CdL in Informatica – A.A. 2024 – 2025

Programmazione 1 – Modulo 2

Lezione 6
17/10/2024

Andrea Loddo

Federico Meloni - Alessandra Perniciano - Fabio Pili

Prossime tappe

- 22 ottobre: lezione regolare
- 24 ottobre: lezione regolare
- 29 ottobre: lezione regolare
- 31 ottobre: festa
- 5 novembre: lezione regolare
- 7 novembre: **primo test parziale**

Argomenti

- Costrutti iterativi
  - For
- Array
  - Array e indici
  - Inizializzazione
  - Esempi utilizzo
  - Insidie ed errori comuni

Costrutti iterativi

Il costrutto for

REMINDER: costrutti iterativi

Il paradigma sequenziale prevede la possibilità di tornare indietro nel flusso d'esecuzione per ripetere una o più istruzioni (salto):

- Salto condizionato: si ha la ripetizione di una o più istruzioni in seguito alla valutazione di una espressione booleana.
  - Numero di iterazioni indefinite: costrutti while e do while.
  - Numero di iterazioni definite: costrutto for.

- Salto incondizionato: le istruzioni vengono ripetute senza effettuare alcuna valutazione (meglio evitare).
  - goto

Costrutto for

- Il costrutto for viene solitamente usato per la ripetizione di un insieme di istruzioni per un numero definito di volte.

```
for ( inizializzazione; condizione; incremento )
{
   /* blocco di istruzioni da svolgere
      finché la condizione è vera */
}
```

- Funzionamento del for:
  1. Si esegue l'inizializzazione del contatore
  2. Si valuta l'espressione booleana condizione;
     - Se la condizione è vera si eseguono le istruzioni del blocco;
       - Al termine del blocco si:
         a. Effettua l'incremento;
         b. Ripete la valutazione di condizione al punto 2.
     - Se la valutazione di condizione risulta falsa, il ciclo termina e il flusso di esecuzione passa alla prima istruzione dopo il for.

For: esempi

```
int i;
/* Che fa questo programma? */
for( i=0; i<10; i++ )
{
    printf("Numero: %d\n", i);
}
```

```
int j;
/* Che fa questo programma? */
for( j=10; j>0; j-- )
{
    printf("Numero: %d\n", j);
}
```

```
int i;
/* Che fa questo programma? */
for( i=0; i<10; i+=2 )
{
    printf("Numero: %d\n", i);
}
```

1. Si esegue l'inizializzazione;
2. Si valuta la condizione: se è vera, viene eseguito il blocco fintanto che è vera;
3. Si esegue l'incremento (o il decremento…);
4. Se la condizione è falsa in partenza salta l'esecuzione.

NB: come nel while, il controllo della condizione viene effettuato prima del blocco di istruzioni.

Costrutto for

- Nel for nessuno degli elementi è obbligatorio `for ( ; ; )`
  - Meglio non farlo: si avrebbe un ciclo infinito

- A volte può risultare utile interrompere il blocco in corso d'opera sul variare di qualche condizione
  - break: causa l'immediata uscita dal ciclo
  - meglio evitare

- A volte può risultare utile saltare l'iterazione attuale, e passare all'iterazione successiva senza uscire dal ciclo
  - continue: causa il salto all'iterazione successiva
  - utilizzare con moderazione

Break e continue: esempi

```
int x, y = 0;

for( x = 0; x < 10; x++ )
{
    /* si "salta" l'iterazione
    in cui x vale 5 */
    if( x == 5 )
    {
          continue;
    }
    y += 100/(x-5);
    printf("%d ", y);
}
```

```
int i, somma = 5;

for( i = 0; i < 100; i++ )
{
    somma *= 10;

    /* si esce dal ciclo se
    somma vale almeno 50 */
    if( somma >= 50 )
    {
          break;
    }
}
```

Tipicamente, è possibile ripensare il flusso d'esecuzione in modo da evitare l'uso di break o continue.

