Introduzione al Linguaggio C/C++

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/

C/C++: linguaggio di alto livello

Il C/C++ è un linguaggio imperativo

C++

- Del linguaggio C++ vedremo solo il sottoinsieme procedurale
- NON vedremo la programmazione ad oggetti
- Sarà argomento di esami futuri

Caratteristiche del C/C++

- Linguaggio *sequenziale*, ***imperativo***, ***strutturato a blocchi***
- Usabile anche come linguaggio di sistema → adatto a software di base, sistemi operativi, compilatori, ecc.
- Portabile, efficiente, sintetico

- Basato su pochi concetti elementari (parte procedurale):
  - **espressione**
  - **dichiarazione / definizione**
  - **istruzione / blocco**
  - **funzione**
- Tuttavia, viene arricchito da un vasto insieme di librerie di funzioni (per operazioni matematiche, di input/output, su stringhe, ecc.)

Iniziamo ...

- Affronteremo lo studio del linguaggio incominciando dai seguenti concetti
  - Espressioni letterali
  - Dati
  - Memoria
  - Variabili
  - Costanti
  - Struttura (semplificata) di un programma

Espressioni letterali

Espressioni letterali

- Denotano valori costanti
- Spesso chiamate semplicemente **letterali** o **costanti senza nome**
- In C/C++ possono essere costanti carattere, costanti stringa, numeri interi, numeri reali

Numeri

Numeri interi	**6**	**12**	**700**
Numeri reali	**24.0**	**2.4e1**	**240.0e-1**

Costanti carattere

- Una ***costante* carattere** è un'astrazione simbolica di un carattere

Esempio: 'A' 'c' '6'

Anche:

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

'\041' '\0' 240.0E-1

Stringa di caratteri ≠ carattere [SI VEDRA' IN SEGUITO]

- "ciao" "Hello\n" "" *(stringa nulla)*

Dati

Tipi di dato

- In C/C++ (come in tutti i linguaggi di programmazione) a ciascun dato è associato anche il **TIPO**, ovvero la classe di valori che il dato può assumere nel corso dell'esecuzione del programma e gli operatori applicabili al valore in esso contenuto

Tipi di Dato Primitivi (“*base*”)

4 tipi di dato primitivi

char	(caratteri)
int	($\subset$ interi)
float	($\subset$ reali)
double	($\subset$ reali in doppia precisione)

Tipo “int”

- **Il tipo “int” è ben diverso dal tipo INTERO inteso in senso matematico** dove

  INTERO **Z** {-∞,…,-2,-1,0,+1,+2,…,+∞}

- Ovvero il tipo “int” ha un insieme di valori limitato a priori:
  - *L’insieme dei valori dipende dalla macchina*
  - Normalmente il tipo “int” è memorizzato in una PAROLA DI MACCHINA (***WORD***), che tipicamente è lunga (16), 32 o 64 bit
  - Se macchina a 16 bit: $[-2^{15}, 2^{15}-1]$ ovvero [-32768, +32767]
  - Se macchina a 32 bit: $[-2^{31}, 2^{31}-1]$ ovvero [-2147483648, +2147483647]
- Con i valori di tipo “int” è possibile effettuare solo un certo tipo di operazioni (ovvero, applicare solo gli operatori “int”)

Operatori “int” aritmetici

Al tipo **int** (e ai tipi ottenuti da questo mediante qualificazione) sono applicabili i seguenti operatori:

+	Addizione	
-	Sottrazione	
*	Moltiplicazione	
/	Divisione intera (≠divisione!)	Es., 10/3 = 3
%	Modulo (resto della divisione intera)	Es., 10%3 = 1 Es., 5%3 = 2
abs()	Ritorna il valore assoluto del numero Es., abs(-3) = 3	

Operatori “int” di confronto

== Operatore di confronto di uguaglianza
(simbolo = denota l’operazione di assegnamento!)

!= Operatore di confronto di diversità
> Operatore di confronto di maggiore stretto
< Operatore di confronto di minore stretto
>= Operatore di confronto di maggiore-uguale
<= Operatore di confronto di minore-uguale

