# PROGRAMACIÓ:

# LLENGUATGE C

JOAN SURRELL i SAURÍ
DEPARTAMENT D'INFORMÀTICA i MATEMÀTICA APLICADA

# LLENGUATGE C

# BIBLIOGRAFIA


- [K&R 91]
  Brian W. Kernighan, Dennis M. Ritchie
  **EL LENGUAGE DE PROGRAMACIÓN C**
  Prentice Hall Hispanoamericana, 1991

- [M&K 93]
  Lawrence H. Miller, Alexander E. Quilici
  **JOY OF C**
  John Willey & Sons, 1993

- [CCM 92]
  Jorge Castro, Felipe Cucker, Xavier Messeguer, Albert Rubio, Lluís Solano, Borja Vallés
  **CURS DE PROGRAMACIÓ**
  McGrawHill, 1992


La descripció del llenguatge algorísmic ha estat feta pel departament d'Informàtica i Matemàtica Aplicada i concorda en la seva pràctica totalitat amb la notació elaborada per Toni Soto, professor del departament de Llenguatges i Sistemes Informàtics de la Universitat Politècnica de Catalunya, i que s'utilitza en la docència de l'Escola Tècnica Superior d'Enginyers Industrials de Barcelona. La codificació en C d'aquesta notació també està basada amb la proposada pel mateix Toni Soto.

# 1. LLENGUATGE ALGORÍSMIC

## 1.1 ALGORISME

*algorisme* **::=**

**algorisme** *nom_algorisme*
*usos*
*definicio_constants*
*definicio_tipus*
*definicio_operacions*
*declaracio_variables*
*sentencies*
**falgorisme**
*implementacio_operacions*

**Àmbit de visibilitat**

- Les definicions de constants i tipus són visibles en la declaració de variables, les sentències i en totes les accions i funcions de l'algorisme.
- Les variables declarades a l'algorisme són visibles només en les sentències. No són visibles dins les accions i funcions.
- Les constants, els tipus i les operacions definides en la part d'especificació dels mòduls que usa l'algorisme són visibles en la definició de constants i tipus, en la declaració de variables, en les sentències i dins de totes les accions i funcions de l'algorisme.
- Les accions i funcions implementades junt amb l'algorisme només són visibles dins les sentències de l'algorisme i les accions i funcions de l'algorisme. No són visibles en cap altre mòdul.

## 1.2 MÒDUL

*modul* **::=**

**modul** *nom_modul*
**especificacio**
*usos*

```
            definicio_constants
            definicio_tipus
            definicio_operacions
        especificacio

        implementacio
            usos
            definicio_constants
            definicio_tipus
            definicio_variables
            definicio_operacions_privades
            implementacio_operacions
            implementacio_operacions_privades
        fimplementacio
    fmodul
```

**Àmbit de visibilitat**

- Les definicions de constants i tipus de la part d'especificació del mòdul també són visibles en la part d'implementació.
- Les constants, els tipus i les operacions importades a la part d'especificació d'un mòdul (definides en els mòduls usats) són visibles també en la part d'implementació.
- Les constants, tipus, variables i mòduls usats en la part d'implementació no són visibles enlloc més que en aquesta part.
- Les variables declarades a la part d'implementació del mòdul són visibles (globals) a totes les operacions, accions i funcions que s'implementin en aquesta part del mòdul.

## 1.3 MÒDULS USATS

*usos* **::=**

```
usa
    nom_modul
    ...
fusa
```

## 1.4 DEFINICIÓ DE CONSTANTS

*definicio_constants* **::=**

```
const
    nom_constant: nom_tipus = expressio_constant
```

```
        ...
fconst
```

## 1.5 DEFINICIÓ DE TIPUS

*definicio_tipus* **::=**

```
tipus
      nou_tipus
      ...
ftipus
```

*nou_tipus* **::=**

*nom_nou_tipus* **=** *tipus_elemental*
*nom_nou_tipus* **= taula [***rang***,...] de** *nom_tipus*
*nom_nou_tipus* **= tupla**
      *nom_camp***,...:** *nom_tipus*
      **...**
**ftupla**
*nom_nou_tipus* **= (***nom*$_1$**,...)**
*nom_nou_tipus* **=** *rang*

*tipus_elemental* **::=**

```
enter
real
caracter
boolea
```

*rang* **::=**

*exp_constant_enumerable***..***exp_constant_enumerable*

## 1.6 OPERACIONS

*definicio_operacions* **::=**

**accio** *nom_accio***(***param_formals***,...)**
**funcio** *nom_funcio***(***param_funcio***,...) retorna** *nom_tipus*

*definicio_operacions_privades* **::=**

**accio privada** *nom_accio***(***param_formals***,...)**
**funcio privada** *nom_funcio***(***param_funcio***,...)retorna** *nom_tipus*

*implementacio_operacions* **::=**

*implementacio_accions*
*implementacio_funcions*

*implementacio_operacions* **::=**

*implementacio_accions_privades*
*implementacio_funcions_privades*

## 1.7 DECLARACIÓ DE VARIABLES

*declaracio_variables* **::=**

**var**
*nom_variable*,...: *nom_tipus*
...
**fvar**

## 1.8 SENTÈNCIES

*sentencia* **::=**

*assignacio*
*condicional*
*iterativa*
*iterador_per*
*iterador_repetir*
*crida_accio*

*assignacio* **::=**

*nom_variable* **:=** *expressio*

*condicional* **::=**

**si**
*expressio_booleana*$_1$ **llavors**
*sentencia*$_1$
...
...
[**altrament**
*sentencia*
...]
**fsi**

*iterativa* **::=**

**mentre** *expressio_booleana* **fer**
*sentencia*
**...**
**fmentre**

*iterador_per* **:=**

**per** *nom_variable_enumerable* **de** $expressio_1$ **fins** $expressio_2$
**[pas** *expressio_entera***] fer**
*sentencia*
**...**
**fper**

*iterador_repetir* **:=**

**repetir**
*sentencia*
**...**
**fins_que** *expressio_booleana*

*crida_accio* **::=**

*nom_accio***(***parametre_actual***,...)**

*parametre_actual* **::=**

*expressio*

- **Assignació**: Es permesa qualsevol assignació sempre que hi hagi concordància de tipus entre l'expressió i la variable. Per tant, considerem correctes assignacions entre taules o tuples.
- **Funcions de conversió de tipus**: Es consideren definides les següents funcions de conversió de tipus:

```
funcio real(ent e: enter) retorna real

funcio truncar(ent r: real) retorna real

funcio arrodonir(ent r: real) retorna real

funcio caracter(ent e: enter) retorna caracter

funcio ordinal(ent c: caracter) retorna enter
```