Break e continue: esempi alternativi

```
int x, y = 0;

for( x = 0; x < 10; x++ )
{
    /*si esegue il blocco if
    solo se x è diverso da 5*/
    if( x != 5 )
    {
          y += 100/(x-5);
          printf("%d ", y);
    }
}
```

```
int i, somma = 5;

for(i=0; i<100 && somma<50; i++)
{
    somma *= 10;
}

printf("Somma = %d", somma);
```

Tipicamente, è possibile ripensare il flusso d'esecuzione in modo da evitare l'uso di break o continue.

REMINDER: Infinite loop

- Se in un ciclo la condizione è sempre vera si entra in un infinite loop (o ciclo infinito)

- Il ciclo infinito è un problema (come la divisione per zero) che non appare nel codice e si può presentare a runtime: una condizione non risulta mai falsa ed il programma va in stallo

```
/* Che succede in questo esempio? */
int i;
for( i = 0; i >= 0; i++ )
{
      printf("questo non e' infinito");
}
```

Dichiarazione variabili nei cicli

- Il C99 permette di dichiarare variabili utilizzabili come indici dei cicli all'interno del ciclo stesso, senza doverle dichiarare in precedenza.

```
for (int i=0; i<n; i++) {
    ...
}
```

- Lo scope della variabile i è limitato al solo blocco dell'istruzione iterativa.
- Istruzioni iterative differenti e non annidate possono dichiarare indici con lo stesso nome, che di fatto corrisponderanno a variabili differenti perché appartenenti a scope diversi:

```
for (int i=0; i<n; i++) {
    ...
}
...
for (int i=0; i<m; i++) {
    ...
}
```

Salti incondizionati: goto

- Per dovere di cronaca: in C esiste un altro costrutto per modificare il flusso d'esecuzione.
- Il famigerato goto (go to, ”vai a”) permette al programma di "saltare" a un'altra istruzione senza riferimenti ai costrutti teorici di iterazione e/o selezione
- Contro:
  - non sfrutta il principio di località legato alle istruzioni iterative del linguaggio;
  - non è un costrutto della programmazione strutturata;
  - complica l'analisi della correttezza dei programmi;
  - non aggiunge niente al linguaggio;
  - crea "spaghetti code".

Tipi di dato non primitivi

Gli array monodimensionali

Array

- Supponiamo di voler tenere traccia del voto di cinque studenti per fare delle analisi statistiche.
- Servirebbero 5 variabili: voto1, voto2, voto3, voto4, voto5.
- Se volessi gestire i voti dell'intero corso di PR1?
  150 studenti → 150 variabili.
- Limite attuale: per ogni dato da memorizzare, dobbiamo dichiarare una variabile.

Array

- Il linguaggio C offre un tipo di dato non primitivo per dichiarare l'insieme dei voti come una singola maxi-variabile, unita alla possibilità di accedere ai singoli elementi dell'insieme.
- In questo modo si ha un'organizzazione migliore del codice.
- La struttura dati che serve per aggregare dati semanticamente affini si chiama array (o vettore in italiano).
- Ogni array specifica un insieme, di dimensione fissata, di variabili dello stesso tipo.

```
/* sintassi di dichiarazione di array */

tipo nomeVettore[dimensione];
```

Array

```
int matricole[150];          // array di 150 elementi interi
float votoEsame[150];        // array di 150 elementi float
```

- In memoria viene allocato staticamente e linearmente lo spazio sufficiente per contenere tutti gli elementi dell'array.
- L'accesso all'elemento singolo si effettua mediante indice, partendo da 0
- L'accesso non è sequenziale: si può accedere a qualsiasi elemento in qualsiasi momento.
- L'operatore parentesi quadre [ ] specifica la posizione (offset).

```
printf("Voto primo studente: %f.", votoEsame[0]); // stampa voto primo studente
```

- Ogni cella dell'array si comporta come variabile a sé stante e può essere usata come tale:

```
matricole[0] = 12345;
votoEsame[5] = (27.0+28.0)/2;
```

Array

- Se si vuole stampare il contenuto di un vettore, cioè un numero definito di elementi, si può scrivere:

```
for( i=0; i<150; i++ )
    printf( "Voto studente numero %d: %f\n", (i+1), votoEsame[i] );
```

- Come viene memorizzato il vettore in memoria?

