UNIVERSITÀ DEGLI STUDI DI BERGAMO

Informatica
Modulo di Programmazione

Informatica
Modulo di Programmazione
Strings

mauro.pelucchi@gmail.com
Mauro Pelucchi

2023/2024

Agenda

- Strings

Passing Arrays to Functions

- Specify name without brackets
  - To pass array **myArray** to **myFunction**

```
int myArray[ 24 ];
myFunction( myArray, 24 );
```

  - Array size usually passed, but not required
    - Useful to iterate over all elements

Passing Arrays to Functions

- Arrays passed-by-reference
  - Functions can modify original array data
  - Value of name of array is address of first element
    - Function knows where the array is stored
    - Can change original memory locations
- Individual array elements passed-by-value
  - Like regular variables
  - **`square( myArray[3] );`**

Passing Arrays to Functions

- Functions taking arrays
  - Function prototype
    - **`void modifyArray( int b[], int arraySize );`**
    - **`void modifyArray( int [], int );`**
      - Names optional in prototype
      - Both take an integer array and a single integer
  - No need for array size between brackets
    - Ignored by compiler
  - If declare array parameter as **`const`**
    - Cannot be modified (compiler error)
    - **`void doNotModify( const int [] );`**

Esercizio 1
Def. funzioni con parametri di tipo matrice

Scrivere un sotto-programma in linguaggio C++ che ricevuta una matrice quadrata come parametro restituisca al chiamante un valore booleano indicante se la matrice è simmetrica oppure no.

Scrivere quindi un programma principale minimale che acquisisca una matrice quadrata di numeri reali dallo standard input e testi la funzionalità del sotto-programma precedente. Se la matrice non è simmetrica occorre annullare le sue diagonali principali riportare su standard-output il risultato ottenuto.

Esercizio 1

$$\begin{pmatrix} 3 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

MATRICE SIMMETRICA

$$\begin{pmatrix} 0 & -1 & -3 \\ 1 & 0 & -2 \\ 3 & 2 & 0 \end{pmatrix}$$

MATRICE ANTISIMMETRICA

Esercizio 2
Def. funzioni con parametri di tipo matrice

Si considerino due matrici di interi A e B, di uguali dimensioni (R e C, costanti, che indicano il numero di righe e colonne).
Diciamo che A domina B se, confrontando i valori in posizioni corrispondenti, risulta che il numero dei valori in A maggiori dei corrispondenti valori in B è più grande del numero di quelli di B maggiori dei corrispondenti in A e inoltre gli elementi corrispondenti non sono mai uguali (se due elementi corrispondenti sono uguali la dominanza non è definita).

Si proponga un prototipo per la funzione domina(...) che riceve le matrici come parametri e restituisce 1 se la prima domina la seconda, -1 se la seconda domina la prima, 0 altrimenti.

Si dia una descrizione breve ma precisa di un algoritmo per implementare la funzione.

Si codifichi la funzione in C.

Fundamentals of Characters and Strings

- Character constant
  - Integer value represented as character in single quotes
  - `'z'` is integer value of `z`
    - `122` in ASCII
- String
  - Series of characters treated as single unit
  - Can include letters, digits, special characters `+`, `-`, `*` ...
  - String literal (string constants)
    - Enclosed in double quotes, for example:

      `"I like C++"`
  - Array of characters, ends with null character `'\0'`
  - String is constant pointer
    - Pointer to string's first character
      - Like arrays

String assignment

- Character array
  - **`char color[] = "blue";`**
    - Creates 5 element **`char`** array **`color`**
      - last element is **`'\0'`**
- Variable of type **`char *`**
  - **`char *colorPtr = "blue";`**
    - Creates pointer **`colorPtr`** to letter **`b`** in string **`"blue"`**
      - **`"blue"`** somewhere in memory
- Alternative for character array
  - **`char color[] = { 'b', 'l', 'u', 'e', '\0' };`**

Reading strings

– Assign input to character array **`word[ 20 ]`**

`cin >> word`

- Reads characters until whitespace or EOF
- String could exceed array size

`cin >> setw( 20 ) >> word;`

- Reads 19 characters (space reserved for **`'\0'`**)

Reading strings

- **`cin.getline`**
  - Read line of text
  - **`cin.getline( array, size, delimiter );`**
  - Copies input into specified **`array`** until either
    - One less than **`size`** is reached
    - **`delimiter`** character is input
  - Example

```
char sentence[ 80 ];
cin.getline( sentence, 80, '\n' );
```

- **`cin.get()`**
  - Read character and returns that character
  - **Example**

```
char c;
c = cin.get();
```

Could use a Condition like

```
((c = cin.get())!= '\n');
```

Cstrings

String handling library **<cstring>** provides functions to

- Manipulate string data
- Compare strings
- Search strings for characters and other strings
- Tokenize strings (separate strings into logical pieces)