# 1.9 EXPRESSIONS

*expressio* **::=**

**(***expressio***)**
*nom_variable*
*constant*
*operador_unari expressio*
*expressio operador_binari expressio*
*expressio***[***expressio_enumerable***,...]**
*expressio***.***nom_camp*
*nom_funcio***(***expressio***,...)**

*operador_unari* **::=**

**-**
**no**

*operador_binari* **::=**

*****
**/**
**div**
**mod**
**+**
**-**
**<**
**>**
**=**
**≤**
**≥**
**≠**
**i**
**o**

• Les **prioritats dels operadors**, de major a menor, són els següents

| | |
|---|---|
| **()** | parèntesi |
| **. []** | accés a tuples i taules |
| **- no** | canvi de signe i negació lògica |
| *** / div mod** | operadors multiplicatius |
| **+ -** | operadors additius |
| **< > ≤ ≥ = ≠** | operadors relacionals |
| **i** | multiplicació lògica |
| **o** | suma lògica |

- L'associativitat entre operadors de la mateixa prioritat és d'esquerra a dreta.

# 1.10 IMPLEMENTACIÓ D'ACCIONS

*implementacio_accions_privades* **::=**

**accio privada** *nom_accio*(*param_formals*,...)
*definicio_constants*
*definicio_tipus*
*declaracio_variables*
*sentencies*
**faccio**

*param_formals* **::=**

**ent** *nom_param*,...: *nom_tipus*
**sort** *nom_param*,...: *nom_tipus*
**ent/sort** *nom_param*, ...: *nom_tipus*

**Accions Privades**

Les accions privades que apareixen a la part d'implementació d'un mòdul han de portar la paraula clau **privada** entre **accio** i ***nom_accio***.

*implementacio_accions_privades* **::=**

**accio privada** *nom_accio*(*param_formals*,...)
*definicio_constants*
*definicio_tipus*
*declaracio_variables*
*sentencies*
**faccio**

# 1.11 IMPLEMENTACIÓ DE FUNCIONS

*implementacio_funcio* **::=**

**funcio** *nom_funcio*(*param_funcio*,...) **retorna** *nom_tipus*
*definicio_constants*
*definicio_tipus*
*declaracio_variables*
*sentencies*
**retorna** *expressio*
**ffuncio**

*param_funcio* **::=**

**ent** *nom_param*,...: *nom_tipus*

**Funcions privades**

- Les funcions privades que apareixen a la part d'implementació d'un mòdul han de portar la paraula clau **privada** entre **funcio** i ***nom_funcio***.

*implementacio_funcio_privada* **::=**

```
funcio privada nom_funcio(param_funcio,...)
  retorna nom_tipus
      definicio_constants
      definicio_tipus
      declaracio_variables
      sentencies
      retorna expressio
ffuncio
```

# 1.12 IDENTIFICADORS

- En els apartsts anteriors tots els símbols que comencen per ***nom*** són identificadors.
- Un identificador és una seqüencia de lletres, dígits i caràcters subratllat (_) que comença en una lletra.
- Les lletres majúscules i minúscules es consideren diferents. Per tant, identidicadors com **punt** i **Punt** són diferents.

# 1.13 CONSTANTS

- Les constants enteres són una seqüència de dígits precedits opcionalment de signe.

```
32        -347      0
```

- Les constants reals s'han d'escriure amb el punt decimal i es poden escriure en notació exponencial.

```
0.0       -3.56e12  48.2E-3
```

- Les constants caràcters s'escriuen entre cometes senzilles.

```
'A'       'a'       '%'
```

- Les constants booleanes són **cert** i **fals**

## 1.14 ENTRADA/SORTIDA

- Es proposen les següents operacions predefinides d'entrada/sortida.

```
accio EscriureEnter(ent e: enter)

accio EscriureReal(ent r: real)

accio EscriureCaracter(ent c: caracter)

funcio LlegirEnter() retorna enter

funcio LlegirReal() retorna real

funcio LlegirCaracter() retorna caracter

accio EscriureFrase("frase")
```

## 1.15 CADENES DE CARÀCTERS

- Són taules de caràcters del tipus

```
tipus
     cadena = taula [0..10] de caracter
     cad = taula [0..3] de caracter
     pcad = taula [0..2] de caracter
ftipus
```

- Existeixen constants del tipus

```
"Hola"
```

  que són constants de tipus taula de caràcter, compatible, pel que fa a l'assignatura, amb qualsevol taula amb nombre d'elements suficient per encabir la constant.

- S'usa el conveni de delimitar una cadena dins d'una taula de caràcter via el caràcter NUL. Això implica que la constant

```
"Que"
```

  és realment representada com

```
'Q'   'u'   'e'   NUL
```

  i té longitud 3 encara que requereix 4 caràcters per emmagatzemar-se. Ex:

```
var
     c1: cadena
     c2: cad
     c3: pcad
fvar

c1 := "Que"     {'Q''u''e' NUL ? ? ? ? ? ? ? }
c2 := "Que"     {'Q''u''e' NUL }
c3 := "Que"     {Assignació invalida, c3 no te suficient espai }
```

- La regla d'assignació és vàlida pel pas de paràmetres d'entrada.

