



2. Introduzione all'informatica e alla programmazione

93



2.1 Introduzione ai calcolatori

94



Che cosa è un computer?

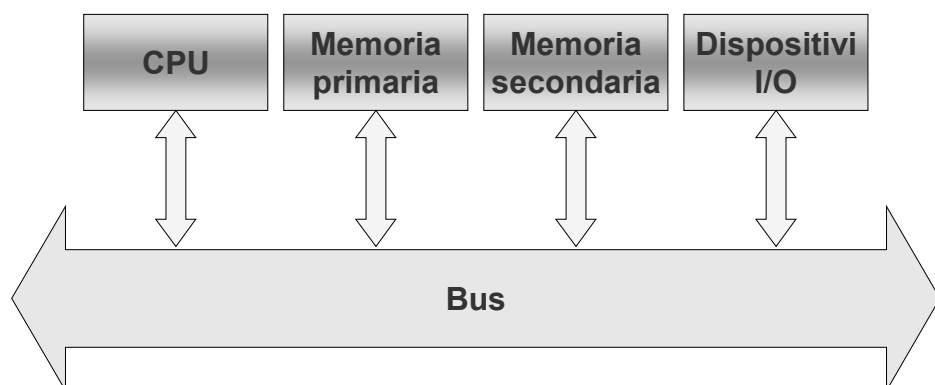
A prescindere dalle dimensioni e dal luogo in cui si trova, può essere definito come un elaboratore elettronico digitale

- **elaboratore** in grado di immagazzinare ed elaborare dati in base ad una serie di istruzioni (il programma)
- **elettronico** evidentemente utilizza componenti elettronici.. :-)
- **digitale** elabora informazioni convertendole in segnali digitali basati sul sistema binario

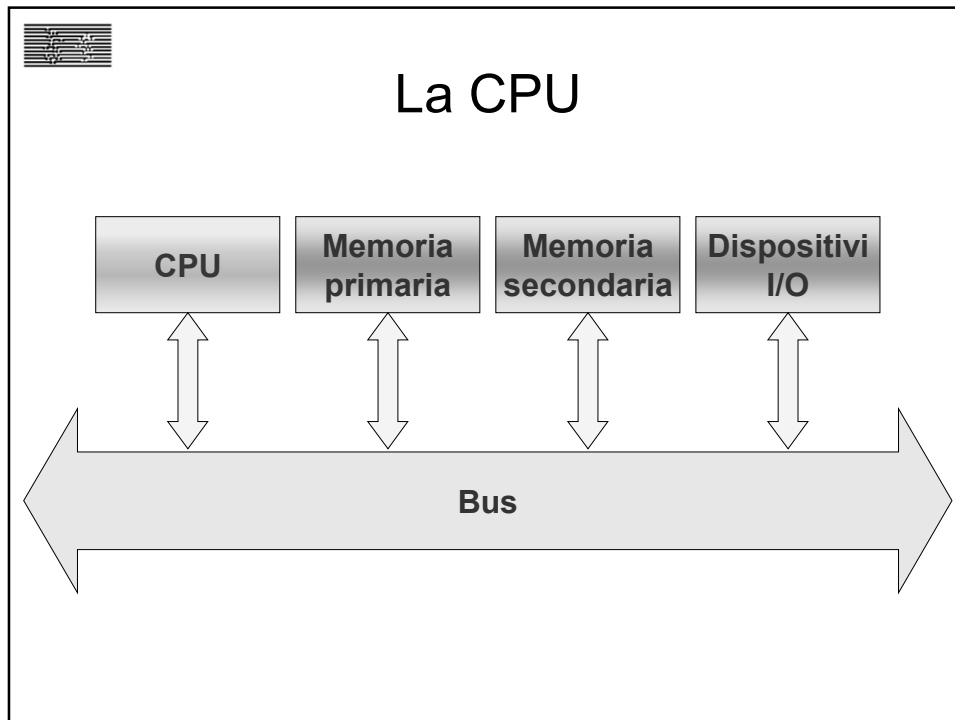
95



Struttura di un computer



96

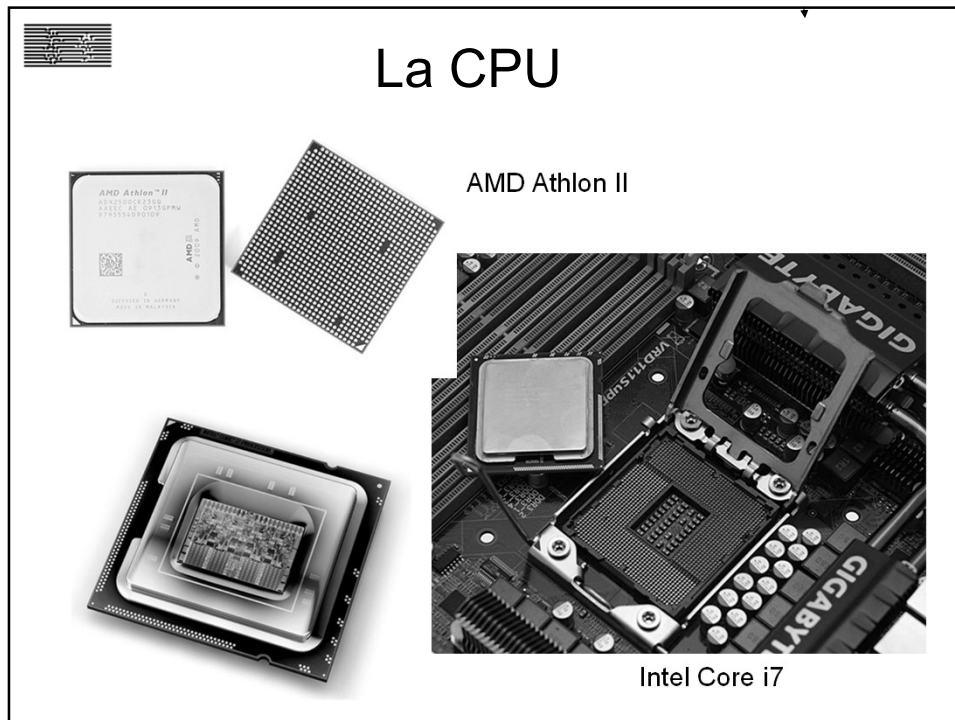


97

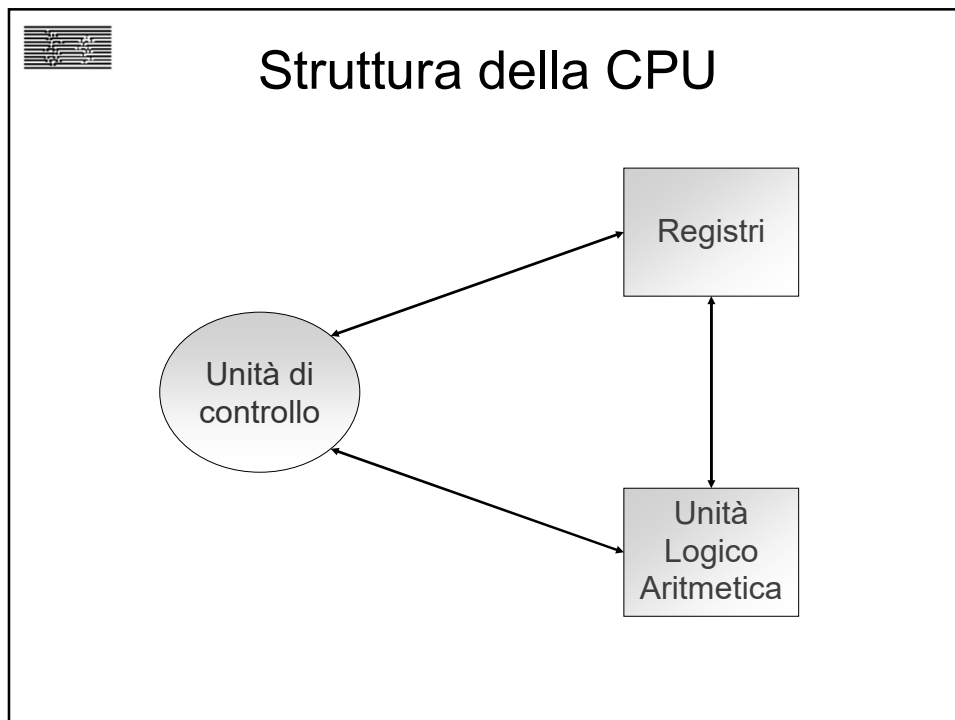
The diagram, titled "La CPU", provides a textual description of the CPU's function and examples. It includes a list of four bullet points:

- L'unità centrale di elaborazione (Central Processing Unit) è un insieme di circuiti, detto microprocessore, che controlla l'attività del computer.
- Il clock di una CPU, misurato in MHz o in GHz (ad esempio 2 GHz), è la frequenza con cui vengono eseguite le istruzioni elementari.
- Compito della CPU è quello di leggere istruzioni e dati dalla memoria e di decodificare ed eseguire le istruzioni; il risultato dell'esecuzione di una istruzione dipende dal dato su cui opera e dallo stato interno della CPU stessa.
- Esempi di CPU: Intel Core i3, i5, i7, AMD Athlon II, ...

