

# Lezione 4

---

## Programmazione strutturata Istruzioni condizionali

---

# Programmazione strutturata

# Problema 1/2

---

- Supponiamo di dover svolgere il seguente esercizio, utilizzando solo le istruzioni apprese finora
- Scrivere un programma che legge un numero intero da standard input (*cin*, *stdin*) che rappresenta il voto preso in Programmazione I e, se il voto è superiore a 27, stampa

*Programmazione I è veramente uno dei migliori corsi di Informatica!!*

Altrimenti stampa:

*Quel def ... di docente di quel noioso ...*

# Problema 2/2

---

- E' praticamente impossibile
- Come mai?

# Risposta e nuova domanda

---

- Perché, siccome le istruzioni sono eseguite l'una dopo l'altra, ogni volta che si fa partire il programma si eseguono sempre le stesse istruzioni
- Noi invece vorremmo che in un caso si stampasse qualcosa, e nell'altro caso qualcos'altro
  - Ossia che in un caso si eseguisse una certa istruzione, e nell'altro caso un'altra
- Supponendo di poter aggiungere nuove istruzioni a quelle che conosciamo, come potremmo riuscire a raggiungere questo scopo?

# Risposta e nuova domanda

---

- Una soluzione sarebbe la seguente
  - All'interno del programma inseriamo sia le istruzioni da eseguire in un caso che le istruzioni da eseguire nell'altro caso
  - Dopo l'istruzione di lettura del valore da stdin
    - inseriamo un controllo, in cui valutiamo quale dei due casi sia vero
    - e **saltiamo** al pezzo di codice relativo a quel caso

# Salto: goto, jump, ...

---

- Il salto è la prima tecnica adottata, storicamente, per eseguire passi diversi a seconda dei dati passati in ingresso
  - In generale, a seconda del valore di qualche condizione
- I tipici nomi delle istruzioni di salto sono:
  - **goto**            nei linguaggi ad alto livello
  - **jump**            nel linguaggio macchina

# Salti e cicli

---

- Le istruzioni di salto permettono anche di ripetere più volte l'esecuzione di un dato pezzo di codice
  - Un pezzo di codice che si ripete più volte viene tipicamente chiamato *ciclo*
  - Per realizzarlo si può utilizzare una istruzione di salto per saltare all'indietro
    - saltare cioè all'inizio del ciclo quando lo si vuole ripetere, e proseguire invece dall'istruzione successiva alla fine del ciclo quando si vuole smettere



# Problemi salto

---

- Stiamo quindi per studiare le istruzioni di salto?
- No
- Come mai?

# Logica a spaghetti 1/2

---

- Perché, se si realizza la logica di un programma mediante salti avanti ed indietro, il programma stesso tende a diventare molto difficile da capire
- A meno che il programmatore non applichi delle regole rigide nell'utilizzo delle istruzioni di salto,
  - si tende alla cosiddetta **logica a spaghetti**

# Logica a spaghetti 2/2

---

- La sequenza di esecuzione delle istruzioni tende cioè a diventare un groviglio



Roberto Gianferrari 16/7/2011

# Quali regole?

---

- Quali sono le regole rigide a cui abbiamo accennato?
- Sono delle regole che fanno sì che l'ordine di esecuzione delle istruzioni coincida con l'ordine che si può ottenere utilizzando solo i costrutti della cosiddetta programmazione strutturata

# Programmazione strutturata

---

- Si parla di **programmazione strutturata** [Dijkstra, 1969] se si utilizzano solo i seguenti costrutti per determinare l'ordine di esecuzione delle istruzioni (detto anche flusso di controllo):
  - **concatenazione e composizione**
    - conosciamo già la concatenazione, mentre la composizione permette di 'trattare' una sequenza di istruzioni come se fosse una sola istruzione
  - **selezione (istruzione condizionale)**
    - fa proseguire il flusso di controllo tra due possibili rami in base al valore vero o falso di una espressione detta *condizione di scelta*
  - **iterazione**
    - permette l'esecuzione ripetuta di un'istruzione o di una sequenza di istruzioni finché permane vera una espressione detta "condizione di iterazione"

# Scopo e possibili limiti

---

- Rendere i programmi più leggibili e facili da mantenere
- Perdiamo qualcosa se utilizziamo solo i costrutti della programmazione strutturata nei nostri programmi?
- Ossia, rischiamo di non essere in grado di codificare qualche algoritmo?
- Ci vuole un pizzico di teoria ...