# 2. CODIFICACIÓ EN C

## 2.1 ALGORISME

*algorisme* **::=**
*inclusions*
*definicio_constants*
*definicio_tipus*
*prototipus_operacions*
**void main(void) {**
*declaracio_variables*
*sentencies*
**}**
*implementacio_operacions*

## 2.2 MÒDUL

*modul* **::=**

```
/* Fitxer nom_modul.h */
#ifndef INCLUDE_nom_modul
#define INCLUDE_nom_modul
```
*inclusions*
*definicio_constants*
*definicio_tipus*
*prototipus_operacions*
```
#endif

/* Fitxer nom_modul.c */
#include "nom_modul.h"
```
*inclusions*
*definicio_constants*
*definicio_tipus*
*prototipus_operacions_privades*
*declaracio_variables*
*implementacio_operacions*
*implementacio_operacions_privades*

## 2.3 MÒDULS USATS

*inclusions* **::=**

```
#include "nom_modul.h"
...
```

- La part d'*inclusions*, a més dels fitxers `nom_modul.h`, ha de contenir la inclusió dels fitxers que calgui de la llibreria estàndard.
- Sempre, en un algorisme o en la part d'implementació d'un mòdul:

```
#include <assert.h>
```

- Quan s'usi l'entrada/sortida estandard (**`getchar`**, **`scanf`**, **`printf`**,...) o el tipus **`FILE`** i les operacions sobre fitxers (**`fopen`**, **`fread`**, **`fwrite`**,...):

```
#include <stdio.h>
```

- Quan s'usin funcions que treballen amb cadenes de caràcters (**`strcpy`**, **`strcmp`**, **`strlen`**,...):

```
#include <string.h>
```

- Quan s'usin funcions matemàtiques (**`exp`**, **`log`**, **`sin`**, **`fabs`**,...):

```
#include <math.h>
```

## 2.4 DEFINICIÓ DE CONSTANTS

*definicio_constants* **::=**

**`#define`** *nom_constant expressio_constant*
**...**

## 2.5 DEFINICIÓ DE TIPUS

*definicio_tipus* **::=**

*nou_tipus*
**...**

*nou_tipus* **::=**

```
typedef tipus_elemental nom_nou_tipus;
typedef nom_tipus nom_nou_tipus[num_elem]...;
typedef struct {
      nom_tipus nom_camp,...;
      ...
} nom_nou_tipus;
```

*tipus_elemental* **::=**

```
int
float
char
```

- Els rangs de les taules comencen en l'index **0** fins ***num_elem*-1**.
- Per definir una taula que pugui emmagatzemar el mateix número d'elements que en el rang

  *primer_rang..darrer_rang*

  ***num_elem*** ha de ser

  *num_elem = darrer_rang - primer_rang + 1*

- No existeix el tipus **boolea**. Es pot definir com:

```
typedef enum {fals=0, cert=1} boolea;
```

  i usar-lo a partir d'aquest moment com en el Llenguatge Algorísmic.

## 2.6 OPERACIONS

*prototipus_operacions* **::=**

```
void nom_accio(parametre_formal,...)
nom_tipus nom_funcio(parametre_funcio,...)
```

*prototipus_operacions_privades* **::=**

```
static void nom_accio(parametre_formal,...)
static nom_tipus nom_funcio(parametre_funcio,...)
```

*implementacio_operacions* **::=**

*implementacio_accions*
*implementacio_funcions*

*implementacio_operacions_privades* **::=**

*implementacio_accions_privades*
*implementacio_funcions_privades*

## 2.7 DECLARACIÓ DE VARIABLES

*declaració_variables* **::=**

*nom_tipus nom_variable*`,...;`
`...`

- Les variables globals declarades en la part d'implementació d'un mòdul han de dur el qualificatiu **`static`** abans del nom del tipus de la variable.

## 2.8 SENTÈNCIES

*sentencia* **::=**

*assignacio*
*condicional*
*iterativa*
*iterador_per*
*iterador_repetir*
*crida_accio*

*assignacio* **::=**

*nom_variable* `=` *expressio*`;`

*condicional* **::=**

```
if (expressio_booleana1)
{
      sentencia1;
      ...
}
else if (expressio_booleana2)
{
      sentencia2;
      ...
...
}
else assert(fals);
```

*condicional* **::=**

```
if (expressio_booleana1)
{
      sentencia1;
      ...
}
else if (expressio_booleana2)
{
      sentencia2;
      ...
...
}
else
{
      sentencia;
      ...
}
```

*iterativa* **::=**

```
while (expressio_booleana)
{
      sentencia;
      ...
}
```

*iterador_per* **::=**

```
for (nom_variable_entera = expressio1;
      nom_variable_entera <= expressio2;
      nom_variable_entera = nom_variable_entera + 1)
{
      sentencia;
      ...
}
```

*iterador_per* **::=**

```
for (nom_variable_entera = expressio1;
      nom_variable_entera <= expressio2;
      nom_variable_entera = nom_variable_entera + expressio_entera)
{
      sentencia;
      ...
}
```

*iterador_repetir* **::=**

```
do
{
```

```
        sentencia;
        ...
    } while (!expressio_booleana)
```

*crida_accio* **::=**

```
    nom_accio(parametre_actual,...);
```

*parametre_actual* **::=**

*nom_variable*
**&***nom_variable*
*expressió*

**Assignació**

- Qualsevol assignació és permesa llevat que els tipus de l'expressió i de la variable siguin taules. En aquest cas, el compilador no generarà un error però l'assignació no funcionarà correctament (no produeix una còpia d'una regió de memòria sobre una altra sinó una còpia d'apuntadors).

**Condicional**

- La darrera línia de la sentència condicional ( `} else assert(fals);` ) força a que, com a mínim, una de les expressions booleanes sigui certa.

**Paràmetres actuals**

- El primer cas s'ha d'usar quan el paràmetre formal corresponent és d'entrada i la variables no és de tipus taula, és a dir, és de tipus elemental, o de tipus tupla, o bé
- quan el paràmetre formal corresponent és d'entrada, sortida o entrada/sortida i la variables és de tipus taula.
- El segon cas s'usa quan el paràmetre formal corresponent és de sortida o d'entrada/sortida i la variables no és de tipus taula, és a dir, és de tipus elemental, o bé de tipus tupla.
- El tercer cas s'usa quan el paràmetre formal corresponent és d'entrada, i no és de tipus taula.
- **ATENCIÓ!** La variable pot ser de la forma ***nom_variable**. Cal recordar que els operadors **&** i ***** són l'invers l'un de l'altre. Per tant, si després d'aplicar les regles anteriors apareix **&*nom_variable**, s'ha de substituir per **nom_variable**.

