
Laboratorio di programmazione

Lezione I

Tatiana Zolo

`zolo@cs.unipr.it`

L'ORGANIZZAZIONE DEL COMPUTER

→ Unità di input: sezione ricevente.

dispositivi di input $\xrightarrow{\text{informazione}}$ altre unità per l'elaborazione.

L'ORGANIZZAZIONE DEL COMPUTER

→ Unità di input: sezione ricevente.

dispositivi di input $\xrightarrow{\text{informazione}}$ altre unità per l'elaborazione.

→ Unità di output: sezione di spedizione.

$\xrightarrow{\text{informazione elaborata}}$ *dispositivi di output* → disponibilità all'esterno.

L'ORGANIZZAZIONE DEL COMPUTER

→ **Unità di input:** sezione ricevente.

dispositivi di input $\xrightarrow{\text{informazione}}$ altre unità per l'elaborazione.

→ **Unità di output:** sezione di spedizione.

$\xrightarrow{\text{informazione elaborata}}$ *dispositivi di output* → disponibilità all'esterno.

→ **Unità di memoria:** sezione di magazzino con accesso rapido e capacità relativamente bassa del computer. Conserva

→ informazioni immesse dall'unità di input $\Rightarrow$ disponibilità immediata;

→ informazioni già elaborate $\Rightarrow$ per invio a dispositivi di output.

L'ORGANIZZAZIONE DEL COMPUTER

→ **Unità di input:** sezione ricevente.

dispositivi di input $\xrightarrow{\text{informazione}}$ altre unità per l'elaborazione.

→ **Unità di output:** sezione di spedizione.

$\xrightarrow{\text{informazione elaborata}}$ *dispositivi di output* → disponibilità all'esterno.

→ **Unità di memoria:** sezione di magazzino con accesso rapido e capacità relativamente bassa del computer. Conserva

→ informazioni immesse dall'unità di input $\Rightarrow$ disponibilità immediata;

→ informazioni già elaborate $\Rightarrow$ per invio a dispositivi di output.

→ **Unità aritmetica e logica (ALU):** sezione di produzione.

→ Responsabile dell'esecuzione dei calcoli (meccanismi di decisione).

L'ORGANIZZAZIONE DEL COMPUTER

→ **Unità di input:** sezione ricevente.

dispositivi di input $\xrightarrow{\text{informazione}}$ altre unità per l'elaborazione.

→ **Unità di output:** sezione di spedizione.

$\xrightarrow{\text{informazione elaborata}}$ *dispositivi di output* → disponibilità all'esterno.

→ **Unità di memoria:** sezione di magazzino con accesso rapido e capacità relativamente bassa del computer. Conserva

→ informazioni immesse dall'unità di input $\Rightarrow$ disponibilità immediata;

→ informazioni già elaborate $\Rightarrow$ per invio a dispositivi di output.

→ **Unità aritmetica e logica (ALU):** sezione di produzione.

→ Responsabile dell'esecuzione dei calcoli (meccanismi di decisione).

→ **Unità di elaborazione centrale (CPU):** sezione amministrativa.

→ Coordinatore del computer.

L'ORGANIZZAZIONE DEL COMPUTER

→ **Unità di input:** sezione ricevente.

dispositivi di input $\xrightarrow{\text{informazione}}$ altre unità per l'elaborazione.

→ **Unità di output:** sezione di spedizione.

$\xrightarrow{\text{informazione elaborata}}$ *dispositivi di output* → disponibilità all'esterno.

→ **Unità di memoria:** sezione di magazzino con accesso rapido e capacità relativamente bassa del computer. Conserva

→ informazioni immesse dall'unità di input $\Rightarrow$ disponibilità immediata;

→ informazioni già elaborate $\Rightarrow$ per invio a dispositivi di output.

→ **Unità aritmetica e logica (ALU):** sezione di produzione.

→ Responsabile dell'esecuzione dei calcoli (meccanismi di decisione).

→ **Unità di elaborazione centrale (CPU):** sezione amministrativa.

→ Coordinatore del computer.

→ **Unità di memoria secondaria:** sezione di magazzino a lungo termine e con alta capacità del computer. Programmi e dati non utilizzati dalle altre unità → dispositivi di memoria secondaria (es. dischi).

