

Tema 2: La classe `string`

Programació 2

Grau en Enginyeria Informàtica
Universitat d'Alacant
Curs 2024-2025



1. Cadenes de caràcters en C
2. La classe `string` en C++
3. Conversions de tipus
4. Comparativa
5. Exercicis

Cadenes de caràcters en C

Declaració (1/3)

- Les *cadena de caràcters* són vectors que contenen una seqüència de tipus `char` acabada amb el caràcter nul (`'\0'`):

```
// El compilador fica el '\0' al final automàticament  
char cad[]="hola";  
// Una altra manera d'inicialitzar, caràcter a caràcter  
char cad[]={ 'h', 'o', 'l', 'a', '\0' };  
// Falta el '\0': no és una cadena de caràcters vàlida  
char cad[]={ 'h', 'o', 'l', 'a' };
```

- Moltes de les funcions* que treballen amb cadenes busquen el `'\0'` per a saber on acaba la cadena
- Si no tenim el `'\0'` pot ser que el resultat d'aquestes funcions no siga l'esperat

*Com aquelles que pertanyen a la llibreria `cstring` i que veurem més endavant

Declaració (2/3)

- Les cadenes de caràcters en C tenen grandària fixa i una vegada declarades no poden canviar de grandària:

```
char cad[10]; // Emmagatzema com a màxim 10 elements
```

- Cal tindre en compte que s'ha de reservar sempre un espai per a emmagatzemar el caràcter nul ('\\0'):

```
char cad[10]; // Emmagatzema com a màxim 9 lletres i el '\\0'
```

- Es poden inicialitzar en declarar-les, i en aquest cas no cal posar la grandària:

```
char cad[]="hola"; // Grandària 5 (4 lletres + '\\0')  
char cad2[10]="hola"; // Grandària 10, encara que només  
ocupa 5
```

- Les cadenes de caràcters en C es poden usar també en C++

Declaració (3/3)

- Errors comuns en declarar cadenes de caràcters:

```
// El vector és massa xicotet per a guardar la cadena  
char cad[5]="parallelepiped"; // Error de compilació
```

```
// S'usen cometes simples (') en lloc de dobles (")  
char cad[]='h'; // Error de compilació  
char cad[]='hola'; // Error de compilació
```

```
// No es posa la grandària i no s'inicialitza  
char cad[]; // Error de compilació
```

```
// S'intenta assignar valor amb '=' després de declarar  
char cad[10];  
cad="hola"; // Error de compilació
```

Eixida per pantalla

- Eixida per pantalla amb `cout` i `cerr` com la resta de tipus simples (`int`, `float`, etc.)
- Podem combinar en l'eixida variables, constants i dades de diferent tipus:

```
char cad[]="Nota";  
int num=10;  
  
cout << cad << " -> " << num; // Mostra "Nota -> 10"
```

Entrada per teclat > Operador >> (1/2)

- Podem llegir una cadena de caràcters des de teclat com amb altres tipus simples, utilitzant `cin` i l'operador `>>`
- Existeixen algunes diferències a l'hora de llegir des de teclat respecte a altres tipus de dades
- Ignora els blancs* abans de la cadena:

```
char cad[32];  
cin >> cad;  
// L'usuari escriu "  hola"  
// La variable cad emmagatzema "hola"
```

*Entenem per "blanc" un espai, tabulador o salt de línia (' \n ')

Entrada per teclat > Operador >> (2/2)

- Acaba de llegir quan troba el primer blanc en la cadena. **No ens permet llegir sencera una cadena que continga blancs:**

```
char cad[32];  
cin >> cad;  
// L'usuari escriu "bona vesprada"  
// La variable cad emmagatzema "bona"
```

- No limita el nombre de caràcters que es lligen. **L'usuari pot escriure una cadena més gran del que admet el vector:**

```
char cad[5];  
cin >> cad;  
// L'usuari escriu "esternoclidomastoidal"  
// Pot envair zones de memòria que no deuria i  
// produir una fallada de segmentació
```

Entrada per teclat > getline (1/4)