# 2.9 EXPRESSIONS

*expressio* **::=**

> **(***expressio***)**
> *nom_variable*
> *constant*
> *operador_unari expressio*
> *expressio operador_binari expressio*
> *expressio***[***expressio_entera***]...**
> *expressió***.***nom_camp*
> *nom_funcio***(***expressio***,...)**

*operador_unari* **::=**

> **-**
> **!**

*operador_binari* **::=**

> **+**
> **-**
> *****
> **/**
> **%**
> **<**
> **>**
> **==**
> **!=**
> **<=**
> **>=**
> **&&**
> **||**

- Les conversions de tipus tenen, en general, el següent format

```
((tipus_destinacio)(expressio_a_convertir))
```

de manera que les funcions de conversió es tradueixen per

| | |
|---|---|
| enter | `((int)(3.5 + 2.1))` |
| real | `((float)(4))` |
| ordinal | `((int)('a'))` |
| caracter | `((char)(120))` |

- Les **prioritats dels operadors**, de major a menor, són

| | |
|---|---|
| **()** | parèntesi |
| **. []** | accés a tuples i taules |
| **- ! & *** | canvi de signe, negació lògica , desreferència i adreça |
| *** / %** | operadors multiplicatius |
| **+ -** | operadors additius |
| **< > <= >=** | operadors relacionals |
| **= !=** | igualtat i desigualtat |
| **&&** | multiplicació lògica |
| **\|\|** | suma lògica |

- **ATENCIÓ!** La prioritat de l'operador de desreferència ***** és més petita que la de l'operador d'accés a un camp d'una tupla. Per tant, per accedir a un camp d'un paràmetre de tipus tupla passat per referència cal usar parèntesi **(*nom_param).camp**, o bé l'operador **nom_param->camp**.
- L'**associativitat** és la mateixa que en Llenguatge Algorísmic llevat dels operadors unaris **-** i **!** que és de dreta a esquerra.

## 2.10 IMPLEMENTACIÓ D'ACCIONS

*accio* **::=**

```
void nom_accio(param_formals,...)
{
        definicio_constants
        definicio_tipus
        declaracio_variables
        sentencia;...
}
```

*param_formals* **::=**

```
nom_tipus  nom_param
const  nom_tipus  nom_param
nom_tipus  *const  nom_param
nom_tipus  nom_param
nom_tipus  *const  nom_param
nom_tipus  nom_param
```

**Accions privades**

- Les accions privades que apareixin a la part d'implementació d'un mòdul han de portar la paraula clau **static** abans de **void**.

```
accio_privada ::=
        static void nom_accio(param_formal,...)
        {
                definicio_constants
                definicio_tipus
                declaracio_variables
                sentencia;...
        }
```

**Paràmetres formals**

- El primer cas és pels paràmetres d'entrada que no siguin de tipus taula o pels paràmetres de sortida i entrada/sortida de tipus taula.
- El segon cas és pels paràmetres d'entrada de tipus taula.
- El tercer cas és pel paràmetres de sortida que no són de tipus taula.
- El quart cas és pels paràmetres de sortida de tipus taula.
- El cinquè cas és pels paràmetres d'entrada/sortida que no són de tipus taula.
- El sisè cas és pel paràmetres d'entrada/sortida de tipus taula.

**Us dels paràmetres formals**

- Tot paràmetre passat per referència (els de *****) s'ha d'usar dins del cos de l'acció com ***nom_param**. Cal tenir en compte que, si aquest paràmetre es vol tornar a passar per referència a una altra acció, **&*nom_param** equival a **nom_param**.
- L'accés a un camp d'un paràmetre de tipus tupla passat per referència es pot fer de dues formes: **(*nom_param).nom_camp** o bé **nom_param->nom_camp**.

## 2.11 IMPLEMENTACIÓ DE FUNCIONS

```
funcio ::=

        nom_tipus  nom_funcio(param_funcio,...)
        {
                definicio_constants
                definicio_tipus
```

```
        declaracio_variables
        sentencia;...
        return expressio;
}
```

*param_funcio* **::=**

```
nom_tipus  nom_param
const  nom_tipus  nom_param
```

**Funcions privades**

- Les funcions privades que apareixin a la part d'implementació d'un mòdul han de portar la paraula clau **`static`** abans de ***`nom_tipus`***.

*funcio_privada* **::=**

```
static  nom_tipus  nom_funcio(param_funcio,...)
{
        definicio_constants
        definicio_tipus
        declaracio_variables
        sentencia;...
        return  expressio
}
```

**Paràmetres formals**

- El primer cas és pel paràmetres d'entrada que no siguin de tipus taula.
- El segon cas és pels paràmetres d'entrada de tipus taula.

# 2.12 IDENTIFICADORS

- Tots els símbols que comencen per ***`nom_`*** són identificadors.
- Tot identificador és una seqüència de lletres, dígits i caràcters subratllar (_) que comença en una lletra.
- Les lletres majúscules i minúscules es consideren diferents. Per tant, identificadors com *`punt`* i *`Punt`* són diferents.

## 2.13 CONSTANTS

- La sintàxi en C per les constants enteres, reals i de caràcter és la mateixa que en Llenguatge Algorísmic.
- No existeix el tipus booleà. Es pot definir com

```
typedef enum {fals=0, cert=1} boolea;
```

  i usar-lo a partir d'aquest moment com en Llenguatge Algorísmic.

- Es poden usar les següents constants especials de tipus caràcter

| | |
|---|---|
| '\n' | salt de línia |
| '\t' | tabulador |
| '\0' | caràcter nul |

## 2.14 ENTRADA/SORTIDA

- Per tal de facilitar l'us de l'entrada/sortida es proposa que, en principi, s'usin les següents accions i funcions d'entrada sortida, que són traducció de les usades en notació algorísmica:

```
void EscriureEnter(int e)
{
     printf("%d", e);
}

void EscriureReal(float r)
{
     printf("%g", r);
}

void EscriureCaracter(char c)
{
     putchar(c);
}

int LlegirEnter(void)
{
     int e, ret;

     ret = scanf("%d", &e);
```

```
        assert(ret == 1);
        return e;
}

float LlegirReal(void)
{
        int ret;
        float r;
        ret = scanf("%f", &f);
        assert(ret == 1);
        return r;
}

char LlegirCaracter(void)
{
        return getchar();
}
```

## 2.15 CADENES DE CARÀCTERS

- S'ha establert, en el Llenguatge Algorísmic, el mateix conveni per les cadenes de caràcters que s'utilitza en C. L'única diferència fa referència a la regla d'assignació: No és possible assignar una constant cadena a una variable d'algun tipus cadena. Si que és possible, però, passar una constant cadena de caràcters com a paràmetre.
- C proporciona un conjunt de funcions a la llibreria **<string.h>** que faciliten el treball amb cadenes.

```
char *strcpy(char *x, const char *y);
/* Copia sobre la cadena x la dena y */

char *strcat(char *x, const char *y);
/*Concatena a la cadena x la cadena y */

int strcmp(const char *x, const char *y);
/*Retorna <0 si x<y, =0 si x=y i >0 si x>y*/
```

- **ATENCIÓ!** El paràmetre **x** de **strcpy** i **strcat** ha de tenir suficient espai per encabir **y** en el primer cas i **x** concatenat amb **y** en el segon.

