

Array, stringhe e sottoprogrammi

Andrea Marin

Università Ca' Foscari Venezia
Laurea in Informatica
Corso di Programmazione

a.a. 2014/2015

Esercizio 1: cosa stampa?

```
int foo(int x, int y) {  
    int k = x + y;  
    return k;  
}  
int x, tot;  
int main() {  
    x = 13;  
    if (x>10) {  
        int x, y;  
        x = 7;  
        y = 3;  
        tot = foo(y + 2, x);  
    }  
    printf("%d", tot + x);  
    return 0;  
}
```



Evoluzione dell'ambiente

```
#include<stdio.h>
int foo(int x, int y)
{
    int k=x+y;
    return k ;
}

int x, tot; ←
int main() {
    x = 13;
    if (x>10) {
        int x, y;
        x =7;
        y=3;
        tot = foo(y + 2, x);
    }
    printf("%d", tot + x);
    return 0;
}
```

La freccia indica la posizione nel codice. In nero le variabili istanziate ma non visibili, in rosso quelle istanziare e visibili

x:?
tot:?



Evoluzione dell'ambiente

```
#include<stdio.h>
int foo(int x, int y)
{
    int k=x+y;
    return k ;
}

int x, tot;

int main() {
    x = 13;
    if (x>10) {
        int x, y;
        x =7;
        y=3;
        tot = foo(y + 2, x);
    }
    printf("%d", tot + x);
    return 0;
}
```

x: 13
tot: ?



Evoluzione dell'ambiente

```
#include<stdio.h>
int foo(int x, int y)
{
    int k=x+y;
    return k ;
}

int x, tot;

int main() {
    x = 13;
    if (x>10) {
        int x, y;
        x =7;
        y=3;
        tot = foo(y + 2, x);
    }
    printf("%d", tot + x);
    return 0;
}
```

x:13
tot: ?
x: 7
y: 3



Evoluzione dell'ambiente

```
#include<stdio.h>
int foo(int x, int y)
{
    int k=x+y;
    return k ;
}

int x, tot;
int main() {
    x = 13;
    if (x>10) {
        int x, y;
        x =7;
        y=3;
        tot = foo(y + 2, x);
    }
    printf("%d", tot + x);
    return 0;
}
```

x:13
tot: ?
x: 7
y: 3

foo(5, 7)



Evoluzione dell'ambiente

```
#include<stdio.h>
int foo(int x, int y)
{
    int k=x+y;
    return k ;
}

int x, tot;

int main() {
    x = 13;
    if (x>10) {
        int x, y;
        x =7;
        y=3;
        tot = foo(y + 2, x);
    }
    printf("%d", tot + x);
    return 0;
}
```



x:	13
tot:	?
x:	7
y:	3
x:	5
y:	7
k:	12



Evoluzione dell'ambiente

```
#include<stdio.h>
int foo(int x, int y)
{
    int k=x+y;
    return k ;
}

int x, tot;

int main() {
    x = 13;
    if (x>10) {
        int x, y;
        x =7;
        y=3;
        tot = foo(y + 2, x);
    }
    printf("%d", tot + x);
    return 0;
}
```

x: 13
tot: 12



esercizio 2: cosa stampa?

```
void scambia(int a, int b) {  
    int sup;  
    sup = a;  
    a = b;  
    b= sup;  
}
```

```
int main() {  
    int x, y;  
    x = 10;  
    y = 20;  
    scambia(x, y);  
    printf("%d %d", x, y);  
    return 0;  
}
```



Passaggio di parametri in C

- ▶ In C l'unico passaggio di parametri è per copia
- ▶ A prima vista il passaggio di vettori e stringhe sembra differente
 - ▶ In realtà non è un'eccezione alla regola, ma lo potremo capire solo quando studieremo puntatori ed indirizzi
- ▶ Quando si passa un array ad una funzione **non** viene copiato ogni singolo elemento dell'array
 - ▶ Questo avviene per esempio in Pascal
- ▶ Come conseguenza **le modifiche effettuate alle celle dell'array sono apportate anche nell'oggetto indicato come parametro attuale**
 - ▶ Questo viene chiamato **side effect**



Dimensioni dell'array (no stringhe)

- ▶ Le dimensioni dell'array devono essere specificate all'ambiente della funzione in qualche modo
- ▶ Due alternative:
 - ▶ Si definisce una costante globale *visibile* nell'ambiente
 - ▶ Si passa come parametro
 - ▶ Maggiore portabilità del codice



