Lezione 2

Introduzione al linguaggio C/C++
Tipi di dato e numeri interi
Variabili e costanti con nome
Struttura di un programma

Promemoria per chi segue

- Non dimenticate le raccomandazioni
  - Se non state lavorando al PC, siete al momento tra i **candidati alla bocciatura**
    - Le slide sono uno strumento molto inefficace se non assimilate i concetti mediante gli esercizi
  - Fate di tutto per tenere alto il vostro livello di concentrazione
  - Non esitate a fare domande!

Rinfresco esercitazione ...

- ... e qualche piccolo altro passo avanti sugli aspetti pratici
  - Seguire le slide della seconda esercitazione fino all'esercizio di stampa di una variabile intera escluso

Storia essenziale del C

- Definito nel 1972 (AT&T Bell Labs) per sostituire l'*assembler* nella programmazione di sistemi operativi: in pratica, nato per creare UNIX
- Prima definizione precisa: Kernigham & Ritchie (1978)
- Prima definizione ufficiale: **ANSI C** (1983)

Ma già nel 1980 ...

... erano in uso varie versioni di un linguaggio denominato “C con le classi”

- Erano le prime versioni di quello che sarebbe stato il C++
- Inventato, definito, ed implementato per la prima volta, da Bjarne Stroustrup http://www.research.att.com/~bs/
- Primo standard nel 1998: ISO/IEC 14882
  - Ora siamo allo standard C++11
- Decisamente di successo:
  http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
  http://www.stroustrup.com/applications.html

Cosa vedremo del C++

- Solo per chi sa di cosa parlo: del linguaggio C++ vedremo solo il sottoinsieme procedurale
- NON vedremo la programmazione ad oggetti
- Sarà argomento dell'insegnamento di **Programmazione ad Oggetti**

Iniziamo ...

- Affronteremo lo studio del linguaggio incominciando dai seguenti concetti:
  - Introduzione essenziale del tipo **int**
  - Memoria di un calcolatore, processore, linguaggio macchina e linguaggi di alto livello
  - Memoria di un programma C/C++ ed oggetti
  - Espressioni letterali
  - Variabili e costanti con nome
  - Compendio tipi di dato primitivi
  - Struttura (semplificata) di un programma

Tipo **int**

- In un programma C/C++ si possono scrivere dei numeri interi, ad esempio
  `6` **`12`** **`700`**
- Si possono inoltre definire delle **variabili** di tipo **int**
  - Sono dei *contenitori* in cui *memorizzare* numeri interi
    - Possono contenere un sottoinsieme limitato dei numeri interi, come vedremo meglio
    - Il contenuto di una variabile di tipo **int** può cambiare durante l'esecuzione del programma

Definizione

- Per poter utilizzare una variabile di tipo **int** in un programma, bisogna prima definirla
- Nella definizione si stabilisce fondamentalmente:
  - l'*identificatore*
    - ossia il nome che useremo per riferirci alla variabile
  - opzionalmente, il valore iniziale della variabile (*inizializzazione*)

Esempi

- Ecco due esempi di definizione di variabili di tipo **int**

```
int a;    // definizione di una
          // variabile di nome a e di
          // tipo int

int k=5;  // definizione di una
          // variabile di nome k e di
          // tipo int, inizializzata col
          // valore 5
```

Esercizi

- Svolgere i primi due esercizi di stampa di una variabile intera contenuti nella seconda esercitazione

Variabili

Variabile

- Una variabile è un contenitore all'interno del quale di può memorizzare un valore
- Tale valore <u>può variare nel tempo</u>

Definizione di una variabile

- In C/C++ è necessario elencare ogni variabile che sarà utilizzata nel programma, prima di utilizzarla
- In particolare si dice che bisogna **definire** ciascuna variabile. All'atto della definizione bisogna attribuire alla variabile
  - un tipo
  - un nome (identificatore) col quale ci si riferirà poi a tale variabile
  - eventualmente un valore iniziale (inizializzazione)

Esempi

- Prima di vedere formalmente la sintassi, ricordiamo i due esempi di definizione di variabili di tipo **int**

```
int a;   // definizione di una
         // variabile di nome a e di
         // tipo int
int k=5; // definizione di una
         // variabile di nome k e di
         // tipo int, inizializzata col
         // valore 5
```

Valore iniziale

- Che valore assume una variabile se non viene inizializzata?
- Per il momento diciamo che assume un valore **casuale**
- Poi vedremo meglio i singoli casi

Nota sulla sintassi

- Nella descrizione della sintassi del linguaggio C/C++ utilizzeremo la notazione con parentesi quadre *[...]* per denotare elementi **opzionali**, ossia parti che possono o meno comparire
- Tutto ciò che non sarà contenuto tra tali parentesi *[...]* quadre sarà **obbligatorio**

Sintassi definizione variabile

- Sintassi della definizione di una variabile:

  nome_tipo nome_variabile [= valore_iniziale] ;

- E' possibile raggruppare le definizioni di più variabili dello stesso tipo in una lista separata da **,**
  - Forma generale definizione variabili:

    nome_tipo nome_variabile1 [=valore_iniziale],
    nome_variabile2 [= valore_iniziale],
    ... ;

