

Strutture di controllo

Programmazione strutturata

L'iterazione

Liceo G.B. Brocchi
Classi seconde Scientifico - opzione scienze applicate
Bassano del Grappa, Settembre 2022

Teorema di Boehm-Jacopini (1966)

Qualsiasi algoritmo può essere realizzato tramite le seguenti 3 strutture di controllo:

- **sequenza;**
- **selezione** (detta anche **ramificazione**);
- **iterazione**

Teorema di Boehm-Jacopini (1966)

Qualsiasi algoritmo può essere realizzato tramite un diagramma di flusso strutturato come:

- **sequenza di sotto-diagrammi di flusso;**
- **selezione tra sotto-diagrammi di flusso;**
- **iterazione di un sotto-diagramma di flusso**

Iterazione

```
int i = 5;  
  
while (i < 4) {  
    cout << "hello world" << endl;  
}
```

```
while (5 != 5) {  
    cout << "hello world" << endl;  
}
```

Iterazione

```
while (7 != 8) {  
    cout << "hello world" << endl;  
}
```

```
int c = 0;  
int i = 3;  
  
while (i <= 30) {  
    if (i % 3 == 0) {  
        cout << i << endl;  
        c++;  
    }  
    i = i + 1;  
}  
cout << endl;  
cout << c << endl;
```

Iterazione

```
int previous = -1;  
int input;  
cin >> input;  
  
while (input != -1 && input != previous) {  
    previous = input;  
    cin >> input;  
}
```

```
int d;  
cin >> d;  
int remainder;  
do {  
    remainder = d % 2;  
    cout << remainder;  
    d = d / 2;  
}  
while (d != 0);
```

Iterazione

```
int k = 0;

while (k < 10){
    cout << k << " ";
    k += 1;
}
```

```
int k = 10;

while (k > 0){
    cout << k << " ";
    k -= 1;
}
```

Iterazione

```
int k = 1;  
int s = 0;  
  
while (k <= 100) {  
    s += k;  
    k += 1;  
}
```


Iterazione

```
while (false == false) {}
```

```
while (k < 5 && k > 5) {}
```

Iterazione

```
int total_product = 1;  
int i;  
cin >> i;  
  
while (i != -99999) {  
    total_product *= i;  
    cin >> i;  
}  
  
cout << total_product << endl;
```

“Srotoliamo” il ciclo

Stato del programma prima di testare la condizione del ciclo per la prima volta

Variabile	Valore
total_product	1
i	-99999

```
int total_product = 1;
```

```
int i;
```

```
cin >> i;
```

```
while (i != -99999) {  
    total_product *= i;  
    cin >> i;  
}
```

```
cout << total_product << endl;
```

Il controllo passa al test di permanenza del ciclo

La condizione di permanenza del ciclo è **falsa** ((i == -99999) != -99999 ha valore **false**)

Si «esce» dal ciclo, ossia il controllo passa all'istruzione successiva al ciclo

“Srotoliamo” il ciclo

Stato del programma prima di testare la condizione del ciclo per la prima volta

Variabile	Valore
total_product	1
i	8

```
int total_product = 1;  
int i;  
cin >> i;
```

```
while (i != -99999) {  
    total_product *= i;  
    cin >> i;  
}
```

```
cout << total_product << endl;
```

→ Il controllo passa al test di permanenza del ciclo

→ La condizione di permanenza del ciclo è **vera** ((i == 8) != -99999 ha valore **true**) →

→ Verrà eseguita un'altra iterazione

“Srotoliamo” il ciclo

Stato del programma al termine dell'iterazione 1

Variabile	Valore
total_product	total_product*i: 1*8 = 8
i	5

```
int total_product = 1;  
int i;  
cin >> i;
```

```
while (i != -99999) {  
    total_product *= i;  
    cin >> i;  
}
```

```
cout << total_product << endl;
```

Da dove vengono questi valori?

Il controllo passa al test di permanenza del ciclo

La condizione di permanenza del ciclo è **vera** ((i == 5) != -99999 ha valore **true**)

Verrà eseguita un'altra iterazione

“Srotoliamo” il ciclo

Stato del programma al termine
dell'iterazione 2

Variabile	Valore
total_product	total_product*i: 8*5 = 40
i	3

```
int total_product = 1;  
int i;  
cin >> i;
```

```
while (i != -99999) {  
    total_product *= i;  
    cin >> i;  
}
```

```
cout << total_product << endl;
```

Da dove vengono questi
valori?

Il controllo passa al test di permanenza
del ciclo

La condizione di permanenza del ciclo è
vera ((i == 3) != -99999 ha valore **true**)

Verrà eseguita un'altra iterazione

“Srotoliamo” il ciclo

Stato del programma al termine
dell'iterazione 3

Variabile	Valore
total_product	total_product*i: 40*3 =120
i	2

```
int total_product = 1;  
int i;  
cin >> i;
```

```
while (i != -99999) {  
    total_product *= i;  
    cin >> i;  
}
```

```
cout << total_product << endl;
```

Da dove vengono questi
valori?

