LENGUAJES DE PROGRAMACIÓN II

C

Forma general de un programa en C

Declaraciones globales

```
main( )
{
  variables locales
  sentencias
}

f1( )
{
 .........
}
...
...
fn ( )
{
 .........
}
```

Nombre de indentificadores

Son los nombres usados para referirse a las variables, funciones, etiquetas y otros objetos definidos por el usuario.

La longitud de un identificador en Turbo C puede variar entre 1 y 32 caráteres. El primer carácter debe ser una letra o un símbolo de subrayado, los carácteres siguientes pueden ser letras, números o símbolos de subrayado.

Correcto -----> cont, cuenta23, balance_total
Incorrecto -----> 1cont, hola!, balance...total

En C las mayúsculas y las minúsculas se tratan como distintas.

Tipos de datos

Existen cinco tipos de datos atómicos en C:

Tipo	bits	rango
char	8	0 a 255
int	16	-32.768 a 32.767
float	32	3,4 E -38 a 3,4 E +38
double	64	1,7 E -308 a 1,7 E +308
void	0	sin valor

El **void** se usa para declarar funciones que no devuelven ningún valor o para declarar funciones sin parámetros.

Modificadores de tipos

signed
unsigned
long
short

Los modificadores **signed, unsigned, long** y **short** se pueden aplicar a los tipos base entero y carácter. Sin embargo, **long** también se puede aplicar a **double**.

Tipo	bits	Rango
char	8	-128 a 127
unsigned char	8	0 a 255
signed char	8	-128 a 127
int	16	-32.768 a 32.767
unsigned int	16	0 a 65.535
signed int	16	-32.768 a 32.767
short int	16	-32.768 a 32.767
unsigned short int	16	0 a 65.535
signed short int	16	-32.768 a 32.767
long int	32	-2147483648 a 2147483647
signed long int	32	-2147483648 a 2147483647
float	32	3,4 E -38 a 3,4 E +38
double	64	1,7 E -308 a 1,7 E +308
long double	64	1,7 E -308 a 1,7 E +308

Modificadores de acceso

Las variables de tipo **const** no pueden ser cambiadas durante la ejecución del programa. Por ejemplo,

const int a;

Declaración de variables

Todas las variables han de ser declaradas antes de ser usadas. Forma general:

```
tipo  lista_de_variables;          int i,j,l;
                                   short int si;
```

Existen tres sitios donde se pueden declarar variables: dentro de las funciones (variables locales), en la definición de parámetros de funciones (parámetros formales) y fuera de todas las funciones (variables globales).

Variables externas

Si una función situada en un fichero fuente desea utilizar una variable de este tipo declarada en otro fichero, la debe declarar (o mejor dicho referenciar) con la palabra **extern**.

Archivo 1

```
int x,y;
char ch;

main ( )
{
 x=120;
  . . . . . .
}
```

Archivo 2

```
extern int x,y;
extern char ch;

 void func1( )
{
  x=y/10;
  . . . . . .
}
```

Variable estáticas (static)

Tienen memoria asignada durante toda la ejecución del programa. Su valor es recordado incluso si la función donde está definida acaba y se vuelve a llamar más tarde. Ejemplo:

```
series (void)
{
  static int num;

  num=num+23;
  return (num);
}
```

Variables registro

El especificador register pide a Turbo C que mantenga el valor de una variable con ese especificador de forma que se permita el acceso más rápido a la misma. Para enteros y carácteres esto significa colocarla en un registro de la CPU.
Sólo se puede aplicar a variables locales y a los parámetros fomales de una función.
Son ideales para el control de bucles.

```
pot_ent (int m, register int e)
{
 register int temp;
 temp=1;
 for ( ; e; e--) temp *=m;
 return (temp);
}
```

Sentencias de asignación

Forma general: nombre_variable = expresion;

Abreviaturas en C

x=x+10 <----------> x+=10
x=x-10 <----------> x-=10

Conversión de tipos

Se da cuando se mezclan variables de un tipo con variables de otro tipo.
El valor de la derecha de la asignación se convierte al tipo del lado izquierdo.
Puede haber pérdida de los bits más significativos en un caso como: short = long

Inicialización de variables

Tipo nombre_variable = constante;

```
char c='a';
int primero=0;
float balance=123.23;
```

Todas las variables globales se inicializan a cero sino se especifica otro valor inicial. Las variables locales y register tendran valores desconocidos antes de que se lleve a cabo su primera asignación.

