GUIA "C" y "PASCAL"

La presente guía tiene como objeto facilitar la traducción de un programa en C a Pascal y viceversa. Es fundamental conocer alguno de los dos lenguajes.
En especial es importante el paso de Pascal a C, pues este último lenguaje es ciertamente más complejo que Pascal, por lo que en general no se estudia en el primer curso universitario de casi cualquier escuela o facultad de informática, pero si es muy común estudiarlo desde el primer curso de telecomunicaciones, al menos en España. Pascal es un lenguaje idóneo para introducirse en la programación, debido a que es muy intuitivo y sencillo de aprender, a la vez que es menos oscuro que C. El lenguaje C es bastante más complicado de aprender y dominar, pero a la vez es más potente y flexible, especialmente en lo que se refiere al manejo de referencias o punteros. El manejo de los punteros en C es la parte del lenguaje más complicada de aprender y entender, aparte de que induce a cometer numerosos errores, en general pequeños despistes a la hora de manipularlos, pero que suelen resultar fatales en la ejecución del programa.

Por otra parte, C/C++ es uno de los lenguajes más utilizados en la actualidad. Los sistemas operativos Windows y Linux se desarrollan en C y ensamblador, y en general muchísimas aplicaciones, en especial videojuegos. También hay que citar numerosos entornos de programación en C/C++ como son Visual C++ y C++ Builder. Actualmente está en auge el lenguaje Java, un lenguaje derivado de C++. El lenguaje C++ es un superconjunto del lenguaje C, que incorpora la programación orientada a objetos.

En estas páginas no se indica ninguna referencia sobre C++, sin embargo, aquellos que tengan conocimientos de C y de programación orientada a objetos podrán trasladar con suma sencillez a C++ todos los algoritmos y estructuras de datos expuestos en estas páginas.

Ejemplo de dos programas equivalentes en C y Pascal

El programa siguiente pide dos números y después muestra su suma. Primero se muestra la codificación de ambos programas y después se procederá a la explicación de cada parte.

Programa en C:

```
/* Programa para multiplicar dos numeros */
#include <stdio.h>

/* Esta funcion devuelve la suma de los enteros a y b */
int sumar(int a, int b)
{
   return a + b;
}

/* Linea principal del programa */
```

```
int main(void)
{
   int a, b;
   printf("Introduce dos numeros enteros y pulsa enter: ");
   scanf("%d %d", &a, &b);
   printf("La suma de %d y %d es %d\n", a, b, sumar(a,b));
   return 0;
}
```

Programa en Pascal:

```
(* Programa para multiplicar dos numeros *)

program multiplicar_dos_numeros;

{ Esta funcion devuelve la suma de los enteros a y b }
function sumar(a, b : integer) : integer;
begin
     sumar := a + b
end;

{ Linea principal del programa }
var
   a, b : integer;

begin
     write('Introduce dos numeros enteros y pulsa enter: ');
     readln(a,b);
     writeln('La suma de ', a, ' y ', b, ' es ', sumar(a,b))
end.
```

Para empezar, los comentarios. En C se representan con `/* coments. */` y en Pascal con `(* coments. *)` o `{ coments. }`. En todos los casos se entiende por comentario todo lo que este encerrado entre los limitadores de comienzo y final, y los comentarios pueden ir en varias líneas. En C++ (y en muchos compiladores de C) se utiliza además el comentario de una sola línea:

```
.. codigo.. ; // comentario hasta fin de linea
```

La instrucción `#include <stdio.h>` de C es necesaria para poder utilizar los procedimientos de entrada/salida (stdio = standard input / output).
La declaración de variables se hace de forma claramente separada en Pascal, mediante la claúsula `Var`, y en C se hace dentro de cada procedimiento. En C se devuelve el valor de una función mediante la instrucción `return` (que además termina la ejecución de la función), y en Pascal empleando el nombre de dicha función asignado a un valor o expresión. Dicha asignación debe realizarse exclusivamente como última instrucción de la función. También se puede observar que la última línea del procedimiento principal (una función, que siempre será main) es la instrucción return, que devuelve el código de terminación del programa al sistema operativo.