98



99



100



Struttura della CPU

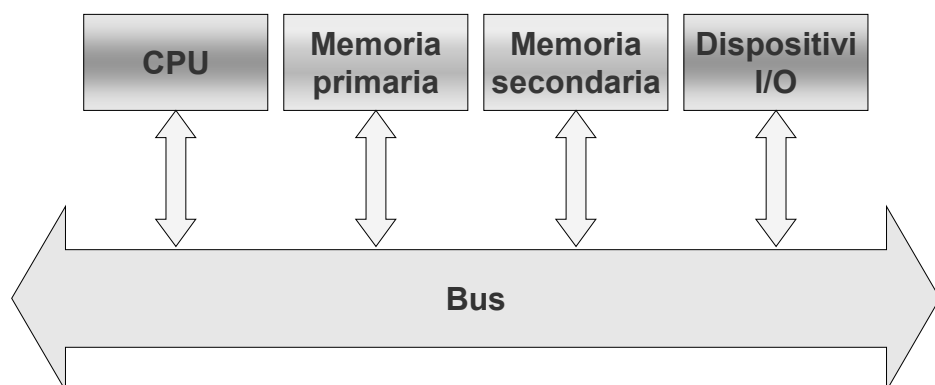
L'unità centrale di elaborazione è costituita da:

- un insieme di registri che sono degli spazi di memorizzazione accessibili ad una velocità superiore di quella della memoria principale;
- una unità logico aritmetica (ALU) che esegue operazioni aritmetiche, logiche e confronti sui dati della memoria principale o dei registri;
- una unità di controllo che esegue le istruzioni.

101



La memoria



102



Memorie: una gerarchia

La memoria di un computer è organizzata in maniera gerarchica, in base alla velocità di accesso. Memorie più veloci sono anche più costose.

- Memoria cache:
molto veloce, piccole dimensioni, costosa
- Memoria principale (primaria):
veloce, medie dimensioni, accessibile
- Memoria secondaria:
lenta, notevoli dimensioni, economica

103



La memoria cache

- Memoria piccola, ma molto veloce, intermedia tra memoria primaria e CPU
- I dati di uso più frequente sono mantenuti nella memoria cache per minimizzare i trasferimenti tra memoria primaria e CPU
- Ci sono due tipi di memorie cache:
 - Cache interna al processore
 - Cache esterna al processore

104



La memoria principale

- La memoria principale è costituita da memoria di tipo RAM (Random Access Memory), che consente un rapido accesso ai suoi dati.
- Una RAM consente l'accesso ai dati in essa contenuti senza dover rispettare un determinato ordine e con lo stesso tempo di accesso.
- La memoria principale è una memoria volatile: allo spegnimento del computer i dati in essa contenuti vanno persi.

105



La memoria principale

- La memoria principale contiene il codice del programma in esecuzione e i dati che questo usa.
- Codice e dati sono contenuti in un insieme di celle identificate da un indirizzo univoco, che ne consente l'individuazione.
- Dato un indirizzo di memoria, le operazioni effettuabili sono:
 - lettura del contenuto della cella corrispondente e suo trasferimento verso il bus
 - scrittura del contenuto proveniente dal bus nella corrispondente cella di memoria

106



Indirizzamento di memoria

- Ogni cella della memoria è identificata da un numero, chiamato indirizzo di memoria.
- In questo modo ogni dato all'interno della memoria presenta una precisa collocazione, ed è pertanto possibile recuperarlo, o decidere dove memorizzarlo, indicandone l'indirizzo.
- Lo spazio di indirizzamento indica il massimo indirizzo di memoria rappresentabile.

Indirizzi	Memoria
.. →	5
.. →	34
34342 →	234
34343 →	43
34344 →	32
34345 →	234
34346 →	112
34347 →	22
34348 →	4
34349 →	22
.. →	54
.. →	125

107



Bit

- All'interno della memoria e, in generale, del calcolatore l'informazione è rappresentata come sequenze di bit.
- Il bit è l'unità elementare dell'informazione trattata da un elaboratore.
- La rappresentazione logica del bit è costituita dai soli valori {0, 1}.
- L'adozione della rappresentazione binaria si deve alla semplicità di realizzare fisicamente un elemento con due stati anziché un numero superiore, ed anche alla corrispondenza diretta con i valori logici vero e falso.

108



Byte

- Sequenze di bit sono raggruppate in entità più vaste, che contengono generalmente un numero di bit pari ad una potenza binaria, pari cioè a 2^n ;
- Il più noto di tali raggruppamenti è il byte (chiamato anche ottetto), corrispondente ad 8 bit, che costituisce l'unità di misura più utilizzata in campo informatico.
- Ad esempio le dimensioni della memoria si misurano in multipli del byte.
- Una cella di memoria contiene una sequenza di byte, ad esempio quattro, detta parola (word).

109



Multipli del byte

Multipli del byte					
Prefissi SI			Prefissi binari		
Nome	Simbolo	Multiplo	Nome	Simbolo	Multiplo
kilobyte	kB	10^3	kibibyte	KiB	2^{10}
megabyte	MB	10^6	mebibyte	MiB	2^{20}
gigabyte	GB	10^9	gibibyte	GiB	2^{30}
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}
exabyte	EB	10^{18}	exbibyte	EiB	2^{60}
zettabyte	ZB	10^{21}	zebibyte	ZiB	2^{70}
yottabyte	YB	10^{24}	yobibyte	YiB	2^{80}

110



Tempi di accesso

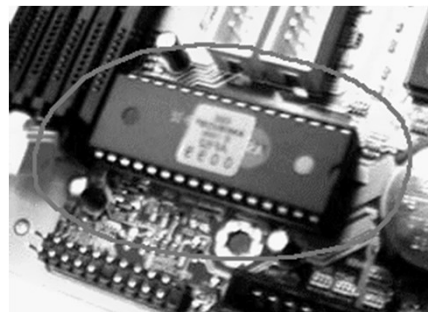
- Un'importante caratteristica della memoria è il suo tempo di accesso (tempo necessario per leggere o scrivere una parola)
- Le memorie principali dei computer attuali sono molto veloci e i loro tempi di accesso sono di pochi *nanosecondi* ($= 10^{-9}$ sec, un miliardesimo di secondo)

111



Memorie ROM

- La ROM (Read Only Memory) è un tipo di memoria che consente solamente l'operazione di lettura dei dati, poiché il suo contenuto viene permanentemente definito in fase di costruzione della memoria stessa.
- È di tipo non volatile quindi il suo contenuto non viene perso se viene a mancare l'alimentazione.



112



BIOS

- Tutti i computer contengono un chip di memoria elettronica non volatile e non modificabile (ROM).
- Contiene un programma (detto Basic Input Output System, BIOS) per l'avviamento del computer e per altre operazioni fondamentali.

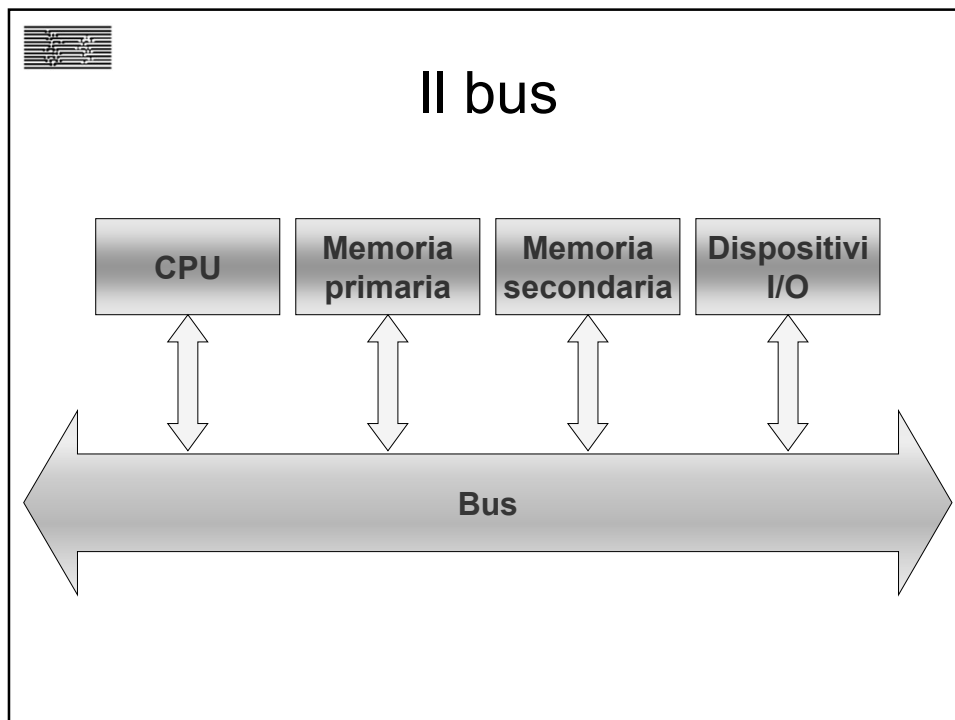
113




La memoria secondaria

- La memoria secondaria (o memoria di massa) è costituita da uno o più dispositivi capaci di contenere molti più dati della memoria principale, a discapito della velocità.
- I dati contenuti nella memoria secondaria sono conservati in maniera permanente anche quando il computer viene spento.
- Esempi di dispositivi di memoria secondaria sono: le unità di memoria a stato solido (SSD), i dischi rigidi (hard disk), i CD e i DVD.

