Resumen.pdf

user_4924206

Fonaments dels Ordinadors

1º Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Escuela Técnica Superior de Ingeniería de Telecomunicación (Etsetb)
Universidad Politécnica de Catalunya

T.2 (libro) EMPEZANDO A PROGRAMAR

VARIABLES

Tipos de datos de una variable

- Enteros (int) → 3, 4, -8 ...
- Reales (float) → 13.0, -4.13 ...
- Caracteres (char) → 'a', 'e' ...

Sintaxis general para declarar una variable

```
- tipo_var nom_var ;                    // Declaración 1 variable
- tipo_var nom_var1 , nom_var2 ;        // Declaración 2 variables mismo tipo
- tipo_var nom_var = valor_inicial      /* Declaración y inicialización de una variable */
```

"c" → Mensaje

'c' → nombre de un caracter ej char c = 'c'

c → caracter que se llama c

CONSTANTES

```
#define NOM_CONS valor_cons
```

#define PI 3,141516

no acaba en ;

EXPRESIONES

Operadores aritméticos

- + (suma)
- − (resta)
- % (módulo) → resto de una división entera
- * (multiplicación)
- / (división)

Operadores relacionales

- < (menor que)
- > (mayor que)
- == (igual a)
- <= (menor o igual a)
- >= (mayor o igual a)
- != (distinto que)

Operadores lógicos

- && (conjunción) → y
- !? (negación) → cierto si la condición es falsa
- || (disyunción) → o

SENTENCIAS

Sentencia de asignación

```
nom_var = expresion;
```

LIBRERIAS

#include <nom_lib.h>

```
- printf(" Bla bla bla es %d", intx );
- scanf(" %d", &intx);
```

int → %d
float → %f
char → %c

* 2 decimales float → %.2f

* (=) → operador de asignación (sentencia)
(==) → operador relacional de igualdad

T.3 (libro) TIPOS DE DATOS ELEMENTALES

CARACTERES

Tabla ASCII

Se cumple:

- 'a' < 'b' < 'c'
- 'A' < 'B' < 'C'
- '0' < '1' < '2'

Operaciones

- Asignación de caracteres: `letra = 'c'`
- Operaciones aritméticas
  - Letra de mim a MAYUS: `letra = letra - ('a' - 'A');`
  - Letra de MAYUS a mim: `letra = letra + ('a' - 'A');`
  - Saber código ASCII: `valor = num - '0';`

Declaración

char (%c)

unsigned char (%hhu)

Tabla ASCII extendida

Tabla ASCII estándar

ENTEROS

Declaración

char		%hhd
short int	(de 2 bytes)	%hd
int	(de 4 bytes)	%d
long int	(8 bytes)	%ld
unsigned short int	(naturales de 2 bytes)	%hu
unsigned int	(naturales de 4 bytes)	%u
unsigned long int	(naturales de 8 bytes)	%lu

REALES

Declaración

float	(4 bytes)	%f
double	(8 bytes)	%lf
long double	(16 bytes)	%Lf

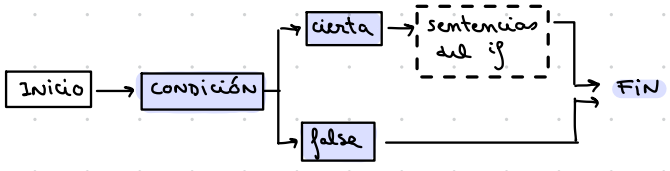
T.4 (libro) SENTENCIAS CONDICIONALES

IF

```
if (condición) {
    sentencias a ejecutar si la condición es cierta;
}
```

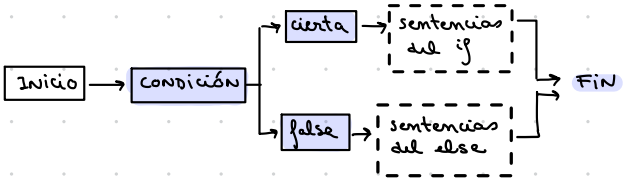
→ Se evalua primero la condición y, si es cierta se ejecutan las sentencias del if