- També podem llegir una cadena de caràcters de teclat mitjançant `cin` i la funció `getline`
- Aquesta funció permet llegir cadenes amb blancs i limitar el nombre de caràcters llegits:

```
const int TAM=100;
char cad[TAM];
// cad: variable on emmagatzemem la cadena
// TAM: nombre de caràcters a llegir
cin.getline(cad,TAM);
// Si l'usuari introdueix "bona vesprada"
// en cad s'emmagatzema "bona vesprada"
```

- Llig com a màxim `TAM-1` caràcters o fins que arribi al final de línia
- El `'\n'` del final de línia es llig però no es guarda en la cadena
- La funció afig `'\0'` al final del que ha llegit (per això només llig `TAM-1` caràcters)

Entrada per teclat > getline (2/4)

- Si l'usuari introdueix més caràcters dels que caben, aquests es queden en el *buffer* de teclat i la següent lectura falla:

```
char cad[10];  
cout << "Cadena 1: ";  
cin.getline(cad,10);  
cout << "Llegit 1: " << cad << endl;  
cout << "Cadena 2: ";  
cin.getline(cad,10);  
cout << "Llegit 2: " << cad << endl;
```

Terminal

```
$ elMeuPrograma  
Cadena 1: hola a tothom  
Llegit 1: hola a to  
Cadena 2: Llegit 2:
```

Entrada per teclat > getline (3/4)

- Poden haver-hi problemes quan llegim de `cin` combinant l'operador `>>` i la funció `getline`:

```
int num;  
char cad[100];  
  
cout << "Num: ";  
cin >> num;  
cout << "Escriu una cadena: " ;  
cin.getline(cad,100);  
cout << "El que he llegit és: " << cad << endl;
```

Terminal

```
$ elMeuPrograma  
Num: 10  
Escriu una cadena: El que he llegit és:
```

Entrada per teclat > getline (4/4)

- Per què succeeix això?
 - Amb l'operador >> es llig 10, però es deixa de llegir quan es troba el primer caràcter no numèric ('\\n' en aquest cas)
 - El primer que troba en el *buffer* la funció `getline` és un '\\n', per la qual cosa acaba de llegir i no guarda res en `cad`
- Solució:

```
...  
cin >> num;  
cin.ignore(); // Afegim aquesta línia  
              // Saca el '\\n' del buffer  
// Ja es pot usar getline sense problema  
...
```

La llibreria `cstring` (1/3)

- La llibreria `cstring` conté una sèrie de funcions que faciliten el treball amb cadenes de caràcters
- Per a poder utilitzar-la cal incloure la llibreria en el codi:

```
#include <cstring>
```

- `strlen` retorna la longitud (nombre de caràcters) d'una cadena:

```
char cad[10]="adeu";  
cout << strlen(cad); // Imprimeix 4
```

- `strcpy` còpia una cadena en una altra. **Cal anar amb cura de no superar la grandària del vector de destinació:**

```
char cad[5];  
strcpy(cad,"hola"); // Cap: 4 + '\0' = 5 caràcters  
strcpy(cad,"adeu!") // No cap!! Violació de segment
```

La llibreria `cstring` (2/3)

- `strcmp` compara dues cadenes en ordre lexicogràfic*, retornant 1 si `cad1 > cad2`, 0 si `cad1 == cad2` i -1 si `cad1 < cad2`:

```
char cad1[]="adios";  
char cad2[]="adeu";  
cout << strcmp(cad1,cad2) << endl; // Imprimeix 1  
cout << strcmp(cad2,cad1) << endl; // Imprimeix -1  
cout << strcmp(cad1,cad1) << endl; // Imprimeix 0
```

- `strcat` afegeix el contingut d'una cadena al final d'una altra. **Ha d'haver-hi prou espai en la cadena destí:**

```
char cad[10]="hola";  
strcat(cad," mu"); // En total 9 caràcters (cabe)  
strcat(cad,"ndo"); // Afegeix 3 més (ja no cap!)
```

*Ordre que segueixen les paraules en un diccionari

La llibreria `cstring` (3/3)