114



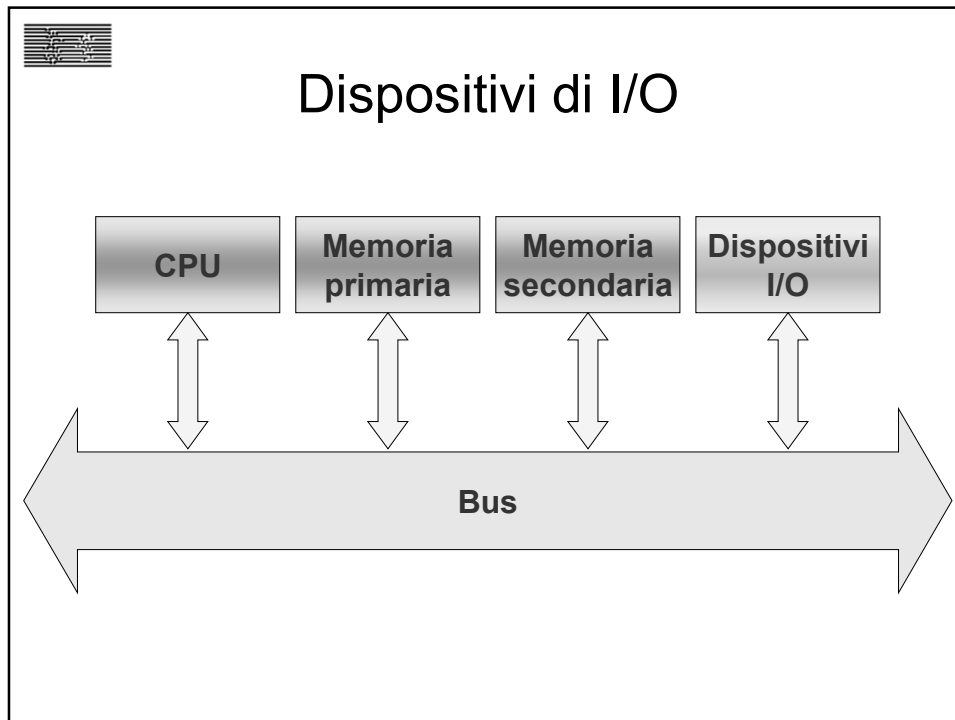
115




Il bus

- Un bus è un canale attraverso il quale due o più dispositivi si scambiano informazioni.
- Senza questo sistema, ogni componente del computer dovrebbe possedere un collegamento fisico distinto verso ognuno degli altri dispositivi (cosiddetti *collegamenti punto-punto*).
- A differenza dei collegamenti punto-punto, il bus consente una maggiore configurabilità.
- Il clock di un bus esprime la velocità con cui vengono scambiati i dati.

116



117



Dispositivi di I/O

- Sono tutti quegli apparati che servono per il trasferimento di dati fra il computer e il mondo esterno
- Esempi di dispositivi di input (ingresso):
tastiera, mouse, microfono, webcam, scanner, ...
- Esempi di dispositivi di output (uscita):
monitor, stampante, altoparlanti, ...
- Esempi di dispositivi sia di input che di output:
scheda di rete, ...

118



Dispositivi di I/O

- Molti dispositivi di I/O sono collegati al computer dall'esterno attraverso le cosiddette porte di I/O.
- Tipiche porte di I/O sono la porta USB, la porta di rete, ecc.
- Talvolta alcuni possono essere inseriti all'interno del computer stesso come schede.



scheda video



stampante



mouse



tastiera

119



Altri dispositivi di I/O



Head-Mounted Display (HMD)



Data gloves



Computer Aided Virtual Environment (CAVE)



120

 Oltre i calcolatori tradizionali...



Simulatori



Applicazioni mediche



Applicazioni per musei





121

 Oltre i calcolatori tradizionali...



122



Il sistema operativo

- Il sistema operativo è un programma che agisce da intermediario fra l'utente e l'hardware dell'elaboratore.
- E' responsabile del diretto controllo e della gestione efficiente dell'hardware dell'elaboratore.
- Fornisce un ambiente nel quale l'utente è in grado di eseguire i programmi.

123



2.2 Introduzione alla programmazione in C++

124



Software

- Informalmente un programma può essere definito come una serie di operazioni elementari, eseguite in sequenza, che trasformano un insieme di dati di ingresso (input) in un insieme di dati di uscita (output).
- L'insieme dei programmi e dei dati presenti su di un calcolatore ne costituiscono il software.

125



Programmazione

- La programmazione è l'attività di progettare e sviluppare programmi per un calcolatore.
- Informalmente, è come se il programmatore "insegnasse" al calcolatore come svolgere un determinato compito.
- Scopo della scrittura di un programma è la risoluzione automatica di un problema.
- Il risultato della programmazione è un programma scritto in un linguaggio di programmazione.

126



Progetto software

- Di solito comporta i seguenti passi:
 - Formulare il problema (*specificazione dei requisiti*)
 - Capire il problema e scomporlo in parti gestibili (*analisi*)
 - Progettare una soluzione (*algoritmo*)
 - Implementare la soluzione (*scrittura del codice*)
 - Testare la soluzione e correggere eventuali errori (*verifica, testing e debugging*)
 - Tenere aggiornato il programma (*manutenzione*)

127



Algoritmo

- Un algoritmo per la risoluzione di un problema è una sequenza di passi discreti
 - di lunghezza finita
 - deterministici (dopo ogni passo si sa precisamente qual è il prossimo)
 - ripetibiliche producono la soluzione del problema.
- Esempio di algoritmi: procedimenti per calcolare il risultato di espressioni aritmetiche o algebriche.

128



Linguaggi di programmazione

- Per scrivere il programma che realizza un algoritmo, il programmatore deve usare un linguaggio di programmazione.
- Un linguaggio di programmazione è costituito, informalmente, da parole e simboli e da un insieme di regole per combinarli.
- A differenza dei linguaggi naturali, le regole dei linguaggi di programmazione (linguaggi formali) sono rigide, per garantire l'assenza di ambiguità.

129



Linguaggi di programmazione

- Se il programma rispetta le regole del linguaggio, può essere tradotto (compilato) in linguaggio macchina, e quindi eseguito sul calcolatore.
- Il compilatore è il programma che traduce una serie di istruzioni scritte in un linguaggio di programmazione (codice sorgente) in linguaggio macchina (codice oggetto).
- I linguaggi che seguono questo approccio si dicono spesso linguaggi compilati.

130



Linguaggi di programmazione

- Altro approccio: linguaggi interpretati.
- Un interprete è un programma che esegue altri programmi.
- Un linguaggio interpretato è un linguaggio di programmazione le cui istruzioni vengono eseguite da un interprete.
- L'esecuzione di programmi scritti in un linguaggio interpretato è di solito meno efficiente rispetto ai linguaggi compilati.
- Esistono anche approcci ibridi.

131



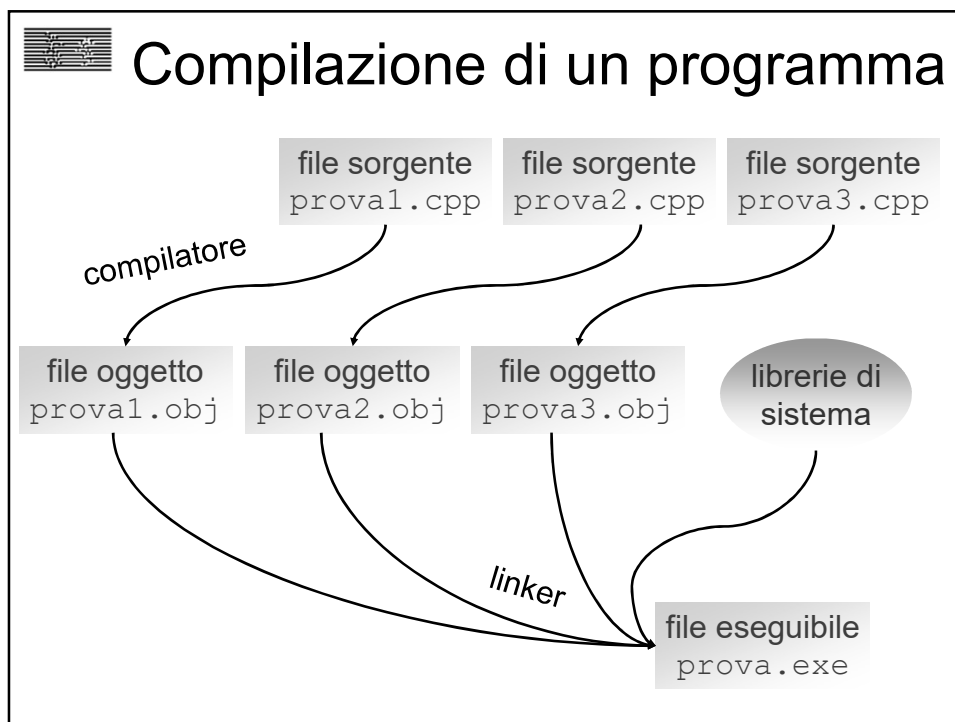
Scrittura di un programma

- Scrittura del programma in un file chiamato file sorgente.
- Creazione e modifica tramite un opportuno strumento chiamato editor (ad esempio wordpad di Windows o vi di Linux).
- Nel caso di programmazione in C/C++ è consigliabile dare al nome del file sorgente l'estensione **cpp** (**nomefile.cpp**).

