Il C nel C++

Struttura di un linguaggio

- Livello lessicale:
  - regole per la definizione i simboli
- Livello sintattico:
  - regole per la composizione dei simboli
- Livello semantico:
  - significato delle strutture sintattiche

Elementi lessicali

- Identificatori C/C++ e Java:
  - Sequenze di lettere (incluso '_') e cifre numeriche. Il primo carattere deve essere una lettera.
  - Sono definiti dal programmatore.

- Parole riservate:
  - Simboli che non possono essere ridefiniti dal programmatore (in C sono ca. 48, in C++ 62, in Java 59).
  - Es: void, while, return, int, catch.

Spazi

- In un programma C/C++ o Java tra due elementi lessicali è possibile porre una combinazione qualunque di “spazi bianchi” e di “a capo”.
- Questa possibilità viene solitamente usata per rendere più leggibile il programma.

Tipi di dato primitivi

- **interi**
  - byte 8 bit 127 -128
  - short 16 bit 32,767 -32,768
  - int 32 bit 2,147,438,647 -2,147,438,648
  - long 64 bit 9,223,372,036,854,775,807 (circa 9.2E18)
- **reali**
  - float 32 bit 3.4e38 (circa)
  - double 64 bit 1.8e308 (circa)
- **caratteri**
  - char 8 bit (ASCII) per C/C++
    16 bit (UNICODE) per Java
- **logici** (SOLO C++ e Java)
  - bool (true, false)

Elementi lessicali - 2

- Operatori:
  - Simboli che indicano operazioni tra operandi
  - (circa 50 organizzati su 16 livelli di precedenza).
  - Es: +, <, > ,=, *.

- Direttive del preprocessore (solo C):
  - Usate per inclusione di file e compilazione condizionale.
  - Cominciano con # e terminano alla fine della riga.

Operatori

Gruppo	Funzione	Operatori
Arithmetic	comparazione	= =, !=, <, <=, >, >=
	unitari	+, -
	algebrici	+, -, *, /, %
	postfissi	++,--
Bit	shift	<<, >>, >>>
	bitwise comparison	~, &, |, ^
Arithmetic(c++)	logici	&, | , ^ , && , ||
String(c++/Java)	concatenazione	+

Elementi lessicali - 3

- Costanti:
  - Rappresentano valori costanti nel programma.
  - Seguono regole diverse per tipi di dato diversi (numeri interi, numeri reali, caratteri, stringhe...).
  - Es: 225, 3.1416, ‘x’, “Ciao a tutti”.
- Commenti:
  - Sono delimitati da /* ... */ oppure da // e fine riga.

Strutture di controllo: if-else

Questa espressione viene valutata e il risultato viene considerato un valore logico (*true* o *false*).

```
if ( espressione logica )
     istruzione
else
     istruzione
```

Può essere una singola istruzione o un blocco indicato tra parentesi graffe

If-else

```
#include <iostream>
using namespace std;
void main() {
  int x;
  cin >> x;
  if(x==0)
    cout << “Valore nullo\n ”;
  else {
    cout << “ Valore non nullo\n ”;
    cout << “ Grazie e ciao\n ”);
  }
}
```

Valori logici

- In C i valori logici sono memorizzati all'interno di variabili numeriche (int).
  - (0 => false, qualsiasi altro numero => true)

- In C++ i valori logici (true e false) sono memorizzati all'interno di variabili booleane (tipo **bool**)

Operatori di confronto

- Confrontano due operandi (dello stesso tipo o di tipi compatibili...) e restituiscono un valore logico

==	uguaglianza
!=	disuguaglianza
< > >= <=	ordinamento

Operatori logici

- Sono operatori che effettuano operazioni logiche e restituiscono un valore logico.
- Il loro comportamento può essere definito tramite tabelle di verità

||	OR
&&	AND
!	NOT

La scaletta if-else if-else

```
...
if(guess==magic)
  cout << “Right!\n”;
else
  if(guess<magic)
    cout << “Too low...\n”;
  else
    cout << “Too high...\n”;
...
```

