
Materiale didattico di supporto alle lezioni del corso di

Fondamenti di Programmazione

Docenti:

Marco Cococcioni (titolare del corso)

Carlo Puliafito

Nicola Messina

Jacopo Cecchetti

Corso di Laurea in Ingegneria Informatica
Dipartimento di Ingegneria dell'Informazione
Scuola di Ingegneria – Università di Pisa

Anno Accademico 2025-2026

A-1

Libri di testo consigliati

1. Andrea Domenici, Graziano Frosini
*Introduzione alla Programmazione ed
Elementi di Strutture Dati con il Linguaggio C++*
Milano: Franco Angeli.
(va bene dalla quinta edizione in poi)



2. Paolo Corsini e Graziano Frosini
*Note sull'organizzazione di un calcolatore e
Rappresentazione dell'informazione*
Edizioni ETS, Pisa, 2011.



3. Raccolta di slide delle lezioni in formato pdf (la presente dispensa)
-

A-2

Dove si possono acquistare i libri di testo

Presso le principali librerie di Pisa

- Libreria Pellegrini (via Curtatone e Montanara, 5)
<http://www.libreriapellegrini.it/>
- Libreria Testi Universitari (via Nelli, 1-3 oppure via Santa Maria 14)
<https://www.libreriatestiuniversitari.it/>
- ...
- oppure su Amazon

A-3

Orario settimanale (TEO= teoria, LAB= laboratorio)

	Lu	Ma	Me	Gi		Ve
08:30-09:30			FdP TEO (3h) F9			
09:30-10:30						
10:30-11:30		FdP TEO (2h) F9				
11:30-12:30						
12:30-13:30						
13:30-14:30						
14:30-15:30				FdP TEO (1h) F9		
15:30-16:30				FdP LAB (2h) Gruppo A A13	FdP LAB (2h) Gruppo B F9	
16:30-17:30						

Gruppo A: studenti aventi matricola che termina per 0,1,2,3

Gruppo B: studenti aventi matricola che termina per 4,5,6,7,8,9

A-4

Argomenti del corso

■ Concetti di base della programmazione

Concetto di algoritmo. Il calcolatore come esecutore di algoritmi. Linguaggi di programmazione ad alto livello. Sintassi e semantica di un linguaggio di programmazione. Metodologie di programmazione strutturata. Principi fondamentali di progetto e sviluppo di semplici algoritmi.

■ Rappresentazione dell'informazione

Rappresentazione dei caratteri, dei numeri naturali, dei numeri interi e dei numeri reali.

■ Programmare in C

Tipi fondamentali. Istruzioni semplici, strutturate e di salto. Funzioni. Ricorsione. Riferimenti e puntatori. Array. Strutture e unioni. Memoria libera. Visibilità e collegamento. Algoritmi di ricerca e di ordinamento.

■ Concetti di base della programmazione a oggetti

Limitazioni dei tipi derivati. Il concetto di tipo di dato astratto.

■ Programmare in C++

Classi. Operatori con oggetti classe. Altre proprietà delle classi. Classi per l'ingresso e per l'uscita.

■ Progettare ed implementare tipi di dato astratti

Alcuni tipi di dato comuni con le classi: Liste, Code, Pile.

A-5

Definizione di informatica

Informatica (definizione informale): è la scienza della rappresentazione e dell'elaborazione dell'informazione

Informatica (definizione formale dell'Association for Computing Machinery - ACM): è lo studio sistematico degli algoritmi che descrivono e trasformano l'informazione, la loro teoria e analisi, il loro progetto, e la loro efficienza, realizzazione e applicazione.

Algoritmo: sequenza precisa e finita di operazioni, comprensibili e perciò eseguibili da uno strumento informatico, che portano alla realizzazione di un compito.

Esempi di algoritmi:

- Istruzioni di montaggio di un elettrodomestico
- Somma in colonna di due numeri
- Bancomat

A-6

Compito dell'esperto informatico: data la descrizione di un problema, produrre algoritmi (cioè capire la sequenza di passi che portano alla soluzione del problema) e codificarli in programmi.

- La descrizione di un problema non fornisce in generale un modo per risolverlo.
- La descrizione del problema deve essere chiara e completa.

Calcolatori Elettronici come esecutori di algoritmi: gli algoritmi vengono descritti tramite programmi, cioè sequenze di istruzioni scritte in un opportuno linguaggio comprensibile al calcolatore.

A-7

Algoritmo (1)

Algoritmo: sequenza precisa (non ambigua) e finita di operazioni, che porta alla realizzazione di un compito.

Le operazioni utilizzate appartengono ad una delle seguenti categorie:

1. Operazioni sequenziali

Realizzano una singola azione. Quando l'azione è terminata passano all'operazione successiva.

2. Operazioni condizionali

Controllano una condizione. In base al valore della condizione, selezionano l'operazione successiva da eseguire.

3. Operazioni iterative

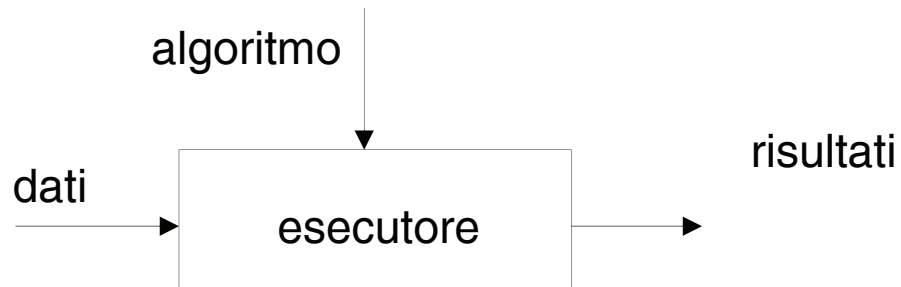
Ripetono l'esecuzione di un blocco di operazioni, finchè non è verificata una determinata condizione.

A-8

Algoritmo (2)

L'esecuzione delle azioni nell'ordine specificato dall'algoritmo consente di risolvere il problema.

Risolvere il problema significa produrre risultati a partire da dati in ingresso



L'algoritmo deve essere applicabile ad un qualsiasi insieme di dati in ingresso appartenenti al dominio di definizione dell'algoritmo (se l'algoritmo si applica ai numeri interi deve essere corretto sia per gli interi positivi che per gli interi negativi)

A-9

Algoritmo (3)

Calcolo equazione $ax+b = 0$

- leggi i valori di a e di b
- calcola $-b$
- dividi quello che hai ottenuto per a e chiama x il risultato
- stampa x

Calcolo del massimo fra due numeri

- leggi i valori di a e di b
- **se** $a > b$ stampa a **altrimenti** stampa b

A-10

Algoritmo (4)

Calcolo del massimo di un insieme

- scegli un elemento come massimo provvisorio *max*
- per ogni elemento *i* dell'insieme:
 - se** $i > max$ eleggi *i* come nuovo massimo provvisorio *max*
- il risultato è *max*

Stabilire se una parola *P* precede alfabeticamente una parola *Q*.
Ipotesi: *P* e *Q* di uguale lunghezza ≥ 1

leggi *P* e *Q*; **inizializza** trovato a 0

- **ripeti finché** (trovato vale 0 e lettere non finite)
 - se** prima lettera di *P* < prima lettera di *Q*
 - allora** scrivi vero; trovato = 1;
 - altrimenti se** prima lettera di *P* > prima lettera di *Q*
 - allora** scrivi falso; trovato = 1;
 - altrimenti** toglì da *P* e da *Q* la prima lettera
- **se** trovato vale 0 **allora** scrivi falso

A-11

Algoritmo (5)

Eseguibilità: ogni azione deve essere eseguibile dall'esecutore in un tempo finito

Non-ambiguità: ogni azione deve essere univocamente interpretabile dall'esecutore

Finitezza: il numero totale di azioni da eseguire, per ogni insieme di dati in ingresso, deve essere finito

Algoritmi equivalenti

- ◆ hanno lo stesso dominio di ingresso
- ◆ hanno lo stesso dominio di uscita
- ◆ in corrispondenza degli stessi valori del dominio di ingresso producono gli stessi valori del dominio di uscita
- ◆ Due algoritmi equivalenti
 - Forniscono lo stesso risultato, ma possono avere diversa efficienza e possono essere profondamente diversi

A-12

Algoritmo (6)

Esempio: calcolo del Massimo Comun Divisore (MCD) fra due interi M e N

Algoritmo 1

- Calcola l'insieme A dei divisori di M
- Calcola l'insieme B dei divisori di N
- Calcola l'insieme C dei divisori comuni
- il massimo comun divisore è il massimo divisore contenuto in C

A-13

Algoritmo (7)

Algoritmo 2 (di Euclide)

Se due numeri, m e n, sono divisibili per un terzo numero, x, allora anche la loro differenza è divisibile per x.

Per dimostrarla, si può utilizzare la proprietà distributiva.

Supponiamo $m > n$.

$$m = kx$$

$$n = hx$$

$$m - n = kx - hx = x(k - h)$$

Dunque si può dire che: $\text{MCD}(m, n) = \text{MCD}((m - n), n)$

Algoritmo

- **ripeti finché** ($M \neq N$):
 - **se** $M > N$, sostituisci a M il valore $M - N$
 - **altrimenti** sostituisci a N il valore $N - M$
- il massimo comun divisore corrisponde a M (o a N)

A-14

Proprietà essenziali degli algoritmi:

Correttezza:

- un algoritmo è corretto se esso perviene alla soluzione del compito cui è preposto, senza difettare in alcun passo fondamentale.

Efficienza:

- un algoritmo è efficiente se perviene alla soluzione del compito cui è preposto nel modo più veloce possibile, compatibilmente con la sua correttezza.

A-15

Programmazione

La formulazione testuale di un algoritmo in un linguaggio comprensibile ad un calcolatore è detta PROGRAMMA.

Ricapitolando, per risolvere un problema:

- Individuazione di un procedimento risolutivo
- Scomposizione del procedimento in un insieme ordinato di azioni – ALGORITMO
- Rappresentazione dei dati e dell'algoritmo attraverso un formalismo comprensibile al calcolatore: LINGUAGGIO DI PROGRAMMAZIONE

A-16

Linguaggi di Programmazione (1)

Perché non usare direttamente il linguaggio naturale?

Il LINGUAGGIO NATURALE è un insieme di parole e di regole per combinare tali parole che sono usate e comprese da una comunità di persone

- non evita le ambiguità
- non si presta a descrivere processi computazionali automatizzabili

Occorre una nozione di linguaggio più precisa.

Un LINGUAGGIO di programmazione è una notazione formale che può essere usata per descrivere algoritmi.

Si può stabilire quali sono gli elementi linguistici primitivi, quali sono le frasi lecite e se una frase appartiene al linguaggio.

A-17

Linguaggi di Programmazione (2)

Un linguaggio è caratterizzato da:

SINTASSI - insieme di regole formali per la scrittura di programmi, che fissano le modalità per costruire frasi corrette nel linguaggio

SEMANTICA - insieme dei significati da attribuire alle frasi (sintatticamente corrette) costruite nel linguaggio

- a parole (poco precisa e ambigua)
- mediante azioni (semantica operativa)
- mediante funzioni matematiche (semantica denotazionale)
- mediante formule logiche (semantica assiomatica)

A-18

Definizione di linguaggio

Alfabeto V (lessico)

- insieme dei simboli con cui si costruiscono le frasi

Universo linguistico V^*

- insieme di tutte le frasi (sequenze finite) di elementi di V

Linguaggio L su alfabeto V

- un sottoinsieme di V^*

Come definire il sottoinsieme di V^* che definisce il linguaggio?

Specificando in modo preciso la sintassi delle frasi del linguaggio TRAMITE una **grammatica formale**

A-19

Grammatiche

Grammatica $G = \langle V, VN, P, S \rangle$

V insieme finito di simboli terminali (ossia entità **atomiche** e **predefinite**)

VN insieme finito di simboli non terminali (sono detti anche "categorie sintattiche")

P insieme finito di regole di produzione

S simbolo non terminale detto simbolo iniziale

Data una grammatica G, si dice Linguaggio LG generato da G l'insieme delle frasi di V

- Derivabili dal simbolo iniziale S
- Applicando le regole di produzione P

Le frasi di un linguaggio di programmazione vengono dette programmi di tale linguaggio.

A-20

Grammatica BNF (1)

GRAMMATICA BNF (Backus-Naur Form) è una grammatica le cui **regole di produzione** sono della forma

X

$A_1 A_2 \dots A_n$

dove **X** è un simbolo non terminale ed $A_1 A_2 \dots A_n$ è una sequenza di simboli (terminali oppure non terminali).

Una grammatica BNF definisce quindi un linguaggio sull'alfabeto terminale V mediante un meccanismo di derivazione (ossia di *risrittura*)

Si dice che A deriva da **X** se esiste una sequenza di derivazioni da **X** ad A

Altra regola di produzione (unica alternativa all'interno di un insieme dato):

X

one of $A_1 A_2 \dots A_n$ // one of è un costrutto del metalinguaggio. Indica che **X** può assumere **una unica alternativa** tra A_1, \dots, A_n

A-21

Grammatica BNF (2)

$G = \langle V, VN, P, S \rangle$

$V = \{\text{lupo, canarino, bosco, cielo, mangia, vola, canta, ., il, lo}\}$

$VN = \{\text{frase, soggetto, verbo, articolo, nome}\}$

$S = \text{frase}$

Notare che il punto è un simbolo terminale che in questo è stato inserito nell'alfabeto

Produzioni P

frase

soggetto verbo .

soggetto

articolo nome

articolo

one of

il lo

nome

one of

lupo canarino bosco cielo

verbo

one of

mangia vola canta

Esempio: derivazione della frase

"il lupo mangia."

frase -> **soggetto verbo.**

-> **articolo nome verbo.**

-> il **nome verbo.**

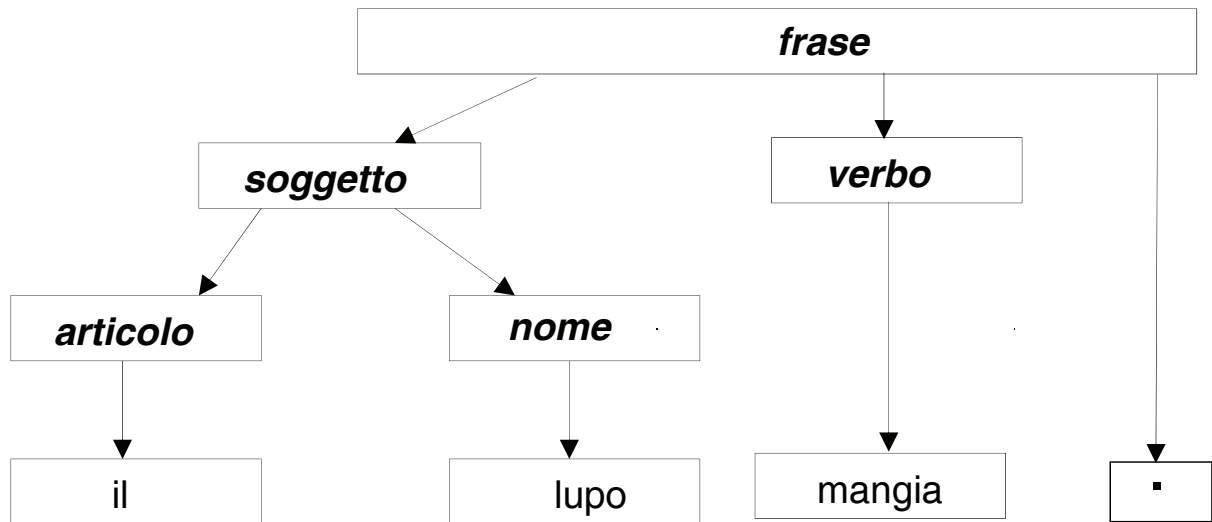
-> il lupo **verbo.**

-> il lupo mangia.

derivazione left-most

A-22

Albero sintattico



Albero sintattico: albero che esprime il processo di derivazione di una frase usando una data grammatica.

Esistono programmi per generare analizzatori sintattici per linguaggi descritti mediante BNF.

A-23

Altri costrutti del metalinguaggio: gli elementi opzionali (1/2)

Un altro costrutto molto comodo del metalinguaggio è la **presenza opzionale** di un simbolo (terminale o non terminale).

X

$A_1 \text{lopt} A_2$

indica che **X** può essere riscritto sia come

$A_1 A_2$

che come

A_2

in quanto A_1 è opzionale.

Analogamente

X

$A_1 A_2 \text{lopt}$

indica che **X** può essere riscritto sia come

$A_1 A_2$

che come

A_1

A-24

Altri costrutti del metalinguaggio: la sequenza (2/2)

Un altro costrutto molto comodo del metalinguaggio è la **sequenza**.

Sia **A** un simbolo terminale oppure non terminale.

Indicheremo con

A-seq

una sequenza di tali simboli (di lunghezza maggiore o uguale ad uno)

Esempio:

nome-seq sarà una sequenza di simboli non terminali **nome**:

nome //ok

nome nome ... nome // ok

Se non esistesse come costrutto predefinito del metalinguaggio, potremmo introdurre noi il nuovo simbolo non terminale **nome-seq** così:

nome-seq
nome nome-seqopt

Pertanto

lupo canarino lupo lupo

è coerente con la regola di produzione **nome-seq**

A-25

Esempio di grammatica per produrre numeri interi (1/2)

Come vengono scritti i numeri interi, usando le cifre arabe alle quali siamo abituati?

Ecco alcuni esempi di numeri interi scritti correttamente:

5898

-30

76

-234

Di contro, alcuni esempi di numeri interi scritti in maniera errata sono:

-12.76 (è un numero decimale, non intero!)

XVI (questo non va bene, perchè utilizza cifre romane)

meno45 (questo non va bene, perchè in un numero non possono comparire caratteri alfabetici)

100dodici (come sopra!)

Come si fa a dare un procedimento per produrre tutti i possibili numeri arabi interi corretti, ossia il linguaggio **L** dei numeri arabi interi?

Al solito, il modo più semplice è darne una grammatica BNF!

A-26

Esempio di grammatica per generare numeri interi (2/2)

Una possibile grammatica per i numeri interi

$V = \{-, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ // alfabeto, contenente i simboli terminali

$VN = \{cifra, cifra-seq, intero\}$ // simboli non terminali

$S = intero$ // simbolo non terminale iniziale

Regole di produzione P

cifra

one of

0 1 2 3 4 5 6 7 8 9

cifra-seq

cifra cifra-seq|opt

intero

-|opt cifra-seq

Notare che il '-' è opzionale e può comparire solo all'inizio

Esempi:

-23

4506

0023 // equivale a 23

-067 // equivale a -67

Con la grammatica introdotta non abbiamo univocità della rappresentazione, però abbiamo la garanzia di costruire/accettare solo numeri interi validi

Numeri come -56 oppure -87-4 oppure 634- non verranno **nè prodotti, nè accettati**.

A-27

Che aspetto avrà la grammatica per il linguaggio C++?

Diamo ora un piccolo assaggio di come potrebbe essere fatta una grammatica per il linguaggio C++ (la daremo più avanti)

```
int main(){  
    istruzione-seq  
}
```

istruzione-seq
istruzione istruzione-seq|opt

istruzione
one of
definizione // *int a = 3;*
selezione // *if (a == 5) ...*
iterazione // *while (a < 10) ...*

Le istruzioni C++ le vedremo più avanti, una ad una. *Don't panic!*

A-28

Il più semplice programma C++

Il seguente è un semplicissimo programmino C++ che visualizza a video la sequenza di caratteri "Ciao mondo!".

Si noti che in un programma C++ tutto quello che segue i due caratteri "//" viene ignorato dal compilatore (dai due caratteri fino alla fine della stessa riga).

Tali sequenze di caratteri che iniziano per "//" prendono il nome di **commenti**.

Nel seguito i commenti sono stati **colorati di verde**, per facilitare la lettura del codice. Durante l'esecuzione tutto avverrà come in assenza dei commenti.

```
#include <iostream> // queste prime due istruzioni servono per
using namespace std; // poter usare cin e cout (le due
                      // istruzioni per leggere da tastiera
                      // e per stampare a video)

int main(){

    cout << "Ciao mondo! " << endl;

    return 0; // "return" serve a terminare il programma
}
```

A-29

Struttura del generico programma C++

La struttura che avranno tutti i nostri programmi sarà sempre la stessa, ossia la seguente:

```
#include <iostream>
using namespace std;
int main(){

    <istruzione 1>
    <istruzione 2>
    ...
    <istruzione n>

    return 0;
}
```

Un programma C++ sorgente è una **sequenza di caratteri** che deve essere salvata **su disco** come file di testo, assegnandogli un **nome**

Esempio:

programma1.cpp // è importante usare l'estensione .cpp per i sorgenti C++

A-30

Sintassi e semantica

Scrivere un programma sintatticamente corretto non implica che il programma faccia quello per cui è stato scritto

La frase “il lupo vola” è sintatticamente corretta ma non è semanticamente corretta (non è significativa)

```
// Somma di due numeri interi inseriti da tastiera
#include <iostream>
using namespace std;
int main(){
    int a, b;
    cout << "Immettere due numeri " << endl;
    cin >> a;
    cin >> b;
    int c = a + b;
    // NB: int c = a + a; errore semantico, non sintattico
    cout << "Somma: " << c << endl;
    return 0;
}
```

A-31

Approccio compilato

Sviluppo di un programma (approccio compilato):

- 1) editing: scrivere il testo e memorizzarlo su supporti di memoria permanenti (*hard disk*)
- 2) compilazione e linking (*il ruolo del linker lo vedremo più avanti*)
- 3) esecuzione

Compilatore&Linker: traduce il programma sorgente in programma **eseguibile**

- ◆ ANALISI programma sorgente
 - analisi lessicale
 - analisi sintattica
- ◆ TRADUZIONE
 - generazione del codice
 - ottimizzazione del codice

Esempi: C, C++, Fortran, Rust, Go, ...

A-32

Sviluppo di un programma (approccio interpretato):

- editing: scrivere il testo e memorizzarlo su supporti di memoria permanenti
- interpretazione
 - ◆ ANALISI programma sorgente
 - analisi lessicale
 - analisi sintattica
 - ◆ ESECUZIONE

ESEMPLI: Python, Matlab, Javascript, ...

In questo corso prenderemo in considerazione solo il linguaggio C++, e dunque, **solo l'approccio compilato**.

A-33

I tre passi su CLion: Editing, Compilazione ed Esecuzione

A laboratorio verrà utilizzato l'ambiente per la programmazione C++ denominato **CLion**:



CLion è un programma, dotato di interfaccia grafica, che permette di:

1. effettuare l'editing del file sorgente contenente il programma C++
2. effettuare la compilazione (ed il *linking*, come vedremo più avanti)
3. nel caso non vi siano errori di compilazione, permette di lanciare il programma (ossia di caricarlo in memoria RAM e porlo in esecuzione)

A-34

Durante il primo laboratorio verrete assistiti nella:

- Creazione di un programma C++ da dentro Clion e a salvarlo su file
- Compilarlo e porlo in esecuzione

Nello stesso laboratorio vi verranno insegnate altre funzionalità di base del programma CLion.

I programmi come CLion sono definiti IDE (**Integrated Development Environment**), che viene tradotto in *ambiente di sviluppo (software) integrato*.

CLion per funzionare ha ovviamente bisogno del compilatore C++.

In questo corso utilizzeremo il compilatore **MinGW**, in ambiente Windows, perchè può essere scaricato gratuitamente da internet.

Inoltre Clion ha bisogno di un ulteriore programma: **CMAKE**.

A-35

Rappresentazione dell'informazione

caratteri, naturali, interi e reali

A-36

Rappresentazione dell'Informazione (1/2)

L'informazione è qualcosa di astratto.
Per poterla manipolare bisogna rappresentarla.

In un calcolatore i vari tipi di informazione (testi, figure, numeri, musica,...) si rappresentano per mezzo di sequenze di bit (cifre binarie).

Bit è l'abbreviazione di **Binary digIT**, ossia *cifra binaria*.

Il **bit** è l'unità di misura elementare dell'informazione, ma anche la base del sistema numerico utilizzato dai computer.

Può assumere soltanto due valori: **0** oppure **1**.

Byte è l'unità di misura dell'informazione che corrisponde ad 8 bit.

A-37

Rappresentazione dell'Informazione (2/2)

Quanta informazione può essere contenuta in una sequenza di n bit?
L'informazione corrisponde a tutte le possibili disposizioni con ripetizione di due oggetti (0 ed 1) in n caselle (gli n bit), ossia 2^n

Esempio: $n=3$

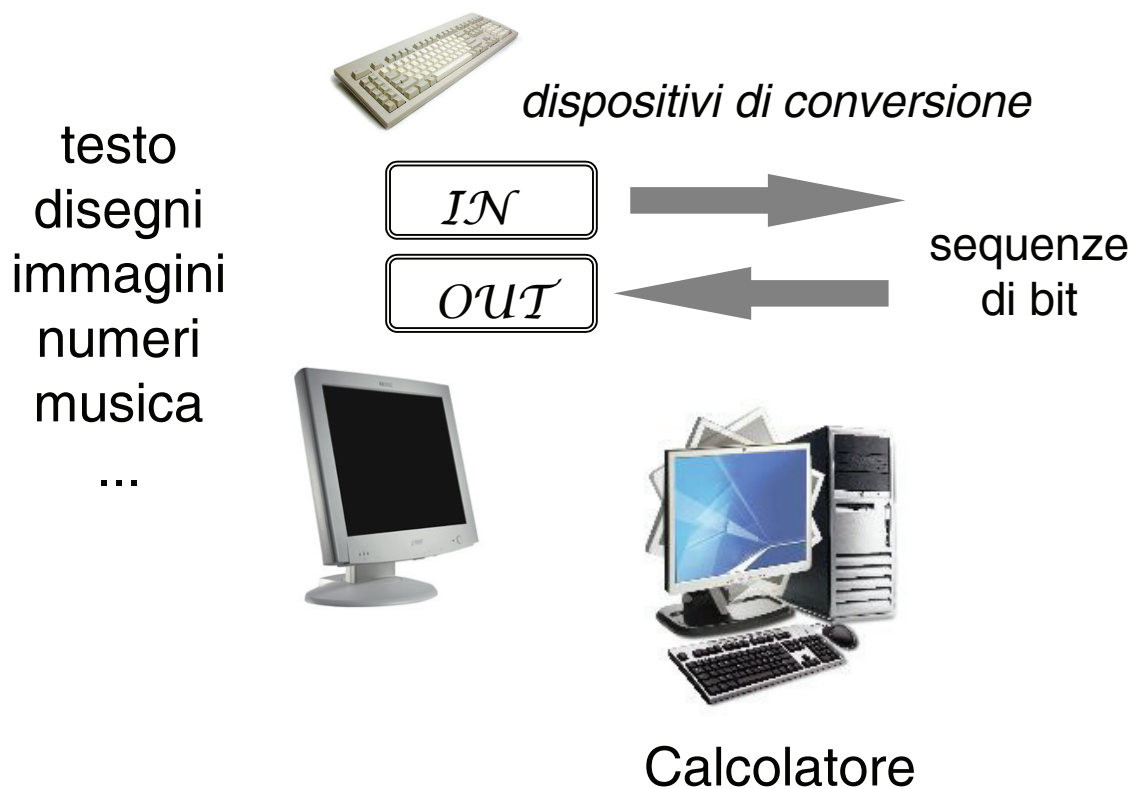
000
001
010
011
100
101
110
111

ATTENZIONE: Una stessa sequenza di bit può rappresentare informazione differente.

Per esempio 01000001 rappresenta

- il numero naturale 65
- il carattere 'A'
- il colore di un puntino sullo schermo
- ...

A-38



A-39

Rappresentazione del testo (1/2)

Codifica ASCII (American Standard Code for Information Interchange)
Standard su 7 bit (il primo bit del byte sempre 0)

Sequenze	Caratteri
00110000	0
00110001	1
00110010	2
...	...
00111001	9
...	...
01000001	A
01000010	B
01000011	C
...	...
01100001	a
01100010	b
...	...

01000011
01100001
01110010
01101111
00100000
01100001
01101110
01101001
01100011
01101111
00101100

Caro amico,

A-40

Rappresentazione del testo (2/2)

Codifica ASCII: utilizza 7 bit Eccone la tabella di corrispondenza

3 cifre più significative

000	001	010	011	100	101	110	111	
NUL	DLE	SP	0	@	P	`	p	0000
SOH	XON	!	1	A	Q	a	q	0001
STX	DC2	"	2	B	R	b	r	0010
ETX	XOFF	#	3	C	S	c	s	0011
EQT	DC4	\$	4	D	T	d	t	0100
ENQ	NAK	%	5	E	U	e	u	0101
ACK	SYN	&	6	F	V	f	v	0110
BEL	ETB	'	7	G	W	g	w	0111
BS	CAN	(8	H	X	h	x	1000
HT	EM)	9	I	Y	i	y	1001
LF	SUB	*	:	J	Z	j	z	1010
VF	ESC	+	;	K	[k	{	1011
FF	FS	,	<	L	\	l		1100
CR	GS	-	=	M]	m	}	1101
SO	RS	.	>	N	^	n	~	1110
SI	US	/	?	O	_	o	DEL	1111

↑
4 cifre meno
significative

La **codifica ASCII estesa** contiene ulteriori 128 caratteri, per un totale di 256 caratteri.

In questa codifica ogni carattere richiede 8 bit (ossia 1 byte)

(per i caratteri della codifica ASCII non estesa all'interno della codifica ASCII estesa *il bit più significativo vale zero*).

NB: Una codifica alternativa per i caratteri è quella UTF-8 dello standard **UNICODE**, in cui i caratteri hanno una occupazione variabile, di 1, 2 o 4 byte. Il Linguaggio C++ **non supporta** in maniera nativa la codifica UTF-8, **ma solamente la codifica ASCII estesa** (quella su 8 bit).

A-41

Rappresentazione dei numeri naturali (1/16)

Base dieci

- ◆ Cifre: 0, 1, 2, 3, 4, 6, 7, 8, 9
- ◆ Rappresentazione posizionale in base dieci:
 $(307)_{dieci}$ *significa* $3 \times 10^2 + 0 \times 10^1 + 7 \times 10^0$

Base due

- ◆ Cifre: 0, 1
- ◆ Rappresentazione posizionale in base due:
 $(11001)_{due}$ *significa* $1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
 pertanto $(11001)_{due}$ corrisponde a 25 in base dieci

A-42

Rappresentazione dei numeri naturali (2/16)

Data una base $\beta \geq \text{due}$

Ogni numero naturale N minore di β ($N < \beta$) è associato ad un simbolo elementare detto **cifra**

BASE	CIFRE
due	0 1
cinque	0 1 2 3 4
otto	0 1 2 3 4 5 6 7
sedici	0 1 2 3 4 5 6 7 8 9 A B C D E F

Cifre addizionali della base 16	Peso intrinseco
A	10
B	11
C	12
D	13
E	14
F	15

- Ad ogni cifra ha associato un peso intrinseco.
- Poi il peso intrinseco verrà «amplificato» da una potenza della base, a seconda della posizione della cifra (si veda la formula della sommatoria nella prossima slide).

A-43

Rappresentazione dei numeri naturali (3/16)

I numeri naturali maggiori o uguali a β possono essere rappresentati in maniera unica da una sequenza di cifre secondo la **rappresentazione posizionale**.