132



133



134



Il primo programma C++

- Scrivere un programma che stampi "Ciao!" a video:

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

int main()
{
    cout << "Ciao!";

    return 0;
}
```

135



Il primo programma C++

- Scrivere un programma che stampi "Ciao!" a video:

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

int main()
{
    cout << "Ciao!";

    return 0;
}
```

Annotations:

- Librerie**: Points to `#include <iostream>`
- Funzione principale**: Points to `int main()`
- Costante stringa**: Points to `"Ciao!"`
- Istruzione per la stampa**: Points to `cout <<`
- Valore di ritorno**: Points to `return 0;`

136



Il secondo programma C++

- Scrivere un programma che calcoli $5 + 3$ e stampi a video il risultato:

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

int main() {
    int a = 5;
    int b = 3;
    int somma = a + b;
    cout << "La somma di " << a
          << " e " << b << " vale "
          << somma << endl;
    return 0; }
```

137



Il secondo programma C++

- Scrivere un programma che calcoli $5 + 3$ e stampi a video il risultato:

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

int main() {
    int a = 5;
    int b = 3;
    int somma = a + b;
    cout << "La somma di " << a
          << " e " << b << " vale "
          << somma << endl;
    return 0; }
```

Diagram annotations:

- Arrow from `int` to "Variabile di tipo intero"
- Arrow from `5` to "Valore (costante numerica)"
- Arrow from `=` to "assegnato alla variabile"
- Arrow from `+` to "Operatori"

138



Il terzo programma C++

- Scrivere un programma che sommi due numeri interi immessi dall'utente:

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

int main() {
    int a; int b;
    cout << "Immettere due numeri:";
    cin >> a >> b;
    int somma = a + b;
    cout << "La somma di " << a
         << " e " << b << " vale "
         << somma << endl;
    return 0; }
```

139



Il terzo programma C++

- Scrivere un programma che sommi due numeri interi immessi dall'utente:

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

int main() {
    int a; int b;
    cout << "Immettere due numeri:";
    cin >> a >> b;
    int somma = a + b;
    cout << "La somma di " << a
         << " e " << b << " vale "
         << somma << endl;
    return 0; }
```

Istruzione per l'input dei dati

Identificatore

140



Struttura di un programma

- Un programma è strutturato in un insieme di funzioni, eventualmente suddivise in più file.
- Una funzione implementa uno specifico algoritmo o funzionalità del programma.
- Ad esempio un programma per la gestione del personale di un'azienda potrà avere una funzione per cercare la scheda di un dipendente dato il suo nominativo, una funzione per calcolare gli stipendi, una funzione per cercare tutti i dipendenti con una certa qualifica, ecc.

141



Programmazione procedurale

- Questo approccio (o paradigma) è detto programmazione procedurale.
- Il paradigma di programmazione procedurale può essere riassunto così:
 - Si definiscano le procedure desiderate
 - Si utilizzino gli algoritmi migliori
- La programmazione procedurale si concentra quindi sul processo, sulla scrittura del miglior algoritmo per eseguire i calcoli desiderati.
- E' il paradigma di programmazione originario e più "antico", ancora oggi piuttosto usato.

142



Funzione main

- Ogni programma deve avere una funzione principale che si deve chiamare **main**.
- L'esecuzione del programma inizia e termina dalla funzione main.

- Esempio:

```
int main() {
    int x=2; int y=6; int z;
    z=x*y;
    cout << z; return 0;}

```

- La funzione main, come tutte le funzioni, è composta da una sequenza di istruzioni che operano su dati (dichiarati).

143



Struttura di un programma

In prima approssimazione, un programma è quindi strutturato in questo modo:

```
dichiarazione1;
...
dichiarazioneN1;

int main()
{
    dichiarazione1;
    ...
    dichiarazioneN2;

    istruzione1;
    ...
    istruzioneN3;
}

```

144



Identificatori (*identifiers*)

- Le entità di un programma C++ devono avere dei nomi che ne consentano l'individuazione.
- Nel caso più semplice i nomi sono degli *identificatori* liberamente scelti.
- Un identificatore è una parola iniziante con una lettera. Notare che:
 - il carattere '`_`' è una lettera
 - il C++ distingue tra maiuscole e minuscole (si dice che è *case sensitive*). Ad esempio **F**act è diverso da **f**act.

145



Espressioni letterali

- Possono essere:
 - costanti carattere: '`a`', '`x`', '`z`', ...
 - costanti stringa: "`Ciao, come stai?`"
 - costanti numeriche intere: 10, 25, 1974, 3, ...
 - costanti numeriche reali: 2.51, 7.36, 51.294, ...

146



Operatori

Alcuni caratteri speciali e loro combinazioni sono usati come operatori, cioè servono a denotare certe operazioni nel calcolo delle espressioni.

Esempio: $2 * 3 + 4$

+	-	*	/	%	^	&		~
!	=	<	>	+=	-=	*=	/=	%=
^=	&=	=	<<	>>	>>=	<<=	==	!=
<=	>=	&&		++	--	,	->*	->
.*	::	()	[]	?:				

147



Variabili e Costanti

- Per memorizzare un valore in un'area di memoria si utilizzano entità chiamate variabili o costanti.
- Le variabili permettono la modifica del loro valore durante l'esecuzione del programma.
- Il valore delle costanti, invece, non può essere modificato.
- L'area di memoria corrispondente è identificata da un nome, che ne individua l'indirizzo.

148



Le variabili

Le variabili sono caratterizzate da una quadrupla:

- **nome (identificatore)**
- **tipo**
- **locazione di memoria**
- **valore**

149



Definizione di una variabile

- Definizione: quando il compilatore incontra una definizione di una variabile, esso predispone l'allocazione di un'area di memoria in grado di contenere la variabile del tipo scelto.

Esempio: `int x;`

- Definizione con inizializzazione:

Esempio: `int x = 3*2;`

150



Dichiarazione di Costanti

Sintassi:

`const tipo identificatore = exp;`
exp deve essere una espressione il cui valore deve poter essere calcolato in fase di compilazione.

Esempi:

```
const int kilo = 1024;
const double pi = 3.14159;
const int mille = kilo - 24;
```

151



Tipi

- Un tipo identifica un insieme di oggetti ed un insieme di operazioni che possono essere effettuate su tali oggetti.
- Oggetti dello stesso tipo utilizzano
 - lo stesso spazio in memoria
 - la stessa codifica
- Vantaggi sull'uso dei tipi:
 - correttezza semantica
 - efficiente allocazione della memoria dovuta alla conoscenza dello spazio richiesto in fase di compilazione

152



Tipi fondamentali e derivati

- Nel C++ i tipi sono distinti in:
 - tipi fondamentali, che servono a rappresentare informazioni semplici, come i numeri interi o i caratteri (**int**, **char**, ...)
 - tipi derivati, che permettono di costruire strutture dati complesse, a partire dai tipi fondamentali. Esempio: un vettore di N numeri interi.

153



Tipi fondamentali

- I tipi **fondamentali** del C++ sono:
 - **int**, **short**, **long**, tipi interi
 - **char**, tipo carattere
 - **float**, **double**, tipi reali

154



Il tipo `int`

- il tipo `int` è costituito dai numeri interi compresi tra due estremi, variabili a seconda dell'implementazione
- con N bit impiegati per la rappresentazione e assumendo una codifica in complemento a 2, si ha:

N	Min (-2^{N-1})	Max ($2^{N-1}-1$)
16	-32768	32767
32	-2147483648	2147483647

155



I tipi `short` e `long`

- Il tipo `short` tipicamente non ha più di 16 bit. Esempio:

```
short x = 1232;
short int x = -132;
```
- Il tipo `long` tipicamente ha almeno 32 bit
Esempio:

```
long y = -34213332;
long int x = -6767899;
```

156



Il tipo `unsigned`

- Il tipo `unsigned` rappresenta numeri interi non negativi di varie dimensioni

- Esempi:

```
unsigned int x = 1232;
unsigned short int x = 567;
unsigned long int
x = 878678687;
```

157



Operatori aritmetici sugli interi

Operatore	Significato
+	addizione
-	sottrazione
*	moltiplicazione
/	divisione intera
%	resto della divisione intera

158



Operatore di assegnamento

Sintassi dell'operatore di *assegnamento semplice*:

exp1 = exp2

exp1 deve far riferimento a una locazione di memoria

exp1 e ***exp2*** devono essere di tipo compatibile

Esempio: `int a, b;`

`a = 5; // a assume il valore 5`

`b = 6; // b assume il valore 6`

`a = b; // ora anche a assume
// il valore 6`

159



Operatori aritmetici e assegnamento

Forma compatta	Forma estesa
<code>x += y</code>	<code>x = x + y</code>
<code>x -= y</code>	<code>x = x - y</code>
<code>x *= y</code>	<code>x = x * y</code>
<code>x /= y</code>	<code>x = x / y</code>
<code>x %= y</code>	<code>x = x % y</code>

160



Operatori di incremento e decremento unitario

x++ incrementa **x** di una unità

x-- decrementa **x** di una unità

Esempio:

```
int a = 5;
a++;    // ora a vale 6
int b = 10;
b--;    // ora b vale 9
```

161



Operatori Relazionali e Logici

Operatore	Significato
==	uguaglianza
!=	diversità
>	maggiore
<	minore
>=	maggiore o uguale
<=	minore o uguale
&&	and
	or
!	not

162



Operatori logici

- Il valore FALSO è rappresentato dallo 0.
- Il valore VERO è rappresentato da un valore intero diverso da 0.
- Esempi:

x	y	!x	x y	x&& y
FALSO	FALSO	VERO	FALSO	FALSO
FALSO	VERO	VERO	VERO	FALSO
VERO	FALSO	FALSO	VERO	FALSO
VERO	VERO	FALSO	VERO	VERO

163



Tipi reali: float e double

- I tipi reali hanno come insieme di valori un sottoinsieme dei numeri reali, ovvero quelli rappresentabili all'interno del computer in un formato prefissato.
- Ne esistono tre tipi a seconda della precisione:
 - **float**
 - **double**
 - **long double**
- La precisione di ciascun tipo è maggiore o uguale a quella del tipo precedente.

