Lezione 2

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

Passi fondamentali 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/
- Standardizzato nel 1998: ISO/IEC 14882
- Decisamente di successo:
  http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Cosa vedremo del C++

- 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:
  - Memoria di un calcolatore, processore, linguaggio macchina e linguaggi di alto livello
  - Memoria di un programma C/C++ ed oggetti
  - Tipi di dato primitivi
  - Espressioni letterali
  - Variabili e costanti con nome
  - Struttura (semplificata) di un programma

Memoria

- Per capire meglio i concetti di base del linguaggio C/C++, vediamo prima qualche dettaglio sul funzionamento interno di un elaboratore
- Partiamo dalla memoria principale

Celle

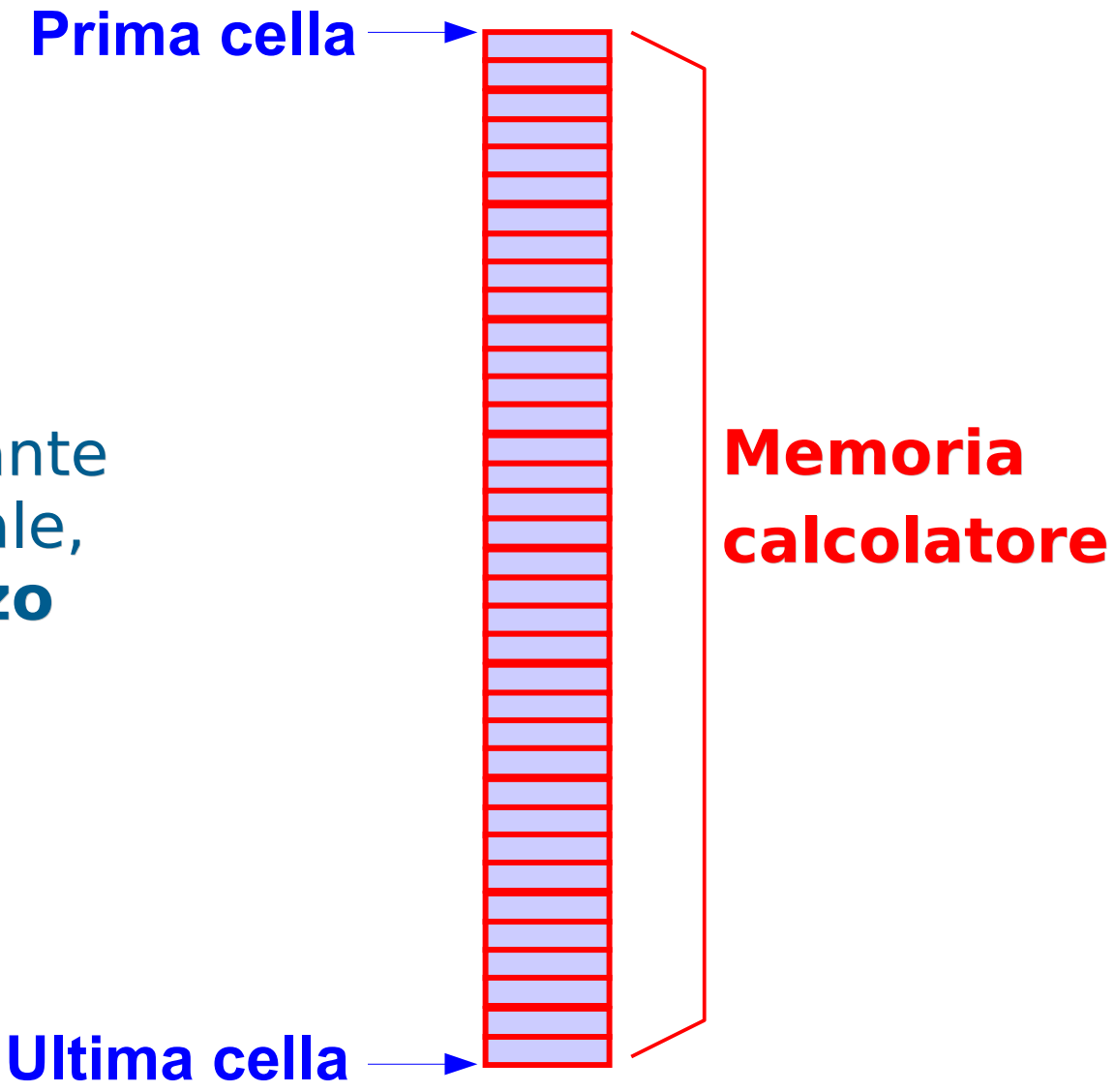
- Definiamo **memoria** di un elaboratore il contenitore in cui sono memorizzati tutti i dati su cui lavora il processore
- Possiamo schematizzare la memoria come una sequenza contigua di **celle** (chiamate anche **locazioni di memoria**)
- Ciascuna cella fornisce l'**unità minima di memorizzazione**, 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'esatta dimensione di 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
- 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 1/3

