño	Rango de valores
char	1 byte	-128 a 127
int	2 bytes	-32768 a 32767
float	4 bytes	3'4 E-38 a 3'4 E+38
double	8 bytes	1'7 E-308 a 1'7 E+308

3. Tipos de datos (II).

- Calificadores de tipo:
  - signed (lleva sigono)
  - unsigned (no lleva signo)
  - short (formato corto) Utilizado por defecto.
  - long (formato largo)
    - signed long int=long int=long
    - unsigned long (4 bytes)

4. Declaración de funciones.

Su sintaxis es:

tipo_función nombre_función (tipo y nombre de argumentos)
{
bloque de sentencias
}

tipo_función: puede ser de cualquier tipo de los que conocemos. El valor devuelto por la función será de este tipo. Por defecto, es decir, si no indicamos el tipo, la función devolverá un valor de tipo entero (**int**). Si no queremos que retorne ningún valor deberemos indicar el tipo vacío (**void**).

Tiempo de vida de los datos.

Según el lugar donde son declaradas puede haber dos tipos de variables.

Globales: las variables permanecen activas durante todo el programa. Se crean al iniciarse éste y se destruyen de la memoria al finalizar. Pueden ser utilizadas en cualquier función.

Locales: las variables son creadas cuando el programa llega a la función en la que están definidas. Al finalizar la función desaparecen de la memoria.

Si dos variables, una global y una local, tienen el mismo nombre, la local prevalecerá sobre la global dentro de la función en que ha sido declarada.

Declaración de las funciones.

```
/* Declaración de funciones. */

#include <stdio.h>

void funcion(void); /* prototipo */
int num=5; /* variable global */
main() /* Escribe dos números */
{
        int num=10; /* variable local */
        printf("%d\n",num);
        funcion(); /* llamada */
}

void funcion(void)
{
        printf("%d\n",num);
}
```


Paso de parámetros a una función.

```
#include <stdio.h>

int suma(int,int); /* prototipo */
main() /* Realiza una suma */
{
        int a=10,b=25,t;
        t=suma(a,b); /* guardamos el valor */
        printf("%d=%d",suma(a,b),t);
        suma(a,b); /* el valor se pierde */
}

int suma(int a,int b)
{
        return (a+b);
}
```

5. Declaración de variables.

[calificador] <tipo> <nombre1>,<nombre2>=<valor>,<nombre3>=<valor>,<nombre4>

```
#include <stdio.h>

main() /* Suma dos valores */
{
        int num1=4,num2,num3=6;
        printf("El valor de num1 es %d",num1);
        printf("\nEl valor de num3 es %d",num3);
        num2=num1+num3;
        printf("\nnum1 + num3 = %d",num2);
}
```


Constantes.

```
#include <stdio.h>
#define pi 3.1416
#define escribe printf
main() /* Calcula el perímetro */
{
        int r;
        escribe("Introduce el radio: ");
        scanf("%d",&r);
        escribe("El perímetro es: %f",2*pi*r);
}
```

6. Sentencias de control.

```
if (condición) sentencia1;
        else sentencia2;
```


Sentencias de control(II)

```
switch (variable){
        case contenido_variable1:
                sentencias;
                break;
        case contenido_variable2:
                sentencias;
                break;
        default:
                sentencias;
    }
```

Cada case puede incluir una o más sentencias sin necesidad de ir entre llaves, ya que se ejecutan todas hasta que se encuentra la sentencia **BREAK**. La variable evaluada sólo puede ser de tipo **entero** o **caracter**. **default** ejecutará las sentencias que incluya, en caso de que la opción escogida no exista.

Sentencias de control (III)

```
while (condición) sentencia;
        do{
                sentencia1;
                sentencia2;
        }while (condición);
for (inicialización;condición;incremento){
                sentencia1;
                sentencia2;
        }
```


7. Directiva #define.

```
#define MÁXIMO 100
#define MÍNIMO 10
#define PI 3.141592

main()
{
char cadena[MÁXIMO];
char cad[MINIMO];
...............
```

```
#define cuadrado(x) x*x
#define cubo(x) cuadrado(x)*x
#define ABS(X) ( (x>=0) ? x:0-x)
#define producto(x,y) x*y
#define IMPRIME(X) printf("\n"#x)
```

8. Ficheros.

```
FILE *puntero;
        puntero = fopen ( nombre del fichero, "modo de apertura" );
        puntero=fopen("DATOS.DAT","r");
        puntero=fopen("C:\\TXT\\SALUDO.TXT","w");
```

Modo texto

- **w** crea un fichero de escritura. Si ya existe lo crea de nuevo.
- **w+** crea un fichero de lectura y escritura. Si ya existe lo crea de nuevo.
- **a** abre o crea un fichero para añadir datos al final del mismo.
- **a+** abre o crea un fichero para leer y añadir datos al final del mismo.
- **r** abre un fichero de lectura.
- **r+** abre un fichero de lectura y escritura.