Completamento esempi

```
int a, c;// definizione di due
         // variabili di nome a e c, di
         // tipo int

int k=5, d;  // definizione di due
             // variabili di nome k e d,
             // di tipo int, di cui la
             // prima è inizializzata
             // col valore 5
```

- Vedremo successivamente esempi di definizioni di variabili di tipo diverso da **int**

Visibilità di una variabile

- Una variabile è **visibile**, ossia può essere utilizzata, solo a partire dal punto in cui viene definita nel testo del programma

Istruzione semplice

- Una definizione è di fatto una istruzione del C/C++
- In particolare si tratta di una cosiddetta istruzione semplice

Assegnamento

- Si può assegnare un nuovo valore ad una variabile mediante una **istruzione di assegnamento**
  nome_variabile = *espressione* ;
- Esempi:

```
int v = 3 ;// definizione variabile v
cout<<v<<endl ;
v = 4 ;         // assegna il valore 4
                // alla variabile v
```

Esercizi

- Svolgere i successivi esercizi della seconda esercitazione, fino alla slide in cui ci si chiede cosa succede se una lettura da *stdin* fallisce

Ultimo standard C++

- Gli standard, ossia le definizioni ufficiali dei linguaggi di programmazione, in generale evolvono col tempo
- Il passaggio da uno standard all'altro introduce spesso cambiamenti che fanno sì che uno stesso programma si comporti in modo diverso o addirittura non si compili più
- L'ultimo standard disponibile per il linguaggio C++ è denominato C++ 2011
  - Spesso abbreviato con C++11
  - Le versioni più recenti dei compilatori tipicamente supportano anche lo standard C++11
  - E' però molto probabile che ancora utilizzino, come configurazione predefinita, lo standard precedente al C++11

Comportamento gcc 1/2

- Questo vale anche per il gcc
- Supposto che stiate utilizzando una versione del gcc che supporta anche il nuovo standard, avete due possibilità
  1) Il compilatore già usa lo standard C++11 di default
  2) Il compilatore non usa lo standard C++11 di default
- Nel gcc lo standard C++11 è denotato come **`c++0x`**
  - Vecchio nome dello standard C++11

Comportamento gcc 2/2

- Potete verificare:
  - se lo standard C++11 è supportato dalla versione del compilatore che state utilizzando, e
  - se è selezionato di default

  controllando, per esempio, la descrizione dell'opzione **`-std`** nella pagina di manuale di g++
- Se il compilatore supporta lo standard C++11 ma non utilizza di default, dovete aggiungere l'opzione
  `-std=c++0x`
  alla riga di comando

Informazioni generali

- Potete controllare lo stato attuale degli standard C++ all'URL
  - http://www.open-std.org/jtc1/sc22/wg21/
- L'ultimo draft dello standard si può scaricare gratuitamente
  - Lo standard è il documento su cui è scritto **TUTTO** quello che riguarda il linguaggio
  - E' la risorsa autoritativa da utilizzare per chiarire ogni dubbio o conoscere ogni dettaglio
- Potete controllare lo stato di avanzamento del gcc in merito allo standard C++11 alla pagina

  http://gcc.gnu.org/projects/cxx0x.html

Esercizi

- Riprendere la seconda esercitazione, fino all'esercizio sulla moltiplicazione escluso

Un pò di nozioni

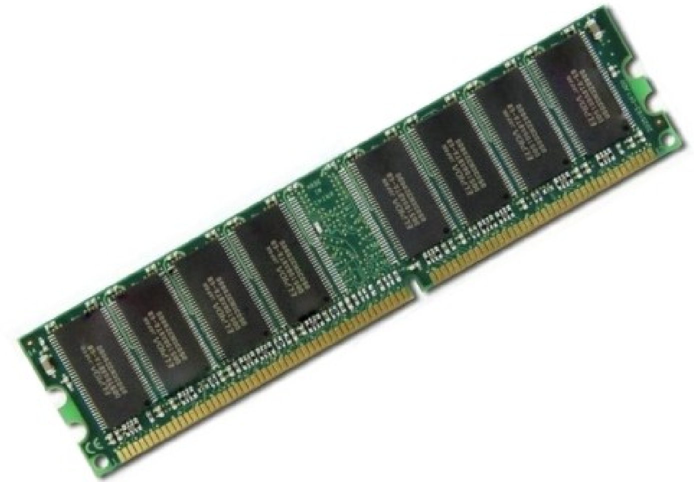
- Ora che abbiamo acquisito un po' più di familiarità col linguaggio, cominciamo ad accrescere le nostre conoscenze
  - Stiamo per affrontare una sequenza relativamente lunga di nuovi concetti prima del prossimo esercizio
- Il concetto fondamentale su cui costruiremo le nozioni riportate in questa presentazione è quello di **memoria**

Memoria principale

- Partiamo da qualche dettaglio sul funzionamento interno di un elaboratore
- In particolare, vedremo:
  - *memoria principale*
  - *processore*
  - *linguaggio macchina*
  - *linguaggi ad alto livello*

Memoria principale e celle

- Definiamo **memoria** (principale) di un elaboratore il contenitore in cui sono memorizzati tutti i dati su cui lavora il processore
- Possiamo schematizzare la memoria come una sequenza contigua di **celle** (chiamate anche **locazioni di memoria**)
- Ciascuna cella fornisce l'**unità minima di memorizzazione**, ossia l'elemento più piccolo in cui si può memorizzare un'informazione