- 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

Rappresentazione numeri 2/3

- 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
- 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 3/3

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

00000000		00000001		00000010	
00000000	0	00000000	256	00000000	512
00000000		00000001		00000010	
00000001	1	00000001	257	00000001	513
00000000		00000001		00000010	
00000010	2	00000010	258	00000010	514
00000000		00000001			
00000011	3	00000011	259	⋮	
...		...			
00000000		00000001			
11111111	255	11111111	511		

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 o 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 1/2

- 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
- Quindi per far eseguire un programma ad un processore, basta memorizzare da qualche parte nella memoria le configurazioni di bit relative a ciascuna istruzione da eseguire, e 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

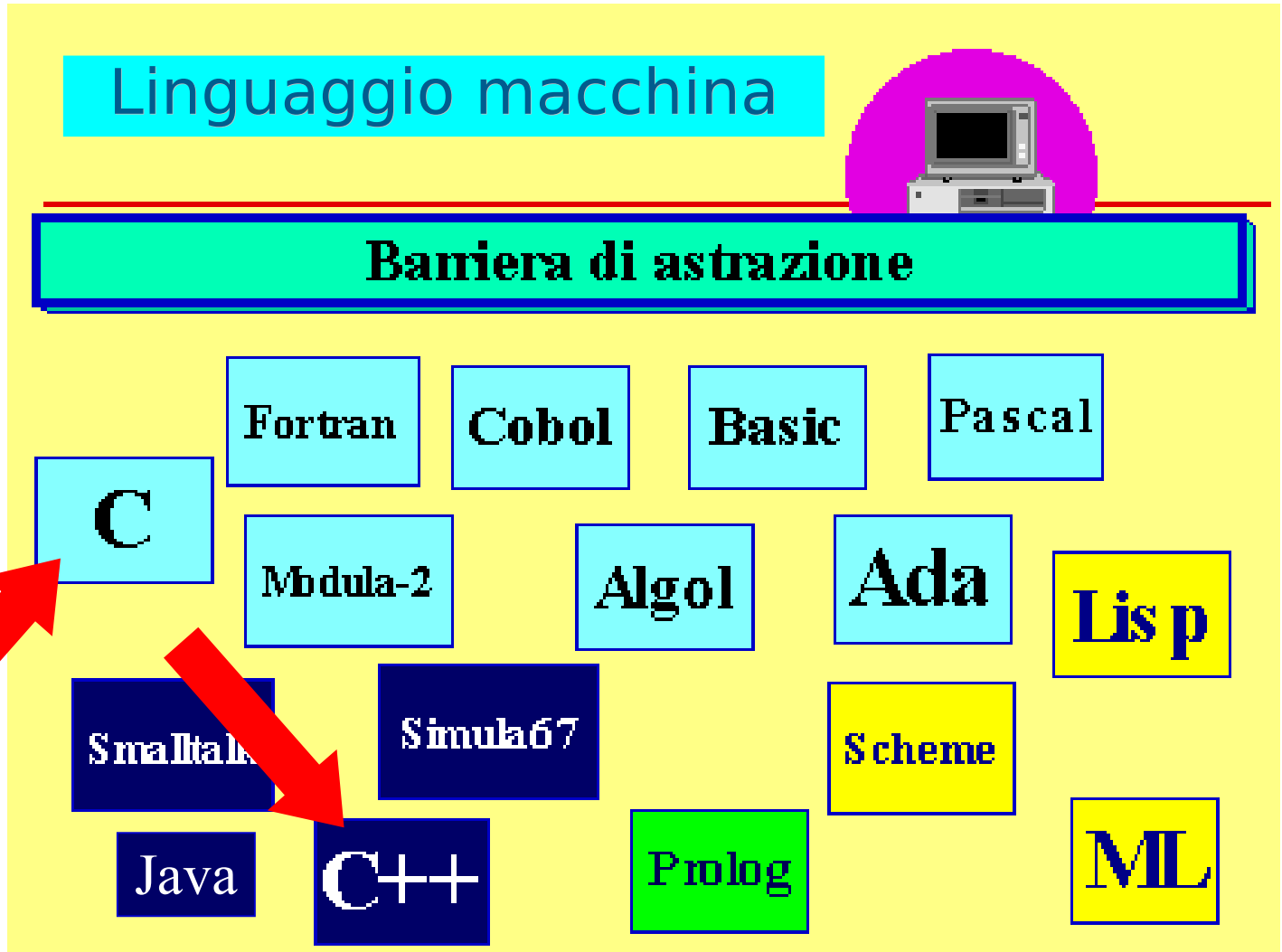
Linguaggio macchina 2/2

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

- 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.
  - Dato un problema da risolvere, questi dati ed operazioni più complessi permettono 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 (processo) il contenitore 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**

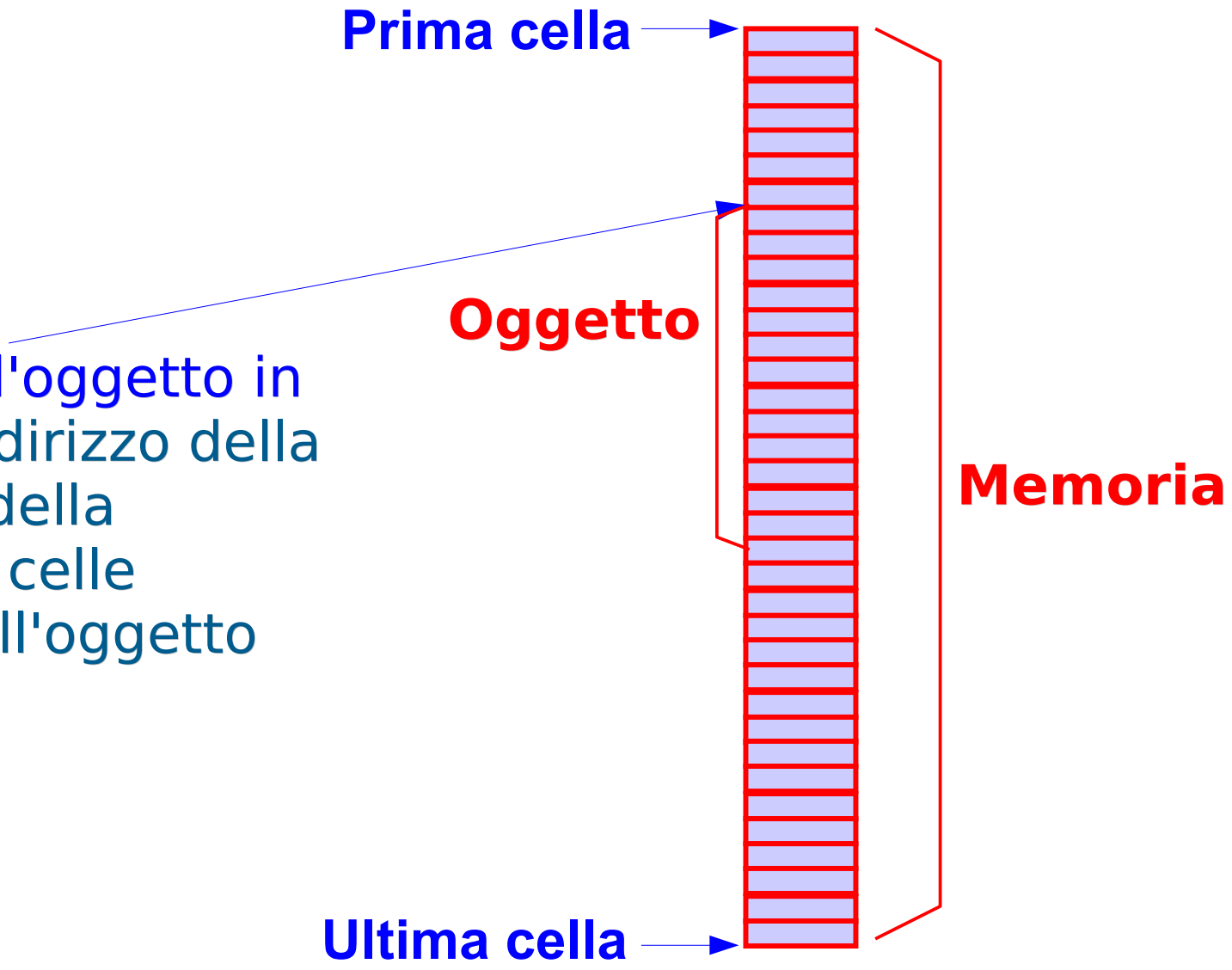
Dalle celle ai dati

- In C/C++ si possono memorizzare le informazioni in dati più complessi dei semplici numeri rappresentabili con una 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