Ifthen.cpp

Selezione Multipla: Switch

```
switch ( espressione ) {
    case costante: istruzioni
    case costante: istruzioni
    default: istruzioni
}
```

Per evitare “sfondamenti” è meglio terminare il gruppo di istruzioni con break

Viene eseguito se nessun caso è applicabile

Selezione multipla

```
int x;
cin >> x;
switch(x) {
 case 1:  cout << “Uno\n”);
          break;
 case -1: cout << “Uno negativo\n”;
          break;
 case 0:  cout << “Valore nullo\n”;
          break;
 default: cout << “Valore non corretto\n”;
}
```

Ciclo for

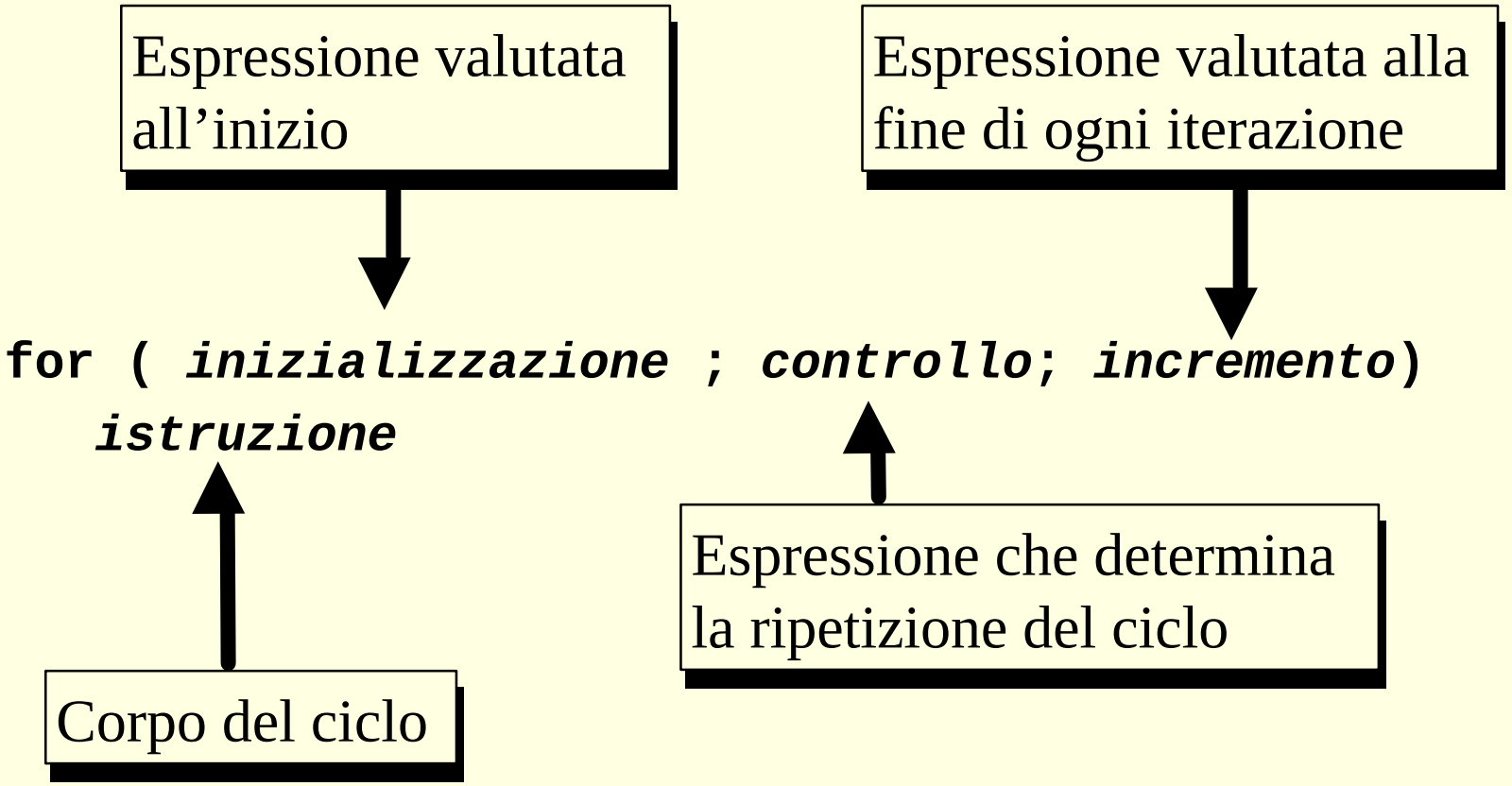
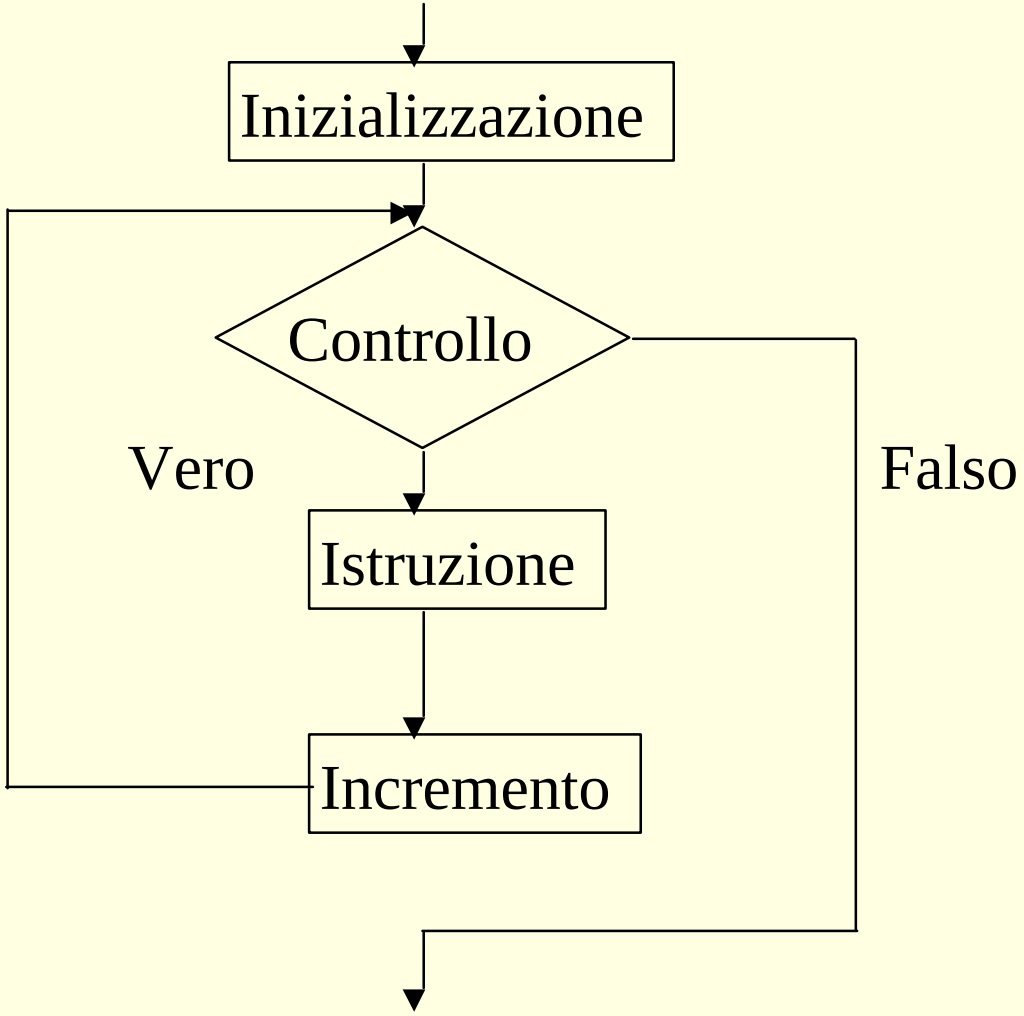