→ Si la condición no es cierta, no se ejecuta nada



IF - ELSE

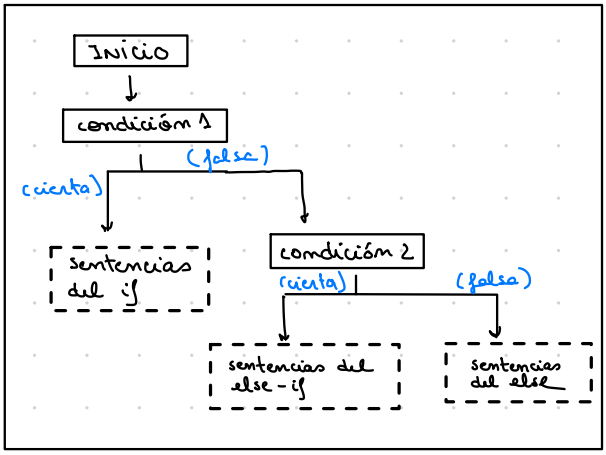
```
if (condicion) {
    sentencias si es cierta;
}
else {
    sentencias si es falsa;
}
```



* En el if y en el else es obligatorio poner las sentencias entre llaves si hay más de 1.

IF - ELSE - IF

```
if (condicion 1) {
    sentencias a ejecutar si condicion 1 es cierta;
}
else if (condición 2) {
    sentencias a ejecutar si condicion 2 es cierta;
}
else {
    sentencias a ejecutar si todas las condiciones
    anteriores son falsas;
}
```



SWITCH - CASE → permite seleccionar las acciones a realizar de acuerdo al valor que tome la variable

```
switch (variable)
{
    case valor_1 : acciones ;
                   break;
    case valor_2 : acciones;
                   break;
    default : acciones;
              break;
}
```

DO - WHILE

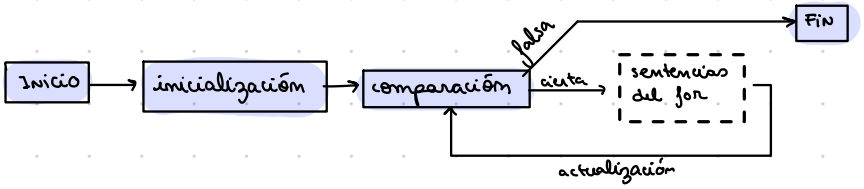
```
inicialización
do {
    instrucciones;
    actualización;
} while (comparación
```

primero ejecuta, luego comprueva

T.5 (libro) SENTENCIAS ITERATIVAS

FOR

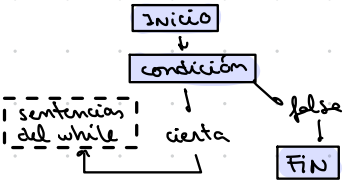
```
for (inicialización; comparación; actualización) {
    sentencias a ejecutar si la comparación es cierta;
}
```



*El for se utiliza cuando se conoce con exactitud la cantidad de iteraciones del bucle

WHILE

```
inicialización
while (comparación) {
    sentencias
    actualización
}
```



T·6 (libro) ESTRUCTURAS

DEFINICIÓN DEL TIPO DE DATO

```
typedef struct
{
  tipo_camp1  nom_camp1;      /* Definición del campo 1 */
  tipo_camp2  nom_camp2;      /* Definición del campo 2 */
   ...
  tipo_campN  nom_campN;      /* Definición del campo N */
} nombre_tipo_estructura;
```

DECLARACIÓN VARIABLE TIPO ESTRUCTURA

```
/* Declaración de una variable de tipo estructura */
nombre_tipo_estructura nom_var;

/* Declaración de dos variables de tipo estructura */
nombre_tipo_estructura nom_var1, nom_var2;

/* Declaración e inicialización de una variable de tipo
   estructura */
nombre_tipo_estructura nom_var={val_inic_camp1, val_inic_camp2,
                                ... , val inic campN};
```

T·7 (libro) VECTORES

DECLARACIÓN DE VECTORES

```
/* Declaración de una variable de tipo vector */
tipo_elemento nom_variable[dimension];

/* Declaración e inicialización de una variable de
   tipo vector */
tipo_elementos nom_variable[dimension]={val_primer_elemento,
                                        val_segundo_elemento,
                                        ...};
```

ejemplo →

```
#define DIM 80
char frase[DIM];     /* Declara la variable frase como un
                        vector de caracteres de dimensión DIM,
                        donde DIM está definido como una
                        constante con valor 80 */
```

ej. →

```
int v[4]={0,0,0,0};  /* Declara e inicializa la variable
                        v como un vector formado por 4
                        enteros, inicializados todos a 0 */
```