# 3. EXEMPLES

## 3.1 EQUACIONS DE SEGON GRAU

Escriure un algorisme per resoldre equacions de segon grau. Cal tenir en compte tots els casos possibles.

```
ALGORISME Equacions2nGrau
     VAR
          a, b, c, Discriminant, x1, x2, x, xReal, xImaginaria: Real
     FVAR

     EscriuFrase("Aquest programa serveix per resoldre")
     EscriuFrase("equacions de segon grau.")
     EscriuFrase("Nomes cal entrar els valors de A, B i C")
     EscriuFrase("i la maquina ja calcula la solucio,")
     EscriuFrase("sigui aquesta real, doble o complexa")
     EscriuFrase("Entra el valor de a:")
     LlegirReal(a)
     EscriuFrase("Entra el valor de b:")
     LlegirReal(b)
     EscriuFrase("Entra el valor de c:")
     LlegirReal(c)
     Discriminant:= b*b - 4.0*a*c
     SI (Discriminant > 0.0) LLAVORS
          x1:= (-b + Arrel(Discriminant)) / (2.0*a)
          x2 = (-b - Arrel(Discriminant)) / (2.0*a)
          EscriuFrase("Les arrels obtingudes son reals i valen:")
          EscriuReal(x1); EscriuReal(x2)
     ALTRAMENT SI (Discriminant = 0.0) LLAVORS
          x = -b / (2.0*a)
          EscriuFrase("L'arrel obtinguda es doble i val:")
          EscriuReal(x)
     ALTRAMENT
          xReal = -b / (2.0*a);
          xImaginaria = Arrel(-Discriminant) / (2.0*a)
          EscriuFrase("Les arrels obtingudes son complexes i valen:")
          EscriuReal(xReal); EscriuFrase(" + ")
          EscriuReal(xImaginaria)
          EscriuReal(xReal); EscriuFrase(" - ")
          EscriuReal(xImaginaria)
     FSI
FALGORISME
```

```
/*************************************/
/* Resolucio d'equacions de segon grau */
/*************************************/

#include <stdio.h>
#include <math.h>

main()
{
     float a, b, c, discriminant, x1, x2, x, x_real, x_imaginaria;

     /* presentacio del programa */
     printf("\n\nAquest programa serveix per resoldre\n");
     printf("equacions de segon grau.\n\n");
     printf("\nNomes cal entrar els valors de A, B i C\n");
     printf("i la maquina ja calcula la solucio,\n");
     printf("sigui aquesta real, doble o complexa\n");

     /* entrada de dades */
     printf("\nEntra el valor de a:\n");
     scanf("%g", &a);
     printf("\nEntra el valor de b:\n");
     scanf("%g", &b);
     printf("\nEntra el valor de c:\n");
     scanf("%g", &c);

     /* calcul del resultats i presentacio */
     discriminant = b*b - 4.0*a*c;
     if (discriminant > 0) {
          /* arrels reals */
          x1 = (-b+sqrt(discriminant)) / (2.0*a);
          x2 = (-b-sqrt(discriminant)) / (2.0*a);
          printf("\nLes arrels obtingudes son reals i valen:\n");
          printf("\n\t x1: %g",x1);
          printf("\n\t x2: %g",x2);
     }
     else if (discriminant == 0) {
          /* una arrel doble */
          x = -b / (2.0*a);
          printf("\nL'arrel obtinguda es doble i val:\n");
          printf("\n\t x: %g",x);
     }
     else { /* discriminant negatiu, arrels complexes */
          x_real = -b / (2.0*a);
          x_imaginaria = sqrt(-discriminant) / (2.0*a);
          printf("\nLes arrels obtingudes son complexes i valen:\n");
          printf("\n\t x1: %g + %g",x_real, x_imaginaria);
          printf("\n\t x2: %g - %g",x_real, x_imaginaria);
     }
     printf("\n\n\n\n");
}
```

## 3.2 DIVISORS D'UN VALOR ENTER

Dissenyar un algorisme per determinar els divisors d'un valor enter positiu.

```
ALGORISME Divisors
     VAR
          Divisor, Numero: Enter
     FVAR

     EscriureFrase("Aquest programa calcula tots els divisors")
     EscriureFrase("enters d'un valor positiu")
     EscriureFrase("Entra el valor a considerar")
     LlegirEnter(Numero)
     Divisor:= 1
     MENTRE (Divisor < Numero) FER
          SI ((Numero MOD Divisor) = 0) LLAVORS
               EscriuFrase("El valor ")
               EscriuEnter(Divisor)
               EscriuFrase(" es divisor de ")
               EscriuEnter(Numero)
          FSI
          Divisor:= Divisor + 1
     FMENTRE
FALGORISME
```

```
/*************************************************/
/* Programa per calcular divisors d'un valor enter */
/*************************************************/

#include <stdio.h>

main()
{
     /* declaracio de variables */
     int divisor, numero;

     /* presentacio del programa */
     printf("\n\nAquest programa calcula tots els divisors \n");
     printf("enters d'un valor positiu");

     /* entrada de dades */
     printf("\n\n Entra el valor a considerar:\n");
     scanf("%d", &numero);

     /* proces i presentacio de resultats */
     divisor = 1;
     while (divisor < numero) {
          if (numero % divisor == 0) {
               printf("\n El valor %d es divisor de %d.",
                              divisor, numero);
          }
          divisor = divisor + 1;
     }
     printf("\n\n\n");
}
```