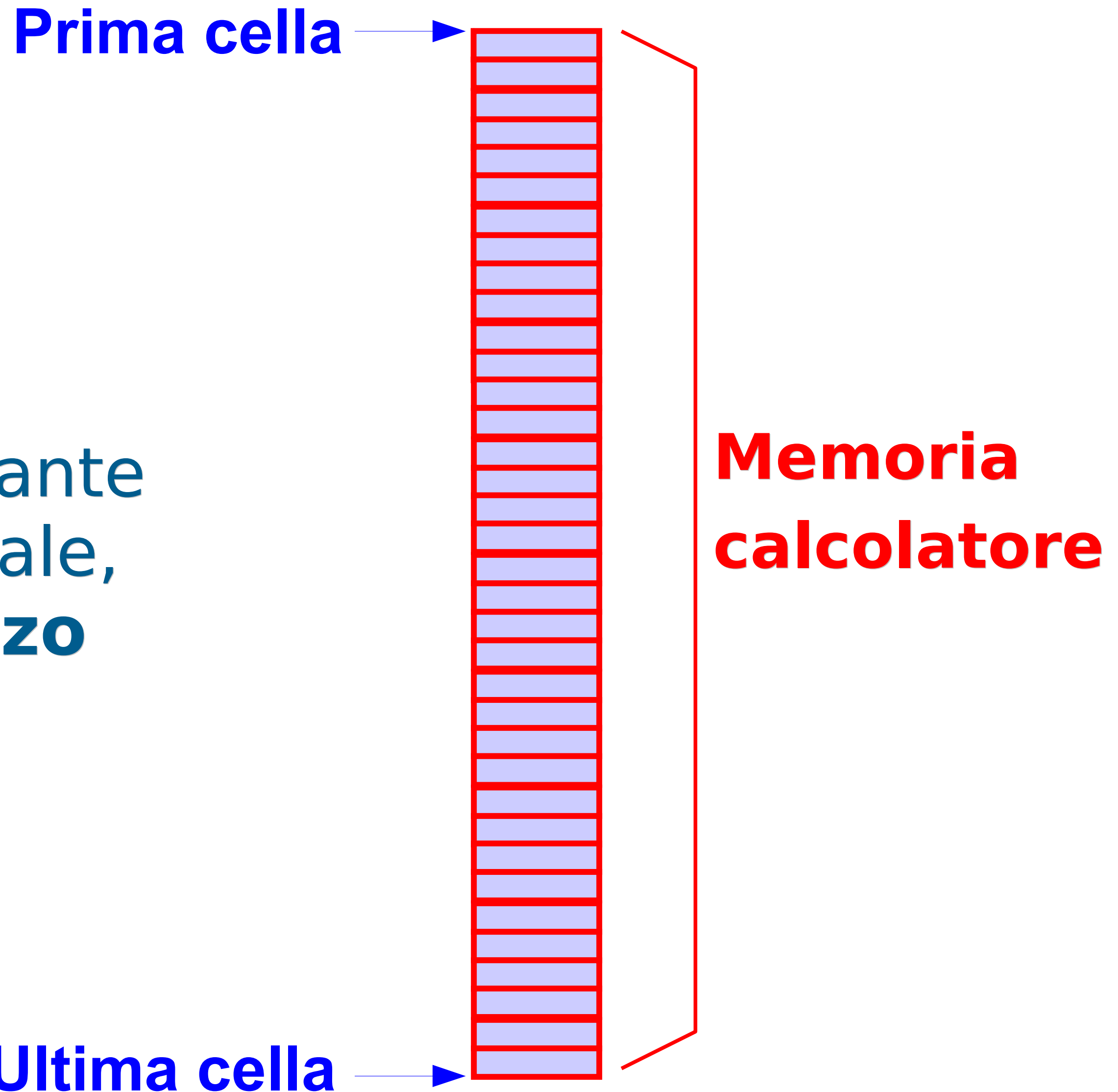


Contenuto cella

- Ogni cella contiene un *byte*, ossia una sequenza di *bit* (cifre binarie)
  - Tipicamente un byte è costituito da 8 bit
    Esempio: 01100101
- Tutte le celle hanno quindi la stessa dimensione in termini di numero di bit
- In generale l'esatto numero di bit in un byte potrebbe variare da una macchina all'altra

Schema memoria

- Ciascuna cella è **univocamente** individuata mediante un numero naturale, chiamato **indirizzo** della cella



Celle di memoria e numeri

- I bit contenuti in una cella possono essere utilizzati per memorizzare un numero
  - Il numero è rappresentato nella cosiddetta notazione binaria
- Senza entrare nei dettagli della notazione binaria, facciamo solo un esempio di come si ottiene questo risultato, per esempio con i numeri *naturali* (ossia gli interi non negativi)

Rappresentazione numeri

- Facciamo corrispondere un numero ad ogni **combinazione** (**configurazione**) di bit
- Esempio in caso di cella da 8 bit:

00000000	0
00000001	1
00000010	2
00000011	3
...	
11111111	255

Numeri negativi

- Con una tecnica simile si possono rappresentare anche numeri negativi, facendo corrispondere un certo sottoinsieme delle possibili configurazioni di bit ai numeri positivi, e l'altro sottoinsieme ai numeri negativi
- Idea più semplice
  - Utilizzare un bit per il segno

Domanda

- E' possibile memorizzare il contenuto di una variabile di tipo **int** all'interno di una cella di memoria?