Se un numero naturale $N \geq \beta$ è rappresentato in base β dalla sequenza di cifre:

$$a_{p-1} a_{p-2} \dots a_1 a_0$$

allora N può essere espresso come segue:

$$N = \sum_{i=0}^{p-1} a_i \beta^i = a_{p-1} \beta^{p-1} + a_{p-2} \beta^{p-2} + \dots + a_2 \beta^2 + a_1 \beta + a_0$$

dove a_0 è detta «cifra meno significativa» e a_{p-1} «cifra più significativa»

Chiamiamo questa formula «formula della sommatoria».

Il fatto che, dato un naturale N , esista e sia unica la sequenza $a_{p-1} \dots a_0$ (avendo rimosso eventuali zeri all'inizio) è dovuto ad un teorema noto con il nome di:
Teorema Fondamentale della Rappresentazione dei Numeri Naturali.

A-44

Rappresentazione dei numeri naturali (4/16)

Nella formula della sommatoria vengono coinvolte le potenze della base β .
Riportiamo di seguito le prime potenze nel caso della base $\beta = 2$.

Potenza di due	Valore in base dieci	Denominazione
2^0	1	
2^1	2	
2^2	4	
2^3	8	
2^4	16	
2^5	32	
2^6	64	
2^7	128	
2^8	256	
2^9	512	
2^{10}	1024	1 Kilo
...	...	
2^{20}	1048576	1 Mega
2^{30}	1073741824	1 Giga
2^{32}	4294967296	4 Giga

Osservazione 1:

La rappresentazione in base due di una qualunque potenza di due si può ottenere immediatamente utilizzando un uno seguito da p zeri, se p è la potenza:

$$2^p \leftrightarrow \underbrace{(100\dots 00)}_p_{\text{due}}$$

Esempio:

La rappresentazione di 2^5 in base 2 è 100000
(ossia trentadue in base dieci)

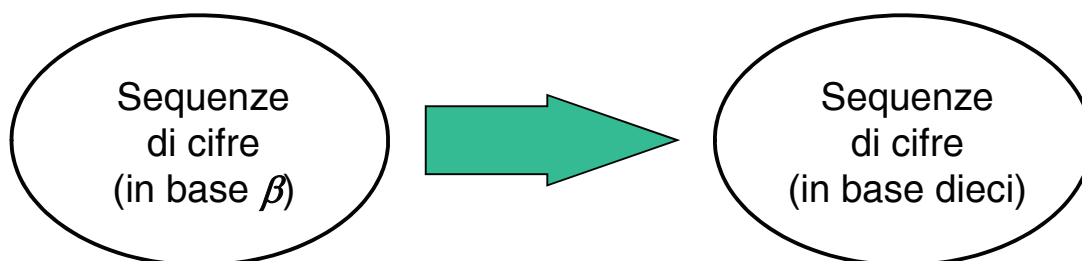
Osservazione 2: Con dieci bit in base due si possono generare 1 kilo sequenze distinte di bit (per la precisione, 1024 *disposizioni con ripetizione*).

Una memoria RAM con 2^{34} celle di memoria avrà 16 Giga locazioni (*memoria da 16 Giga Byte*)

A-45

Rappresentazione dei numeri naturali (5/16)

Data una sequenza di cifre in base β ,
a quale numero naturale corrisponde?



$$a_{p-1} a_{p-2} \dots a_1 a_0$$

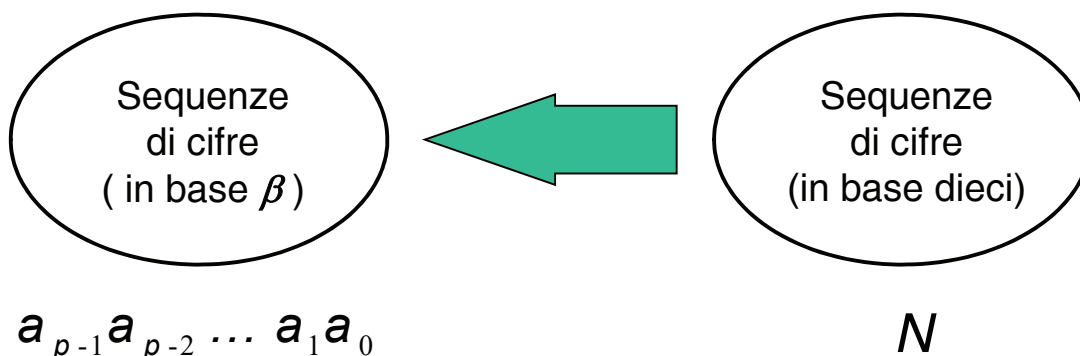
$$N ?$$

$$\begin{array}{l}
 (656)_{\text{otto}} \\
 (A03)_{\text{sedici}} \\
 (1101)_{\text{due}}
 \end{array}
 \begin{array}{l}
 \xrightarrow{\quad} 6 \cdot 8^2 + 5 \cdot 8^1 + 6 \cdot 8^0 \\
 \xrightarrow{\quad} 10 \cdot 16^2 + 0 \cdot 16^1 + 3 \cdot 16^0 \\
 \xrightarrow{\quad} 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0
 \end{array}
 \begin{array}{l}
 \xrightarrow{\quad} (430)_{\text{dieci}} \\
 \xrightarrow{\quad} (2563)_{\text{dieci}} \\
 \xrightarrow{\quad} (13)_{\text{dieci}}
 \end{array}$$

A-46

Rappresentazione dei numeri naturali (6/16)

Data la base β ed un numero naturale N ,
trovare la sequenza di cifre che rappresenta N in base β



A-47

Rappresentazione dei numeri naturali (7/16)

Esempio: da base 10 a base 2

$N = 23$

inizio

	QUOZIENTE	RESTO	Rappresentazione
	23	-	
div 2	11	1	$11 \cdot 2 + 1$
div 2	5	1	$((5 \cdot 2) + 1) \cdot 2 + 1$
div 2	2	1	$((((2 \cdot 2) + 1) \cdot 2) + 1) \cdot 2 + 1$
div 2	1	0	$(((((1 \cdot 2) + 0) \cdot 2) + 1) \cdot 2) + 1) \cdot 2 + 1$
div 2	0	1	$(((((0 \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 1) \cdot 2) + 1$

fine

$(10111)_{\text{due}}$
 $a_4 a_3 a_2 a_1 a_0$

che infatti in base dieci vale proprio $1 \cdot 2^4 + \dots + 1 = (23)_{\text{dieci}}$
(controprova, effettuata mediante algoritmo della sommatoria)

A-48

Rappresentazione dei numeri naturali (8/16)

Sia **mod** il resto e **div** il quoziente della divisione intera

Esempio:

$$23 \text{ div } 2 = 11$$

$$23 \text{ mod } 2 = 1$$

Procedimento "Div & Mod"

Se $N = 0$ porre $a_0 = 0$; \Rightarrow fine

Altrimenti: porre $q_0 = N$ e poi eseguire la seguente procedura iterativa:

$$q_1 = q_0 \text{ div } \beta \quad a_0 = q_0 \text{ mod } \beta$$

$$q_2 = q_1 \text{ div } \beta \quad a_1 = q_1 \text{ mod } \beta$$

...

$$q_p = q_{p-1} \text{ div } \beta \quad a_{p-1} = q_{p-1} \text{ mod } \beta$$

NB: Il risultato della **mod** è sempre una cifra valida in base β , perché restituisce sempre un numero fra 0 e $\beta-1$ (estremi inclusi).

fino a quando q_p diventa uguale a 0

Il procedimento si arresta quando $q_p = 0$ (più precisamente subito dopo aver calcolato a_{p-1}). Inoltre p è proprio il numero di cifre necessario per rappresentare N in base β

A-49

Rappresentazione dei numeri naturali (9/16)

Inizio $q_0 = N$

	QUOZIENTE	RESTO
	$q_0 = N$	-
div β	$q_1 = q_0 / \beta$	a_0
div β	$q_2 = q_1 / \beta$	a_1
div β	$q_3 = q_2 / \beta$	a_2
div β	$q_4 = q_3 / \beta$	a_3
div β	$q_5 = q_4 / \beta$	a_4
div β	$q_6 = q_5 / \beta$	a_5
	$q_7 = 0$	a_6

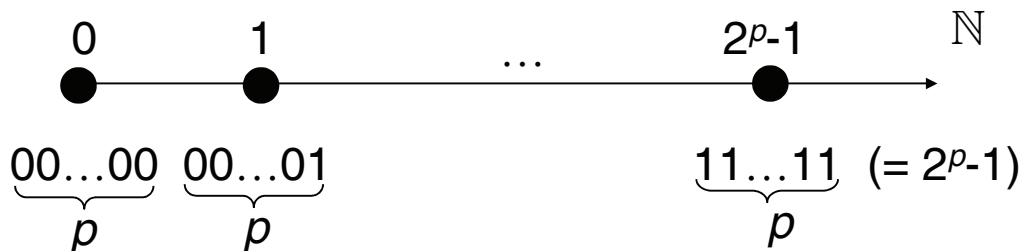
fine

$$N = a_6 \cdot \beta^6 + a_5 \cdot \beta^5 + a_4 \cdot \beta^4 + a_3 \cdot \beta^3 + a_2 \cdot \beta^2 + a_1 \cdot \beta^1 + a_0 \cdot \beta^0$$

A-50

Rappresentazione dei numeri naturali (10/16)

Intervallo di rappresentabilità con p bit



p	Intervallo $[0, 2^p-1]$
8	$[0, 255]$
16	$[0, 65535]$
32	$[0, 4294967295]$

NB: per la generica base β ,
l'intervallo di rappresentabilità
con p cifre è $[0, \beta^p-1]$

A-51

Rappresentazione dei numeri naturali (11/16)

Calcolatore lavora con un numero finito di bit

- ◆ Supponiamo che $p = 16$ bit
- ◆ $A = 0111011011010101$ (30421)
 $B = 1010100001110111$ (43127)
- ◆ Poiché $A + B$ (73552) è maggiore di 2^p-1 (65535), quindi non è rappresentabile su p bit, si dice che la somma ha dato luogo ad un traboccamento (**overflow**, in inglese)
- ◆ In generale sono sempre sufficienti $p+1$ bit per rappresentare la somma di due numeri su p bit (vale a dire: non ne servono di più)

$A = 1111111111111111$ (65535)

$B = 1111111111111111$ (65535)

11111111111111110 (131070)

A-52

Rappresentazione dei numeri naturali (12/16)

Somma fra due numeri naturali espressi in binario.

Per sommare 10011 e 101101, basta metterli in colonna allineandoli a destra (come si fa con i numeri in base 10) e poi tenere presenti le seguenti regole:

			genera un riporto	genera un riporto	eventuale riporto generatosi precedentemente
0 +	0 +	1 +	1 +	1 +	
0 =	1 =	0 =	1 =	1 =	
0	1	1	0	1	

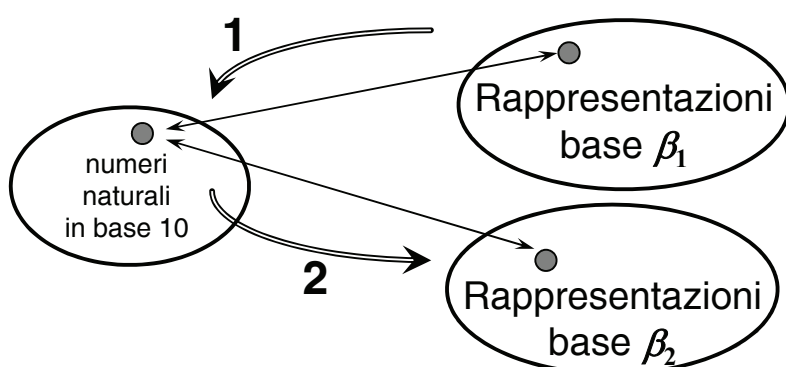
Complessivamente si legge così:
1+1+1 fa 1
con il riporto di 1

Esempio: calcolare la somma di 101100 e 111010

$$\begin{array}{r}
 1111 \leftarrow \text{riporti generati} \\
 101011 + \\
 111010 = \\
 \hline
 1100101
 \end{array}$$

A-53

Rappr. Naturali - Cambio di base (13/16)



Da base β_1 a base β_2

Trovare la rappresentazione in base 9 di $(1023)_{cinque}$

1) Trasformo in base 10 ed ottengo 138

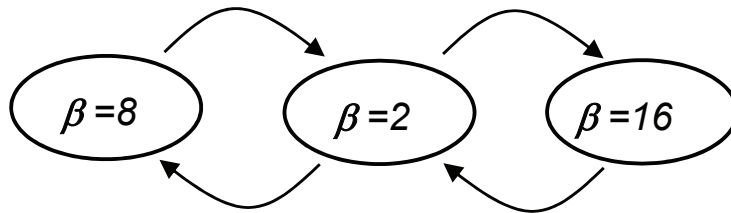
2) Applico il procedimento Div&Mod ed ottengo $(163)_{nove}$

A-54

Casi particolari (potenze di 2)

$\beta = 8$

0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111



$\beta = 16$

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

$(657)_{\text{otto}}$
 $(110\ 101\ 111)_{\text{due}}$

$(A03)_{\text{sedici}}$
 $(1010\ 0000\ 0011)_{\text{due}}$

$(FFE4)_{\text{sedici}}$
 $(1111\ 1111\ 1110\ 0100)_{\text{due}}$

A-55

Somma di potenze distinte della base (15/16)

Supponiamo ancora di dover rappresentare un numero naturale in base β , ma questa volta ce lo danno già espresso come somma di potenze della base β in questione.

Se le potenze coinvolte **sono tutte distinte**, allora trovarne la rappresentazione in base β è facilissimo, immediato. Altrimenti si consiglia di applicare il metodo generale, ossia prima ricavare N in base dieci e poi applicare l'algoritmo del Div&Mod.

Esempio: mi viene fornito il numero naturale $2^3+2^7+2^5$ e sono interessato a conoscerne la rappresentazione proprio in base due. Certamente potrei utilizzare la tecnica generale, e calcolare innanzitutto $N = 8+128+32 = (168)_{\text{dieci}}$. Applicando Div&Mod ottendo che la sua rappresentazione in base due è 10101000.

Avrei potuto trovare lo stesso risultato rappresentando in base due singolarmente le singole potenze, per poi sommarle tra loro in colonna:

```

1000+
10000000+
100000=
-----
10101000

```

NB: Se le potenze non hanno esponenti distinti non si può fare! Infatti, se per esempio dovessi trovare la rappresentazione di $2^3+2^7+2^5+2^3$, non avrei altra scelta che applicare il metodo generale!

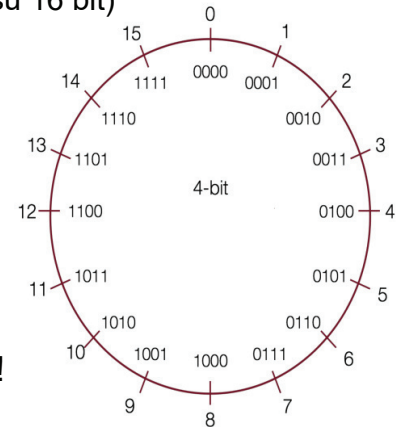
A-56

In C++ si possono facilmente definire numeri naturali su 32 bit (tipo «unsigned int») e su 16 bit (tipo «unsigned short int»). Per i numeri naturali su 64 bit si può provare ad usare il tipo «unsigned long int», ma non è detto che allochi un naturale su 64 bit (potrebbe allocarlo ancora su 32 bit).

```
int main(){
    unsigned short int n;
    cout<<sizeof(n)<<endl; // stampa 2 (ossia n viene rappresentato su 16 bit)

    n = 65535;
    cout<<n<<endl;        // stampa 65535

    ++n;
    cout<<n<<endl;        // stampa 0
}
```



NB: In C++ i numeri naturali **appaiono organizzati ad anello!**

In pratica nella maggior parte delle architetture hardware la somma è intesa come somma modulare (figura a destra).

*Attenzione però al fatto che questo comportamento **non fa parte dello standard.***

A-57

Numeri Interi (1/17)

Prima rappresentazione: *rappresentazione in modulo e segno*

Trasformazione $a \Rightarrow A$ (codifica)

Sia a (es. $a=+3$, $a=-3$, ...) il numero intero che si vuole rappresentare **in modulo e segno** su p bit. La rappresentazione A (es. 0011, 1011, ...) di a è data da:

$$A = a_{p-1} \dots a_0 = (\text{segno}_a, \text{ABS}_a)$$

dove ABS_a è la rappresentazione (come numero naturale) del valore assoluto di a su $p-1$ bit (in particolare ABS_a è rappresentato mediante i bit a_{p-2}, \dots, a_0), e segno_a è un unico bit che vale 0 se $a \geq 0$ e 1 se $a < 0$.

NB: ad $a=0$ corrispondono due rappresentazioni: $+\text{zero}$ (0,00...0) e $-\text{zero}$ (1,00...0)

Ad esempio, per $p = 4$ si ha:

$a=+3 \Rightarrow \text{segno}_a = 0$, $\text{ABS}_a = 011$ (naturale 3 rappresentato su 3 cifre) $\Rightarrow A=0011$
 $a=-3 \Rightarrow \text{segno}_a = 1$, $\text{ABS}_a = 011$ (naturale 3 rappresentato su 3 cifre) $\Rightarrow A=1011$

A-58

Prima rappresentazione: *rappresentazione in modulo e segno*

Trasformazione $A \Rightarrow a$ (decodifica)

Data una rappresentazione **A** di un intero **in modulo e segno** su p bit, come si risale all'intero **a** da esso rappresentato?

La rappresentazione **A** va vista come composta di due parti: $A = (\underbrace{a_{p-1}}_{\text{segno}}, \underbrace{a_{p-2} \dots a_0}_{\text{modulo}})$ dopodiché:

$$a = (a_{p-1} == 0) ? +ABS_a : -ABS_a$$

dove ABS_a è il naturale corrispondente ad $a_{p-2} \dots a_0$

Ad esempio, per $p = 4$ si ha:

A=0011 \Rightarrow viene visto come (0,011) $\Rightarrow a = +011$ (+tre)

A=1011 \Rightarrow viene visto come (1,011) $\Rightarrow a = -011$ (-tre)

NB: 0000 rappresenta +zero, mentre 1000 rappresenta -zero.

A-59

N	A	a
0	0000	+0
1	0001	+1
2	0010	+2
3	0011	+3
4	0100	+4
5	0101	+5
6	0110	+6
7	0111	+7
8	1000	-0
9	1001	-1
10	1010	-2
11	1011	-3
12	1100	-4
13	1101	-5
14	1110	-6
15	1111	-7

**Rappresentazione di interi
in modulo e segno su
calcolatore con $p=4$ bit**

numero negativo \Leftrightarrow bit più significativo
della rappresentazione uguale a 1
(zero rappresentato due volte)

A-60

Intervallo di rappresentabilità in modulo e segno su p bit
(doppia rappresentazione dello zero)

$$[-(2^{(p-1)}-1), +(2^{(p-1)}-1)]$$

- $p = 4$ [-7, +7]
- $p = 8$ [-127, +127]
- $p = 16$ [-32767, +32767]

NB: prima di applicare l'algoritmo per $a \Rightarrow A$ occorre verificare che a sia rappresentabile su p bit, altrimenti l'algoritmo conduce a risultati sbagliati.

Ad esempio:

tentando di rappresentare $a=-9$ su $p=4$ bit, si ottiene:

$A=(segno_a, ABS_a)$, dove $segno_a=1$ e $ABS_a = 9$

ma il naturale 9 (1001) non è rappresentabile su 3 bit!!

Seconda rappresentazione: *rappresentazione in complemento a 2* **Trasformazione $a \Rightarrow A$ (codifica)**

Sia a (es. $a=+3$, $a=-3$, ...) il numero intero che si vuole rappresentare **in complemento a 2** su p bit. La rappresentazione A (es. 0011, 1101, ...) di a è data da:

$$A = a_{p-1} \dots a_0 = (a \geq 0) ? ABS_a : (2^p - ABS_a)$$

dove sia ABS_a che $(2^p - ABS_a)$ sono rappresentati in base due come numeri naturali su p bit.

Ad esempio, per $p = 4$ si ha:

$a = 0 \Rightarrow ABS_a = 0$, e il naturale 0 ha rappr. 0000 su 4 bit $\Rightarrow A = 0000$

$a = 1 \Rightarrow ABS_a = 1$, e il naturale 1 ha rappr. 0001 su 4 bit $\Rightarrow A = 0001$

$a = 7 \Rightarrow ABS_a = 7$, e il naturale 7 ha rappr. 0111 su 4 bit $\Rightarrow A = 0111$

$a = -1 \Rightarrow ABS_a = 1$, $16-1=15$, dove il naturale 15 ha rappr. 1111 su 4 bit $\Rightarrow A = 1111$

$a = -2 \Rightarrow ABS_a = 2$, $16-2=14$, dove il naturale 14 ha rappr. 1110 su 4 bit $\Rightarrow A = 1110$

$a = -8 \Rightarrow ABS_a = 8$, $16-8=8$, dove il naturale 8 ha rappr. 1000 su 4 bit $\Rightarrow A = 1000$

NB: La rappresentazione in complemento a 2 è anche detta in *complemento alla base*. Infatti lo stesso procedimento può essere generalizzato per rappresentare interi in basi diverse da due.

Seconda rappresentazione: *rappresentazione in complemento a 2* Trasformazione $A \Rightarrow a$ (decodifica)

Data una rappresentazione $A = a_{p-1} \dots a_0$ di un intero in **complemento a due** su p bit, come si risale all'intero a da esso rappresentato?

$$a = (a_{p-1} == 0) ? +A : -(2^p - A)$$

dove sia A che $(2^p - A)$ vengono visti come naturali su p bit.

Ad esempio, per $p = 4$ si ha:

$A = 0000 \Rightarrow a = 0$ (zero)

$A = 0001 \Rightarrow a = +1$ (+uno)

$A = 0111 \Rightarrow a = +7$ (+sette)

$A = 1000 \Rightarrow 16-8 = 8 \Rightarrow a = -8$ (-otto)

$A = 1001 \Rightarrow 16-9 = 7 \Rightarrow a = -7$ (-sette)

$A = 1111 \Rightarrow 16-15=1 \Rightarrow a = -1$ (-uno)

A-63

N	A	a
0	0000	0
1	0001	+1
2	0010	+2
3	0011	+3
4	0100	+4
5	0101	+5
6	0110	+6
7	0111	+7
8	1000	-8
9	1001	-7
10	1010	-6
11	1011	-5
12	1100	-4
13	1101	-3
14	1110	-2
15	1111	-1

**Rappresentazione di interi
in complemento a due su
calcolatore con $p=4$ bit**

Anche in questo caso:

*numero negativo \Leftrightarrow bit più significativo
della rappresentazione uguale a 1*

Inoltre, a differenza della
rappresentazione in modulo e segno,
non viene sprecata nessuna
rappresentazione (lo zero è
rappresentato una volta sola)

A-64

Intervallo di rappresentabilità in complemento a 2 su p bit

$$[-(2^{(p-1)}), +(2^{(p-1)} - 1)]$$

- $p = 4$ [-8 , +7]
- $p = 8$ [-128 , +127]
- $p = 16$ [-32768 , +32767]

NB: prima di applicare l'algoritmo per $a \Rightarrow A$ occorre verificare che a sia rappresentabile su p bit, altrimenti l'algoritmo conduce a risultati sbagliati.

Ad esempio, tentando di rappresentare $a = -9$ su $p = 4$ bit, si ottiene:
 $A = (2^{4-9}) = 16 - 9 = 7 \Rightarrow 0111$, **che è un risultato sbagliato!**

Infatti -9 **non è rappresentabile** su 4 bit, ne servono almeno 5!

NB2: Che il risultato fosse sbagliato si poteva dedurre dal fatto che la rappresentazione del negativo -9 iniziava per 0 (0111 è infatti la rappresentazione di $a = +7$ su 4 bit!).

NB3: su 5 bit, $a = -9$ ha rappresentazione $(2^{5-9}) = 23 \Rightarrow A = 10111$

A-65

In complemento a due, il numero intero $a = -1$ avrà sempre come rappresentazione una sequenza di p bit ad 1.

Esempio: la rappresentazione di -1 su 16 bit in complemento a due è:

$$A = 1111111111111111$$

Mentre -1 su $p = 32$ bit avrà rappresentazione $A = \underbrace{11111111 \dots 11111111}_{32 \text{ bit ad uno}}$

A-66

Numeri Interi (10/17)

Supponiamo di dover rappresentare $a = -14$ in complemento a due su cinque bit. Osservo che è rientra nell'intervallo di rappresentabilità ed applico l'algoritmo di codifica: $2^5 - \text{ABS}_a = 32 - 14 = 18$. Applicando Div&Mod ottengo $A = 10010$.

Un modo alternativo per trovare la rappresentazione A , consiste nell'applicare il «metodo dei tre passi»:

Passo 1: Rappresentare il valore assoluto del numero in base 2 su $p=5$ bit

Passo 2: Invertire ciascun bit (questa operazione si chiama anche *complemento a uno*)

Passo 3: Sommargli uno (usando la somma in colonna).

Esempio:

Passo 1: Applico Div&Mod a +14 e ottengo 1110. Estendo su cinque bit: 01110

Passo 2: Inverto i bit e ottengo 10001

Passo 3: Ci sommo uno:

$$\begin{array}{r} 10001+ \\ 1= \\ \hline 10010 \end{array}$$

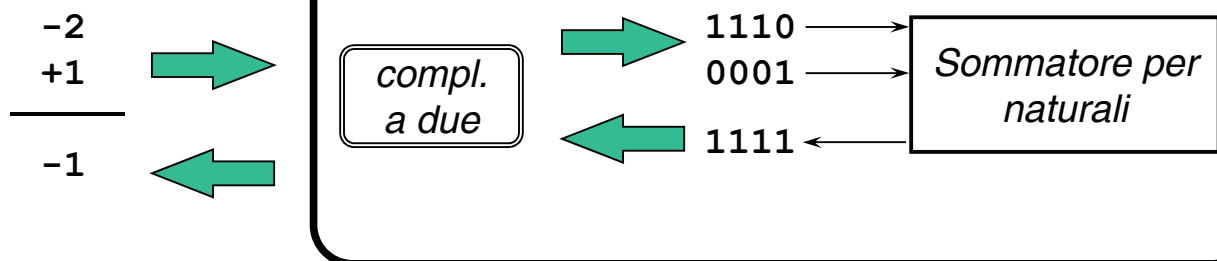
Si può dimostrare che i due metodi sono equivalenti.

Infatti quello che viene fatto al **Passo 2** equivale all'operazione matematica

$$(2^p - 1) - \text{ABS}_a$$

A-67

Numeri Interi (11/17)



Operazioni su numeri Operazioni sulle rappresentazioni

QUESTO è il motivo per cui i calcolatori rappresentano gli interi in complemento a due: **non occorre una nuova circuiteria per sommare e sottrarre numeri interi, viene utilizzata la stessa dei numeri naturali!**

A-68

Sommando due numeri interi si verifica un **overflow** quando i due numeri hanno lo stesso segno ed il risultato ha segno diverso

$7+5=12$

0	1	1	1	+	(+7)
0	1	0	1	=	(+5)
<hr style="border: 0.5px solid black;"/>					
1	1	0	0	=	(-4) overflow!

$7-1=6$

0	1	1	1	+	(+7)
1	1	1	1	=	(-1)
<hr style="border: 0.5px solid black;"/>					
1	0	1	1	0	(+6) corretto!

$-6-8=-14$

1	0	1	0	+	(-6)
1	0	0	0	=	(-8)
<hr style="border: 0.5px solid black;"/>					
1	0	0	1	0	(+2) overflow!

NB: L'eventuale (p+1)-esimo bit viene sempre scartato

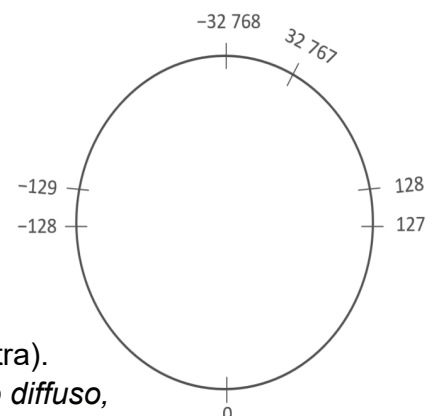
A-69

In C++ si possono facilmente definire numeri interi su 32 bit (tipo «int») e su 16 bit (tipo «short int»). Per i numeri interi su 64 bit si può provare ad usare il tipo «long int», ma non è detto che allochi un intero su 64 bit (potrebbe allocarlo ancora su 32 bit).

```
int main(){
    short int n = 0;
    cout<<sizeof(n)<<endl; // nella maggior parte delle architetture stampa 2
                           // (il che prova che è stato effettivamente rappresentato su 16 bit)
    n = 32767;
    cout<<n<<endl; // stampa a video 32367, come ci si aspetta

    n = n + 1;
    cout<<n <<endl; // stampa a video -32768
                   // questo è il «famoso» problema
                   // dell'INTEGER OVERFLOW

    return 0;
}
```



NB: In C++ gli interi **appaiono organizzati ad anello** (figura a destra).
 Attenzione però al fatto che questo comportamento, seppure molto diffuso,
 non fa parte dello standard.

A-70

Terza Rappresentazione: *rappresentazione con bias*

Trasformazione $a \Rightarrow A$ (codifica)

Sia a (es. $a=+3$, $a=-3$, ...) il numero intero che si vuole rappresentare *in rappresentazione con bias* su p bit.

La rappresentazione A (es. 1010, 0100, ...) di a è data da:

$$A = a_{p-1} \dots a_0 = a + (2^{p-1} - 1)$$

dove $a + (2^{p-1} - 1)$ è supposto essere non negativo e dunque viene rappresentato come un naturale su p bit. La quantità $(2^{p-1} - 1)$ è detta *bias* e $A = a + \text{bias}$

Ad esempio, per $p = 4$ si ha:

$a = 0 \Rightarrow 0 + (2^{4-1} - 1) = 7$,	e il naturale 7 ha rappr. 0111 su 4 bit $\Rightarrow A = 0111$
$a = 1 \Rightarrow 1 + (2^{4-1} - 1) = 8$,	e il naturale 8 ha rappr. 1000 su 4 bit $\Rightarrow A = 1000$
$a = 8 \Rightarrow 8 + (2^{4-1} - 1) = 15$,	e il naturale 15 ha rappr. 1111 su 4 bit $\Rightarrow A = 1111$
$a = -1 \Rightarrow -1 + (2^{4-1} - 1) = 6$,	e il naturale 6 ha rappr. 0110 su 4 bit $\Rightarrow A = 0110$
$a = -2 \Rightarrow -2 + (2^{4-1} - 1) = 5$,	e il naturale 5 ha rappr. 0101 su 4 bit $\Rightarrow A = 0101$
$a = -7 \Rightarrow -7 + (2^{4-1} - 1) = 0$,	e il naturale 0 ha rappr. 0000 su 4 bit $\Rightarrow A = 0000$

NB: come si vedrà nelle prossime slides, questa rappresentazione viene utilizzata nella rappresentazione dei numeri reali in virgola mobile

A-71

Terza Rappresentazione: *rappresentazione con bias*

Trasformazione $A \Rightarrow a$ (decodifica)

Sia A (es. 1010, 0100, ...) la rappresentazione di un numero intero nella *rappresentazione con bias* su p bit. Il numero intero a (es. $a=+3$, $a=-3$, ...) associato ad A è dato da:

$$a = A - (2^{p-1} - 1)$$

dove A viene visto come numero naturale su p bit. Dunque $a = A - \text{bias}$

Ad esempio, per $p = 4$ si ha:

$A = 0111 \Rightarrow 7 - (2^{4-1} - 1) = 0 \Rightarrow a = 0$
$A = 1000 \Rightarrow 8 - (2^{4-1} - 1) = 1 \Rightarrow a = 1$
$A = 1111 \Rightarrow 15 - (2^{4-1} - 1) = 8 \Rightarrow a = 8$
$A = 0110 \Rightarrow 6 - (2^{4-1} - 1) = -1 \Rightarrow a = -1$
$A = 0101 \Rightarrow 5 - (2^{4-1} - 1) = -2 \Rightarrow a = -2$
$A = 0000 \Rightarrow 0 - (2^{4-1} - 1) = -7 \Rightarrow a = -7$

NB: Lo zero viene rappresentato una sola volta (come accade in compl. a 2).