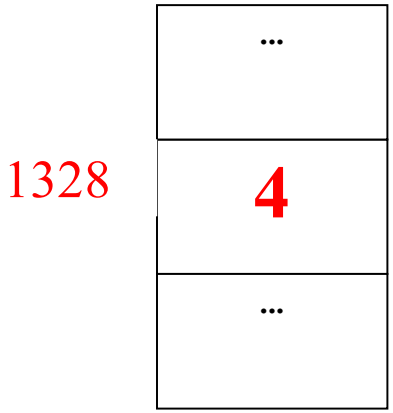
Oggetto in memoria, indirizzo



Indirizzo dell'oggetto in memoria: indirizzo della prima cella della sequenza di celle occupate dall'oggetto

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*
char	caratteri
int	sottoinsieme dei numeri interi
float	sottoinsieme dei numeri reali
double	sottoinsieme dei numeri reali con maggiore precisione rispetto al tipo **float**

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 degli interi Z è dato da
  $\{-\infty,\ldots,-2,-1,0,+1,+2,\ldots,+\infty\}$
- Ovvero il tipo **int** ha un insieme di valori limitato a priori:
  - *L'insieme esatto dei valori dipende dalla macchina*
  - Normalmente il tipo **int** è memorizzato 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]
  - ...

Astrazione

- Attraverso l'esempio del tipo **int**, ora si può capire più chiaramente il primo vantaggio dell'uso di oggetti appartenenti uno dei precedenti quattro tipi:
  - si astrae dalle singole celle di memoria: con il tipo **int** (e lo stesso accadrà con gli altri tipi di dato), non si vedono più le singole celle di memoria in cui sono memorizzati i numeri
  - si può ragionare e scrivere il programma direttamente in termini di numeri interi
    - si lavora cioè ad alto livello, senza preoccuparsi di come saranno realmente memorizzati e manipolati tali numeri a basso livello

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

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 i numeri interi, i numeri reali, le costanti carattere, le costanti stringa

Numeri

Numeri interi	`6`	`12`	`700`
Numeri reali	`24.0`	`.4`	`13.4e-1`

Costanti carattere

- Una costante letterale carattere è un'astrazione simbolica di un carattere, denotata racchiudendo il carattere tra apici
  Esempio: **`'A'  'c'  '6'`**

Mediante le costanti letterali carattere si possono però denotare anche:

- caratteri speciali: '\n', '\t', '\'', '\\', '\"'
- caratteri indicati tramite codice ASCII: '\nnn', '\0xhhh'
  (nnn = numero ottale, hhh = numero esadecimale)
  '\041' '\0' '\xfa'

Costanti stringa

- Sequenze di caratteri racchiuse tra doppi apici

```
"ciao"

"Hello\n"
```

"" (stringa nulla)

Variabili e costanti con nome

Variabile

- Una variabile è un oggetto, nel senso specificato nelle precedenti slide, il cui valore può variare nel tempo

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, vediamo 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 a e di
         // tipo int, inizializzata col
         // valore 5
```

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],
    … ;

Definizioni e dichiarazioni

- Come vedremo meglio in seguito, le definizioni sono casi particolari di **dichiarazioni**
  - Una dichiarazione è una istruzione in cui si introduce un nuovo identificatore
- Una definizione è una particolare dichiarazione 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

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**
- Svolgere l'esercizio di stampa di una variabile intera contenuto nella seconda esercitazione

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

Vedremo in dettaglio in seguito tutti i tipi di *espressioni* che si possono scrivere

Esercizi

- Svolgere i rimanenti esercizi della seconda esercitazione, fino all'esercizio di stampa di una costante escluso

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
- L'associazione identificatore-valore **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

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 (per poi essere 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

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>
using namespace std ;
main()
{
        <istruzione qualsiasi>
        <istruzione qualsiasi>
        ...
        <istruzione qualsiasi>
}
```

Direttive per il pre-processore

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
  - è la prima funzione che viene eseguita ovunque si trovi all'interno del file sorgente
  - quando termina l'esecuzione del *main()*, termina il 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