Constantes

Tipo dato	Ejemplo de constantes
char	'a' '\n' '9'
int	1 123 -234
float	123.23

Una constante de tipo cadena de carácteres está constituida por una secuencia de carácteres entre comillas dobles "Hola".

Carácteres con barra invertida

\n	Nueva línea
\t	Tabulación horizontal
\b	Espacio atras
\r	Retorno de carro
\f	Salto de página
\\	Barra invertida
\'	Comilla simple
\"	Comilla doble

Operadores

En C hay tres clases de operadores: aritméticos, relacionales y lógicos, y a nivel de bits.

Aritmeticos

-	resta
+	suma
*	producto
/	división
%	módulo (resto de la división entera)
--	decrementar
++	incrementar

```
x=10;                    x=10;
y=++x;                   y=x++;
y=11                     y=10
```

Relacionales

En C cierto es cualquier valor distinto de cero. Falso es cero.

>	mayor que
>=	mayor o igual que
<	menor que
<=	menor o igual que
==	igual
!=	distinto

Lógicos

&&	y
||	o
!	no

El operador ?

Exp 1 ? Exp 2 : Exp 3

Se evalua exp1 si es cierto se evalua exp2 y toma ese valor para la expresión. Si exp1 es falso evalua exp3 tomando su valor para la expresión.

Ejemplo: x=10:
y=x>9 ? 100 : 200 --------> y = 100

Los operadores de punteros & y *

& devuelve la dirección de memoria del operando.

Ejemplo: m=&cont; coloca en m la dirección de memoria de la variable cont
& (la dirección de)

***** devuelve el valor de la variable ubicada en la dirección que se especifica.

Ejemplo: q=*m; coloca el valor de cont en q. ***(en la dirección)**

Sizeof

Es un opeador monario que devuelve la longitud, en bytes, de la variable o del especificador de tipo al que precede.

Ejemplo: flat f;
printf ("%f",sizeof f); Mostrara 4
printf ("%d", sizeof (int)); Mostrara 2

El nombre del tipo debe ir entre paréntesis.

ESTRUCTURAS CONDICIONALES

If

```
if (expresion) {
  .....
  .....
}
else {
  .....
  .....
}
```

Switch

```
switch (variable) {
  case cte1 :
     .....
     .....
     break;
  case cte2 :
     .....
     .....
     break;
  .....
  .....
  default :
     .....
     .....
}
```

Switch sólo puede comprobar la igualdad.

BUCLES

For

```
for (inicialización; condición; incremento) sentencia
```

inicialización ----> asignación
condición ----> expresión relacional

Ejemplo: `for (x=1; x<=100; x++) printf ("%d",x);` Imprime los numeros del 1 al 100

While

```
while (condición)  sentencia;
```

Ejemplo: `while (c!='A')  c=getchar( );`

Do / While

Analiza la condició al final.

```
do {
   .....
   .....
} while (condicion);
```

Break

Tiene dos usos:

- para finalizar un case en una sentencia switch.
- para forzar la terminación inmediata de un bucle.

Exit

Para salir de un programa anticipadamente. Da lugar a la terminación inmediata del programa, forzando la vuelta al S.O. Usa el archivo de cabecera stdlib.h

Ejemplo:

```
#include <stdlib.h>

main (void)
{
 if (!tarjeta_color( )) exit(1);
 jugar( );
}
```

Continue

Hace comenzar la iteración siguiente del bucle, saltando así la secuencia de instrucciones comprendida entre el **continue** y el fin del bucle.

```
do {
   scanf("%d",&num);
   if (x<0) continue;
   printf("%d",x);
} while (x!=100);
```

Funciones

```
tipo  nombre_funcion  (lista de parametros)
{
  .....
  .....
}
```

tipo, especifica el tipo de valor que devuelve la sentencia return de la función.