A-72

Intervallo di rappresentabilità nella *rappresentazione con bias*

$$[-(2^{(p-1)}+1), +2^{(p-1)}], \text{ ossia } [-bias, bias+1]$$

- $p = 4$ [-7 , +8] ($bias=7$)
- $p = 5$ [-15 , +16] ($bias=15$)
- $p = 7$ [-63 , +64] ($bias=63$)
- $p = 10$ [-511 , +512] ($bias=511$)
- $p = 14$ [-8191 , +8192] ($bias=8191$)

NB: prima di applicare l'algoritmo per $a \Rightarrow A$ occorre verificare che a sia rappresentabile su p bit, altrimenti l'algoritmo conduce a risultati sbagliati.

Ad esempio, tentando di rappresentare $a=-9$ su $p=4$ bit, si ottiene:

$$A = -9 + (2^{4-1} - 1) = -9 + 7 = -2, \text{ che non è un numero naturale!}$$

NB2: nella rappresentazione con bias i numeri **positivi** si possono distinguere immediatamente: iniziano per 1 (ossia hanno il bit più significativo ad 1).
Quelli **non positivi** (negativi o nulli), invece, iniziano per 0.

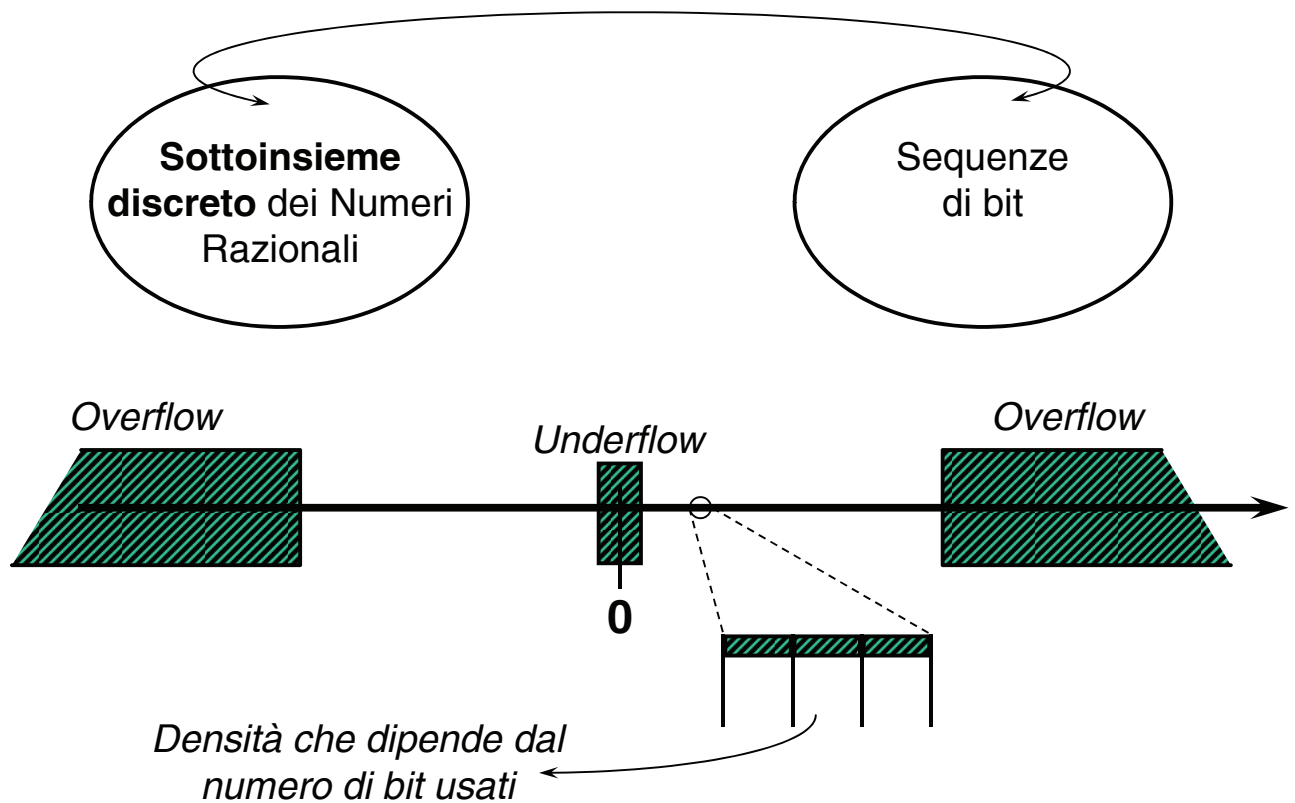
A-73

Rappresentazioni degli interi su $p=5$ bit messe a confronto

N	A	a _{ms}	a ^{c2}	a _{bias}
0	00000	+0	0	-15
1	00001	+1	+1	-14
2	00010	+2	+2	-13
3	00011	+3	+3	-12
4	00100	+4	+4	-11
5	00101	+5	+5	-10
6	00110	+6	+6	-9
7	00111	+7	+7	-8
8	01000	+8	+8	-7
9	01001	+9	+9	-6
10	01010	+10	+10	-5
11	01011	+11	+11	-4
12	01100	+12	+12	-3
13	01101	+13	+13	-2
14	01110	+14	+14	-1
15	01111	+15	+15	0
16	10000	-0	-16	+1
17	10001	-1	-15	+2
18	10010	-2	-14	+3
19	10011	-3	-13	+4
20	10100	-4	-12	+5
21	10101	-5	-11	+6
22	10110	-6	-10	+7
23	10111	-7	-9	+8
24	11000	-8	-8	+9
25	11001	-9	-7	+10
26	11010	-10	-6	+11
27	11011	-11	-5	+12
28	11100	-12	-4	+13
29	11101	-13	-3	+14
30	11110	-14	-2	+15
31	11111	-15	-1	+16

A-74

Numeri Reali



A-75

Numeri Reali – Virgola fissa (1/5)

- ☐ Si usa un numero fisso di bit per la parte intera ed un numero fisso di bit per la parte frazionaria.
- ☐ Sia r un numero reale da rappresentare. Di seguito, indichiamo con $I(r)$ la parte intera e con $F(r)$ la parte frazionaria di r
- ☐ Siano p i bit per rappresentare r : f per la parte frazionaria e $(p-f)$ i bit la parte intera:

$$R = a_{p-f-1} \dots a_0 a_{-1} \dots a_{-f} \quad r \cong \sum_{i=-f}^{p-f-1} a_i \beta^i = a_{p-f-1} \beta^{p-f-1} + \dots + a_0 \beta^0 + \underbrace{a_{-1} \beta^{-1} + \dots + a_{-f} \beta^{-f}}_{\text{parte frazionaria}}$$

Esempio: 1110.01 in base *due* vale 14.25 in base *dieci*

- ☐ NB: La virgola non si rappresenta
- ☐ La parte intera $I(r)$ si rappresenta con le tecniche note
- ☐ Per la parte frazionaria si usa il procedimento seguente:

A-76

Numeri Reali – Virgola fissa(2/5)

Si usa la così detta procedura *parte frazionaria-parte intera*:

$$f_0 = F(r)$$

Se $f_0 \neq 0$ eseguire la seguente procedura iterativa:

$$f_{-1} = F(f_0 * 2) \quad a_{-1} = I(f_0 * 2)$$

$$f_{-2} = F(f_{-1} * 2) \quad a_{-2} = I(f_{-1} * 2)$$

...

fino a che f_j è uguale a zero oppure si è raggiunta la precisione desiderata

Esempio:

$$p = 16 \text{ e } f = 5$$

$$r = 331.6875$$

A-77

Numeri Reali – Virgola fissa(3/5)

QUOZIENTE	RESTO
331	-
165	1
82	1
41	0
20	1
10	0
5	0
2	1
1	0
0	1



- Dalla rappresentazione dei numeri naturali: $331 \Leftrightarrow 101001011$;

A-78

Numeri Reali – Virgola fissa(4/5)

F	I
$f_0 = 0.6875$	
$f_1 = F(0.6875 \cdot 2 = \mathbf{1.375}) = 0.375$	$a_1 = I(\mathbf{1.375}) = 1$
$f_2 = F(0.375 \cdot 2 = \mathbf{0.75}) = 0.75$	$a_2 = I(\mathbf{0.75}) = 0$
$f_3 = F(0.75 \cdot 2 = \mathbf{1.5}) = 0.5$	$a_3 = I(\mathbf{1.5}) = 1$
$f_4 = F(0.5 \cdot 2 = \mathbf{1.0}) = \mathbf{0}$	$a_4 = I(\mathbf{1.0}) = 1$



- Rappresentazione della parte intera: 101001011 (necessita di 9 bit)
- Rappresentazione della parte frazionaria su 4 bit: 1011
- Siccome si cercava la rappresentazione su 5 bit, la parte fraz. diventerà 10110
- Rappresentazione complessiva: $R = (\mathbf{00101001011.10110})_{base\ due}$
- Controprova riguardo alla parte frazionaria:
 $1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} \Rightarrow 0.5 + 0.125 + 0.0625 \Rightarrow 0.6875$
- In altre parole abbiamo verificato che $(0.6875)_{base\ dieci} \Leftrightarrow (0.1011)_{base\ due}$

A-79

Numeri Reali – Virgola fissa(5/5)

ERRORE DI TRONCAMENTO

Rappresentare $x = 2.3$ con $f = 6$.

F	I
$f_0 = 0.3$	
$f_1 = F(0.3 \cdot 2 = \mathbf{0.6}) = 0.6$	$a_1 = I(\mathbf{0.6}) = 0$
$f_2 = F(0.6 \cdot 2 = \mathbf{1.2}) = \mathbf{0.2}$	$a_2 = I(\mathbf{1.2}) = \mathbf{1}$
$f_3 = F(0.2 \cdot 2 = \mathbf{0.4}) = 0.4$	$a_3 = I(\mathbf{0.4}) = 0$
$f_4 = F(0.4 \cdot 2 = \mathbf{0.8}) = 0.8$	$a_4 = I(\mathbf{0.8}) = 0$
$f_5 = F(0.8 \cdot 2 = \mathbf{1.6}) = 0.6$	$a_5 = I(\mathbf{1.6}) = 1$
$f_6 = F(0.6 \cdot 2 = \mathbf{1.2}) = \mathbf{0.2}$	$a_6 = I(\mathbf{1.2}) = \mathbf{1}$

Per esprimere x sono necessarie un numero infinito di cifre!

$(10.0 \text{ } \mathbf{1001} \text{ } \mathbf{1001} \text{ } \mathbf{1001} \dots = 10.0\underline{1001} \text{ dove } \underline{1001} \text{ è la parte periodica})$

Ne abbiamo a disposizione solo 6. Quindi si opera un troncamento alla 6^a cifra dopo la virgola.

Quindi, in realtà si rappresenta non x ma il numero x'

$$x' = 10.010011$$

$$x' = 2 + 2^{-2} + 2^{-5} + 2^{-6} = 2 + 0.25 + 0.03125 + 0.015625 = 2.296875$$

L'errore di troncamento è $(2.3 - 2.296875) = 0.00312499$ ($< 2^{-6} = 0.015625$)

A-80

CALCOLO DI FISICA ASTRONOMICA

$$m_e = 9 \times 10^{-28} \text{ gr}; m_{\text{sole}} = 2 \times 10^{+33} \text{ gr} \Rightarrow \text{Intervallo di variazione} \approx 10^{+60}.$$

Sarebbero quindi necessarie almeno 62 cifre, di cui 28 per la parte frazionaria.

Si hanno tante cifre perché l'intervallo da rappresentare è grande \Rightarrow si separa l'intervallo di rappresentabilità dalla precisione, cioè dal numero di cifre.

Notazione Scientifica

$$r = \pm m \cdot \beta^e \quad m = \text{significando}; \quad e = \text{esponente}$$

L'intervallo di rappresentabilità è fissato dal numero di cifre dell'esponente.

L'accuratezza è determinata dal numero di cifre del significando.

Diverse rappresentazioni

3.14 $0.314 \cdot 10^{+1}$ $31.4 \cdot 10^{-1}$

Se ne sceglie una in particolare, che verrà chiamata **forma normalizzata**, in modo da avere una rappresentazione unica

Rappresentazione di un numero binario in virgola mobile:

+1110.01 numero reale binario in virgola fissa

$$\begin{array}{ccc}
 & \swarrow & \searrow \\
 +1.11001 \cdot due^{+11} & \dots & \dots & +111001 \cdot due^{-10}
 \end{array}$$

esponenti espressi in binario

(alcune possibili rappresentazioni in virgola mobile)

Numeri reali *normalizzati*:

- significando con parte intera costituita da un solo bit di valore 1
- la rappresentazione è una tripla, costituita da tre numeri naturali

$$r \leftrightarrow R = \langle s, E, F \rangle$$

Standard IEEE 754-1985

(aggiornato nel 2008 e nel 2019)

La rappresentazione R è composta da tre naturali (s , E ed F), dove:

s = codifica del segno (1 bit)

F = codifica della parte frazionaria del significando su G bit

E = codifica dell'esponente su K bit

$$r = (s == 0) ? [(1+f) \cdot due^e] : [-(1+f) \cdot due^e]$$

$f = F/2^G$ è la parte frazionaria del significando: $m=1+f$ ($= 1 + F/2^G$)

$e = +E - (2^{K-1} - 1)$ è l'esponente rappresentato dal numero naturale E (*rappresentazione con BIAS*)

Conseguenza dell' "uno implicito" \Rightarrow *lo zero non è rappresentabile!*

A-83

Half precision: 16 bit, $K = 5$ e $G = 10$. Esempi di decodifica $R \Rightarrow r$

Esempio 1

$R = \{1, 10011, 1110100101\}$ rappresenta il numero reale negativo con:

$$f = F/2^G = F/2^{dieci} = 0.1110100101$$

$$e = E - (2^{K-1} - 1) = +10011 - (+01111) = +100 \quad (bias=quindici)$$

$$r = -1.1110100101 \cdot due^{+quattro} = -11110.100101$$

$$r = -30.578125 \text{ in base dieci}$$

Esempio 2

$R = \{0, 01111, 0000000001\}$ rappresenta il numero reale positivo:

$$f = F/2^G = 1/2^G = due^{-dieci} = 0.0009765625$$

$$e = E - (2^{K-1} - 1) = 01111 - 01111 = 15 - 15 = 0$$

$$r = +1.0000000001 \cdot due^{+zero} = +1.0000000001$$

$$r = 2^0 + 2^{-10} = 1 + 0.0009765625 = +1.0009765625 \text{ in base dieci}$$

A-84

Half precision, esempi di codifica $r \Rightarrow R$

[In questi primi due esempi A e B vedremo un metodo empirico. Metodo applicabile solo quando r è una potenza di 2]

Esempio A – Rappresentare $r=2$ in half precision

La rappresentazione di $r=2$ è $R = \{0\ 10000\ 0000000000\}$. Infatti, decodificando:

$$f = F/2^G = 0$$

$$e = E - (2^{K-1} - 1) = 10000 - 01111 = 1$$

$$r = +1.0000000000 \cdot due^{+1} = 2 \text{ in base dieci}$$

Esempio B – Rappresentare $r=0.5$ in half precision

La rappresentazione di $r = 0.5$ è $R = \{0\ 01110\ 0000000000\}$. Infatti, decodificando:

$$f = F/2^G = 0$$

$$e = E - (2^{K-1} - 1) = 01110 - 01111 = -1$$

$$r = +1.0000000000 \cdot due^{-1} = 0.5 \text{ in base dieci}$$

A-85

Half precision, esempi di codifica $r \Rightarrow R$

[In questo terzo esempio viene illustrato il metodo generale: prima si calcola la rappresentazione di r in virgola fissa, poi la si usa per trovare la rappresentazione R in virgola mobile]

Esempio C – Rappresentare $r=2.3$ in half precision

Sapendo che r ha rappresentazione in virgola fissa 10.01001, iniziamo a portarlo in forma normalizzata: 1.001001 $\cdot due^{+1}$

Ora ci servono 10 cifre per la parte frazionaria del significando, le restanti verranno scartate, provocando il consueto *errore di troncamento*:

$$1.\underline{0010011001} \cancel{10011001} \dots \cdot due^{+1}$$

F (prime 10 cifre a destra della virgola. Le altre vengono ignorate)

A questo punto la rappresentazione è immediata (per trovare E si faccia riferimento all'esempio A: $R = \{0\ 10000\ 0010011001\}$)

A-86

Numeri Reali – Virgola mobile(7/11)

Numero massimo e minimo: +max_pos, -max_pos

$\mathbf{R}=\{0\ 11111\ 1111111111\}$ (massimo numero rappresentabile, +max_pos)

$\mathbf{R}=\{1\ 11111\ 1111111111\}$ (minimo numero rappresentabile, -max_pos)

$$f = F/2^G = F/2^{dieci} = 0.1111111111$$

$$e = +E - (2^{K-1} - 1) = +11111 - (+01111) = +10000 \text{ (+sedici)}$$

$$|\mathbf{r}| = 1.1111111111 \cdot due^{+10000} < 2^{(2^{(K-1)+1})} = \mathbf{2^{bias+2}} = 2^{17} = 131072$$

$$|\mathbf{r}| = 131008 \cong 1.3 \cdot 10^5 \text{ in base } dieci$$

Riassumendo, nella rappresentazione *half precision*:

max_pos corrisponde a circa $+2^{+17}$

NB: L'intervallo di rappresentabilità: $[-\text{max_pos}, +\text{max_pos}]$ è approssimato dunque dall'intervallo: $[-2^{\text{bias}+2}, 2^{\text{bias}+2}]$

A-87

Numeri Reali – Virgola mobile(8/11)

Numeri con modulo minimo: +min_pos e -min_pos

$\mathbf{R}=\{0, 00000,0000000000\}$ (minimo numero positivo, 0^+)

$\mathbf{R}=\{1,00000,0000000000\}$ (massimo numero negativo, 0^-)

$$f = F/2^G = F/2^{dieci} = 0.0000000000$$

$$e = +E - (2^{K-1} - 1) = +00000 - (+01111) = -01111 = -quindici$$

$$|\mathbf{r}| = 1.0000000000 \cdot due^{-01111} = \mathbf{2^{-bias}}$$

$$|\mathbf{r}| = 2^{-15} \cong 0.31 \cdot 10^{-4}$$

Riassumendo, min_pos (a volte denotato come 0^+) nella rappresentazione *half precision* corrisponde a:

$$+2^{-15} \cong +0.31 \cdot 10^{-4}$$

Il numero negativo più grande (0^-) corrisponde a:

$$-2^{-15} \cong -0.31 \cdot 10^{-4} \text{ (= -min_pos)}$$

A-88

Numeri Reali – Virgola mobile(9/11)

Standard IEEE 754 (1985, 2008, 2019) prevede:

Rappresentazione in **single precision** su **32 bit** con **$K = 8$** e **$G = 23$**
Intervallo di rappresentabilità:

$$\begin{array}{ll} \text{massimo modulo} & \cong 2^{(bias+2)} = 2^{+129} \cong 6.8 \cdot 10^{+38} \\ \text{minimo modulo} & = 2^{-bias} = 2^{-127} \cong 0.58 \cdot 10^{-38} \end{array}$$

Rappresentazione in **double precision** su **64 bit** con **$K = 11$** e **$G = 52$**
Intervallo di rappresentabilità:

$$\begin{array}{ll} \text{massimo modulo} & \cong 2^{(bias+2)} = 2^{+1025} \cong 3.6 \cdot 10^{+308} \\ \text{minimo modulo} & = 2^{-bias} = 2^{-1023} \cong 1.1 \cdot 10^{-308} \end{array}$$

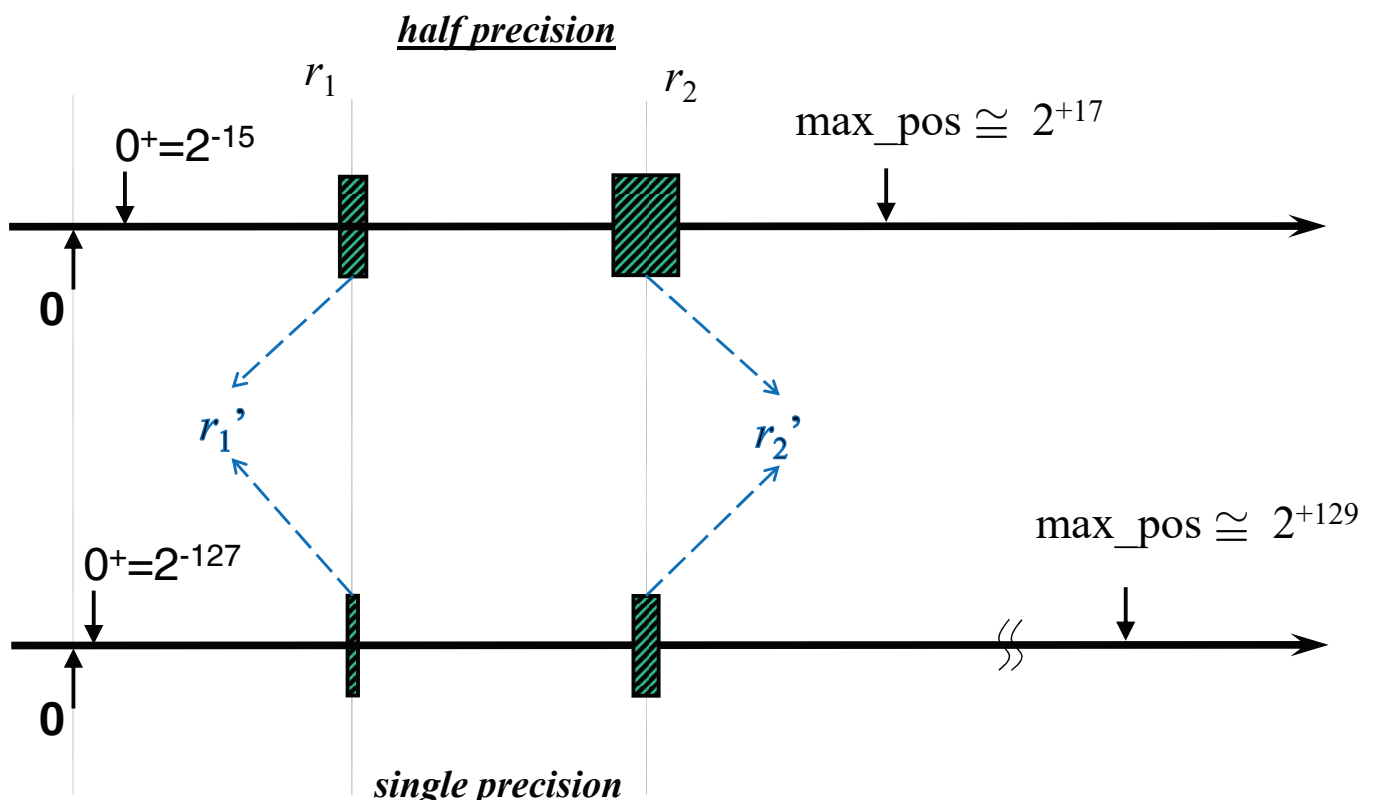
IEEE 754-2008 aggiunge:

rappresentazione in **quadruple precision** su **128 bit** con **$K = 15$** e **$G = 112$**

$$\begin{array}{ll} \text{massimo modulo} & \cong 2^{(bias+2)} = 2^{+16385} \cong 1.2 \cdot 10^{+4932} \\ \text{minimo modulo} & = 2^{-bias} = 2^{-16383} \cong 1.6 \cdot 10^{-4932} \end{array}$$

A-89

Numeri Reali - Virgola mobile (10/11)



A-90

La IEEE sta lavorando in questi ultimi anni alla standardizzazione dei numeri reali in virgola mobile su 8 bit.

Presto ne conosceremo tutti i dettagli. E' possibile che ne vengano standardizzate diverse versioni, con diverso numero di bit per l'esponente.

Ad esempio, standardizzare un float8 con $K = 4$ sembra una scelta del tutto ragionevole, ma anche $K = 3$ è al momento preso in considerazione.

Addittura si sta valutando di standardizzare il float8 con $K = 5$, perché è di interesse per la comunità del *machine learning*.

2.1 Linguaggio di Programmazione C++ (I)

- Per definire sintassi e semantica di un linguaggio occorre utilizzare un altro linguaggio, ossia un *metalinguaggio*
- Metalinguaggio per la sintassi C++:
 - insieme di notazioni (non ambigue), che possono essere spiegate con poche parole del linguaggio naturale.
- Metalinguaggio per la semantica C++:
 - risulta assai complesso, per cui si ricorre direttamente al linguaggio naturale.
- Notazione utilizzata per la sintassi C++:
 - derivata dal classico formalismo di Backus e Naur (*BNF*, *Backus-Naur Form*).

B-2

Parte II

(linguaggio C++, classi escluse)

B-1

2.1 Metalinguaggio per il C++ (I)

NOTAZIONE UTILIZZATA

basata sulla grammatica BNF;
terminologia inglese;
rispetto alla sintassi *ufficiale*, regole semplificate,
caratterizzate dal prefisso *basic*;
diversa organizzazione delle categorie sintattiche.

Regole

- una regola descrive una *categoria sintattica*, utilizzando altre categorie sintattiche, costrutti di metalinguaggio, simboli terminali
- le forme alternative possono stare su righe separate, oppure essere elencate dopo il simbolo del metalinguaggio one of.

Categorie sintattiche:

- scritte in *corsivo*.

Costrutti di metalinguaggio:

- scritti con sottolineatura.

Simboli terminali:

- scritti con caratteri normali.

B-3

2.1 Metalinguaggio per il C++ (II)

- Esempio

frase

soggetto verbo .

soggetto

articolo nome

articolo

one of

il lo

nome

one of

lupo canarino bosco cielo scoiattolo

verbo

one of

mangia vola canta

- Frasi sintatticamente corrette (secondo la semplice sintassi introdotta)

il canarino vola.

il lupo mangia.

il lupo canta.

il scoiattolo vola.

- **ATTENZIONE:**

Una sintassi corretta non implica una semantica corretta.

B-4

2.1 Metalinguaggio per il C++ (III)

Elementi di una categoria sintattica:

- possono essere opzionali:
 - » vengono contrassegnati con il suffisso lopt (simbolo di metalinguaggio).
- possono essere ripetuti più volte:
 - » per far questo, vengono introdotte categorie sintattiche aggiuntive.

Categorie sintattiche aggiuntive

1. Sequenza di un qualunque genere di elementi:

some-element-seq
some-element
some-element some-element-seq

2. Lista di un qualunque genere di elementi (separati da virgola):

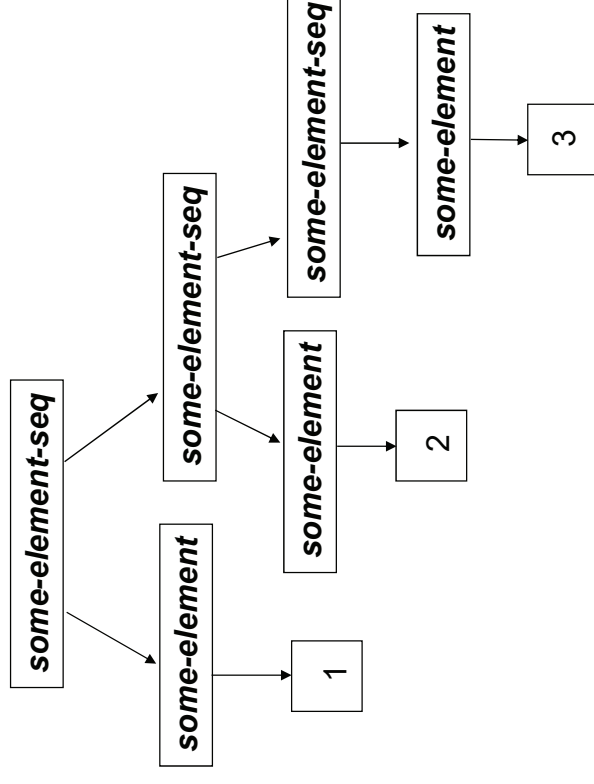
some-element-list
some-element
some-element , some-element-list

B-5

2.1 Metalinguaggio per il C++ (IV)

Albero di derivazione per la sequenza: 1 2 3

some-element
one of
 0 1 2 3 4 5 6 7 8 9



B-6

2.2 Sintassi C++ (I)

- **Programma C++:**
 - costituito da sequenze di *parole (token)*;
 - le parole possono essere delimitate da *spazi bianchi (whitespace)*.
- **Parole:**
 - costituite dai seguenti caratteri:
token-character
digit
letter
special
digit
one of
0 1 2 3 4 5 6 7 8 9
letter
one of
_ a b ... z A B ... Z
special
one of
! % ^ ... /

B-7

2.2 Sintassi C++ (II)

- **Spazi bianchi:**
 - carattere *spazio*;
 - caratteri *tabulazione* (orizzontale e verticale);
 - caratteri *nuova riga* e *ritorno carrello*.
- **Commenti:**
 - sequenze di parole e spazi bianchi racchiuse fra i caratteri */** e **/*, oppure fra i caratteri *//* e la fine della riga;
 - hanno lo scopo di documentare un programma;
 - possono essere inseriti liberamente nel testo e non hanno alcun effetto sull'esecuzione del programma.
- **Spazi bianchi e commenti:**
 - costituiscono le *spaziature*.

B-8

2.2 Sintassi C++ (III)

- **Categorie sintattiche elementari (elementi lessicali):**
 - opportune sequenze di caratteri (token-character o whitespace);
 - non possono includere spaziature (aggiuntive) fra un carattere e un altro.
- **Elementi lessicali:**
 - identificatori (*identifier*);
 - parole chiave (*keyword*);
 - espressioni letterali (*literal*);
 - operatori (*operator*);
 - separatori (*separator*).

B-9

2.2.1 Identificatori

- **Entità usate in un programma:**
 - devono possedere *nomi*;
 - i nomi possono essere *identificatori*:
identifier
letter
letter identifier-char-seq
identifier-char
letter
digit
- **Il carattere di sottolineatura `_` è una lettera.**
 - la doppia sottolineatura all'interno degli identificatori è sconsigliata, perché riservata alle implementazioni ed alle librerie.
- **Il C++ distingue fra maiuscole e minuscole (è *case sensitive*).**
- **Esempi:**
ident
_ident
Ident

B-10

2.2.2 Parole Chiave e Espressioni Letterali

- **Nota:**
 - i termini *nome* e *identificatore* spesso vengono usati intercambiabilmente, ma è necessario distinguerli:
 - » un nome può essere un identificatore, oppure un identificatore con altri simboli aggiuntivi.
- **Parole chiave:**
 - simboli costituiti da parole inglesi (formate da sequenze di lettere), il cui significato è stabilito dal linguaggio:
keyword
one of
and ... while
- Un identificatore non può essere uguale ad una parola chiave.
- **Espressioni letterali:**
 - chiamate semplicemente *letterali*;
 - denotano valori costanti (costanti senza nome);
 - » numeri interi (per es. 10);
 - » numeri reali (per es. –12.5);
 - » letterali carattere (per es. 'a');
 - » letterali stringa (per es. "informatica").