# Macchina di Turing

---

- Macchina dotata di
  - una testina
  - un nastro costituito da un numero di celle adiacenti concettualmente infinito
- La testina può: spostarsi da una cella all'altra, leggere/scrivere la cella su cui si trova
- <http://www.google.com/doodles/alan-turings-100th-birthday>
  - Se siete curiosi cercatevi le istruzioni su quale è lo scopo del *doodle* e su come programmare la macchina di turing per cercare di raggiungere lo scopo



# Tesi di Church-Turing

---

- Ogni algoritmo può essere eseguito (calcolato) da una **Macchina di Turing**
- Questa tesi è indimostrabile, o perlomeno mai dimostrata, ma è ormai universalmente accettata



# Teorema di Jacopini-Boem

---

- Assumendo che la tesi di Church-Turing sia vera, tale teorema afferma che ogni algoritmo può essere tradotto in un programma scritto con un linguaggio caratterizzato solo da
  - **Tipo di dato:** Naturali con l'operazione di somma (+)
  - **Istruzioni:** assegnamento  
istruzione composta  
istruzione condizionale  
istruzione di iterazione
- Quindi con la programmazione strutturata si può esprimere qualsiasi algoritmo

- In questa prima presentazione vedremo
  - la selezione
    - ossia le istruzioni condizionali
  - la composizione
    - ossia le istruzioni composte

---

# Istruzioni condizionali

# Istruzioni condizionali

---

- In C/C++ disponiamo di due tipi di istruzioni condizionali:
  - Istruzione di SCELTA SEMPLICE o ALTERNATIVA
  - Istruzione di SCELTA MULTIPLA  
Non è essenziale, ma migliora l'espressività del linguaggio

# Scelta semplice

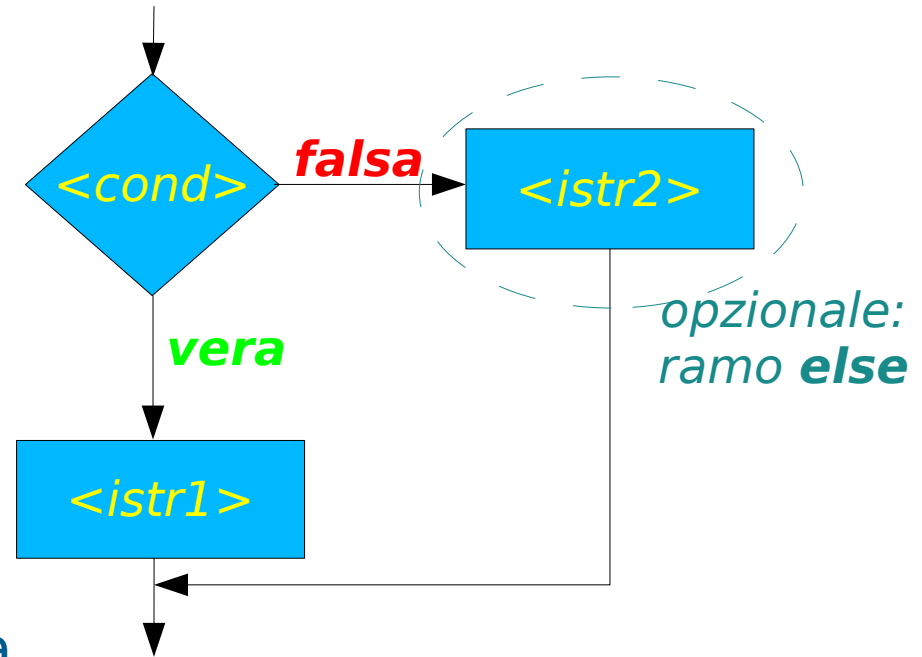
- Consente di scegliere fra due istruzioni alternative in base al verificarsi di una data *condizione*

```
<istruzione-di-scelta> ::=  
    if (<condizione>  
        <istruzione1>  
    [ else  
        <istruzione2> ]
```

- <condizione> è un'espressione logica che viene valutata al momento dell'esecuzione dell'istruzione **if**

# Diagramma di flusso

```
if (<condizione>
    <istruzione1>
[ else
    <istruzione2> ]
```



- Se *<condizione>* risulta vera si esegue *<istruzione1>*, altrimenti si esegue *<istruzione2>*
- In entrambi i casi l'esecuzione continua poi con l'istruzione che segue l'istruzione **if**.
- **NOTA**  
Se *<condizione>* è falsa e la parte **else** (opzionale) è omessa, si passa subito all'istruzione che segue l'istruzione **if**

```
int a=3, n=-6, b=0;  
if (n <= 0)  
    a = b + 5;
```