Il controllo passa al test di permanenza
del ciclo

La condizione di permanenza del ciclo è
vera ((i == 2) != -99999 ha valore **true**)

Verrà eseguita un'altra iterazione

“Srotoliamo” il ciclo

Stato del programma al termine dell'iterazione 4

Variabile	Valore
total_product	total_product*i: 120*2 = 240
i	-99999

```
int total_product = 1;  
int i;  
cin >> i;
```

```
while (i != -99999) {  
    total_product *= i;  
    cin >> i;  
}
```

```
cout << total_product << endl;
```

Da dove vengono questi valori?

Il controllo passa al test di permanenza del ciclo

La condizione di permanenza del ciclo è **falsa** ((i == -99999) != -99999 ha valore **false**)

Si «esce» dal ciclo: il controllo passa all'istruzione successiva al ciclo

Una rappresentazione alternativa

	total_produc t	i (standard_input)
termine iterazione 0 (prima di valutare la condizione del ciclo la prima volta)	1	-99999



La condizione di
permanenza viene
valutata **false**

Una rappresentazione alternativa

	total_product	i (standard_input)	
termine iterazione 0 (prima di valutare la condizione del ciclo la prima volta)	1	8	→ La condizione di permanenza viene valutata true
termine iterazione 1	1 * 8 = 8	5	→ La condizione di permanenza viene valutata true
termine iterazione 2	8 * 5 = 40	3	→ La condizione di permanenza viene valutata true
termine iterazione 3	40 * 3 = 120	2	→ La condizione di permanenza viene valutata true
termine iterazione 4	120 * 2 = 240	-99999	→ La condizione di permanenza viene valutata false

Iterazione

```
int previous = -1;  
int input;  
cin >> input;  
  
while (input != -1 && input != previous) {  
    previous = input;  
    cin >> input;  
}
```

Srotolare!

```
int previous = -1;  
int input;  
cin >> input;
```

```
while (input != -1 && input != previous) {  
    previous = input;  
    cin >> input;  
}
```

fine iterazione 0

previous

**input
(stdin)**

-1

-1

→ La condizione di permanenza viene valutata **false**

Srotolare!

```
int previous = -1;
int input;
cin >> input;

while (input != -1 && input != previous) {
    previous = input;
    cin >> input;
}
```

fine iterazione 0	previous	input (stdin)	→ La condizione di permanenza viene valutata true				
	-1	5					
fine iterazione 1		previous	input (stdin)	→ La condizione di permanenza viene valutata true			
			7				
fine iterazione 2			previous	input (stdin)	→ La condizione di permanenza viene valutata true		
				8			
fine iterazione 3				previous	input (stdin)	→ La condizione di permanenza viene valutata true	
					4		
fine iterazione 4					previous	input (stdin)	→ La condizione di permanenza viene valutata false
						-1	

Srotolare!

```
int previous = -1;
int input;
cin >> input;

while (input != -1 && input != previous) {
    previous = input;
    cin >> input;
}
```

fine iterazione 0	previous	input (stdin)	→ La condizione di permanenza viene valutata true			
	-1	5				
fine iterazione 1		previous	input (stdin)	→ La condizione di permanenza viene valutata true		
			7			
fine iterazione 2			previous	input (stdin)	→ La condizione di permanenza viene valutata true	
				8		
fine iterazione 3			previous	input (stdin)	→ La condizione di permanenza viene valutata true	
				4		
fine iterazione 4				previous	input (stdin)	→ La condizione di permanenza viene valutata false
					4	

Iterazione

1. Scrivere un programma che accetti in input una successione di interi finché è verificata la condizione seguente: l'intero in input è diverso da 99999 e l'ultimo intero inserito è maggiore o uguale al precedente;
2. Scrivere un programma che accetti in input una successione di interi finché è verificata la condizione seguente: l'intero in input è diverso da 99999 e l'ultimo intero inserito è minore del precedente;
3. Scrivere un programma che accetti in input una successione di interi finché è verificata la condizione seguente: l'intero in input è diverso da 99999 e l'ultimo intero inserito è uguale al doppio del precedente. La condizione di permanenza del ciclo deve essere vera almeno la prima volta che viene valutata;
4. Scrivere un programma che accetti in input una successione di interi finché è verificata la condizione seguente: l'intero in input è diverso da 99999 e l'ultimo intero inserito è uguale al quadrato del precedente. La condizione di permanenza del ciclo deve essere vera almeno la prima volta che viene valutata;
5. Scrivere un programma che memorizzi il valore massimo di una successione di interi letta da standard input finché è verificata la condizione seguente: l'intero in input è diverso da 99999;

Iterazione

Header contenente la macro **INT_MIN**

```
#include <iostream>
#include <climits>
using namespace std;
```

Il numero più piccolo
rappresentabile con il tipo **int**


```
int main (int argc, char* argv[]) {
    int previous = INT_MIN;
    int input;
    cin >> input;

    while (input != 99999 && input >= previous) {
        previous = input;
        cin >> input;
    }

    return 0;
}
```


Iterazione

perché questo tipo?



```
long long previous = 2;  
long long input = 4;
```

```
while (input != 99999 && input == previous*previous) {  
    previous = input;  
    cin >> input;  
}
```

Iterazione

```
int max = INT_MIN;  
int input;  
cin >> input;
```

```
while (input != -99999) {  
    if (input > max) {  
        /*che istruzioni inseriresti qui?*/  
    }  
    cin >> input;  
}
```

Perché questa istruzione non è dentro un eventuale **else** dell'**if** precedente?