Linguaggio C++

Istruzioni iterative

Strutture di controllo: ITERAZIONE

FINTANTOCHE'(cond) **ESEGUI**
INIZIO
… //si esce dal ciclo falso
FINE
	while (cond) {
	… //si esce dal ciclo falso
	}

RIPETI … //si esce dal ciclo vero **FINO_A_CHE** (cond)	**do** { … //si esce dal ciclo falso }**while** (cond);

Pulizia del buffer di tastiera

Spiegazione del ciclo che effettua la pulizia del buffer di tastiera:

```
while(getchar() != '\n');     //in C
while(cin.get() != '\n');      //in C++
```

Strutture di controllo: ITERAZIONE

Esercizio: Controllo dell'input → $1 \leq x \leq 5$

Soluzione in C/C++:

```
do {
   cout<<"Digita un numero intero positivo\n";
                        //printf("Digita un numero intero positivo \n");
   cin>>x;              //scanf("%d",&x);
}while ((x < 1) || (x > 5));
```

Strutture di controllo: ITERAZIONE

Esercizio: Determinare l'output del seguente frammento di codice C/C++.

```
int num = 0;
    while(num <= 10) {
        num = num + 1;
        //printf("%d\n", num);
        cout<<num<<endl;
    }
```

Output:

1
2
3
4
5
6
7
8
9
10
11 ←

Strutture di controllo: ITERAZIONE

- 3^ forma di iterazione: for()

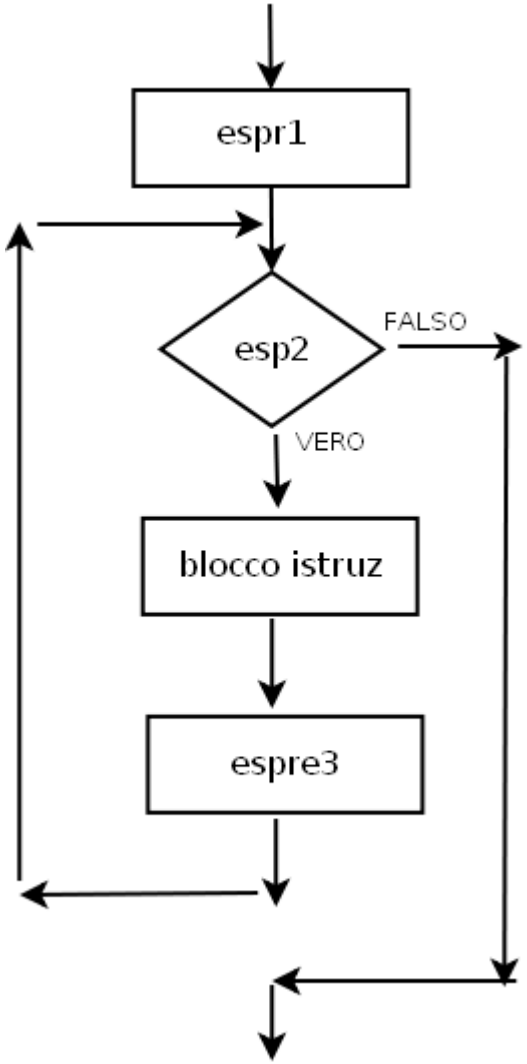
SINTASSI:

```
for(<espr1>; <espr2>; <espr3>) {
      blocco istruz.
}
```

- espr1: inizializzazione
- espr2: controllo del ciclo (condizione)
- espr3: aggiornamento
- NOTE:
  - Si esce dal ciclo per condizione falsa.
  - I due punti e virgola devono essere sempre presenti (anche in assenza di una o più “espr”).
  - Vedremo che le tre “espr” possono mancare in numero di 1, 2 o anche 3.

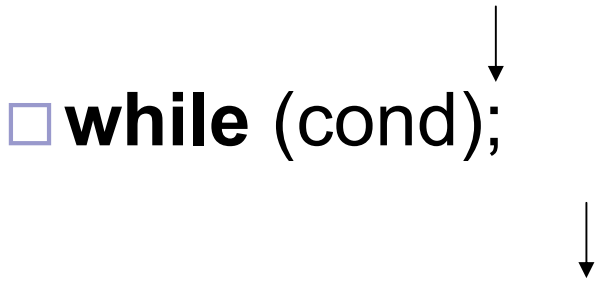
Strutture di controllo: ITERAZIONE

- SEMANTICA:



Strutture di controllo: ITERAZIONE

- Attenzione al punto e virgola: è molto probabile tu l'abbia messo per distrazione.