Restituiscono: falso o vero

- Torneremo sull'argomento nelle prossime lezioni

Memoria

Memoria

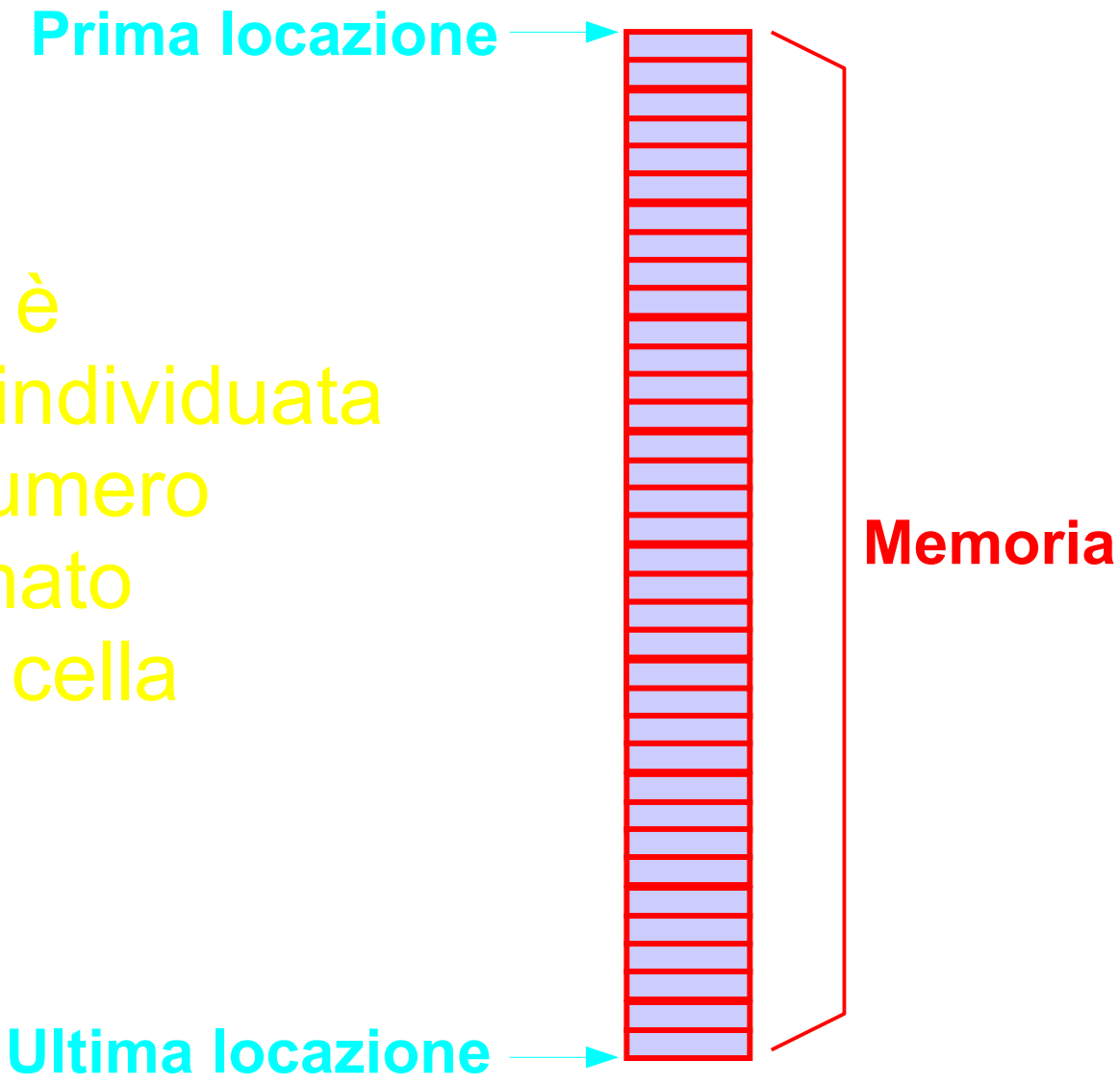
- Definiamo **memoria** (di un programma in esecuzione) il contenitore in cui sono memorizzati tutti i dati di un programma (e non solo i dati, come vedremo in seguito) durante la sua esecuzione
- Nei programmi C/C++ la memoria è vista come una sequenza contigua di **celle**

Cella

- Ciascuna cella è una sequenza di bit
- Esempio di contenuto di una cella di memoria: 01100101
- Tutte le celle hanno la stessa dimensione, ossia lo stesso numero di bit

Schema memoria

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



Celle e numeri

- I bit contenuti nella cella possono essere utilizzati per memorizzare numeri
- Senza entrare nei dettagli della notazione binaria, facciamo solo un esempio di come è ottenuto questo risultato

Rappresentazione di numeri 1/2

- Supponiamo che ogni cella contenga 8 bit
- Facciamo corrispondere un numero ad ogni combinazione di bit
- Es.:

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

Rappresentazione di numeri 2/2

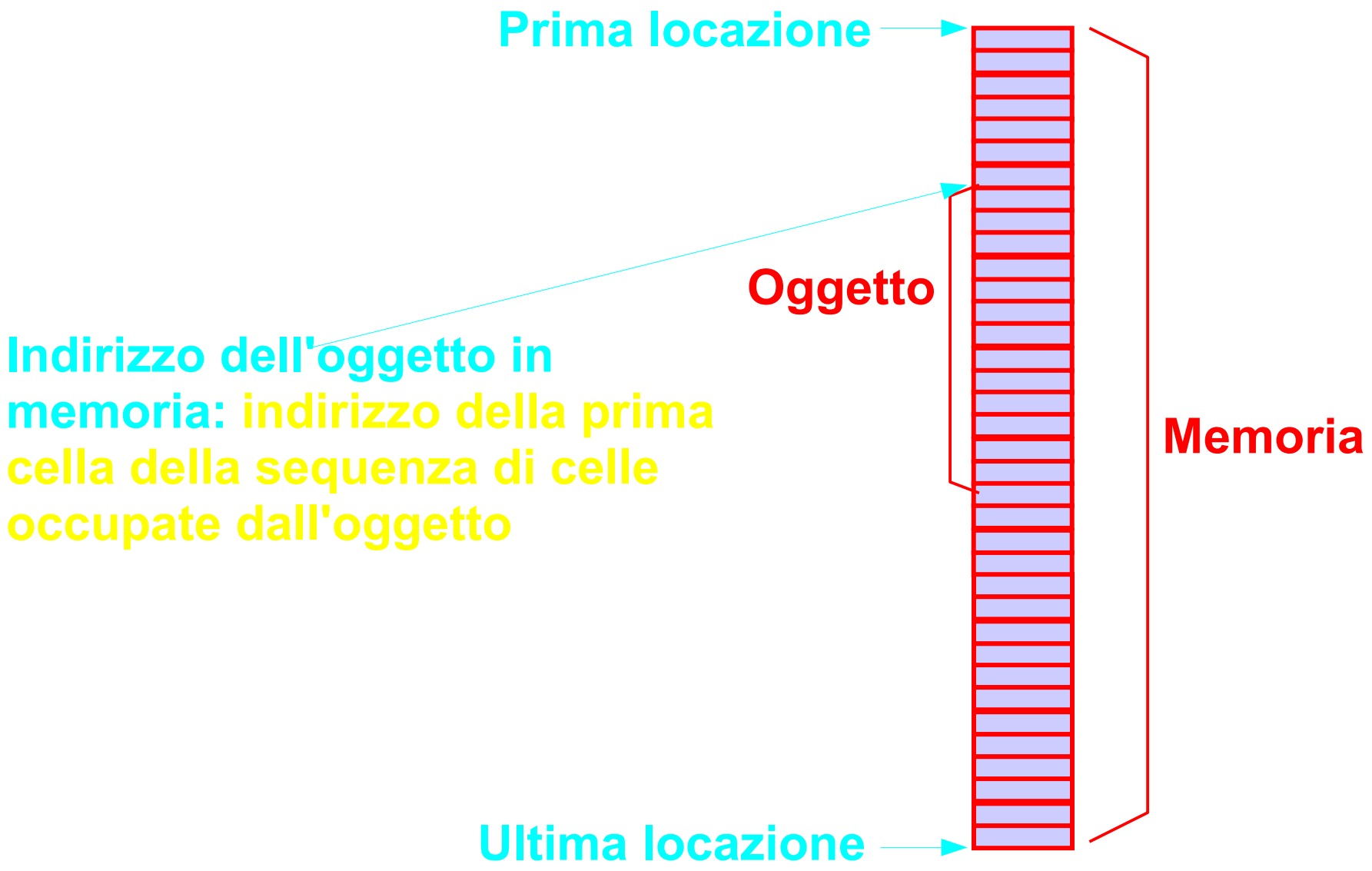
- L'esempio nella slide precedente illustra la rappresentazione in base 2 dei numeri naturalu da 0 a 255 su 8 bit
- La dimensione tipica della singola cella di memoria è effettivamente di 8 bit
- Più precisamente, lo standard del linguaggio C/C++ prevede che l'oggetto più piccolo indirizzabile in memoria abbia le dimensioni di un char, che a sua volta è tipicamente rappresentato su 8 bit

Variabili

Variabili

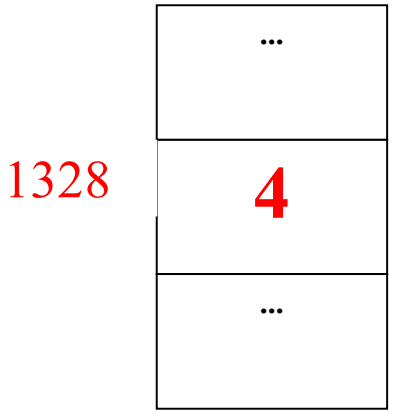
- Una *variabile* è un'astrazione di cella di memoria
  - E' un oggetto contenente un valore
  - Memorizzato in memoria mediante una sequenza di celle contigue
    - Così come si può rappresentare ogni numero naturale con una determinata configurazione di 8 bit, si può rappresentare ogni valore di un generico oggetto con una determinata configurazione di 8 bit per ciascuna cella della sequenza (di celle) utilizzata per rappresentare l'intero valore