Espressione valutata all'inizio
Espressione valutata alla fine di ogni iterazione
for (inizializzazione ; controllo; incremento)
istruzione
Espressione che determina la ripetizione del ciclo
Corpo del ciclo

Diagramma di flusso - for



Cicli

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

void main() {
  for(int i=0; i<10; i++)
    cout << i;
}
```

Charlist.cpp

Esercizio

- Scrivere un programma che stampi i numeri pari da 0 a 40. Utilizzare una istruzione for.

Un for ellittico...

- La scrittura:

```
for(;;)
   printf (“Immortale...”);
```

- è perfettamente lecita, come anche:

```
for(;;);   // Noioso
```

- Ogni elemento è opzionale e può essere eliminato.

Filtri a caratteri

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

void main() {
  char c;
  for( cin.get(c) ; c!='0'; cin.get(c) )
    cout.put(c);
}
```

filtro.cpp

Filtro per esercizio

- Costruire un filtro a caratteri che mandi in output solo le cifre numeriche lette da input

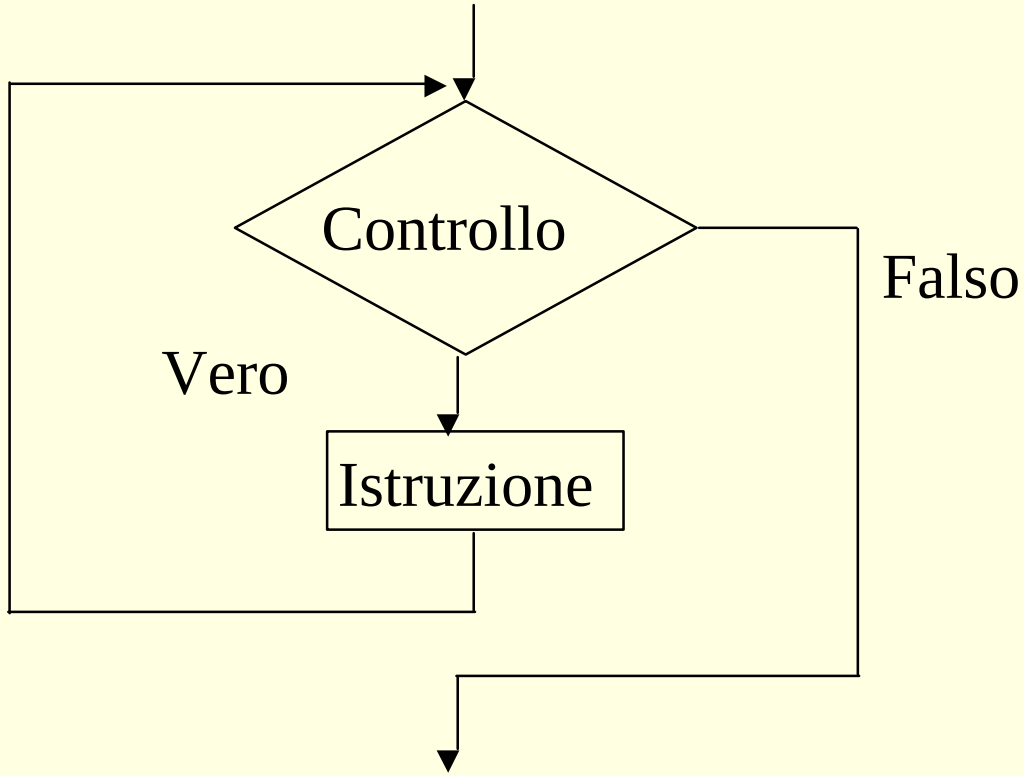
While

Espressione che determina, se falsa, l'uscita dal ciclo

```
while ( controllo )
    istruzione
```

