Informatica - Mod. Programmazione
Lezione 02

Prof. Giuseppe Psaila

Laurea Triennale in Ingegneria Informatica
Università di Bergamo

Programma Base

Programma: Salve_Mondo.cpp

```
#include <iostream>

using namespace std;

int main()
{
    cout << "SALVE MONDO";
}
```

Programma Base

- `#include <iostream>`
  Include il file di intestazione delle librerie che definisce i simboli e gli operatori per input/output
- `using namespace std;`
  Indica di usare il namespace (spazio dei nomi) `std` quando non viene specificato. Tutti i simboli e gli operatori della libreria standard del C++ fanno parte del namespace `std`

Programma Base

- `int main()`
  è il programma principale (`main`), cioè il programma che viene eseguito all'avvio dell'eseguibile
- `cout << "SALVE MONDO";`
  Manda il testo `SALVE MONDO` al dispositivo di uscita standard `cout` tramite l'operatore di output `<<` (flusso da destra verso sinistra)
- Il programma principale è racchiuso tra parentesi graffe (come ogni altro blocco di programma)
- Le istruzioni operative (che fanno qualche cosa) sono terminate da un `;`

Programma Base

Programma: Salve_Mondo_02.cpp

```
#include <iostream>

using namespace std;

int main()
{
    cout << "SALVE MONDO";
    cout << "Io sono un Programma";
}
```

Programma Base

- Anche se sono due istruzioni diverse
- Il testo è stampato su un'unica riga
- Perché occorre dire esplicitamente di andare a capo

Programma Base

Programma: Salve_Mondo_03.cpp

```
#include <iostream>

using namespace std;

int main()
{
    cout << "SALVE MONDO" << endl;
    cout << "Io sono un Programma";
}
```

Programma Base

- Il simbolo `endl`
  sta per **end line**
- Ora il secondo messaggio viene stampato sulla seconda riga

Programma: Salve_Mondo_04.cpp

```
#include <iostream>

using namespace std;

int main()
{
    cout << "SALVE MONDO" << endl;
    cout << "Io sono un Programma";

    return 0;
}
```

Programma Base

- L'istruzione `return 0;`
  serve per dire al sistema operativo che l'esecuzione è andata a buon fine
- Convenzione:
  0 = tutto OK
  altri valori: Errore

Variabili e Input

- Per poter interagire con il mondo esterno, un programma deve anche **acquisire** dati dall'esterno
  In altre parole, deve fare un'operazione di **Input**
- Ma i valori letti devono essere **Memorizzati** nella memoria centrale, per non essere persi
- Il programmatore gestisce la memoria centrale tramite il concetto di **Variabile**

Variabili

- Una **Variabile** è uno **spazio di memoria centrale** con associato un **Nome**
- `int a;`
  Definisce (o **Dichiara**) la variabile di nome a
  `int` è il **TIPO DI DATO**: indica quanti Byte servono e come devono essere usati per gestire a
  La variabile a memorizza numeri **Interi con Segno**

Input

- `cin >> a;`
  `cin` indica il **Dispositivo di Ingresso Standard** (tastiera)
  >> è l'**Operatore di Input**
  Il flusso da sinistra verso destra indica che il valore viene letto dal dispositivo di ingresso e finisce nella variabile alla destra
- Siccome la variabile a memorizza numeri interi, l'operatore >> legge i caratteri dal dispositivo di ingresso e li interpreta come numero intero, convertendolo in binario