## 3.3 NOMBRES PRIMERS

Dissenyar un algorisme per determinar si un cert valor és o no un nombre primer.

```
ALGORISME NPrimers
     VAR
          n, i: Enter
          Primer: Boolea
     FVAR

     EscriuFrase("Aquest programa determina si un valor enter")
     EscriuFrase("es o no primer")
     EscriuFrase("Entra un valor enter positiu:")
     LlegirEnter(n)

     Primer:= Cert
     i:= 2
     MENTRE (Primer AND (i < n)) FER
          SI (n MOD i = 0) LLAVORS
               Primer:= Fals
          FSI
          i:= i + 1
     FMENTRE
     SI (Primer) LLAVORS
          EscriuFrase("El valor ")
          EscriuEnter(n)
          EscriuFrase(" es un nombre primer")
     ALTRAMENT
          EscriuFrase("El valor ")
          EscriuEnter(n)
          EscriuFrase(" no es un nombre primer")
     FSI
FALGORISME
```

```
/*********************************************/
/* programa per trobar si un nombre es primer */
/*********************************************/

#include <stdio.h>

/* definicio dels valors booleans */
typedef enum {Fals = 0, Cert = 1} Boolea;

main()
{
     int n, i;
     Boolea primer;

     /* presentacio */
     printf("\n\n\n Aquest programa determina si un valor\n");
     printf("es o no primer\n\n");

     /* entrada de dades */
     printf("\n Entra un valor enter positiu:\n");
     scanf("%d", &n);

     /* proces */
     primer = Cert;
     i = 2;
     while (primer && (i < n)) {
          if (n % i == 0) {
               primer = Fals;
          }
          i = i + 1;
     }
     if (primer) {
          printf("\n\n El valor %d es un nombre primer\n\n", n);
     }
     else {
          printf("\n\n El valor %d no es un nombre primer\n\n", n);
     }
}
```

## 3.4 COMPTAR ELEMENTS D'UNA SEQÜENCIA

Dissenyar un algorisme que determini el nombre de vegades que es repeteix la lletra A en un text acabar en un punt.

```
ALGORISME nDeAs
     VAR
          n_de_A: Enter
          c: Caracter
     FVAR

     EscriuFrase("Aquest programa compte el n. d'As")
     EscriuFrase("que hi ha en un text. Nomes cal entrar")
     EscriuFrase("el text acabat en un punt i el programa")
     EscriuFrase("ja treu el resultat.")

     n_de_A:= 0
     Llegir(c)
     MENTRE (c ≠ '.') FER
          SI (c = 'A') LLAVORS
               n_de_A:= n_de_A + 1
          FSI
          LlegirCaracter(c)
     FMENTRE

     Escriu("El nombre de As del text es: )
     Escriu(n_de_A)
FALGORISME
```

```
/************************************************/
/* programa per comptar el nombre de A d'un text */
/************************************************/
#include <stdio.h>

main()
{
     /* declaracio de variables */
     int n_de_A;
     char c;

     /* presentacio del programa */
     printf("\n\n Aquest programa compte el n. d'As\n");
     printf(" que hi ha en un text. Nomes cal entrar\n");
     printf(" el text acabat en un punt i el programa");
     printf(" ja treu el resultat.\n\n");

     /* proces de recorregut */
     n_de_A = 0;
     c = getchar();
     while (c != '.') {
          if (c =='A') {
          n_de_A = n_de_A + 1;
          }
          c = getchar();
     }

     /* sortida */
     printf("\n\nEl nombre de As del text es: %d\n\n\n", n_de_A);
}
```

## 3.5 COMPTAR PARELLES D'UNA SEQÜÈNCIA

Disenyar un algorisme que permeti comptar el nombre de vegades que es repeteix una parella de lletres en un text acabat en un punt. Fer-ne diverses variants.

```
ALGORISME nDeLAs
     VAR
          n_de_LA: Enter
          c: Caracter
     FVAR

     EscriuFrase(" Aquest programa compte el n. de LAs")
     EscriuFrase(" que hi ha en un text. Nomes cal entrar")
     EscriuFrase(" el text acabat en un punt i el programa")
     EscriuFrase(" ja treu el resultat.")

     n_de_LA:= 0
     Llegir(c)
     MENTRE (c ≠ '.') FER
          SI (c = 'L') LLAVORS
               LlegirCaracter(c)
               SI (c = 'A') LLAVORS
                    n_de_LA = n_de_LA + 1
               FSI
          ALTRAMENT
               LlegirCaracter(c)
     FMENTRE

     EscriuFrase("El nombre de LAs del text es: ")
     EscriuEnter(n_de_LA)
FALGORISME
```

```
/*******************************************************/
/* programa per comptar el nombre de LAs d'un text, v1 */
/*******************************************************/
#include <stdio.h>

main()
{
     /* declaracio de variables */
     int n_de_LA;
     char c;

     /* presentacio del programa */
     printf("\n\n Aquest programa compte el n. de LAs\n");
     printf(" que hi ha en un text. Nomes cal entrar\n");
     printf(" el text acabat en un punt i el programa");
     printf(" ja treu el resultat.\n\n");

     /* proces de recorregut */
     n_de_LA = 0;
     c = getchar();
     while (c != '.') {
          if (c =='L') {
               c = getchar();
               if (c =='A')
                    n_de_LA = n_de_LA + 1;
          }
          else
               c = getchar();
     }

     /* sortida */
     printf("\n\nEl nombre de LAs del text es: %d\n\n\n", n_de_LA);
}
```

```
ALGORISME nDeLAs2
     VAR
          n_de_LA: Enter
          Parella: TAULA [2] DE Caracter
     FVAR

     EscriuFrase(" Aquest programa compte el n. de LAs")
     EscriuFrase(" que hi ha en un text. Nomes cal entrar")
     EscriuFrase(" el text acabat en un punt i el programa")
     EscriuFrase(" ja treu el resultat.")

     n_de_LA:= 0
     Parella[0] = ' '
     LlegirCaracter(Parella[1])
     MENTRE (Parella[1] ≠ '.') FER
          SI ((Parella[0] = 'L') AND (Parella[1] = 'A')) LLAVORS
               n_de_LA:= n_de_LA + 1
          FSI
          Parella[0] = Parella[1]
          LlegirCaracter(Parella[1])
     FMENTRE

     EscriuFrase("El nombre de LAs del text es: )
     EscriuEnter(n_de_LA)
FALGORISME
```

```
/*******************************************************/
/* programa per comptar el nombre de LAs d'un text, v2 */
/*******************************************************/
#include <stdio.h>

main()
{
     /* declaracio de variables */
     int n_de_LA;
     char parella[2];

     /* presentacio del programa */
     printf("\n\n Aquest programa compte el n. de LAs\n");
     printf(" que hi ha en un text. Nomes cal entrar\n");
     printf(" el text acabat en un punt i el programa");
     printf(" ja treu el resultat.\n\n");

     /* iniciar tractament */
     n_de_LA = 0;

     /* accedir a la primera parella */
     parella[0] = ' ';
     parella[1] = getchar();
     while (parella[1] != '.') /* darrera parella */ {
          if ((parella[0] == 'L') && (parella[1] == 'A')) {
               n_de_LA = n_de_LA + 1; /* tractar la parella */
          }
          /* seguent parella */
          parella[0] = parella[1];
          parella[1] = getchar();
     }

     /* sortida */
     printf("\n\nEl nombre de LAs del text es: %d\n\n\n", n_de_LA);
}
```

