Primi programmi in C++ - Parte I
Le variabili, l'input e l'output

Corso AMAT C++

Liceo Scientifico Statale "A. Volta"

A.S. 2018/19

Struttura di un programma in C++

```
1  #include <libreria1>
2  #include <libreria2>
3  #include <libreria3>
4  #include "liberaria_speciale1.h"
5  #include "liberaria_speciale2.h"
6  int main(){
7     istruzione1;
8     istruzione2;
9     istruzione3;
10    istruzione4;
11    istruzione5;
12    return 0;
13 }
```

Struttura di un programma in C++

- Le *direttive d'inclusione* servono per richiamare le *librerie*, che sono raccolte di strumenti a disposizione di chi programma. Per esempio:
  - la libreria `iostream` fornisce gli strumenti per input e output a terminale (cosicché un comando come `std::cout << "Ciao"` si può dare solo a patto di aver incluso la libreria `iostream`).
  - la libreria `cmath` definisce molte operazioni matematiche non elementari (per esempio, la radice quadrata, le funzioni trigonometriche, ecc.)

Struttura di un programma in C++

- Le *direttive d'inclusione* servono per richiamare le *librerie*, che sono raccolte di strumenti a disposizione di chi programma. Per esempio:
  - la libreria `iostream` fornisce gli strumenti per input e output a terminale (cosicché un comando come `std::cout << "Ciao"` si può dare solo a patto di aver incluso la libreria `iostream`).
  - la libreria `cmath` definisce molte operazioni matematiche non elementari (per esempio, la radice quadrata, le funzioni trigonometriche, ecc.)
- Le *istruzioni* sono i comandi che chiediamo al computer di eseguire in successione.

Le istruzioni

Va bene

```
1 int main(){
2    istruzione1; istruzione2;
3    istruzione3
4    che continua qui;
5    return 0;
6 }
```

Si possono scrivere più istruzioni su una stessa riga. Si può dividere un'istruzione lunga su più righe - e il ritorno a capo che spezza l'istruzione in due ha lo stesso valore di uno spazio bianco.

Non va bene

```
1 int main(){
2    istruzione1
3    istruzione2
4    istruzione3
5    return 0
6 }
```

Le istruzioni vanno obbligatoriamente separate l'una dall'altra mediante un punto e virgola.

Le istruzioni

Va bene

```
1 #include <iostream>
2 int main(){
3    int a; a = 5;
4    std::cout
5    << a;
6    return 0;
7 }
```

Si possono scrivere più istruzioni su una stessa riga. Si può dividere un'istruzione lunga su più righe - e il ritorno a capo che spezza l'istruzione in due ha lo stesso valore di uno spazio bianco.

Non va bene

```
1 #include <iostream>
2 int main(){
3    int a
4    a = 5
5    std::cout << a
6    return 0
7 }
```

Le istruzioni vanno obbligatoriamente separate l'una dall'altra mediante un punto e virgola.

Alcune istruzioni fondamentali

Abbiamo già incontrato quattro istruzioni fondamentali:

1. dichiarazione di variabili

```
int pinguino;
```

2. assegnazione

```
pinguino = 5;
```

3. scrittura su terminale (dalla libreria `iostream`)

```
std::cout << pinguino;
```

4. lettura da terminale (dalla libreria `iostream`)

```
std::cin >> pinguino;
```

Variabili

Una *variabile* è uno spazio di memoria che viene riservato allo scopo di conservarvi un'informazione. Una variabile consta di:

- un tipo (immutabile, assegnato una volta per tutte al momento della dichiarazione)
- un nome (immutabile, assegnato una volta per tutte al momento della dichiarazione)
- un valore, che può essere impostato (accesso in scrittura alla variabile) e letto (accesso in lettura della variabile) lungo l'esecuzione del programma

Possiamo pensare che una variabile sia una scatoletta (o un cassetto, una casella, un contenitore, un barattolo, . . .), provvista di un'etichetta: il tipo della variabile corrisponde alla forma e alla capienza della scatola; il nome della variabile è l'etichetta che vi abbiamo apposto sopra; il valore della variabile è il contenuto della scatola.

Tipo di una variabile

Il tipo di una variabile determina:

- la dimensione, cioè quanti bit di memoria vengono riservati per la variabile;
- l'insieme dei valori rappresentabili (cioè che cosa la variabile è in grado di ospitare);
- la codifica utilizzata per rappresentare tali valori.

Alcuni tipi C++

Tipo	**Dim.**	**Codifica**	**Val. rappresentabili**
unsigned short	16bit	Cifre bin.	Naturali da 0 a $2^{16}-1$
unsigned int	32bit	Cifre bin.	Naturali da 0 a $2^{32}-1$
unsigned long long	64bit	Cifre bin.	Naturali da 0 a $2^{64}-1$
short	16bit	Compl2	Interi da -2^{15} a $2^{15}-1$
int	32bit	Compl2	Interi da -2^{31} a $2^{31}-1$
long long	64bit	Compl2	Interi da -2^{63} a $2^{63}-1$
float	32bit	binary32	Reali (esponenti da -126 a $+127$, 23bit di mantissa)
double	64bit	binary64	Reali (esponenti da -1022 a $+1023$, 52bit di mantissa)
bool	1bit	Vero 1, Falso 0	Valore di verità

Nome di una variabile

Il nome di una variabile:

- è case-sensitive (maiuscole e minuscole non sono intercambiabili)
- deve consentire di individuare univocamente la variabile (non possono quindi in generale coesistere variabili con nomi diversi);
- non deve confliggere con una parola chiave riservata del C++ (non si può chiamare una variabile `return`, `if`, `while`, `int`, ecc.)
- deve soddisfare alcuni requisiti formali (può contenere solo lettere, numeri, e underscore, non può cominciare con un numero né contenere spazi)

Dichiarazione di una variabile

La *dichiarazione* è quell'istruzione mediante la quale una nuova variabile viene introdotta, e viene ad essa riservato lo spazio di memoria necessario (allocazione). Per dichiarare una variabile, basta scriverne il tipo e il nome che si desidera attribuirle (separati da uno spazio), come negli esempi qui sotto:

```
int numero_studenti;
long numero_abitanti_terra;
double temperatura_stanza;
bool giorno_festivo;
```

Assegnazione

L'*assegnazione* è quell'istruzione mediante la quale si imposta il valore di una variabile. La sintassi dell'assegnazione è la seguente:

variabile = espressione ;

Assegnazione

L'*assegnazione* è quell'istruzione mediante la quale si imposta il valore di una variabile. La sintassi dell'assegnazione è la seguente:

variabile = espressione ;

Immaginando che la variabile scritta a sinistra dell'uguale si chiami a, ecco alcuni esempi di assegnazione:

```
a = 5 ;          a = 2*(8*(2+3)+7)+20/3 ;
a = 2+3 ;        a = (b+5)*2 ;
```

Assegnazione

L'*assegnazione* è quell'istruzione mediante la quale si imposta il valore di una variabile. La sintassi dell'assegnazione è la seguente:

variabile = espressione ;

Immaginando che la variabile scritta a sinistra dell'uguale si chiami a, ecco alcuni esempi di assegnazione:

```
a = 5 ;          a = 2*(8*(2+3)+7)+20/3 ;
a = 2+3 ;        a = (b+5)*2 ;
```

Una siffatta assegnazione viene eseguita in due passi:

Assegnazione

L'*assegnazione* è quell'istruzione mediante la quale si imposta il valore di una variabile. La sintassi dell'assegnazione è la seguente:

variabile = espressione ;

Immaginando che la variabile scritta a sinistra dell'uguale si chiami a, ecco alcuni esempi di assegnazione:

```
a = 5 ;          a = 2*(8*(2+3)+7)+20/3 ;
a = 2+3 ;        a = (b+5)*2 ;
```

Una siffatta assegnazione viene eseguita in due passi:

1. il calcolatore *valuta* (cioè calcola) l' espressione che si trova a destra dell'uguale;

Assegnazione

L'*assegnazione* è quell'istruzione mediante la quale si imposta il valore di una variabile. La sintassi dell'assegnazione è la seguente:

variabile = espressione ;

Immaginando che la variabile scritta a sinistra dell'uguale si chiami a, ecco alcuni esempi di assegnazione:

```
a = 5 ;          a = 2*(8*(2+3)+7)+20/3 ;
a = 2+3 ;        a = (b+5)*2 ;
```

Una siffatta assegnazione viene eseguita in due passi:

1. il calcolatore *valuta* (cioè calcola) l' espressione che si trova a destra dell'uguale;
2. la quantità che risulta da tale valutazione viene *scritta* nello spazio di memoria riservato alla variabile a.