Llamada por valor

Copia el valor de un argumento en el parámetro formal de la subrutina. Los cambios en los parámetros de la subrutina no afectan a las variables usadas en la llamada.

```
int  cuad (int x);

main ( )                                cuad (int x)
{                                       {
 int t=10;                                x=x*x;
 printf ("%d %d",cuad(t),t);                 return(x);
 return 0;                               }
}
```

Salida es << 100 10 >>

Llamada por referencia

Es posible causar una llamada por referencia pasando un puntero al argumento. Se pasa la dirección del argumento a la función, por tanto es posible cambiar el valor del argumento exterior de la función.

```
int x,y;                        inter (int *x,int *y)
inter (&x,&y);                   {
                                 int temp;
                                 temp=*x;
                                 *x=*y;
                                 *y=temp;
                                }
```

Arrays

Todos los arrays tienen el 0 como índice de su primer elemento.

char p [10]; array de carácteres que tiene 10 elementos, desde p[0] hasta p[9].

Para pasar arrays unidimensionales a funciones, en la llamada a la función se pone el nombre del array sin índice. Ejemplo:

```
main ( )
{
  int i[10];

  func1 (i);
}
```

Si una función recibe un array unidimensional, se puede declarar el parámetro formal como un puntero, como un array delimitado o como un array no delimitado.

```
func1 (int *x)      /puntero/
func1 (int x[10])   /array delimitado/
func1 (int x[ ])    /array no delimitado/
```

Inicialización de arrays

Forma general de inicialización de un array:

```
tipo  nombre_array [tamaño] = {lista de valores};
```

lista de valores, es una lista de constantes separadas por comas, cuyo tipo es compatible con el tipo del array. La primera constante se coloca en la primera posición del array, la segunda constante en la segunda posición y así sucesivamente.

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

Los arrays de carácteres que contienen cadenas permiten una inicialización de la forma:

```
char  nombre_array [tamaño]="cadena";
```

Se añade automáticamente el terminador nulo al final de la cadena.

Ejemplo:

`char  cad[5]="hola";` equivalentes `char  cad[5]={'h','o','l','a','\o'};`

Es posible que C calcule automáticamente las dimensiones de los arrays utilizando arrays indeterminados. Si en la inicialización no se especifica el tamaño el compilador crea un array suficientemente grande para contener todos los inicializadores presentes.

```
char  e1[ ]="error de lectura \n";
```

Cadenas

Aunque C no define un tipo cadena, estas se definen como un array de carácteres de cualquier longitud que termina en un carácter nulo ('\0').

Array que contenga 10 carácteres: `char  s[11];`

Una constante de cadena es una lista de carácteres encerrada entre dobles comillas.

Funciones de manejo de cadenas

Archivo de cabecera **string.h**

char ***strcpy** (char *s1, const char *s2); copia la cadena apuntada por s2 en la apuntada por s1. Devuelve s1.

char ***strcat** (char *s1, consta char *s2); concatena la cadena apuntada por s2 en la apuntada por s1, devuelve s1.

int **strlen** (const char *s1); devuelve la longitud de la cadena apuntada por s1.

int **strcmp** (const char *s1, const char *s2); compara s1 y s2, devuelve 0 si con iguales, mayor que cero si s1>s2 y menor que cero si s1<s2. Las comparaciones se hacen alfabéticamente.

Arrays Bidimensionales

Se declaran utilizando la siguiente forma general:

tipo nombre_array [tamaño 2ª dim] [tamaño 1ª dim];

Ejemplo -----> int d [10][20];

Cuando se utiliza un array bidimensional como argumento de una función realmente sólo se pasa un puntero al primer elemento, pero la función que recibe el array tiene que definir al menos la longitud de la primera dimensión para que el compilador sepa la longitud de cada fila.

Ejemplo: función que recibe un array bidimensional de dimensiones 5,10 se declara así:

```
func1 (int x[ ][10])
{
  .....................
}
```

Arrays y Punteros

Un nombre de array sin índice es un puntero al primer elemento del array.