- Les funcions `strncmp`, `strncpy` i `strncat` comparen, copien o concatenen només els `n` primers caràcters:

```
char cad[8];  
strncpy(cad, "hola, mon", 4); // Només copia "hola"  
cad[4]='\0'; // No afegeix el '\0' de manera automàtica  
// L'hem d'afegir nosaltres al final
```

```
char cad1[8]="adios";  
char cad2[8]="adeu";  
// Només compara els dos primers caràcters  
cout << strncmp(cad1, cad2, 2) << endl; // Imprimeix 0
```

```
char cad1[50]="Hola, ";  
char cad2[]="mon meravellos";  
strncat(cad1, cad2, 3); // cad1 valdrà "Hola, mon"
```


Conversió a int i float

- Per a passar una cadena de caràcters a `int` o `float` es poden usar les funcions `atoi` o `atof`
- Aquestes funcions pertanyen a la llibreria `cstdlib`:

```
#include <cstdlib> // Sempre que s'usen les funcions

char cad[]="100";
int num=atoi(cad); // num val 100

char cad2[]="10.5";
float num2=atof(cad2); // num2 val 10.5
```

La classe `string` en C++

Definició (1/2)

- En C++ es poden usar les cadenes de caràcters en C, però a més compta amb la classe* `string` que permet treballar de manera més còmoda i flexible amb cadenes de caràcters:

```
// Declaració d'una variable de tipus string  
string s; // No cal indicar la grandària de la cadena  
// Declaració amb inicialització  
string s2="Alacant";  
// Declaració d'una constant  
const string SALUTACIO="hola";
```

*Més informació sobre el que és una “classe” en el Tema 5

Definició (2/2)

- Un `string` té grandària variable i pot créixer en funció de les necessitats d'emmagatzematge del programa:

```
string s="hola"; // Emmagatzema 4 caràcters  
s="hola a tothom"; // Emmagatzema 13 caràcters*  
s="ok"; // Emmagatzema 2 caràcters
```

- No cal preocupar-se del `'\0'`
- El pas de paràmetres (valor i referència) es fa com amb qualsevol tipus simple:

```
void laMeuaFuncio(string s1,string &s2){  
    // s1 es passa per valor  
    // s2 es passa per referència  
}
```

*Un espai en blanc compta com un caràcter més

- Eixida per pantalla amb `cout` i `cerr` igual que amb els vectors de caràcters en C:

```
string s="Nota";  
int num=10;  
  
cout << s << " -> " << num; // Mostra "Nota -> 10"
```

Entrada per teclat > Operador >>

- Es pot llegir de teclat amb `cin` i l'operador `>>` de la mateixa forma que amb vectors de caràcters en C
- Ignora els blancs abans de la cadena i acaba de llegir quan troba el primer blanc:

```
string s;  
cin >> s;  
// L'usuari escriu "    hola"  
// La variable s emmagatzema "hola"  
...  
// L'usuari escriu "bona vesprada"  
// La variable s emmagatzema "bona"
```

Entrada per teclat > getline (1/2)

- Igual que amb els vectors de caràcters en C, podem usar la funció `getline` per a llegir cadenes
- Permet llegir cadenes que continguin blancs:

```
string s;  
getline(cin,s);  
// Si l'usuari introdueix "bona vesprada"  
// en s s'emmagatzema "bona vesprada"
```

- No limita el nombre de caràcters que es llegeixen, ja que amb un `string` no és necessari
- **Compte!** Canvia la sintaxi respecte als vectors de caràcters en C

Entrada per teclat > getline (2/2)

- Si combinem lectures amb l'operador >> i `getline` tenim el mateix problema que amb els vectors de caràcters en C*
- Per defecte, `getline` llig fins que troba el caràcter salt de línia (`'\n'`)
- Podem passar-hi un paràmetre addicional per a indicar que lligi fins a un determinat caràcter:

```
string s;  
// Llig fins que troba la primera coma  
getline(cin,s,',');  
// Llig fins que troba el primer claudàtor  
getline(cin,s,'[');
```