164



Esempi di variabili con tipi reali

```
double a = 2.2, b = -14.12e-2;
double c = .57, d = 6.;
```

```
float g = -3.4F; //literal float
float h = g -.89F; //suffisso F (f)
```

```
long double i = +0.001;
long double j = 1.23e+12L;
// literal long double
// suffisso L (l)
```

165



Tipo carattere

- Il tipo **char** ha come insieme di valori i caratteri di stampa (es. 'a', 'Y', '6', '+').
- Il tipo **char** è un sottoinsieme del tipo **int**: generalmente un carattere occupa 1 byte.
- Il valore numerico associato ad un carattere è detto codice e dipende dalla codifica utilizzata dal computer (es. ASCII, UNICODE, ...).
- La codifica più usata è quella ASCII, ma il tipo **char** è indipendente dalla particolare codifica adottata.

166



Esempio sui caratteri

```
char c = 'f';
char n = '\n';
char l = 'a';
l += 3;           // l diventa 'd'
l--;             // l diventa 'c'
l -= 'a'-'A';    // l diventa 'C'
```

Nota: `l -= 'a'-'A';` è equivalente a
`l = l - 'a' + 'A';`

167



Operazioni miste e conversioni di tipo

- In un'espressione si possono usare operandi di tipo diverso o si può assegnare ad una variabile un valore di tipo diverso da quello della variabile.
- Esempio:


```
int prezzo = 27500;
double peso = 0.3;
int costo = prezzo * peso;
```
- In ogni operazione mista è sempre necessaria una conversione di tipo che può essere implicita o esplicita.

168



Conversioni implicite

- Le conversioni implicite vengono effettuate dal compilatore. Le più significative sono:
 - In ambito numerico, gli operandi sono convertiti al tipo di quello di dimensione maggiore.
Esempio: `3*2.6` viene calcolata come `3.0*2.6` e non come `3*2`
 - Nell'assegnamento, un'espressione viene sempre convertita al tipo della variabile.
Esempi:

```
float x = 3;    //equivale a: x = 3.0
int y = 2*3.6; //equivale a: y = 7
```

169



Conversioni esplicite

- Il programmatore può richiedere una conversione esplicita di un valore da un tipo ad un altro (casting)
- Esistono due notazioni:
 - Esempio:

```
int i = (int) 3.14;
```
 - Esempio:

```
double f = double(3)
```

170



Istruzioni semplici

- Le istruzioni semplici sono la base delle istruzioni più complesse (istruzioni strutturate).
- Si distinguono in:
 - dichiarazioni (declaration-statement) di
 - variabili, es. `int x`
 - costanti, es. `const int kilo = 1024`
 - espressioni (expression-statement)
 - di assegnamento, es. `x=2`
 - aritmetiche, es. `(x-3)*sin(x)`
 - logiche, es. `x==y && x!=z`
 - costanti, es. `3*12.7`

171



Istruzioni strutturate

- Le istruzioni strutturate consentono di specificare azioni complesse
- Si distinguono in
 - istruzioni composte (compound-statement)
 - istruzioni condizionali (conditional-statement)
 - istruzioni iterative (iteration-statement)
 - istruzioni di salto (jump-statement)

172



Istruzione composta

- L'istruzione composta consente, per mezzo della coppia di delimitatori '{' e '}', di trasformare una qualunque sequenza di istruzioni in una singola istruzione.
- Esempio:


```
{ int a = 4;
  a *= 6;
  char b = 'c';
  b += 3; }
```
- Le definizioni possono comparire in qualunque punto del blocco e sono visibili solo all'interno del blocco.

173



Istruzioni condizionali

- Le istruzioni condizionali consentono al programmatore di specificare che una istruzione (semplice o composta) venga eseguita solo al verificarsi di una determinata condizione.
- Qualche esempio:
 - Eseguire la divisione tra due numeri solo se il divisore è diverso da zero
 - Considerare un carattere immesso da tastiera solo se si tratta di una vocale
 - Calcolare l'incasso settimanale solo se il giorno corrente è sabato.

174



Istruzioni condizionali

- Elementi delle istruzioni condizionali:
 - Condizione: espressione il cui valore determina se eseguire o meno l'istruzione (semplice o composta) associata all'istruzione condizionale.
 - Corpo: l'istruzione (semplice o composta) che viene eseguita nel caso in cui la condizione sia verificata.
- Esempio:

```
leggi x
leggi y
se (y diverso da 0)  condizione
                    corpo
    stampa x / y
```

175



Istruzione condizionale `if`

Sintassi:

`if (exp) istruzione1;`

Se *exp* è vera viene eseguita *istruzione1*

Esempio:

`if (x!=0) y=1/x;`

Sintassi:

`if (exp) istruzione1;`

`else istruzione2;`

Se *exp* è vera viene eseguita *istruzione1*
altrimenti viene eseguita *istruzione2*

Esempio:

`if (x<0) y=-x; else y=x;`

176



Esempio

```
#include <iostream>
using namespace std;
// calcolo soluzioni di  $ax + b = 0$ 

int main() {
    float a; float b;
    cin >> a >> b;
    if (a == 0)
        if (b == 0)
            cout<<"Infinite soluzioni"<<endl;
        else
            cout<<"Non esiste soluzione"<<endl;
    else
        cout<<"Soluzione: "<< -b/a <<endl;
    return 0; }
```

177



Un altro esempio

```
#include <iostream>
using namespace std;
int main() { //semplice calcolatrice
    double op1; double op2; char op;
    cin >> op1 >> op >> op2;
    if (op == '+')
        cout << op1 + op2;
    else if (op == '-')
        cout << op1 - op2;
    else if (op == '*')
        cout << op1 * op2;
    else if ((op=='/') && (op2!=0))
        cout << op1 / op2;
    else cout<<"Errore!";

    return 0; }
```

178



Istruzione condizionale `switch`

Sintassi:

```
switch (exp) {
  case const-exp1: istruzione1; break;
  case const-exp2: istruzione2; break;
  ...
  default: istruzione-default; }
```

L'esecuzione dell'istruzione `switch` consiste:

- nel calcolo dell'espressione `exp`
- nell'esecuzione dell'istruzione corrispondente all'alternativa specificata dal valore calcolato
- se nessuna alternativa corrisponde, se esiste, viene eseguita *istruzione-default*

179



Esempio

```
#include <iostream>
using namespace std;
int main() {
  //semplice calcolatrice
  double op1; double op2; char op;
  cin >> op1 >> op >> op2;
  switch (op)
  {
    case '+': cout << op1+op2; break;
    case '-': cout << op1-op2; break;
    case '*': cout << op1*op2; break;
    case '/': cout << op1/op2; break;
    default : cout << "Errore!"; }
  return 0; }
```

180



Istruzioni iterative

- Le istruzioni iterative (dette anche cicli o loop) consentono al programmatore di specificare che una istruzione (semplice o composta) dovrà essere ripetuta più volte.
- Qualche esempio:
 - Scrivere 100 volte la parola “Ciao”
 - Stampare tutti i numeri da 1 a 1000
 - Sommare i numeri interi inseriti da tastiera finché non viene inserito un numero negativo
 - Calcolare ogni giorno l’incasso giornaliero finché non si giunge alla fine del mese.

181



Istruzioni iterative

- Un ciclo è costituito da quattro elementi:
 - Inizializzazione: assegnamento del valore iniziale di tutte le variabili usate durante il ciclo
 - Condizione di ripetizione: determina se compiere una nuova iterazione del ciclo o uscire dal ciclo
 - Corpo: l’istruzione (semplice o composta) che occorre ripetere; usa le variabili inizializzate e ne può modificare il valore
 - Aggiornamento: modifica una o più variabili in grado di aggiornare il valore della condizione di ripetizione, tenendo così traccia del progresso dell’iterazione.

182



Istruzioni iterative

- Esempio: compi 10 giri di pista; ad ogni giro incrementa il numero di km percorsi di 5 km e stampa quanta benzina resta nel serbatoio

```

n_giri = 0
km_percorsi = 0  inizializzazione

finché (n_giri < 10)  condizione di ripetizione
corpo
    km_percorsi = km_percorsi + 5
    stampa benzina rimasta
    n_giri = n_giri + 1  aggiornamento
  
```

183



Istruzioni iterative

- Un ciclo può essere eseguito:
 - un numero prefissato di volte, ovvero il numero di iterazioni è noto a priori al programmatore.
Si parla allora di *ciclo definito* o *ciclo controllato da un contatore*
Esempio: compi 10 giri di pista
 - un numero imprecisato di volte, ovvero il numero di iterazioni non è noto a priori al programmatore.
Si parla di *ciclo indefinito* o di *ciclo controllato da un valore sentinella* (o valore flag o valore dummy)
Esempio: continua a compiere giri di pista finché la benzina non sta per finire

184



Ciclo definito

- Il numero di volte per cui il ciclo è ripetuto è determinato da un contatore. Si avrà:
 1. inizializzazione del contatore
 2. aggiornamento consistente nell'incremento o nel decremento del contatore
- Esempio:


```
1. num_giri = 0
2. num_giri = num_giri + 1 oppure
   num_giri++
```

185



Ciclo indefinito

- Il numero di volte per cui il ciclo è ripetuto è determinato da un valore sentinella:
 - Il valore sentinella indica la “fine dei dati” e di solito è letto da tastiera o calcolato nel corpo del ciclo
 - NON scegliere un valore sentinella che sia anche un valore legittimo per i dati
 - E' buona norma ricordare all'utente quale è il valore sentinella
 - Esempio: se il ciclo opera su numeri interi positivi, il valore sentinella può essere un numero negativo (ad esempio -1)

186



Ciclo indefinito

- La condizione dipende dal valore sentinella
- E' difficile distinguere corpo da aggiornamento
- Occorre fare attenzione a dove mettere l'aggiornamento!
- Problema dell'inizializzazione: che valore sentinella si usa?
- Esempio: calcolare la media di una sequenza di voti di esame immessi dall'utente finché l'utente non inserisce un valore non valido (cioè un valore non compreso tra 18 e 30).