Ejemplo: Estas sentencias son idénticas:

```
char  p[10];                    -  p
                                -  &p[0]
```

```
int *p, i[10];

p=i;                    ambas sentencias ponen el valor 100 en el sexto elemento de i.
i[5]=100;
*(p+5)=100;
```

Esto también se puede aplicar con los arrays de dos o más direcciones.

```
int a[10][10];

a=&a[0][0];
a[0][4]=*((*a)+4);
```

Memoria dinámica

Malloc (n) reserva una porción de memoria libre de n bytes y devuelve un puntero sobre el comienzo de dicho espacio.

Free (p) libera la memoria apuntada con el puntero p.

Ambas funciones utilizan el archivo de cabecera stdlib.h

Si no hay suficiente memoria libre para satisfacer la petición, malloc ( ) devuelve un nulo.

Ejemplo:

```
char  *p;
p=malloc(1000);
```

Estructuras

La forma general de una definición de estructura es:

```
struct etiqueta {
    tipo nombre_variable;
    tipo nombre_variable;
    .....
    .....
}  variables _de_estructura
```

Ejemplo:

```
struct dir {
   char                 nombre[30];
   char                 calle[40];
   char                 ciudad[20];
   char                 estado[3];
   unsigned long int  codigo;
}  info_dir;
```

A los elementos individuales de la estructura se hace referencia utilizando . (punto).

Ejemplo:

```
info_dir.codigo = 12345;
```

Forma general es: nombre_estructura.elemento

Una estructura puede inicializarse igual que los vectores:

```
struct  familia {
    char    apellido[10];
    char    nombrePadre[10];
    char    nombreMadre[10];
    int      numerohijos;
}  fam1={"Garcia","Juan","Maria",7};
```

Arrays de estructuras

Se define primero la estructura y luego se declara una variable array de dicho tipo.

Ejemplo:

```
struct  dir  info_dir [100];
```

Para acceder a una determinada estructura se indexa el nombre de la estructura:

```
info_dir [2].codigo = 12345;
```

Paso de estructuras a funciones

Cuando se utiliza una estructura como argumento de una función, se pasa la estructura íntegra mediante el uso del método estándar de llamada por valor.

Ejemplo:

```
struct tipo_estructura {
      int  a,b;
      char  c;
 };

 void  f1 (struct tipo_estructura  param);

 main ( )
 {
  struct tipo_estructura arg;
  arg.a = 1000;
  f1(arg);
  return 0;
 }

 void f1 (struct tipo_estructura  param)
 {
  printf ("%d",param.a);
 }
```

Punteros a estructuras

Declaración: struct dir * pruntero_dir;

Existen dos usos principales de los punteros a estructuras:

1) para pasar la dirección de una estructura a una función.
2) para crear listas enlazadas y otras estructuras de datos dinámicas.

Para encontrar la dirección de una variable de estructura se coloca & antes del nombre de la estructura.

Ejemplo:

```
struct  bal {
   float  balance;
   char  nombre[80];
} persona;

struct bal  *p;

p = &persona;        (coloca la dirección de la estructura persona en el puntero p)
```

No podemos usar el operador punto para acceder a un elemento de la estructura a través del puntero a la estructura. Debemos utilizar el operador flecha ->

```
p -> balance
```

Tipo enumerado

enum identificador {lista de constantes simbólicas};

Ejemplo: enum arcoiris {rojo, amarillo, verde, azul, blanco};
(realmente asigna rojo=0, amarillo=1, ...)

printf ("%d %d", rojo, verde); imprime 0 2 en pantalla

Podemos especificar el valor de uno o más símbolos utilizando un inicializador. Lo hacemos siguiendo el símbolo con un signo igual y un valor entero.

enum moneda {penique, niquel, diez_centavos, cuarto=100, medio_dolar, dolar};

Los valores son: penique 0, niquel 1, diez_centavos 2, cuarto 100, medio_dolar 101, dolar 102

(Libro pag. 167)

Punteros

int x=5, y=6;
int *px, *py;

px=py; copia el contenido de py sobre px, de modo que px apuntará al mismo objeto que apunta py.