LINGUAGGI

→ Linguaggi macchina

- Unico linguaggio che la CPU è in grado di comprendere.
- Legame con la progettazione dell'hardware del computer.
- Sequenza di numeri ($\Rightarrow$ 0 e 1) che ordinano al computer di eseguire le operazioni più elementari.
- es. programma paga:

+1300042774

+1400593419

+1200274027

LINGUAGGI

→ Linguaggi macchina

- Unico linguaggio che la CPU è in grado di comprendere.
- Legame con la progettazione dell'hardware del computer.
- Sequenza di numeri ($\Rightarrow$ 0 e 1) che ordinano al computer di eseguire le operazioni più elementari.
- es. programma paga:

+1300042774
+1400593419
+1200274027

→ Linguaggi assembly

- Riscrittura “simbolica” del linguaggio macchina.
- Sviluppo di programmi traduttori: [assembler \(assemblatori\)](#).
- es. programma paga:

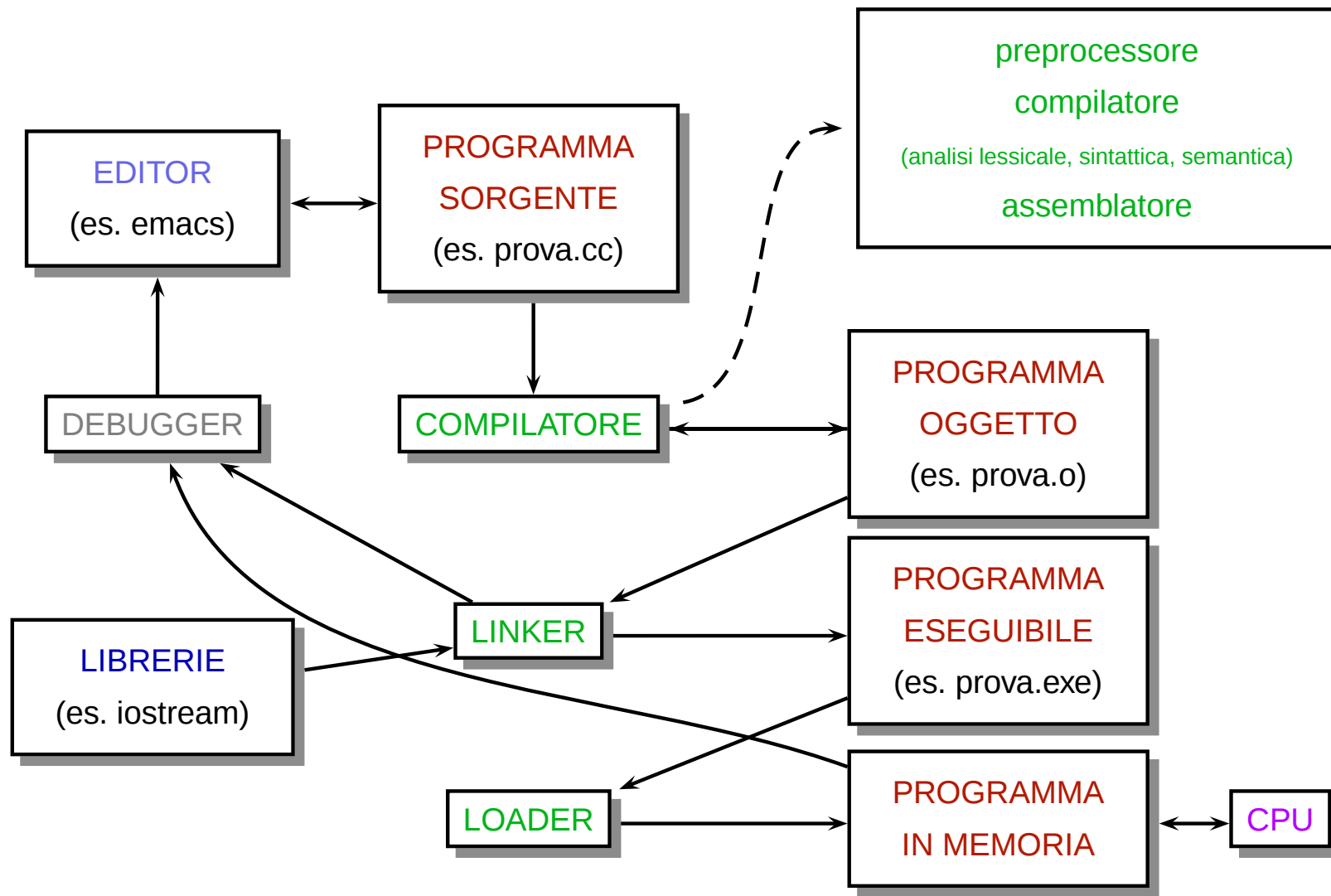
LOAD BASEPAY
ADD OVERPAY
STORE GROSSPAY

LINGUAGGI

→ Linguaggi di alto livello

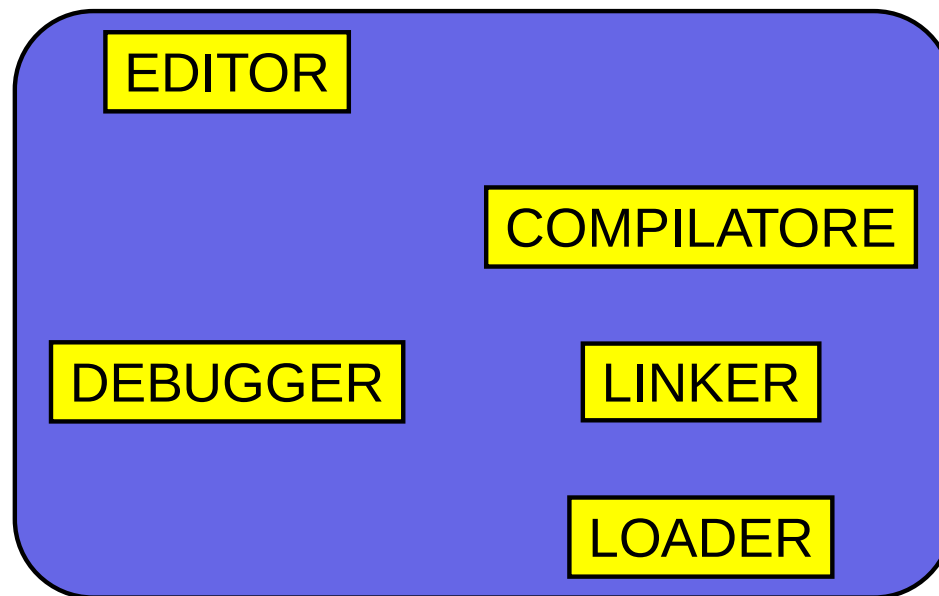
- Con una singola istruzione si può eseguire un compito essenziale.
- Maggiore orientamento verso il problema da trattare.
- Sviluppo di programmi traduttori: [compilatori](#).
- Il risultato della compilazione (e del linking) è indipendente dai sorgenti ed autocontenuto.
- es. programma paga:
$$\text{grossPay} = \text{basePay} + \text{overTimePay}$$
- Il **C** ed il **C++** sono tra i più potenti e più diffusamente utilizzati linguaggi di alto livello.

LA PROGRAMMAZIONE



AMBIENTE DI SVILUPPO (JFE, DEV-C++, ...)

- L'ambiente di sviluppo integra i componenti per la creazione di codice eseguibile a partire dal listato del programma.