Indice **i**	0	1	2	3	4	…	146	147	148	149
votoEsame	**27**	**28**	**25**	**18**	**30**	**…**	**22**	**24**	**19**	**21**

Array

- La dimensione di un array deve essere COSTANTE e PREFISSATA: deve essere noto, prima della dichiarazione, il numero di elementi.
- Il numero di elementi non può essere modificato in esecuzione.
- La dimensione massima è legata alla memoria disponibile e al massimo valore rappresentabile dal tipo dell'indice (int).

Esempio:

```
int n;
int array[n];
```

- È sintatticamente valida ma semanticamente scorretta.
- Qual è la dimensione dell'array?
- Problema: n potrebbe essere troppo grande da non permettere l'allocazione dell'array.
- In generale, siamo nel campo degli undefined behaviour.
- Come per le variabili, ciascuna cella risulta non inizializzata al momento della dichiarazione.

Inizializzazione array

Modo 1: scorrere elemento per elemento dando alle celle il valore desiderato

```
int voti[50];
for( i=0; i<50; i++ )
    voti[i] = 0;
```

Modo 2: solo in fase di dichiarazione, direttamente con i valori da assegnare.

```
int voti[5] = {22, 24, 24, 25, 28};
```

- Gli elementi vengono assegnati in ordine. Se la serie prevede più valori di quanti l'array può contenere, i valori superflui vengono ignorati:

```
int voti[10] = {22, 24, 24, 25, 28, 30, 19, 22, 25, 30, 30};
```

- Se ne contiene di meno, i valori restanti saranno posti a zero:

```
int voti[10] = {22, 24, 24};
```

- L'inizializzazione mediante graffe permette di specificare array senza dimensione (che viene automaticamente dedotta dal numero di elementi presenti):

```
int voti[] = {22, 24, 25}; // array da 3 elementi
```

Array: esempi d'uso

Quale sarà il risultato delle seguenti inizializzazioni?

```
int vettore[10], i = 0;

for ( i=0; i<10; i++ )
    vettore[i] = 0;

for ( i=0; i<10; i++ )
    vettore[i] = i;

for ( i=9; i>=0; i-- )
    vettore[i] = i;

for ( i=0; i<10; i++ )
    vettore[i] = 2*i;

for ( i=0; i<10; i++ )
    vettore[i] = pow(2, i);
```

Array: esempi d'uso

Quale sarà il risultato delle seguenti inizializzazioni?

```
int vettore[10], i;
for ( i=0; i<10; i++ )
    vettore[i] = 0;
```

0	0	0	0	0	0	0	0	0	0

Array: esempi d'uso

Quale sarà il risultato delle seguenti inizializzazioni?

```
int vettore[10], i;
for ( i=0; i<10; i++ )
    vettore[i] = i;
```

0	1	2	3	4	5	6	7	8	9

Array: esempi d'uso

Quale sarà il risultato delle seguenti inizializzazioni?

```
int vettore[10], i;
for ( i=9; i>=0; i-- )
    vettore[i] = i;
```

0	1	2	3	4	5	6	7	8	9

Array: esempi d'uso

Quale sarà il risultato delle seguenti inizializzazioni?

```
int vettore[10], i;
for ( i=0; i<10; i++ )
    vettore[i] = 2*i;
```

0	2	4	6	8	10	12	14	16	18

Array: esempi d'uso

Quale sarà il risultato delle seguenti inizializzazioni?

```
int vettore[10], i;
for ( i=0; i<10; i++ )
    vettore[i] = pow(2, i);
```

1	2	4	8	16	32	64	128	256	512

Insidie ed errori comuni

- Il C non effettua alcun controllo a runtime.
- Ciò fa sì che sia possibile specificare un indice che non fa parte del range di valori dell'array:

```
int vettore[10], i;
for ( i=0; i<20; i++ )
    vettore[i] = i;
```

```
int vettore[10], i;
vettore[-5] = 0;
```

- Sintatticamente corretti, ma ci sarà un probabile crash.
- Il programmatore deve verificare che gli indici siano sempre validi e non vadano fuori range.