Osservazioni.

- Una variabile non può avere più d'un valore: questo significa che, ogniqualvolta si assegna un valore nuovo ad una variabile, quello vecchio viene sovrascritto.

Osservazioni.

- Una variabile non può avere più d'un valore: questo significa che, ogniqualvolta si assegna un valore nuovo ad una variabile, quello vecchio viene sovrascritto.
- Per scrivere l'espressione, si possono usare le parentesi tonde, come nelle espressioni matematiche, allo scopo di regolare la precedenza tra le operazioni.

Osservazioni.

- Una variabile non può avere più d'un valore: questo significa che, ogniqualvolta si assegna un valore nuovo ad una variabile, quello vecchio viene sovrascritto.
- Per scrivere l'espressione, si possono usare le parentesi tonde, come nelle espressioni matematiche, allo scopo di regolare la precedenza tra le operazioni.
- Quando in una espressione figura una variabile (per esempio, la `b` nell'ultimo degli esempi proposti), il suo valore verrà *letto* (ma non modificato!) al momento dell'assegnazione. Per esempio, se, `b` vale 3 e si esegue l'istruzione `a=(b+5)*2`, la valutazione dell'espressione restituisce $(3+5)\cdot 2 = 16$, e questo 16 verrà impostato come nuovo valore di `a`. Una volta eseguita l'istruzione, il valore di `a` sarà dunque diventato 16, mentre il valore di `b` sarà rimasto 3.

Operatori

Per comporre le espressioni, il C++ mette a nostra disposizione diversi *operatori*. Un operatore accetta in ingresso un fissato numero di operandi di un ben preciso tipo, e restituisce un risultato di un ben preciso tipo.

Operatori

Per comporre le espressioni, il C++ mette a nostra disposizione diversi *operatori*. Un operatore accetta in ingresso un fissato numero di operandi di un ben preciso tipo, e restituisce un risultato di un ben preciso tipo.
Per esempio, l'operatore "somma di numeri interi a 32 bit" accetta in ingresso due numeri interi di tipo int, e restituisce la loro somma sotto forma di numero intero intero di tipo int.

Operatori

Per comporre le espressioni, il C++ mette a nostra disposizione diversi *operatori*. Un operatore accetta in ingresso un fissato numero di operandi di un ben preciso tipo, e restituisce un risultato di un ben preciso tipo.
Per esempio, l'operatore "somma di numeri interi a 32 bit" accetta in ingresso due numeri interi di tipo int, e restituisce la loro somma sotto forma di numero intero intero di tipo int. Analogamente, non è difficile figurarsi un operatore "somma di numeri interi a 64bit", un operatore "somma di numeri reali di tipo double", ecc.

Operatori

Per comporre le espressioni, il C++ mette a nostra disposizione diversi *operatori*. Un operatore accetta in ingresso un fissato numero di operandi di un ben preciso tipo, e restituisce un risultato di un ben preciso tipo.
Per esempio, l'operatore "somma di numeri interi a 32 bit" accetta in ingresso due numeri interi di tipo int, e restituisce la loro somma sotto forma di numero intero intero di tipo int. Analogamente, non è difficile figurarsi un operatore "somma di numeri interi a 64bit", un operatore "somma di numeri reali di tipo double", ecc. Tutti questi operatori, benché distinti, si indicano con un unico simbolo: +, ed è il contesto ad indicare al compilatore quale degli operatori che tale simbolo indica sia da adoperarsi (si parla, a questo proposito, di *overloading*).

Operatori dell'aritmetica intera

Per l'aritmetica intera, il C++ mette a disposizione i seguenti operatori (nella tabella, TI indica un qualche tipo naturale o intero fissato):

Nome	**Simbolo**	**Operandi**	**Tipo risultato**
Somma	$+$	Due, di tipo TI	TI
Differenza	$-$	Due, di tipo TI	TI
Opposto	$-$	Uno, di tipo TI	TI
Prodotto	$*$	Due, di tipo TI	TI
Quoziente della divisione euclidea	$/$	Due, di tipo TI	TI
Resto della divisione euclidea	%	Due, di tipo TI	TI

Operatori dell'aritmetica reale

Per l'aritmetica reale, il C++ mette a disposizione i seguenti operatori (nella tabella, TR vuole indicare un fissato tipo reale, come float o double):

Nome	**Simbolo**	**Operandi**	**Tipo risultato**
Somma	$+$	Due, di tipo TR	TR
Differenza	$-$	Due, di tipo TR	TR
Opposto	$-$	Uno, di tipo TR	TR
Prodotto	$*$	Due, di tipo TR	TR
Quoziente	$/$	Due, di tipo TR	TR

Operatori dell'aritmetica booleana

Per l'aritmetica booleana, il C++ mette a disposizione i seguenti operatori:

Nome	**Simbolo**	**Operandi**	**Tipo risultato**
Congiunzione	`&&`, `and`	Due, di tipo `bool`	`bool`
Disgiunzione	`||`, `or`	Due, di tipo `bool`	`bool`
Negazione	`!`, `not`	Uno, di tipo `bool`	`bool`

Operatori di confronto

Gli operatori di confronto accettano in ingresso due quantità di uno stesso tipo T, le confrontano e restituiscono un risultato bool (vero o falso):

Nome	**Simbolo**	**Operandi**	**Tipo risultato**
Uguaglianza	==	Due, di tipo T	`bool`
Non uguaglianza	!=	Due, di tipo T	`bool`
Maggiore	>	Due, di tipo T	`bool`
Maggiore o uguale	>=	Due, di tipo T	`bool`
Minore	<	Due, di tipo T	`bool`
Minore o uguale	<=	Due, di tipo T	`bool`

Un esercizio di analisi di un codice...

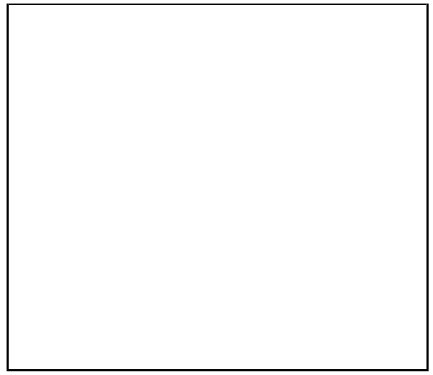
```
1  #include <iostream>
2  int main(){
3     int a;
4     bool b;
5
6     a = 5;
7     b = false;
8
9     a = (a/((a*2)%(a+3)));
10    b = ((!((a==5)||(a>8)))&&(a!=(a*2))&&(!b));
11
12    return 0;
13 }
```

Un esercizio di analisi di un codice...