OPERACIONES CON VECTORES

- Acceder a un elemento concreto de un vector:

```
nom_variable[índice]   /* Acceso a un elemento de un vector */
```

ej.

```
cont[0]          /* Acceso al elemento de la posición 0 (primer
                    elemento) del vector cont */
```

- Operaciones con un elemento del vector

  - las operaciones que se pueden realizar con un element concreto de un vector son las correspondientes a su tipo

    * Si es entero → hara operaciones de enteros...

- Operaciones con vectores COMPLETOS

* NO SE PUEDE
* Se tiene que recorrer todo el vector y aplicar la operación elemento a elemento.

```
float v1[10]={1,2,3,4,5,6,7,8,9,10},
      v2[10]={2,4,6,8,0,1,3,5,7,9}, v3[10];

v3 = v1 + v2;       /* CÓDIGO INCORRECTO */

/* CÓDIGO CORRECTO */
for (i=0; i<10; i++)
   v3[i] = v1[i] +  v2[i];
```

* la primera posición de un vector es 0
* la última posición de un vector es dimension - 1

ALGORITMOS BÁSICOS DE VECTORES

• Búsqueda de un elemento de un vector

```
/* Búsqueda (primera aparición) en un vector DESORDENADO */
main()
{
  tvector v={10, {23, 45, 2, 44, 88, 39, 322, 34, 22, 10}};
  int elem, i, encontrado;

  printf("Introduzca el elemento que buscas: ");
  scanf("%d%*c", &elem);

  encontrado = 0;
  i = 0;
  while (i<v.nelem && encontrado==0)
  {
    if (v.vector[i]==elem)
      encontrado = 1;
    else
      i = i+1;
  }

  if (encontrado==0)
    printf("Elemento NO encontrado\n");
  else
    printf("Elemento encontrado en la posicion %d\n", i);
}
```

```
/* Búsqueda (primera aparición) en un vector ORDENADO */
main()
{
  tvector v={10, {322, 88, 45, 44, 39, 34, 23, 22, 10, 2 }};
  int elem, i;

  printf("Introduzca el elemento que buscas: ");
  scanf("%d%*c", &elem);

  i = 0;
  while (i<v.nelem && v.vector[i]>elem)
     i = i+1;

  if (i<v.nelem && v.vector[i]==elem)
    printf("Elemento encontrado en la posicion %d\n", i);
  else
    printf("Elemento NO encontrado\n");
}
```

• Inserción de un elemento en un vector

```
/* Inserta un elemento en un vector DESORDENADO. La inserción se
   realiza en la primera posición libre del vector */
main()
{
  tvector v={10, {23, 45, 2, 44, 88, 39, 322, 34, 22, 10}};
  int elem, i;

  printf("Introduzca el elemento a insertar:");
  scanf("%d%*c", &elem);

  /* Se comprueba si el vector está lleno */
  if (v.nelem == MAX)
    printf ("No se pudo insertar. El vector esta lleno.\n");
  else
  {
    /* Se inserta el elemento al final */
    v.vector[v.nelem] = elem;
    v.nelem = v.nelem+1;
    printf("El elemento ha sido insertado en la ultima
            posicion\n");

    printf ("VECTOR: ");
    for (i=0; i< v.nelem-1; i++)
      printf("%d, ", v.vector[i]);
    printf("%d\n", v.vector[i]);
  }
}
```

```
/* Inserta un elemento en un vector ORDENADO de mayor a menor.
   La inserción del elemento se realiza manteniendo el orden
   de los elementos del vector */
main()
{
  tvector v={10, {322, 88, 45, 44, 39, 34, 23, 22, 10, 2}};
  int elem, i, pos;

  printf("Introduzca el elemento a insertar:");
  scanf("%d%*c", &elem);

  /* Se comprueba si el vector está lleno */

  if (v.nelem == MAX)
    printf ("No se pudo insertar. El vector esta lleno.\n");
  else
  {
    /* Se busca la posición a insertar para mantener el orden */
    i = 0;
    while (i<v.nelem && v.vector[i]>elem)
      i = i+1;

    pos = i;  /* Posición donde hay que insertar el elemento */

    /* Se desplazan los elementos una posición a la derecha */
    for (i=v.nelem-1; i>=pos; i--)
      v.vector[i+1] = v.vector[i];

    /* Se inserta el elemento */
    v.vector[pos] = elem;
    v.nelem = v.nelem+1;
    printf("El elemento ha sido insertado\n");

    printf ("VECTOR: ");
    for (i=0; i< v.nelem-1; i++)
      printf("%d, ", v.vector[i]);
    printf("%d\n", v.vector[i]);
  }
}
```