  - **while** (cond);
  - **for** (… ; … ; …);

Strutture di controllo: ITERAZIONE

Problema
Determinare la somma degli interi compresi tra n1 e n2, con n1 $\leq$ n2 (estremi inclusi).

Controllo dell'input:

```
do{
      cout<<"Digita due interi\n";
      //printf("Digita due interi \n");
      cin>>n1>>n2;
      //scanf("%d%d",&n1,&n2);
}while (n1 > n2);
```

```
sum = 0;
for (i=n1; i<=n2; i=i+1){
        sum = sum + i;
}
```

```
sum = 0;
i = n1;
while(i <= n2){
      sum = sum + i;
      i = i +1;
}
```

```
sum=0;
i = n1;
do{
      sum = sum + i;
      i = i + 1;
}while (i <= n2);
```

Strutture di controllo: ITERAZIONE

In C/C++ le tre forme di iterazione sono equivalenti:

while() ≡ do…while() ≡ for()

Dimostrazione: →

```
for(espr1; espr2; espr3){
        blocco istruz;
}
```

```
espr1;
while(espr2){
      blocco istruz;
      espr3;
}
```

```
espr1;
if(espr2){
      do{
            blocco istruz;
            espr3;
      }while (espr2);
}
```

Strutture di controllo: ITERAZIONE

```
int i;
int sum = 0;
    for(i=1; i<=3; i=i+1)
        sum = sum + i;
    cout<<“Somma = \n”<<sum;          → 1+2+3=6
    //printf(“Somma = %d\n”, sum);    → 1+2+3=6
    cout<<‘\n’<<i;                    → 4
    //printf(“%d\n”,i);               → 4
```

Il ciclo viene eseguito 3 volte.

int numElem = i -1; → 3 (la “i” (var di controllo del ciclo) meglio non usarla direttamente)

Strutture di controllo: ITERAZIONE

- for(i=1; i<=1; i=i+1)
  - Il ciclo viene eseguito 1 volta
- for (i=1; i<=-2; i=i+1)
  - Il ciclo non viene mai eseguito

Incrementi e decrementi

- Pre e post incremento

```
z = z +1;
z++;
++z;
```

- In generale sono equivalenti, sono diversi quando sono contenuti in espressioni.

Incrementi e decrementi

- Scrivendo z++, il valore di z viene prima usato e poi incrementato.

```
int x, z;
    z = 4;
    x = z++;

    (x vale 4, z vale 5)
```

Incrementi e decrementi

- Scrivendo ++z, il valore di z viene prima incrementato e poi usato.

```
int x, z;
    z = 4;
    x = ++z;

    (x vale 5, z vale 5)
```

Incrementi e decrementi

- Idem per:

```
z = z -1;
--z;
z--;
```

Attenzione:

```
int x = 1;
    cout<<x++;          //stampa 1, poi incrementa x di una unità
    //printf("%d",x++); //stampa 1, poi incrementa x di una unità
```

Incrementi e decrementi

km += 37	→	km = km + 37;
attesa -= 6	→	attesa = attesa – 6;
lanci *= 2;	→	lanci = lanci * 2;
torta /= 3;	→	torta = torta / 3;
carte %= 5;	→	carte = carte % 5;
↑		↑
Forma compatta		Forma classica

In generale: **<variabile> [<operatore>] = <espressione>;**

dove operatore: + , -, *, /, %

Incrementi e decrementi

```
for(i=1; i<=3; ++i) {
    …..
}
```

```
for(i=1; i<=3; i++) {
    …..
}
```

```
for(i=1; ++i<=3; ) {
    …..
}
```

```
for(i=1; i++<=3; ) {
    …..
}
```

Incrementi e decrementi

```
j=0;
for(i=1; ++i<=3; ) {
    cout<<“Ciclo: ”<<++j<<endl;
}
cout<<“Cicli: “<<j<<“ i: “<<i<<endl;
```

```
Ciclo: 1
Ciclo: 2
Cicli:  2   i: 4
```

```
j=0;
for(i=1; i++<=3; ) {
    cout<<“Ciclo: ”<<++j<<endl;
}
cout<<“Cicli: “<<j<<“ i: “<<i<<endl;
```

```
Ciclo: 1
Ciclo: 2
Ciclo: 3
Cicli:  3  i: 5
```

Operatore virgola

Operatore virgola = operatore di sequenza

```
for(numero=1, somma=0; numero;) {       //numero != 0
      //printf("…..");
      cout<<" ….. ";
      cin>>numero;
      //scanf("%d",&numero);
      somma +=numero;
}
```

Brutto! Preferisco una lettura prima del ciclo … (vedi lavagna).

