# Stringhe e array multidimensionali

Lezione 11 di Programmazione 2

Docente: Giuseppe Scollo

Università di Catania, sede di Comiso (RG)
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Studi in Informatica applicata, AA 2006-7

**Indice**

## proprietà delle stringhe

**proprietà algebriche astratte:**

- $A^*$ : monoide generato dall'alfabeto (di solito finito) $A$
  - monoide **generato**: ogni elemento di $A$ è in $A^*$
  - operazione binaria associativa: la **concatenazione** di stringhe
  - elemento **neutro** rispetto alla concatenazione: la **stringa vuota λ**
- **altre operazioni** sulle stringhe:
  - **lunghezza** $|\_| : A^* \rightarrow N$
  - **eguaglianza** $\_==\_ : A^* \times A^* \rightarrow \{0,1\}$
  - **ordinamento lessicografico** (se l'alfabeto è ordinato)

**proprietà delle stringhe in C++:**

- stringhe di **caratteri**
  - non un tipo di dati generico, come array o liste, ovvero stringhe su un alfabeto qualsiasi, bensì **specifico**: l'alfabeto è l'insieme dei valori di tipo `char`
- rappresentate da (un puntatore a) un **array di caratteri**
  - di **lunghezza variabile:** il carattere **NUL** (codice: `0`) indica la fine della stringa

## operazioni sulle stringhe in C/C++

in C++, la **lunghezza** di una stringa è data dal numero di locazioni che occupa la sua rappresentazione come array **meno uno** (il carattere di fine stringa non si conta)

l'**eguaglianza** di due stringhe **non** è eguaglianza dei puntatori che le rappresentano, bensì identità delle sequenze di caratteri che le costituiscono

l'**ordinamento lessicografico** sulle stringhe è basato sull'ordine dei valori di tipo `char` (codice ASCII)

in C++ sono disponibili le operazioni della classe `String`, considerate in una lezione successiva, nonché quelle di libreria del linguaggio C, alcune delle quali sono esaminate ora:

confronto per eguaglianza e per ordinamento lessicografico sono realizzati da una stessa operazione, `strcmp(a,b)`, che restituisce un intero: `0` se le due stringhe sono uguali, positivo se `a>b` nell'ordine lessicografico, negativo altrimenti

sono inoltre disponibili funzioni di

- calcolo della **lunghezza** di una stringa: `strlen(a)`
- **copia** della stringa `b` nella stringa `a`: `strcpy(a,b)`
- **confronto lessicografico del prefisso** di lunghezza data: `strncmp(a,b,n)` opera come `strcmp(a,b)`, ma solo sui prefissi di lunghezza `n` delle due stringhe
- **concatenazione** di due stringhe: `strcat(a,b)`

## un'implementazione delle stringhe in C

le seguenti definizioni implementano in C alcune delle predette operazioni sulle stringhe: hanno però **effetti collaterali** e non sono del tutto affidabili... (v. esercizi sulle stringhe)

```
int strlen(char *a)
  { int i;
    for (i = 0; a[i] != 0; i++) ;
    return i;
  }

void strcpy(char *a, char *b)
  { for (int i = 0; (a[i] = b[i]) != 0; i++) ; }

int strcmp(char *a, char *b)
  { int i;
    for (i = 0; a[i] == b[i]; i++)
      if (a[i] == 0) return 0;
    return a[i] - b[i];
  }

void strcat(char *a, char *b)
  { strcpy(a+strlen(a), b); }
```

## ricerca su stringhe

il programma 3.15 nel testo (Sedgewick, 2003), p.114, trova le occorrenze di una stringa, data da linea di comando, in una stringa letta da standard input, e ne produce in output le posizioni iniziali in quest'ultima

```
#include <iostream>
using namespace std;
static const int N = 10000;
int main(int argc, char *argv[])
{ int i; char t;
  char a[N], *p = argv[1]; // p: puntatore alla stringa da linea di comando
  for (i = 0; i < N-1; a[i] = t, i++)
    if(!cin.get(t)) break; // lettura di un carattere da standard input
  a[i] = 0;                // fine stringa
  for (i = 0; a[i] != 0; i++)   // ciclo di scansione della stringa letta
  { int j;
    for (j = 0; p[j] != 0; j++) // ricerca della stringa *p a partire dalla
      if (a[i+j] != p[j]) break;// posizione i della stringa letta in a[]
    if (p[j] == 0) cout << i << " ";
  }
cout << endl;
}
```