Se observa que los bloques se señalan con `llaves: { ... }` en C y con `Begin ... End` en Pascal.

C es totalmente sensible a mayúsculas y minúsculas, Pascal no. En C *var* y *Var* son diferentes.

Tipos de datos

- Enteros

Rango	**C**	**Pascal**	**Tamaño (en bits)**
-128 .. 127	char	shortint	8
0 .. 255	unsigned char	byte	8
-32768 .. 32767	int, short int	integer	16
0 .. 65535	unsigned int	word	16
-2,147,483,648 .. 2,147,483,647	long int, long	LongInt	32
4,294,967,295	unsigned long	¿?	32
(-2^63) + 1 .. (2^63) - 1	¿?	comp	64

- Reales

Rango	**C**	**Tamaño (en bits)**
3.4 * (10^-38) .. 3.4 * (10^+38)	float	32
1.7 * (10^-308) .. 1.7 * (10^+308)	double	64
3.4 * (10^-4932) .. 1.1 * (10^+4932)	long double	80

Rango	**Pascal**	**Dígitos**	**Tamaño (en bits)**
2.9 * (10^-39) .. 1.7 * (10^+38)	real	11 - 12	48
1.5 * (10^-45) .. 3.4 * (10^+38)	single	7 - 8	32

5 * (10^-324) .. 1.7 * (10^+308)	double	15 - 16	64
3.4 * (10^-4932) .. 1.1 * (10^+4932)	extended	19 - 20	80

- Otros

C dispone de un tipo neutro o nulo, llamado `void`. Suele utilizarse especialmente para definir punteros que no tengan ningún tipo, o para definir funciones que no reciben o devuelven ningún valor.

- Booleanos

En C se representan mediante int o char, mediante las equivalencias siguientes:
Falso -> 0.
Cierto -> cualquier valor distinto de 0. En general se utiliza el 1.

En Pascal sí existen definidos una serie de tipos booleanos, que son los siguientes:
`Boolean` (8 bits)
`WordBool` (16 bits)
`LongBool` (32 bits)
`ByteBool` (8 bits)

`Boolean` es el que se utiliza normalmente.
Equivalencias:
Falso -> `False`
Cierto -> `True`

En Pascal además se cumplen las siguientes relaciones:

```
False < True
Ord(False) = 0
Ord(True) = 1
Succ(False) = True
Pred(True) = False
```

`Nota:` `Ord` devuelve el ordinal del valor pasado, y `Succ` y `Pred` devuelven el sucesor y el predecesor del valor pasado.

- Cadenas de caracteres

Ver apartado *Cadenas de caracteres*, más adelante en la guía.

- Ordinales, conjuntos y tipos enumerados.

- Arrays

En C:

```
tipo nombre[i1][i2]...
```
i1, i2, ... >= 1

Acceso: `nombre[a][b]...`
0 <= a < i1; 0 <= b < i2; etc.
Notar que debe ser estrictamente menor. *a* y *b* pueden ser expresiones aritméticas o booleanas.

Ejemplo:
```
int datos[10];
...
datos[0] = 1;
datos[9] = -datos[0];
datos[10] = 1; /* acceso incorrecto */
```

En Pascal:

```
nombre : array[i1..e1, i2..e2, ...] of tipo;
```
i1 <= e1; i2 <= e2; etc.
i1, e1, etc, deben ser tipos ordinales, es decir, tipos enteros, naturales o booleanos.

Acceso: `nombre[a, b, ...]`
`i1 <= a <= e1; i2 <= b <= e2; etc.`
a y *b* pueden ser expresiones aritméticas o booleanas.