Operatore virgola

- Esempio con più di una variabile di controllo.

- Determinare output:

```
int m,n;
    for(m=1, n=8; m < n; m++, n--)
        cout<<m<<n<<endl;
        //printf("%d    %d\n",m, n);
```

Operatore virgola

- Output:

m	**n**
1	8
2	7
3	6
4	5

Nesting di loops: cicli annidati

```
int a, i, j;
   a = 0;
    for(i=1; i <= 3; i++)
        for(j=1; j <= 5; j++)
            a = a + i * j;
```

Per capire la semantica eseguiamo la trace.

Cicli annidati

i	j	a
1	1	0 + 1 * 1 = 1
1	2	1 + 1 * 2 = 3
1	3	3 + 1 * 3 = 6
1	4	6 + 1 * 4 = 10
1	5	10 + 1 * 5 = 15
2	1	15 + 2 * 1 = 17
2	2	17 + 2 * 2 = 21
2	3	21 + 2 * 3 = 27
2	4	27 + 2 * 4 = 35
2	5	35 + 2 * 5 = 45
3	1	45 + 3 * 1 = 48
3	2	48 + 3 * 2 = 54
3	3	54 + 3 * 3 = 63
3	4	63 + 3 * 4 = 75
3	5	75 + 3 * 5 = **90**

Cicli annidati

Per ogni iterazione esterna viene completata quella interna.

Cicli annidati

```
int a = 0, i, j,k;

  for(i = 1; i <= 3; i++)
      for(j = 1; j <= 5; j++)
          for (k = 1; k <= 7; k++)
              a = a + i + j + k;
```

Quante volte viene eseguita l'istruzione
"a = a + i + j + k;" ?

3 * 5 * 7 = 105 volte
Per casa fai la trace.

Trace (con “estremi sup.” 2, 3, 2)

i	j	k	a
1	1	1	3
1	1	2	7
1	2	1	11
1	2	2	16
1	3	1	21
1	3	2	27
2	1	1	31
2	1	2	36
2	2	1	41
2	2	2	47
2	3	1	53
2	3	2	60

Cicli annidati

Eseguire la trace di questo pgm.

```
int main() {
int i, j, sum1, sum2;
        sum1 = sum2 = 0;
        for(i=1; i <= 5; i++) {
                sum1++;
                for(j=1;  j <= 4; j++)
                        sum2 ++;
        }
        cout<<sum1<<sum2<<endl;
        //printf("%d   %d\n", sum1, sum2);
}
```

Cicli annidati

i	j	sum1	sum2
?	?	0	0
1	1	1	1
1	2	1	2
1	3	1	3
1	4	1	4
2	1	2	5
2	2	2	6
2	3	2	7
2	4	2	8
3	1	3	9
3	2	3	10
3	3	3	11
3	4	3	12
4	1	4	13
4	2	4	14
4	3	4	15
4	4	4	16
5	1	5	17
5	2	5	18
5	3	5	19
5	4	**5**	**20**

Cicli annidati

Problema

Stampare la tavola pitagorica.

Esempio 5x5

			colonne		
	1	2	3	4	5
	2	**4**	6	8	10
righe	3	6	**9**	12	15
	4	8	12	**16**	20
	5	10	15	20	**25**

- In grassetto ho evidenziato la diagonale principale.
- Chiamiamo la “tabella” sopra disegnata: matrice (5x5 o di ordine 5).
- E’ una matrice quadrata in quanto il numero di righe è uguale al numero di colonne.
- Indice di riga: i $\in$ {1,2,3,4,5}
- Indice di colonna: j $\in$ {1,2,3,4,5}
- La matrice disegnata è una matrice simmetrica.