## prestazioni dell'elaborazione di stringhe

nel programma precedente, la scansione della stringa letta è chiaramente superflua dalla posizione `strlen(a)-strlen(p)` in poi; inoltre, non si fa uso delle funzioni di libreria per le operazioni sulle stringhe: la direttiva per questo è:

```
#include <cstring>
```

l'uso delle funzioni di libreria può migliorare la compattezza del codice, ma occorre fare attenzione a non degradare le prestazioni; ad esempio, se si rimpiazza il ciclo di scansione della stringa letta con il seguente frammento di codice:

```
  for (i = 0; i < strlen(a)-strlen(p); i++)
  { if (strncmp(&a[i], p, strlen(p)) == 0) cout << i << " "; }
```

un tempo di esecuzione lineare della funzione di libreria `strlen` rende quello dell'algoritmo **quadratico** nella lunghezza della stringa letta

l'errore di prestazione in questo caso è dovuto all'invocazione della funzione di libreria, che restituisce sempre lo stesso valore, **all'interno del ciclo**: una semplice correzione risolve il problema:

```
  int k = strlen(p), m = strlen(a);
  for (i = 0; i < m-k; i++)
  { if (strncmp(&a[i], p, k) == 0) cout << i << " "; }
```

## array bidimensionali

Le **matrici** sono strutture dati dalle più svariate applicazioni (tabelle, orari, statistiche, etc.), per le quali una ricca collezione di operazioni è nota dall'**algebra lineare**

gli **array bidimensionali**, ossia con due indici, ne sono una naturale rappresentazione in C++; a titolo di esempio, il **prodotto di matrici quadrate** di numeri (interi o reali) si codifica facilmente come segue:

```
for (i = 0; i < N; i++)
  for (j = 0; j < N; i++)
  { c[i][j] = 0; for (k = 0; k < N; i++) c[i][j] += a[i][k]*b[k][j]; }
```

d'altra parte, è anche possibile adoperare array unidimensionali per la rappresentazione di matrici di due o più dimensioni (tutta la memoria può essere vista come un singolo array unidimensionale, avente per indici gli indirizzi di memoria)

ad esempio, nel prodotto di matrici quadrate, l'istruzione nel ciclo più interno diverrebbe:

```
c[N*i+j] += a[N*i+k]*b[N*k+j]
```

## array di array

un array bidimensionale può essere considerato come un **array di array** unidimensionali; l'allocazione dinamica della memoria per una tale struttura (di interi, a titolo di esempio), dati il numero `r` di righe e `c` di colonne, può effettuarsi con la seguente funzione, che alloca preliminarmente la memoria per un array di `r` puntatori, e quindi la memoria per un array di dimensione `c` per ciascuno di essi

```
int **malloc2d(int r, int c)
{ int **t = new int*[r];
  for (int i = 0; i < r; i++) t[i] = new int[c];
  return t;
}
```

la rappresentazione vista sopra si generalizza induttivamente ad array di qualsiasi numero (finito) di dimensioni (v. esercizi)

d'altra parte, essendo un array rappresentato da un puntatore, non è necessario che gli array interni in un array di array abbiano tutti la stessa dimensione: si possono realizzare strutture più flessibili, quali ad esempio gli **array di stringhe**

## array di stringhe

poiché una stringa è rappresentata in C++ da un puntatore a un array di caratteri, un array di stringhe è concretamente un array di tali puntatori

come già sappiamo, ad esempio, l'array `argv[]` usato per il passaggio di argomenti alla funzione `main` contiene puntatori alle stringhe nel buffer della linea di comando (e l'intero `argc` dà il numero di tali puntatori)

ad esempio, per rappresentare la frase (celebremente ambigua: ha quattro derivazioni sintattiche nella grammatica della lingua inglese!)