Ejemplo:
```
datos : array[-1..1] of Integer;
...
datos[-1] := 1;
datos[1] := -datos[-1];
datos[2] := 0; { acceso incorrecto }
```

- Composición de tipos de datos: `typedef` (C) y `Type` (Pascal)

En C:

Mediante la cláusula typedef. A continuación se muestra un pequeño programa de ejemplo:

```
typedef char unbyte;
typedef unbyte cadena[20];
typedef int lst[5];
```

```
typedef float real;

int main(void)
{
  lst primos = { 2, 3, 5, 7, 11};
  real otros[3] = { 0.0, -1.2, 2e-10 };
  cadena mensaje = "un mensaje";

  return 0;
}
```

En Pascal:

Mediante la cláusula Type. El programa anterior tiene el siguiente programa equivalente en Pascal:

```
program tipos;

type
   cadena = string[20];
   lst = array[0..4] of integer;
   real = single;
var
   primos : lst;
   otros : array[0..2] of real;
   mensaje : cadena;
begin
   primos[0] := 2; primos[1] := 3; primos[2] := 5;
   primos[3] := 7; primos[4] := 11;
   otros[0] := 0.0; otros[1] := -1.2; otros[2] := 2e-10;
   mensaje := 'un mensaje';

end.
```

- Constantes

...

- Operador sizeof

Devuelve el tamaño de la expresión pasada como argumento. Su uso es idéntico tanto en Pascal como en C.

- Registros

...

- Casts

...

Operaciones aritméticas y lógicas

- Asignación:

En C:
`var1 = expresion;`
También admite: `var1 = var2 = ... = expresion;`
Asigna expresion a var1, var2, etc.

En Pascal:
`var1 := expresion;`

- Suma, resta y multiplicación:

Son idénticos en C y Pascal.

- División:

En C:
`a / b`
El tipo resultante de la operación depende de a y b. a y b pueden ser expresiones aritméticas.

En Pascal:
`a div b`
El tipo resultante es entero, a y b deben ser enteros.

`a / b`
El tipo resultante es real. a y b pueden ser enteros y/o reales.

- Resto:

En C: `a % b`
En Pascal: `a mod b`

Operaciones lógicas y de orden:

Operación	**C**	**Pascal**

Y	`exp1 && exp2 && ...`	`exp1 And exp2 And ...`
O	`exp1 || exp2 || ...`	`exp1 Or exp2 Or ...`
O exclusivo	No existe	`exp1 Xor exp2 Xor ...`
Negación	`!exp`	`Not exp`
Igualdad	`exp1 == exp2`	`exp1 = exp2`
Desigualdad	`exp1 != exp2`	`exp1 <> exp2`

Operación	**C**	**Pascal**
Igualdad	`exp1 == exp2`	`exp1 = exp2`
Desigualdad	`exp1 != exp2`	`exp1 <> exp2`
Menor, menor o igual	`e1 < e2, e1 <= e2`	`e1 < e2, e1 <= e2`
Mayor, mayor o igual	`e1 > e2, e1 >= e2`	`e1 > e1, e1 >= e1`

Ejemplos:

```
a := a xor b; (* Pascal *)
a = (!a && b) || (a && !b); /* C */

z := a <= ((3 - 1) div 2) mod 4; (* Pascal; z debe ser de tipo boolean
*)
z = a <= ((3-1) / 2) % 4; /* C */
```

Operaciones a nivel de bits:

...

Otras:

Asignación especial en C:

```
a = a + b; <-> a +=b;
a = a - b; <-> a -= b;
a = a * b; <-> a *= b;
a = a / b; <-> a /= b;
a = a % b; <-> a %= b;
```

También se aplican a nivel de bits.