Insidie ed errori comuni

- Inoltre, l'identificatore dell'array (senza indici) si riferisce a un tipo di dato particolare (puntatore) che non è compatibile con le operazioni viste per le variabili:

Esempio:

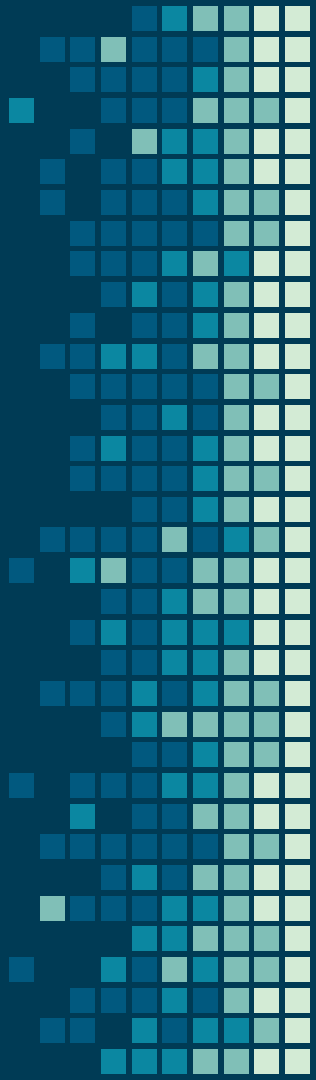
```
int vettore[5] = {1, 2, 3, 4, 5};
int vettoreBis[5];

vettoreBis = vettore;
```

- È un errore sintattico: non è possibile riassegnare l'array.
- Per effettuare una copia è necessario agire sui singoli elementi:

```
for( i=0; i<5; i++ )
    vettoreBis[i] = vettore[i];
```

FINE!

Domande?



Autovalutazione ed esercizi

Sul costrutto for

Autovalutazione

1. Per cosa deve essere utilizzato, di norma, il costrutto for?
2. Che differenze e somiglianze esistono tra i costrutti while, do while e for?
3. Posso omettere una o tutte le componenti del for (inizializzazione, condizione, incremento)?
4. Quando viene eseguita la componente legata all'incremento?
5. Quando viene eseguito il blocco di codice?
6. A cosa serve l'istruzione break?
7. A cosa serve l'istruzione continue?
8. Perché deve essere evitato il goto?

Esercizi

1. Scrivere un programma che stampi i primi n quadrati (elevamento del numero stesso alla potenza di 2). Chiedere in input il valore di n.
   Es.: inserendo 10, il programma deve stampare 1, 4, 9, 16, 25, 36, 49, 64, 81, 100.
2. Scrivere un programma che chieda all'utente un numero n e stampi tutti i quadrati dispari compresi tra 1 e n, al contrario.
   Es.: inserendo 10, il programma deve stampare la successione 81, 49, 25, 9, 1.
3. Generare un numero casuale compreso tra 0 e 100 e chiedere all'utente un numero nello stesso intervallo. Se sono uguali stampare un messaggio «Numeri uguali»; in caso contrario comunicare se il numero inserito è maggiore o minore di quello generato. Il programma deve terminare quando l'utente indovina il numero generato (for, while o do while?).
4. Scrivere un programma che stampi il calendario di un mese. L'utente deve specificare il numero di giorni nel mese e il giorno della settimana in cui questo comincia (0: lunedi', 6: domenica).
5. Scrivere un programma che stampi i primi n numeri primi.
6. Scrivere un programma che permetta la stampa delle tabelline a seconda del numero inserito dall'utente.
   Es: se l'utente inserisce 10, devono essere stampate le tabelline dalla 1 alla 10.