B-11

2.2.4 Operatori e separatori

- **Operatori:**
 - caratteri speciali e loro combinazioni;
 - servono a denotare operazioni nel calcolo delle espressioni;
 - esempi:
 - ☐ carattere +
 - ☐ carattere -
 - ☐ ...
- **Separatori:**
 - simboli di interpunzione, che indicano il termine di una istruzione, separano elementi di liste, raggruppano istruzioni o espressioni, eccetera;
 - esempi:
 - ☐ carattere ;
 - ☐ coppia di caratteri ()
 - ☐ ...



B-12

2.2.4 Proprietà degli operatori (I)

- posizione rispetto ai suoi operandi (o argomenti):
 - *prefisso*: se precede gli argomenti
□ *op arg*
dove *op* e' l'operatore e *arg* e' l'argomento
Esempio: +5
 - *postfisso*: se segue gli argomenti
□ *arg op*
Esempio: x++ (operatore incremento)
 - *infixso*: in tutti gli altri casi;
□ *arg1 op arg2*
Esempio: 4 + 5
- numero di argomenti (o arietà):
Esempio: *op arg* (arietà 1)
 arg1 op arg2 (arietà 2)

B-13

2.2.4 Proprietà degli operatori (II)

- precedenza (o priorità) nell'ordine di esecuzione:
 - gli operatori con priorità più alta vengono eseguiti per primi;
Esempio:
 $arg1 + arg2 * arg3$
(operatore prodotto priorità maggiore dell'operatore somma)
- associatività (ordine in cui vengono eseguiti operatori della stessa priorità):
 - operatori associativi a sinistra: vengono eseguiti da sinistra a destra;
Esempio: $arg1 + arg2 + arg3$

 $(arg1 + arg2) + arg3$
 - operatori associativi a destra: vengono eseguiti da destra a sinistra.
Esempio: $arg1 = arg2 = arg3$

 $arg1 = (arg2 = arg3)$

B-14

3. Un semplice programma

Il più semplice programma C++

```
int main()
{ }
```

Un passo avanti

```
#include <iostream> // direttiva per il preprocessore
// che include la libreria per
// l'ingresso e l'uscita
```

```
using namespace std;
```

```
/* direttiva che indica al compilatore che tutti i nomi
   usati nel programma si riferiscono allo standard
   ANSI-C++ */
```

```
int main() // dichiarazione della funzione main
{
    cout<< "Ciao Mondo!";
    cout<<endl; // serve a spostare il cursore all'inizio
                // della riga successiva
    return 0; // restituisce 0 ovvero tutto OK!!!
}
```

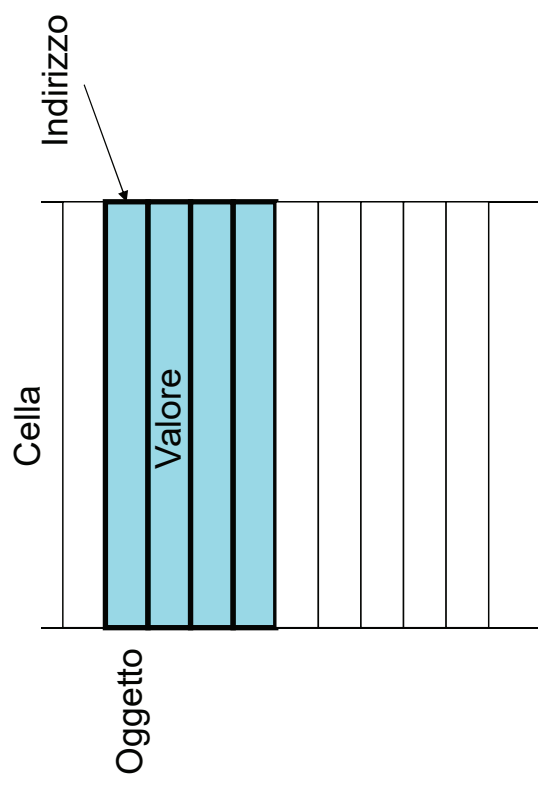
Ciao Mondo!

B-15

3.1 Oggetti (I)

Memoria: insieme di celle.

Cella: in genere dimensione di un byte (8 bit)



Oggetto: gruppo di celle consecutive che vengono considerate dal programmatore come un'unica cella informativa.

Attributi di un oggetto:

Indirizzo della prima cella

Valore (contenuto di tutte le celle)

B-16

3.1 Oggetti (II)

- **Oggetti costanti (costanti con nome) e oggetti variabili:**
 - l'indirizzo comunque non cambia;
 - il valore non può o può subire modifiche, rispettivamente.
- **Programmatore:**
 - si riferisce a un oggetto mediante un nome (caso particolare di nome: identificatore).
- **Oggetto:**
 - ha un tipo.
- **Tipo di un oggetto:**
 - insieme di valori (detti elementi o costanti del tipo);
 - insieme di operazioni definite sugli elementi (con risultato appartenente allo stesso tipo o ad un altro tipo).
- **Associare un tipo a un oggetto:**
 - permette di rilevare in maniera automatica valori che non siano compresi nell'insieme di definizione e operazioni non consentite.

B-17

3.2 Dichiarazioni e Definizioni

- **Costrutti che introducono nuove entità:**
 - dichiarazioni;
 - definizioni.
- **Dichiarazioni:**
 - entità a cui il compilatore non associa locazioni di memoria o azioni eseguibili;
 - esempio: dichiarazioni di tipo.
- **Definizioni:**
 - entità a cui il compilatore associa locazioni di memoria o azioni eseguibili;
 - esempio: definizioni di variabili o di costanti (con nome).
- **Nomenclatura consentita in C++:**
 - spesso non è semplice né conveniente trattare separatamente dichiarazioni e definizioni;
 - con *dichiarazione* si può intendere sia una *dichiarazione* vera e propria sia una *definizione* (le dichiarazioni comprendono le definizioni).

B-18

3.2 Tipi del C++

Tipi: Tipi fondamentali Tipi derivati

- **Tipi fondamentali:**
 - tipi predefiniti;
 - tipi enumerazione.

Tipi predefiniti:

- tipo intero (int) e tipo naturale (unsigned);
- tipo reale (double);
- tipo booleano (bool);
- tipo carattere (char).
- I tipi fondamentali sono chiamati anche *tipi aritmetici*.
- Il tipo intero e il tipo reale sono detti tipi *numerici*.
- Il tipo intero, il tipo booleano, il tipo carattere ed i tipi enumerati sono detti *tipi discreti*.
- **Tipi derivati:**
 - si ottengono a partire dai tipi predefiniti;
 - permettono di costruire strutture dati più complesse.

B-19

3.3 Tipo intero (I)

```
#include <iostream>
using namespace std;
int main()
{
    int i;
    int i1 = 7;
    int i2(7);
    int i3 = 0, i4, i5 = 6;
    i1 = -7;
    i2 = i1 + 3;
    i2 = i1 - 1;
    i2 = i1 * 2;
    i4 = 1 / 2;
    i5 = 1 % 2;
    i3 = 1 / 2 * 2 + 1 % 2;
    cout << i3 << endl;
}
```

// i1 = -7 (cambiamento di segno)
// i2 = -4 (somma)
// i2 = -8 (sottrazione)
// i2 = -14 (moltiplicazione)
// i4 = 0 (quoziente)
// i5 = 1 (resto)
// i3 = 1 (a=(a/b)*b + a%b)

1

B-20

3.3 Tipo intero (II)

```
#include <iostream>
using namespace std;
int main()
{
    // tipo short int
    short int s1 = 1;      // letterale int
    short s2 = 2;

    // tipo long int
    long int ln1 = 6543;   // letterale int
    long ln2 = 6543L;      // letterale long int (suffisso L)
    long ln3 = 6543l;      // letterale long int (suffisso l)

    // letterale int ottale, prefisso 0 (zero)
    int ott = 011;         // ott = 9 (letterale intero ottale)

    // letterale int esadecimale, prefisso 0x o 0X
    int esad1 = 0xF;        // esad1 = 15
    int esad2 = 0XF;        // esad2 = 15

    cout << ott << endl << esad1 << endl;
    cout << esad2 << endl;
}
```

```
9
15
15
```

B-21

3.3 Tipo intero (III)

Definizione di un intero con il formalismo di Backus e Naur

```
basic-int-definition
    int int-specifier-list ;
int-specifier-list
    int-specifier
int-specifier
    int-specifier, int-specifier-list
int-specifier
    identifier int-initializeropt
int-initializer
    = expression
    ( expression )
```

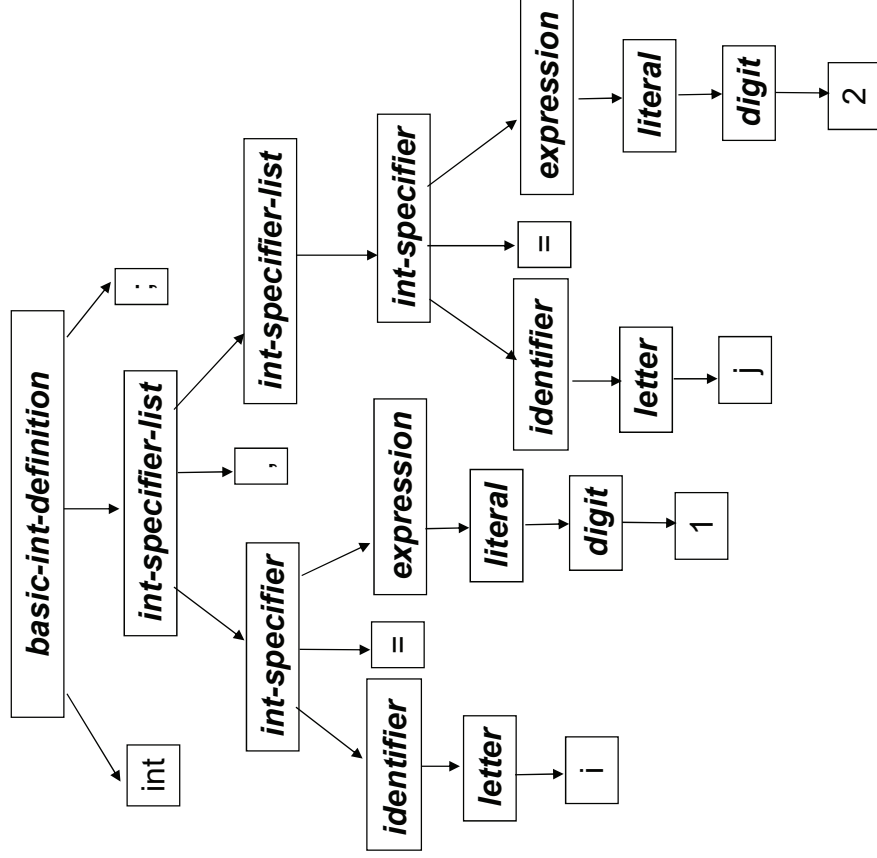
Osservazioni:

- se N è il numero di bit impiegati per rappresentare gli interi, i valori vanno da $-2^{**}(N-1)$ a $2^{**}(N-1)-1$;
- valore tipico di N: 32.

B-22

3.3 Tipo Intero (IV)

Albero di derivazione per la definizione: `int i = 1, j = 2;`



B-23

3.3.1 Tipo unsigned (I)

```
#include <iostream>
using namespace std;
int main()
{
    // tipo unsigned int
    unsigned int u1 = 1U;
    // letterale unsigned, suffisso U
    unsigned u2 = 2u; // letterale unsigned, suffisso u

    // tipo unsigned short int
    unsigned short int u3 = 3;
    unsigned short u4 = 4;

    // tipo unsigned long int
    unsigned long int u5 = 5555;
    unsigned long u6 = 6666UL;
    unsigned long u7 = 7777LU;
    // letterale unsigned long, suffisso UL (ul)

    unsigned short int u8 = -0X0001; // Warning

    cout << u1 << '\t' << u2 << endl;
    cout << u3 << '\t' << u4 << endl;
    cout << u5 << '\t' << u6 << '\t' << u7 << endl;
    cout << u8 << endl;
}
```

```
1 2
3 4
5555 6666 7777
65535
```

B-24

3.3.1 Tipo unsigned (II)

Osservazioni:

- se N è il numero di bit impiegati per rappresentare gli interi, i valori vanno da 0 a $2^{(N-1)}$
- Il tipo unsigned è utilizzato principalmente per operazioni a basso livello:
 - il contenuto di alcune celle di memoria non è visto come un valore numerico, ma come una configurazione di bit.

Operatori bit a bit:

- | OR bit a bit
- & AND bit a bit
- ^ OR esclusivo bit a bit
- ~ complemento bit a bit
- << traslazione a sinistra
- >> traslazione a destra

3.3.1 Tipo unsigned (III)

a	b		&	^	~a	~b
0	0	0	0	0	1	1
0	1	1	0	1	1	0
1	0	1	0	1	0	1
1	1	1	1	0	0	0

3.3.1 Tipo unsigned (IV)

```
#include <iostream>
using namespace std;
int main()
{
    unsigned short a = 0xFFFF9; // in esadecimale
    // 1111 1111 1111 1001 (65529)
    unsigned short b = ~a;
    // 0000 0000 0000 0110 (6)
    unsigned short c = 0x0013;
    // 0000 0000 0001 0011 (19)
    unsigned short d = 021; // in ottale (17)
    unsigned short e = 0b00000000000010010; // in binario (18)

    unsigned short c1, c2, c3;
    c1 = b | c; // 0000 0000 0001 0111 (23)
    c2 = b & c; // 0000 0000 0000 0010 (2)
    c3 = b ^ c; // 0000 0000 0001 0101 (21)

    unsigned short b1, b2;
    b1 = b << 2; // 0000 0000 0001 1000 (24)
    b2 = b >> 1; // 0000 0000 0000 0011 (3)

    cout << a << '\t' << b << '\t' << c << endl;
    cout << c1 << '\t' << c2 << '\t' << c3 << endl;
    cout << b1 << '\t' << b2 << endl;

    cout << d << '\t' << e << endl;
}
```

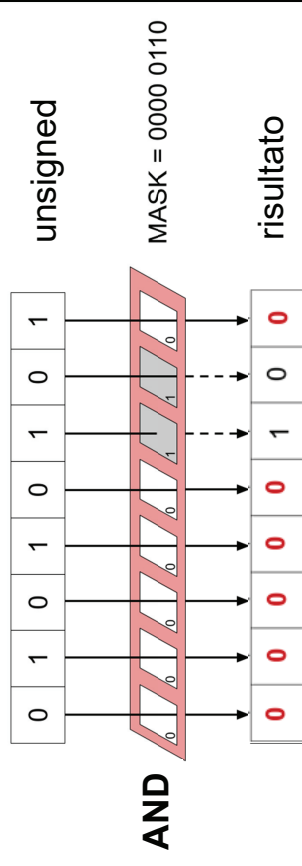
65529	6	19
23	2	21
24	3	
17	18	

B-27

3.3.1 Tipo unsigned (V)

L'AND bit-a-bit può essere usato per forzare a zero alcuni bit, lasciando invariati gli altri.

Occorre dotarsi di una «maschera», contenente degli zeri in corrispondenza dei bit che voglio azzerare (*resettare*), e degli uni in corrispondenza di quelli che voglio lasciare invariati.

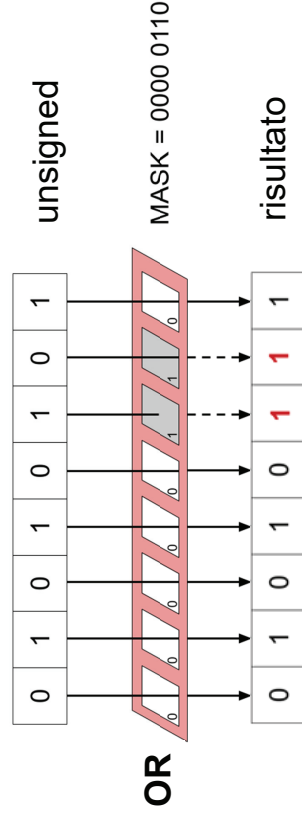


B-28

3.3.1 Tipo unsigned (VI)

L'OR bit-a-bit può essere usato per forzare a uno alcuni bit, lasciando invariati gli altri.

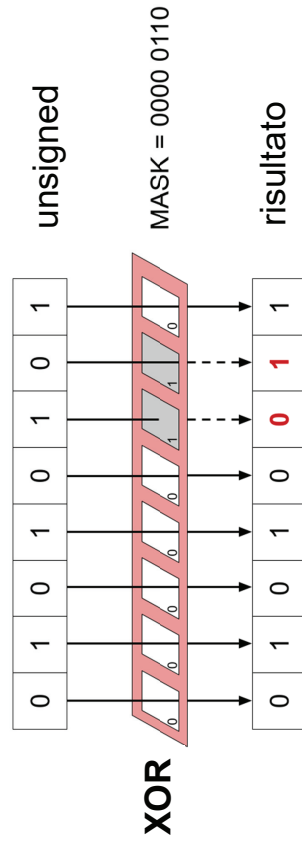
Occorre dotarsi di una «maschera», contenente degli uni in corrispondenza dei bit che voglio forzare ad uno (*settare*), e degli zeri in corrispondenza di quelli che voglio lasciare invariati.



3.3.1 Tipo unsigned (VII)

Lo XOR bit-a-bit può essere usato per invertire alcuni bit, lasciando invariati gli altri.

Occorre dotarsi di una «maschera», contenente degli uni in corrispondenza dei bit che voglio invertire, e degli zeri in corrispondenza di quelli che voglio lasciare inalterati.



3.4 Tipo reale (I)

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

int main()
{
    // tipo double
    double d1 = 3.3;
    double d2 = -12.14e-3, d3 = 1.51;

    // tipo float
    float f = -2.2f;
    float g = f - 12.12F;
    // letterale float, suffisso F (f)

    long double h = +0.1;
    long double k = 1.23e+12L;
    // letterale long double, suffisso L (l)

    cout << d1 << '\t' << d2 << '\t' << d3 << endl;
    cout << f << '\t' << g << endl;
    cout << h << '\t' << k << endl;
}
```

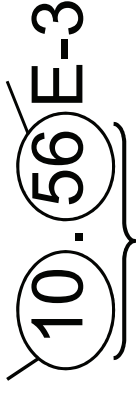
3.3	-0.01214	1.51
-2.2	-14.32	
0.1	1.23e+012	

B-31

3.4 Tipo reale (II)

Letterale reale (forma estesa):

Parte Intera Parte Frazionaria

10.56E-3

Componente in virgola fissa

- la parte intera o la parte frazionaria, se valgono zero, possono essere omesse.

Le operazioni sugli interi e sui reali si indicano con gli stessi simboli (*sovrapposizione o overloading*), ma sono operazioni diverse.

```
#include <iostream>
using namespace std;
int main()
{
    int i = 1, j = 2;
    int z = i / j;
    double d1 = 1.0 / 2.0;
    double d2 = 1 / 2;
    double d3 = (double)i / j;
    cout << z << '\t' << d1 << '\t' << d2 << '\t' << d3 << endl;
}
```

0	0.5	0	0.5
---	-----	---	-----

B-32

3.5 Tipo bool (I)

Tipo *bool*:

valori: costanti predefinite *false* e *true* (codificati con gli interi 0 e 1, rispettivamente).

Operazioni:

|| OR logico o disgiunzione
&& AND logico o congiunzione
! NOT logico o negazione

p	q	p q	p && q	! p
false	false	false	false	true
false	true	true	false	true
true	false	true	false	false
true	true	true	true	false

3.5 Tipo bool (II)

```
#include <iostream>
using namespace std;
int main()
{
    bool b1 = true, b2 = false;           // b3 = false
    bool b3 = b1 && b2;                   // b4 = true
    bool b4 = b1 || b2;

    bool b5 = b1 || b2 && false;
    // b5 = true (AND precedenza maggiore di OR)

    bool b6 = !b2 || b2 && false;
    // b6 = true (NOT prec. maggiore di AND e OR)

    cout << b3 << '\t' << b4 << '\t' << b5;
    cout << '\t' << b6 << endl;
}
```

0	1	1	1
---	---	---	---

3.5 Tipo bool (III)

Nel caso di stampa a video dell'AND logico o dell'OR logico di due variabili, ricordarsi di mettere le parentesi.

```
#include <iostream>
using namespace std;
int main(){
```

```
    bool b1 = true, b2 = false;
```

```
    cout << endl << b1 && b2;    // STAMPA 1!!!
```

```
    cout << endl << ( b1 && b2 ); // STAMPA 0
                                   // (risultato corretto)
```

```
    cout << endl << boolalpha << b1;
    cout << '\t' << b2;
```

```
    cout << endl << noboolalpha << b1;
    cout << '\t' << b2 << endl;
}
```

```
1
0
true false
1 0
```

B-35

3.5 Tipo bool (IV)

Altro esempio: Implicazione logica

$p \rightarrow q$ Si legge «p implica q», oppure
 «se p allora segue q»

L'implicazione è falsa solo se, a fronte di un antecedente p vero, il conseguente q è falso.

In tutti gli altri casi l'implicazione risulta vera.

In particolare, in presenza di un antecedente falso l'implicazione risulterà vera, indipendentemente dal valore di verità del conseguente q.

Tabella di verità

p	q	$p \rightarrow q$	$\neg p$	$\neg p \vee q$
false	false	true	true	true
false	true	true	true	true
true	false	false	false	false
true	true	true	false	true

Dalla tabella di verità qui sopra si evince che $p \rightarrow q$ è equivalente alla formula booleana $\neg p \vee q$, in quanto hanno la stessa tabella di verità.

Queste sono le basi del calcolo proposizionale e della logica booleana. Seguiranno maggiori approfondimenti nel corso di Reti Logiche.

Attenzione: l'implicazione NON è una operazione predefinita per il tipo bool.

B-36

3.5 Operatori di confronto e logici (I)

I tipi aritmetici possono utilizzare gli operatori di confronto:

- == uguale
- != diverso
- > maggiore
- >= maggiore o uguale
- < minore
- <= minore o uguale

Operatori di confronto:

- il risultato è un booleano, che vale *false* se la condizione espressa dall'operatore non è verificata, *true* altrimenti;
- gli operatori di confronto si dividono in:
 - *operatori di uguaglianza* (== e !=);
 - *operatori di relazione*;
- i primi hanno una precedenza più bassa degli altri.

B-37

3.5 Operatori di confronto e logici (II)

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

int main()
{
    bool b1, b2, b3, b4, b5;
    int i = 10;
    float f = 8.0f;

    b1 = i > 3 && f < 5.0;    // false
    b2 = i == f < 5.0;        // false
    b3 = i == i;              // true
    b4 = 4 < i < 7;           // true ???
    b5 = 4 < i && i < 7;        // false

    cout << b1 << '\t' << b2 << '\t' << b3 << endl;
    cout << b4 << '\t' << b5 << endl;
}
```

0	0	1
1	0	

B-38

3.6 Tipo carattere (I)

- insieme di valori: caratteri opportunamente codificati (generalmente un carattere occupa un byte).
- operazioni sui caratteri: sono possibili tutte le operazioni definite sugli interi, che agiscono sulle loro codifiche.

Codifica usata:

- dipende dall'implementazione;
- la più comune è quella ASCII.

Letterale carattere:

- carattere racchiuso fra apici;
- esempio:
 - *// letterale 'a' rappresenta il carattere a.*

Caratteri di controllo:

- rappresentati da combinazioni speciali che iniziano con una barra invertita (sequenze di *escape*).

Alcuni esempi:

- nuova riga (LF) \n
- tabulazione orizzontale \t
- ritorno carrello (CR) \r
- barra invertita \\
- apice \'
- virgolette \"

3.6 Tipo carattere (II)

Ordinamento:

- tutte le codifiche rispettano l'ordine alfabetico fra le lettere, e l'ordine numerico fra le cifre;
- la relazione fra lettere maiuscole e lettere minuscole, o fra caratteri non alfabetici, non è prestabilita (per esempio, in ASCII si ha 'A' < 'a').

Carattere:

- può essere scritto usando il suo valore nella codifica adottata dall'implementazione (per esempio ASCII). Il valore può essere espresso in decimale, ottale ed esadecimale.

Valori ottali:

- formati da cifre ottali precedute da una barra invertita.

Valori esadecimali:

- formati da cifre esadecimali precedute da una barra invertita e dal carattere x (non X).

Nota:

- le sequenze di escape e le rappresentazioni ottale e esadecimale di un carattere, quando rappresentano un *letterale carattere*, vanno racchiuse fra apici;
- esempi:
 - '\n' '\15'

3.6 Tipo carattere (III)

```
#include <iostream>
using namespace std;
int main()
{
    char c1 = 'c', t = '\t', d = '\n';
    char c2 = '\x63';           // 'c' (in esadecimale)
    char c3 = '\143';           // 'c' (in ottale)
    char c4 = 99;                // 'c' (in decimale)

    cout << c1 << t << c2 << t << c3 << t << c4 << d;

    char c5 = c1 + 1;           // 'd'
    char c6 = c1 - 2;           // 'a'
    char c7 = 4 * d + 3;         // '+' (!!!)
    int i = c1 - 'a';            // 2

    cout << c5 << t << c6 << t << c7 << t << i << d;

    bool m = 'a' < 'b', n = 'a' > 'c';    // m = true, n = false

    cout << m << '\n' << n << '\n';
}
```

c	c	c	c
d	a	+	2
1			
0			

B-41

3.7 Tipi enumerazione (I)

Tipi enumerazione (o enumerati):

- costituiti da insiemi di costanti intere, definite dal programmatore, ciascuna individuata da un identificatore e detta *enumeratore*;
- utilizzati per variabili che assumono solo un numero limitato di valori;
- servono a rappresentare informazioni non numeriche;
- non sono predefiniti, ma definiti dal programmatore.

Nota:

- è possibile effettuare separatamente la dichiarazione di un tipo enumerazione e la definizione di variabili di quel tipo.

Operazioni:

- tipicamente, quelle di confronto;
- sono possibili tutte le operazioni definite sugli interi, che agiscono sulla codifica degli enumeratori.

B-42

3.7 Tipi enumerazione (II)

```
#include <iostream>
using namespace std;
int main()
{
    enum Giorni {LUN,MAR,MER,GIO,VEN,SAB,DOM};
    Giorni oggi = MAR;
    oggi = MER;

    int i = oggi;           // 2, conversione implicita
    // oggi = MER-MAR;     // ERRORE! MER-MAR->intero
    // oggi = 3;           // ERRORE! 3 costante intera
    // oggi = i;           // ERRORE! i e' un intero

    cout << int(oggi) << endl;    // 2
    cout << oggi << endl;        // 2, conv. implicita

    enum {ROSSO, GIALLO, VERDE} semaforo;
    semaforo = GIALLO;
    cout << semaforo << endl;    // 1

    enum {INIZ1=10, INIZ2, INIZ3=9, INIZ4};
    cout << INIZ1 << 't' << INIZ2 << 't';
    cout << INIZ3 << 't' << INIZ4 << endl;
}
```

```
2
2
1
10 11 9 10
```

B-43

3.8.1 Conversioni implicite (I)

```
#include <iostream>
using namespace std;
int main()
{
    int i = 10, j;
    float f = 2.5f, h;
    double d = 1.2e+1;
    char c = 'd';

    h = f + 1;           // 3.5
    cout << h << 't';

    j = f + 3.1f;        // 5
    cout << j << endl;

    d = i + 1;           // 11
    cout << d << 't';

    d = f + d;           // 13.5
    cout << d << endl;

    j = c - 'a';         // 3
    cout << j << endl;
}
```

```
3.5 5
11 13.5
3
```

B-44

3.8.1 Conversioni implicite (II)

Osservazione:

- nella conversione da *double* a *int* si può avere una perdita di informazione, poiché avviene un troncamento della parte decimale;
- in alcuni casi, nella conversione da *int* a *double* si può verificare una perdita di precisione per arrotondamento, poiché gli interi sono rappresentati in forma esatta ed i reali sono rappresentati in forma approssimata.
- Esempi:
 - il reale 1588.5 convertito nell'intero 1588;
 - l'intero 0X7FFFFFFF0 (2147483632) convertito nel reale 0X80000000 (2147483648)

Conversioni più significative per gli operatori binari (aritmetici):

- se un operatore ha entrambi gli operandi interi o reali, ma di lunghezza diversa, quello di lunghezza minore viene convertito al tipo di quello di lunghezza maggiore;
- se un operatore ha un operando intero ed uno reale, il valore dell'operando intero viene convertito nella rappresentazione reale, ed il risultato dell'operazione è un reale.

B-45

3.8.1 Conversioni implicite (III)

Conversioni più significative per l'assegnamento:

- a una variabile di tipo reale può essere assegnato un valore di tipo intero;
- a una variabile di tipo intero può essere assegnato un valore di tipo reale, di tipo booleano, di tipo carattere o di un tipo enumerazione;
- a una variabile di tipo carattere può essere assegnato un valore di tipo intero, di tipo booleano, o di un tipo enumerazione.

Nota:

- a una variabile di tipo enumerazione non può essere assegnato un valore che non sia del suo tipo.

Conversioni implicite in sequenza:

- esempio: a una variabile di tipo reale può essere assegnato un valore di tipo carattere (conversione da carattere a intero, quindi da intero a reale).

B-46

3.8.2 Conversioni esplicite

Operatore *static_cast*:

- effettua una conversione di tipo quando esiste la conversione implicita inversa;
- può essere usato per effettuare conversioni di tipo previste dalla conversione implicita.

```
#include <iostream>
using namespace std;
int main()
{
    enum Giorni {LUN,MAR,MER,GIO,VEN,SAB,DOM};
    int i; Giorni g1 = MAR, g2, g3;
    i = g1;
    g1 = static_cast<Giorni>(i);           // cast
    g2 = (Giorni) i;                      // notazione funzionale
    g3 = Giorni (i);
    cout << g1 << '\t' << g2 << '\t' << g3 << endl;

    int j = (int) 1.1;                    // cast, 1
    float f = float(2);                   // notazione funzionale
    cout << j << '\t' << f << endl;
}
```

1	1	1
1	2	

3.9 Dichiarazioni di oggetti costanti

Oggetto costante:

- si usa la parola *const* nella sua definizione;
- è richiesto sempre un inizializzatore.

```
#include <iostream>
using namespace std;
int main()
{
    const long int i = 0;
    const double e1 = 3.5;
    const long double e2 = 2L * e1;

    cout << i << '\t' << e1 << '\t' << e2 << endl;

    // i = 3;           // ERRORE!
    // const int j;     // ERRORE!
}
```

0	3.5	7
---	-----	---

3.10 Operatore sizeof (I)

```
#include <iostream>
using namespace std;
int main()
{
    cout << "char \t" << sizeof(char) << endl; // 1
    cout << "short \t" << sizeof(short) << endl; // 2
    cout << "int \t" << sizeof(int) << endl; // 4
    cout << "long \t" << sizeof(long) << endl; // 4

    cout << "unsigned char \t"; // 1
    cout << sizeof(unsigned char) << endl; // 1
    cout << "unsigned short \t"; // 2
    cout << sizeof(unsigned short) << endl; // 2
    cout << "unsigned int \t"; // 4
    cout << sizeof(unsigned int) << endl; // 4
    cout << "unsigned long \t"; // 4
    cout << sizeof(unsigned long) << endl; // 4

    cout << "float \t" << sizeof(float) << endl; // 4
    cout << "double \t"; // 8
    cout << sizeof(double) << endl; // 8
    cout << "long double \t"; // 12
    cout << sizeof(long double) << endl; // 12
}
```

B-49

3.10 Operatore sizeof (II)

char	1		
short	2		
int	4		
long	4		
unsigned char	1		
unsigned short	2		
unsigned int	4		
unsigned long	4		
float	4		
double	8		
long double	12		

B-50

3.10 Operatore sizeof (III)

```
#include <iostream>
using namespace std;
int main()
{
    cout << "costante carattere ";           // 1
    cout << sizeof 'c' << endl;
    cout << "costante carattere ";           // 1
    cout << sizeof('c') << endl;

    char c = 0;
    cout << "variabile carattere " << sizeof c << endl; // 1

    // cout << "char " << sizeof char << endl; ERRORE!

}
```

costante carattere 1
costante carattere 1
variabile carattere 1

B-51

4.1 Struttura di un programma

basic-main-program

```
int main () compound-statement
compound-statement
{ statement-seq }
statement
declaration-statement
definition-statement
expression-statement
structured-statement
jump-statement
labeled-statement
```

Istruzioni di dichiarazione/definizione:

```
declaration-statement
definition-statement
```

hanno la forma vista in precedenza.