Ficheros (II).

Modo binario

- **wb** crea un fichero de escritura. Si ya existe lo crea de nuevo.
- **w+b** crea un fichero de lectura y escritura. Si ya existe lo crea de nuevo.
- **ab** abre o crea un fichero para añadir datos al final del mismo.
- **a+b** abre o crea un fichero para leer y añadir datos al final del mismo.
- **rb** abre un fichero de lectura.
- **r+b** abre un fichero de lectura y escritura.

```
FILE *pf;
        pf=fopen("datos.txt","r");
        if (pf == NULL) printf("Error al abrir el fichero");
```

Ficheros (III)

```
FILE *pf;
        pf=fopen("AGENDA.DAT","rb");
        if ( pf == NULL ) printf ("Error al abrir el fichero");
        else fclose(pf);
```


Ficheros (IV)

Escribir y leer carácter:

fputc(variable_carácter, puntero_fichero)
variable_carácter=fgetc(puntero_fichero)

Escribir y leer número entero:

putw(variable_entera, puntero_fichero)
variable_entera=getw(puntero_fichero)

Ficheros (V)

Escribir y leer cadenas:

fputs(variable_array, puntero_fichero);
fgets(variable_array, variable_entera, puntero_fichero)

Escribir y leer con formato:

fprintf(puntero_fichero, formato, argumentos);
fscanf(puntero_fichero, formato, argumentos)

Ficheros (VI)

Escribir y leer estructuras:

fwrite(*buffer, tamaño, nº de veces, puntero_fichero)
fread(*buffer, tamaño, nº de veces, puntero_fichero)

Otras funciones:
rewind(puntero_fichero);
rename(nombre1, nombre2);
feof(puntero_fichero);

Apéndice. Estructuras.

```
struct trabajador
        {
                char nombre[20];
                char apellidos[40];
                int edad;
                char puesto[10];
        };
        struct trabajador fijo, temporal;
```


Estructuras (II)

```
void visualizar(struct trabajador);
main() /* Rellenar y visualizar */
{
        struct trabajador fijo;
        printf("Nombre: ");
        scanf("%s",fijo.nombre);
        printf("\nApellidos: ");
        scanf("%s",fijo.apellidos);
        printf("\nEdad: ");
        scanf("%d",&fijo.edad);
        printf("\nPuesto: ");
        scanf("%s",fijo.puesto);
        visualizar(fijo);
}

void visualizar(struct trabajador datos)
{
        printf("Nombre: %s",datos.nombre);
        printf("\nApellidos: %s",datos.apellidos);
        printf("\nEdad: %d",datos.edad);
        printf("\nPuesto: %s",datos.puesto);
}
```

Typedef

```
typedef int entero;
entero a, b=3;

typedef struct trabajador datos;
datos fijo, temporal;
```


Ejemplo.

```
#include <stdio.h>
#include <conio.h>
struct registro
{
char nombre[40];
char edad[3];
}
main()
{

int op;
char opcion[5];
```

```
do
{
clrscr();
printf("1. alta\n");
printf("2. baja\n");
printf("3. consulta\n");
printf("4. listado\n");
printf("5. modificación\n");
printf("6. salir\n");
gets(opcion);
op=atoi(opcion);
if (op==1)
  altas();
if (op==4)
  listado();

}
while (op!=6);
```

Ejemplo.

```
altas()
{
char fichero[40];
FILE *f;
struct registro persona;
clrscr();
strcpy(fichero,"DATOS.DAT");
f=fopen(fichero,"a");
if (f==NULL)
   f=fopen(fichero,"w");

printf("Nombre: ");
gets(persona.nombre);
printf("Edad: ");
gets(persona.edad);
fwrite(&persona,sizeof(persona),1,f);
fclose(f);
}
```


Ejemplo.

```
listado()
{
char fichero[40];
FILE *f;
struct registro persona;

clrscr();
strcpy(fichero,"DATOS.DAT");
f=fopen(fichero,"r");
while (!feof(f))
{
clrscr();
fread(&persona,sizeof(persona),1,f);
printf("Nombre: %s",persona.nombre);
printf("Edad: %s",persona.edad);
getche();
}
fclose(f);}
```