187



Istruzione iterativa `while`

Sintassi:

`while` (*exp*) *istruzione* ;

L'esecuzione dell'istruzione **`while`** comporta:

- il calcolo dell'espressione *exp*;
- se *exp* è vera, l'esecuzione di *istruzione* e la ripetizione dell'esecuzione dell'istruzione **`while`**.

188



Esempio

```
#include <iostream>
using namespace std;
int main() {
    int num = 0; int ndiv2 = 0;
    cout << "Immetti un numero: ";
    cin >> num;
    while (num % 2 == 0) {
        ndiv2++;
        num /= 2;
    }
    cout << "E' divisibile " << ndiv2
         << " volte per 2" << endl;
    return 0;}

```

189



Istruzione iterativa do

Sintassi:

`do { istruzione } while (exp) ;`

L'esecuzione dell'istruzione `do` comporta:

- l'esecuzione di *istruzione*;
- il calcolo dell'espressione *exp*;
- se *exp* è vera, la ripetizione dell'esecuzione dell'istruzione `do`.

190



Esempio

```
#include <iostream>
using namespace std;
int main() { //conversione di base
    //notazione rovesciata
    int cifra; int num; int base;
    cin >> num >> base;
    if (base >= 2 && base <= 10)
        do {
            cifra = num % base;
            num /= base;
            cout << cifra;
        } while (num != 0);
    else cout << "Base non valida!";
    return 0;}
```

191



Istruzione iterativa `for`

Sintassi:

`for` (*init-exp*; *exp1*; *exp2*) *istruzione*;

L'esecuzione dell'istruzione **`for`** comporta:

1. L'esecuzione di *init-exp* (che di solito è un assegnamento che inizializza una variabile, detta di controllo);
2. Il calcolo dell'espressione *exp1*;
3. Se *exp1* è vera, viene eseguita *istruzione*, poi *exp2* (che di solito è un incremento o un decremento della variabile di controllo) e si reinizia dal passo 2.

192



Esempio

```
#include <iostream>
using namespace std;
int main() {
    //calcolo del fattoriale
    int fattoriale; int num;
    cout << "Inserisci un numero: ";
    cin >> num;
    fattoriale = 1;
    for (int i = 1; i <= num; i++)
        fattoriale *= i;
    cout << "il suo fattoriale e' "
         << fattoriale;
    return 0;}

```

193



Trasformazione di for in while

`for (exp1; exp2; exp3) exp4;`
 equivale a
`exp1; while (exp2) {exp4; exp3};`

Esempio:

`for (int i=1; i<10; i++) x*=2;`
 equivale a
`int i=1; while (i<10) {x*=2; i++};`

194



Un esempio più complesso ...

```
#include <iostream>
using namespace std;
int main(){//numeri divisibili per 3
    int start; int stop; char c='s';
    while (c == 's') {
        cout<<"Inserisci un intervallo: ";
        cin >> start >> stop;
        for (int i=start; i<=stop; i++) {
            if ((i % 3)==0)
                cout << i <<
                    " divisibile per 3\n"; }
        do {
            cout<<"Vuoi continuare (s/n)? ";
            cin >> c; cout << endl;
        } while (c != 's' && c != 'n');
    }
}
```

195



Istruzione di salto `break`

- All'interno di un ciclo, l'istruzione **break** termina direttamente il ciclo.

```
while (...) {
    ...
    while (...) {
        ...
        break;
        ...
    }
    ...
}
```

196



Esempio

```
#include <iostream>
using namespace std;
int main(){
    int a; int b; char c;
    while (1) {
        cout << "Inserisci due numeri: ";
        cin >> a >> b;
        cout << a << '+' << b << '='
             << a + b << endl;
        cout << "Vuoi continuare (s/n)?";
        cin >> c;
        if (c == 'n') break;
    }
}
```

197



Istruzione di salto `continue`

- L'istruzione `continue` termina l'iterazione che attualmente è in esecuzione e passa all'iterazione successiva (nel caso di `for` viene saltata l'istruzione di aggiornamento).

```
while (...) {
    ...
    while (...) {
        ...
        continue;
    }
    ...
}
```

198



Esempio

```
#include <iostream>
using namespace std;
int main() {
    // somma i numeri multipli di 5 o 8
    // nell'intervallo da 1 a 30
    int i = 0; int somma = 0;
    while (i < 30) {
        i++;
        if ((i % 5) != 0 && (i % 8) != 0)
            continue;
        cout << i << endl;
        somma += i;
    }
    cout << "somma = " << somma << endl; }
}
```

199



Concetto di funzione

- Nella realizzazione di un programma
 - il codice può diventare molto lungo
 - spesso una stessa sequenza di operazioni viene ripetuta in più punti
 - È quindi opportuno e conveniente strutturare il codice raggruppando delle sue parti in moduli autonomi, detti funzioni, che vengono eseguiti in ogni punto in cui è richiesto
- L'organizzazione a funzioni consente di isolare parti di un programma che possono essere modificate in maniera indipendente.

200



Esempio

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

int somma(int a, int b) {
    int c = a + b;
    return c;
}

int main() {
    int x; int y; int z;
    cin >> x >> y;
    z = somma(x, y);
    cout << z;
    return 0; }
```

201



Definizione, Dichiarazione e Chiamata

- Definizione:
tipo id (tipo1 id1, ..., tipoN idN) { corpo }
- Dichiarazione:
tipo id (tipo1 id1, ..., tipoN idN) ;
id1, ..., idN sono i parametri formali della funzione
- Chiamata:
id (exp1, ..., expN)
exp1, ..., expN sono i parametri attuali della chiamata e devono essere di tipo compatibile ai tipi dei corrispondenti parametri formali

202



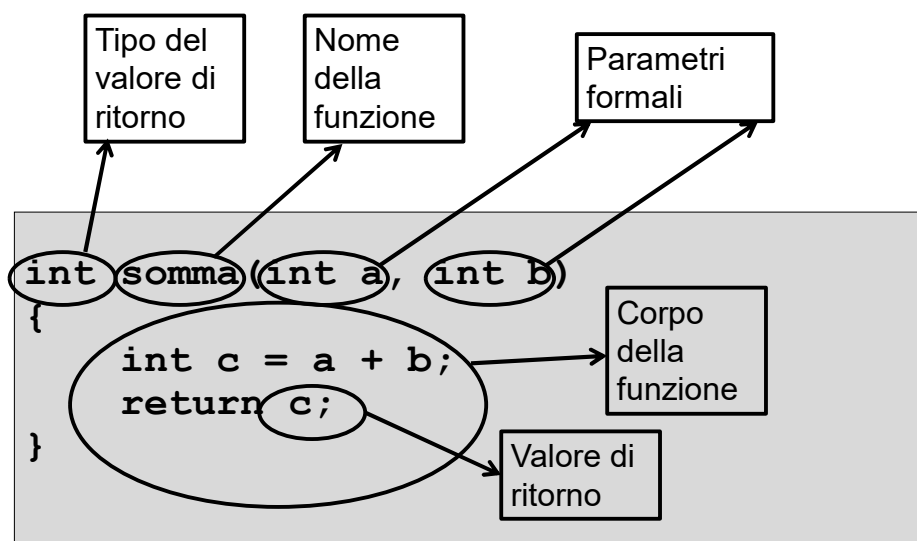
Istruzione `return`

- Il corpo di una funzione può contenere una o più volte l'istruzione **`return`**
- Sintassi:
`return expression;`
- L'istruzione **`return`** fa terminare la funzione e fa sì che il valore restituito dalla funzione sia il valore dell'espressione *expression*.
- L'espressione può mancare nel caso in cui si abbia a che fare con funzioni che non restituiscono un risultato (procedure).

203



Esempio



204

Esempio

```

int main() {
    int x; int y; int z;
    cin >> x >> y;
    z = somma(x, y);
    cout << z;
    return 0; }
  
```

Tipo del valore di ritorno

Nome della funzione

Parametri attuali

Chiamata della funzione

205

Un altro esempio di funzione

```

// funzione che calcola il massimo
// comune divisore tra due numeri
// interi positivi

int mcd(int a, int b)
{
    int resto;
    while (b != 0) {
        resto = a % b;
        a = b;
        b = resto;
    }
    return a;
}
  
```

206



Chiamata di funzione