Simboli introdotti dal programmatore:

- devono essere dichiarati/definiti prima di essere usati;
- non è necessario che le dichiarazioni/definizioni precedano le altre istruzioni.

B-52

4.2 L'istruzione espressione

Sintassi:

basic-expression-statement

expr |opt ;

expr

term

expr *infix-binary-op* *expr*

term

primary-exp

pre-fixed-unary-op *expr*

expr *post-fixed-unary-op*

primary-exp

literal

identifier

(*expr*)

Espressione:

- formata da letterali, identificatori, operatori, ecc., e serve a calcolare un valore;
- opzionale, perché in certi casi può essere utile usare una istruzione vuota (che non compie alcuna operazione) indicando il solo carattere ‘,’.

Esempi di istruzioni espressione:

```
5;           // letterale seguito da ;
a;           // nome di variabile seguito da ;
-a;          // operatore unario prefisso
a+b;         // operatore binario infisso
(a+7)*-b;    // combinazione dei precedenti
```

4.2 Espressioni di assegnamento (I)

Nel linguaggio C++ l'assegnamento viene modellizzato come un operatore e pertanto potrà comparire nelle espressioni.

In particolare, l'espressione di assegnamento viene utilizzata per assegnare un nuovo valore ad una variabile pre-esistente.

Sintassi:

basic-assignment-expression

variable-name = *expression*

Effetto:

- calcolare il valore dell'espressione a destra dell'operatore di assegnamento ('=');
- sostituirlo al valore della variabile.

Nome a sinistra dell'operatore di assegnamento:

- individua una variabile, ossia un *lvalue* (left value).

Espressione a destra dell'operatore di assegnamento :

- rappresenta un valore, ossia un *rvalue* (right value).

4.2 Espressioni di assegnamento (II)

```
#include <iostream>
using namespace std;
int main()
{
    int i = 0, j = 1, k;

    i = 3;
    cout << i << endl; // 3
    j = i;
    cout << j << endl; // 3

    k = j = i = 5; // associativo a destra
    cout << i << '\t' << j << '\t' << k << endl; // 5 5 5

    k = j = 2 * (i = 3);
    cout << i << '\t' << j << '\t' << k << endl; // 3 6 6

    // k = j + 1 = 2 * (i = 100); // ERRORE!

    (j = i) = 10;
    cout << j << endl; // 10 (restituisce un l-value)
}
```

```
3
3
5 5 5
3 6 6
10
```

B-55

4.2.1 Altri operatori di assegnamento

basic-recurrence-assignment
variable-name = variable-name \oplus expression

basic-compound-assignment
variable-name \oplus = expression

```
#include <iostream>
using namespace std;
int main()
{
    int i = 0, j = 5;

    i += 5; // i = i + 5
    cout << i << endl; // 5

    i *= j + 1; // i = i * (j + 1);
    cout << i << endl; // 30

    i -= j - 1; // associativo a destra;
    cout << i << endl; // 26

    (i += 12) = 2;
    cout << i << endl; // restituisce un l-value
}
```

```
5
30
26
2
```

B-56

4.2.2 Incremento e decremento

```
#include <iostream>
using namespace std;
int main()
{
    int i, j;

    i = 0; j = 0;           // i += 1; j -= 1;
    ++i; --j;              // 1 -1
    cout << i << '\t' << j << endl;

    i = 0;
    j = ++i;               // i += 1; j = i;
    cout << i << '\t' << j << endl; // 1 1

    i = 0;
    i++;
    cout << i << endl;      // 1

    i = 0;
    j = i++;               // j = i; i += 1;
    cout << i << '\t' << j << endl; // 1 0

    // j = ++i++;
    j = (++i)++;
    // j = i++++;
    int k = ++++i;
    cout << i << '\t' << j << '\t' << k << endl; // 5 2 5

    return 0;
}
```

B-57

4.3 Espressioni aritmetiche e logiche (!)

Calcolo delle espressioni:

- vengono rispettate le precedenze e le associatività degli operatori che vi compaiono;

Precedenza:

- per primi vengono valutati i fattori, calcolando i valori delle funzioni e applicando gli operatori unari (prima incremento e decremento postfissi, poi incremento e decremento prefissi, NOT logico (!), meno unario (-) e più unario (+));
- poi vengono valutati i termini, applicando gli operatori binari nel seguente ordine:
 - quelli moltiplicativi (*, /, %);
 - quelli additivi (+, -);
 - quelli di relazione (<, ...);
 - quelli di uguaglianza (==, !=);
 - quelli logici (nell'ordine, &&, ||);
 - quelli di assegnamento (=, ...);

Parentesi tonde (coppia di separatori):

- fanno diventare qualunque espressione un fattore, che viene quindi calcolato per primo.

B-58

4.3 Espressioni aritmetiche e logiche (II)

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

int main()
{
    int i = 2, j;
    j = 3 * i + 1;
    cout << j << endl;    // 7

    j = 3 * (i + 1);
    cout << j << endl;    // 9

    return 0;
}
```

7
9

B-59

4.3 Espressioni aritmetiche e logiche (III)

Associatività:

- gli operatori aritmetici binari sono associativi a sinistra;
- gli operatori unari sono associativi a destra;
- gli operatori di assegnamento sono associativi a destra.

```
#include <iostream>
using namespace std;
int main()
{
    int i = 8, j = 4, z;
    z = i / j / 2;
    cout << z << endl;    // 1

    z = i / j * 2;
    cout << z << endl;    // 4

    z = i / (j * 2);
    cout << z << endl;    // 1

    z = j * 2 / i;
    cout << z << endl;    // 1

    return 0;
}
```

1
4
1
1

B-60

4.3 Espressioni aritmetiche e logiche (IV)

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

int main() {
    bool k;
    const int i = 0;
    const int j = 5;
    k = i >= 0 && j <= 1;           // (i >= 0) && (j <= 1)
    cout << k << endl;             // 0
    k = i && j || !k;                // (i && j) || (!k)
    cout << k << endl;             // 1
    k = 0 < j < 4;                  // ATTENZIONE!
    cout << k << endl;             // 1
    k = 0 < j && j < 4;
    cout << k << endl;             // 0

    return 0;
}
```

0
1
1
0

B-61

4.3 Espressioni aritmetiche e logiche (V)

Operatori && e ||:

- sono associativi a sinistra;
- il calcolo di un'espressione logica contenente questi operatori termina appena si può decidere se l'espressione è, rispettivamente, falsa o vera.

Questo comportamento va sotto il nome di *regola del cortocircuito*, o *regola della scorciatoia* (dall'inglese «shortcut rule»)

```
#include <iostream>
using namespace std;
int main() {
    bool k;
    const int i = 0;
    k = (i >= 0) || (i++);
    cout << k << '\t' << i << endl; // 1 0
    k = (i > 0) || (i++);
    cout << k << '\t' << i << endl; // 0 1
    k = (i >= 0) && (i <= 100);
    cout << k << endl;             // 1
    cin >> i; // ora in i ci può essere qualunque intero
    // k = (10 / i >= 3) && (i != 0);
    // versione sbagliata (se i==0 // si avrebbe divisione per zero!)
    k = (i != 0) && (10 / i >= 3); // versione corretta
    cout << k << endl;             // 0 o 1 (a seconda del val di i)

    return 0;
}
```

B-62

4.4 Operatore condizionale (I)

e1 ? e2 : e3

e1 espressione logica

Se **e1** è vera, il valore restituito dall'operatore condizionale è il valore di **e2**, altrimenti di **e3**.

// Legge due interi e stampa su uscita standard il minore

```
#include <iostream>
using namespace std;
int main()
{
    int i, j, min;
    cout << "Inserisci due numeri interi" << endl;

    cin >> i >> j;
    min = (i < j ? i : j);
    cout << "Il numero minore e': " << min << endl;

    return 0;
}
```

```
Inserisci due numeri interi
2
4
Il numero minore e': 2
```

B-63

4.4 Operatore virgola (I)

A volte è comodo poter inserire due o più espressioni laddove la grammatica ne prevederebbe una sola.

In questi casi viene in aiuto l'operatore virgola.

Si tratta di un operatore binario infisso, associativo a sinistra.

Sintassi:

esp1, esp2

Funzionamento:

viene prima valutata l'espressione di sinistra (**esp1**), dopodichè viene valutata l'espressione di destra (**esp2**). L'operatore restituisce, come risultato, il risultato prodotto dalla seconda espressione. *Il risultato prodotto dalla valutazione delle prima espressione viene ignorato.*

Esempio di utilizzo:

```
int main()
{
    int a = 2;
    int b = 3;
    a = (b++, 5);    // chiamata dell'operatore virgola
    cout << b << endl; // stampa 4
    cout << a << endl; // stampa 5
}
```

B-64

4.4 Operatore virgola (II)

Attenzione! L'operatore virgola è quello a più bassa priorità di tutti! Più bassa anche di quella dell'operatore di assegnamento.

Pertanto il seguente codice non produce lo stesso risultato del codice precedentem in quanto prima viene effettuato l'assegnamento e successivamente viene valutato l'operatore virgola:

```
int main()
{
    int a = 2;
    int b = 3;
    a = b++, 5;
    cout<<b<<endl; // stampa 4
    cout<<a<<endl; // stampa 3
}
```

Per quanto detto, il codice riportato qui sopra è equivalente al seguente codice:

```
int main()
{
    int a = 2;
    int b = 3;
    (a = b++), 5;
    cout<<b<<endl; // stampa 4
    cout<<a<<endl; // stampa 3
}
```

// NB: L'operatore virgola capiterà di vederlo utilizzato // nell'istruzione for

B-65

6. Istruzioni strutturate

Istruzioni strutturate:

- consentono di specificare azioni complesse.

structured-statement
compound-statement
selection-statement
iteration-statement

Le istruzioni strutturate sono importantissime, perché è stato dimostrato un teorema ([Teorema di Böhm-Jacopini](#)) che dice che ogni problema può essere risolto usando opportune combinazioni delle tre istruzioni strutturate. Non ne servirebbero altre.

Istruzione composta:

- già esaminata nella sintassi di programma;
- consente, per mezzo della coppia di delimitatori { e }, di trasformare una qualunque sequenza di istruzioni in una singola istruzione.
- ovunque la sintassi preveda un'istruzione, si può mettere una sequenza comunque complessa di istruzioni racchiusa tra i due delimitatori.

Istruzioni condizionali:

selection-statement
if-statement
switch-statement
if-statement
if (condition) statement
if (condition) statement else statement

B-66

6.3.1 Istruzione if (I)

// Trova il maggiore tra due interi

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, max;
    cin >> a >> b;
    if (a > b)
        max = a;
    else
        max = b;
    cout << max << endl;
    return 0;
}
```

// 4 6

4
6
6

B-67

6.3.1 Istruzione if (II)

// Incrementa o decrementa

```
#include <iostream>
using namespace std;
int main()
{
    int a, b;
    cin >> a >> b;
    if (a >= b)
    {
        a++;
        b++;
    }
    else
    {
        a--;
        b--;
    }
    cout << a << '\t' << b << endl;
    return 0;
}
```

4
6
3 5

B-68

6.3.1 Istruzione if (III)

```
// Valore assoluto (if senza parte else)

#include <iostream>
using namespace std;
int main()
{
    int a;
    cout << "Inserisci un numero intero " << endl;
    cin >> a;
    if (a < 0)
        a = -a;
    cout << "Il valore assoluto e' ";
    cout << a << endl;

    return 0;
}
```

Inserisci un numero intero
-4
Il valore assoluto e' 4

B-69

6.3.1 Istruzione if (IV)

```
// Legge un numero, incrementa il numero e
// lo scrive se è diverso da 0
// (if senza espressione relazionale)

#include <iostream>
using namespace std;
int main()
{
    int i;
    cout << "Inserisci un numero intero " << endl;
    cin >> i;
    if (i++)
        cout << "Numero incrementato " << i << endl;

    return 0;
}
```

Inserisci un numero intero
2
Numero incrementato 3

Inserisci un numero intero
0

N.B.: L'espressione nella condizione può restituire un valore aritmetico: se il valore è 0, la condizione è falsa; altrimenti è vera.

B-70

6.3.1 Istruzione if (V)

```
// If in cascata
// if (a > 0) if (b > 0) a++; else b++;

#include <iostream>
using namespace std;
int main()
{
    int a, b;
    cout << "Inserisci due numeri interi" << endl;
    cin >> a >> b;
    if (a > 0)
        if (b > 0)
            a++;
        else
            b++;
    cout << a << "\t" << b << endl;
    return 0;
}
```

Inserisci due numeri interi

3
5
4 5

NOTA BENE

la parte *else* si riferisce alla condizione più vicina (nell'esempio, alla condizione `b > 0`);

B-71

6.3.1 Istruzione if (VI)

```
// Scrittura fuorviante

#include <iostream>
using namespace std;
int main()
{
    int a, b;
    cout << "Inserisci due numeri interi" << endl;
    cin >> a >> b;
    if (a > 0)
        if (b > 0)
            a++;
        else
            b++;
    cout << a << "\t" << b << endl;
    return 0;
}
```

Inserisci due numeri interi

5
7
6 7

B-72

6.3.1 Istruzioni if (VII)

```
// Scrive asterischi
#include <iostream>
using namespace std;
int main()
{
    int i;
    cout << "Quanti asterischi? " << endl;
    cin >> i;
    if (i == 1)
        cout << '*';
    else
        if (i == 2)
            cout << "***";
        else
            if (i == 3)
                cout << "*****";
            else
                cout << '!';
    cout << endl;
    return 0;
}
```

Quanti asterischi?

2
**

B-73

6.3.1 Istruzione if (VIII)

```
// Equazione di secondo grado
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    double a, b, c;
    cout << "Coefficienti? " << endl;
    cin >> a >> b >> c;
    if ((a == 0) && (b == 0))
        cout << "Equazione degenera" << endl;
    else
        if (a == 0)
        {
            cout << "Equazione di primo grado" << endl;
            cout << "x: " << -c / b << endl;
        }
        else
        {
            double delta = b * b - 4 * a * c;
            if (delta < 0)
                cout << "Soluzioni immaginarie" << endl;
            else
            {
                delta = sqrt(delta);
                cout << "x1: " << (-b + delta) / (2 * a) << endl;
                cout << "x2: " << (-b - delta) / (2 * a) << endl;
            }
        }
    return 0;
}
```

B-74

6.3.1 Istruzione if (IX)

Coefficienti?

1
6
9
x1: -3
x2: -3

B-75

6.3.2 Istruzioni switch e break (I)

Sintassi:

switch-statement

switch (*expression*) *switch-body*
switch-body

{ *alternative-seq* }
alternative

case-label-seq *statement-seq*

case-label

case *constant-expression* :
default :

espressione:

- comunemente costituita da una variabile a valori discreti (int, char, enum, ecc...);

Etichette (case-label):

- contengono (oltre alla parola chiave case) espressioni costanti il cui valore deve essere del tipo del risultato dell'espressione;
- individuano le varie alternative nel corpo dell'istruzione *switch*;
- i valori delle espressioni costanti devono essere distinti.

Alternativa con etichetta default:

- se presente, deve essere unica.

B-76

6.3.2 Istruzioni switch e break (II)

Esecuzione dell'istruzione *switch*:

- viene valutata l'espressione;
- viene eseguita l'alternativa con l'etichetta in cui compare il valore calcolato (ogni alternativa può essere individuata da più etichette);
- se nessuna alternativa ha un'etichetta in cui compare il valore calcolato, allora viene eseguita, se esiste, l'alternativa con etichetta *default*;
- in mancanza di etichetta *default* l'esecuzione dell'istruzione *switch* termina.

Alternativa:

- formata da una o più istruzioni (eventualmente vuote o strutturate).

Terminazione:

- può essere ottenuta con l'istruzione *break* (rientra nella categoria delle istruzioni di salto):
break-statement
break ;

Attenzione:

- Se l'ultima istruzione di un'alternativa non fa terminare l'istruzione *switch*, e se l'alternativa non è l'ultima, viene eseguita l'alternativa successiva.

B-77

6.3.2 Istruzioni switch e break (III)

// Scrive asterischi (uso istruzione *break*)

```
#include <iostream>
using namespace std;
int main()
{
    int i;
    cout << "Quanti asterischi? " << endl;
    cin >> i;
    switch (i)
    {
        case 1:
            cout << '*';
            break;
        case 2:
            cout << "***";
            break;
        case 3:
            cout << "*****";
            break;
        default:
            cout << '!';
    }
    cout << endl;

    return 0;
}
```

Quanti asterischi?

2
**

B-78

6.3.2 Istruzioni switch e break (IV)

```
// Scrive asterischi (in cascata)

#include <iostream>
using namespace std;
int main()
{
    int i;
    cout << "Quanti asterischi? " << endl;
    cin >> i;
    switch (i)
    {
        case 3:
            cout << ".*";
            // in cascata
        case 2:
            cout << ".*";
            // in cascata
        case 1:
            cout << ".*";
            break;
        default:
            cout << "I";
    }
    cout << endl;

    return 0;
}
```

Quanti asterischi?

2
**

B-79

6.3.2 Istruzioni switch e break (V)

```
// Creazione menù
// Selezione tramite caratteri

#include <iostream>
using namespace std;
int main()
{
    cout << "Seleziona un'alternativa" << endl;
    cout << "A - Prima Alternativa" << endl;
    cout << "B - Seconda Alternativa" << endl;
    cout << "C - Terza Alternativa" << endl;
    cout << "Qualsiasi altro tasto per uscire" << endl;
    char c;
    cin >> c;
    switch (c)
    {
        case 'a': case 'A':
            cout << "Prima alternativa" << endl;
            break;
        case 'b': case 'B':
            cout << "Seconda alternativa" << endl;
            break;
        case 'c': case 'C':
            cout << "Terza alternativa" << endl;
            // Manca il caso di default
            // Se non è una delle alternative, non scrive niente
    }

    return 0;
}
```

B-80

6.3.2 Istruzioni switch e break (VI)

```
// Creazione menù  
// Selezione tramite caratteri
```

```
Seleziona un'alternativa  
A - Prima Alternativa  
B - Seconda Alternativa  
C - Terza Alternativa  
Qualsiasi altro tasto per uscire  
B  
Seconda alternativa
```

B-81

6.3.2 Istruzioni switch e break (VII)

```
// Scrittura di enumerazioni  
  
#include <iostream>  
using namespace std;  
int main()  
{  
    enum COLORE{ROSSO, GIALLO, VERDE};  
    COLORE colore;  
    char c;  
    cout << "Seleziona un colore " << endl;  
    cout << "R - rosso " << endl;  
    cout << "G - giallo " << endl;  
    cout << "V - verde " << endl;  
    cin >> c;  
    switch (c)  
    {  
        case 'r': case 'R':  
            colore = ROSSO;  
            break;  
        case 'g': case 'G':  
            colore = GIALLO;  
            break;  
        case 'v': case 'V':  
            colore = VERDE;  
    }  
    /* ... */  
}
```

B-82

6.3.2 Istruzioni switch e break (VIII)

// Scrittura di enumerazioni (continua)

```
switch (colore)
{
    case ROSSO:
        cout << "ROSSO";
        break;
    case GIALLO:
        cout << "GIALLO";
        break;
    case VERDE:
        cout << "VERDE";
        cout << endl;
    }
    return 0;
}
```

Seleziona un colore

R - rosso

G - giallo

V - verde

V

VERDE

B-83

6.4.1 Istruzione ripetitive

Sintassi:

iteration-statement

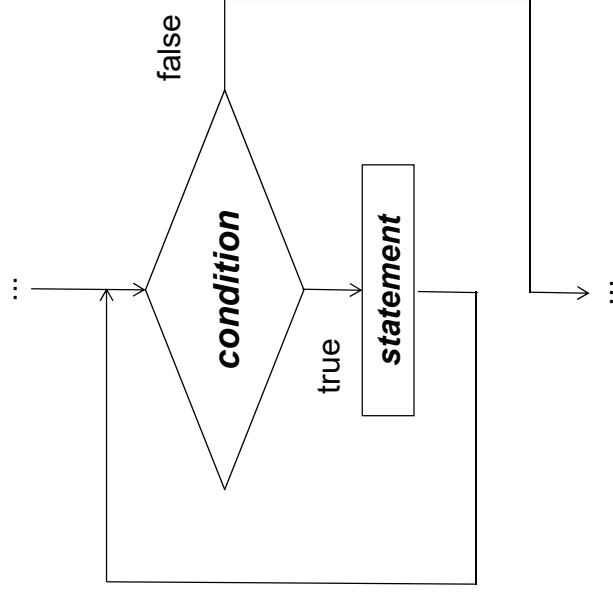
while-statement

do-statement

for-statement

while-statement

while (*condition*) *statement*



B-84

6.4.1 Istruzione while (I)

```
// Scrive n asterischi, con n dato (i)

#include <iostream>
using namespace std;
int main()
{
    int n, i = 0;
    cout << "Quanti asterischi? " << endl;
    cin >> n;
    while (i < n)
    {
        cout << "*";
        i++;
    }
    // n conserva il valore iniziale
    cout << endl;

    return 0;
}
```

Quanti asterischi?

6

B-85

6.4.1 Istruzione while (II)

```
// Scrive n asterischi, con n dato (ii)

#include <iostream>
using namespace std;
int main()
{
    int n;
    cout << "Quanti asterischi? " << endl;
    cin >> n;
    while (n > 0)
    {
        cout << "*";
        n--;
    }
    // al termine, n vale 0
    cout << endl;

    return 0;
}
```

Quanti asterischi?

6

B-86

6.4.1 Istruzione while (III)

```
// Scrive n asterischi, con n dato (iii)

#include <iostream>
using namespace std;
int main()
{
    int n;
    cout << "Quanti asterischi? " << endl;
    cin >> n;
    while (n-- > 0)
        cout << "*";
    cout << endl;

    return 0;
}

//~~~~~//
// Scrive n asterischi, con n dato (iv)

#include <iostream>
using namespace std;
int main()
{
    int n;
    cout << "Quanti asterischi? " << endl;
    cin >> n;
    while (n--)
        cout << "*";
    cout << endl;

    return 0;
}
```

B-87

6.4.1 Istruzione while (III)

```
// Legge, raddoppia e scrive interi non negativi
// Termina al primo negativo

#include <iostream>
using namespace std;
int main()
{
    int i;
    cout << "Inserisci un numero intero" << endl;
    cin >> i;
    while (i >= 0)
    {
        cout << 2 * i << endl;
        cout << "Inserisci un numero intero" << endl;
        cin >> i;
    }

    return 0;
}
```

```
Inserisci un numero intero
1
2
Inserisci un numero intero
3
6
Inserisci un numero intero
-2
```

B-88

6.4.1 Istruzione while (IV)

```
// Calcola il massimo m tale che la somma dei primi
// m interi positivi e` minore o uguale ad un dato intero
// positivo n

// Esempio: dato n = 8, il programma deve stabilire che
// il massimo intero m vale 3, in quanto
// 1+2+3 = 6, che è minore o uguale ad 8, mentre
// 1+2+3+4 = 10, che è maggiore di 8
```

```
#include <iostream>
using namespace std;
int main()
```

```
{
    unsigned int somma = 0, m = 0, n;
    cout << "Inserisci n " << endl;
    cin >> n;
    while (somma <= n)
        somma += ++m;
    m--;
    cout << m << endl;

    return 0;
}
```

Inserisci n

8

3

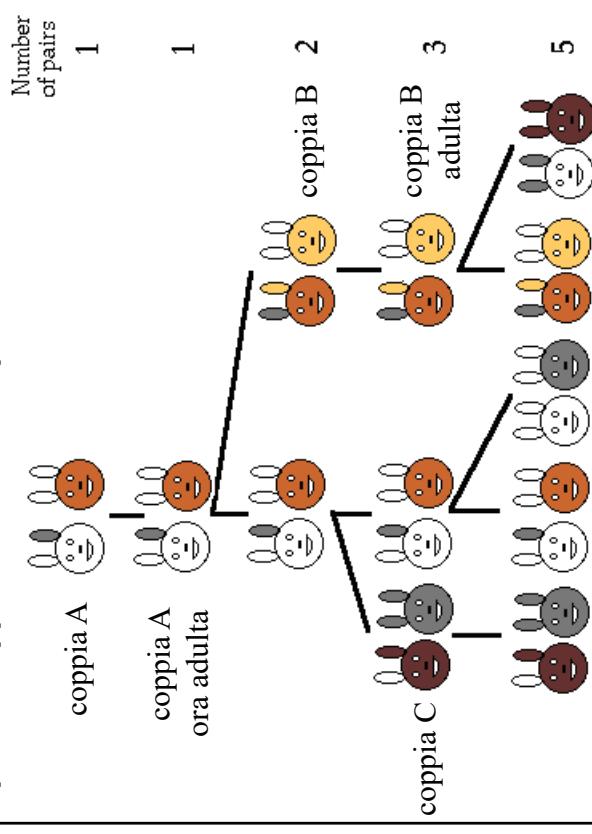
B-89

6.4.1 Istruzione while (V)

```
// Calcola il massimo termine della successione di
// Fibonacci minore o uguale al dato intero positivo n
// Serie di Fibonacci:
```

Curiosità: da dove nasce la serie di Fibonacci?

Supponiamo di avere una coppia di conigli (maschio e femmina). I conigli sono in grado di riprodursi all'età di un mese. Supponiamo che i nostri conigli non muoiano mai e che la femmina produca sempre una nuova coppia (un maschio ed una femmina) ogni mese dal secondo mese in poi. Il problema posto da Fibonacci fu: quante coppie ci saranno dopo un anno?



Il numero delle coppie di conigli all'inizio di ciascun mese sarà 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

B-90

6.4.1 Istruzione while (VI)

```
// Calcola il massimo termine della successione di
// Fibonacci minore o uguale al dato intero positivo n
// Serie di Fibonacci:
//  $a_n = a_{n-1} + a_{n-2}$ 
//  $a_1 = 1$ 
//  $a_0 = 0$ 

#include <iostream>
using namespace std;
int main()
{
    int i = 0, j = 1, s, n;           // i = a0, j = a1, s = a2
    cout << "Inserisci n " << endl;
    cin >> n;
    if (n <= 0) cout << "Valore non consistente" << endl;
    else
    {
        while ((s = j + i) <= n)
        {
            i = j;
            j = s;
        }
        cout << j << endl;
    }

    return 0;
}
```

Inserisci n
7
5

B-91

6.4.1 Istruzione while (VII)

```
// Calcola il massimo termine della successione di
// Fibonacci minore o uguale al dato intero positivo n
// Serie di Fibonacci:
//  $0, 1, 1, 2, 3, 5, 8, 13, \dots$ 
// Soluzione con due variabili

#include <iostream>
using namespace std;
int main()
{
    int i = 0, j = 1, n;           // i = a0, j = a1
    cout << "Inserisci n " << endl;
    cin >> n;
    if (n <= 0) cout << "Valore non consistente" << endl;
    else
    {
        while (((i = j+i) <= n) && ((j = j+i) <= n));
        if (j < i) cout << j << endl;
        else cout << i << endl;
    }

    return 0;
}
```

Inserisci n
7
5

B-92

6.4.2 Istruzione do (I)

do-statement

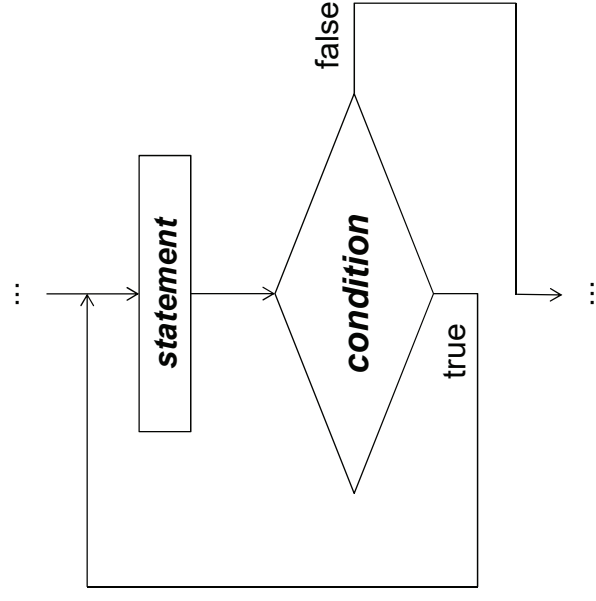
```
do statement while (condition) ;
```

Esecuzione dell'istruzione do:

- viene eseguita l'istruzione racchiusa tra *do* e *while* (corpo del *do*);
- viene valutata la condizione;
- se questa risulta vera l'istruzione *do* viene ripetuta;
- se questa risulta falsa l'istruzione *do* termina.

Nota:

- il corpo dell'istruzione *do* viene eseguito almeno una volta, prima della valutazione della condizione di terminazione.



B-93

6.4.2 Istruzione do (III)

```
// Leggi un intero n da tastiera. Dopodichè leggi una
// sequenza di interi e sommali, finchè la loro somma
// non supera n
```

```
#include <iostream>
using namespace std;
int main(){
    int n, i, s = 0, count = 0;
    cout << "Inserisci n " << endl;
    cin >> n;
```

```
do{
    cout << "Inserisci il prossimo intero " << endl;
    cin >> i;
    s += i;
    count++;
} while ( s <= n);
cout << "La somma dei primi " << count;
cout << " numeri inseriti vale " << s;
cout << " ( > "<<n<<" )"<<endl;
    return 0;
}
```

```
Inserisci n
11
Inserisci il prossimo intero
7
Inserisci il prossimo intero
5
La somma dei primi 2 numeri inseriti vale 12 ( > 11 )
```

B-94

6.4.2 Istruzione do (III)

// Calcola il massimo termine della successione di
// Fibonacci minore o uguale al dato intero positivo n

0,1,1,2,3,5,8,13, ...

