# Parte 3

# Costanti e Variabili

## Identificatori

- Un identificatore è un nome che viene associato a diverse **entità** (costanti, tipi, variabili, funzioni, ecc.) e serve ad identificare la particolare entità
- Gli identificatori in C devono soddisfare determinati vincoli
  - Un identificatore deve iniziare con un carattere o con il simbolo '_' (underline), e può contenere solo lettere, numeri e il simbolo underline
  - int a, _a, a2; // identificatore corretto
  - char 2b; //identificatore non corretto

# Costanti e variabili

- Una costante è un identificatore a cui è associato un valore che resterà fisso per tutta la durata dell'esecuzione del programma
  - la costante mantiene inalterato il suo valore
- Una variabile è invece un identificatore il cui valore può variare più volte durante l'esecuzione del programma



# **Costanti** (*numeriche*)

- Una **costante** numerica è un'astrazione simbolica di un valore

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

**(notazione esponenziale: eX equivale a 10 elevato a X, con segno + facoltativo, segno – obbligatorio)**

# Costanti (*caratteri*)

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

  **Esempio:** 'A' 'c' '6'

Anche caratteri speciali: '\n', '\t', '\'', '\\', '\"'

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

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



# Dichiarazione di costanti

- Si utilizza la parola chiave **const**
  const tipo id = valore;
- dopo la parola chiave const si indicherà il tipo della costante, l'identificatore e quindi il valore

- **Esempi**
  - const int a = 1;
  - const char b = 'a';

- **NOTA: ';' per terminare una istruzione!**

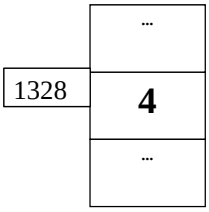
# Variabili

- **Una *variabile* è un'astrazione della cella di memoria**
- **E' un oggetto (in memoria) contenente un valore**
- Formalmente, una **variabile** è un **simbolo** associato a un *indirizzo fisico* (**L-VALUE – left value**)

| *simbolo* | *indirizzo* |
|---|---|
| **x** | **1328** |

che **denota un valore** (**R-VALUE**)

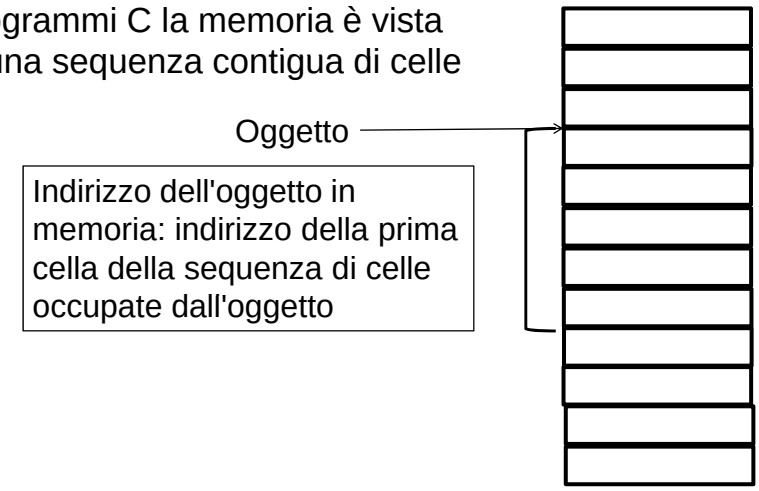
- Perciò, l' R-VALUE di x è attualmente 4:






# Oggetto in memoria

**Memoria**
Nei programmi C la memoria è vista come una sequenza contigua di celle




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

# Cella di memoria

- Ciascuna cella è una sequenza di bit
- La dimensione tipica della singola cella di memoria è effettivamente di 8 bit (1 byte)
- Tutte le celle hanno la stessa dimensione, ossia lo stesso numero di bit
- Esempio di contenuto di una cella di memoria: 01100101
- Ciascuna cella è univocamente individuata mediante un **numero naturale**, chiamato **indirizzo della cella**



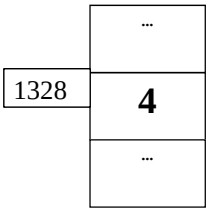
# Celle e dati

- I bit contenuti nella cella possono essere utilizzati per memorizzare numeri
- Senza entrare nei dettagli della notazione binaria, chiediamoci soltanto quanti bit/byte ci servono per rappresentare un numero naturale piuttosto che un carattere…