- Alla fine dell'esecuzione
  - `a == ?`
  - `b == ?`
  - `n == ?`

```
int a=3, n=-6, b=0;  
if (n > b)  
    a = b + 5;  
else  
    n = b*5;
```

- Alla fine dell'esecuzione
  - `a == ?`
  - `b == ?`
  - `n == ?`



- Svolgere gli esercizi della terza esercitazione fino alla slide 21

# Problema

---

- E se vogliamo eseguire più di una istruzione in uno dei due rami o in entrambi?
- Esempio:

```
if (<condizione> )  
    <varie istruzioni>  
else  
    <varie istruzioni>
```

Abbiamo bisogno delle *istruzioni composte* ...

---

# Istruzioni composte

# Istruzione composta

---

- Sequenza di istruzioni racchiuse tra parentesi graffe:  
{  
    <*istruzione1*>  
    <*istruzione2*>  
    ...  
}
- Ovunque la sintassi preveda una istruzione si può inserire tanto una istruzione *semplice* (ossia non composta) che una istruzione *composta*
- Ai fini della sintassi e della semantica, una istruzione composta è trattata come una singola istruzione semplice
- L'esecuzione di una istruzione composta implica l'esecuzione ordinata di tutte le istruzioni della sequenza tra parentesi graffe

---

# Completamento istruzioni di scelta semplice

# Forma completa

---

- Identica a quella già vista:  
 $\langle \textit{istruzione-di-scelta} \rangle ::=$   
    **if** ( $\langle \textit{condizione} \rangle$ )  $\langle \textit{istruzione-ramo-if} \rangle$   
    [ **else**  $\langle \textit{istruzione-ramo-else} \rangle$  ]
- Sia l'istruzione del ramo **if** che quella del ramo **else** possono essere una qualsiasi istruzione semplice (istruzione espressione, istruzione condizionale, istruzione iterativa) o composta
- Le istruzioni alternative da eseguire sono spesso chiamate anche *corpo del ramo if* o *corpo del ramo else*

```
if (n > 0)
    { /* inizio blocco */
        a = b + 5;
        c = x + a - b;
    } /* fine blocco */
else
    n = b*5;
```

# Esercizio

---

- Svolgere la terza esercitazione fino alla slide 49



# Istruzioni di scelta annidate

---

- Come caso particolare, *<istruzione-ramo-if>* o *<istruzione-ramo-else>* potrebbero essere a loro volta un'istruzione di scelta

- Esempio:

```
if (n > 0)
    if (a>b) n = a;
    else n = b*5;
```

- A quale **if** è associato il ramo **else**, il primo o il secondo?

- In base alla sintassi del linguaggio C/C++, un ramo **else** è sempre associato all'**if** più interno (vicino)
- Se questa non è l'associazione desiderata, occorre racchiudere l'**if** più interno in un blocco { }
- Cerchiamo di capire meglio con degli esempi

# Esempi 1/2

---

```
NO → if (n > 0)
SI  → if (a > b) n = a;
    else n = b*5; // associato all'if
                  // più interno
                  // (vicino)
```

# Esempi 2/2

---

Per far sì che l'`else` si riferisca al primo `if`:

```
if (n > 0){
    if (a>b)
        n = a;
} else
    n = b*5;
```

Per maggiore leggibilità, si possono usare le parentesi anche nell'altro caso:

```
if (n > 0) {
    if (a>b) n = a;
    else n = b*5;
}
```

- Risolvere il problema alla slide 50 della terza esercitazione
- Svolgere gli esercizi per casa della terza esercitazione

---

# Istruzioni di scelta multipla

# Esercizio

---

- Scrivere un programma che legge un valore intero e, in base al valore letto, stampa uno dei seguenti messaggi

Valore letto

**1**

**2**

**3**

**4**

**5**

Messaggio

**Primo**

**Secondo**

**Terzo**

**Quarto**

**Quinto**

- Scrivere il programma in maniera tale da non eseguire mai codice inutilmente

# Tentativo di soluzione

---

```
main()
{
    int n; cin>>n ;
    if (n == 1)
        cout<<"Primo"<<endl ;
    if (n == 2)
        cout<<"Secondo"<<endl ;
    if (n == 3)
        cout<<"Terzo"<<endl ;
    if (n == 4)
        cout<<"Quarto"<<endl ;
    if (n == 5)
        cout<<"Quinto"<<endl ;
}
```



- La precedente soluzione rispetta fedelmente la traccia?