Risposta

- Se il valore è, per esempio, più grande di 255 allora certamente no!

Uso di celle consecutive 1/2

- Infine, per rappresentare numeri più grandi di quelli rappresentabili con una sola cella, si **accorpano** più celle consecutive
  - Si usano per esempio tutte le configurazioni possibili di bit di una sequenza di due o quattro celle contigue
  - Vediamo un esempio

Rappresentazione numeri 2/2

- Esempio in caso di due celle da 8 bit ciascuna:

00000000 00000000 0

00000000 00000001 1

00000000 00000010 2

00000000 00000011 3

...

00000000 11111111 255

00000001 00000000 256

00000001 00000001 257

00000001 00000010 258

00000001 00000011 259

...

00000001 11111111 511

00000010 00000000 512

00000010 00000001 513

00000010 00000010 514

.
.
.

Rappresentazione int

- Una variabile di tipo int è tipicamente rappresentata su 4 celle consecutive
- Vedremo i dettagli in seguito

Processore

- Gli altri elementi da considerare per capire i concetti alla base del linguaggio C/C++ sono il processore ed il suo linguaggio

- Tutte le operazioni di elaborazione delle informazioni effettuate da un calcolatore sono
  - svolte direttamente dal processore, oppure
  - svolte da altri componenti dietro comando del processore

Operazioni

- Un processore è in grado di compiere solo operazioni molto semplici:
  - lettura/scrittura/copia di una o più celle di memoria
  - somma/sottrazione/moltiplicazione/divisione del contenuto di una o più celle di memoria
  - lettura/scrittura in zone di memoria 'speciali' per pilotare dispositivi di ingresso/uscita (ad esempio schede video)
  - altre semplici operazioni sulle celle di memoria
- Tipicamente un processore riesce a lavorare su un certo numero di celle contigue alla volta. Tale sequenza di celle è detta **parola di macchina (machine word)**
  - Si dice che un processore ha una architettura a 16, 32 oppure 64 bit se lavora su parole da 2, 4 oppure 8 byte

Linguaggio macchina

- Ogni processore è caratterizzato da un proprio insieme di **istruzioni**, tramite le quali è possibile fargli svolgere le precedenti operazioni

- L'insieme delle istruzioni di un processore viene chiamato **linguaggio macchina** di quel processore

- Ogni istruzione è identificata da una certa configurazione di bit

- Segue un esempio di programma in linguaggio macchina

Esempio programma

0011001000110110

0101010100100011

0011011001010101

1110001111100011

...

1001101011100011

Linguaggio macchina

- Per far eseguire un programma ad un processore, basta
  - memorizzare da qualche parte nella memoria la sequenza di configurazioni di bit relativa alle istruzioni da eseguire
  - dire al processore a che indirizzo si trova la prima di tali istruzioni
- Il processore eseguirà, una dopo l'altra, le istruzioni che trova a partire da tale indirizzo

Ordine di esecuzione 1/2

- Ordine di esecuzione *predefinito* delle istruzioni: l'una dopo l'altra

0011001000110110

0101010100100011

0011011001010101

1110001111100011

...

1001101011100011

Ordine di esecuzione 2/2

L'ordine con cui sono eseguite le istruzioni cambia solo se vengono incontrate speciali <u>istruzioni di salto</u> verso un diverso indirizzo

0011001000110110

0101010100100011

0011011001010101

1110001111100011

...

1001101011100011

Cambio di ordine dovuto ad una istruzione di salto *in avanti*

Un salto può anche avvenire all'*indietro*, ossia verso un indirizzo inferiore rispetto a quello in cui si trova l'istruzione di salto stessa

Difficoltà linguaggio macchina

- In definitiva, data la semplicità delle istruzioni e dei dati su cui lavora un processore si ha che:
  - scrivere (interamente) in linguaggio macchina un programma che faccia cose complesse,
    - quale ad esempio un sistema operativo o anche più semplicemente un programma che deve disegnare/aggiornare un'interfaccia grafica ed usarla per interagire con gli utenti,

  diviene un lavoro estremamente impegnativo e costoso

Linguaggi di alto livello 1/3

- Questo è fondamentalmente il motivo per cui sono stati inventati moltissimi altri linguaggi cosiddetti ad alto livello, che sono molto più 'vicini' al linguaggio umano rispetto al linguaggio macchina
- Tali linguaggi si basano sul concetto di ***astrazione*** dalla macchina sottostante: astraggono dai dettagli, cosiddetti di *basso livello*, quali ad esempio celle di memoria ed indirizzi, e permettono al programmatore di ragionare e di scrivere il proprio programma in termini di dati ed operazioni più complessi.