Corpo del ciclo

Diagramma di flusso - while



While

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

void main() {
  int x=0;
  while(x<10) {
    cout << x;
    x++;
  }
}
```

Guess.cpp
while.cpp

Traduzione

- Tradurre il filtro a caratteri costruito precedentemente usando while al posto di for

Do-while

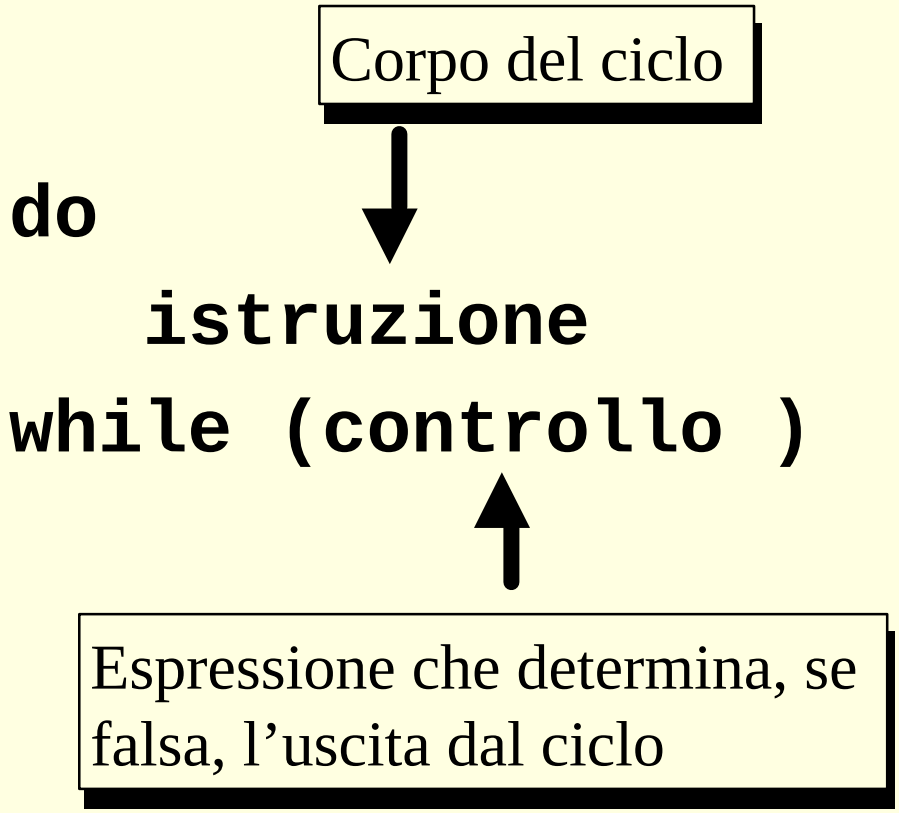
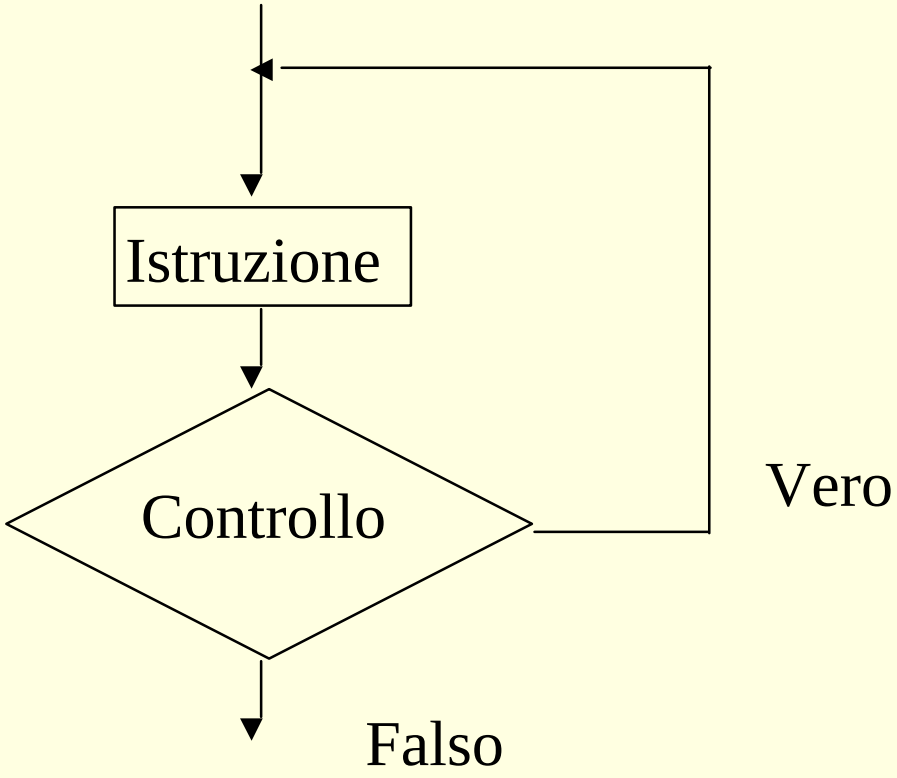

Corpo del ciclo
do
istruzione
while (controllo)
Espressione che determina, se falsa, l'uscita dal ciclo

Diagramma di flusso - do-while



Do-while

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

void main() {
  char c;
  do {
    c=cin.get();
    if(c!=EOF)
      cout.put(c);
  } while(c!=EOF);
}
```