• Eliminación de un elemento de un vector

```
/* Elimina un elemento del vector */
main()
{
  tvector v={10, {23, 45, 2, 44, 88, 39, 322, 34, 22, 10}};
  int i, pos;

  printf("Introduzca la posicion del elemento a eliminar:");
  scanf("%d%*c", &pos);

  /* Se comprueba si la posición es válida */
  if (pos<0 || pos>=v.nelem)
    printf ("No se pudo eliminar. Posicion no valida.\n");
  else
  {
```

```
    /* Eliminar elemento del vector */
    for (i=pos; i<v.nelem-1; i++)
      v.vector[i] = v.vector[i+1];
    v.nelem = v.nelem-1;
    printf("El elemento ha sido eliminado\n");

    printf ("VECTOR: ");
    for (i=0; i< v.nelem-1; i++)
      printf("%d, ", v.vector[i]);
    printf("%d\n", v.vector[i]);
  }
}
```


JESSE EISENBERG
WOODY HARRELSON
DAVE FRANCO
ISLA FISHER
CON ROSAMUND PIKE
Y MORGAN FREEMAN
JUSTICE SMITH
DOMINIC SESSA
ARIANA GREENBLATT
AHORA ME VES 3
DESCUBRE EL TRUCO
VER MÁS
14 DE NOVIEMBRE EN CINES
LIONSGATE

- Ordenación de los elementos de un vector

```
/* Ordena los elementos del vector de mayor a menor */
main()
{
  tvector v={10, {23, 45, 2, 44, 88, 39, 322, 34, 22, 10}};
  int aux, posmax, i, j;

  printf("Vector original: ");
  for (i=0; i<v.nelem-1; i++)
    printf ("%d - ", v.vector[i]);
  printf ("%d\n", v.vector[i]);

  /* Ordenación por el método de la selección directa */
  for (i=0; i<v.nelem-1; i++)
  {
   /* Se busca la posición del elemento mayor desde i
      a v.nelem-1 */
      posmax = i;
      for (j=i+1; j<v.nelem; j++)
        if (v.vector[j]>v.vector[posmax])
          posmax = j;

      /* Se intercambia la posición i y posmax */
      aux = v.vector[i];
      v.vector[i] = v.vector[posmax];
      v.vector[posmax] = aux;
   }

   printf("Vector ordenado: ");
   for (i=0; i<v.nelem-1; i++)
     printf ("%d - ", v.vector[i]);
   printf ("%d\n", v.vector[i]);
}
```

Matrices (ANEXO)

DECLARACION DE MATRICES

```
/* Declaración de una variable de tipo matriz */
tipo_elemento nom_variable[dimension1][dimension2];


/* Declaración e inicialización de una variable de tipo matriz */
tipo_elemento nom_variable[dimension1] [dimension2] =
{{valores_primera_fila},…{valores_ultima_fila}};


/* Declaración e inicialización de una variable de tipo matriz */
tipo_elemento nom_variable[dimension1] [dimension2] =
{val_primer_elemento, val_segundo_elemento,..., val_ultimo_elemento};
```

```
#define ALUMNOS 80
#define ASIGNATURAS 6
float notas[ALUMNOS][ASIGNATURAS];
                    /* Declara la variable notas como una
                       matriz de reales de dimensión 80x6 */

int sudoku[9][9];   /* Declara la variable sudoku como una
                       matriz de 9x9 enteros  */

int m[5][4]={{0,0,0,0},{1,1,1,1},{2,2,2,2},{3,3,3,3},{4,4,4,4}};
                    /* Declara e inicializa la variable
                       m como una matriz formada por 5x4
                       enteros, inicializada cada fila al número
                       de fila correspondiente */
```

OPERACIONES CON MATRICES

- Acceso a un elemento de la matriz

```
nom_variable[índice1][índice2]   /* Acceso a un elemento de
                                    una matriz */
```

```
notas[3][1]  /* Acceso al elemento de la posición (3,1) de la
                matriz notas*/
```

* no se pueden hacer operaciones con la matriz completa, se tiene que hacer elemento a elemento
* Cada elemento de la matriz puede hacer las operaciones de su tipo.