Cicli annidati

- In generale, ma sempre nel caso 5x5:

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$
$a_{4,1}$	$a_{4,2}$	$a_{4,3}$	$a_{4,4}$	$a_{4,5}$
$a_{5,1}$	$a_{5,2}$	$a_{5,3}$	$a_{5,4}$	$a_{5,5}$

- Elemento generico della matrice:

$$a_{i,j}$$

i: indice di riga

j: indice di colonna

Es. $a_{4,2} = 8$

Cicli annidati

- Soluzione

```
int i,j;
    for(i = 1; i <= 10; i++) {
        for(j=1; j <= 10; j++)
            //printf("%3d", i*j);
            cout<<(i*j);
        cout<<'\n';
        //printf("\n");
    }
```

Cicli annidati

- Esercizio

Stampa il triangolo inferiore, compresa la diagonale principale, della tavola pitagorica.

Cicli annidati

- Soluzione

```
int i,j;
    for(i = 1; i <= 10; i++) {
        for(j=1; j <= 10; j++)
          if(i >= j)
            //printf("%3d", i*j);
            cout<<(i*j);
        cout<<'\n';
        printf("\n");
    }
```

Cicli annidati

- Altra soluzione:

```
int i,j;
    for(i = 1; i <= 10; i++) {
        for(j=1; j <= i; j++)
            //printf("%3d", i*j);
            cout<<(i*j);
        cout<<'\n';
        printf("\n");
    }
```

Cicli annidati

- Esercizio

Stampare la tavola di verità della seguente espressione booleana:

a and b or c

Osservazione:

n. 3 variabili booleane, $2^3 = 8$ righe della tavola di verità

Cicli annidati

a	b	c	a and b or c
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Cicli annidati

- Soluzione:

```
int a, b, c;
for(a=0; a <= 1; a++)
    for(b=0; b <= 1; b++)
        for(c=0; c <= 1; c++)
            cout<<a<<b<<c<<(a && b || c)<<endl;
            //printf("%d  %d  %d  %d\n", a, b, c, a && b || c);
```

Generazione di numeri casuali

Per la generazione di numeri casuali, la libreria standard del C ti mette a disposizione le seguenti funzioni:

srand()
rand()

Ti riporto la spiegazione delle funzioni così come riportata nell'help.

Syntax
#include <stdlib.h>
void srand(unsigned seed);
Description
Initializes random number generator.
The random number generator is reinitialized by calling srand with an argument value of 1. It can be set to a new starting point by calling srand with a given seed number.
Return Value
None.

Generazione di numeri casuali

Syntax
#include <stdlib.h>
int rand(void);
Description
Random number generator.
rand uses a multiplicative congruential random number generator with period 2 to the 32nd power to return successive pseudorandom numbers in the range from 0 to RAND_MAX. The symbolic constant RAND_MAX is defined in stdlib.h.
Return Value
rand returns the generated pseudorandom number.

Generazione di numeri casuali

Example

```
#include <stdlib.h>
#include <iostream>

int main(void) {
 int i;
   srand(time(NULL));
   cout<<"Ten random numbers from 0 to 99\n\n";
   for(i=0; i<10; i++)
       cout<<rand() % 100<<endl;
   return 0;
}
```

Riepilogando

NUMERI CASUALI (o meglio pseudocasuali)

rand() dichiarata in <stdlib.h>
per la generazione di numeri pseudo-casuali nell'intervallo compreso tra 0 e RAND_MAX (costante definita in stdlib.h).

Esempi:

```
numero = rand();                //genera un numero casuale
numero = rand() % 100;          //genera un numero casuale tra 0 e 99 (estremi inclusi)
```

Numeri pseudo-casuali ottenuti per via algoritmica, per cui in realtà casuali non lo sono affatto, difatti se non si inizializza l'algoritmo con un opportuno seme la sequenza di valori risulta essere sempre la stessa. La funzione utilizzata allo scopo è:
srand(seme)   dichiarata in <stdlib.h>

Perché sia efficace, il seme deve poter variare. Si usa perciò chiamarla così:

```
srand(time(NULL)); //inizializza il generatore di numeri casuali (va scritta una sola volta, ad es. all'inizio del
                   //programma)
```

time() restituisce l'ora corrente ed è perciò differente ogni volta che la si chiama.

Esercizio

- Generare casualmente un numero intero nell'intervallo chiuso [-50,50].