es. programma media.cc.

INPUT E OUTPUT

Libreria `iostream`: inclusione del file header di sistema associato `#include <iostream>`.

1. **standard input** → `cin`: consente di leggere dati dal terminale dell'utente.
2. **standard output** → `cout`: permette di scrivere dati sul terminale dell'utente.
3. **standard error** → `cerr`: su di esso vengono inviati i messaggi di errore del programma.

LETTURA DI UNO STREAM

- `>>` operatore di estrazione per operatori di lettura (input).
- **Sintassi**: `stream_utilizzato >> variabile_tipata`.
- **Semantica**: legge da `stream_utilizzato` (o attende finché arrivano i dati) e assegna la lettura alla `variabile_tipata`.

LETTURA DI UNO STREAM

- `>>` operatore di estrazione per operatori di lettura (input).
- **Sintassi**: `stream_utilizzato >> variabile_tipata`.
- **Semantica**: legge da `stream_utilizzato` (o attende finché arrivano i dati) e assegna la lettura alla `variabile_tipata`.

→ Esempio:

```
int x;  
cin >> x;
```

LETTURA DI UNO STREAM

- `>>` operatore di estrazione per operatori di lettura (input).
- **Sintassi**: `stream_utilizzato >> variabile_tipata`.
- **Semantica**: legge da `stream_utilizzato` (o attende finché arrivano i dati) e assegna la lettura alla `variabile_tipata`.

→ Esempio:

```
int x;  
cin >> x;
```

→ Esempio: cascata di letture in ordine da sinistra a destra

```
int x, y;  
cin >> x >> y;
```

Stream cin: 3 5 1 ...

$(x, y) = (?, ?)$, $(x, y) = (3, ?)$, il 3 è *consumato* dallo stream

$(x, y) = (3, 5)$.

Le prossime letture partiranno dal numero 1.