Lo standard del linguaggio 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

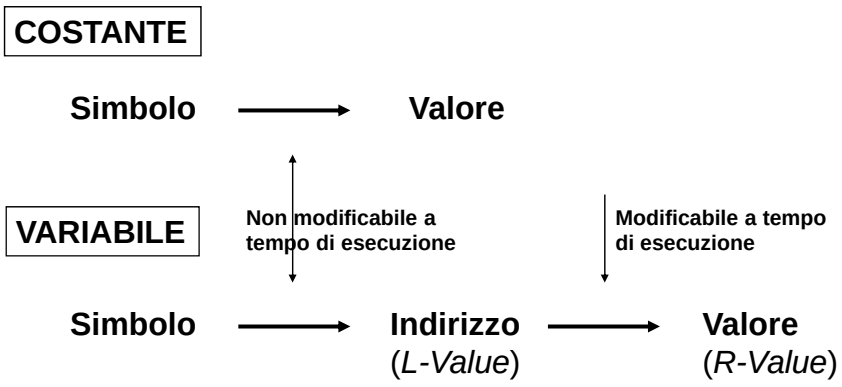
# Indirizzo e valore

- Un oggetto (di cui una variabile è un caso particolare) è quindi caratterizzato da
  - un *indirizzo (a*d esempio 1328, il che vuol dire che l'oggetto si trova in memoria a partire dalla cella di indirizzo 1328)
  - un *valore (i*n questo esempio l'oggetto è di tipo numerico, ed il suo valore è attualmente 4)






# Costanti e Variabili



# Costanti e Variabili

- Una ***costante*** è un'astrazione simbolica di un valore
- L'associazione simbolo-valore non cambia mai durante l'esecuzione
- Una ***variabile*** è un **simbolo** associato a un *indirizzo fisico* (**L-VALUE**) che contiene un valore (**R-VALUE**)
- L'associazione **simbolo-indirizzo** non cambia mai durante l'esecuzione, ma può cambiare l'associazione **indirizzo-valore**
- Pertanto, nel caso di variabile, ad uno stesso simbolo possono corrispondere valori differenti in diversi momenti dell'esecuzione del programma

Attenzione
- **R-VALUE** può cambiare nel corso dell'esecuzione
- **L-VALUE** è fissato (e non cambia durante l'esecuzione)



# Dichiarazione di variabili

**Scopo:**
- Elencare tutte le *variabili* che saranno utilizzate nella parte esecutiva
- Attribuire ad ogni variabile un **tipo, un nome (identificatore)** ed eventualmente **un valore iniziale**
- E' possibile raggruppare le dichiarazioni di più variabili dello stesso tipo in una lista separata da **,**

Esempio
```
int a;
float x, y;
char c, w, z;
```
Cercare di utilizzare sempre nomi significativi per le variabili!

## Nota sulla 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 dichiarazioni di variabili

- Sintassi della definizione di una variabile:
  *nome_tipo nome_variabile [= valore_iniziale] ;*
  → ***l'inizializzazione è opzionale***

- E' possibile raggruppare le dichiarazioni di più variabili dello stesso tipo in una lista separata da **,**
- Forma generale:
  *nome_tipo nome_variabile1 [=valore_iniziale],*
  *nome_variabile2 [= valore_iniziale] ;*