Codice:

```
int a;
bool b;
a = 5;
b = false;
a = (a/((a*2)%(a+3)));
b = ((!((a==5)||(a>8)))&&...
```

Memoria:

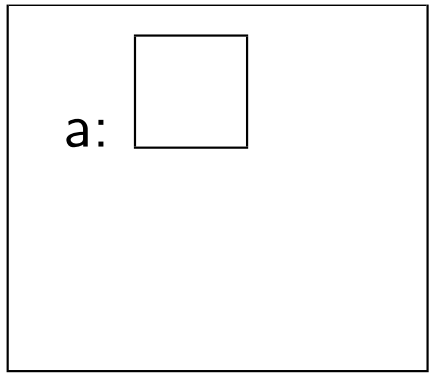


Un esercizio di analisi di un codice...

Codice:

```
int a;
---
bool b;
a = 5;
b = false;
a = (a/((a*2)%(a+3)));
b = ((!((a==5)||(a>8)))&&...
```

Memoria:

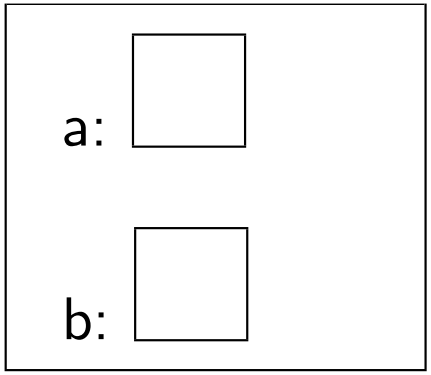


Un esercizio di analisi di un codice...

Codice:

```
int a;
bool b;
---
a = 5;
b = false;
a = (a/((a*2)%(a+3)));
b = ((!((a==5)||(a>8)))&&...
```

Memoria:



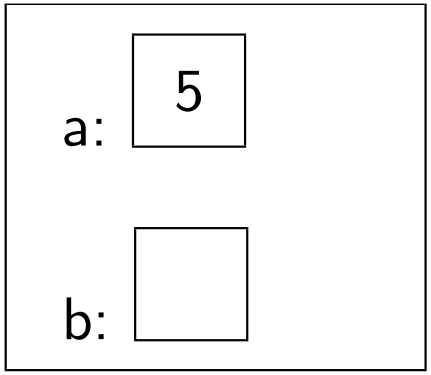
Un esercizio di analisi di un codice...

Codice:

```
int a;
bool b;
a = 5;
```

```
b = false;
a = (a/((a*2)%(a+3)));
b = ((!((a==5)||(a>8)))&&...
```

Memoria:



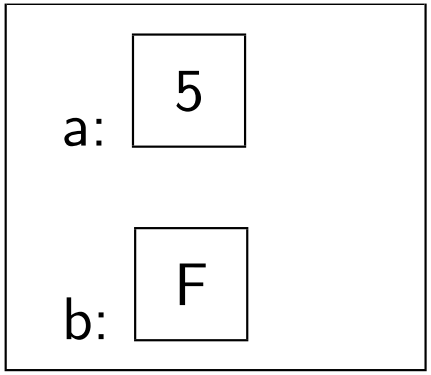
Un esercizio di analisi di un codice...

Codice:

```
int a;
bool b;
a = 5;
b = false;
```

```
a = (a/((a*2)%(a+3)));
b = ((!((a==5)||(a>8)))&&...
```

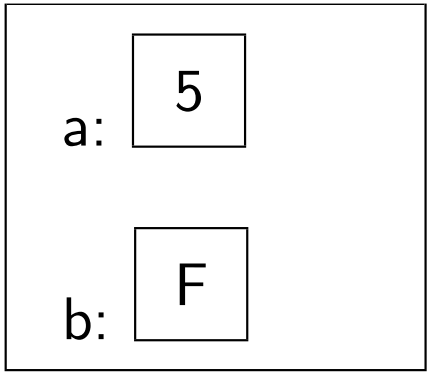
Memoria:



Un esercizio di analisi di un codice...

Calcoliamo: `(a/((a*2)%(a+3)));`

Memoria:

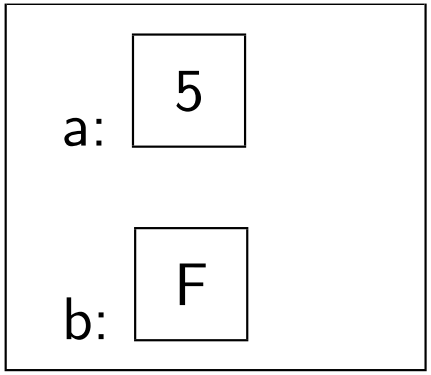


Un esercizio di analisi di un codice...

Calcoliamo: `(a/((a*2)%(a+3)));`

- Sostituisco ad a il suo valore:
  `(5/((5*2)%(5+3)))`

Memoria:

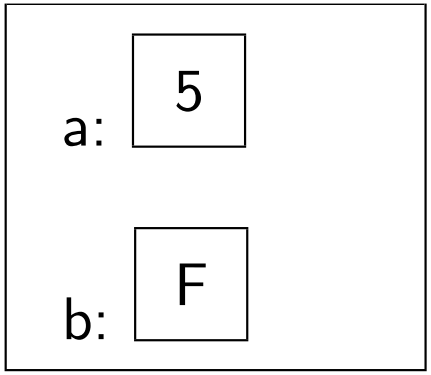


Un esercizio di analisi di un codice...

Calcoliamo: `(a/((a*2)%(a+3)));`

- Sostituisco ad a il suo valore: `(5/((5*2)%(5+3)))`
- Semplifico: `(5/(10%8))`

Memoria:

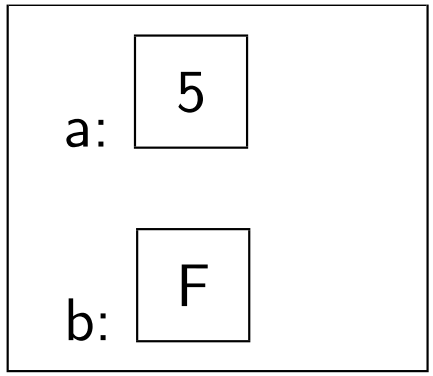


Un esercizio di analisi di un codice...

Calcoliamo: `(a/((a*2)%(a+3)));`

- Sostituisco ad a il suo valore: `(5/((5*2)%(5+3)))`
- Semplifico: `(5/(10%8))`
- Semplifico: `5/2`

Memoria:

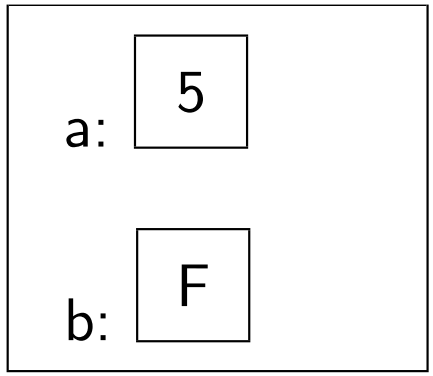


Un esercizio di analisi di un codice...

Calcoliamo: `(a/((a*2)%(a+3)));`

- Sostituisco ad a il suo valore: `(5/((5*2)%(5+3)))`
- Semplifico: `(5/(10%8))`
- Semplifico: `5/2`
- Il risultato è: 2

Memoria:

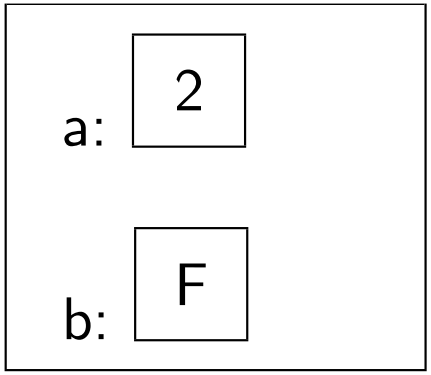


Un esercizio di analisi di un codice...