## 3.6 RECOMPTE DE FREQÜENCIA DE LLETRES

Dissenyar un algorisme per fer el recompte de la freqüència de les diferents lletres d'un text acabat en un punt.

```
ALGORISME FrequenciaLletres
     VAR
          Frequencia: TAULA [26] DE Enter
          i: Enter
          c: Caracter
     FVAR

     Escriu("Aquest programa compte la frequencia")
     Escriu("de les lletres que hi ha en un text.")
     Escriu("Nomes cal entrar el text acabat en un punt ")
     Escriu("i el programa ja treu el resultat.")
     Escriu("ATENCIO! Cal entrar el text en majuscules.")

     i:= 0
     MENTRE (i < 26) FER
          Frequencia[i]:= 0
          i:= i+1
     FMENTRE

     LlegirCaracter(c)
     MENTRE (c ≠ '.') FER
          SI (c ≥ 'a') AND (c ≤ 'z') LLAVORS
               c:= Caracter(Ordinal(c) - Ordinal('a') + Ordinal('A'))
          FSI
          SI (c ≥ 'A') AND (c ≤ 'A') LLAVORS
               Frequencia[Ordinal(c)]:= Frequencia[Ordinal(c)] + 1
          FSI
          LlegirCaracter(c)
     FMENTRE

     i:= 0
     MENTRE (i < 26) FER
          EscriuFrase("La frequencia de la lletra ")
          EscriuCaracter(Caracter(i + Ordinal('A'))
          EscriuFrase(" es ")
          EscriuEnter(Frequencia[i])
     FMENTRE
FALGORISME
```

```
/*************************************************************/
/* programa per comptar la frequencia de les lletres d'un text */
/*************************************************************/

#include <ctype.h> /* permet simplificar algunes espressions */
#include <stdio.h>
#include <string.h>

typedef enum {fals = 0, cert = 1} boolea;

main()
{
     /* declaracio de variables */
     int frequencia[26], i;
     char c;

     /* presentacio del programa */
     printf("\n\nAquest programa compte la frequencia\n");
     printf("de les lletres que hi ha en un text.\n");
     printf("\n\nNomes cal entrar el text acabat en un punt ");
     printf("\ni el programa ja treu el resultat.\n\n");
     printf("\n\n ATENCIO! Cal entrar el text en majuscules");
     printf("\n i al final pitxar el retorn (despres del punt)\n\n");

     /* proces de recorregut */

     /* iniciar tractament */
     for (i=0; i<26; i++)
     {
          frequencia[i] = 0;
     }

     /* accedir al primer element */
     c = getchar();
     while (c != '.') {
          if (islower(c))
               c = toupper(c);
          if (isupper(c))
               frequencia[c-'A']++;
          /* seguent element */
          c = getchar();
     }

     /* finalitzar el tractament */
     printf("\n\n");
     for (i=0; i<26; i++)
     {
          printf("La frequencia de la lletra %c es %d.\n",
                    i+'A',frequencia[i]);
     }
}
```

## 3.7 PALINDROMS

Dissenyar un algorisme que determini si un cert text és o no un palíndrom.

```
ALGORISME TrobarPalindrom
     CONST
          MidaMaximaFrase = 100
     FCONST

     TIPUS
          Frase: TAULA [MidaMaximaFrase] DE Caracter
     FTIPUS

     VAR
          FraseLlegida: Frase
          i, j, nLletresFrase: Enter
          Lletra, Lletra1, Lletra2: Caracter
          LletresIguals: Boolea
     FVAR

     EscriuFrase("Aquest programa determina si una frase")
     EscriuFrase("acabada en punt es o no un palindrom.")

     nLletresFrase:= 0
     EscriuFrase("Entra la frase a comprovar:")
     LlegirCaracter(Lletra)
     MENTRE (Lletra ≠ '.') FER
          FraseLlegir[nLletresFrase]:= Lletra
          nLletresFrase:= nLletresFrase + 1
     FMENTRE

     LletresIguals = Cert
     i:= 0
     j:= nLletresFrase
     MENTRE ((i<j) AND (LletresIguals)) FER
          Lletra:= FraseLlegida[i]
          MENTRE (NO (((Lletra ≥ 'A') AND (Lletra ≤ 'Z')) OR
                      ((Lletra ≥ 'a') AND (Lletra ≤ 'z')))) FER
               Lletra:= FraseLlegida[i]
               i:= i+1
          FMENTRE
          Lletra:= FraseLlegida[j]
          MENTRE (NO (((Lletra ≥ 'A') AND (Lletra ≤ 'Z')) OR
                      ((Lletra ≥ 'a') AND (Lletra ≤ 'z')))) FER
               Lletra:= FraseLlegida[j]
               j:= j-1
          FMENTRE
          Lletra1:= FraseLlegida[i]
          Lletra2:= FraseLlegida[j]
```

```
          SI ((Lletra1 ≥ 'A') AND (Lletra1 ≤ 'Z')) LLAVORS
               Lletra1:= Caracter(Ordinal(Lletra1) + 32)
          FSI
          SI ((Lletra1 ≥ 'A') AND (Lletra1 ≤ 'Z')) LLAVORS
               Lletra2:= Caracter(Ordinal(Lletra1) + 32)
          FSI
          LletresIguals:= (Lletra1 = Lletra2)
          i:= i + 1
          j:= j - 1
     FMENTRE

     SI (LletresIguals) LLAVORS
          EscriuFrase("La frase es un palindrom")
     ALTRAMENT
          EscriuFrase("La frase no es un palindrom")
     FSI
FALGORISME
```

```
/*****************************************************/
/* programa per cercar si una cadena es un palindron */
/*****************************************************/

#include <ctype.h>
#include <stdio.h>
#include <string.h>

#define MIDA_MAXIMA_FRASE 100

typedef char frase[MIDA_MAXIMA_FRASE];
typedef enum {Fals = 0,Cert = 1} boolea;

main()
{
     frase frase_llegida;
     int i, j, n_lletres_frase;
     char lletra, lletra1, lletra2;
     boolea lletres_iguals;

     /* presentacio */
     printf("\n");
     printf("Aquest programa determina si una frase \n");
     printf("acabada en punt es o no un palindrom.\n\n");

     /* llegir la frase */
     n_lletres_frase = 0;
     printf("Entra la frase a comprovar:\n");
     gets(frase_llegida);
     n_lletres_frase = strlen(frase_llegida);

     /* determinar si es palindrom */
     lletres_iguals = Cert;
     i = 0;
     j = n_lletres_frase - 1;
     while ((i<j) && (lletres_iguals))
     {
          while (!isalpha(frase_llegida[i]))
               i++;
          while (!isalpha(frase_llegida[j]))
               j--;
          lletra1 = frase_llegida[i];
          lletra2 = frase_llegida[j];
          if (isupper(lletra1))
               lletra1 = tolower(lletra1);
          if (isupper(lletra2))
               lletra2 = tolower(lletra2);
          lletres_iguals = (lletra1 == lletra2);
          i++;
          j--;
     }

     /* sortida de resultats */
```

```
      if (lletres_iguals)
      {
            printf("\nLa frase es un palindrom\n");
      }
      else
      {
            printf("\nLa frase no es un palindrom\n");
      }
}
```