Esempio: stampare gli elementi di un array di float

```
void stampa_array(float array[], unsigned int dimensione) {  
    int i;  
    for (i=0; i<dimensione; i++) {  
        printf('%f ', array[i]);  
    }  
}  
  
int main() {  
    float mio_array[] = {13.0, 3.0, 60.60};  
    stampa_array(mio_array, 3);  
    return 0;  
}
```



Array multidimensionale come parametro

- ▶ Se si desidera passare un array multidimensionale come parametro ad una funzione il parametro formale deve specificare tutte le dimensioni tranne (eventualmente) la prima
- ▶ In una matrice si specifica il numero di colonne ma non serve di righe

```
#define RIGHE 20
#define COLONNE 20

void stampa_matrice(int matr[][COLONNE], int righe) {
    int r,c;
    for (r=0; r<righe; r++)
        for (c=0; c<COLONNE; c++)
            printf("%d", matr[r][c]);
}
```



Uso del side effect

- Sostituire i numeri dispari in un array con zeri

```
#define DIM 20

void togli_dispari(int vet[], unsigned int dimensione) {
    int i;
    for (i=0; i<dimensione; i++) {
        if (vet[i] % 2 == 1)
            vet[i] = 0;
    }
}

int main() {
    int vettore[DIM];
    leggi_array(vettore, DIM);
    togli_dispari(vettore, DIM);
    stampa_array(vettore, DIM);
    return 0;
}
```



Celle riservate e celle usate

- ▶ Prendiamo il problema: caricare un vettore con numeri letti da standard input. La sequenza in input termina con lo 0
- ▶ Quanto grande deve essere il vettore?
 - ▶ Impossibile determinarlo
- ▶ In questi casi si definisce una dimensione massima dell'array e se ne occupa la parte che serve
 - ▶ Nel caso la sequenza sia più lunga delle celle riservate?
Gestione del buffer overflow
- ▶ Una variabile è usata per indicare la porzione utilizzata



Passaggio di stringhe come parametri


- ▶ Le stringhe sono vettori di caratteri, quindi valgono le regole fin qui dette
- ▶ Da notare che non serve passare come parametro il numero di caratteri impiegati grazie alla codifica del fine stringa col carattere `'\0'`
- ▶ Talvolta è importante passare la dimensione massima della stringa (cioè il numero di caratteri effettivamente allocati)



Lunghezza di una stringa

```
#define DIM 100
int strlen(char str[]) {
    int i = 0;
    while (str[i])
        i++;
    return i;
}

int main() {
    char stringa[DIM];
    scanf("%s", stringa); /* gets(stringa); */
    int dimensione = strlen(stringa);
    printf("La stringa ha %d caratteri", dimensione);
    return 0;
}
```



Uppcase di una stringa

```
void upcase(char str[]) {  
    int i=0;  
    while (str[i]) {  
        if (str[i]>='a' && str[i]<='z')  
            str[i] = str[i] - 'a' + 'A';  
        i++;  
    }  
}
```

- Si sfrutta il side effect



Acquisire una stringa in C

- ▶ `scanf`: legge una sequenza di caratteri (si ferma allo spazio o al new line)
- ▶ `gets`: legge una sequenza di caratteri (si ferma solo al new line)
- ▶ `getline`: legge una sequenza di caratteri (si ferma solo al new line), controllo sul buffer overflow
 - ▶ Comportamento più sofisticato, vedremo in seguito



Esercizi

Da svolgere scrivendo funzioni per risolvere i problemi, e un main di prova

- ▶ Data una stringa, contare di quante parole è formata. Le parole sono separate da uno o più spazi consecutivi.
- ▶ Dato un intero positivo e una stringa trasformare il numero in stringa
- ▶ Data una stringa che contiene la codifica di un numero intero positivo trasformarlo in int
- ▶ Date due stringhe, dire se sono uguali
- ▶ Date due stringhe di caratteri maiuscoli, dire se la prima è prefisso della seconda
- ▶ Realizzare un programma che consenta di simulare un mazzo di carte. Le carte sono codificare con in numeri da 1 a 40. Realizzare le funzioni:
 - ▶ `prepara_mazzo` che prepara il mazzo con le 40 carte
 - ▶ `pesca_carta` che pesca una carta causale dal mazzo.Attenzione: se la carta è stata pescata in precedenza, non può essere ripescata (almeno che non venga chiamata la funzione `prepara_mazzo`)