- No
- Se, ad esempio,  $n == 1$ , dopo aver stampato **Primo**, si eseguono lo stesso altri quattro controlli inutili sul valore di  $n$

# Soluzione corretta

---

```
main()
{
    int n; cin>>n ;
    if (n == 1)
        cout<<"Primo"<<endl ;
    else if (n == 2)
        cout<<"Secondo"<<endl ;
    else if (n == 3)
        cout<<"Terzo"<<endl ;
    else if (n == 4)
        cout<<"Quarto"<<endl ;
    else if (n == 5)
        cout<<"Quinto"
            <<endl ;
}
```

- Quanto è leggibile la precedente soluzione?
  - Molto poco
- Avremmo bisogno di un costrutto sintattico che ci permetta di eseguire solo l'istruzione giusta in base al valore della variabile
  - Senza la pesantezza sintattica in cui si incorre utilizzando l'istruzione condizionale

# Istruzione di scelta multipla

---

- Consente di scegliere fra molti casi in base al valore di un'**espressione di selezione**

# Sintassi e semantica 1/3

---

```
<istruzione-di-scelta-multipla> ::=  
    switch (<espressione di selezione>) {  
        case <etichetta1> : <sequenza_istruzioni1> [ break; ]  
        case <etichetta2> : <sequenza_istruzioni2> [ break; ]  
        ...  
        [ default : <sequenza_istruzioniN> ]  
    }
```

<espressione di selezione> è un'espressione che restituisce un valore **numerabile** (intero, carattere, enumerato, ...), e viene valutata al momento dell'esecuzione dell'istruzione switch

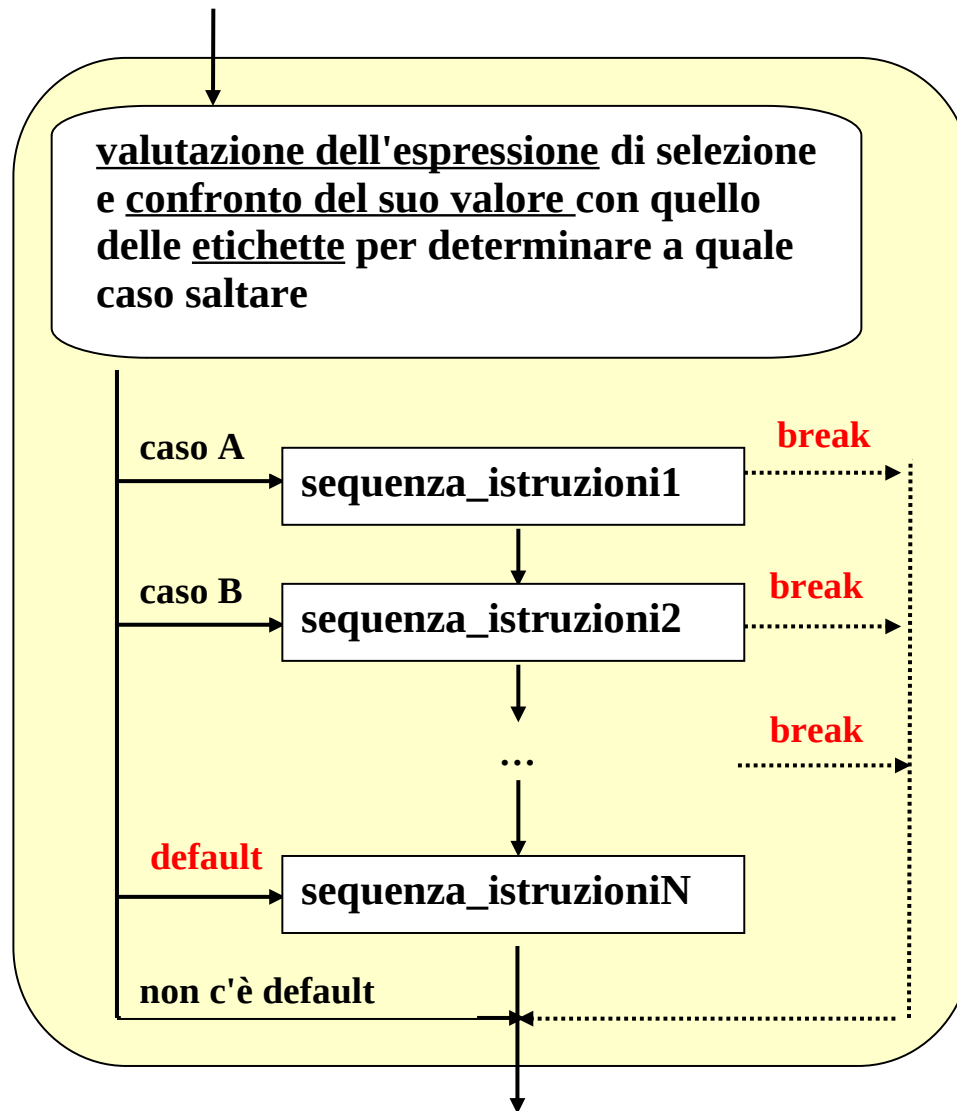
Le etichette <etichetta1>, <etichetta2>, ... devono essere delle costanti dello stesso tipo dell'espressione di selezione