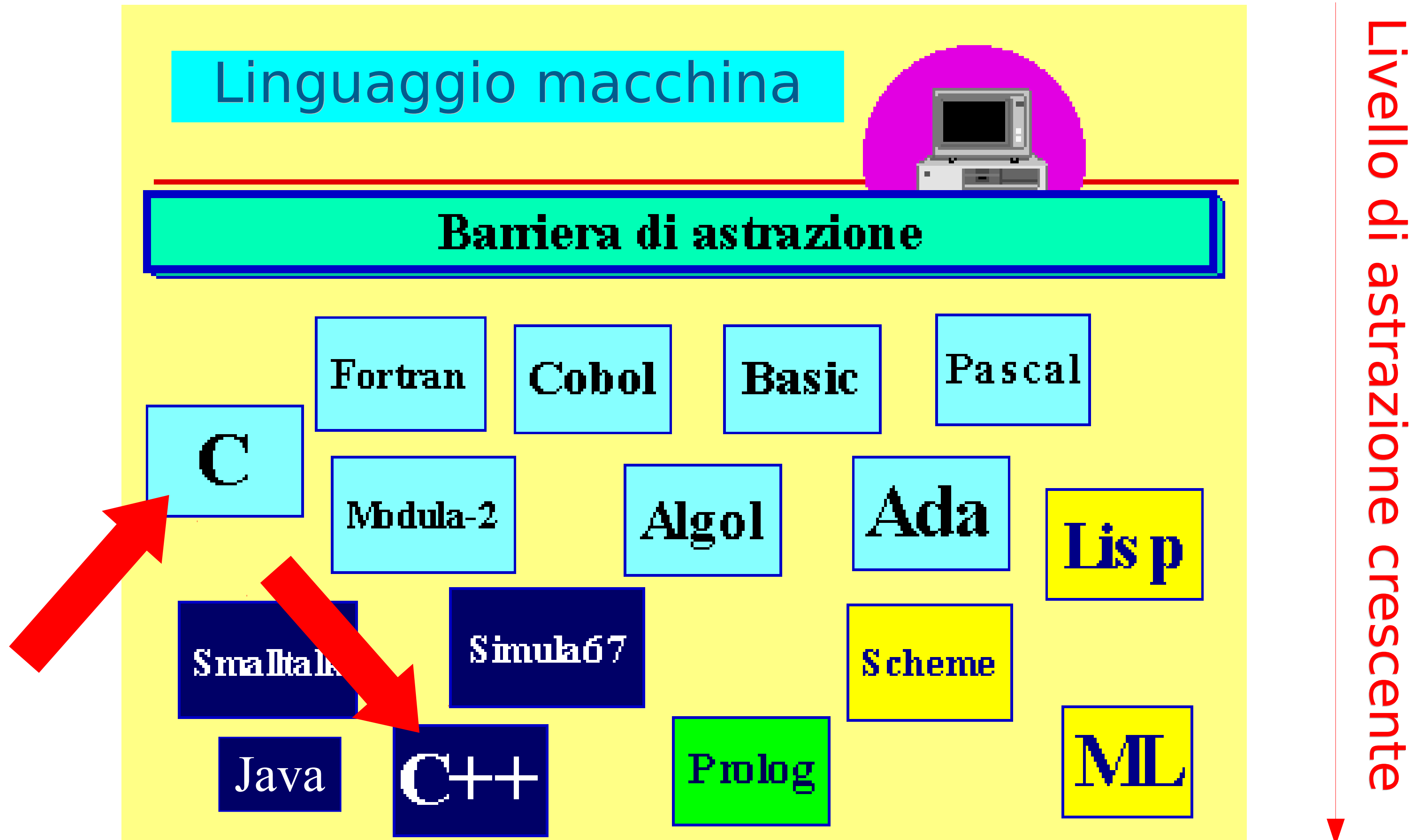
Linguaggi di alto livello 2/3

- Ad esempio, quando abbiamo utilizzato variabili di tipo **int**
  - Ci siamo preoccupati di come memorizzare i valori in una sequenza di celle?
  - Ci siamo preoccupati di dove memorizzare esattamente in memoria tale sequenza di celle?
- No, abbiamo utilizzato il tipo **int** nel suo significato astratto di contenitore di numeri interi

Linguaggi di alto livello 3/3

- Riassumendo, col tipo **int** (e lo stesso accadrà con gli altri tipi di dato)
  - **si astrae dalle singole celle di memoria**: non si vedono più le singole celle di memoria in cui sono memorizzati i numeri
  - si può quindi ragionare e scrivere il programma direttamente in termini di numeri interi
    - si lavora cioè ad alto livello, senza preoccuparsi di come e dove saranno realmente memorizzati e manipolati tali numeri a basso livello
- In generale, dato un problema da risolvere, disporre di dati ed operazioni più astratti e complessi permette di descrivere in modo molto più semplice e chiaro gli elementi del problema ed i passi che si debbono effettuare

Linguaggio ad alto livello 1/2



Linguaggio ad alto livello 2/2

- Il C/C++ è quindi un linguaggio di alto livello
- Il fatto di non coincidere con il linguaggio macchina di nessun processore ha però un prezzo
  - Per poter essere eseguito da un calcolatore, un programma scritto in C/C++ va prima tradotto nel linguaggio macchina del processore del calcolatore su cui lo vogliamo eseguire
  - Questa operazione viene comunemente chiamata **compilazione**, ed i programmi che la eseguono vengono chiamati **compilatori**