```
#include <iostream>
using namespace std;
// programma che chiama la funzione
// mcd per il calcolo del M.C.D.
int main()
{
    int n1; int n2;
    cout << "Inserisci una coppia "
           "di numeri ";
    cin >> n1 >> n2;
    cout << "Il M.C.D. fra " << n1
         << " e " << n2 << " vale "
         << mcd(n1, n2) << endl;}
```

207



Parametri e variabili locali (1)

- Un parametro formale è una variabile cui viene assegnato il corrispondente parametro attuale ad ogni chiamata della funzione.
- Le variabili dichiarate all'interno di una funzione sono dette locali a tale funzione e appartengono solo alla funzione in cui sono dichiarate (sono visibili solo nella funzione).

208



Parametri e variabili locali (2)

- I parametri formali e le variabili locali esistono solo durante l'esecuzione della rispettiva funzione.
- All'atto della chiamata viene riservata un'area di memoria per contenere i valori dei parametri formali (passati per valore) e delle variabili locali.
- I parametri formali e le variabili locali vengono utilizzati per le dovute elaborazioni.
- Al termine della funzione la memoria da essi occupata viene resa disponibile.

209



Passaggio parametri

In C++ esistono due modalità di associazione dei parametri attuali ai parametri formali:

- Passaggio parametri per valore: il parametro formale ha una sua propria cella di memoria in cui, all'atto della chiamata, viene copiato il valore del parametro attuale corrispondente.
- Passaggio parametri per riferimento: la cella di memoria associata al parametro formale è quella stessa del corrispondente parametro attuale (che si presuppone ne abbia una).

210



Chiamata della funzione `mcd`

- La chiamata di funzione `mcd(n1, n2)` viene eseguita nel modo seguente:
 - vengono calcolati i valori dei parametri attuali `n1` e `n2` (l'ordine non è specificato);
 - i valori vengono copiati, nell'ordine, nei contenitori `a` e `b` (parametri formali);
 - viene eseguita la funzione e modificati i valori di `a`, `b` e della variabile locale `resto` (`n1` e `n2` rimangono con il loro valore originale);
 - la funzione `mcd` restituisce al programma chiamante il valore dell'espressione che appare nell'istruzione `return`;

211



Esempio di passaggio per riferimento

```
// procedura che scambia due interi
void scambia(int &n, int &m)
{
    int t;
    t = n; n = m; m = t;
}

// procedura errata!
void scambia(int n, int m)
{
    int t;
    t = n; n = m; m = t;
}
```

212



Procedure (funzioni `void`)

- L'importanza di individuare un corpo funzionalmente rilevante all'interno di un programma può valere anche se questo corpo non ritorna un valore.
- In C++ c'è perciò la possibilità di definire procedure, ovvero funzioni il cui valore di ritorno è di tipo `void`.

213



Esempio

```
#include <iostream>
using namespace std;
void stampa_coppia (int a, int b)
{ cout << '(' << a << ', '
  << b << ')'; }
int main()
{
  int n1; int n2;
  cout << "Inserisci due numeri ";
  cin >> n1 >> n2;
  cout << "La coppia e' ";
  stampa_coppia(n1, n2);
}
```

214



Funzioni di libreria

- Una libreria è un insieme di funzioni precompilate. Alcune librerie C++ sono disponibili in tutte le implementazioni e con le stesse funzioni.
- Una libreria è formata da una coppia di file:
 - un file di intestazione (header) contenente le dichiarazioni delle funzioni stesse che in genere termina con l'estensione `.h`
 - un file contenente le funzioni compilate
- Per utilizzare una libreria bisogna:
 - includere il file di intestazione della libreria con la direttiva `#include <nomelibreria.h>`
 - indicare al linker il file contenente le funzioni compilate della libreria

215



Esempio: funzioni aritmetiche

Nella libreria `<cstdlib>`

- `abs (n)` valore assoluto
- `rand ()` numero pseudocasuale tra 0 e la costante `RAND_MAX`
- `srand (n)` inizializza la funzione `rand`

Nella libreria `<cmath>`

- `fabs (x)` valore assoluto di tipo float
- `sqrt (x)` radice quadrata di `x`
- `pow (x, y)` eleva `x` alla potenza di `y` (x^y)
- `exp (x)` eleva `e` alla potenza di `x` (e^x)
- `log (x)` logaritmo naturale di `x`
- `log10 (x)` logaritmo in base 10 di `x`
- `sin (x)` e `asin (x)` seno e arcseno trigonometrico
- `cos (x)` e `acos (x)` coseno e arcocoseno trigonometrico
- `tan (x)` e `atan (x)` tangente e arcotangente trigonometrica

216



Tipi e Variabili Array

- Un array è una sequenza finita di elementi omogenei (dello stesso tipo).
- Sintassi:
`tipo id[dim] ;`
`tipo id[dim]={lista_valori} ;`
`tipo id[]={lista_valori} ;`
- Esempi:
`double a[25]; //array di 25 double`
`const int c = 2;`
`char b[2*c]={'a','e','i','o'};`
`char d[]={ 'a','e','i','o','u'};`

217



Operazioni sugli array

- Sugli array non sono definite né operazioni aritmetiche, né operazioni di confronto, né l'assegnamento.
- L'unica operazione definita è la selezione con indice (subscripting), ottenuta nella forma:
`identifier[expression]`
 - L'identificatore è il nome dell'array;
 - Il valore dell'espressione, di tipo discreto, è l'indice dell'elemento cui si vuole accedere.
- Gli elementi di un array di dimensione **N** sono numerati da **0** a **N-1**.
- Esempio: `a[3]=7` assegna il quarto elemento.

218



Esempio

```
#include <iostream>
using namespace std;
int main() {
    const int size = 6;
    int a[size] = {3, 2, 4, 1, 9, 2};
    int min = a[0]; int index = 0;
    for (int k = 1; k < size; k++) {
        if (a[k] < min) {
            min = a[k]; index = k;
        }
    }
    cout << "Il minimo è " << min
         << " e il suo indice è " << index
         << endl; }
```

219



Funzione con parametri di tipo array

- Una funzione può avere un parametro formale del tipo “array di oggetti di tipo T”.
- Il corrispondente parametro attuale è costituito dal nome di un array di oggetti di tipo T che equivale all’indirizzo del suo primo elemento.
- Quando viene passato un array ad una funzione, non vengono copiati i suoi elementi nel corrispondente parametro attuale ma viene ricopiato solo l’indirizzo del primo elemento.

220



Parametri di tipo array

- Nella dichiarazione di un parametro formale di tipo array (monodimensionale) si può omettere la specifica della dimensione per poter operare su array di dimensioni diverse.

```
// funzione che somma i primi n
// elementi di un array
int somma(int v[], int n) {
    int i; int s = 0;
    for (i = 0; i < n; i++)
        s += v[i];
    return s;
}
```

221



Esempio di passaggio array

```
#include <iostream>
using namespace std;
void main() {
    int a[100], n;
    cout<<"Quanti numeri (max 100)?";
    cin >> n; if (n > 100) return;
    for (int i = 0; i < n; i++) {
        cout << endl << i + 1 << ": ";
        cin >> a[i];
    };
    cout << "La media vale "
        << somma(a, n)/n << endl;}
```

222



Stringhe

- Una stringa è un array di `char`, il cui ultimo elemento è il carattere nullo (`'\0'`).
- L'inizializzazione di una variabile stringa si può effettuare utilizzando una costante stringa
Esempio: `char stringa[] = "Ciao";`
- L'array stringa contiene 5 elementi: i 4 caratteri della costante stringa e il carattere nullo che viene inserito automaticamente.
- Nota: dato che negli array non è definito l'assegnamento, l'uso di una costante stringa per specificare il valore di una stringa è permesso solo nell'inizializzazione.

223



Input e Output di Stringhe (1)

- Gli operatori di ingresso e di uscita accettano una stringa come argomento.
- L'operatore di ingresso legge caratteri dallo stream di ingresso e li memorizza in sequenza finché non incontra un carattere di spaziatura.
- L'occorrenza di tale carattere, che non viene letto, causa il termine dell'operazione e la memorizzazione nella stringa del carattere nullo dopo l'ultimo carattere letto.

224



Input e Output di Stringhe (2)

- L'operatore di uscita scrive i caratteri della stringa (escluso il carattere nullo finale) sullo stream di uscita.
- Il seguente esempio legge una stringa al massimo di 255 caratteri (più il carattere nullo finale) e la stampa:

```
char buffer[256];
cin >> buffer;
cout << buffer;
```

225



Esempio

```
#include <iostream>
#include <ctype>
using namespace std;
int main() {
    char buffer[256];
    cout << "Inserisci una stringa: ";
    cin >> buffer;
    char c = buffer[0]; int k = 0;
    while (c != '\0') {
        if (isalpha(c) && islower(c))
            buffer[k] -= 'a' - 'A';
        k++;
        c = buffer[k];
    }
    cout << buffer;
}
```

226



Funzioni di <cstring>

Nelle funzioni *s* e *t* sono stringhe e *c* è un carattere

strcpy (<i>s</i> , <i>t</i>)	copia <i>t</i> in <i>s</i>
strcat (<i>s</i> , <i>t</i>)	concatena <i>t</i> al termine di <i>s</i>
strcmp (<i>s</i> , <i>t</i>)	confronta <i>s</i> e <i>t</i> e restituisce un valore negativo, nullo o positivo se <i>s</i> è alfabeticamente minore, uguale o maggiore di <i>t</i> .
strchr (<i>s</i> , <i>c</i>)	restituisce un puntatore alla prima occorrenza di <i>c</i> in <i>s</i> , oppure 0 se <i>c</i> non si trova in <i>s</i> .
strrchr (<i>s</i> , <i>c</i>)	come sopra ma per l'ultima occorrenza di <i>c</i> in <i>s</i> .
strlen (<i>s</i>)	restituisce la lunghezza di <i>s</i> .

227



Array multidimensionali

- È possibile dichiarare array i cui elementi sono a loro volta degli array, generando così degli array multidimensionali.
- Sintassi:
 $\text{tipo id}[dim_1][dim_2] \dots [dim_n];$
 $\text{tipo id}[dim_1][dim_2] \dots [dim_n] = \{ \text{lista_valori} \};$
 $\text{tipo id}[] [dim_2] \dots [dim_n] = \{ \text{lista_valori} \};$
- Quindi in C++ un array multidimensionale $dim_1 \times dim_2 \times \dots \times dim_n$ può essere pensato come un array di dim_1 array multidimensionali $dim_2 \times \dots \times dim_n$

228



Array bidimensionali

- La forma più semplice è quella a due dimensioni, in cui gli elementi dell'array possono essere individuati attraverso due indici. Gli array bidimensionali sono usati per implementare le matrici.
- Sintassi:
 $\text{tipo id}[\text{dim}_1][\text{dim}_2];$
 $\text{tipo id}[\text{dim}_1][\text{dim}_2]=\{\text{lista_valori}\};$
 $\text{tipo id}[][\text{dim}_2]=\{\text{lista_valori}\};$
- dim_1 specifica il numero di righe della matrice.
 dim_2 specifica il numero di colonne.

229



Array bidimensionali: esempio

- `int a[3][4];`

	Colonna 0	Colonna 1	Colonna 2	Colonna 3
Riga 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Riga 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Riga 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Diagram illustrating the structure of a 2D array `a` with 3 rows and 4 columns. The array is represented as a table with rows labeled Riga 0, Riga 1, and Riga 2, and columns labeled Colonna 0, Colonna 1, Colonna 2, and Colonna 3. Each cell contains an element address, e.g., `a[0][0]`, `a[0][1]`, etc. Arrows point to the components of the address `a[2][1]`: `a` is the **nome della matrice** (name of the matrix), `2` is the **indice riga** (row index), and `1` is the **indice colonna** (column index).

230



Array bidimensionali

- Dichiarazione e inizializzazione:

```
int c[2][2] = {{1,2},{3,4}};
```

– 1 e 2 inizializzano $c[0][0]$ e $c[0][1]$

– 3 e 4 inizializzano $c[1][0]$ e $c[1][1]$

```
int c[][2]={{1,2},{3,4}};
```

– il compilatore deduce il numero di righe dal numero di coppie di parentesi graffe interne.

- Una matrice a di n righe e m colonne occupa $n \times m \times L$ byte, essendo L il numero di byte con cui è rappresentato il tipo degli elementi. La matrice è memorizzata in locazioni contigue.

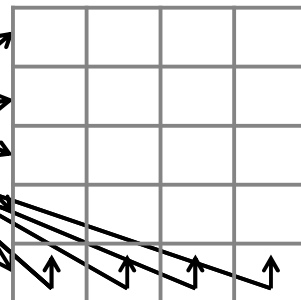
231



Array multidimensionali

- La selezione con indice in un array multidimensionale avviene specificando il valore dell'indice per ogni dimensione.
- L'accesso sequenziale ai dati richiede, tipicamente, dei cicli annidati.

```
const int m=5, n=4;
int matrix[m][n];
for (int i=0; i<m; i++)
    for (int j=0; j<n; j++)
        cout << a[i][j];
```



232



Esempio

```
#include <iostream>
int main() {
    const int size1 = 2; const int size2 = 3;
    int a[size1][size2] = {{3,2,4},{1,9,2}};
    int min = a[0][0];
    int index1 = 0, index2 = 0;
    for (int h = 0; h < size1; h++)
        for (int k = 0; k < size2; k++)
            if (a[h][k] < min) {
                min = a[h][k];
                index1 = h; index2 = k;
            }
    cout << "Il minimo è " << min
         << " e i suoi indici sono "
         << index1 " e " << index2 << endl;
}
```

233



Ancora su array e funzioni

- Per passare ad una funzione un array multidimensionale le seguenti scritture sono equivalenti:
`int f(int arr[dim1][dim2]...[dimN]);`
`int f(int arr[][dim2]...[dimN]);`
- Una funzione non può restituire direttamente un array come valore di ritorno.

234



Algoritmi di ordinamento

- *Problema*: ordinare gli elementi di un array.
- Un algoritmo di ordinamento è un algoritmo che ordina gli elementi di un insieme (ad esempio di un array) secondo una sequenza stabilita da una relazione d'ordine.
- *Esempio*: dato un array di numeri interi ordinarli in senso crescente o decrescente.

235



Esempio: ordinamento BubbleSort

```
void BubbleSort (int v[], int n) {
    int i, j, tmp;
    for (i = n - 1; i > 0; i--)
        for (j = 0; j < i; j++)
            if (v[j] > v[j+1]) {
                tmp = v[j];
                v[j] = v[j+1];
                v[j+1] = tmp;
            }
}
```

236



Concetto di stream

- Un programma comunica con l'esterno tramite uno o più flussi (stream).
- Uno stream è una struttura logica costituita da una sequenza di caratteri, in numero teoricamente infinito, terminante con un apposito carattere che ne identifica la fine.
- Gli stream vengono associati (con opportuni comandi) ai dispositivi fisici collegati al computer (tastiera, video, stampante) o a file residenti sulla memoria di massa.

237



Stream predefiniti

- In C++ esistono i seguenti stream predefiniti:
 - **cin** (stream standard di ingresso)
 - **cout** (stream standard di uscita)
 - **cerr** (stream standard di errore)
- Le funzioni che operano su questi stream sono in una libreria di ingresso/uscita e per usarle occorre la direttiva
#include <iostream>

238



Uso dei file

- Un'apposita libreria del C++ consente di utilizzare stream che vengono associati a file del sistema operativo.
- È utilizzabile con la direttiva **`#include <fstream>`**
- Stream di questo tipo vengono dichiarati con **`fstream nomeidentificatore;`**
- Ad esempio:
`fstream ingresso, uscita;`

239



Accesso ai file

- Uno stream associato ad un file può essere aperto mediante la funzione **`open()`** secondo le modalità, identificate dalle costanti tra ()
 - lettura (**`ios::in`**)
 - scrittura (**`ios::out`**)
 - append (scrittura alla fine del file) (**`ios::app`**)
- Il nome del file viene specificato come una sequenza di caratteri (es. **`"file1.in"`**)
- Esempio:
`fstream ingr, usc;`
`ingr.open("file1.in", ios::in);`
`usc.open("file2.out", ios::out);`

240



Apertura di uno stream associato a file

- Apertura in lettura:
 - il file associato deve già essere presente;
 - il puntatore si sposta all'inizio dello stream;
- Apertura in scrittura:
 - il file associato se non è presente viene creato;
 - il puntatore si posiziona all'inizio dello stream (eventuali dati presenti vengono perduti);
- Apertura in append (concatenazione)
 - il file associato se non è presente viene creato;
 - il puntatore si posiziona alla fine dello stream;

241



Chiusura di uno stream associato a file

- Uno stream, quando è stato utilizzato, può essere chiuso mediante la funzione `close()`
- Esempio:


```
ingr.close();
usc.close();
```
- Alla fine del programma tutti gli stream aperti vengono automaticamente chiusi.
- Una volta chiuso, uno stream può essere riaperto in qualunque modalità e associato a qualunque file.

242



Esempio

```
#include <fstream>
#include <iostream>
using namespace std;
int main() {
    fstream fin;
    fin.open("test.txt", ios::in);
    double somma = 0; double x; fin >> x;
    while(!fin.eof()) {
        somma = somma + x;
        fin >> x; }
    fin.close();
    cout << "La somma dei numeri contenuti
    nel file e' " << somma << endl;
    return 0;}
```

243



Strutture

- Una struttura è una collezione ordinata di elementi eventualmente eterogenei (di tipi diversi), detti membri o campi, ciascuno avente specifico tipo, nome e valore.
- Sintassi:


```
struct new_struct_id {
    tipo1 campo1;
    ...
    tipon campon; } ;
```
- Una variabile struttura può essere definita così: *new_struct_id var_id*;

244



Esempi di strutture

```
struct data {
    int giorno;
    int mese;
    int anno;
};

struct persona {
    char nome[20];
    char cognome[25];
    char comune_nascita[25];
    data data_nascita;
};
```

245



Accesso ai campi di una struttura

- Se `s` è una struttura e `field` è l'identificatore di un campo, allora `s.field` denota il campo della struttura.

- Esempio:

```
struct complex {
    double re; double im;
};

complex c;
c.re = 2.5;
```

246



Assegnamento di strutture

- A differenza degli array, l'assegnamento è definito per le variabili di tipo **struct**.
- L'assegnamento di strutture comporta la copia dei valori di tutti i campi.
- Esempio:

```
struct complex {
    double re; double im;
};
complex c1, c2;
c1.re = 2.5; c1.im = 3;
c2 = c1; //assegnamento di struct
```

247



Parametri di tipo struttura

- Una funzione può avere parametri formali di tipo struttura.
- Il passaggio di strutture a funzioni avviene nel modo usuale, per valore o per riferimento.
- Una funzione può restituire valori di ritorno di tipo struttura.
- Esempio:

```
int calcola_età(persona p, data oggi);
complex somma(complex a, complex b);
```

248



Parametri di tipo struttura

- Esempio: una funzione che somma due numeri complessi.

```
complex somma(complex a, complex b)
{
    complex s;
    s.re = a.re + b.re;
    s.im = a.im + b.im;
    return s;
}
```

249



Array di strutture

- Si possono utilizzare array di strutture per rappresentare archivi di dati.
- Esempio. I dipendenti di una azienda, i libri in una biblioteca, i modelli di automobile in vendita presso un concessionario:

```
persona personale_azienza[100];
libro biblioteca[1000];
automobile modelli[30];
```

250