```
#include <iostream>
using namespace std;
int main()
{
    int i, j = 0, s = 1, n;
    cout << "Inserisci n " << endl;
    cin >> n;
    if ( n<=0 )
        cout << "Valore non consistente" << endl;
    else
    {
        do
        {
            i = j;
            j = s;
        } while ((s = j + i) <= n);
        cout << j << endl;
    }
    return 0;
}
```

Inserisci n
7
5

B-95

6.4.3 Istruzione for (I)

for-statement
for (*initialization* ; *condition* ; *step*)
statement

initialization
expression|*opt*
definition|*opt*

condition
bool (*expression*|*opt*)
step
expression|*opt*

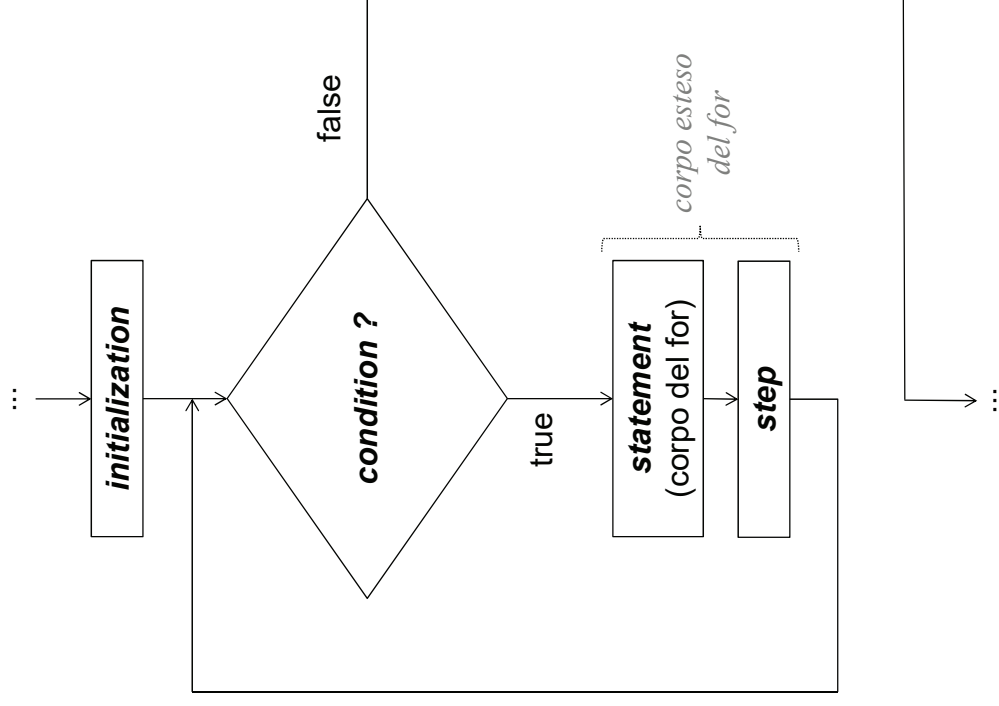
Comportamento:

L'istruzione for viene eseguita come se fosse scritta nel seguente modo:

```
{
    initialization // init. eseguita una sola volta (all'inizio)
    while ( condition )
    { // corpo esteso del for (include lo step)
        statement // corpo del for (step escluso)
        step ;
    }
}
```

B-96

6.4.3 Istruzione for (II)



B-97

6.4.3 Istruzione for (III)

```
// Scrive n asterischi, con n dato (i)

#include <iostream>
using namespace std;
int main()
{
    int n;
    cout << "Quanti asterischi? " << endl;
    cin >> n;
    for (int i = 0; i < n; i++)
        cout << '*';
    cout << endl;
    // al termine, i vale n

    return 0;
}
```

Quanti asterischi?

6

B-98

6.4.3 Istruzione for (IV)

```
// Scrive n asterischi, con n dato (ii)

// variante 1
#include <iostream>
using namespace std;
int main(){
    int n;
    cout << "Quanti asterischi? " << endl;
    cin >> n;
    for (; n > 0; n--)
        cout << '*';
    cout << endl;
    // al termine, n vale 0

    return 0;
}

// variante 2
#include <iostream>
using namespace std;
int main(){
    int n;
    cout << "Quanti asterischi? " << endl;
    cin >> n;
    for (; n > 0; ){
        cout << '*';
        n--;
    }
    cout << endl;
    // al termine, n vale 0

    return 0;
}
```

B-99

6.4.3 Istruzione for (V)

```
// Scrive asterischi e punti esclamativi (I)

#include <iostream>
using namespace std;
int main()
{
    int n;
    cout << "Quanti? " << endl;
    cin >> n;
    for (int i = 0; i < n; i++)
        cout << '*';
    cout << endl;

    for (int i = 0; i < n; i++)
        cout << 'I';
    cout << endl;
    // visibilita' i limitata
    // al blocco for

    return 0;
}
```

```
Quanti?
6
*****
!!!!!!
```

B-100

6.4.3 Istruzione for (VI)

// Scrive asterischi e punti esclamativi (II)

```
#include <iostream>
using namespace std;
int main()
{
    int n,i;
    cout << "Quanti? " << endl;
    cin >> n;
    for (i = 0; i < n; i++)
        cout << '*',
        cout << endl;
    for (i = 0; i < n; i++)
        cout << '!',
        cout << endl;
    return 0;
}
```

Quanti?

6

!!!!!!

B-101

6.4.3 Istruzione for (VII)

// Scrive una matrice di asterischi formata da r righe
// e c colonne, con r e c dati

```
#include <iostream>
using namespace std;
int main(){
    int r, c;
    cout << "Numero di righe? " << endl;
    cin >> r;
    cout << "Numero di colonne? " << endl;
    cin >> c;
    for (int i = 0; i < r; i++){
        for (int j = 0; j < c; j++){
            cout << '*',
            cout << endl;
        }
    }
    return 0;
}
```

Numero di righe?

3

Numero di colonne?

5

B-102

6.4.3 Istruzione for (VII)

```
// Scrive una matrice di asterischi formata da r righe
// e c colonne, con r e c dati

// variante 1 (del tutto equivalente alla soluzione prec.)
#include <iostream>
using namespace std;
int main(){
    int r, c, i, j;
    cout << "Numero di righe? " << endl;
    cin >> r;
    cout << "Numero di colonne? " << endl;
    cin >> c;
    for (i = 0; i < r; i++){
        j = 0;
        for ( ; j < c; j++)
            // ora è più chiaro ancora
            // che l'istr. j=0; viene eseguita
            cout << " ";
            // ogni volta che viene eseguita
            cout << endl;
            // l'iterazione più esterna (righe)
        }
    }

    return 0;
}
```

Numero di righe?

3

Numero di colonne?

5

6.5 Istruzioni di salto

jump-statement

break-statement

continue-statement

goto-statement

return-statement

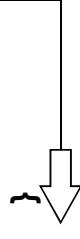
- Istruzione break (già vista):

- salto all'istruzione immediatamente successiva al corpo del ciclo o dell'istruzione switch che contengono l'istruzione *break*:

```
while ( ... )
```

```
{ ...
```

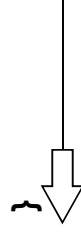
```
    break; ...
```



```
switch ( .... )
```

```
{ ....
```

```
    break; ...
```



6.5.1 Istruzione break (I)

```
// Legge e scrive interi non negativi
// Termina al primo negativo

#include <iostream>
using namespace std;
int main()
{
    int j;           // ciclo infinito; altra forma: while(true)
    for (;;)
    {
        cout << "Inserisci un numero intero " << endl;
        cin >> j;
        if (j < 0)
            break;
        cout << j << endl;
    }

    return 0;
}
```

```
Inserisci un numero intero
3
3
Inserisci un numero intero
5
5
Inserisci un numero intero
-1
```

B-105

6.5.1 Istruzione break (II)

```
// Legge e scrive al più cinque interi non negativi
// Termina al primo negativo

#include <iostream>
using namespace std;
int main()
{
    const int N = 5;
    for (int i = 0, j; i < N; i++)
    {
        cout << "Inserisci un numero intero " << endl;
        cin >> j;
        if (j < 0)
            break;
        cout << j << endl;
    }

    return 0;
}
```

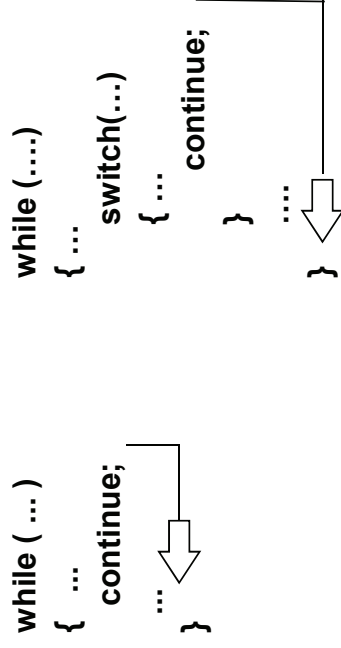
```
Inserisci un numero intero
3
3
Inserisci un numero intero
5
5
Inserisci un numero intero
-1
```

B-106

6.5.2 Istruzione continue (I)

continue-statement

- `continue ;`
- provoca la terminazione di un'iterazione del ciclo che la contiene;
- salta alla parte del ciclo che valuta di nuovo la condizione di controllo:



Nota:

- le istruzioni `break` e `continue` si comportano in modo diverso rispetto al tipo di istruzione strutturata in cui agiscono;
- l'istruzione `continue` “ignora” la presenza di un eventuale istruzione `switch`.

Istruzione `switch`:

- quando è l'ultima di un ciclo, nelle alternative si può usare l'istruzione `continue` invece che l'istruzione `break`.

6.5.2 Istruzione continue (II)

// Legge cinque interi e scrive i soli non negativi

```
#include <iostream>  
using namespace std;  
int main()  
{  
    const int N = 5;  
    for (int i = 0, j; i < N; i++)  
    {  
        cout << "Inserisci un numero intero " << endl;  
        cin >> j;  
        if (j < 0) continue;  
        cout << j << endl;  
    }  
    return 0;  
}
```

Inserisci un numero intero

1

1

Inserisci un numero intero

2

2

Inserisci un numero intero

-2

Inserisci un numero intero

-3

Inserisci un numero intero

4

4

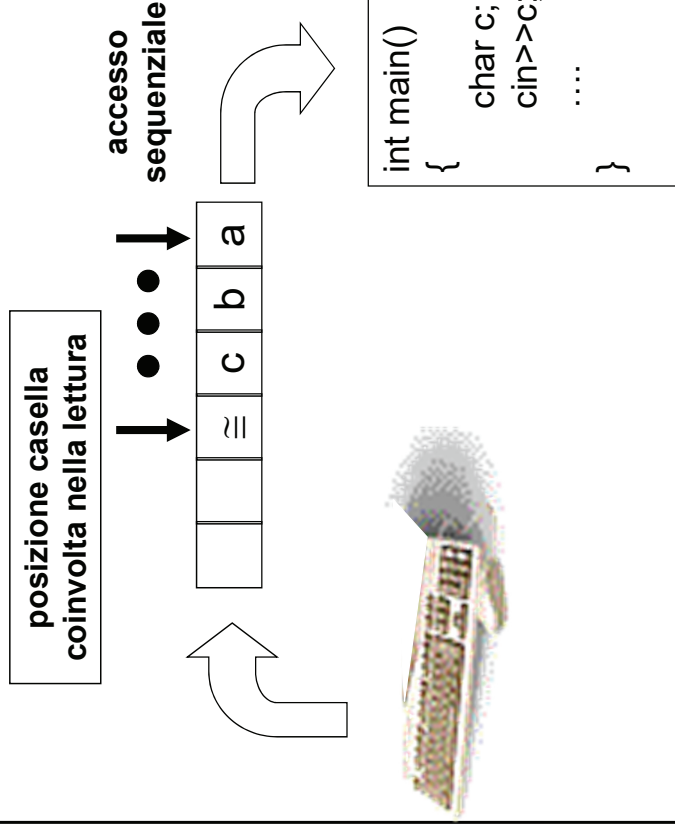
5.1 Concetto di stream (I)

Programma:

- comunica con l'esterno tramite uno o più *flussi* (stream);

Stream:

- struttura logica costituita da una sequenza di caselle (o celle), ciascuna di un byte, che termina con una marca di fine stream (il numero delle caselle è illimitato);



B-109

5.2 Concetto di stream (II)

Gli stream predefiniti sono tre:

- *cin*, *cout* e *cerr*;

Funzioni che operano su questi stream:

- si trovano in una libreria di ingresso/uscita, e per il loro uso occorre includere il file di intestazione `<iostream>`.

Osservazione:

- quanto verrà detto per lo stream *cout* vale anche per lo stream *cerr*.

Stream di ingresso standard (*cin*):

- per prelevarvi dati, si usa l'istruzione di lettura (o di ingresso):

basic-input-expression-statement
input-stream >> variable-name ;

Azioni:

- prelievo dallo stream di una sequenza di caratteri, consistente con la sintassi delle costanti associate al tipo della variabile (tipo intero: eventuale segno e sequenza di cifre, e così via);
- la conversione di tale sequenza in un valore che viene assegnato alla variabile.

B-110

5.2 Operatore di Lettura

Operatore di lettura definito per:

- singoli caratteri;
- numeri interi;
- numeri reali;
- sequenze di caratteri (costanti stringa).

Esecuzione del programma:

- quando viene incontrata un'istruzione di lettura, il programma si arresta in attesa di dati;
- i caratteri che vengono battuti sulla tastiera vanno a riempire lo stream *cin*;
- per consentire eventuali correzioni, i caratteri battuti compaiono anche in eco sul video;
- tali caratteri vanno effettivamente a far parte di *cin* solo quando viene battuto il tasto di ritorno carrello.

Ridirezione:

- col comando di esecuzione del programma, lo stream *cin* può essere ridiretto su un file *file.in* residente su memoria di massa;
- comando Linux/DOS/Windows (*leggi.exe* è un file eseguibile):
 leggi.exe < file.in

5.2 Lettura di Caratteri (I)

```
char c;  
cin >> c;
```

Azione:

- se il carattere contenuto nella casella selezionata dal puntatore non è uno spazio bianco:
 - viene prelevato;
 - viene assegnato alla variabile *c*;
 - il puntatore si sposta sulla casella successiva;
- se il carattere contenuto nella casella selezionata dal puntatore è uno spazio bianco:
 - viene ignorato;
 - il puntatore si sposta sulla casella successiva, e così via, finché non viene letto un carattere che non sia uno spazio bianco.

Letture di un carattere qualsivoglia (anche di uno spazio bianco):

```
char c;  
cin.get(c);
```

Nota:

- una funzione (come *get()*), applicata ad uno specifico stream (come *cin*), si dice funzione membro.

5.2 Lettura di Caratteri (II)

```
// Legge caratteri e li stampa su video
// Termina al primo carattere 'q'.

#include <iostream>
using namespace std;
int main()
{
    char c;
    while (true)    // while(1)
    {
        cout << "Inserisci un carattere " << endl;
        cin >> c;
        if (c != 'q')
            cout << c << endl;
        else
            break;
    }

    return 0;
}
```

```
Inserisci un carattere
a wq
a
Inserisci un carattere
w
Inserisci un carattere
```

B-113

5.2 Lettura di Caratteri (III)

```
// Legge caratteri e li stampa su video
// Termina al primo carattere 'q'.

#include <iostream>
using namespace std;
int main()
{
    char c;
    while (true)
    {
        cout << "Inserisci un carattere " << endl;
        cin.get(c);
        if (c != 'q')
            cout << c << endl;
        else
            break;
    }

    return 0;
}
```

```
Inserisci un carattere
a wq
a
Inserisci un carattere

Inserisci un carattere

Inserisci un carattere
w
Inserisci un carattere
```

B-114

5.2 Lettura di Interi

```
int i;  
cin >> i;
```

Azione:

- il puntatore si sposta da una casella alla successiva fintanto che trova caratteri consistenti con la sintassi delle costanti intere, saltando eventuali spazi bianchi in testa, e si ferma sul primo carattere non previsto dalla sintassi stessa;
- il numero intero corrispondente alla sequenza di caratteri letti viene assegnato alla variabile *i*.

```
#include <iostream>  
using namespace std;  
int main()  
{  
    int i, j;  
    cout << "Inserisci due numeri interi " << endl;  
    cin >> i;  
    cin >> j;  
    cout << i << endl << j << endl;  
  
    return 0;  
}
```

```
Inserisci due numeri interi  
-10  3  
-10  
3
```

5.2 Lettura di Reali

```
float f;  
cin >> f;
```

Azione:

- il puntatore si sposta da una casella alla successiva fintanto che trova caratteri consistenti con la sintassi delle costanti reali, saltando eventuali spazi bianchi in testa, e si ferma sul primo carattere non previsto dalla sintassi stessa;
- il numero reale corrispondente alla sequenza di caratteri letti viene assegnato alla variabile *f*.

```
#include <iostream>  
using namespace std;  
int main()  
{  
    float f, g;  
    cout << "Inserisci due numeri reali " << endl;  
    cin >> f;  
    cin >> g;  
    cout << f << endl << g << endl;  
  
    return 0;  
}
```

```
Inserisci due numeri reali  
-1.25e-3 .1e4  
-0.00125  
1000
```

5.2 Letture Multiple

Istruzione di ingresso:

- rientra nella categoria delle istruzioni espressione;
- l'espressione produce come risultato lo stream coinvolto;
- consente di effettuare più letture in sequenza.

```
cin >> x >> y;  
equivalente a:  
cin >> x;    cin >> y;
```

Infatti, essendo l'operatore >> associativo a sinistra, prima viene calcolata la subespressione `cin >>x`, che produce come risultato `cin`, quindi la subespressione `cin >>y`.

```
#include <iostream>  
using namespace std;  
int main()  
{  
    float f, g;  
    cout << "Inserisci due numeri reali " << endl;  
    cin >> f >> g;  
    cout << f << endl << g << endl;  
    return 0;  
}
```

Inserisci due numeri reali

-1.25e-3 .1e4

-0.00125

1000

5.3 Errori sullo stream di ingresso (I)

Istruzione di lettura:

- il puntatore non individua una sequenza di caratteri consistente con la sintassi delle costanti associate al tipo della variabile:
 - l'operazione di prelievo non ha luogo e lo stream si porta in uno *stato di errore*;

Caso tipico:

- si vuole leggere un numero intero, e il puntatore individua un carattere che non sia il segno o una cifra;
- caso particolare:
 - si tenta di leggere la marca di fine stream.

Stream di ingresso in stato di errore:

- occorre un ripristino, che si ottiene con la funzione `cin.clear()`.

Stream di ingresso:

- può costituire una *condizione* (nelle istruzioni condizionali o ripetitive);
- la condizione è vera se lo stream non è in uno stato di errore, falsa altrimenti.

Tastiera del terminale:

- se un programma legge dati da terminale fino ad incontrare la marca di fine stream, l'utente deve poter introdurre tale marca;
- questo si ottiene premendo Control e D nei sistemi Unix-Linux (Ctrl-D), e premendo Control e Z nei sistemi DOS/Windows (Ctrl-Z).

5.3 Errori sullo stream di ingresso (II)

```
// Legge e stampa numeri interi.  
// Termina quando viene inserito un carattere non  
// consistente con la sintassi delle costanti intere  
// Versione 1  
  
#include <iostream>  
using namespace std;  
int main()  
{  
    int i;  
    while (true)  
    {  
        cout << "Inserisci un numero intero " << endl;  
        cin >> i;  
        if (cin)  
            cout << "Numero intero: " << i << endl;  
        else  
            break;  
    }  
    return 0;  
}
```

```
Inserisci un numero intero  
3  
Numero intero: 3  
Inserisci un numero intero  
x
```

B-119

5.3 Errori sullo stream di ingresso (III)

```
// Legge e stampa numeri interi.  
// Termina quando viene inserito un carattere non  
// consistente con la sintassi delle costanti intere  
// Versione 2  
  
#include <iostream>  
using namespace std;  
int main()  
{  
    int i;  
    while (cin)  
    {  
        cout << "Inserisci un numero intero " << endl;  
        cin >> i;  
        if (cin)  
            cout << "Numero intero: " << i << endl;  
    }  
    return 0;  
}
```

```
Inserisci un numero intero  
3  
Numero intero: 3  
Inserisci un numero intero  
x
```

B-120

5.3 Errori sullo stream di ingresso (IV)

```
// Legge e stampa numeri interi.  
// Termina quando viene inserito un carattere non  
// consistente con la sintassi delle costanti intere  
// Versione 3  
  
#include <iostream>  
using namespace std;  
int main()  
{  
    int i;  
    cout << "Inserisci un numero intero " << endl;  
    while (cin >> i)  
    {  
        cout << "Numero intero: " << i << endl;  
        cout << "Inserisci un numero intero " << endl;  
    }  
    return 0;  
}
```

```
Inserisci un numero intero  
3  
Numero intero: 3  
Inserisci un numero intero  
x
```

B-121

5.3 Errori sullo stream di ingresso (V)

```
// Legge e stampa numeri interi.  
// Termina quando viene inserito un carattere non  
// consistente con la sintassi delle costanti intere  
// Versione 4  
  
#include <iostream>  
using namespace std;  
int main()  
{  
    int i;  
    while (cout << "Inserisci un numero intero " << endl  
           && cin >> i)  
        cout << "Numero intero: " << i << endl;  
    return 0;  
}
```

```
Inserisci un numero intero  
3  
Numero intero: 3  
Inserisci un numero intero  
x
```

B-122

5.3 Errori sullo stream di ingresso (VI)

```
// Legge e stampa caratteri.  
// Termina quando viene inserito il fine stream  
// ATTENZIONE! Nei sistemi operativi DOS/WINDOWS  
// usare ^Z al posto di ^D per interrompere la lettura  
  
#include <iostream>  
using namespace std;  
int main()  
{  
    char c;  
    while (cout << "Inserisci un carattere " << endl &&  
           cin >> c)  
        cout << "Carattere: " << c << endl;  
    return 0;  
}
```

```
Inserisci un carattere  
s e  
Carattere: s  
Inserisci un carattere  
Carattere: e  
Inserisci un carattere  
a  
Carattere: a  
Inserisci un carattere  
^D
```

B-123

5.3 Errori sullo stream di ingresso (VII)

```
// Legge e stampa caratteri.  
// Termina quando viene inserito il fine stream  
  
#include <iostream>  
using namespace std;  
int main()  
{  
    char c;  
    while (cout << "Inserisci un carattere " << endl &&  
           cin.get(c))  
        cout << "Carattere: " << c << endl;  
    return 0;  
}
```

```
Inserisci un carattere  
s e  
Carattere: s  
Inserisci un carattere  
Carattere:  
Inserisci un carattere  
Carattere:  
Inserisci un carattere  
Carattere: e  
Inserisci un carattere  
Carattere:  
Inserisci un carattere  
^D
```

B-124

5.3 Errori sullo stream di ingresso (VIII)

```
// Uso della funzione membro clear()
#include <iostream>
using namespace std;
int main()
{
    char c;
    cout << "Inserisci i caratteri (termina con ^D)\n";
    while (cin>>c)
        cout << c << endl;
    cout << "Inserisci i caratteri (termina con ^D)\n";
    while (cin>>c)
        cout << c << endl;
    cout << "Stream in stato di errore" << endl;
    cin.clear();
    cout << "Inserisci i caratteri (termina con ^D)\n";
    while (cin>>c)
        cout << c << endl;
    return 0;
}
```

Inserisci i caratteri (termina con ^D)

a

a

^D

Inserisci i caratteri (termina con ^D)

Stream in stato di errore

Inserisci i caratteri (termina con ^D)

^D

5.3 Errori sullo stream di ingresso (IX)

```
#include <iostream>
using namespace std;
int main()
{
    int i; char c;
    cout << "Inserisci numeri interi" << endl;
    while (cin>>i)
        cout << i << endl;
    if (!cin)
        cout << "Stream in stato di errore " << endl;
    cin.clear();
    cout << "Inserisci numeri interi" << endl;
    while (cin>>i)
        cout << i << endl;
    if (!cin)
        cout << "Stream in stato di errore" << endl;
    cin.clear();
    while (cin.get(c) && c!='\n');
    cout << "Inserisci numeri interi" << endl;
    while (cin>>i)
        cout << i << endl;
    return 0;
}
```

Inserisci numeri interi

p

Stream in stato di errore

Inserisci numeri interi

Stream in stato di errore

Inserisci numeri interi

1

1

...

5.4 Stream di uscita

Stream di uscita standard (cout):

- per scrivere su cout si usa l'istruzione di scrittura (o di uscita), che ha la forma:

basic-output-expression-statement

output-stream << expression ;

Azione:

- calcolo dell'espressione e sua conversione in una sequenza di caratteri;
- trasferimento di questi nelle varie caselle dello stream, a partire dalla prima;
- il puntatore e la marca di fine stream si spostano in avanti, e in ogni momento il puntatore individua la marca di fine stream.

Possono essere scritti:

- numeri interi;
- numeri reali;
- singoli caratteri;
- sequenze di caratteri (costanti stringa).

Nota:

- un valore di tipo booleano o di un tipo enumerato viene implicitamente convertito in intero (codifica del valore) .

5.4 Istruzione di scrittura (l)

Istruzione di uscita:

- è un'istruzione espressione;
- il risultato è lo stream;
- analogamente all'istruzione di ingresso, consente di effettuare più scritture in sequenza.

Formato di scrittura (parametri):

- *ampiezza del campo* (numero totale di caratteri impiegati, compresi eventuali spazi per l'allineamento);
- *lunghezza della parte frazionaria* (solo per i numeri reali);

Parametri:

- valori default fissati dall'implementazione;
- possono essere cambiati dal programmatore.

Ridirezione:

- col comando di esecuzione del programma, lo stream *cout* può essere ridiretto su un file *file.out* residente su memoria di massa;
- comando Linux/DOS/Windows (*scrivi.exe* è un file eseguibile):
scrivi.exe > file.out

5.4 Istruzione di scrittura (II)

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    double d = 1.564e-2, f=1.2345, i;
    cout << d << endl;
    cout << setprecision(2) << d << '\t' << f << endl;
    cout << d << endl;
    cout << setw(10) << d << ' ' << f << endl;
    cout << d << endl;
    cout << hex << 10 << '\t' << 11 << endl;
    cout << oct << 10 << '\t' << 11 << endl;
    cout << dec << 10 << '\t' << 11 << endl;
    return 0;
}
```

```
0.01564
0.016  1.2
0.016
0.016  1.2
0.016
a      b
12     13
10     11
```

B-129

5.5 Manipolazione dei file (I)

Stream associati ai file visti dal sistema operativo:

- gestiti da una apposita libreria;
- occorre includere il file di intestazione `<fstream>`.

Dichiarazione:

```
basic-file-stream-definition
fstream identifier-list ;
```

Esempio:

```
fstream ingr, usc;
```

Funzione `open()`:

- associa uno stream ad un file;
- apre lo stream secondo opportune modalità;
- le modalità di apertura sono rappresentate da opportune costanti
 - lettura => costante `ios::in`;
 - scrittura => costante `ios::out`;
 - scrittura alla fine del file (append)
=> costante `ios::out | ios::app`;
- il nome del file viene specificato come stringa (in particolare, come costante stringa).

B-130

5.5 Manipolazione dei file (II)

Esempio:

```
ingr.open("file1.in", ios::in);  
usc.open("file2.out", ios::out);
```

Stream aperto in lettura:

- il file associato deve essere già presente;
- il puntatore si sposta sulla prima casella.

Stream aperto in scrittura:

- il file associato, se non presente, viene creato;
- il puntatore si posiziona all'inizio dello stream, sul quale compare solo la marca di fine stream (eventuali dati presenti nel file vengono perduti).

Stream aperto in append:

- il file associato, se non presente, viene creato;
- il puntatore si sposta alla fine dello stream, in corrispondenza della marca di fine stream (eventuali dati presenti nel file non vengono perduti).

5.5 Manipolazione dei file (III)

Stream aperto:

- utilizzato con le stesse modalità e gli stessi operatori visti per gli stream standard;
 - le operazioni effettuate sugli stream coinvolgono i file a cui sono stati associati.
- Scrittura:
- usc << 10;
- Lettura:
- ingr >> x

Funzione close():

- chiude uno stream aperto, una volta che è stato utilizzato.
 ingr.close();
 usc.close();

Stream chiuso:

- può essere riaperto, associandolo ad un qualunque file e con una modalità arbitraria.

Fine del programma:

- tutti gli stream aperti vengono automaticamente chiusi.

Errori:

- quanto detto per lo stream *cin* vale anche per qualunque altro stream aperto in lettura.

5.5 Manipolazione dei file (IV)

```
// Scrive 4 numeri interi nel file "esempio".
// Apre il file in lettura e stampa su video il suo contenuto
#include <cstdlib>
#include <fstream>
#include <iostream>
using namespace std;
int main()
{
    fstream fs;
    int num;
    fs.open("esempio", ios::out);
    if (!fs)
    {
        cerr << "Il file non puo' essere aperto" << endl;
        exit(1); // funzione exit
    }
    for (int i = 0; i < 4; i++)
        fs << i << endl; // ATT. separare numeri
    fs.close();
    fs.open("esempio", ios::in);
    if (!fs)
    {
        cerr << "Il file non puo' essere aperto" << endl;
        exit(1);
    }
    while (fs >> num) // fino alla fine del file
        cout << num << '\t';
    cout << endl;
    return 0;
}
```

0	1	2	3
---	---	---	---

B-133

5.5 Manipolazione dei file (V)

```
// Apre in lettura il file "esempio" e legge N numeri interi.
// Controlla che le istruzioni di lettura non portino
// lo stream in stato di errore
#include <cstdlib>
#include <fstream>
#include <iostream>
using namespace std;
int main()
{
    fstream fs;
    int i, num;
    const int N = 6;
    fs.open("esempio", ios::in);
    if (!fs)
    {
        cerr << "Il file non puo' essere aperto" << endl;
        exit(1);
    }
    for (i = 0; i < N && fs >> num; i++)
        cout << num << '\t';
    cout << endl;
    if (i != N)
        cerr << "Errore nella lettura del file " << endl;

    return 0;
}
```

0	1	2	3
Errore nella lettura del file			

B-134

5.5 Manipolazione dei file (VI)

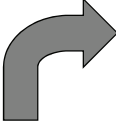
```
// Esempio apertura in append.
#include <cstdlib>
#include <fstream>
#include <iostream>
using namespace std;
int main()
{
    fstream fs;
    int i; char c;
    fs.open("esempio", ios::out);
    if (ifs)
    { cerr << "Il file non puo' essere aperto" << endl;
      exit(1);
    }
    for (int i = 0; i < 4; i++)
        fs << i << '\t';
    fs.close();
    fs.open("esempio", ios::out | ios::app);
    fs << 5 << '\t' << 6 << endl;
    fs.close();
    fs.open("esempio", ios::in);
    if (ifs)
    { cerr << "Il file non puo' essere aperto" << endl;
      exit(1);
    }
    while (fs.get(c))
        cout << c;
    return 0;
}
```

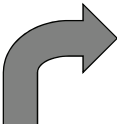
0	1	2	3	5	6
---	---	---	---	---	---

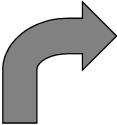
B-135

5.5 Manipolazione dei file (VII)

Versioni alternative

```
fstream fs;
fs.open("file", ios::in);
    
ifstream ifs("file");
```

```
fstream fs;
fs.open("file", ios::out);
    
ofstream ofs("file");
```

```
fstream fs;
fs.open("file", ios::out|ios::app);
    
ofstream ofs("file", ios::app);
```

B-136

7.1 Concetto di funzione (I)

```
// Problema

#include <iostream>
using namespace std;
int n;
int main()
{
    int f;
    cout << "Inserisci un numero intero: ";
    cin >> n;
    f = 1;
    for (int i = 2; i <= n; i++)
        f *= i;
    cout << "Il fattoriale e': " << f << endl;
    /* Altre elaborazioni */
    cout << "Inserisci un numero intero ";
    cin >> n;
    f = 1;
    for (int i = 2; i <= n; i++)
        f *= i;
    cout << "Il fattoriale e': " << f << endl;
    /* Altre elaborazioni */

    return 0;
}
```

Inserisci un numero intero: 4
Il fattoriale e': 24
Inserisci un numero intero: 5
Il fattoriale e': 120

B-137

7.1 Concetto di funzione (II)

```
// Soluzione 1

#include <iostream>
using namespace std;
int n;
int fatt()
{
    int ris = 1;
    for (int i = 2; i <= n; i++)
        ris *= i;
    return ris;
}

int main()
{
    int f;
    cout << "Inserisci un numero intero: ";
    cin >> n;
    f = fatt();
    cout << "Il fattoriale e': " << f << endl;
    /* Altre elaborazioni */
    cout << "Inserisci un numero intero: ";
    cin >> n;
    f = fatt();
    cout << "Il fattoriale e': " << f << endl;
    /* Altre elaborazioni */

    return 0;
}
```

Inserisci un numero intero: 4
Il fattoriale e': 24
Inserisci un numero intero: 5
Il fattoriale e': 120

B-138

7.1 Concetto di funzione (III)

```
// Soluzione 2
#include <iostream>
using namespace std;

int fatt(int n)
{
    int ris = 1;
    for (int i = 2; i <= n; i++)
        ris *= i;
    return ris;
}

int main()
{
    int i, f;
    cout << "Inserisci un numero intero: ";
    cin >> i;
    f = fatt(i);
    cout << "Il fattoriale e': " << f << endl;
    /* Altre elaborazioni */
    cout << "Inserisci un numero intero: ";
    cin >> i;
    f = fatt(i);
    cout << "Il fattoriale e': " << f << endl;
    /* Altre elaborazioni */

    return 0;
}
```

```
Inserisci un numero intero: 4
Il fattoriale e': 24
Inserisci un numero intero: 5
Il fattoriale e': 120
```

B-139

7.1 Concetto di funzione (IV)

Variabili definite in una funzione:

- *locali* alla funzione;

Nomi di variabili locali e di argomenti formali:

- associati ad oggetti che appartengono alla funzione in cui sono definiti;
- se uno stesso nome viene utilizzato in funzioni diverse, si riferisce in ciascuna funzione ad un oggetto diverso;
- in questo caso si dice che il nome è *visibile* solo nella rispettiva funzione.