*La solució és la mateixa que en la transparència 11

Extraure paraules d'una string

- Es poden extraure paraules fàcilment d'una string usant la classe stringstream:

```
#include <sstream> // Necessari si s'usa stringstream
...
stringstream ss("Hola mon cruel 666");
string s;

// En cada iteració del bucle llig fins a trobar blanc
while(ss>>s){ // Extraiem les paraules una a una
    cout << "Paraula: " << s << endl;
}
```

Mètodes de string (1/3)

- Per ésser una classe, els mètodes s'invoquen posant un punt després del nom de la variable
- `length` retorna el nombre de caràcters de la cadena:

```
// unsigned int length()  
string s="hola, mon";  
cout << s.length(); // Imprimeix 9
```

- `find` retorna la posició en la qual apareix una subcadena dins d'una cadena:

```
// size_t find(const string &s,unsigned int pos=0)  
cout << s.find("mon"); // Imprimeix 6  
// Si no troba la subcadena retorna string::npos
```

Mètodes de string (2/3)

- `replace` substitueix una cadena (o part d'ella) per una altra:

```
// string& replace(unsigned int pos,unsigned int grand,  
    const string &s)  
string s="hola mon";  
s.replace(0,4,"hello"); // s val "hello mon"
```

- `erase` permet eliminar part d'una cadena:

```
// string& erase(unsigned int pos=0,unsigned int grand=  
    string::npos);  
string cad="hola mon";  
cad.erase(4,3); // cad vale "holan"
```

- `substr` retorna una subcadena de la cadena original:

```
// string substr(unsigned int pos=0,unsigned int tam=  
    string::npos) const;  
string cad="hola mon";  
string subcad=cad.substr(2,5); // subcad val "la mo"
```

Mètodes de string (3/3)

- Exemple d'ús:

```
string a="Hi ha un coco en aquesta cuina amb cocos";
string b="coco";
unsigned int tam=a.length(); // Longitud de a
// Busquem la primera paraula "coco"
size_t trobat=a.find(b);
if(trobat!=string::npos){
    cout << "Primera en: " << trobat << endl;
    // Busquem la segona paraula "coco"
    trobat=a.find(b,trobat+b.length());
    if(trobat!=string::npos)
        cout << "Segona en: " << trobat << endl;
}
else{
    cout << "Paraula '" << b << "' no trobada";
}
// Substituïm el primer "coco" per "albercoc"
a.replace(a.find(b),b.length(),"albercoc");
cout << a << endl;
```

Operadors (1/2)

- Comparacions: == (igual), != (diferent), > (major estricta), >= (major o igual), < (menor estricta) i <= (menor o igual)

```
string s1,s2;  
cin >> s1; cin >> s2;  
if(s1==s2) // La comparació és en ordre lexicogràfic  
    cout << "Són iguals" << endl;
```

- Assignació d'una cadena a una altra amb l'operador =, com qualsevol tipus simple:

```
string s1="hola";  
string s2;  
s2=s1;
```

- Concatenació de cadenes amb l'operador +:

```
string s1="hola";  
string s2="mon";  
string s3=s1+", "+s2; // s3 val "hola, mon"
```

Operadors (2/2)

- Accés a components com si fóra un vector de caràcters en C, amb l'operador []:

```
string s="hola";  
char c=s[3]; // s[3] val 'a'  
s[0] = 'H';  
cout << s << ":" << c << endl ; // Imprimeix "Hola:a"
```

- No es poden assignar caràcters en posicions que no pertanyen al string:

```
string s;  
s[0]='h'; s[1]='o'; s[2]='l'; s[3]='a';  
// No emmagatzema res, perquè s és una cadena buida i  
aquestes posicions no les té reservades
```

- Exemple de recorregut d'un string caràcter a caràcter:

```
string s="hola, mon";  
for(unsigned int i=0;i<s.length(); i++)  
    s[i]='f'; // Substitueix cada caràcter per 'f'
```