Incrementos y decrementos en una unidad:

En C:

```
a = a + 1; <-> a++;
a = a - 1; <-> a--;
```

En Pascal:

```
a := a + 1; <-> Inc(a);
a := a - 1; <-> Dec(a);
```

Sentencias de control, salto y bucles

Condicionales (if .. else):

En C:

```
 if (expresion)
    ...
 else
    ...
```

```
Ejemplo:
 if (a && b) {
   a = b; b++;

 }
 else if (a < b)
   b = a;
 else
   b -= a;
```

En Pascal:

```
 if expresion then
   ...
 else
   ...
```

```
Ejemplo:
 if a and b then begin
   a := b; b := b - 1

 end

 else if a < b then
   b := a

 else
```

```
   b := b - a;
```

C además dispone de otro tipo de condicional mediante los operadores `?` y `:`

...

Selección entre varias opciones:

En C:

```
switch(expresion) {
  case const1:
    instrucciones, sin llaves;
  case const2:
    ...
  default:
    ...
}
```

expresion debe devolver un valor entero, y const1, const2, etc deben ser números enteros. El apartado default es opcional, cuyas instrucciones se ejecutan exclusivamente cuando ninguna de las constantes anteriores es igual al resultado de la expresión. Por otra parte, cada bloque de instrucciones suele ir acompañado de la instrucción `break` (ver sentencia break), puesto que si no se incluye el programa continúa su ejecución por las restantes opciones, aunque estas no se cumplan. Ejemplo:

```
switch(opcion) {
case 1:
  printf("Se ha escogido opcion 1\n");
  ejecutar(1);
  break;
case 2:
  printf("Opcion 2 identica a opcion 3\n");
case 3:
  printf("Se ha escogido opcion 2 o 3\n");
  ejecutar2_3(2);
  break;
default:
  printf("La opcion escogida no es valida\n");
}
```

En Pascal:

```
case expresion of
  rango: sentencias;
  ...
  rango: sentencias;
 else:
   sentencias;
end;
```

La expresión devolverá un valor entero, y rango puede ser una constante o rango. La parte else es opcional. Ejemplo:

```
case Ch of
  'A'..'Z', 'a'..'z':  WriteLn('Letra');
  '0'..'9':            WriteLn('Dígito');
  '+', '-', '*', '/':  WriteLn('Operador');
else
  WriteLn('otro carácter');
```

Bucle while:

En C:

```
while (expresion)
  ...
```

Ejemplo:

```
n = 5; a = i = 1;
while (i <= n) {
  a *= i;
  i++;
}
```

En Pascal:

```
while expresion do
  ...
```

Ejemplo:

```
n := 5; a := 1; i := 1;
while i <= n do begin
  a := a * i;
  inc(i)
end;
```

Bucle con comprobación de la expresion al final:

En C:

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

Se ejecuta mientras *expresion* sea verdadera. Requiere llaves aun en el caso de que haya una única instrucción. Ejemplo:

```
i = 1;
do { i++; } while (i < 10);
```

En Pascal:

```
repeat
  ...
until expresion;
```

Se ejecuta hasta que *expresion* sea verdadera, es decir, se ejecuta mientras *expresion* sea falsa. No se utilizan los señaladores de bloques Begin .. End. Ejemplo:

```
i := 1;
repeat
  writeln(i);
  inc(i)
until i = 10;
```

Bucle For:

En C:

```
for (insts1 ; exp; insts2)
   instrucciones;
```

La equivalencia con el bucle while es la siguiente:

```
insts1;
while (exp) {
  instrucciones
  insts2;
}
```

insts1, exp, insts2, e instrucciones son opcionales y pueden omitirse.
Ejemplos:

```
for (n = 1, a = i = 1; a *= i, i++)
  ;
for (n = 1, a = i = 1; i++)
  a *= i;
for (n = 1, a = i = 1; i++) {
  printf("vuelta %d\n", i);
  a *= i;
}
```

En Pascal:

```
for i := valor to valor2 do
  ...
```

y

```
for i := valor downto valor2 do
  ...
```

que son equivalentes con while de esta manera:

```
i := valor;
while i <= valor do begin
  ...
```

```
  inc(i)
end;
```
y
```
i := valor;
while i >= valor do begin
  ...
  dec(i)
end;
```

Ejemplos:

```
a := 1; n := 5;
for i := 0 to n do
  a := a * i;
```

Si se pone más de una instrucción se utilizarán los bloques Begin .. End.