Memoria di un programma C/C++ ed oggetti

Memoria di un programma

- Definiamo **memoria** di un programma in esecuzione, o *processo*, il contenitore (logico) in cui sono memorizzati tutti i dati del programma (ed altre informazioni che vedremo in seguito) durante la sua esecuzione
- Nei programmi C/C++ la memoria di un programma ha la stessa identica struttura della memoria del calcolatore vista precedentemente: è una sequenza contigua di **celle** (**locazioni di memoria**) che costituiscono l'unità minima di memorizzazione
- Le celle, tutte della stessa dimensione, contengono un *byte* ciascuna

Dimensione byte

- L'esatta dimensione che deve avere un *byte* non è specificata nello standard del linguaggio C/C++, e, come abbiamo visto, teoricamente può variare da una macchina all'altra
  - Lo standard specifica solo che un byte **deve** essere grande abbastanza da contenere un oggetto di tipo **char**
  - Vedremo in seguito cosa è un oggetto di tipo **char**, per ora ci basta sapere che è utilizzato principalmente per memorizzare caratteri

Dalle celle ai dati

- In C/C++ si possono memorizzare delle informazioni più complesse dei semplici numeri interi rappresentabili con una singola cella di memoria
- Si possono memorizzare i dati all'interno di contenitori che chiameremo genericamente **oggetti**

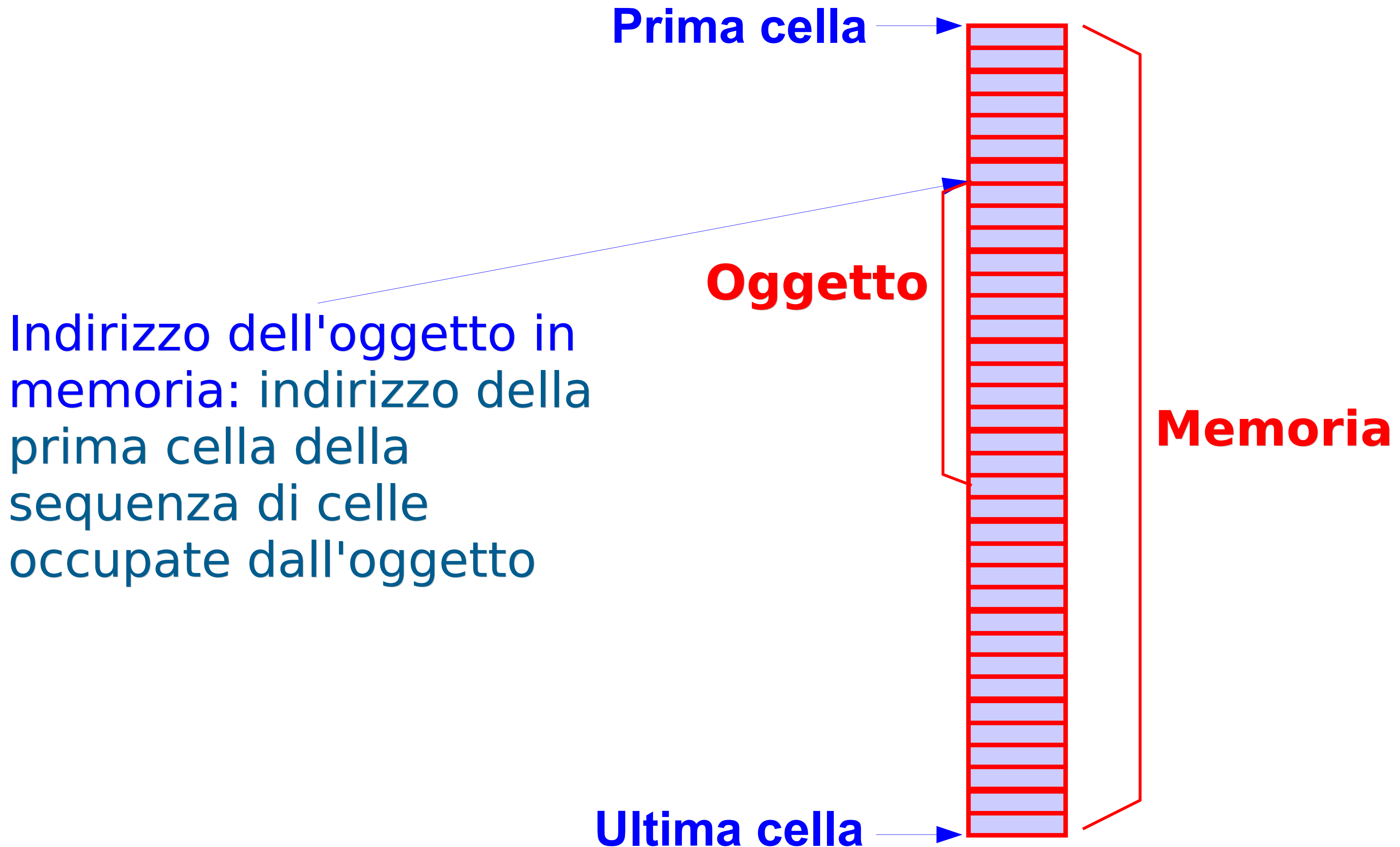
Oggetto, valore, memoria

- Un **oggetto** è un'astrazione di cella di memoria
  - E' caratterizzato da un **valore**
  - E' memorizzato in una **sequenza di celle contigue**
    - Consideriamo per esempio, come oggetto, un numero naturale maggiore di 255
    - Come abbiamo visto, così come si può rappresentare ogni numero naturale da 0 a 255 con una determinata configurazione di 8 bit, si può rappresentare un valore naturale maggiore di 255 su *N* celle consecutive, con una determinata configurazione dei risultanti 8**N* bit