Chiamata:

- gli argomenti formali e le variabili locali vengono allocati in memoria;
- gli argomenti formali vengono inizializzati con i valori degli argomenti attuali (passaggio per valore);
- gli argomenti formali e le variabili locali vengono utilizzati per le dovute elaborazioni;
- al termine della funzione, essi vengono deallocati, e la memoria da essi occupata viene resa disponibile per altri usi.

Istanza di funzione:

- nuova copia degli argomenti formali e delle variabili locali (nascono e muoiono con l'inizio e la fine della esecuzione della funzione);
- il valore di una variabile locale ottenuto durante una certa esecuzione della funzione non viene conservato per le successive istanze.

B-140

7.1 Concetto di funzione (V)

- Quando una funzione viene invocata, viene creata in memoria un'istanza della funzione;
- L'istanza ha un tempo di vita pari al tempo di esecuzione della funzione

// Istanza di funzione

Variabili locali
Argomenti formali

Esempio: funzione *int fatt(int n)* precedente

1	int ris
Valore di i	int n

Esempio: funzione *main()* precedente

...	int f
...	int i

7.3.1 Istruzione return (I)

// Restituisce il massimo termine della successione di Fibonacci minore o uguale al dato intero positivo n

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

unsigned int fibo(unsigned int n)
{
    unsigned int i = 0, j = 1, s;
    for (;;)
    {
        if ((s = i + j) > n)
            return j;
        i = j;
        j = s;
    }
}

int main()
{
    unsigned int n;
    cout << "Inserisci un numero intero " << endl;
    cin >> n;
    cout << "Termine successione Fibonacci: ";
    cout << fibo(n) << endl;

    return 0;
}
```

...	int s
1	int j
0	int i
Valore di n(main)	int n

Istanza della funzione fibo()

...	int n
-----	-------

Istanza della funzione main()

Inserisci un numero intero
12

Termine successione Fibonacci: 8

7.3.1 Istruzione return (II)

// Controlla se un intero e' positivo, negativo o nullo

```
#include <iostream>
using namespace std;
enum val {N, Z, P};
val segno(int n)
{
    if (n > 0)
        return P;
    if (n == 0)
        return Z;
    return N;
}

int main ()
{
    int i;
    // termina se legge un valore illegale per i
    while (cout << "Numero intero? " && cin >> i)
        switch (segno(i))
        {
            case N:
                cout << "Valore negativo" << endl;
                continue;
            case Z:
                cout << "Valore nullo" << endl;
                continue;
            case P:
                cout << "Valore positivo" << endl;
        }
    return 0;
}
```

B-143

7.3.1 Istruzione return (III)

// Controlla se un intero e' positivo, negativo o nullo

Numero intero? -10
Valore negativo
Numero intero? 3
Valore positivo
Numero intero? 0
Valore nullo
Numero intero? ^Z

Prima istanza funzione segno()

-10 int n

Seconda istanza funzione segno()

3 int n

Terza istanza funzione segno()

0 int n

B-144

7.4 Dichiarazioni di funzioni (I)

```
// Controlla se i caratteri letti sono lettere minuscole o
// numeri.

#include <iostream>
using namespace std;
int main()
{
    char c;
    while (cout << "Carattere:? " << endl && cin >> c)
        if (!is_low_dig(c)) // ERRORE!
        {
            return 0;
        }
}

bool is_low_dig(char c)
{
    return (c >= '0' && c <= '9' ||
            c >= 'a' && c <= 'z') ? true : false;
}
```

ERRORE SEGNALATO IN FASE DI COMPILAZIONE

9: 'is_low_dig' undeclared (first use this function)

7.4 Dichiarazioni di funzioni (II)

```
// Controlla se i caratteri letti sono lettere minuscole o
// numeri.

#include <iostream>
using namespace std;

bool is_low_dig(char c); // oppure bool is_low_dig(char);

int main()
{
    char c;
    while (cout << "Carattere:? " << endl && cin >> c)
        if (!is_low_dig(c))
        {
            return 0;
        }
}

bool is_low_dig(char c)
{
    return (c >= '0' && c <= '9' ||
            c >= 'a' && c <= 'z') ? true : false;
}
```

Carattere:?
r
Carattere:?
3
Carattere:?
A

7.6 Argomenti e variabili locali (I)

```
// Somma gli interi compresi tra i dati interi n ed m,
// estremi inclusi, con n <= m

#include <iostream>
using namespace std;

int sommalnteri(int n, int m)
{
    int s = n;
    for (int i = n+1; i <= m; i++)
        s += i;
    return s;
}

int main ()
{
    int a, b;
    while (cout << "Due interi? " && cin >> a >> b)
        // termina se legge un valore illegale per a, b
        cout << sommalnteri(a, b) << endl;

    return 0;
}
```

Due interi? 1 2
3

Due interi? 4 5
9

Due interi? s

1	int s
2	int m
1	int n

4	int s
5	int m
4	int n

B-147

7.6 Argomenti e variabili locali (II)

```
// Calcola il massimo fra tre numeri interi

#include <iostream>
using namespace std;

int massimo(int a, int b, int c)
{
    return ((a > b) ? ((a > c) ? a : c) : (b > c) ? b : c);
}

int main()
{
    int i, j, k;
    cout << "Inserisci tre numeri: ";
    cin >> i >> j >> k;
    int m = massimo (i, j, k);
    cout << m << endl;
    /* ... */
    double x, y, z;
    cout << "Inserisci tre numeri: ";
    cin >> x >> y >> z;
    double w = massimo (x, y, z);
    // Si applicano le regole standard per la conversione di tipo.
    cout << w << endl;

    return 0;
}
```

Inserisci tre numeri: 3 4 5

5

Inserisci tre numeri: 3.3 4.4 5.5

5

B-148

7.7 Funzioni void

```
// Scrive asterischi
#include <iostream>
using namespace std;

void scriviAsterischi(int n)
{
    for (int i = 0; i < n; i++)
        cout << '*';
    cout << endl;
}

int main()
{
    int i;
    cout << "Quanti asterischi? ";
    cin >> i;
    scriviAsterischi(i);

    return 0;
}
```

Quanti asterischi? 20

B-149

7.8 Funzioni senza argomenti

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

const int N = 20;
void scriviAsterischi(void) // anche void scriviAsterischi()
{
    for (int i = 0; i < N; i++)
        cout << '*';
    cout << endl;
}

int main()
{
    scriviAsterischi();

    return 0;
}
```

B-150

7.9 Funzioni ricorsive (I)

Funzione:

- può invocare, oltre che un'altra funzione, anche se stessa;
- in questo caso si ha una funzione *ricorsiva*.

Funzione ricorsiva:

- naturale quando il problema risulta formulato in maniera ricorsiva;
- esempio (fattoriale di un numero naturale n):

```
fattoriale(n) = 1      se n = 0  
n*fattoriale(n-1)    se n > 0
```

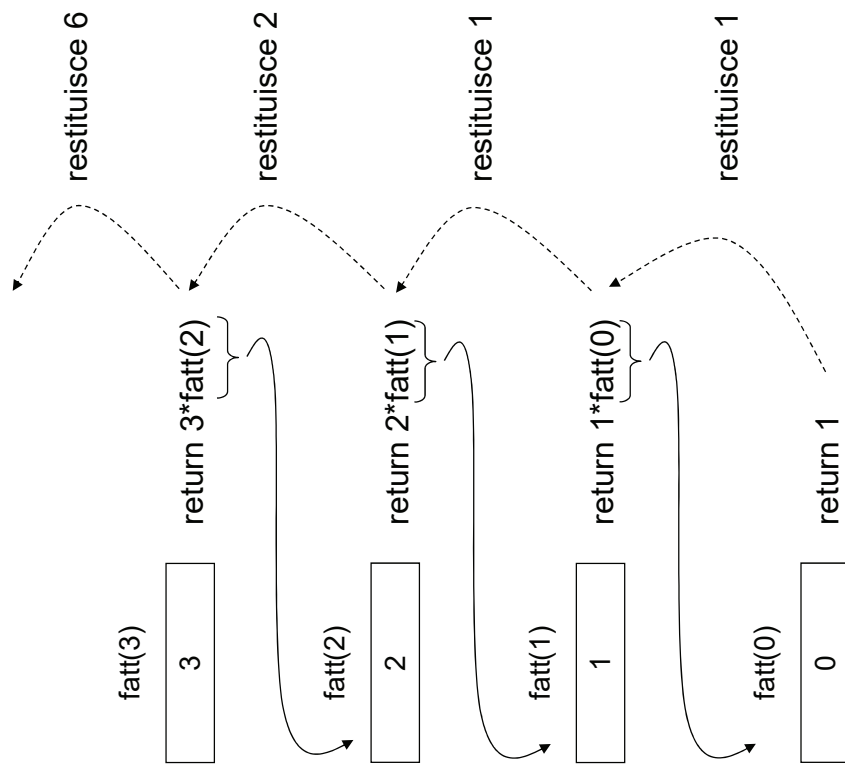
```
#include <iostream>  
using namespace std;
```

```
int fatt(int n)  
{ if (n == 0) return 1;  
  return n * fatt(n - 1);  
}
```

```
int main()  
{  
    cout << "Il fattoriale di 3 e' : " << fatt(3) << endl;  
    return 0;  
}
```

Il fattoriale di 3 e': 6

7.9 Funzioni ricorsive (II)



7.9 Funzioni ricorsive (III)

Massimo comun divisore:

```
int mcd(int alfa, int beta)
{
    if (beta == 0) return alfa;
    return mcd(beta, alfa % beta);
}
```

Somma dei primi n naturali:

```
int somma(int n)
{
    if (n == 0) return 0;
    return n + somma(n - 1);
}
```

Reale elevato a un naturale:

```
double pot(double x, int n)
{
    if (n == 0) return 1;
    return x * pot(x, n - 1);
}
```

Elementi della serie di Fibonacci:

```
int fib(int n)
{
    if (n == 1) return 0;
    if (n == 2) return 1;
    return fib(n - 1) + fib(n - 2);
}
```

B-153

7.9 Funzioni ricorsive (IV)

// Legge una parola terminata da un punto, e la scrive
// in senso inverso. Per esempio, "pippo" diventa
// "oppip".

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

```
void leggiInverti()
```

```
{
    char c;
    cin >> c;
    if (c != '.')
    {
        leggiInverti();
        cout << c;
    }
}
```

```
int main()
```

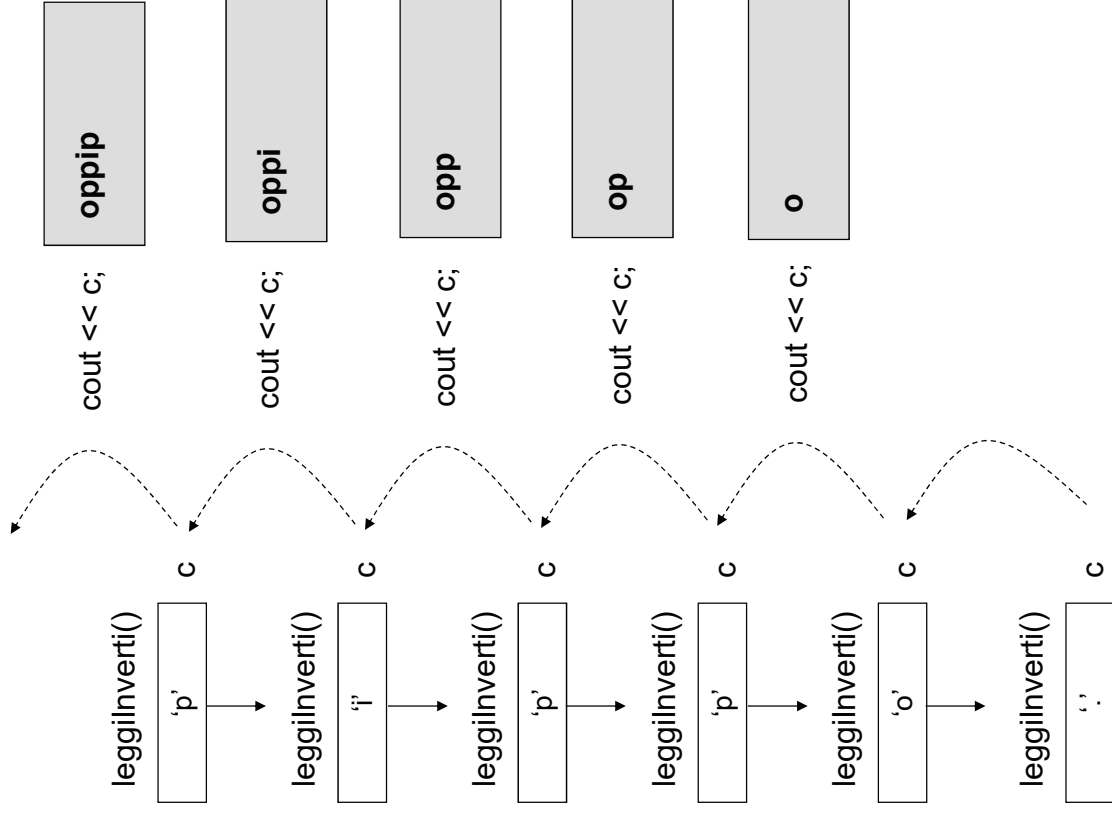
```
{
    leggiInverti();
    cout << endl;

    return 0;
}
```

```
pippo.
oppip
```

B-154

7.9 Funzioni ricorsive (V)



B-155

7.9 Funzioni ricorsive (VI)

Formulazione ricorsiva di una funzione:

- individuazione di uno o più *casi base*, nei quali termina la successione delle chiamate ricorsive;
- definizione di uno o, condizionatamente, di più passi ricorsivi;
- ricorsione controllata dal valore di un argomento di controllo, in base al quale si sceglie se si tratta di un caso base o di un passo ricorsivo;
- in un passo ricorsivo, la funzione viene chiamata nuovamente passando un nuovo valore dell'argomento di controllo;
- il risultato di questa chiamata, spesso ulteriormente elaborato, viene restituito al chiamante dell'istanza corrente;
- nei casi base, il risultato viene calcolato senza altre chiamate ricorsive.

Schema di calcolo:

- parallelo a quello usato nelle computazioni iterative.

B-156

7.9 Funzioni ricorsive (VII)

NOTA BENE:

- ogni funzione può essere formulata sia in maniera ricorsiva che in maniera iterativa;
- spesso, la formulazione iterativa è più conveniente, in termini di tempo di esecuzione e di occupazione di memoria.
- in diversi casi è più agevole (per il programmatore) esprimere un procedimento di calcolo in maniera ricorsiva;
- questo può riflettersi in una maggiore concisione e chiarezza del programma, e quindi una minore probabilità di commettere errori.

B-157

7.9 Funzioni ricorsive (Esempio)

Scrivere una funzione ricorsiva che stampi su uscita standard un triangolo rettangolo rovesciato composto di asterischi. I due cateti del triangolo contengono lo stesso numero N di asterischi. Nell'esempio seguente N = 3.

```
* * *  
* *  
*  

```

B-158

```
void scrivi(int n)
{
    if (n==0) return;
    for (int i=0; i<n; i++)
        cout << 'i';
    cout << endl;
    scrivi(n-1);
}
```

Libreria:

- insieme di funzioni (sottoprogrammi) precompilate;
- formata da coppie di file;
- per ogni coppia un file contiene alcuni sottoprogrammi compilati ed uno contiene le dichiarazioni dei sottoprogrammi stessi (quest'ultimo è detto file di intestazione e il suo nome termina tipicamente con l'estensione *h*).

Utilizzo di funzioni di libreria:

- nella fase di scrittura del programma, includere il file di intestazione della libreria usando la direttiva `#include`;
- nella fase di collegamento, specificare la libreria da usare, secondo le convenzioni dell'ambiente di sviluppo utilizzato.

Esempio:

- programma contenuto nel file *mioprogramma.cpp*, che usa delle funzioni della libreria matematica;
- deve contenere la direttiva:
`#include <cmath>`
- Alcune librerie C++ sono disponibili in tutte le implementazioni e contengono gli stessi sottoprogrammi.

3.11 Libreria standard

- *abs(n)* valore assoluto di n;
- *rand()* intero pseudocasuale compreso fra 0 e la costante predefinita *RAND_MAX*;
- *srand(n)* inizializza la funzione *rand()*.

cctype

Restituiscono un valore booleano

- *isalnum(c)* lettera o cifra;
- *isalpha(c)* lettera;
- *isdigit(c)* cifra;
- *islower(c)* lettera minuscola;
- *isprint(c)* carattere stampabile, compreso lo spazio;
- *isspace(c)* spazio, salto pagina, nuova riga, ritorno carrello, tabulazione orizzontale, tabulazione verticale;
- *isupper(c)* lettera maiuscola;

B-161

3.11 Libreria standard

cmath

- funzioni trigonometriche (x è un double)
 - *sin(x)* seno di x;
 - *cos(x)* coseno di x;
 - *tan(x)* tangente di x;
 - *asin(x)* arcoseno di x;
 - *acos(x)* arcocoseno di x
 - *atan(x)* arcotangente di x
- funzioni esponenziali e logaritmiche
 - *exp(x)* e elevato alla x;
 - *log(x)* logaritmo in base e di x;
 - *log10(x)* logaritmo in base 10 di x;
- altre funzioni (anche y è un double)
 - *fabs(x)* valore assoluto di x;
 - *ceil(x)* minimo intero maggiore o uguale a x;
 - *floor(x)* massimo intero minore o uguale a x;
 - *pow(x, y)* x elevato alla y;
 - *sqrt(x)* radice quadrata di x;

B-162

8.1 Tipi derivati

Tipi fondamentali:

- da questi si possono *derivare* altri tipi;
- dal tipo *int* si deriva il tipo *puntatore a int*.
 - variabile appartenente a questo tipo: può assumere come valori indirizzi di interi.
- dal tipo *int* si deriva il tipo *array di 4 int*:
 - variabile appartenente a questo tipo: può assumere come valori 4 interi.

Meccanismi di derivazione:

- possono essere composti fra di loro, permettendo la definizione di tipi derivati arbitrariamente complessi;
- prendendo gli interi come base, si possono definire array di puntatori a interi, puntatori ad array di interi, array di array di interi, eccetera.

Tipi derivati:

- riferimenti;
- puntatori;
- array;
- strutture;
- unioni;
- classi.

B-163

8.2 Riferimenti (I)

Riferimento:

- identificatore che individua un oggetto;
- riferimento default: il nome di un oggetto, quando questo è un identificatore.
- oltre a quello default, si possono definire altri riferimenti di un oggetto (sinonimi o alias).

Tipo riferimento:

- possibili identificatori di oggetti di un dato tipo (il tipo dell'oggetto determina il tipo del riferimento).

Dichiarazione di un tipo riferimento e definizione di un riferimento sono contestuali.

Sintassi:

```
basic-reference-definition  
reference-type-indicator identifier  
reference-type-indicator  
type-indicator &  
reference-initializer  
= object-name
```

- indicatore di tipo:
 - *specifica tipo dell'oggetto riferito;*

Non si possono definire riferimenti di riferimenti.

B-164

8.2 Riferimenti (II)

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

int main ()
{
    int i = 10;
    int &r = i;
    int &s = r;
    int &s;          ERRORE, deve essere sempre iniz.
    int &s = 10;      ERRORE
    cout << i << "i" << r << "r" << s << endl; // 10 10 10
    r++;
    cout << i << "i" << r << "r" << s << endl; // 11 11 11

    int h = 0, k = 1;
    int &r1 = h, &r2 = k; // due riferimenti
    int &r3 = h, j=3;     // un riferimento ed un intero

    return 0;
}
```

10	10	10
11	11	11

112	1	k, r2
116	0	h, r1, r3
120	10	i, r, s

8.2.1 Riferimenti const (I)

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

int main () {
    int i = 1;
    const int &r = i; // Notare: i non è costante

    cout << r;        // 1
    r = 2;            ERRORE!
    i = 2;            // OK
    cout << r;        // 2

    int j = 10;
    j = r;            // OK
    cout << j << endl; // 2

    const int k = 10;
    const int &t = k; // OK
    cout << t << endl; // 10
    int &tt = k;      ERRORE!

    return 0;
}
```

1
2
2
10

112	10	k, t
116	10	j
120	1	i, r

8.2.2 Riferimenti come argomenti (I)

```
// Scambia interi (ERRATO)
#include <iostream>
using namespace std;

void scambia (int a, int b)
{ // scambia gli argomenti attuali
  int c = a;
  a = b;
  b = c;
}

int main ()
{
  int i, j;
  cout << "Inserisci due interi: " << endl;
  cin >> i >> j;
  scambia (i, j);
  cout << i << '\t' << j << endl;

  return 0;
}
```

Questa qui sotto è la situazione un attimo prima che la funzione termini

Istanza (scambia)		Istanza (scambia)	
96	3	96	2
100	2	100	3
Istanza (main)		Istanza (main)	
116	3	116	3
120	2	120	2

8.2.2 Riferimenti come argomenti (II)

Argomento di una funzione:

- può essere di un tipo riferimento;
- in questo caso:
 - l'argomento formale corrisponde a un contenitore senza nome, che ha per valore il riferimento;
 - nel corpo della funzione, ogni operazione che coinvolge l'argomento formale agisce sull'entità riferita.

Chiamata della funzione:

- il riferimento argomento formale viene inizializzato con un riferimento del corrispondente argomento attuale;

Argomenti riferimento:

- devono essere utilizzati quando l'entità attuale può essere modificata.

In sintesi:

- la funzione agisce sulle entità riferite dagli argomenti attuali.

8.2.2 Riferimenti come argomenti (III)

```
// Scambia interi (trasmissione mediante riferimenti)
#include <iostream>
using namespace std;

void scambia (int &a, int &b)
{ // scambia i valori degli oggetti riferiti
  int c = a;
  a = b;
  b = c;
}

int main ()
{
  int i, j;
  cout << "Inserisci due interi: " << endl;
  cin >> i >> j;
  scambia (i, j);
  cout << i << ' ' << j << endl;

  return 0;
}
```

Istanza (scambia)

112	c
-----	---

Situazione nel main
quando la funzione
termina

Istanza (main)

116	3	j, b	116	2	j
120	2	i, a	120	3	i

8.2.3 Riferimenti const come argomenti (I)

```
// Calcolo dell'interesse (trasmissione mediante
// riferimenti di oggetti costanti)

#include <iostream>
using namespace std;

float interesse(int importo, const float& rateo)
{
  float inter = rateo*importo; //OK
  // rateo += 0.5;             ERRORE!
  return inter;
}

int main()
{
  cout << "Interesse : " << interesse(1000,1.2) << endl;

  return 0;
}
```

Interesse : 1200

8.2.2 Riferimenti risultato di funzione (I)

```
// Riferimenti come valori restituiti (i)

#include <iostream>
using namespace std;

int& massimo(int &a, int &b)
{
    return a > b ? a : b;
}

int main ()
{
    int i, j;
    cout << "Inserisci due interi: " << endl;
    cin >> i >> j;
    cout << "Valore massimo ";
    cout << massimo(i, j) << endl;
    massimo(i, j) = 5;
    cout << i << '\t' << j << endl;
    massimo(i, j)++;
    cout << i << '\t' << j << endl;

    return 0;
}
```

Inserisci due interi:

1 3

Valore massimo 3

1 5

1 6

B-171

8.2.2 Riferimenti risultato di funzioni (II)

```
// Riferimenti come valori restituiti (ii)

#include <iostream>
using namespace std;

int& massimo(int &a, int &b)
{
    int &p = a > b ? a : b;
    return p;
}

// OK. Restituisce un riferimento

// ~~~~~
// Riferimenti come valori restituiti (iii)

#include <iostream>
using namespace std;

int& massimo(int a, int b)
{
    return a > b ? a : b;
}

// ERRORE. Riferimento ad un argomento attuale che
// viene distrutto
```

NOTA BENE: l'errore non è segnalato dal compilatore.

B-172

8.2.3 Riferimenti risultato di funzioni (III)

```
// Riferimenti const come valori restituiti (iii)

#include <iostream>
using namespace std;

const int& massimo(const int& a, const int& b)
{
    return a > b ? a : b;
}

int main ()
{
    int i, j;
    cout << "Inserisci due interi: " << endl;
    cin >> i >> j;
    cout << "Valore massimo ";
    cout << massimo(i, j) << endl;

    // massimo(i, j) = 5;      ERRORE

    return 0;
}
```

```
Inserisci due interi:
1 3
Valore massimo 3
```

B-173

8.2.3 Riferimenti (IV)

Argomento formale di una funzione e risultato prodotto da una funzione:

- possono essere riferimenti con l'attributo *const*.

Argomento formale con l'attributo *const*:

- può avere come corrispondente un argomento attuale senza tale attributo, ma non è lecito il contrario.

Risultato con l'attributo *const*:

- una istruzione *return* può contenere un'espressione senza tale attributo, ma non è lecito il contrario.

Operatore *const_cast*:

- converte un riferimento *const* in un riferimento non *const*.

B-174

8.2.3 Riferimenti (V)

```
// Riferimenti come valori restituiti (i)
#include <iostream>
using namespace std;

int& massimo(const int &a, const int &b)
{
    return const_cast<int&>(a > b ? a : b);
}

// ERRATO
// int& massimoErrato(const int& a, const int& b)
// {
//     return a > b ? a : b;
// }

int main ()
{
    int i, j;
    cout << "Inserisci due interi: " << endl;
    cin >> i >> j;
    cout << "Valore massimo ";
    cout << massimo(i, j) << endl;
    massimo(i, j) = 5;
    cout << i << '\t' << j << endl;

    return 0;
}
```

```
Inserisci due interi:
1 3
Valore massimo 3
1 5
```

B-175

8.2.3 Riferimenti (VI)

Esempio:

- il risultato della funzione è di tipo *int&*;
- nella istruzione *return* compare un riferimento *const*;
- si rende opportuna una conversione esplicita di tipo:

```
int& maxr1(const int& ra, const int& rb)
{
    if (ra >= rb) return const_cast<int&>( ra);
    return const_cast<int&>( rb);
}

const int& maxr1(const int& ra, const int& rb)
{
    if (ra >= rb) return const_cast<int&>( ra);
    return rb;
}
```

B-176

8.3 Puntatori (I)

Puntatore:

- oggetto il cui valore rappresenta l'indirizzo di un altro oggetto o di una funzione.

Tipo puntatore:

- insieme di valori: indirizzi di oggetti o di funzioni di un dato tipo (il tipo dell'oggetto o della funzione determina il tipo del puntatore).

Dichiarazione di un tipo puntatore e definizione di un puntatore sono contestuali.