T·8 (libro) FUNCIONES: Paso parámetros por valor

LLAMADA A UNA FUNCIÓN

· sintaxis general:

```
nombre_función(preal1, preal2,…);
                    /* Si la función no devuelve un valor */

nom_var = nombre_función(preal1, preal2,…);
                    /* Si la función devuelve un valor */
```

DEFINICION DE UNA FUNCION

· sintaxis general:

```
tipo_result nombre_función(tipo1 pformal1,…,tipon pformaln)
{
  declaración de variables locales;

  sentencias de la función;
  return(expresión);
}
```

tipo_result

* tipo de dato del resultado que devuelve la función
* (si la función no retorna ningun valor → "void")
* Si retorna valor NO puede ser de tipo VECTOR

tipo pformal1,..., tipon pformaln → Argumento

* lista de parámetros de la función
  ↳ indican su tipo de dato y su nombre

nombre_funcion

* Identificador de la función
* Utilizan minúsculas i (-&_) no ESPACIOS.

return

* Palabra utilizada para devolver el resultado que calcula la función y que corresponde al valor de la expresión. (si la función no devuelve no se incluye)

Ejemplo:

```
float factorial(int num)
{
  int i;
  float f=1.0;

  for(i=num; i>1; i--)
    f = f*i;

  return (f);
}
```

PROTOTIPO DE UNA FUNCIÓN

· sintaxis general:

```
tipo_result nombre_función(tipo1 pformal1,…, tipon pformaln);
```

· los prototipos sirven para indicar al compilador la cantidad de parámetros formales, el tipo de dato y el tipo de resultado de la función → Así el compilador puede comprobar si la función es llamada correctamente sin conocer la definición completa.

T9 - FUNCIONES: PASO DE PARÁMETROS POR REFERENCIA

Se utiliza para funciones que → generan más de un resultado
→ necesitan modificar el valor de las variables del programa principal

ej. intercambiar_dos_valores
leer_datos ...

PASO DE PARÁMETROS POR REFERENCIA

(de tipos elementales, estructuras y vectores)

Recuérdese que la idea del paso de parámetros por referencia consiste en pasar a la función la "dirección de memoria" de las variables, en lugar de pasar el "valor" de las variables. De esta manera, los parámetros formales de la función son punteros que almacenan la dirección de memoria de otras variables, es decir, los parámetros formales "apuntan" a variables declaradas en otras funciones o en el programa principal. Así, durante la ejecución de la función se puede modificar el valor de dichas variables a través del puntero (parámetro formal).

PUNTEROS

→ Es una variable que almacena una dirección de memoria
→ Se utilizan para el paso de parámetros por referencia
→ "Apuntan" a un dato → al dato almacenado en esa dirección

DECLARACIÓN DE PUNTEROS

```
tipo *nom_var;          /* Declaración de la variable nom_var de
                           tipo puntero */
```

donde:

- `nom_var` : Es el nombre de la variable de tipo puntero.
- `tipo *` : Indica que la variable `nom_var` es de tipo puntero y que apunta a un dato del tipo especificado en `tipo`.

Ejemplos:

```
char *pcar;     /* pcar es un puntero que apunta a un dato de
                   tipo char */

float *pres;    /* pres es un puntero que apunta a un dato de
                   tipo float */
```

OPERADORES RELACIONADOS CON PUNTEROS

→ & → operador de referencia → Se puede utilizar delante de cualquier variable
→ Proporciona la dirección de memoria de dicha variable

→ * → operador de indirección → Delante de una variable de tipo puntero
→ Proporciona el contenido de la dirección de memoria que almacena dicha variable

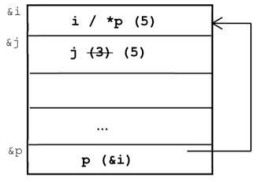
Ejemplo 2:

```
int i=5,j=3;
int *p;

p = &i;    /* La variable p es inicializada con la dirección de
              la variable i. p apunta a i */

j = *p;    /* Se asigna a la variable j el valor al cual apunta
              p, es decir, se asigna el valor de i. Estas dos
              sentencias son equivalentes a j = i; */
```

La figura 9.4 muestra cómo se almacenan en la memoria las variables de este ejemplo y a qué hace referencia cada una de las diferentes expresiones relacionadas con los punteros. De nuevo, el valor almacenado en cada posición de la memoria se indica entre paréntesis.