Programma: Input_01.cpp

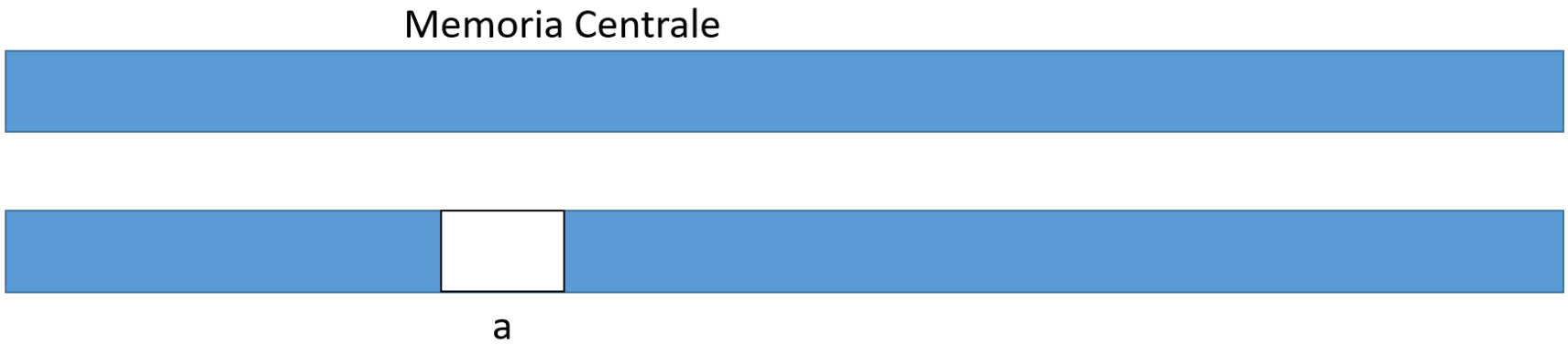
```
#include <iostream>

using namespace std;

int main()
{
    int a;

    cin >> a;
    cout << "Valore: " << a << endl;

    return 0;
}
```

Traccia dell'Esecuzione in Memoria

Prima di `int a;`

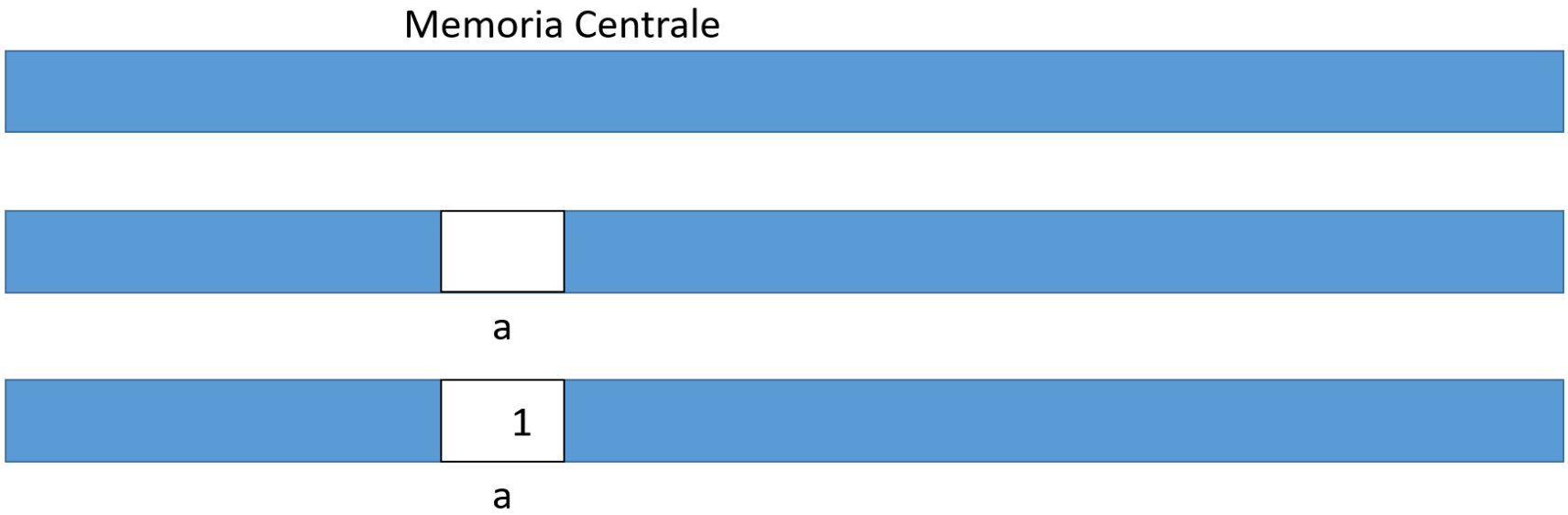
Traccia dell'Esecuzione in Memoria

Dopo `int a;`



Traccia dell'Esecuzione in Memoria

Dopo l'input



Assegnamento

- Come far calcolare il valore di una variabile al programma?
- Con l'operatore di **ASSEGNAMENTO** =
- `b = a;`
- **Valuta il valore dell'espressione alla sua destra e lo assegna alla variabile alla sua sinistra**
- Altro esempio:

  `b = a * 2;`

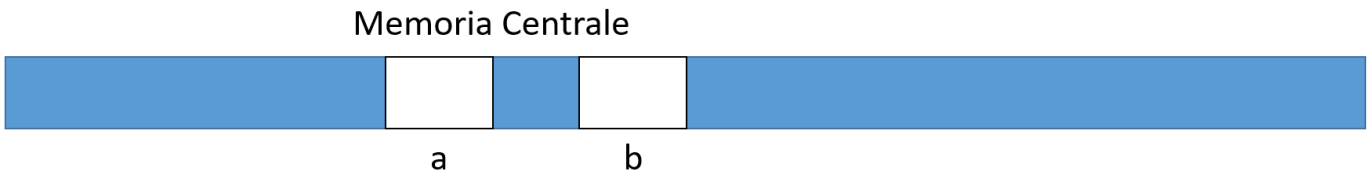
  Moltiplica il valore della variabile a per due e assegna il risultto alla variabile `b`
- **ATTENZIONE AL VERSO: DA DESTRA VERSO SINISTRA**

Programma: Input_02.cpp

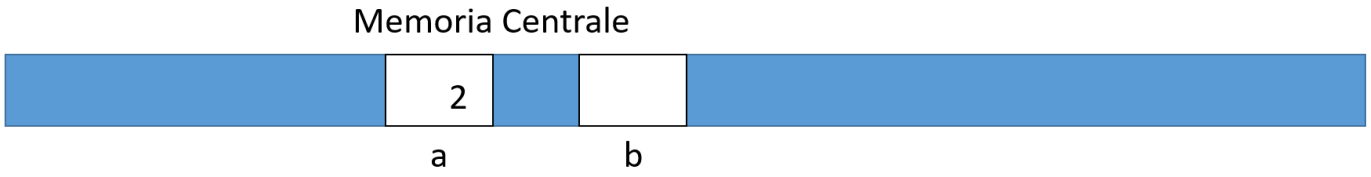
```
#include <iostream>

using namespace std;

int main()
{
    int a;
    int b;

    cin >> a;
    b = a * 2;
    cout << "Doppio: " << b << endl;

    return 0;
}
```

Traccia dell'Esecuzione in Memoria

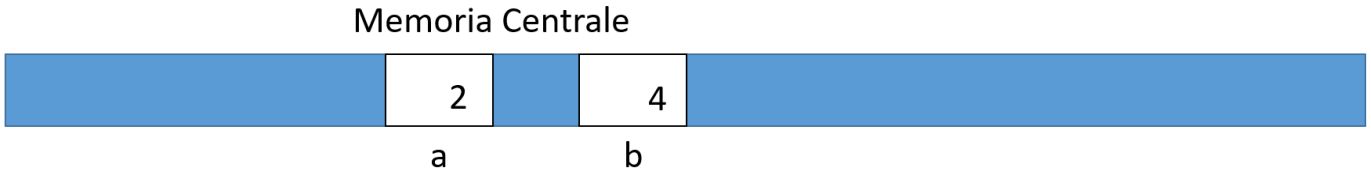
Dopo la definizione delle variabili



Dopo l'Input



Dopo l'assegnamento



Operatori Aritmetici

- +

  \-

  Somma e sottrazione
- *

  /

  Prodotto e Divisione
- %

  Resto della divisione (tra interi)
- (e)

  Le parentesi tonde forzano le sotto-espressioni (modificando la precedenza)

Assegnamenti Incrementali

- a +=*espressione*;
  Equivale ad a = a + *espressione*;
  Incrementa la variabile a con il valore dell'espressione alla sua destra
- a -=*espressione*;
  Equivale ad a = a - *espressione*;
  Decrementa la variabile a con il valore dell'espressione alla sua destra

Assegnamenti Incrementali

- `a *=`*espressione*`;`
  Equivale ad `a = a *` *espressione*`;`
  Moltiplica la variabile `a` per il valore dell'espressione alla sua destra
- `a /=`*espressione*`;`
  Equivale ad `a = a /` *espressione*`;`
  Divide la variabile `a` per il valore dell'espressione alla sua destra