Digressione su oggetti

- Per chi avesse già avuto a che fare con i cosiddetti *linguaggi ad oggetti*
  - Gli oggetti di cui parliamo in questo corso sono un concetto più generale di quello di oggetto definito in tali linguaggi
  - Useremo cioè il termine oggetto col significato generale di contenitore di informazioni (valori)

Oggetto in memoria, indirizzo



Domanda

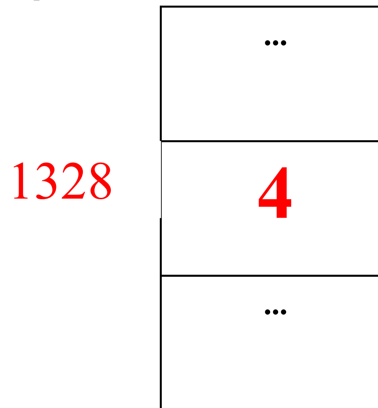
- Abbiamo già utilizzato qualche tipo di oggetto?

Risposta

- Sì, gli oggetti di tipo **int**

Indirizzo, valore e tipo 1/2

- Un oggetto è caratterizzato da
  - un *indirizzo*
    - Ad esempio 1328, il che vuol dire che l'oggetto si trova in memoria a partire dalla cella di indirizzo 1328



  - un *valore*
    - In questo semplice esempio l'oggetto è di tipo numerico, occupa una sola cella e la configurazione di bit della cella rappresenta il valore 4

Indirizzo, valore e tipo 2/2

- un **tipo (di dato)**
  - Specifica i valori possibili per l'oggetto e le operazioni che si possono effettuare sull'oggetto

Tipi di dato primitivi

Tipo di dato

- **Tipo di un dato (oggetto)**

  Insieme di valori che l'oggetto può assumere ed insieme di operazioni che si possono effettuare su quell'oggetto

- Quali tipi di dato esistono in C/C++?
  - Partiamo dai tipi di dato primitivi

Tipi di dato primitivi

Quattro tipi di dato primitivi

Nome tipo	*Categoria di dati che rappresenta*
int	sottoinsieme dei numeri interi
float	sottoinsieme dei numeri reali
double	sottoinsieme dei numeri reali con maggiore precisione rispetto al tipo **float**
char	caratteri
bool	booleani (vero/falso, solo C++)

Per ora vedremo più in dettaglio il solo tipo **int**

Tipo **int**

- Il tipo **int** è diverso dal tipo INTERO inteso in senso matematico, dove l'insieme infinito degli interi Z è dato da
  {...,-2,-1,0,+1,+2,...}
- Ovvero il tipo **int** ha un insieme di valori limitato:
  - *L'insieme esatto dei valori possibili dipende dalla macchina*
  - Normalmente il compilatore è configurato in maniera tale che gli oggeti di tipo **int** siano memorizzati in una PAROLA DI MACCHINA, che tipicamente è lunga 2, 4 o 8 byte, ossia 16, 32 o 64 bit
  - Se la macchina ha parole a 16 bit:
    $[-2^{15}, 2^{15}-1]$ ovvero [-32768, +32767]
  - Se la macchina ha parole a 32 bit:
    $[-2^{31}, 2^{31}-1]$ ovvero [-2147483648, +2147483647]
  - ...

Operazioni aritmetiche **int**

- Al tipo **int** sono applicabili i seguenti operatori:

+	Addizione
-	Sottrazione
*****	Moltiplicazione
/	Divisione intera (diverso dalla divisione reale!) Es., 10/3 = 3
%	Modulo (resto della divisione intera) Es., 10%3 = 1 5%3 = 2 In formula: dati tre numeri naturali *divid*, *divis* e *ris*, dove ris=divid/divis (divisione intera), il resto è il numero naturale *res* tale che *divid = ris * divis + res*

Esempio

```
int v ;  // definizione variabile v
v = 4 ;      // assegna il valore 4
             // alla variabile v
v = 2 * 3 ;  // assegna il valore 6
             // alla variabile v
```

In seguito, vedremo in dettaglio tutti i tipi di *espressioni* che si possono scrivere

Esercizi

- Svolgere i successivi esercizi della seconda esercitazione, fino all'esercizio di scambio dei valori di due variabili escluso

Espressioni letterali

Espressioni letterali

- Le espressioni letterali denotano **valori costanti**
- Sono spesso chiamate semplicemente *letterali* o *costanti senza nome*
- Le possibili espressioni letterali utilizzabili in C/C++ sono
  - numeri interi
  - numeri reali
  - costanti carattere
  - costanti stringa
- Vedremo le ultime tre categorie più avanti