FUNCIONES CON STRINGS

un **string o cadena** es un vector de elementos tipo char, con particularidades:

- Tiene marca de fin (carácter '\0')
- Se puede escribir como texto dentro de comillas dobles.

* El compilador de C proporciona una librería estándar (string.h) con funciones para facilitar su uso.
  ↳ #include <string.h>

ESPECIFICADOR %s

→ printf() y scanf() tienen el formato %s para escribir o leer un string (respectivamente)

↳ lee todos los caracteres hasta ' ' y añade al final un '\0'.

↳ escribe todos los caracteres excepto el '\0'.

Ejemplo de uso:

```
#include <stdio.h>
#include <string.h>
#define MAX 40
int main()
{
    char string[MAX] = "Hola Mundo!";

    printf("%s\n", string);
```

esto es la variable → NO HACE FALTA QUE SE LLAME 'STRING'

si hiciera un printf() para introducir una frase por teclado y se introduce "Hola mundo!" el scanf solo leería hasta el espacio "Hola".

ESPECIFICADOR %[^\n]

Para poder leer la frase entera hasta llegar al salto de línea ('\n')

* Se puede sustituir '\n' por otro carácter que marque el final de la frase

FUNCIONES DE LA LIBRERÍA STRING:

size_t strlen (const char* s);

- Calcula la longitud de la cadena s
- size_t es un alias de unsigned int
- strlen devuelve el número de caracteres de s, sin contar el carácter '\0'

Ejemplo de uso:

```
#include <stdio.h>
#include <string.h>
#define MAX 40
int main()
{
   char cadena[MAX] = "Hola Mundo!";
   printf("%d\n", strlen(cadena));
}
```

`char* strcpy (char* dest, const char* src);`

- Copia la cadena src a dest, acabando con el carácter nulo de la terminación.
- Devuelve un puntero al vector dest (RECORDAR que un vector es un puntero).

Ejemplo de uso:

```
#include <stdio.h>
#include <string.h>
#define MAX 10
int main(){
      char cadena[MAX];
      char str1[MAX] = "abcdefghi";

      strcpy(cadena, str1);
      printf("%s\n", cadena);
}
```

`char* strcat (char* dest, const char* src);`

- Añade la cadena src al final de dest.
- La longitud de la cadena resultante es strlen(dest) + strlen(src)

Ejemplo de uso:

```
#include <stdio.h>
#include <string.h>
#define MAX 25
int main()
{
  char dest[MAX];
  char c[MAX]=" Mundo!", saludo[MAX]="Hola";

  strcpy(dest, saludo);
  strcat(dest, c);
  printf("%s\n", dest);
}
```

`int strcmp (const char* str1, const char* str2);`

- Compara los dos strings carácter a carácter, hasta llegar a un carácter que sea diferente o llegar a '\0' en uno de los dos strings
- Devuelve 0 si son idénticos
  - > 0 si str1 es mayor.
  - < 0 si str2 es mayor.

Ejemplo de uso:

```
#include <stdio.h>
#include <string.h>
#define MAX 25
int main(){
    char str1[MAX]= "abcd", str2[MAX]= "Abcd";

    if (!strcmp(str1, str2))
    printf("%s es igual a %s\n",str1,str2);
```

Observaciones

- **const** `char* str1` significa que el string debe contener una cadena (palabra o frase). Es decir, ha de estar inicializado.
- La función strcmp devuelve 0 en caso que las palabras sean iguales. La expresión:

  ```
  if (!strcmp(str1, str2))
  ```

  es equivalente a:

  ```
  if (strcmp(str1, str2)== 0)
  ```

- **strlen** devuelve la longitud del string, **no** la del vector de caracteres que lo contiene.