Instrucciones de salto:

- break

...

- continue

...

- etc

Procedimientos y funciones

En C, procedimientos y funciones se definen de la misma manera. Simplemente el procedimiento es una función que no devuelve ningún valor; se declara con void. En Pascal un procedimiento se declara con `Procedure`, y una función con `Function`.

Si hay parámetros en la función o procedimiento, en C se separan con comas, y en Pascal con punto y coma. Ejemplos:

```
void proc1(int var1, int var2, long var3); /* C */
procedure proc1(var1, var2 : integer; var3 : LongInt); { Pascal }

float sqrt(int val);
function sqrt(val : integer) : single;

void mostrar(void);
procedure mostrar;
```

En Pascal, si se desea modificar alguno de los parámetros que se pasan a la función o

procedimiento se antepone al nombre de la variable la palabra reservada `var`. La variable se puede utilizar dentro del procedimiento o función como una variable normal. Ejemplo:

```
procedure intercambiar(var x, y : integer);
var
  aux : Integer;
begin
  aux := x;
  x := y;
  y := aux
end;
```

En C esta operación no es tan sencilla. Deben utilizarse punteros. El programa equivalente es el siguiente:

```
void intercambiar(int *x, int *y)
{
  int aux;

  aux = *x;
  *x = *y;
  *y = aux;
}
```

Al comienzo de la guía se expone un programa en C y Pascal que muestra cómo devolver un valor en una función.

Si un procedimiento o función en Pascal no tiene parámetros, entonces al llamarlo se omiten los paréntesis. En el mismo caso en C, se pondrían los paréntesis, pero sin nada dentro.

Prototipos, declaraciones Forward:

...

Cadenas de caracteres

En C, las cadenas de caracteres son simplemente arrays de caracteres. Esto es muy diferente a Pascal, que dispone de un tipo de datos llamado `string`, además de una mayor facilidad para manipular cadenas. En C, todas las cadenas deben terminar con el carácter `'\0'` o el entero `0`. Por ejemplo, las siguientes definiciones en C son todas equivalentes:

```
char cad1[2] = "a";
char cad2[2] = { 'a', '\0' };
```

```
char cad3[2] = { 'a', 0 };
```

El carácter '\0', aunque pertenece a la cadena, no aparece al utilizar funciones como printf. Ese carácter le indicará a dicha función que deje de imprimir caracteres. Si no se incluyese dicho carácter (cosa totalmente válida y nada recomendable) entonces la función avanzaría sobre el array hasta encontrar un '\0'. Posiblemente haciendo esto se salga de los límites del array, pudiendo producir un error en memoria por leer datos de donde no debe. Una cadena de caracteres vacía se representa simplemente con "". Ejemplo:
`char vacia[1] = {'\0'} ... char vacia[1] = "";`

"a", que es una cadena de caracteres en C, es igual que { 'a', '\0' }.
Notar la diferencia para una cadena de caracteres (doble comilla) y para un sólo caracter (comilla simple).

En C, puede asignarse un valor directamente a la cadena tras su declaración. Si no se asigna un valor, debe utilizarse la función `strcpy`, incluida en la librería `<string.h>`.

En Pascal, una cadena de caracteres se declara así:

```
cadena : string[tamaño];
```
tamaño debe estar comprendido entre 1 y 255.

La cadena de caracteres se representa así: `'cadena de caracteres'`. Los ordinales, como por ejemplo 'a', se señalan con comilla simple.

Si una operación sobre la cadena rebasa el tamaño de esta, como por ejemplo asignar una cadena más grande, entonces dicha cadena se trunca.