Codice:

```
int a;
bool b;
a = 5;
b = false;
a = (a/((a*2)%(a+3)));
---
b = (((!((a==5)||(a>8)))&&...
```

Memoria:

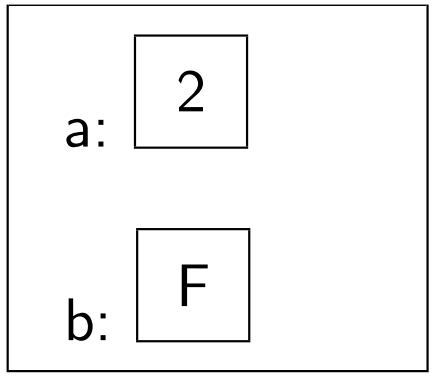


Un esercizio di analisi di un codice...

Calcoliamo:

```
(!((a==5)||(a>8)))&&(a!=(a*2))&&(!b)
```

Memoria:



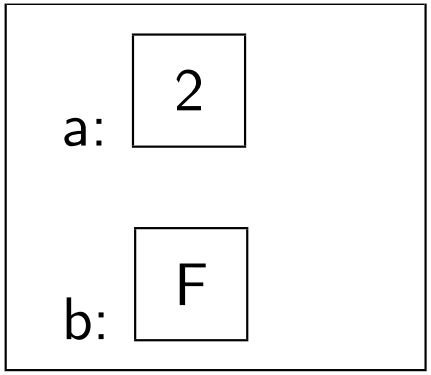
Un esercizio di analisi di un codice...

Calcoliamo:

`(!((a==5)||(a>8)))&&(a!=(a*2))&&(!b)`

- Sostituisco ad a e b i loro valori:
  `(!((2==5)||(2>8)))&&(2!=(2*2))&&(!F)`

Memoria:



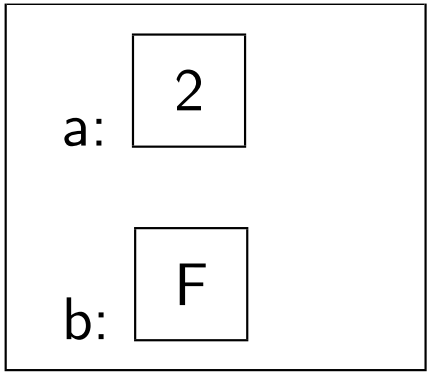
Un esercizio di analisi di un codice...

Calcoliamo:

```
(!((a==5)||(a>8)))&&(a!=(a*2))&&(!b)
```

- Sostituisco ad a e b i loro valori:
  ```
  (!((2==5)||(2>8)))&&(2!=(2*2))&&(!F)
  ```
- Semplifico:
  ```
  (!(F||F)) && (2!=4) && (!F)
  ```

Memoria:



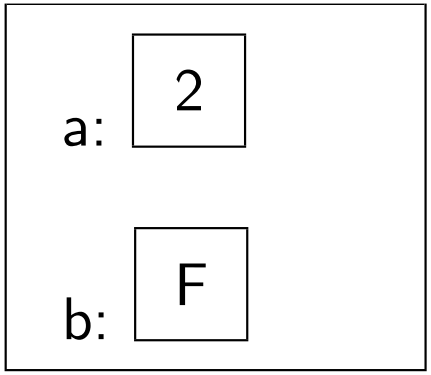
Un esercizio di analisi di un codice...

Calcoliamo:

`(!((a==5)||(a>8)))&&(a!=(a*2))&&(!b)`

- Sostituisco ad a e b i loro valori:
  `(!((2==5)||(2>8)))&&(2!=(2*2))&&(!F)`
- Semplifico:
  `(!(F||F)) && (2!=4) && (!F)`
- Semplifico: `(!F) && V && V`

Memoria:



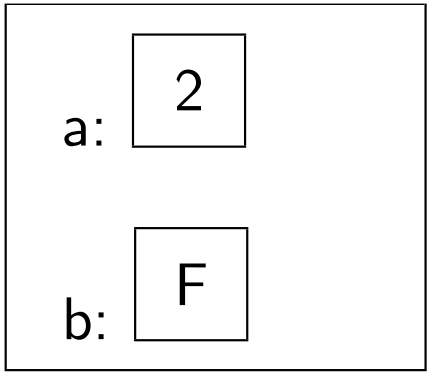
Un esercizio di analisi di un codice...

Calcoliamo:

`(!((a==5)||(a>8)))&&(a!=(a*2))&&(!b)`

- Sostituisco ad a e b i loro valori:
  `(!((2==5)||(2>8)))&&(2!=(2*2))&&(!F)`
- Semplifico:
  `(!(F||F)) && (2!=4) && (!F)`
- Semplifico: `(!F) && V && V`
- Semplifico: `V && V && V`

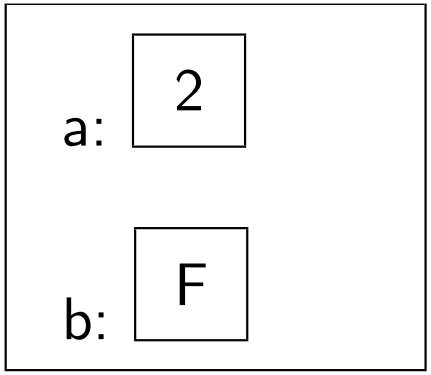
Memoria:



Un esercizio di analisi di un codice...

Calcoliamo:
`(!((a==5)||(a>8)))&&(a!=(a*2))&&(!b)`

Memoria:



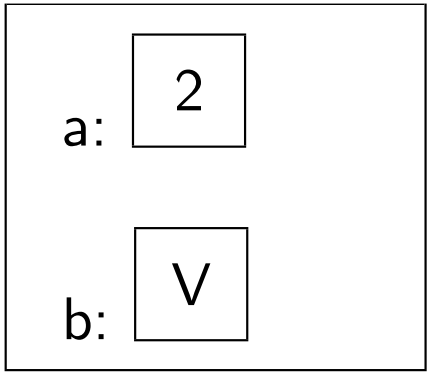
- Sostituisco ad a e b i loro valori:
  `(!((2==5)||(2>8)))&&(2!=(2*2))&&(!F)`
- Semplifico:
  `(!(F||F)) && (2!=4) && (!F)`
- Semplifico: `(!F) && V && V`
- Semplifico: `V && V && V`
- Il risultato è: `V`

Un esercizio di analisi di un codice...

Codice:

```
int a;
bool b;
a = 5;
b = false;
a = (a/((a*2)%(a+3)));
b = ((!((a==5)||(a>8)))&&...
```

Memoria:



Lettura da terminale

Se a è il nome di una variabile, l'istruzione

```
std::cin >> a ;
```

viene eseguita in questo modo:

1. si attende che l'utente immetta, sul terminale, un valore, e che prema invio;
2. il valore immesso da tastiera viene *scritto* nello spazio di memoria riservato alla variabile a.

Si tratta quindi di un'istruzione analogo all'assegnazione (di cui abbiamo appena discusso), con la sola differenza che il valore assegnato ad a non proviene dalla valutazione di un'espressione, ma viene fornito da un *input* dell'utente.