# Sintassi e semantica 2/3

---

- Definiamo **corpo dell'istruzione switch**, la parte del costrutto compresa tra le parentesi graffe
- Il valore dell'espressione di selezione viene confrontato con le **costanti** che etichettano i vari casi
  - L'esecuzione salta al ramo dell'etichetta corrispondente, se esiste
- Vedi diagramma di flusso nella prossima slide (la spiegazione continua poi nella slide ancora successiva)

# Diagramma di flusso





# Sintassi e semantica 3/3

---

- Dopo il salto al ramo di una delle etichette
  - L'esecuzione prosegue poi **sequenzialmente** fino alla fine del corpo dell'istruzione **switch**
    - **A meno che non si incontri un'istruzione break**, nel qual caso si esce dal corpo dello **switch**: ossia l'esecuzione prosegue dall'istruzione successiva all'istruzione **switch**
- Se nessuna etichetta corrisponde al valore dell'espressione, si salta al ramo **default** (se specificato)
  - Se tale ramo non esiste, l'esecuzione prosegue con l'istruzione successiva all'istruzione **switch**

# Esempio/esercizio

---

```
int a = 2, n ;
cin>>n; // considerare separatamente i casi in cui
        // l'utente immette 1, 2, 3, 4, oppure 0
switch (n){
    case 1:
        cout<<"Ramo A"<<endl;
        break;
    case 2:
        cout<<"Ramo B"<<endl;
        a = a*a;
        break;
    case 3:
        cout<<"Ramo C"<<endl;
        a = a*a*a;
        break;
    default:
        a=1;
}
cout<<a<<endl; // cosa viene stampato ?
```

# Osservazioni

---

- *<sequenza\_istruzioni>* denota una sequenza di istruzioni, quindi non è necessaria un'istruzione composta
  - L'idea è che **si salta** all'inizio di uno dei rami
- In accordo al punto precedente, i vari rami non sono mutuamente esclusivi: una volta saltato all'inizio di un ramo, l'esecuzione prosegue in generale con le istruzioni dei rami successivi fino alla fine del corpo dello **switch**
- Per avere rami mutuamente esclusivi occorre forzare esplicitamente l'uscita mediante l'istruzione **break**

# Esempio/esercizio

---

```
int a = 2, n, b = 1;
cin>>n; // considerare separatamente i casi in cui
        // l'utente immette 0, 1, 2, 3
switch (2 - n) {
    case 0:
        b *= a;
    case 1:
        b *= a;
    case 2:
        break;
    default:
        cout<<"Valore non valido per n\n" ;
}
cout<<b<<endl; // cosa viene stampato ?
```

# Esercizio

---

- Svolgere l'esercizio *primo\_menu.cc* della quarta esercitazione
- Non dimenticare di inserire, dove necessaria, l'istruzione **break**;

# Esercizio

---

- Svolgere gli esercizi *menu\_multiplo.cc* e *calcolatrice.cc* della quarta esercitazione

# Pro e contro scelta multipla

---

- L'istruzione **switch** garantisce maggiore leggibilità rispetto all'**if** quando c'è da scegliere tra più di due alternative
- Altrimenti è ovviamente un costrutto più ingombrante
- Ulteriori limitazioni dell'istruzione **switch**:
  - è utilizzabile solo con espressioni ed etichette di tipo numerabile (intero, carattere, enumerato, ...)
  - non è utilizzabile con numeri reali (**float**, **double**) o con tipi strutturati (stringhe, vettori, strutture...)

- Terminare la quarta esercitazione