STREAMS FORMATTATI

In base al tipo di dato che scorre sullo stream viene cambiata la formattazione. Per esempio `char 'A'` e `int 65` in memoria sono rappresentati allo stesso modo, ma vengono convertiti diversamente sullo stream! (es. `format_A_65.cc`)

STREAMS FORMATTATI

In base al tipo di dato che scorre sullo stream viene cambiata la formattazione. Per esempio `char 'A'` e `int 65` in memoria sono rappresentati allo stesso modo, ma vengono convertiti diversamente sullo stream! (es. `format_A_65.cc`)

→ Esempio:

```
int x;  
cin >> x;  
cin = -12  $\implies$  x = -12.
```

STREAMS FORMATTATI

In base al tipo di dato che scorre sullo stream viene cambiata la formattazione. Per esempio `char 'A'` e `int 65` in memoria sono rappresentati allo stesso modo, ma vengono convertiti diversamente sullo stream! (es. `format_A_65.cc`)

→ Esempio:

```
int x;  
cin >> x;  
  
cin = -12  $\implies$  x = -12.
```

→ Esempio:

```
char x, y;  
cin >> x >> y;  
  
cin = -12  $\implies$  x = '-', y = '1'.
```

STREAMS FORMATTATI

In base al tipo di dato che scorre sullo stream viene cambiata la formattazione. Per esempio `char 'A'` e `int 65` in memoria sono rappresentati allo stesso modo, ma vengono convertiti diversamente sullo stream! (es. `format_A_65.cc`)

→ Esempio:

```
int x;  
cin >> x;  
  
cin = -12  $\implies$  x = -12.
```

→ Esempio:

```
char x, y;  
cin >> x >> y;  
  
cin = -12  $\implies$  x = '-', y = '1'.
```

es. `format_succ.cc`.

SCRITTURA DI UNO STREAM

- << operatore di inserimento per operatori di scrittura (output).
- **Sintassi**: `stream_utilizzato << espressione_tipata`.
- **Semantica**: immette sullo `stream_utilizzato` la valutazione della `espressione_tipata`.

SCRITTURA DI UNO STREAM

- `<<` operatore di inserimento per operatori di scrittura (output).
- **Sintassi**: `stream_utilizzato << espressione_tipata`.
- **Semantica**: immette sullo `stream_utilizzato` la valutazione della `espressione_tipata`.

→ Esempio:

```
int x = 1;  
cout << x;
```

SCRITTURA DI UNO STREAM

- `<<` operatore di inserimento per operatori di scrittura (output).
- **Sintassi**: `stream_utilizzato << espressione_tipata`.
- **Semantica**: immette sullo `stream_utilizzato` la valutazione della `espressione_tipata`.

→ Esempio:

```
int x = 1;  
cout << x;
```

Esempio:

```
int x = 1;  
int y = 2;  
cout << x + y;
```

⇒ stream cout: 3.

SCRITTURA DI UNO STREAM

→ Esempio: cascata di scritture (associativo a sinistra)

```
int x = 1;  
int y = 2;  
cout << x << ' ' << y << endl << x + y << endl;
```

Stream cout:

1 2

3

SCRITTURA DI UNO STREAM

→ Esempio: cascata di scritture (associativo a sinistra)

```
int x = 1;
int y = 2;
cout << x << ' ' << y << endl << x + y << endl;
```

Stream cout:

1 2

3

→ `endl` -> **manipolatore**: è applicato all'oggetto stream nello stesso modo di un dato, ma anziché provocare lettura o scrittura dei dati, modifica lo stato interno dell'oggetto stream. Per esempio, `endl` inserisce un "a capo" nello stream: invece di scrivere

```
cout << '\n';
```

si scrive

```
cout << endl;
```

DIRAMAZIONI IF..ELSE

→ **if**: esecuzione condizionale di un'istruzione o blocco di istruzioni in base al valore di verità di un'istruzione logica specificata.

```
if (condizione)
    istruzione
```

```
if (condizione) {
    istruzione1;
    istruzione2;
}
```

DIRAMAZIONI IF..ELSE

- **if**: esecuzione condizionale di un'istruzione o blocco di istruzioni in base al valore di verità di un'istruzione logica specificata.

```
if (condizione)
    istruzione
```

```
if (condizione) {
    istruzione1;
    istruzione2;
}
```

- **if-else**: se condizione risulta true, istruzione1 è eseguita; altrimenti è eseguita istruzione2

```
if (condizione)
    istruzione1;
else
    istruzione2;
```

DIRAMAZIONI IF..ELSE

→ Scala if-else-if:

```
if (condizione1)
  istruzione1;
else if (condizione2)
  istruzione2;
else if (condizione3)
  istruzione3;
...
else
  istruzione;
```

Le espressioni condizionali vengono valutate dall'alto verso il basso.

es. programmi max2.cc, eta.cc.