Scrittura su terminale di stringhe

Per stampare letteralmente un carattere, una parola, una frase sul terminale, occorre utilizzare un comando del tipo:

```
std::cout << "Ciao ciao";
```

Scrittura su terminale di stringhe

Per stampare letteralmente un carattere, una parola, una frase sul terminale, occorre utilizzare un comando del tipo:

```
std::cout << "Ciao ciao";
```

Per scrivere un "a capo", bisognerà usare il carattere speciale \n:

```
std::cout << "Ciao\nciao";
```

Scrittura su terminale di stringhe

Per stampare letteralmente un carattere, una parola, una frase sul terminale, occorre utilizzare un comando del tipo:

```
std::cout << "Ciao ciao";
```

Per scrivere un "a capo", bisognerà usare il carattere speciale \n:

```
std::cout << "Ciao\nciao";
```

Per stampare alcuni caratteri particolari (come le doppie virgolette o il backslash \) bisogna farli precedere da un backslash \:

```
std::cout << "Qui stampo una virgoletta:\". Ecco fatto.";
std::cout << "Qui stampo un backslash:\\. Ecco fatto.";
```

Scrittura su terminale

L'istruzione

```
std::cout <<  espressione  ;
```

viene eseguita in questo modo:

1. il calcolatore *valuta* l' espressione che si trova a destra dell'uguale;
2. la quantità così ottenuta viene *stampata su terminale*.

Esempi:

```
std::cout <<  5 ;
std::cout <<  2+3 ;
std::cout <<  2*(8*(2+3)+7)+20/3 ;
std::cout <<  (b+5)*2 ;
```

Circa le espressioni e la loro valutazione, valgono tutte le considerazioni che già in precedenza esposte: un'espressione può coinvolgere costanti numeriche, parentesi tonde, variabili, operatori, ecc.; quando viene valutata, le eventuali variabili presenti vengono *lette*, senza che il loro valore venga però alterato.

Primi programmi in C++ - Parte II
Analizzare un codice

Corso AMAT C++

Liceo Scientifico Statale "A. Volta"

A.S. 2018/19

Le quattro istruzioni finora viste

Abbiamo finora discusso di quattro sole istruzioni:

- dichiarazione di variabili

```
tipo variabile ;
```

introduce una variabile, allocando lo spazio di memoria necessario.

- assegnazione

```
variabile = espressione ;
```

calcola l'espressione scritta a destra dell'uguale, e imposta il risultato ottenuto come nuovo valore della variabile indicata a sinistra dell'uguale.

- scrittura su terminale

```
std::cout << espressione ;
```

calcola l'espressione, e scrive il risultato ottenuto sul terminale.

- lettura da terminale

```
std::cin >> variabile ;
```

attende che l'utente digiti un valore sul terminale, e poi lo imposta come nuovo valore della variabile indicata.

Uso in lettura e in scrittura delle variabili

Una variabile:

- viene **dichiarata** mediante una dichiarazione della forma `tipo variabile ;`
- viene utilizzata **in lettura** ogniqualvolta compare all'interno di un' espressione : in tal caso, il suo valore viene *letto* dal computer al fine di calcolare l'espressione, ma non viene modificato.
- viene utilizzata **in scrittura** ogniqualvolta è bersaglio di un'*assegnazione* (cioè compare a sinistra dell'uguale in un'istruzione del tipo `variabile = espressione;`) o di un input da tastiera (cioè compare in un'istruzione del tipo `std::cin >> variabile ;`): in entrambi i casi, la variabile assume un *nuovo valore*, e quello vecchio va perduto.

Esercizio 02

L'esercizio chiedeva di analizzare i seguenti programmi:

```cpp
#include <iostream>
int main(){
   int a;
   a = 5;

   std::cout << (a+1);
   std::cout << "\n";

   std::cout << (a+1);
   std::cout << "\n";

   return 0;
}
```

```cpp
#include <iostream>
int main(){
   int a;
   a = 5;

   a = a+1;
   std::cout << a;
   std::cout << "\n";

   a = a+1;
   std::cout << a;
   std::cout << "\n";

   return 0;
}
```

Esercizio 02 - primo programma

Codice:

```
int a;
a = 5;
std::cout << (a+1);
std::cout << "\n";
std::cout << (a+1);
std::cout << "\n";
```

Memoria:

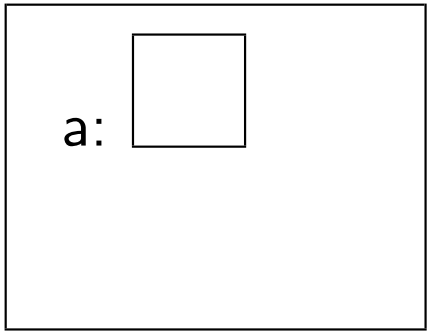
Terminale:

Esercizio 02 - primo programma

Codice:

```
int a;
---
a = 5;
std::cout << (a+1);
std::cout << "\n";
std::cout << (a+1);
std::cout << "\n";
```

Memoria:



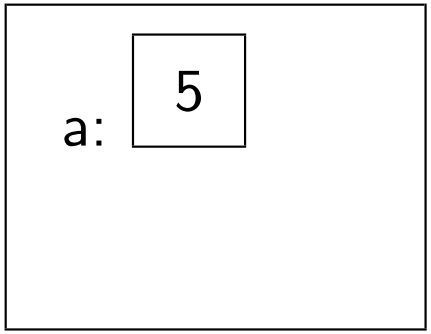
Terminale:

Esercizio 02 - primo programma

Codice:

```
int a;
a = 5;
---
std::cout << (a+1);
std::cout << "\n";
std::cout << (a+1);
std::cout << "\n";
```

Memoria:



Terminale:

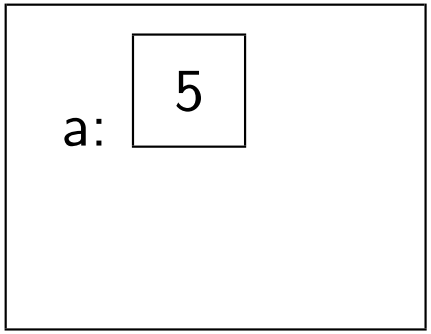
Esercizio 02 - primo programma

Codice:

```
int a;
a = 5;
std::cout << (a+1);
```

```
std::cout << "\n";
std::cout << (a+1);
std::cout << "\n";
```

Memoria:



Terminale:

```
6
```

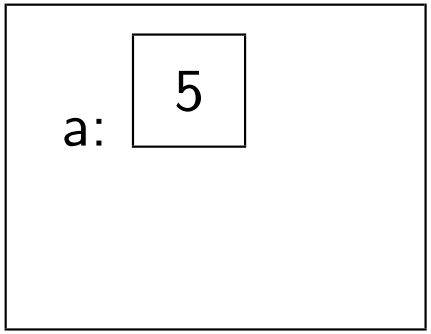
Esercizio 02 - primo programma

Codice:

```
int a;
a = 5;
std::cout << (a+1);
std::cout << "\n";
```

```
std::cout << (a+1);
std::cout << "\n";
```

Memoria:



Terminale:

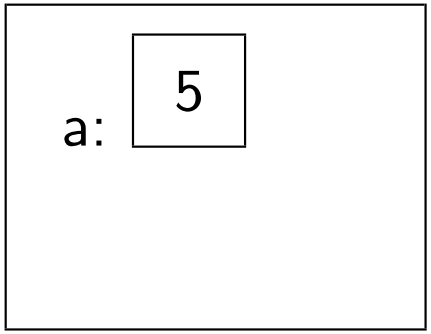
```
6
```

Esercizio 02 - primo programma

Codice:

```
int a;
a = 5;
std::cout << (a+1);
std::cout << "\n";
std::cout << (a+1);
std::cout << "\n";
```

Memoria:



Terminale:

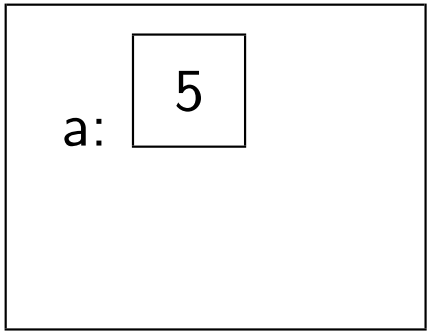
```
6
6
```

Esercizio 02 - primo programma

Codice:

```
int a;
a = 5;
std::cout << (a+1);
std::cout << "\n";
std::cout << (a+1);
std::cout << "\n";
```