> time flies like an arrow

si può memorizzarla sequenzialmente in un buffer `b[25]`, ponendovi il carattere `\0` dopo ogni parola (e non memorizzando gli spazi), e costruire quindi l'array `a[5]` dei puntatori alle sue parole:

```
a[0] = &b[0]; a[1] = &b[5]; a[2] = &b[11]; a[3] = &b[16]; a[4] = &b[19];
```

## esempio: ordinamento di stringhe

la rappresentazione di una sequenza di stringhe come array di puntatori permette di ordinarla **operando sull'array di puntatori** piuttosto che sulle stringhe

si può usare a tal fine la funzione di libreria **qsort**, con 4 argomenti:

- un puntatore all'array degli oggetti da ordinare (i puntatori alle stringhe, qui)
- il numero di oggetti da ordinare
- la dimensione di ogni oggetto
- una funzione di confronto, con valore di ritorno intero

nel nostro caso gli oggetti da ordinare sono puntatori a stringhe: la loro dimensione è data dalla funzione di libreria **sizeof(char*)**

**problema:** per il confronto si vuole usare la funzione di libreria **strcmp**, che effettua il confronto lessicografico fra due stringhe di cui vuole i puntatori, ma **qsort** vuole una funzione di confronto che abbia due puntatori a **void** come parametri

**soluzione:** definire la funzione voluta da **qsort**, invocando nel suo corpo la funzione **strcmp**, con conversione di tipo dei puntatori e successiva risoluzione del riferimento dei puntatori in gioco, che sono quelli presenti nell'array

## ordinamento di array di stringhe in C++

il seguente programma effettua l'ordinamento secondo le precedenti indicazioni:

```
#include <iostream>
#include <cstdlib>
#include <cstring>
using namespace std;
int compare (const void *i, const void *j)
{ return strcmp(*(char **)i, *(char **)j); }
int main()
{ const int Nmax = 1000;
  const int Mmax = 10000;
  char* a[Nmax]; int N;                // array dei puntatori a stringhe
  char b[Mmax]; int M = 0;             // buffer della sequenza di stringhe
  for (N = 0; N < Nmax; N++)           // ciclo di lettura delle stringhe
  { a[N] = &b[M];                      // puntatore alla stringa da leggere
    if (!(cin >> a[N])) break;         // lettura di una stringa (terminata
    M += strlen(a[N]) + 1;             // da <SP> o <NL>: da \0 nel buffer)
  }
  qsort(a, N, sizeof(char*), compare); // ordinamento dell'array di puntatori
  for (int i = 0; i < N; i++)          // output delle stringhe in tale ordine
    cout << a[i] << endl;
}
```

## esercizi sulle stringhe

1. in che errore si incorre se si invoca **strcat(p,p)**, con l'implementazione di **strcat** qui proposta ? più in generale, per quali valori del puntatore **q** l'invocazione **strcat(p,q)**, con la detta implementazione di **strcat**, incorre in tale errore? modificare l'implementazione di **strcat** così da non incorrere nell'errore in questione, per qualsiasi valore del puntatore **q**
2. il testo (Sedgewick, 2003), p.112, propone la seguente implementazione dell'operazione **strncmp**, prima descritta, evidentemente erronea poiché non usa il parametro **n** : correggerla

```
int strncmp(char *a, char *b, int n)
{ int i;
  for (i = 0; a[i] == b[i]; i++)
    if (a[i] == 0) return 0;
  if (a[i] == 0) return 0;
  return a[i] - b[i];
}
```

3. v. esercizio 3.56 del testo (Sedgewick, 2003), p.116: definire una funzione C++ che verifichi se una data stringa è palindroma (cioè non cambia invertendola), trascurando gli spazi
4. v. esercizio 3.59 del testo (Sedgewick, 2003), p.116: definire una funzione C++ che rimpiazzi con un solo spazio le sottostringhe di due o più spazi consecutivi in una stringa data

## esercizi su array di array

1. generalizzare il codice qui proposto per la moltiplicazione di matrici quadrate alla moltiplicazione di matrici di dimensioni qualsiasi (ma tali che il prodotto sia definito)
2. definire una funzione C++ **int ***malloc3d(int p, int r, int c)** per l'allocazione dinamica della memoria per un array tridimensionale, visto come array di array bidimensionali
3. quale semplice modifica al programma 3.17 nel testo (Sedgewick, 2003), p.120, qui riprodotto, è sufficiente ad ottenere l'output della sequenza di stringhe in ordine lessicografico decrescente?