- Operaciones sobre las cadenas

Asignación:

En C:
```
char cadena[] = "cadena de caracteres";
```

o

```
char cadena[20];
char cadena2[] = "otra cadena";
...
strcpy(cadena, "una cadena");
strcyp(cadena, cadena2);

-> strcpy(cadena_destino, cadena_origen);
```
cadena_destino no puede ser una constante, debe ser una variable. Por ejemplo, no es válido lo siguiente:
`strcpy("cadena antigua", "cadena nueva");`
pues "cadena antigua" es una constante de tipo cadena de caracteres.

Nota: la función strcpy no determina si se ha rebasado la capacidad del array destino.

En Pascal:

```
cadena = 'cadena de caracteres';
```

Comparación:

En C:
Mediante la función strcmp:
`strcmp(cad1, cad2);`

Esta función devuelve un entero, que será:
entero < 0 si cad1 < cad2
entero = 0 si cad1 = cad2
entero > 0 si cad1 > cad2

Ejemplo:
```
if (strcmp(cad1, cad2) == 0) printf("Cadenas iguales");
```

En Pascal:
mediante los operadores de igualdad y desigualdad: =, <>, <, >, <=, >=.

Ejemplo:
```
if cad1 = cad2 then write('Cadenas iguales');
```

Longitud de una cadena:

En C:
mediante la función strlen. No cuenta el carácter nulo.
Ejemplo: `longitud = strlen("cadena");`

En Pascal:
mediante la función length.
Ejemplo: `longitud := length('cadena');`

Concatenar dos cadenas:

En C:

mediante la función strcat(cadenaA, cadenaB).
El resultado es cadenaA seguida de la cadenaB; cadenaA no puede ser una constante.

En Pascal:
mediante el operador + o la función Concat.
Ejemplo: `nombre_apellido := nombre + apellido;`

Punteros

El puntero nulo en C es `NULL`, en Pascal es `NIL`.
La aritmética de punteros en Pascal es muy restringida. Sólo se admite comparar dos punteros mediante los operandos = y <>.
C admite sumar y restar valores a un puntero, como se indicará más adelante, además de los operadores <=, <, >=, >, ==, !=.

Declaración en C:

```
tipo* nombre;
tipo * nombre;
tipo *nombre;
```

Tres formas equivalentes que no deben confundirse con la multiplicación. Ejemplo de puntero a entero:

```
typedef int * punteroAentero;
punteroAentero p; /* p es un puntero a entero */
```

Otro ejemplo:

```
typedef struct { int x, y; } * punto;
punto ptpunto;
```

Puede definirse un puntero sin tipo determinado con void, que luego puede manipular o ser manipulado por otros tipos de punteros :

```
void *ptvoid;
int *p;
ptvoid = (void *) 0x0000;
p = (int *) ptvoid;
p++; /* o p = p + 1 */
ptvoid = (void *)p;
```

A ptvoid se le asigna una dirección de memoria (la 0000). Posteriormente se asigna a p la dirección de memoria de ptvoid. Se incrementa p, y por último se actualiza ptvoid a p. ptovoid apuntará a la dirección de memoria 0002, pues p es un puntero a entero.

Para realizar operaciones aritméticas con un puntero de tipo void es necesario convertirlo primero:

```
ptvoid = (int *) ptvoid + 1;
```

Se obtiene lo mismo que antes.

En Pascal:

Se usa el operador `^`. Ejemplos:

```
Type
  punto = RECORD
    x, y : Integer;
  end;
  ppunto = ^punto;

Var
  ptentero = ^Integer;
  ptpunto = ^ppunto;
  sintipo = Pointer;
```

ptentero es un puntero a entero. ptpunto es un puntero a punto. sintipo es un puntero sin tipo (mediante el tipo Pointer, de alguna manera equivalente al void * de C).