Assegnamenti Incrementali

- `++`

  Incrementa la variabile di `1`
- `--`

  Decrementa la variabile di `1`

Si possono usare prima e dopo la variabile da incrementare/decrementare

- `a++`

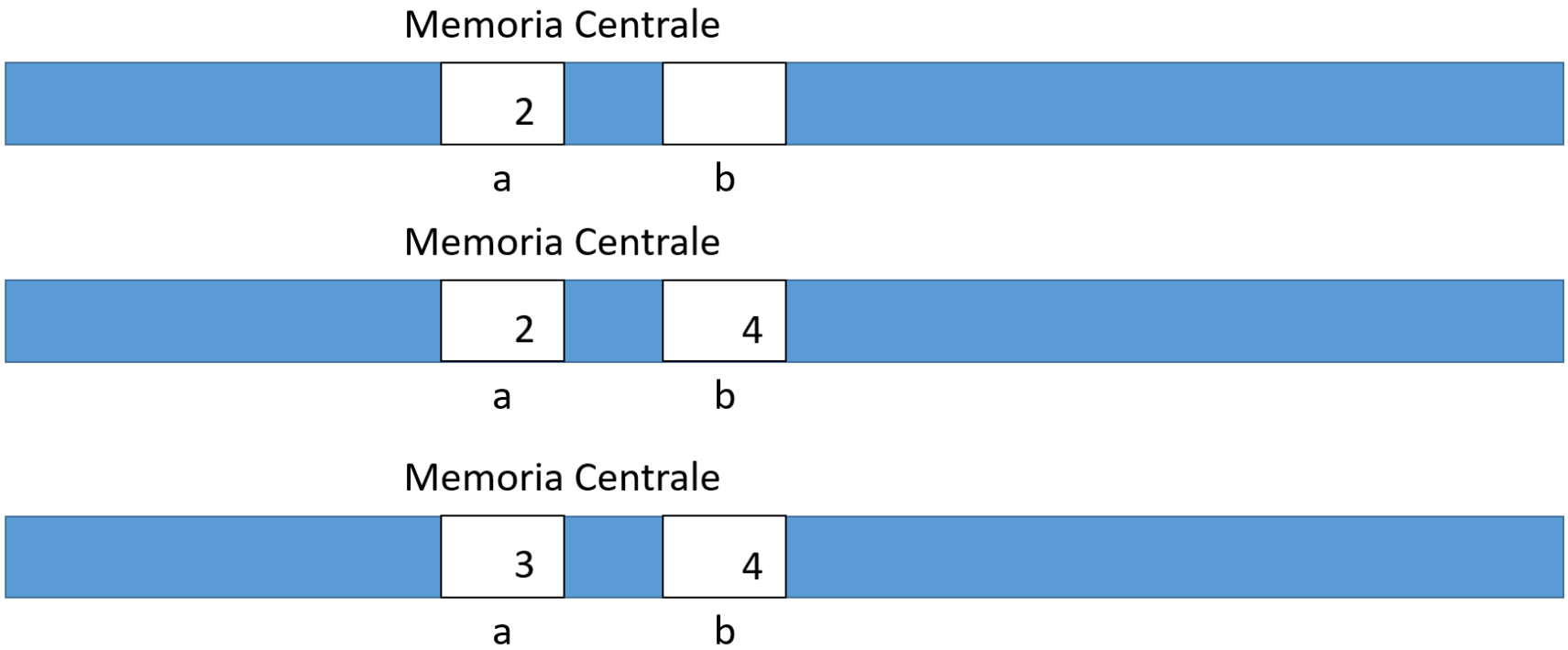
  Prima valuta e poi incrementa (la variabile viene incrementata dopo che il suo valore è stato usato
- `++a`

  Prima incrementa e poi valuta (la variabile viene incrementata prima che il suo valore venga usato

Programma: Input_03a.cpp

```
#include <iostream>

using namespace std;

int main()
{
    int a;
    int b;

    cin >> a;
    b = a++ * 2;
    cout << "a: " << a << endl;
    cout << "b: " << b << endl;

    return 0;
}
```