Soluzione

- num = (rand() %101) - 50;

Esercizio

Scrivi un programma in linguaggio C che consenta ad un giocatore di giocare al gioco "Indovina il numero" contro il computer. Le regole del gioco sono le seguenti:

- Il calcolatore sceglie un numero a caso compreso tra 1 e 100.
- Il giocatore deve indovinare il numero facendo dei tentativi; ad ogni tentativo, se il numero è corretto, il giocatore vince (viene comunicato il numero di tentativi); altrimenti il calcolatore risponde con un suggerimento che può essere "Prova con un numero più alto" oppure "Prova con un numero più basso", a seconda che il tentativo sia più alto o più basso del numero da indovinare.
- Il calcolatore tiene traccia dei tentativi effettuati dal giocatore.
- In qualsiasi momento il giocatore deve poter interrompere la partita (ad esempio inserendo il numero zero) e visualizzare il numero da indovinare.

Scrivi poi un altro pgm che stabilisce un numero massimo di tentativi, esauriti i quali interrompe il gioco e visualizza il numero non indovinato.

Soluzione

```
#define MAX_NUM 100
int main() {
int numCasuale;
int tentativi=0;
int num;
int numCasuale;

        srand(time(NULL));
        numCasuale=rand()%MAX_NUM + 1;              //genera un numero tra 1 e MAX_NUM
        do {
                do{
                      cout<<"Digita un numero intero positivo minore o uguale di “<<MAX_NUM<<endl;
                       cin>>num;
                }while ( (num <= 0) || (num >MAX_NUM));
                if(num > numCasuale)
                                cout<<"Il numero da indovinare e' minore\n";
                else if(num < numCasuale)
                                cout<<"Il numero da indovinare e' maggiore\n";
                tentativi++;
        }while(num != numCasuale);

        cout<<"Hai vinto dopo “<<tentativi<<“ tentativi”<<endl;;
        getchar();
        return 0;
}
```

Variabili di tipo virgola mobile

Tipi per numeri reali: float, double

```
float x;
double y;

    x = 0.347;
    y = 7E+20;                    (= 7*10^20) notazione esponenziale
    cout<<y;
```

- Se la parte decimale non entra completamente nel sottocampo ad essa riservato, le cifre meno significative vengono perdute. Al contrario, se la parte intera è più grande, il campo viene esteso fino a contenerla tutta.

```
#define PI_GRECO 3.14F
```

Operazioni in virgola mobile

- double fabs (double num); //calcola il valore assoluto di un numero reale
- double sqrt (double num); //calcola la radice quadrata
- double pow (double x, double y); //calcola x elevato a y

Operazione di divisione

Operazione di divisione

(in pseudocodice 2 operatori: div e /)

In C/C++, operatore: / (sia con numeri interi che con numeri reali)

Osserva i seguenti casi:

1° caso:

```
int media;
    media = 7 / 2;        //dentro media c'è: 3 (divisione intera)
```

2° caso:

```
double media:
    media = 7 / 2;        //dentro media c'è: 3.0
```

3° caso:

```
double media:
    media = 7.0 / 2; //oppure media = 7/2.0;  oppure media = 7.0/2.0; dentro media c'è: 3.5
```

Operazione di divisione

4° caso:

```
int somma;
int numElem;
double media;
    media = (double) somma / numElem;
```

Ho usato l'operatore di cast: **(nome tipo)**
L'operatore di cast esegue una conversione esplicita di somma da int a double (N.B. La variabile somma rimane di tipo int, è solo una conversione interna).
Dentro media c'è: 3.5
L'operatore di cast puoi metterlo anche a denominatore.

```
media = somma / (double) numElem;
```

Operazione di divisione

5° caso:

```
int somma;
int numElem;
double media;
    media = (double) (somma / numElem);
```

Ho usato l'operatore di cast: (nome tipo)
Avendo messo le parentesi tonde nel modo che vedi: prima viene eseguita la divisione intera, poi il risultato è convertito in double e quindi assegnato alla variabile media.
Evidentemente il risultato non è corretto (vedi caso n. 2).

Operazione di divisione

- **Viene eseguita una divisione intera se entrambi gli operandi (dividendo e divisore) sono interi.**
- **Viene eseguita la divisione reale se almeno uno degli operandi (o tutti e due) è un numero reale.**
- **Se hai solo numeri interi e non vuoi cambiare il tipo della variabile (da int a double) puoi usare l'operatore di cast (conversione esplicita).**

Operazione di divisione

Ultimo esempio:

```
int somma = 5;
int numElem = 100;
double num;
    num = somma / numElem;          //dentro  num c'è:  0
    num = (double) somma / numElem; //dentro num c'è: 0.05
```