Conversions de tipus

Conversió entre `string` i vectors de caràcters en C

- Per a assignar un vector de caràcters en C a `string` s'utilitza l'operador d'assignació (=):

```
char cad[]="hola";  
string s;  
s=cad;
```

- Per a assignar un `string` a un vector de caràcters en C cal usar `strcpy` i `c_str`.*

```
char cad[10];  
string s="mon";  
// Ha d'haver-hi espai suficient en cad  
strcpy(cad,s.c_str());
```

*El mètode `c_str` retorna un vector de caràcters en C amb el contingut del `string`

Conversió entre `string` i nombre

- Convertir un nombre enter o real a `string` :

```
#include <string>
...
int num=100;
string s=to_string(num);
```

- Convertir un `string` a nombre enter:*

```
string s="100";
int num=stoi(s);
```

- Convertir un `string` a nombre real:

```
string s="10.5";
float num=stof(s);
```

*Les funcions `to_string`, `stoi` i `stof` estan disponibles a partir de la versió 2011 de C++

Comparativa

Vector de caràcters en C vs. string

Vector de caràcters en C	string
<pre>char cad[TAM]; char cad[]="hola"; strlen(cad) cin.getline(cad,TAM); if(!strcmp(cad1,cad2)){...} strcpy(cad1,cad2); strcat(cad1,cad2); strcpy(cad,s.c_str());</pre>	<pre>string s; string s="hola"; s.length() getline(cin,s); if(s1==s2){...} s1=s2; s1=s1+s2; s=cad;</pre>
Acaben amb '\0'	No acaben amb '\0'
Grandària reservada fixa	La grandària reservada pot variar
Grandària ocupada variable	Grandària ocupada == grandària reservada
S'usen amb fitxers binaris	No es poden usar amb fitxers binaris

Exercicis

Exercicis (1/4)

Exercici 1

Dissenyeu una funció `subcadena` que retorne la subcadena de longitud `n` que comença en la posició `p` d'una altra cadena. Tant l'argument com el valor de retorn han de ser de tipus `string`.

```
subcadena("hoooola",2,5) // Retorna "la"
```

Exercici 2

Dissenyeu una funció `esborraCaracterCadena` que, donats un `string` i un caràcter, esborre totes les aparicions del caràcter en el `string` i el retorne.

```
esborraCaracterCadena("cocobongo",'o') // Retorna "ccbng"
```

Exercici 3

Dissenyeu una funció `buscarSubcadena` que busque la primera aparició d'una subcadena `a` dins d'una cadena `b` i retorne la posició, o `-1` si no hi és. Tant `a` com `b` han de ser de tipus `string`.

```
buscarSubcadena("ool", "hoooola") // Retorna 2
```

Ampliacions:

1. Afegir un altre paràmetre a la funció que indique el número d'aparició (si val 1 seria com la funció original)
2. Crear una altra funció que retorne el nombre d'aparicions de la subcadena en la cadena

Exercicis (3/4)

Exercici 4

Dissenyeu una funció `codifica` que codifiqui una cadena sumant una quantitat `n` al codi ASCII de cada caràcter, però tenint en compte que el resultat ha de ser una lletra.

Per exemple, si `n=3`, l'`a` es codifica com `d`, la `b` com `e`,..., la `x` com `a`, la `y` com `b`, i la `z` com `c`.

La funció ha d'admetre lletres majúscules i minúscules. Els caràcters que no siguin lletres no s'han de codificar. L'argument ha de ser de tipus `string`.

```
codifica("hola, mundo",3) // Retorna "krod, pxqgr"
```

Exercicis (4/4)

Exercici 5

Dissenyeu una funció `esPalindrom` que retorne `true` si el `string` que se li passa com a paràmetre és palíndrom.

```
esPalindrom("elboplaualpoble") // Retorna true  
esPalindrom("hola, aloh") // Retorna false
```

Exercici 6

Dissenyeu una funció `crearPalindrom` que afegisca a un `string` el mateix `string` invertit, de manera que el resultat siga un palíndrom.

```
crearPalindrom("hola") // Retorna "holaaloh"
```