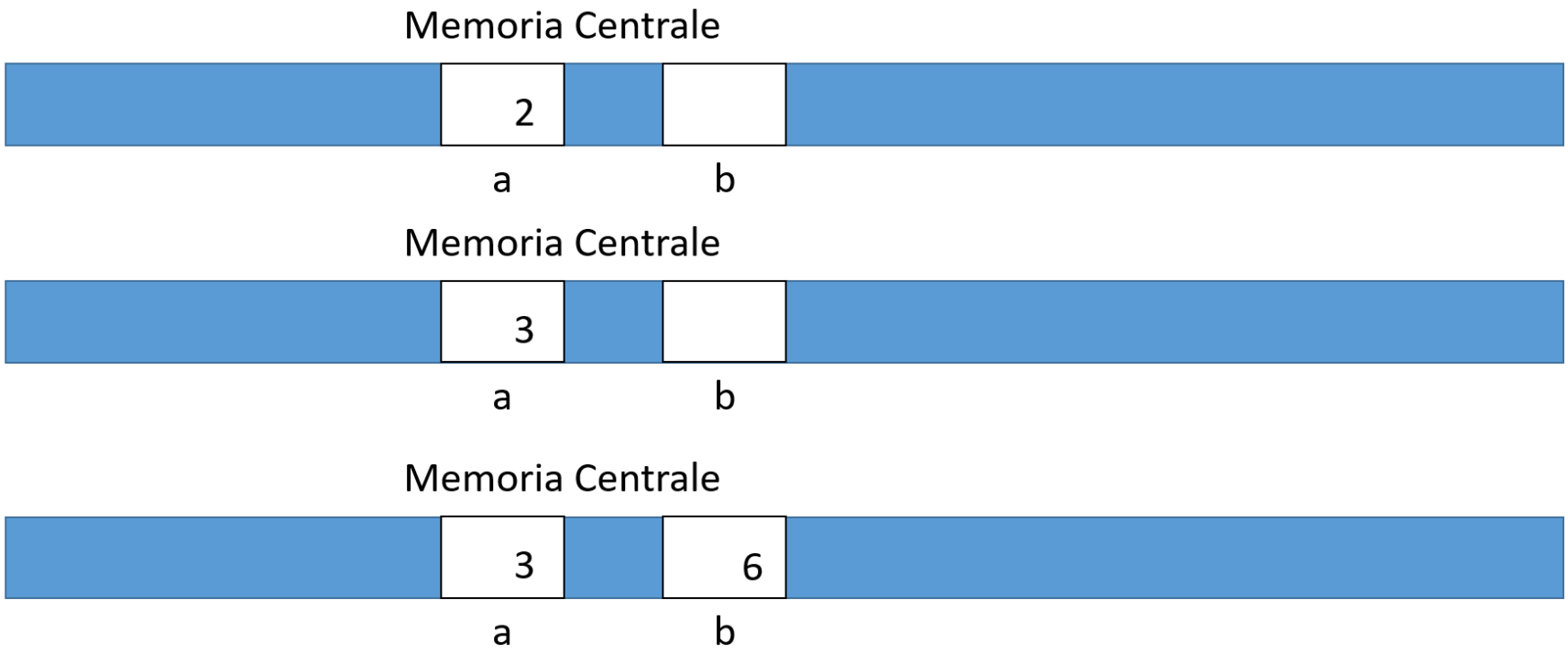
Traccia dell'Esecuzione in Memoria



Programma: Input_03b.cpp

```
#include <iostream>

using namespace std;

int main()
{
    int a;
    int b;

    cin >> a;
    b = ++a * 2;
    cout << "a: " << a << endl;
    cout << "b: " << b << endl;

    return 0;
}
```

Traccia dell'Esecuzione in Memoria



Programma: Input_03c.cpp

```
int main()
{
    int a;
    int b;

    cin >> a;
    a++;
    b = a * 2;
    cout << "a: " << a << endl;
    cout << "b: " << b << endl;

    return 0;
}
```

Tipi di Dato

- `int`
  Numeri interi con segno, su 32 bit
  (da -2.147.483.648 a +2.147.483.647)
- `unsigned int`
  Numeri interi senza segno (da 0 a +4.294.967.295)
- `short int`, `unsigned short int`
  Numeri interi (con e senza segno, su 16 bit (da -32768 a +32767 e da 0 a +65535)

Tipi di Dato

- `float`
  Numeri in **Virgola Mobile** (con parte frazionaria) su 32 bit
- `double`
  Numeri in virgola mobile su 64 bit
  Aumenta la precisione con numeri molto piccoli o la possibilità di rappresentare numeri molto più grandi

Tipi di Dato

- `char`
  Caratteri singoli, gestiti con il codice della tabella ASCII
  1 Byte, valori da -128 a + 127
- `unsigned char`
  Come `char` ma senza segno
  Un Byte, valori da 0 a +255.