Numeri interi

In quanto invece ai numeri interi, ecco alcuni ovvi esempi dei letterali utilizzabili in un programma C/C++:

```
6          12          700
```

Costanti con nome

Costanti con nome

- Una definizione di una *costante con nome* associa permanentemente un oggetto di valore costante ad un identificatore
- La definizione è identica a quella di una variabile, a parte
  - Aggiunta della parola chiave **`const`** all'inizio
  - Obbligo di inizializzazione
- Esempi:

```
const int N = 100;
const int L ;    // errato: manca
                 // inizializzazione
```

- Per ora consideriamo solo costanti con nome di tipo **int**

Costanti e variabili

- Una costante è un'astrazione **simbolica** di un valore: si da cioè un nome ad un valore
- E' una associazione identificatore-valore che **non cambia mai** durante l'esecuzione
- **Non si può** quindi **assegnare un nuovo valore ad una costante** mediante una istruzione di assegnamento

- Invece, nel caso di una **variabile**
  - L'associazione identificatore-indirizzo non cambia mai durante l'esecuzione, ma può cambiare l'*associazione identificatore-valore*
  - Uno stesso identificatore può denotare valori differenti in momenti diversi dell'esecuzione del programma

Esercizio 1/2

- Scrivere un programma in cui si definisce una costante intera e se ne stampa il valore sullo schermo col seguente formato:

Il valore della costante è 10.

- E si va a capo

Esercizio 2/2

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

main()

{

  const int i = 10 ;

  cout<<"Il valore è “<<i<<”.”<<endl ;

}
```

Struttura (semplificata) di un programma

Struttura programmi

- In questo insegnamento vedremo solo programmi sviluppati su di un unico file sorgente
  - Vedrete lo sviluppo di un programma su più file nel corso di **Programmazione II**
- Nelle prossime slide iniziamo a vedere la struttura semplificata di un programma
- Come primo passo, per motivare la presenza delle cosiddette *direttive* in un programma, partiamo dal menzionare il *pre-processore*

Pre-processore

- Prima della compilazione vera e propria, il file sorgente viene manipolato dal cosiddetto **pre-processore**, il cui compito è effettuare delle modifiche o delle aggiunte al testo originario
- La nuova versione del programma viene memorizzata in un **file temporaneo**, ed è questo il vero file che viene passato al compilatore
  - Il file temporaneo è poi automaticamente distrutto alla fine della compilazione
- Vedremo in seguito cosa fa il pre-processore in dettaglio, quello che ci basta sapere per ora è che il pre-processore viene pilotato dal programmatore mediante le cosiddette **direttive** inserite nel file sorgente

Dichiarazioni e definizioni

- Nelle prossime slide metteremo in evidenza un tipo di istruzioni chiamate **dichiarazioni**
  - Una dichiarazione è una istruzione in cui si introduce un nuovo identificatore
- Le definizioni sono casi particolari di dichiarazioni
  - Sono dichiarazioni la cui esecuzione provoca l'allocazione di spazio in memoria
  - In particolare, la definizione di una variabile o di una costante con nome provoca l'allocazione di spazio in memoria per la variabile o costante che viene definita

Struttura programma C

```
#include <stdio.h>
```
← Direttive per il pre-processore

```
main()
{
        <dichiarazione>
        <dichiarazione>
        ...
        <dichiarazione>

        <istruzione diversa da dichiarazione>
        <istruzione diversa da dichiarazione>
        ...
        <istruzione diversa da dichiarazione>
}
```

Obbligatorio: prima tutte le dichiarazioni, poi qualsiasi altro tipo di istruzione

Struttura programma C++

```
#include <iostream>
```
Direttive per il pre-processore

```
using namespace std ;

main()

{

	<istruzione qualsiasi>
	<istruzione qualsiasi>
	...
	<istruzione qualsiasi>
}
```

Diversamente dal C, in C++ si possono mescolare tutti i tipi di istruzioni

Funzione *main*

- *main()* è una funzione speciale con tre caratteristiche:
  - deve essere sempre presente
  - la prima istruzione della funzione *main()* è la prima istruzione del programma che sarà eseguita, indipendentemente da dove si trova la funzione *main()* all'interno del file sorgente
  - quando termina l'esecuzione del *main()*, ossia dopo dopo l'esecuzione dell'ultima istruzione contenuta nella funzione *main()*, termina l'intero programma
- Come si è visto, in C la funzione *main()* contiene due sezioni
  - Parte dichiarativa
  - Parte esecutiva vera e propria

Ordine di esecuzione

- In che ordine vengono eseguite le istruzioni?
- Si definisce **sequenza** o **concatenazione** una sequenza di istruzioni scritte l'una di seguito all'altra all'interno di un programma
- Le istruzioni/dichiarazioni di una sequenza sono **eseguite l'una dopo l'altra**

  ESEMPIO

```
int N ;            // prima si esegue la definizione
N = 3 ;            // poi l'assegnamento
cout<<N<<endl;     // infine la stampa
```

Esercizi

- Svolgere tutti i rimanenti esercizi della seconda esercitazione
- Prestare molta attenzione alla descrizione del processo risolutivo riportata in tale esercitazione