Memoria:



Terminale:

```
6
6
```

Esercizio 02 - primo programma

Istr.	Istruzione	Variabili in memoria e loro valore (una volta eseguita l'istruzione)	Output prodotto dall'istruzione
#1	`int a;` Dichiara la variabile a	a:	Nessuno
#2	`a = 5;` Calcola il valore dell'espressione a destra (ottenendo 5), e lo imposta come nuovo valore della variabile a	a: 5	Nessuno
#4	`std::cout << (a+1);` Calcola il valore dell'espressione a destra (ottenendo 6), e lo scrive sul terminale	a: 5	6
#5	`std::cout << "\n ";` Scrive il carattere "a capo" sul terminale	a: 5	A capo
#6	`std::cout << (a+1);` Calcola il valore dell'espressione a destra (ottenendo 6), e lo scrive sul terminale	a: 5	6
#7	`std::cout << "\n ";` Scrive il carattere "a capo" sul terminale	a: 5	A capo

Esercizio 02 - secondo programma

Codice:

```
int a;
a = 5;
a = a+1;
std::cout << a;
std::cout << "\n";
a = a+1;
std::cout << a;
std::cout << "\n";
```

Memoria:

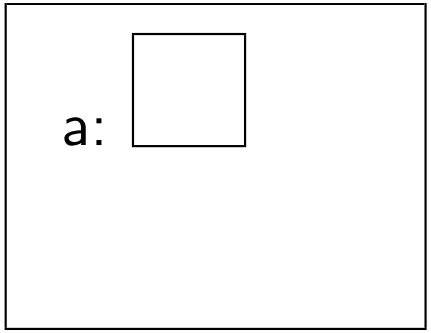
Terminale:

Esercizio 02 - secondo programma

Codice:

```
int a;
a = 5;
a = a+1;
std::cout << a;
std::cout << "\n";
a = a+1;
std::cout << a;
std::cout << "\n";
```

Memoria:



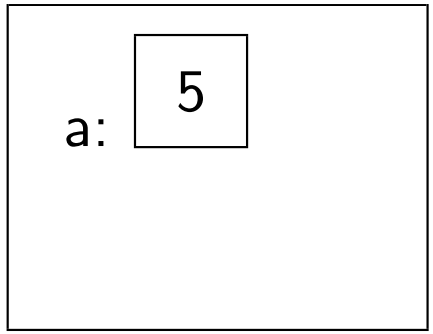
Terminale:

Esercizio 02 - secondo programma

Codice:

```
int a;
a = 5;
---
a = a+1;
std::cout << a;
std::cout << "\n";
a = a+1;
std::cout << a;
std::cout << "\n";
```

Memoria:



Terminale:

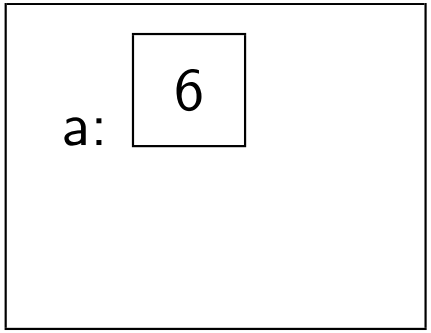
Esercizio 02 - secondo programma

Codice:

```
int a;
a = 5;
a = a+1;
```

```
std::cout << a;
std::cout << "\n";
a = a+1;
std::cout << a;
std::cout << "\n";
```

Memoria:



Terminale:

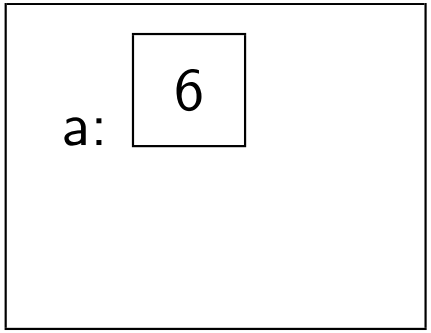
Esercizio 02 - secondo programma

Codice:

```
int a;
a = 5;
a = a+1;
std::cout << a;
```

```
std::cout << "\n";
a = a+1;
std::cout << a;
std::cout << "\n";
```

Memoria:



Terminale:

```
6
```

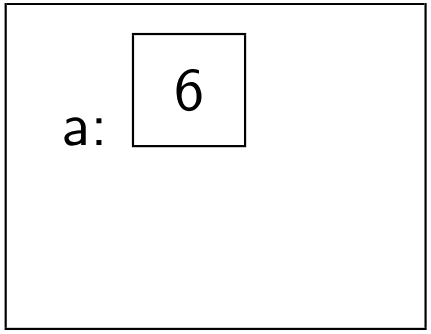
Esercizio 02 - secondo programma

Codice:

```
int a;
a = 5;
a = a+1;
std::cout << a;
std::cout << "\n";
```

```
a = a+1;
std::cout << a;
std::cout << "\n";
```

Memoria:



Terminale:

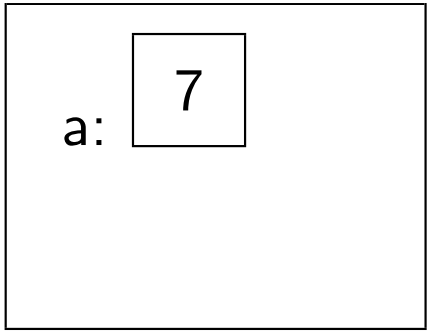
```
6
```

Esercizio 02 - secondo programma

Codice:

```
int a;
a = 5;
a = a+1;
std::cout << a;
std::cout << "\n";
a = a+1;
---
std::cout << a;
std::cout << "\n";
```

Memoria:



Terminale:

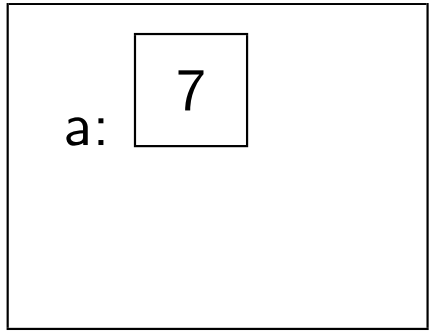
```
6
```

Esercizio 02 - secondo programma

Codice:

```
int a;
a = 5;
a = a+1;
std::cout << a;
std::cout << "\n";
a = a+1;
std::cout << a;
std::cout << "\n";
```

Memoria:



Terminale:

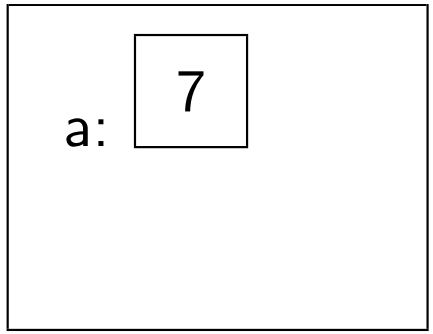
```
6
7
```

Esercizio 02 - secondo programma

Codice:

```
int a;
a = 5;
a = a+1;
std::cout << a;
std::cout << "\n";
a = a+1;
std::cout << a;
std::cout << "\n";
```

Memoria:



Terminale:

```
6
7
```

Esercizio 02 - secondo programma

Istr.	Istruzione	Variabili in memoria e loro valore (una volta eseguita l'istruzione)	Output prodotto dall'istruzione
#1	`int a;` Dichiara la variabile a	a:	Nessuno
#2	`a = 5;` Calcola il valore dell'espressione a destra (ottenendo 5), e lo imposta come nuovo valore della variabile a	a: 5	Nessuno
#3	`a = a+1;` Calcola il valore dell'espressione a destra (ottenendo 6), e lo imposta come nuovo valore della variabile a	a: 6	Nessuno
#4	`std::cout << a;` Calcola il valore dell'espressione a destra (ottenendo 6), e lo scrive sul terminale	a: 6	6
#5	`std::cout << "\n ";` Scrive il carattere "a capo" sul terminale	a: 6	A capo
#6	`a = a+1;` Calcola il valore dell'espressione a destra (ottenendo 7), e lo imposta come nuovo valore della variabile a	a: 7	Nessuno
#7	`std::cout << a;` Calcola il valore dell'espressione a destra (ottenendo 7), e lo scrive sul terminale	a: 7	7
#8	`std::cout << "\n ";` Scrive il carattere "a capo" sul terminale	a: 7	A capo

Esercizio 03

Analizzare il seguente programma (immaginando l'utente immetta da terminale, nell'ordine, i numeri 10 e 20):

```
1  #include <iostream>
2  int main(){
3     int a;
4     int b;
5
6     std::cout << "Inserisci il numero a: ";
7     std::cin >> a;
8
9     std::cout << "Inserisci il numero b: ";
10    std::cin >> b;
11
12    a = b;
13    b = a;
14
15    return 0;
16 }
```

Esercizio 03

Codice:

```
int a;
int b;
std::cout << "Inserisci a: ";
std::cin >> a;
std::cout << "Inserisci b: ";
std::cin >> b;
a = b;
b = a;
```

Memoria:

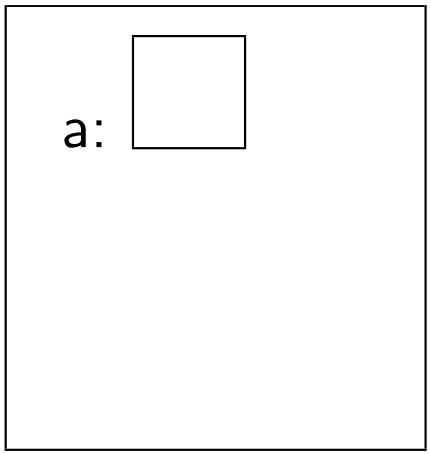
Terminale:

Esercizio 03

Codice:

```
int a;
int b;
std::cout << "Inserisci a: ";
std::cin >> a;
std::cout << "Inserisci b: ";
std::cin >> b;
a = b;
b = a;
```

Memoria:



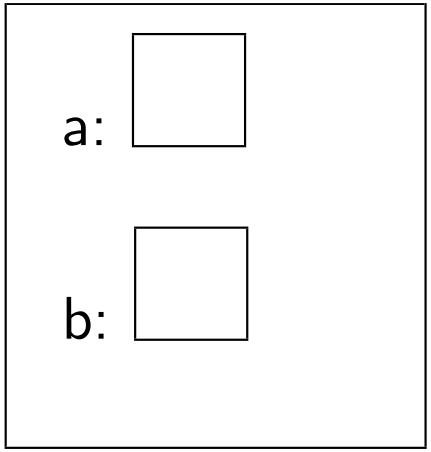
Terminale:

Esercizio 03

Codice:

```
int a;
int b;
---
std::cout << "Inserisci a: ";
std::cin >> a;
std::cout << "Inserisci b: ";
std::cin >> b;
a = b;
b = a;
```

Memoria:



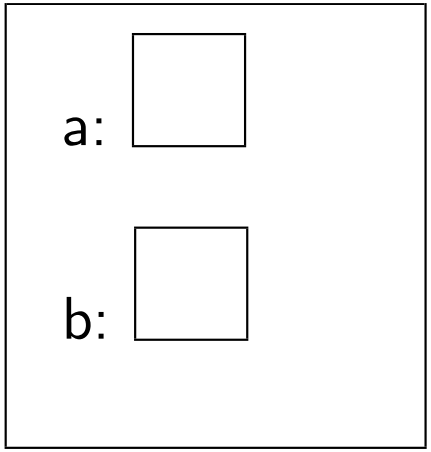
Terminale:

Esercizio 03

Codice:

```
int a;
int b;
std::cout << "Inserisci a: ";
std::cin >> a;
std::cout << "Inserisci b: ";
std::cin >> b;
a = b;
b = a;
```

Memoria:



Terminale:

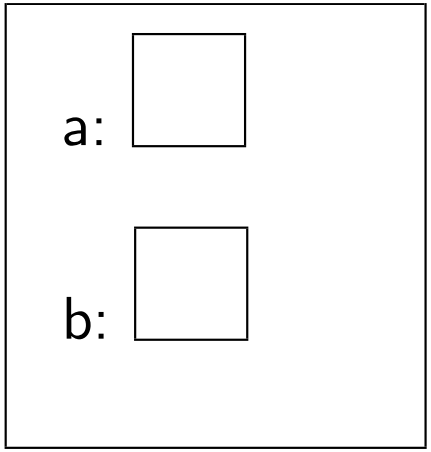
```
Inserisci a:
```

Esercizio 03

Codice:

```
int a;
int b;
std::cout << "Inserisci a: ";
std::cin >> a;
std::cout << "Inserisci b: ";
std::cin >> b;
a = b;
b = a;
```

Memoria:



Terminale:

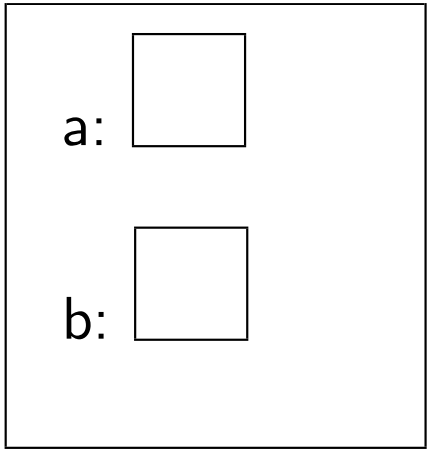
```
Inserisci a:
```

Esercizio 03

Codice:

```
int a;
int b;
std::cout << "Inserisci a: ";
std::cin >> a;
std::cout << "Inserisci b: ";
std::cin >> b;
a = b;
b = a;
```

Memoria:



Terminale:

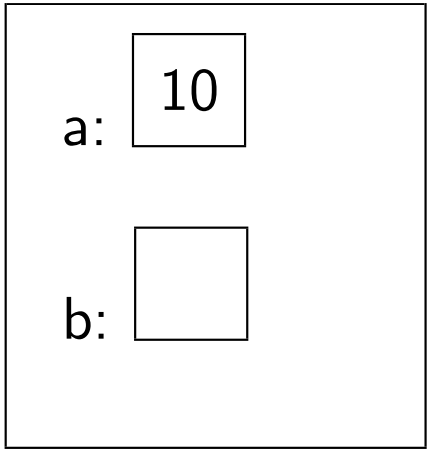
```
Inserisci a: 10
```

Esercizio 03

Codice:

```
int a;
int b;
std::cout << "Inserisci a: ";
std::cin >> a;
---
std::cout << "Inserisci b: ";
std::cin >> b;
a = b;
b = a;
```

Memoria:



Terminale:

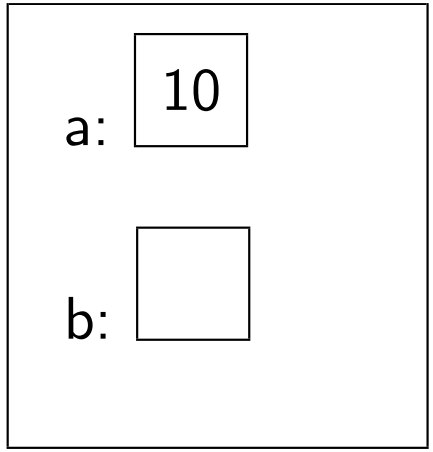
```
Inserisci a: 10
```

Esercizio 03

Codice:

```
int a;
int b;
std::cout << "Inserisci a: ";
std::cin >> a;
std::cout << "Inserisci b: ";
std::cin >> b;
a = b;
b = a;
```