Tabella ASCII

- American Standard Code for Information Interchange
- Definisce codici da 0 a 127 (usa 7 bit)
- Ad ogni codice è associato un simbolo (carattere)

Tabella ASCII

Wikipedia:
`https://it.wikipedia.org/wiki/ASCII`

Valori Costanti

- `12`
  Numero intero
- `12.0`
  Numero in virgola mobile
- `'A'`
  Costante carattere

Tipi e Espressioni

- Consideriamo le variabili

  ```
  int a=2;
  float b = 3.5;
  ```

- Qual è il tipo dell'espressione seguente?

  ```
  a + b
  ```

- È il tipo più **Capiente**, cioè quello che contiene più valori,

  Nel nostro caso, il `float`

- `int + float` $\rightarrow$ `float`
- Quindi il valore dell'espressione è `5.5`

Tipi e Assegnamenti

- Consideriamo

```
int a=2;
float b = 3.5;
float r;
r = a + b;
```

- In termini di tipi, è

```
float = int + float
```

cioè

```
float = float
```

Quindi si può fare

Tipi e Assegnamenti

- Consideriamo

```
int a=2;
int b = 3;
float r;
r = a + b;
```

- In termini di tipi, è

```
float = int + int
```

cioè

```
float = int
```

Quindi si può fare (i numeri interi sono contenuti nei numeri in virgola mobile)

Tipi e Assegnamenti

- Consideriamo

```
int a=2;
float b = 3.5;
int r;
r = a + b;
```

- In termini di tipi, è

```
int = int + float
```

cioè

```
int = float
```

Quindi **NON** si può fare (i numeri in virgola mobile NON sono contenuti nei numeri interi)

Conversione di Tipo

- **Operatore di Casting**
  Forza la conversione di tipo del valore alla sua destra
- (*tipo*) *espressione*
- Esempio:
  `(float)a`
  converte il valore della varibaile `a` (intera) in un numero in virgola mobile
- `(int)b`
  Converte il valore delal variabile `b` (virgola mobile) in intero, troncando la parte frazionaria
- ATTENZIONE Non cambia il valore memorizzato nelle variabili, ma cambia il tipo nell'espressione

Tipi e Assegnamenti

- Consideriamo

```
int a=2;
float b = 3.5;
int r;
r = (int)(a + b);
```

- In termini di tipi, è

```
int = int
```

  Quindi si può fare

Tipi e Divisione

- Il comportamento dell'operatore / dipende dal tipo dei suoi operandi
- `int / int` $\rightarrow$ `int`
  Divisione tra interi
  il valore calcolato è il **quoziente (intero)** della divisione
- `float / float` $\rightarrow$ `float`
  `int / float` $\rightarrow$ `float`
  `float / int` $\rightarrow$ `float`
  Divisione tra numeri in virgola mobile
  il valore calcolato è il **valore esatto (con parte frazionaria)** della divisione

Programma: Divisione_01c.cpp

```
int main()
{
    int a;
    int b;
    float r;

    cout << "inserisci due valori" << endl;
    cin >> a;
    cin >> b;

    r = a / b;
    cout << "r: " << r << endl;

    return 0;
}
```

Divisione

- Se inseriamo 3 e 2
  il valo stampato è `1`
  anche se `r` è definita come `float`
- Perchè i suoi operandi sono interi

Programma: Divisione_02c.cpp

```
int main()
{
    int a;
    int b;
    float r;

    cout << "inserisci due valori" << endl;
    cin >> a;
    cin >> b;

    r = a / (float)b;
    cout << "r: " << r << endl;

    return 0;
}
```

Programma: Divisione_03c.cpp

```
int main()
{
    int a;
    int b;
    float r;

    cout << "inserisci due valori" << endl;
    cin >> a;
    cin >> b;

    r = (float)a / b;
    cout << "r: " << r << endl;

    return 0;
}
```

Divisione

- Se inseriamo 3 e 2
  il valore stampato è `1.5`
- Notare che l'operatore di casting **ha precedenza** sulla divisione
  Se avete dubbi, scrivete
  `r = ((float)a) / b;`