

Stringhe/operatori di incremento e decremento

Andrea Marin

Università Ca' Foscari Venezia
Laurea in Informatica
Corso di Programmazione

a.a. 2014/2015

Operatori ++ e --

- ▶ Gli operatori ++ e -- servono ad incrementare o decrementare di 1 una **variabile** intera
- ▶ Questi operatori possono essere prefissi o postfissi
 - ▶ `int x=7;`
 - ▶ Prefisso: `++x` e `--x`
 - ▶ Postfisso: `x++` e `x--`
- ▶ Questi operatori applicati ad una variabile costituiscono un'espressione
- ▶ Il valore dell'espressione è
 - ▶ Quello della variabile **precedentemente** all'incremento in caso di operatori postfissi
 - ▶ Quello della variabile **dopo** l'incremento in caso di operatori prefissi
- ▶ Anche se producono un'espressione, questi operatori possono essere usati come istruzioni:
 - ▶ `x--;`
 - ▶ Diventa equivalente a `x = x - 1;`



Postfisso vs. Prefisso

- ▶ Data l'inizializzazione `int x=10;`
- ▶ L'istruzione `printf('%d', x++);` stampa il valore 10 e, dopo la valutazione dell'espressione `x++` il valore della variabile `x` è 11
- ▶ L'istruzione `printf('%d', --x);` stampa il valore 9 perchè la modifica del valore della variabile avviene prima della valutazione dell'espressione



Esempio

- L'uso principe degli operatori di incremento e decremento si vede nei cicli for

```
..  
double vect [DIM];  
int i;  
  
int main() {  
    ..  
    for (i=0; i < DIM; i++)  
        vect[i] = i * 1.0;  
    ..  
}
```

- Attenzione alla leggibilità del codice!



Codice compresso

- ▶ Esempio di prima rivisitato
- ▶ È importante saper leggere il codice scritto in questo modo
- ▶ Preferibilmente da non usare per favorire la leggibilità
- ▶ Non sarete gli unici a mettere le mani su del codice

```
double    vect [DIM];  
int    i;  
  
int    main() {  
    ..  
    i=DIM;  
    while ( i ) vect[--i]=i*1.0;  
    ..  
}
```



Altri operatori

- ▶ `somma += x;`
 - ▶ `somma = somma + x;`
- ▶ `somma -= x;`
 - ▶ `somma = somma - x;`
- ▶ `prod *= x;`
 - ▶ `prod = prod * x;`
- ▶ `prod /= x;`
 - ▶ `prod = prod / x;`
- ▶ `prod %= x;`
 - ▶ `prod = prod % x;`



Stringhe in C

- ▶ Il C a differenza di altri linguaggi non ha il tipo `string`
- ▶ Una stringa è un array di `char` con particolari convenzioni
 - ▶ Esempio: `char miastringa[DIM];`
- ▶ Il punto chiave per la comprensione delle stringhe è cogliere la differenza tra lo spazio che riserviamo per memorizzare una stringa e la lunghezza della stringa
- ▶ Se riserviamo m `char` per la memorizzazione allora la lunghezza della stringa è al massimo $m - 1$



Esempio: leggo una stringa da standard input

```
#include <stdio.h>
#define DIM 10

char miastringa [DIM];

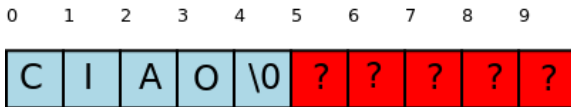
int main() {
    scanf("%s", miastringa);
    ..
    printf("La stringa letta e' %s\n", miastringa);
    ..
}
```

- Cosa osserviamo nella `scanf`?



Esempio di acquisizione

- ▶ Supponiamo che da standard input venga acquisita la stringa "CIAO"
- ▶ Nell'array è memorizzato quanto segue



- ▶ '\0' è un carattere speciale che denota il fine stringa (occupa un byte)
- ▶ Il suo encoding è quello del valore 0 (non del carattere '0')
- ▶ Per questo una stringa di n carattere occupa $n + 1$ bytes

Inizializzazione di stringhe

- ▶ Le stringhe, come gli array, possono essere inizializzate
- ▶ `char str[] = "Pippo";` istanzia un array di 6 caratteri (la dimensione minima necessaria)
- ▶ `char str[100] = "Pippo";` istanzia un array di 100 caratteri dei quali solo i primi 6 significativi
- ▶ **Attenzione:** l'operatore di assegnamento può essere usato solo per **inizializzare** le stringhe (come per gli array) ma non per fare assegnamenti
- ▶ L'operatore `==` **non** confronta le stringhe carattere per carattere per deciderne l'uguaglianza



Scorrere una stringa

- ▶ Per scorrere una stringa si intende considerarne tutti i caratteri da quello indicizzato dallo 0 a quello terminale `\0`
- ▶ Usiamo il ciclo `while`

```
#include <stdio.h>
#define DIM 100

/*contiene al piu' DIM-1 caratteri*/
char stringa[DIM];
int i;

int main() {
    ..
    i = 0;
    while (stringa[i]!='\0') {
        /*operazione su carattere i-mo*/
        i++;
    }
    ..
}
```



Abbreviazione

- Sfruttando il fatto che la codifica del carattere `'\0'` è il valore 0, il ciclo while della slide precedente può essere riscritto come

```
..  
while (stringa[i]) {  
    ..  
}
```



Confrontare se due stringhe sono uguali

- Ricorriamo al patter della proprietà universale: per ogni carattere della prima stringa, quello della seconda deve essere identico fino al raggiungimento del fine stringa



Soluzione errata

- Fornire un esempio in cui la computazione fallisce

```
..
char str1[DIM], str2[DIM];
int i, uguali;

int main() {
    ..
    i = 0;

    uguali = 1;

    while (str1[i] && str2[i] && uguali) {
        if (str1[i] != str2[i])
            uguali = 0;
        i++;
    }

    if (uguali)
        printf("Stringhe uguali");
    else
        printf("Stringhe diverse");
    ..
}
```



Soluzione **corretta**

```
..
char str1[DIM], str2[DIM];
int i, uguali;

int main() {
    ..
    i = 0;

    uguali = 1;

    while ( (str1[i] || str2[i]) && uguali) {
        if (str1[i] != str2[i])
            uguali = 0;
        i++;
    }

    if (uguali)
        printf("Stringhe uguali");
    else
        printf("Stringhe diverse");
    ..
}
```



Questa soluzione è corretta?

- ▶ Provare la correttezza di questo codice è più complicato del precedente, lo stile conta?
- ▶ Fornire un controesempio se il programma è errato o argomenta la correttezza altrimenti

```
..
char str1[DIM], str2[DIM];
int i;

int main() {
    ..
    int i;
    i = 0;

    while ( (str1[i] == str2[i]) && str1[i++] );

    if (str1[i-1] || str2[i-1])
        printf("Stringhe diverse");
    else
        printf("Stringhe uguali");
    ..
}
```