Oggetto in memoria



Indirizzo e valore

- Un oggetto (di cui una variabile è un caso particolare) è quindi 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 esempio l'oggetto è di tipo numerico, ed il suo valore è attualmente 4

Definizione di variabili

- In C/C++ è necessario
  - Elencare tutte le *variabili* che saranno utilizzate nel programma. Nell'elenco bisogna attribuire ad ogni variabile
    - un **tipo**
    - un nome (**identificatore**)
    - eventualmente un valore iniziale (**inizializzazione**)
- Si ottiene questo risultato con le cosiddette **definizioni**

Nota sulla descrizione della sintassi

- Nella descrizione della sintassi del linguaggio utilizzeremo la notazione con parentesi quadre

  […] per denotare elementi opzionali, ossia parti che possono o meno comparire in un dato costrutto

Sintassi delle definizioni di variabili

- 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:

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

Visibilità di una variabile

- Una variabile può essere utilizzata solo a partire dal punto in cui viene definita
  - Si vedranno poi in dettaglio anche tutte le altre regole di visibilità, che stabiliscono in modo univoco in quali parti di un programma una certa variabile può essere utilizzata

Definizione di variabili di tipo *int*

Esempio

```
int a;
int b, c;
int k=5; // inizializzata a 5
```

Assegnamento valore

- Si può assegnare un nuovo valore ad una variabile mediante una **istruzione di assegnamento**

  nome_variabile = espressione ;

- Esempio:

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

- Ci torneremo sopra in lezioni successive

Definizione di oggetti costanti

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

Esempi

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

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 **indirizzo-valore**
  - Uno stesso identificatore può denotare valori differenti in momenti diversi dell'esecuzione del programma

Dichiarazioni e definizioni

- Le dichiarazioni sono costrutti del linguaggio che introducono nuove entità (tipi, oggetti, funzioni, ...)
- Se una dichiarazione comporta, da parte del compilatore, l'associazione di locazioni di memoria o azioni eseguibili all'entità introdotta nella dichiarazione, si denota tale dichiarazione come una **definizione**
  - Una definizione è quindi un caso particolare di dichiarazione
  - Finora abbiamo visto solo definizioni
- Esempio:
  int N;

L'esecuzione della **definizione** provoca l'allocazione di uno spazio in memoria pari a quello necessario a contenere un dato del tipo specificato

Struttura (semplificata) programmi

Pre-processore

- Prima della compilazione vera e propria, il programma può essere 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)
- Vedremo in seguito cosa fa il pre-processore in dettaglio, quello che ci basta sapere per ora è che il pre-processore viene pilotato mediante le cosiddette **direttive** inserite nel programma originario

Struttura dei programmi C – Es. 1

Esempio

```
#include <stdio.h>

main()
   { dichiarazioni;
     istruzioni;
   }
```

Direttive al pre-processore

Parte dichiarativa

Parte esecutiva

Ciascuna **dichiarazione** è separata dall'altra dal **;**

Ciascuna **istruzione** è separata dall'altra dal **;**

Non vi sono altri simboli speciali per separare parte dichiarativa da parte esecutiva

Struttura dei programmi C++ – Es. 1

Esempio

```
#include <iostream>

main()
   {
    dichiarazione o istruzione;
    dichiarazione o istruzione;
    ...
  }
```

Direttive al pre-processore

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 (stessa cosa vale nel caso di programma su più file)
  - quando termina l'esecuzione del **main()**, termina il programma
- In C la funzione `main()` contiene due sezioni, racchiuse entro il ***blocco*** denotato da parentesi `{ }`
  - **Parte dichiarativa**
  - **Parte esecutiva**

Ordine di esecuzione

- In che ordine vengono eseguite le istruzioni?
- Si definisce **sequenza** o **concatenazione** una sequenza di istruzioni/dichiarazioni 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 ;
N = 3 ;
cout<<N<<endl;
```

Parole chiave della lezione

- **Linguaggio C/C++**
- **Tipi di dato**
- **Tipo int**
  - Range di valori
  - Operatori
- **Dichiarazioni e definizioni**
- **Costanti e Variabili**
- **Struttura programma in linguaggio C/C++**
- **Funzione main()**