

Tema 2: La classe `string`

Programació 2

Grau en Enginyeria Informàtica
Universitat d'Alacant
Curs 2025-2026



1. La classe `string` en C++
2. Vectors STL
3. Arguments del programa
4. Exercicis

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 lligin, 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 a la diapositiva 44 del Tema 1

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'
```

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++

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

Vectors STL

Vectors STL (1/3)

- La *Standard Template Library* (STL) és una llibreria de funcions per a C++
- Proporciona diferents estructures de dades i algorismes
- Inclou la classe `vector`, que permet emmagatzemar elements de qualsevol tipus, com un vector normal, però sense haver de preocupar-nos de la grandària:

```
#include <vector> // Sempre que usem vector
vector<int> vec; // Declara un vector d'enters
                // No és necessari indicar la grandària
```

- La grandària inicial d'un vector STL és 0 i creix de manera dinàmica en funció de les necessitats
- Per a afegir elements al final del vector usem `push_back`:*

```
vec.push_back(12); // Afig 12 al final del vector
vec.push_back(8); // Afig 8 darrere del 12
```

*Com és una classe, els mètodes s'invoquen posant un punt després del nom de la variable

Vectors STL (2/3)

- Accés a elements mitjançant l'operador []:

```
vec[10]=23; // Igual que un vector convencional  
cout << vec[8] << endl;
```

- Amb `size` obtenim el nombre d'elements del vector:

```
// Recorrem tots els elements del vector  
for(unsigned int i=0;i<vec.size();i++){  
    vec[i]=10;  
}
```

- Recorregut de vectors basat en rang:

```
// Recorrem tots els elements del vector  
for(int num:vec){  
    cout << num << endl;  
}
```

Vectors STL (3/3)

- Amb `clear` podem esborrar tots els elements i amb `erase` un en concret:

```
vec.erase(vec.begin()+3); // Elimina el quart element  
vec.clear(); // Elimina tots els elements del vector
```

- Error habitual: **no es poden guardar elements en posicions que no pertanyen al vector**

```
vector<int> vec;  
vec[0]=78; vec[1]=9; vec[2]=17;  
for(int i=0;i<vec.size();i++){  
    cout << vec[i] << endl; // No imprimeix res  
}
```

- Existeixen moltes altres funcions per treballar amb vectors STL*

*Més informació a <http://www.cplusplus.com/reference/vector/vector/>

Arguments del programa

Arguments del programa (1/4)

- Els *arguments* d'un programa s'usen per a proporcionar-hi informació (normalment opcions) des de línia d'ordres
- Aquest ús és molt habitual i permet modificar el comportament del programa:

Terminal

```
$ ls          // Mostra els fitxers d'un directori
$ ls -a       // Mostra també els fitxers ocults (opció "-a")
$ ls -a -l    // Afig informació extra de cada fitxer (opció "-l")
```


Arguments del programa (2/4)

- El `main` és una funció i per tant pot rebre dos paràmetres: `argc` i `argv`
- Aquests paràmetres permeten gestionar el pas d'arguments per línia d'ordres al programa:

```
// Sempre en aquest ordre  
int main(int argc, char *argv[]){  
    ...  
    return 0;  
}
```

- `int argc`: nombre d'arguments passats al programa (comptant també el nom del programa)
- `char *argv[]`: vector de cadenes de caràcters amb els arguments passats al programa

Arguments del programa (3/4)

- Exemple d'ús:

```
int main(int argc, char *argv[]){  
    for(int i=0; i<argc; i++){  
        cout << "Arg. " << i << " : " << argv[i] << endl;  
    }  
}
```

Terminal

```
$ ./elMeuPrograma -a -h X    // Exemple de crida amb tres paràmetres  
Arg. 0 : ./elMeuPrograma  
Arg. 1 : -a  
Arg. 2 : -h  
Arg. 3 : X
```

- Els arguments no han de començar amb un guió (-) necessàriament però és una pràctica prou habitual

Arguments del programa (4/4)

- Sembla fàcil gestionar els arguments del programa, però de vegades pot ser complicat
- L'usuari no sempre usa el mateix ordre a l'hora d'introduir els arguments:

Terminal

```
$ g++ -Wall -o prog prog.cc -g  
$ g++ -g -Wall prog.cc -o prog
```

- Pot haver-hi errors en la introducció i cal mostrar missatges d'ajuda a l'usuari
- És recomanable usar una funció a banda per a gestionar els arguments

Exercicis

Exercicis (1/5)

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 (3/5)

Exercici 4

Dissenyeu una funció `codifica` que codifique 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 (4/5)

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"
```


Exercici 7

Implementa un programa que continga una funció amb el següent prototip: `int primeNumber(int n)`. Aquesta funció tornarà el n -èsim nombre primer. El programa haurà d'imprimir nombres primers per pantalla amb les següents opcions:

- `-L` imprimir cada nombre en una línia diferent (per defecte s'imprimeixen tots a la mateixa línia)
- `-N n` imprimir els n primers nombres primers (per defecte 10)

Exemples d'execució:

Terminal

```
$ primes -N 5
1 2 3 5 7
$ primes -N -L 5
Error: primes [-L] [-N n]
```