*px=*py; copia el objeto apuntado por py a la dirección apuntada por px.

px=&x; px apunta a x.

py=0; hace que py apunte a nada (NULL).

px++; apunta al elemento siguiente sobre el que apuntaba inicialmente

Ejemplo:


```
char c;
char *pc=&c
```

Se puede sumar o restar enteros a y de punteros.

p1=p1+9; p1 apunta al noveno elemento del tipo p1 que está más allá del elemento al que apunta actualmente.

Punteros y arrays

```
char  cad[80], *p1;
p1 = cad                  p1 ha sido asignado a la dirección del primer elemento del array
                          cad.
```

Para acceder al quinto elemento de cad se escribe:

cad[4] o *(p1+4)

Arrays de punteros

Array de punteros a enteros:

int *x [10];

Para asignar la dirección de una variable entera llamada *var* al tercer elemento del array de punteros, se escribe:

x[2]=&var;

Para encontrar el valor de *var*:

*x[2]

Punteros a punteros

puntero ----------> variable Indirección simple

puntero ---------> puntero ---------> variable Indirección múltiple

float **balancenuevo;

balancenuevo no es un puntero a un número en coma flotante, sino un puntero a un puntero a float.

Ejemplo:

```
main(void)
{
 int x, *p, **q;

 x=10;
 p=&x;
 q=&p;
 printf("%d",**q);          /* imprime el valor de x */
 return 0;
}
```

E/S por consola

getche ( ) lee un carácter del teclado, espera hasta que se pulse una tecla y entonces devuelve su valor. El eco de la tecla pulsada aparece automáticamente en la pantalla. Requiere el archivo de cabecera **conio.h**

putcahr ( ) imprime un carácter en la pantalla.

Los prototipos son:

```
int getche (void);
int putchar (int c);
```

Ejemplo:

```
main ( )          /* cambio de mayúscula / minúscula */
{
 char car;
 do {
   car=getche( );
   if (islower(car)) putchar (toupper (car));
   else putchar (tolower (car));
 } while (car=!'.')
}
```

Hay dos variaciones de getche ( ) :

- Getchar ( ): función de entrada de carácteres definida por el ANSI C. El problema es que guarda en un buffer la entrada hasta que se pulsa la tecla INTRO.

- Getch ( ): trabaja igual que getche ( ) excepto que no muestra en la pantalla un eco del carácter introducido.

gets ( ) y puts ( )

Permiten leer y escribir cadenas de carácteres en la consola.

gets ( ) lee una cadena de carácteres introducida por el teclado y la situa en la dirección apuntada por su argumento de tipo puntero a carácter. Su prototipo es:

```
char * gets (char *cad);
```

Ejemplo:

```
main ( )
{
 char  cad[80];
 gets (cad);
 printf ("La longitud es %d", strlen (cad));
 return 0;
}
```

puts ( ) escribe su argumento de tipo cadena en la pantalla seguido de un carácter de salto de línea. Su prototipo es:

```
char * puts (const char *cad);
```

E/S por consola con formato

printf ( ) El prototipo de printf ( ) es:

```
int printf (const char  *cad_fmt, ...);
```

La cadena de formato consiste en dos tipos de elementos: carácteres que se mostrarán en pantalla y órdenes de formato que empiezan con un signo de porcentaje y va seguido por el código del formato.

%c	un único carácter
%d	decimal
%i	decimal
%e	notación científica
%f	decimal en coma flotante
%o	octal
%s	cadena de carácteres
%u	decimales sin signo
%x	hexadecimales
%%	imprime un signo %
%p	muestra un puntero

Las órdenes de formato pueden tener modificadores que especifiquen la longitud del campo, número de decimales y el ajuste a la izquierda.

Un entero situado entre % y el código de formato actua como un especificador de longitud mínima de campo. Si se quiere rellenar con ceros, se pone un 0 antes del especificador de longitud de campo.

%05	rellena con ceros un número con menos de 5 dígitos.
%10.4f	imprime un número de al menos diez carácteres con cuatro decimales.

Si se aplica a cadenas o enteros el número que sigue al punto especifica la longitud máxima del campo.

%5.7s	imprime una cadena de al menos cinco carácteres y no más de siete.

scanf ( ) Su prototipo es:

```
int scanf ( ) (const char  *cadena_fmt, ...);
```

Ejemplo:

```
scanf ("%d",&cuenta);
```