Otros ejemplos:

```
int **p; /* puntero a un puntero a entero */

Type
  ptint = ^Integer;
  ptaptint = ^ptint;
Var
  p : ptaptint; { o p : ^ptint }

{ ptaptint es un puntero a un puntero a entero }
```

Operador de derreferencia `&` y `@`:

Para obtener la dirección de memoria de un operando se utiliza el operador `&` en C y el operador `@` en Pascal.
Ejemplos:

```
int *ptint, entero;
ptint = &entero; /* ptint apunta a entero */

var ptint : ^Integer;
    entero : Integer;
...
ptint = @entero; { ptint apunta a entero }
```

Operador de referencia `*` y `^`:

Para obtener el contenido de una dirección apuntada por un puntero se utiliza el operador `*` en C, y el operador `^` en Pascal. Ejemplos:

```
int *p, entero;
...
entero = 5;
p = &entero;
printf("%d\n", *p);
(*p)--;
printf("%d\n", *p);
```

El programa anterior produce la siguiente salida:
5
4

Notar que se ha puesto `(*p)--;`. Si se omiten los paréntesis entonces altera el valor del puntero, pero no altera el contenido de la dirección a la que apunta. La instrucción anterior es equivalente a: `*p = *p -1`.

El mismo programa en Pascal queda así:

```
var p : ^Integer; entero : Integer;
...
entero := 5;
p := @entero;
writeln(p^);
dec(p^);
writeln(p^);
```

Otro ejemplo, un poco más complicado:

```
int *p1, *p2, **ptp,
    a, b;
...
a = 1; b = 2;
p1 = &a; p2 = &b;
ptp = &p1;
(**ptp)++;
ptp = &p2;
(**ptp)++;
printf("%d,%d\n", a, b);
```

La salida de este programa es:
2,3

ptp apunta primero a p1. *ptp contiene la dirección de p1. **ptp contiene el valor de la dirección apuntada por *ptp (o p1). Lo mismo para cuando apunte a p2.

El mismo programa en Pascal queda así:

```
type
   ptint = ^Integer;
   pttoint = ^ptint;
var
   p1, p2 : ptint;
   ptp : pttoint;
   a, b : integer;

...
a := 1; b := 2;
p1 := @a; p2 := @b;
ptp := @p1;
inc(ptp^^);
ptp := @p2;
inc(ptp^^);
writeln(a, ',', b);
```

Manipulación de la memoria:

Para reservar memoria, en C se utiliza la función malloc, en Pascal se utiliza la función new. Ejemplo:

...

Para liberar memoria, en C se utiliza la función free, y en Pascal se utiliza la función dispose. Ejemplo:

...

Otras funciones permiten alterar el contenido de una región de memoria. En C esto se puede hacer con la función memset, y en Pascal con la función FillChar. Ejemplo:

...

Arrays y punteros en C

En C, un puntero puede manipularse como un array.
Consultar un manual.

Registros y punteros.

...

Parámetros por valor y parámetros por referencia:

En Pascal, para pasar un parámetro por referencia (ya sea un array, puntero u otro) se utiliza la instrucción Var antecediendo al nombre de la variable. En C, en lugar de pasar un parámetro por referencia, se debe pasar un puntero con la dirección de la variable que será modificada dentro de la función.

Ejemplo en Pascal:

```
procedure doble(var a : integer);
begin
     a := a + a
end;
```

Esto mismo en C queda así:

```
void doble(int *a)
{
  *a += *a;  /* *a = *a + *a */
}
```

¿Qué ocurre si se quiere cambiar el puntero en lugar de cambiar el contenido al que apunta?.
En Pascal se puede resolver utilizando Var o un puntero a puntero:

```
procedure memoria(var p : ^integer);
begin
        new(p)
end;
```

En C se debe utilizar un puntero a puntero:

```
void memoria(int **p)
{
  *p = (int *) malloc(sizeof(int));
}
```

Entrada - Salida

En C:
Comúnmente se utiliza la cabecera <stdio.h> (standard input output). Las funciones más comunes son printf, scanf, fprintf, fscanf, getchar, etc.

En Pascal:
Se utilizan write, writeln, assign, reset, etc...

Macros y preprocesado

Son exclusivas de C y C++. Se usan #define, #include, etc.