Memoria:



Terminale:

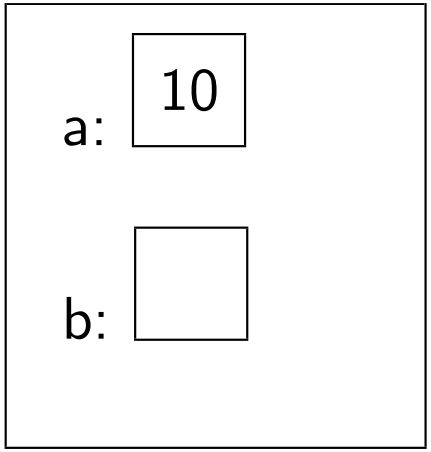
```
Inserisci a: 10
Inserisci b:
```

Esercizio 03

Codice:

```
int a;
int b;
std::cout << "Inserisci a: ";
std::cin >> a;
std::cout << "Inserisci b: ";
std::cin >> b;
a = b;
b = a;
```

Memoria:



Terminale:

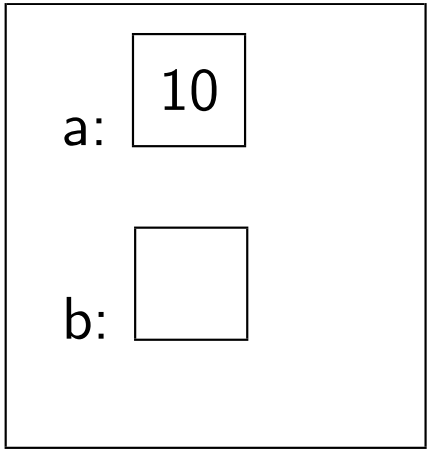
```
Inserisci a: 10
Inserisci b:
```

Esercizio 03

Codice:

```
int a;
int b;
std::cout << "Inserisci a: ";
std::cin >> a;
std::cout << "Inserisci b: ";
std::cin >> b;
a = b;
b = a;
```

Memoria:



Terminale:

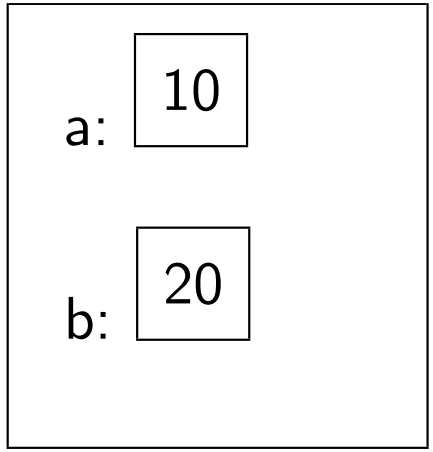
```
Inserisci a: 10
Inserisci b: 20
```

Esercizio 03

Codice:

```
int a;
int b;
std::cout << "Inserisci a: ";
std::cin >> a;
std::cout << "Inserisci b: ";
std::cin >> b;
---
a = b;
b = a;
```

Memoria:



Terminale:

```
Inserisci a: 10
Inserisci b: 20
```

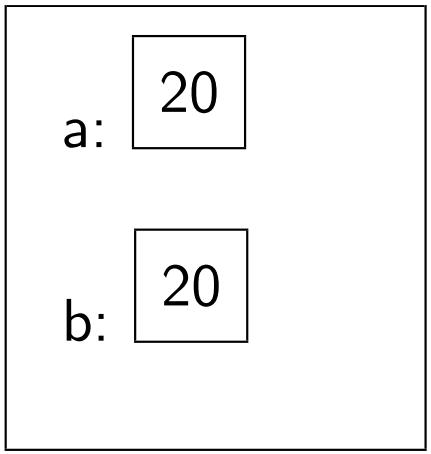
Esercizio 03

Codice:

```
int a;
int b;
std::cout << "Inserisci a: ";
std::cin >> a;
std::cout << "Inserisci b: ";
std::cin >> b;
a = b;
```

```
b = a;
```

Memoria:



Terminale:

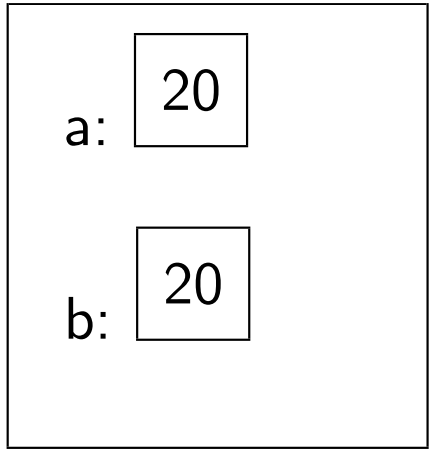
```
Inserisci a: 10
Inserisci b: 20
```

Esercizio 03

Codice:

```
int a;
int b;
std::cout << "Inserisci a: ";
std::cin >> a;
std::cout << "Inserisci b: ";
std::cin >> b;
a = b;
b = a;
```

Memoria:



Terminale:

```
Inserisci a: 10
Inserisci b: 20
```

Esercizio 03

Istr.	Istruzione	Variabili in memoria e loro valore (una volta eseguita l'istruzione)	Output prodotto dall'istruzione
#1	`int a;` Dichiara la variabile a	a:	Nessuno
#2	`int b;` Dichiara la variabile b	a: b:	Nessuno
#3	`std::cout << "Inserisci a:";` Scrive la stringa indicata sul terminale	a: b:	Inserisci a:
#4	`std::cin >> a;` Aspetta che l'utente immetta un numero, e lo imposta come nuovo valore di a	a: 10 b:	Nessuno
#5	`std::cout << "Inserisci b:";` Scrive la stringa indicata sul terminale	a: 10 b:	Inserisci b:
#6	`std::cin >> b;` Aspetta che l'utente immetta un numero, e lo imposta come nuovo valore di b	a: 10 b: 20	Nessuno
#7	`a = b;` Calcola il valore dell'espressione a destra (ottenendo 20), e lo imposta come nuovo valore della variabile a	a: 20 b: 20	Nessuno
#8	`b = a;` Calcola il valore dell'espressione a destra (ottenendo 20), e lo imposta come nuovo valore della variabile b	a: 20 b: 20	Nessuno

Esercizio 03

Per aver conferma che il programma venga effettivamente eseguito proprio nel modo appena descritto, possiamo "costringere" il computer, con opportuni comandi di output (`std::cout<<`), a comunicarci il valore delle due variabili `a` e `b` nei momenti salienti dell'esecuzione del programma. Per esempio, se inserissimo queste righe di codice:

```
1 std::cout << "In questo momento, la variabile a vale ";
2 std::cout << a;
3 std::cout << " mentre b vale ";
4 std::cout << b;
5 std::cout << "\n";
```

nei tre punti seguenti:

Esercizio 03

```
1  #include <iostream>
2  int main(){
3     int a;
4     int b;
5
6     std::cout << "Inserisci il numero a: ";
7     std::cin >> a;
8
9     std::cout << "Inserisci il numero b: ";
10    std::cin >> b;
11          //QUI!!!
12    a = b;
13          //QUI!!!
14    b = a;
15          //QUI!!!
16    return 0;
17 }
```

Esercizio 03

sul terminale, al termine dell'esecuzione del programma, vedremmo scritto:

```
Inserisci il numero a: 10
Inserisci il numero b: 20
In questo momento, la variabile a vale 10 mentre b vale 20
In questo momento, la variabile a vale 20 mentre b vale 20
In questo momento, la variabile a vale 20 mentre b vale 20
```