Guess2.cpp

dowhile.cpp

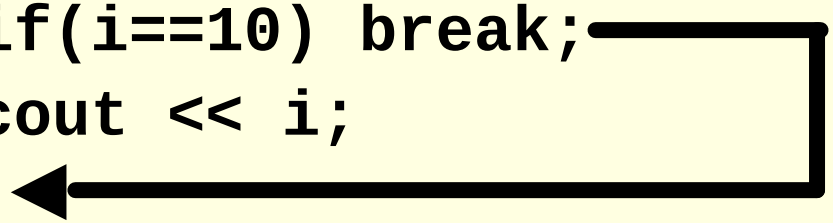
Break

- Determina l'uscita dalla struttura di controllo di flusso più interna in cui si trova.
- Solitamente è usato all'interno di switch, ma può anche essere usato in while, for, do-while.

Esempio

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

void main() {
  for(int i=0;i<100;i++) {
    if(i==10) break;
    cout << i;
  }
}
```



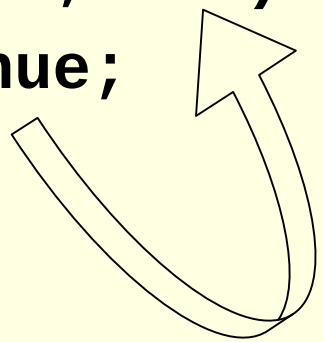
Continue

- Forza la prossima iterazione del ciclo in cui è introdotto
- In **while** e **do-while**, il controllo passa alla valutazione della espressione di uscita
- In **for** il controllo passa alla espressione di incremento e poi a quella di uscita

Esempio

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

void main() {
   for(int i=-20; i<20; i++) {
      if(i==0) continue;
      cout << i;
   }
}
```



continue.cpp

Esercizio

- Costruire un filtro a caratteri (usando cin.get() e cout.put()) che mandi in output solo i caratteri racchiusi da doppi apici.
  - Es:
  - alfa”beta”gamma  beta