## 3.8 DIBUIX D'UNA FIGURA

Fer un algorisme per dibuixar un romb que tingui per amplada un cer valor entrat per teclat. Utilitzar accions i funcions per fer el disseny descendent.

```
ALGORISME Figura
     VAR
          NEstrelles: Enter
     FVAR

     ACCIO Presentacio
     FUNCIO EntrarParell RETORNA Enter
     ACCIO EscriureCaracter(Caracter, Enter)
     ACCIO TriangleSuperior(Enter)
     ACCIO TriangleInferior(Enter)

     Presentacio
     NEstrelles:= EntrarParell
     TriangleSuperior(NEstrelles)
     TriangleInferior(NEstrelles)
FALGORISME


ACCIO Presentacio
     EscriuFrase("Aquest programa presenta una")
     EscriuFrase("figura a la pantalla. La seva mida ")
     EscriuFrase("depen d'un valor enter entrat per teclat.")
FACCIO


FUNCIO EntrarParell RETORNA Enter
     VAR
          ValorEntrat: Enter
     FVAR

     EscriuFrase("De quina mida vols el dibuix?")
     LlegirEnter(ValorEntrat)
     MENTRE (valor_entrat < 0) FER
          EscriuFrase(" Valor incorrecte.")
          EscriuFrase(" Ha de ser mes gran o igual a 0")
          EscriuFrase(" De quina mida vols el dibuix?")
          LlegirEnter(ValorEntrat)
     FMENTRE
     SI ((ValorEntrat MOD 2) = 0) LLAVORS
          ValorEntrat:= ValorEntrat + 1
     FSI
     RETORNA(valor_entrat)
FACCIO
```

```
ACCIO EscriureCaracter(c: Caracter, NVegades: Enter)
     VAR
          Index: Enter
     FVAR

     PER Index DE 1 FINS NVegades FER
          EscriuCaracter(c)
     FPER
FACCIO


ACCIO TriangleSuperior(NEstrelles: Enter)
     VAR
          IndexLinia, NEspais: Enter
     FVAR

     IndexLinia:= 1
     MENTRE (IndexLinia <= NEstrelles) FER
          NEspais:= (NEstrelles - IndexLinia) DIV 2
          EscriureCaracter(' ', NEspais)
          EscriureCaracter('*', IndexLinia)
          SaltarLinia
          IndexLinia:= IndexLinia + 2
     FMENTRE
FACCIO

ACCIO TriangleInferior(NEstrelles: Enter)
     VAR
          IndexLinia, NEspais: Enter
     FVAR

     IndexLinia:= NEstrelles - 2
     MENTRE (IndexLinia > 0) FER
          NEspais = (NEstrelles - IndexLinia) / 2
          EscriureCaracter(' ', NEspais)
          EscriureCaracter('*', IndexLinia)
          SaltarLinia
          IndexLinia = IndexLinia - 2
     FMENTRE
FACCIO
```

```
/*****************************************************************/
/*          programa per fer una figura                          */
/*****************************************************************/

#include <stdio.h>

void presentacio(void);
int entrar_parell(void);
void escriure_caracter(char, int);
void triangle_superior(int);
void triangle_inferior(int);

main()
{
     int n_estrelles;

     presentacio();
     n_estrelles = entrar_parell();
     triangle_superior(n_estrelles);
     triangle_inferior(n_estrelles);
}

void presentacio(void)
{
     printf("\n\n Aquest programa presenta una \n");
     printf("figura a la pantalla. La seva mida \n");
     printf("depen d\'un valor enter entrat per teclat.\n");
}

int entrar_parell(void)
{
     int valor_entrat;

     printf("\n De quina mida vols el dibuix?\n");
     scanf("%d", &valor_entrat);
     while (valor_entrat < 0)
     {
          printf("\n Valor incorrecte.\n");
          printf(" Ha de ser mes gran o igual a 0\n");
          printf("\n De quina mida vols el dibuix?\n");
          scanf("%d", &valor_entrat);
     }
     if ((valor_entrat % 2) == 0)
     {
          valor_entrat = valor_entrat + 1;
     }
     return(valor_entrat);
}

void escriure_caracter(char c, int n_vegades)
{
     int index;
```

```
     for (index = 0; index < n_vegades; index++)
     {
          printf("%c", c);
     }
}

void triangle_superior(int n_estrelles)
{
     int index_linia, n_espais;

     index_linia = 1;
     while (index_linia <= n_estrelles)
     {
          n_espais = (n_estrelles - index_linia) / 2;
          escriure_caracter(' ', n_espais);
          escriure_caracter('*', index_linia);
          printf("\n");
          index_linia = index_linia + 2;
     }
}

void triangle_inferior(int n_estrelles)
{
     int index_linia, n_espais;

     index_linia = n_estrelles - 2;
     while (index_linia > 0)
     {
          n_espais = (n_estrelles - index_linia) / 2;
          escriure_caracter(' ', n_espais);
          escriure_caracter('*', index_linia);
          printf("\n");
          index_linia = index_linia - 2;
     }
}
```