Esercizi

7. Scrivere un programma che permetta di convertire numeri binari in decimale. Il programma acquisisce prima il numero di cifre che formano il numero binario, dopodiché acquisisce una per volta le cifre del numero binario.
8. Scrivere un programma che simula un gioco tra due utenti: viene chiesto prima quanti tiri effettuare (n), dopodiché ogni utente tira n volte un dado. Vince chi ottiene la somma dei numeri ottenuti maggiore.

Autovalutazione ed esercizi

Sugli array

Autovalutazione

1. Che differenza c’è tra vettori e array monodimensionali?
2. Con lo standard C90 si può dichiarare un vettore senza specificarne la dimensione? Se sì, come?
3. Di che tipo può essere un vettore?
4. Quale indice rappresenta la prima cella di un vettore?
5. A cosa serve l'operatore [ ] in fase di dichiarazione? E in una fase diversa dalla dichiarazione?
6. Cosa rappresenta l'offset?
7. Quale segnaposto si deve utilizzare per stampare il contenuto della cella di un vettore?
8. Come viene memorizzato un vettore in memoria?
9. Che valore hanno le celle di un vettore di dieci interi dichiarato ma non inizializzato?
10. Cosa succede se si inizializza un vettore senza specificarne la dimensione?
11. Si può copiare un vettore in un altro usando l'operatore di assegnamento?

Esercizi

1. Scrivere un programma che generi una serie di N numeri casuali all'interno di un array. Dopodiché stampare in output tutti gli elementi in ordine inverso.
2. Scrivere un programma in cui, dato un insieme di valori in un array, si calcoli la media dei valori e vengano stampati in output tutti i valori minori della media, tutti i valori maggiori della media e la media stessa.
   Infine, stampare quanti sono i valori minori della media e quanti i valori maggiori.
3. Scrivere un programma in cui si dichiarino due array, A e B. Inizializzare il primo array a piacere e il secondo in modo che contenga gli stessi elementi di A in ordine inverso. Stampare entrambi gli array come verifica della correttezza.
4. Scrivere un programma in cui si dichiari e inizializzi a piacimento un array A, quindi normalizzarlo. Stampare l'array prima e dopo la normalizzazione.
   Hint: normalizzare un intervallo vuol dire portare tutti gli elementi dell'intervallo ad avere valori compresi tra 0 e 1, secondo la formula x_norm = (x − min)/(max − min).
5. Scrivere un programma in cui si dichiara un vettore A e si inizializza in maniera casuale. Trovare l'elemento, tra quelli presenti, più vicino a un numero inserito dall'utente.

Esercizi

6. Scrivere un programma che inizializzi un vettore, di dimensione N a piacere, casualmente. Il programma deve chiedere un valore all'utente e deve stampare in output se quel valore è contenuto nel vettore oppure no.
7. Scrivere un programma che inizializzi un vettore, di dimensione N a piacere, casualmente. Il programma deve chiedere un valore all'utente e deve stampare in output quante volte quel valore è contenuto nel vettore.
8. Scrivere un programma in cui si dichiara e inizializza un array A, quindi stamparlo. Senza utilizzare altri array d'appoggio, invertire l'ordine degli elementi di A e ristampare il vettore così modificato.
9. Scrivere un programma in cui si dichiara un vettore di 5 elementi. Assegnare casualmente 5 valori da 1 a 90 senza ripetizioni.
10. Aggiungere all'esercizio precedente un'interfaccia utente che permetta all'utente di effettuare una giocata del tipo: estratto, ambo, terno, quaterna, cinquina. In caso di vittoria dovrà essere comunicata all'utente la vincita, altrimenti dovrà essere concesso un nuovo tentativo senza una nuova estrazione.

Esercizi

11. Scrivere un programma che permetta di convertire numeri binari in decimale. Il programma acquisisce prima il numero di cifre che formano il numero binario, dopodiché acquisisce una per volta le cifre del numero binario. Il numero binario deve essere memorizzato in un array. Supporre che il numero binario possa avere al massimo 20 cifre.
12. Scrivere un programma che inizializzi, a piacere, un vettore di dimensione N. Il programma deve stampare il secondo elemento maggiore all'interno del vettore.