Cstrings

`char *strcpy( char *s1, const char *s2 );`	Copies the string **`s2`** into the character array **`s1`**. The value of **`s1`** is returned.
`char *strncpy( char *s1, const char *s2, size_t n );`	Copies at most **`n`** characters of the string **`s2`** into the character array **`s1`**. The value of **`s1`** is returned.
`char *strcat( char *s1, const char *s2 );`	Appends the string **`s2`** to the string **`s1`**. The first character of **`s2`** overwrites the terminating null character of **`s1`**. The value of **`s1`** is returned.
`char *strncat( char *s1, const char *s2, size_t n );`	Appends at most **`n`** characters of string **`s2`** to string **`s1`**. The first character of **`s2`** overwrites the terminating null character of **`s1`**. The value of **`s1`** is returned.
`int strcmp( const char *s1, const char *s2 );`	Compares the string **`s1`** with the string **`s2`**. The function returns a value of zero, less than zero or greater than zero if **`s1`** is equal to, less than or greater than **`s2`**, respectively.

Cstrings

`int strncmp( const char *s1, const char *s2, size_t n );`	Compares up to **n** characters of the string **s1** with the string **s2**. The function returns zero, less than zero or greater than zero if **s1** is equal to, less than or greater than **s2**, respectively.
`char *strtok( char *s1, const char *s2 );`	A sequence of calls to **strtok** breaks string **s1** into "tokens"—logical pieces such as words in a line of text—delimited by characters contained in string **s2**. The first call contains **s1** as the first argument, and subsequent calls to continue tokenizing the same string contain **NULL** as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, **NULL** is returned.
`size_t strlen( const char *s );`	Determines the length of string **s**. The number of characters preceding the terminating null character is returned.

Esercizio 3
Stringhe

Scrivere un programma in linguaggio C++ che verifichi se una parola inserita dallo standard input è palindroma e in caso contrario stampa a video la parola inserita partendo dall'ultimo carattere sino al primo.
Le parole palindrome sono quelle che possono essere lette indifferentemente da sinistra a destra e da destra a sinistra.

Esempio:
"anna", "abba", "onorarono", "radar",
"a" (parola di una lettera).

Esercizio 4
Uso di funzioni per la manipolazione di stringhe

Facendo uso dei comandi messi a disposizione dalla libreria standard del C++ definiti in <cstring>, risolvere il seguente problema.

Leggere due stringhe str1, str2 dallo standard input, stampare a video lunghezza di str1 e str2, stampare il risultato della concatenazione delle due stringhe: str1 concatenata str2, confrontare le due stringhe e stamparle in ordine alfabetico.

Esercizio 5
Uso di funzioni per la manipolazione di stringhe

Scrivere un programma in linguaggio C++ che conti il numero di parole contenute in una frase inserita dall'utente.

Considerare qualsiasi carattere non alfanumerico come un possibile separatore di parola.

Strutturare il programma prevedendo la definizione e l'utilizzo di un sottoprogramma con il seguente prototipo

bool alfanum(char c)

che restituisce true nel caso il carattere passato come parametro sia alfanumerico (['A'...'Z', 'a'...'z', '0'...'9']), false in caso contrario.

Esempi:

"Oggi piove" : 2 parole
"Oggi c'e' il sole" : 5 parole

Esercizio 6
Def. funzioni con parametri di tipo vettore

Si progetti e codifichi una funzione C++ che riceve come parametri due stringhe che rappresentano due parole e restituisce un valore intero,
da interpretarsi come valore di verità, che indichi se le due parole sono anagrammi, cioè se sono ottenibili l'una dall'altra tramite una permutazione
delle lettere che le compongono.

Ad esempio le parole POLENTA e PENTOLA sono anagrammi.

Si presti attenzione al fatto che parole come TAPPO e PATTO non sono anagrammi, anche se ogni lettera dell'una è contenuta nell'altra.

Esercizio 7
Def. funzioni con parametri di tipo vettore

Definiamo la "somiglianza" S fra due stringhe come il rapporto tra il doppio del numero di "triplette di caratteri consecutivi" comuni ad entrambe le stringhe e la somma delle "triplette di caratteri consecutivi" in ciascuna stringa.

Nel caso in cui almeno una delle due stringhe sia composta da meno di 3 caratteri, la somiglianza è nulla (S = 0.0).

Scrivere un programma C++, che acquisisca due stringhe da tastiera (standard input) e riporti sullo schermo (standard output) la loro "somiglianza", secondo la definizione precedente.

Esercizio 7
Def. funzioni con parametri di tipo vettore

Esempio: `string1: "ALEXANDRE"`

`string2: "ALEKSANDER"`

triplette in comune: `2; "ALE", "AND"`

Numero di triplette in `string1:` **7**

(Oss.: num. caratteri di `string1` **meno** `2`**)**

Numero di triplette in `string2:` **8**

(Oss.: num. caratteri di `string2` **meno** `2`**)**

somiglianza: `S = (2·2)/(7+8) =` **0.27**