## Esempi di dichiarazione

**Variabili – esempi**

- `int b, c;`
- `int k=5;` **`INIZIALIZZAZIONE DI k!`**

**Costanti – esempi**

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

**→ Obbligo di inizializzazione!!**



# Tipi di dato “primitivi” (Il tipo “int”)

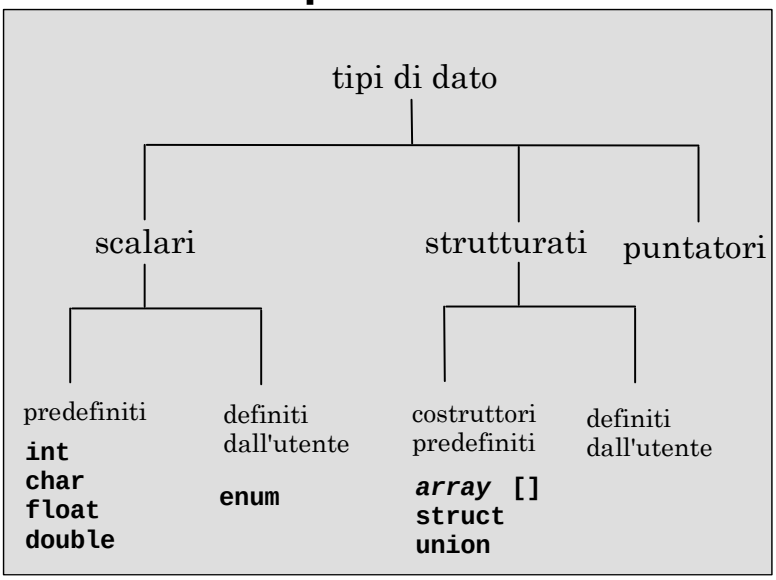
# Tipi di dato

- In C (come in tutti i linguaggi di programmazione) a ciascuna variabile e costante è associato anche il **TIPO**, ovvero la classe di valori che la costante o variabile può assumere nel corso dell'esecuzione del programma (e quindi gli operatori applicabili al valore in essa contenuto)
- L'associazione di un nome (di costante o di variabile) ad un tipo di dato non cambia mai durante l'esecuzione del programma.

**Quali sono i tipi di dato ammissibili in C?**



# Tipi di Dato



# 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** $\{-\infty,\ldots,-2,-1,0,+1,+2,\ldots,+\infty\}$

- Ovvero il tipo “int” ha un insieme di valori e di operazioni definibili su di essi limitato a priori (**range di valori**):
  - *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]

# Operatori “int” (aritmetici)

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

+ Addizione
- Sottrazione
* Moltiplicazione
/ Divisione intera (diversa!) 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” (relazionali)

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

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

**Restituiscono vero o falso**

c2

# Operatori “int” (*cont*.)

| | |
|---|---|
| abs(n) | Valore assoluto di n (previa inclusione dell’header file <***math.h***>) |
| n++ | Successivo di n (*notazione postfissa*) |
| ++n | Successivo di n (*notazione prefissa*) |
| n-- | Precedente di n (*notazione postfissa*) |
| --n | Precedente di n (*notazione prefissa*) |



# Assegnamento

## Diapositiva 25

**c2** Priorità

c = ++a + b

c = a++ + b

claudia; 03/10/2005

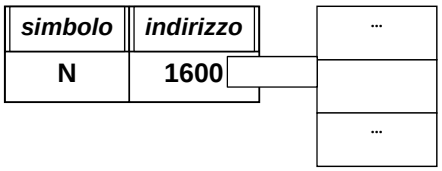
# Istruzione di assegnamento

- Denotata mediante il simbolo **=**

  (l'operatore relazionale di uguaglianza è denotata con il simbolo **==** )
- Viene utilizzata per assegnare ad una variabile (non ad una costante!) il valore di un'espressione

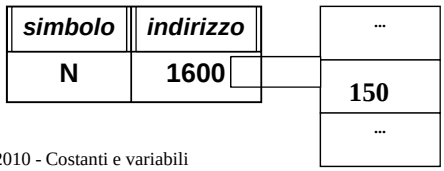
Esempio

```
int  N;
```




L'esecuzione di una **dichiarazione** provoca l'allocazione di uno spazio in memoria equivalente a quello necessario a contenere un dato del tipo specificato

```
N = 150;
```




L'esecuzione di un **assegnamento** provoca l'inserimento nello spazio relativo alla variabile del valore indicato a destra del simbolo **=**



# Istruzione di assegnamento (*cont.*)

- L'esecuzione di un'istruzione di assegnamento comporta innanzitutto la valutazione di tutta l'espressione a destra dell'assegnamento.

  Es.,
```
c = 2;
d = (c+5)/3 - c;
d = (d+c)/2;
```

- Dopodiché, si inserisce il valore risultante nella locazione di memoria relativa alla variabile (posta a sinistra dell'assegnamento)

- Il primo assegnamento di un valore ad una variabile dichiarata viene detto **inizializzazione**.

  **In C, l'inizializzazione si può effettuare anche al momento della dichiarazione.**

  Es.,
```
int a, b=56;
```

# Parole chiave della lezione

- **Linguaggio C**
  - Linguaggio imperativo
  - Componenti: caratteri, identificatori, parole riservate, commenti
- **Costanti e Variabili**
- **Tipi di dato**
- **Tipo int**
  - Range di valori
  - Operatori
- **Assegnamento**