// Definizione di puntatori

```
#include <iostream>
using namespace std;
int main ()
{
    int *p1;           // puntatore a interi
    int* p2;
    int * p3;

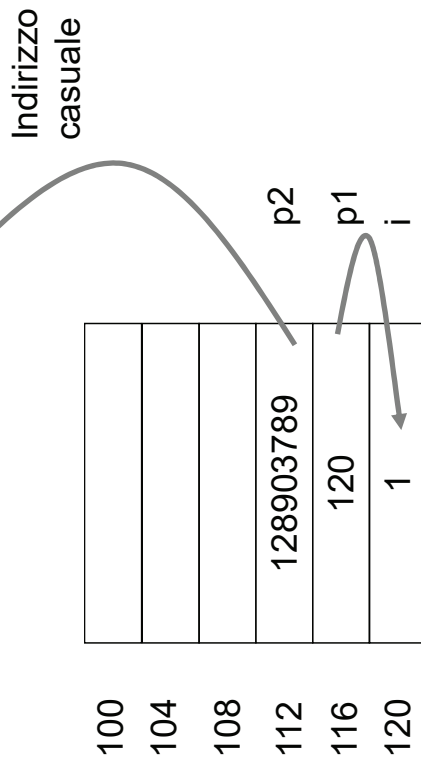
    int *p4, *p5;      // due puntatori
    int *p6, i1;        // un puntatore ed un intero
    int i2, *p7;        // un intero ed un puntatore

    return 0;
}
```

B-177

8.3 Puntatori (II)

```
#include <iostream>
using namespace std;
int main ()
{
    int i = 1;
    int *p1 = &i;      // operatore indirizzo
    int *p2;
    ...
}
```



B-178

8.3 Puntatori (III)

```
// Operatore indirizzo e operatore di indirezione
#include <iostream>
using namespace std;
int main ()
{
    int i = 1;           // operatore indirizzo
    int *p1 = &i;        // operatore di indirezione
    *p1 = 10;            // restituisce un l-value
    int *p2 = p1;        // Due puntatori allo stesso oggetto

    cout << i << "\t" << *p1 << "\t" << *p2 << endl;

    *p2 = 20;
    cout << i << "\t" << *p1 << "\t" << *p2 << endl;

    cout << p1 << "\t" << p2 << endl;
    cout << &p1 << "\t" << &p2 << endl;

    // int *p3 = i;      // ERRORE: assegna un int ad un punt.
    // i = p1;           // ERRORE: assegna un punt. ad un int.

    // p2 = *p1;         // ERRORE: assegna un int ad un punt.
    // *p2 = p1          // ERRORE: assegna un punt. ad un int.

    int *p3;
    *p3 = 2;            // ATTENZIONE: puntatore non iniz.
}
```

```
10  10  10
20  20  20
120 120
116 112
```

B-179

8.3 Puntatori (IV)

```
// Puntatori allo stesso oggetto
#include <iostream>
using namespace std;
int main ()
{
    char a, b;
    char *p = &a, *q = &b;
    cout << "Inserisci due caratteri " << endl;
    cin >> a >> b;
    *p = *q;
    cout << a << "\t" << b << endl;    // Esempio: 'b'
    cout << *p << "\t" << *q << endl;    // Esempio: 'b'

    cout << "Inserisci due caratteri " << endl;
    cin >> a >> b;
    p = q;
    cout << a << "\t" << b << endl;    // Esempio: 'c'
    cout << *p << "\t" << *q << endl;    // Esempio: 'f'

    return 0;
}
```

```
Inserisci due caratteri
ab
b  b
b  b
Inserisci due caratteri
cf
c  f
f  f
```

B-180

8.3 Puntatori (V)

```
// Puntatori a costanti

#include <iostream>
using namespace std;
int main ()
{
    int i = 0;           // Nessuna inizializzazione
    const int *p;        // OK
    p = &i;              // N.B.: i non e' costante
    // N.B.: i non e' costante

    int j;
    j = *p;              // OK
    *p = 1;              // ERRORE! Il valore di i non puo'
                        // essere modificato attraverso p

    const int k = 10;
    const int *q = &k;   // OK

    int *qq;
    qq = &k;
    qq = q;

    return 0;
}
```

B-181

8.3 Puntatori (VI)

```
// Puntatori costanti

#include <iostream>
using namespace std;
int main ()
{
    char c = 'a';
    char *const p = &c;
    cout << *p << endl;    // Sempre inizializzato
                        // 'a'

    *p = 'b';
    cout << *p << endl;    // 'b'

    char d = 'c';
    p = &d;                 // ERRORE!

    char *p1, *const p2 = &d;
    p1 = p;
    cout << *p1 << endl;   // 'b'
    p = p1;                 // ERRORE
    p = p2;                 // ERRORE!

    return 0;
}
```

a
b
b

B-182

8.3 Puntatori (VII)

```
// Puntatore a puntatore

#include <iostream>
using namespace std;
int main (){
    int i = 2, j = 7;
    int *pi = &i, *pj = &j;

    int **q1 = &pi;           // visualizza 2
    cout << **q1 << endl;
    *q1 = pj;                 // visualizza 7
    cout << **q1 << endl;

    pi = &i;                  // ripristino il valore
                                // del puntatore pi

    int **q2 = nullptr;
    q2 = &pi;                 // visualizza 2
    cout<<**q2<<endl;

    q2 = &pj;                  // visualizza 7
    cout<<**q2<<endl;

    // int **q3;   // punt. a punt. non inizializzato.
    // *q3 = pi;   // Questa istruzione provocherebbe un
                  // errore semantico gravissimo: andrei a
                  // sovrascrivere una zona di memoria
                  // casuale, non nota al programmatore.

    return 0;
}
```

2
7
2
7

B-183

8.3 Puntatori (VIII)

```
// Puntatori nulli

#include <iostream>
using namespace std;
int main (){
    int *p = nullptr;         // equivale a scrivere: int *p = 0;

    *p = 1;                   // ERRORE A TEMPO DI
                                // ESECUZIONE!

    if (p == nullptr)
        cout << "Puntatore nullo " << endl;
    if (p == 0)
        cout << "Puntatore nullo " << endl;
    if (!p)
        cout << "Puntatore nullo " << endl;

    return 0;

    // Quando non esisteva la keyword nullptr, veniva
    // usato il simbolo NULL, definito mediante la seguente
    // direttiva del
    // pre-processore:

    #define NULL 0

    // Torneremo sulle direttive del pre-processore alla
    // fine del corso.

    // NB: Il simbolo NULL viene definito da molte delle
    // librerie standard, iostream inclusa
}
```

B-184

8.3 Operazioni sui puntatori (I)

Operazioni possibili:

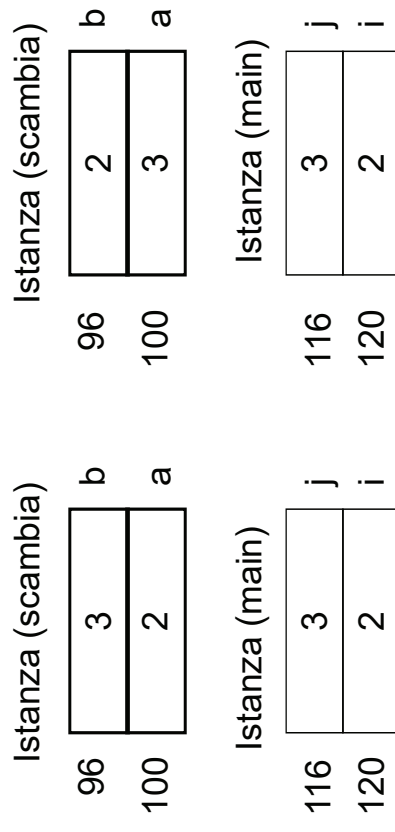
- assegnamento di un'espressione che produce un valore indirizzo, ad una variabile puntatore;
- uso di un puntatore per riferirsi all'oggetto puntato;
- confronto fra puntatori mediante gli operatori '==' e '!=';
- Stampa su uscita standard utilizzando l'operatore di uscita '<<'. In questo caso, viene stampato il valore in esadecimale del puntatore ossia l'indirizzo dell'oggetto puntato.

Un puntatore può costituire un argomento di una funzione:

- nel corpo della funzione, per mezzo di indirezioni, si possono modificare gli oggetti puntati.

8.3.4 Puntatori come argomenti (I)

```
// Scambia interi (ERRATA!)\n\n#include <iostream>\nusing namespace std;\n\nvoid scambia (int a, int b)\n{\n    // scambia gli argomenti attuali\n    int c = a;\n    a = b;\n    b = c;\n}\n\nint main (){\n    int i, j;\n    cout << "Inserisci due interi: " << endl;\n    cin >> i >> j;\n    scambia (i, j);\n    cout << i << ' ' << j << endl;    // Esempio: 2   3\n\n    return 0;\n}
```



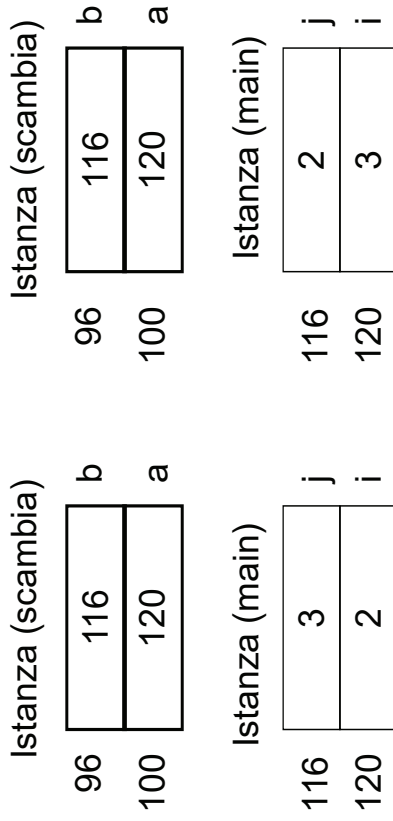
8.3.4 Puntatori come argomenti (II)

```
// Scambia interi (corretta)
#include <iostream>
using namespace std;

void scambia (int *a, int *b)
{ // scambia i valori degli oggetti puntati
  int c = *a;
  *a = *b;
  *b = c;
}

int main (){
  int i = 2, j = 3;
  scambia (&i, &j);
  cout << i << "i" << j << endl; // visualizza: 3 2

  return 0;
}
```



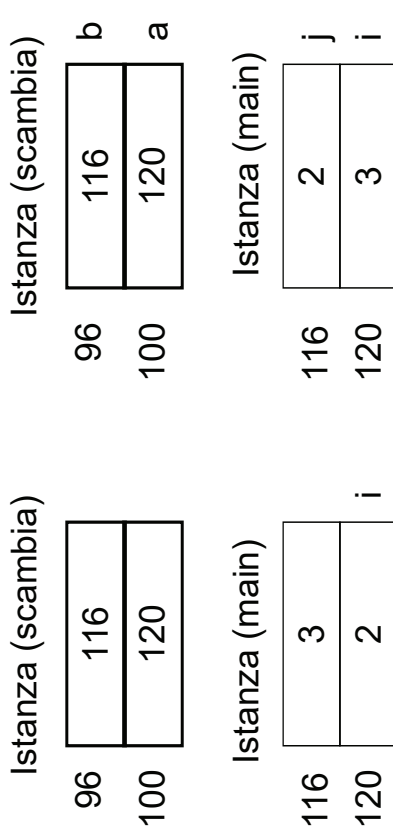
B-187

8.3.4 Puntatori come argomenti (III)

```
// Scambia interi corretta
#include <iostream>
using namespace std;

void scambia (int *a, int *b)
{ // scambia i valori degli oggetti puntati
  int c = *a;
  *a = *b;
  *b = c;
}

int main (){
  int i = 2, j = 3;
  int *p=&i, *q = &j;
  scambia (p, q); // le variabili puntatore p e q vengono
                  // passate alla funzione per valore
  cout << i << "i" << j << endl; // visualizza: 3 2
}
```



B-188

8.3.4 Puntatori come argomenti (IV)

```
// Incrementa il maggiore tra due interi

#include <iostream>
using namespace std;
void incrementa(int *a, int *b)
{
    if (*a > *b)
        (*a)++;
    else
        (*b)++;
}

int main ()
{
    int i, j;
    cout << "Inserisci due interi: " << endl;
    cin >> i >> j;
    incrementa(&i, &j);
    cout << i << " " << j << endl;    // Esempio: 2 4

    return 0;
}
```

Istanza (incrementa)		Istanza (incrementa)	
96	116	96	116
100	120	100	120
Istanza (main)		Istanza (main)	
116	3	116	4
120	2	120	2

B-189

8.3.4 Puntatori come risultato di funzioni (I)

```
// Puntatori come valori restituiti (i)

#include <iostream>
using namespace std;
int* massimo(int *a, int *b)
{
    return *a > *b ? a : b;
}

int main ()
{
    int i, j;
    cout << "Inserisci due interi: " << endl;
    cin >> i >> j;
    cout << "Valore massimo ";
    cout << *massimo(&i, &j) << endl;
    // Esempio: 2 3

    *massimo(&i, &j) = 5;
    cout << i << " " << j << endl;    // Esempio: 2 5

    // massimo(&i, &j)++;
    // DAREBBE ERRORE: il valore restituito non e' un l-value

    (*massimo(&i, &j))++;
    cout << i << " " << j << endl;    // Esempio: 2 6

    return 0;
}
```

Inserisci due interi:

2 3

Valore massimo 3

2 5

2 6

B-190

8.3.4 Puntatori come risultato di funzioni (II)

```
// Puntatori come valori restituiti (ii)

#include <iostream>
using namespace std;

int* massimo(int *a, int *b)
{
    int *p = *a > *b ? a : b;
    return p;
    // Restituisce un puntatore
}

// ~~~~~
// Puntatori come valori restituiti (iii)

#include <iostream>
using namespace std;

int* massimo(int *a, int *b)
{
    int i = *a > *b ? *a : *b;
    return &i;
    // ATTENZIONE: restituisce l'indirizzo di una variabile
    // locale
    // [Warning] address of local variable 'i' returned
}
```

B-191

8.3.4 Puntatori come argomenti costanti (I)

```
// Trasmissione dei parametri

#include <iostream>
using namespace std;

int* massimo(int *a, int *b)
{
    return *a > *b ? a : b;
}

int main ()
{
    int i = 2;
    const int N = 3;
    cout << "Valore massimo ";
    cout << *massimo(&i, &N) << endl;
    // ERRORE: invalid conversion from 'const int*' to 'int*'

    return 0;
}
```

B-192

8.3.4 Puntatori come argomenti costanti (II)

```
// Trasmissione di parametri mediante puntatori a
// costanti

#include <iostream>
using namespace std;

int* massimo(const int *a, const int *b)
{
    return const_cast<int*> (*a > *b ? a : b);
}

/*
int* massimo(const int *a, const int *b)
{
    return *a > *b ? a : b;
    // ERRORE: invalid conversion from 'const int*' to 'int*'
} */

int main ()
{
    int i = 2;
    const int N = 3;
    cout << "Valore massimo ";
    cout << *massimo(&i, &N) << endl;

    return 0;
}
```

B-193

8.3.4 Puntatori come risultato di funzioni

```
// Puntatori a costanti come valori restituiti

#include <iostream>
using namespace std;

const int* massimo(const int *a, const int *b)
{
    return *a > *b ? a : b;
}

int main ()
{
    int i = 2;
    const int N = 3;
    cout << "Valore massimo ";
    cout << *massimo(&i, &N) << endl;

    // int *p1 = massimo(&i, &N);    ERRORE

    const int *p2 = massimo(&i, &N);

    // *massimo(&i, &N) = 1;        ERRORE

    return 0;
}
```

Valore massimo 3

B-194

9.1 Tipi e oggetti array (I)

Array di dimensione n :

- n -upla ordinata di elementi dello stesso tipo allocati in celle contigue della memoria, e ai quali ci si riferisce mediante un indice, che rappresenta la loro posizione all'interno dell'array.

Tipo dell'array:

- dipende dal tipo degli elementi.

Dichiarazione di un tipo array e definizione di un array sono contestuali.

// Somma gli elementi di un dato vettore di interi

```
#include <iostream>
using namespace std;
int main () {
    const int N = 5;           // dimensione del vettore costante
    int v[N];
    cout << "Inserisci 5 numeri interi " << endl;
    for (int i = 0; i < N; i++)
        cin >> v[i];          // operatore di selezione con indice
    int s = v[0];              // restituisce un l-value
    for (int i = 1; i < N; i++)
        s += v[i];
    cout << s << endl;

    return 0;
}
```

Inserisci 5 numeri interi

1 2 3 4 5
15

B-195

9.1 Tipi e oggetti array (II)

ATTENZIONE: l'identificatore dell'array identifica l'indirizzo del primo elemento dell'array

$v = \&v[0]$

Nell'esempio, $v = 104$;

104	0	$v[0]$
108	1	$v[1]$
112	2	$v[2]$
116	3	$v[3]$
120	4	$v[4]$

B-196

9.1 Tipi e oggetti array (III)

```
// Inizializzazione degli array

#include <iostream>
using namespace std;
int main ()
{
    const int N = 6;
    int a[] = {0, 1, 2, 3};           // array di 4 elementi
    int b[N] = {0, 1, 2, 3};         // array di N elementi
    cout << "Dimensioni array: ";
    cout << sizeof a << '\t' << sizeof b << endl; // 16 24
    cout << "Numero di elementi: ";
    cout << sizeof a / sizeof(int) << '\t';      // 4
    cout << sizeof b / sizeof(int) << endl;      // 6
    // ERRORE! NON SEGNALATO IN COMPILAZIONE
    // Nessun controllo sul valore degli indici
    for (int i = 0; i < N; i++)
        cout << a[i] << '\t';
    cout << endl;

    for (int i = 0; i < N; i++)
        cout << b[i] << '\t';
    cout << endl;

    return 0;
}
```

Dimensioni array:	16	24			
Numero di elementi:	4	6			
0	1	2	3	37879712	2009179755
0	1	2	3	0	0

B-197

9.1 Tipi e oggetti array (V)

```
// Operazioni sugli array. NON SONO PERMESSE
// OPERAZIONI ARITMETICHE, DI CONFRONTO, DI
// ASSEGNAMENTO

#include <iostream>
using namespace std;

int main ()
{
    const int N = 5;
    int u[N] = {0, 1, 2, 3, 4}, v[N] = {5, 6, 7, 8, 9};
    // v = u;      ERRORE: assegnamento non permesso
    cout << "Ind. v:" << v << "\t Ind. u: " << u << endl;

    if (v == u)           // Attenzione confronta gli indirizzi
        cout << "Array uguali " << endl;
    else
        cout << "Array diversi " << endl;
    if (v > u) // operatori di confronto agiscono sugli indirizzi
        cout << "Indirizzo v > u " << endl;
    else
        cout << "Indirizzo v <= u " << endl;

    // v + u;      operatori aritmetici non definiti

    return 0;
}
```

Ind. v:	0x22ff18	Ind. u:	0x22ff38
Array diversi			
Indirizzo v <= u			

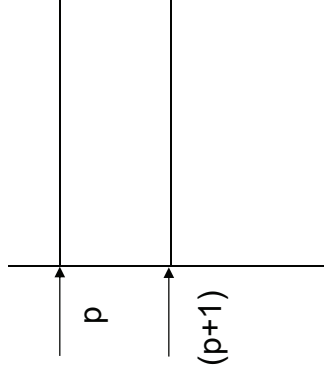
B-198

8.3.3 Array e puntatori (I)

// Aritmetica dei puntatori

Permette di calcolare indirizzi con la regola seguente:

- se l'espressione p rappresenta un valore indirizzo di un oggetto di tipo T , allora l'espressione $(p+1)$ rappresenta l'indirizzo di un oggetto, sempre di tipo T , che si trova consecutivamente in memoria.



In generale:

- se i è un intero, allora l'espressione $(p+i)$ rappresenta l'indirizzo di un oggetto, sempre di tipo T , che si trova in memoria, dopo i posizioni.

Nota:

- Se l'espressione p ha come valore *addr* e se T occupa n locazioni di memoria, l'espressione $p+i$ ha come valore $addr+n*i$.

Aritmetica dei puntatori:

- si utilizza quando si hanno degli oggetti dello stesso tipo in posizioni adiacenti in memoria (array).

8.3.3 Array e puntatori (II)

```
// Aritmetica dei puntatori
#include <iostream>
using namespace std;
int main ()
{
    int v[4];
    int *p = v;
    // v ==> &v[0]

    *p = 1;
    *(p + 1) = 10;
    p += 3;
    *(p - 1) = 100;
    *p = 1000;
    p = v;
    cout << "v[" << 4 << "]" = "[" << *p;
    for (int i = 1; i < 4; i++)
        cout << 't' << *(p + i);    // v[4] = [1 10 100 1000]
    cout << ']' << endl;
    cout << p + 1 - p << endl;    // 1 aritmetica dei puntatori
    cout << int(p + 1) - int(p) << endl;    // 4 (byte)

    char c[5];
    char* q = c;
    cout << int(q + 1) - int(q) << endl;    // 1 (byte)

    int* p1 = &v[1];
    int* p2 = &v[2];
    cout << p2 - p1 << endl;    // 1 (elementi)
    cout << int(p2) - int(p1) << endl;    // 4 (byte)

    return 0;
}
```


8.3.3 Array e puntatori (III)

```
// Inizializza a 1
#include <iostream>
using namespace std;
int main ()
{
    const int N = 5;
    int v[N];

    int *p = v;
    while (p <= &v[N-1])
        *p++ = 1; // equivalente a: *(p++) = 1;

    p = v;
    cout << "v[" << N << "]" = [" << *p;
    for (int i = 1; i < N; i++)
        cout << 't' << *(p + i); // v[5] = [1 1 1 1 1]
    cout << ']' << endl;

    return 0;
}
```

v[5] = [1 1 1 1 1]

B-201

9.2 Array multidimensionali (I)

```
// Legge e scrive gli elementi di una matrice di R righe
// e C colonne

#include <iostream>
using namespace std;

int main ()
{
    const int R = 2;
    const int C = 3;
    int m[R][C];

    cout << "Inserisci gli elementi della matrice" << endl;
    for (int i = 0; i < R; i++)
        for (int j = 0; j < C; j++)
            cin >> m[i][j];

    int* p = &m[0][0];
    for (int i = 0; i < R; i++)
    {
        for (int j = 0; j < C; j++)
            cout << *(p + i*C + j) << 't';
        cout << endl;
    }

    return 0;
}
```

104	1	m[0][0]
108	2	m[0][1]
112	3	m[0][2]
116	4	m[1][0]
120	5	m[1][1]
124	6	m[1][2]

// 1 2 3 4 5 6

// anche: m[0]

Inserisci gli elementi della matrice

1 2 3 4 5 6
1 2 3
4 5 6

B-202

9.2 Array multidimensionali (II)

```
// Inizializzazione di vettori multidimensionali

#include <iostream>
using namespace std;
int main ()
{
    int m1[2][3] = {1, 2, 3, 4, 5, 6}; // anche: int m1[][3]
    cout << "Matrice m1 " << endl;
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 3; j++)
            cout << m1[i][j] << "t";
        cout << endl;
    }
    int m2[3][3] = {{0, 1, 2}, {10, 11}, {100, 101, 102}};
    // anche: int m2[][3]. N.B.: m2[1][2] inizializzato a 0
    cout << "Matrice m2 " << endl;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
            cout << m2[i][j] << "t";
        cout << endl;
    }
    return 0;
}
```

```
Matrice m1
1 2 3
4 5 6
Matrice m2
0 1 2
10 11 0
100 101 102
```

B-203

9.4 Array come argomenti di funzioni (I)

```
// Somma gli elementi di un dato vettore di interi (i)
#include <iostream>
using namespace std;

int somma(int v[4]){
    // Il valore 4 (dimensione)
    // viene ignorato!

    // firma alternativa, del tutto equivalente:
    // int somma(int v[])
    // int somma(int *v)

    int s = 0;
    for (int i = 0; i < 4; i++)
        s += v[i];
    return s;
}

int main ()
{
    int vett[] = {10, 11, 12, 13};
    cout << "La somma degli elementi e': ";
    cout << somma(vett) << endl; // 46

    return 0;
}
```

104	112	$v = \&vett[0]$
108	...	
112	10	$vett[0]$
116	11	$vett[1]$
120	12	$vett[2]$
124	13	$vett[3]$

B-204

9.4 Array come argomenti di funzioni (II)

```
// Somma gli elementi di un dato vettore di interi (ii)
// (ERRATO)

#include <iostream>
using namespace std;

int somma(int v[])
{
    int s = 0;
    int n = sizeof v / sizeof(int);
    for (int i = 0; i < n; i++)
        s += v[i];
    return s;
}

int main ()
{
    int vett[] = {10, 11, 12, 13};
    cout << "La somma degli elementi e': ";
    cout << somma(vett) << endl;

    return 0;
}
```

La somma degli elementi e':10

B-205

9.4 Array come argomenti di funzioni (III)

```
// Somma gli elementi di un dato vettore di interi (iii)

#include <iostream>
using namespace std;

int somma(int v[], int n)    // La dimensione va passata
                             // come argomento
{
    int s = 0;
    for (int i = 0; i < n; i++)
        s += v[i];
    return s;
}

int main ()
{
    int vett[] = {10, 11, 12, 13};
    cout << "La somma degli elementi e': ";
    cout << somma(vett, sizeof vett / sizeof(int)) << endl;

    return 0;
}
```

La somma degli elementi e': 46

B-206

9.4 Array come argomenti di funzioni (IV)

```
// Incrementa gli elementi di un dato vettore di interi (i)

#include <iostream>
using namespace std;

void stampa(int v[], int n)
{
    if (n > 0)
    {
        cout << '[' << v[0];
        for (int i = 1; i < n; i++)
            cout << ' ' << v[i];
        cout << ']' << endl;
    }
}

void incrementa(int v[], int n)
{
    for (int i = 0; i < n; i++)
        v[i]++;
}

int main ()
{
    int vett[] = {10, 11, 12, 13};
    stampa(vett, 4);
    incrementa(vett, 4);
    stampa(vett, 4);

    return 0;
}
```

```
[10 11 12 13]
[11 12 13 14]
```

B-207

9.4 Argomenti array costanti (I)

```
// Somma gli elementi di un dato vettore di interi (iv)

#include <iostream>
using namespace std;

int somma(const int v[], int n)
{
    int s = 0;
    for (int i = 0; i < n; i++)
        s += v[i];
    return s;
}

int main ()
{
    int vett[] = {10, 11, 12, 13};
    cout << "La somma degli elementi e': ";
    cout << somma(vett, sizeof vett / sizeof(int)) << endl;

    return 0;
}
```

La somma degli elementi e': 46

B-208

9.4 Argomenti array costanti (II)

```
// Incrementa gli elementi di un dato vettore di interi (ii)

#include <iostream>
using namespace std;

void stampa(const int v[], int n)           // OK!!!
{
    if (n > 0)
    {
        cout << "[" << v[0];
        for (int i = 1; i < n; i++)
            cout << ' ' << v[i];
        cout << "]" << endl;
    }
}

void incrementa(const int v[], int n)      // ERRORE
{
    for (int i = 0; i < n; i++)
        v[i]++;
}

int main ()
{
    int vett[] = {10, 11, 12, 13};
    stampa(vett, 4);
    incrementa(vett, 4);
    stampa(vett, 4);

    return 0;
}
```

B-209

9.5.3 Argomenti array costanti (III)

```
// Trova il massimo valore in un dato vettore di interi

#include <iostream>
using namespace std;

void leggi(int v[], int n)
{
    for (int i = 0; i < n; i++)
    {
        cout << '[' << i << "] = ";
        cin >> v[i];
    }
}

int massimo(const int v[], int n)
{
    int m = v[0];
    for (int i = 1; i < n; i++)
        m = m >= v[i] ? m : v[i];
    return m;
}

int main ()
{
    const int MAX = 10; int v[MAX], nElem;
    cout << "Quanti elementi? ";
    cin >> nElem;
    leggi(v, nElem);
    cout << "Massimo: " << massimo(v, nElem) << endl;

    return 0;
}
```

```
Quanti elementi? 2
[0] = 13
[1] = 45
Massimo: 45
```

B-210

9.5.3 Array multidimensionali (I)

// La dichiarazione di un vettore a piu' dimensioni come
// argomento formale deve specificare la grandezza di
// tutte le dimensioni tranne la prima.
// Se M e' il numero delle dimensioni, l'argomento
// formale e' un puntatore ad array M-1 dimensionali

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

```
const int C = 3;
```

```
void inizializza(int m[][C], int r)  
{  
    for (int i = 0; i < r; i++)  
        for (int j = 0; j < C; j++)  
            m[i][j] = i + j;  
}
```

```
void dim(const int m[][C])
```

```
{  
    cout << "Dimensione (ERRATA) ";  
    cout << sizeof m / sizeof(int) << endl;  
}
```

```
// void riempiErrata(int m[][]);  
                                     ERRORE!
```

B-211

9.5.3 Array multidimensionali (II)

```
void stampa(const int m[][C], int r)  
{  
    for (int i = 0; i < r; i++)  
    {  
        for (int j = 0; j < C; j++)  
            cout << m[i][j] << 't';  
        cout << endl;  
    }  
}  
  
int main ()  
{  
    int mat1[2][C], mat2[2][5];  
    inizializza(mat1, 2);  
    dim(mat1);  
    stampa(mat1, 2);  
    // inizializza(mat2, 2);  
                                     ERRORE: tipo arg. diverso  
    return 0;  
}
```

Dimensione (ERRATA) 1			
0	1	2	
1	2	3	

B-212

9.5.3 Array multidimensionali (III)

```
// Trasmissione mediante puntatori
#include <iostream>
using namespace std;

void inizializza(int* m, int r, int c)
{
    for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
            *(m + i * c + j) = i + j;
}

void stampa(const int* m, int r, int c)
{
    for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < c; j++)
            cout << *(m + i * c + j) << "t";
        cout << endl;
    }
}

int main ()
{
    int mat1[2][3], mat2[2][6];
    // inizializza(mat1, 2, 3); ERRORE passing 'int (*)[3]' as
    //      argument 1 of 'inizializza(int *, int, int)'
    inizializza(&mat1[0][0], 2, 3);
    stampa((int*) mat1, 2, 3);
    inizializza((int*) mat2, 2, 6);
    stampa(&mat2[0][0], 2, 6);

    return 0;
}
```

B-213

9.3 Stringhe (I)

Stringa:

- Sequenza di caratteri.

In C++ non esiste il tipo stringa.

Variabili stringa:

- array di caratteri, che memorizzano stringhe (un carattere per elemento) e il carattere nullo ('\\0') finale.

// Lunghezza ed inizializzazione di stringhe

```
#include <iostream>
#include <cstring>
using namespace std;
int main ()
```

```
{
    char c1[] = "C++";           // inizializzazione
    cout << sizeof c1 << endl;   // 4
    cout << strlen(c1) << endl;  // 3
    char c2[] = {'C', '+', '+'}; // Manca il '\\0!'
    cout << sizeof c2 << endl;   // 3
    char c3[] = {'C', '+', '+', '\\0'}; // OK
    cout << sizeof c3 << endl;   // 4
    char c4[4];
    // c4 = "C++"; ERRORE! Assegnamento tra array

    return 0;
}
```

4	3	3	4
---	---	---	---

B-214

9.3 Stringhe (II)

Operatori di ingresso e di uscita:

- accettano una variabile stringa come argomento.

Operatore di ingresso:

- legge caratteri dallo stream di ingresso (saltando eventuali caratteri bianchi di testa) e li memorizza in sequenza, fino a che incontra un carattere spazio: un tale carattere (che non viene letto) causa il termine dell'operazione e la memorizzazione nella variabile stringa del carattere nullo dopo l'ultimo carattere letto;
- l'array che riceve i caratteri deve essere dimensionato adeguatamente.

Operatore di uscita:

- scrive i caratteri della stringa (escluso il carattere nullo finale) sullo stream di uscita.

```
#include <iostream>
using namespace std;
int main ()
{
    char stringa[12];    // al piu` 11 caratteri oltre a '\0'
    cout << "? ";
    cin >> stringa;      // Esempio: Prima stringa
                        // Attenzione: nessun controllo
                        // sulla dimensione
    cout << stringa << endl; // Esempio: Prima
    return 0;
}
```

? Prima stringa
Prima

B-215

9.3 Stringhe (III)

```
// Stringhe e puntatori
#include <iostream>
using namespace std;
int main ()
{
    char s1[] = "Universita' ";
    char s2[] = {'d','i',' ','\0'};
    char *s3 = "Pisa";
    char *s4 = "Toscana";

    cout << s1 << s2 << s3 << s4 << endl;
    // puntatori a caratteri interpretati come stringhe
    s4 = s3;

    cout << s3 << endl;    // Pisa
    cout << s4 << endl;    // Pisa

    char *const s5 = "oggi";
    char *const s6 = "domani";

    // s6 = s5;             ERRORE!

    cout << (void *)s3 << endl; // Per stampare il puntatore
    return 0;
}
```

Universita' di Pisa Toscana
Pisa
Pisa
0x40121d

B-216

9.3 Stringhe (IV)

// Conta le occorrenze di ciascuna lettera in una stringa

```
#include <iostream>
using namespace std;
int main ()
{
    const int LETTERE = 26;
    char str[100];
    int conta[LETTERE];
    for (int i = 0; i < LETTERE; i++)
        conta[i] = 0;
    cout << "Inserisci una stringa: ";
    cin >> str;

    for (int i = 0; str[i] != '\0'; i++)
        if (str[i] >= 'a' && str[i] <= 'z')
            ++conta[str[i] - 'a'];
        else if (str[i] >= 'A' && str[i] <= 'Z')
            ++conta[str[i] - 'A'];

    for (int i = 0; i < LETTERE; i++)
        cout << char('a' + i) << ": " << conta[i] << '\t';
    cout << endl;

    return 0;
}
```

Inserisci una stringa: prova
a: 1 b: 0 c: 0 d: 0 e: 0 f: 0 g: 0 h: 0 i: 0 j: 0
k: 0 l: 0 m: 0 n: 0 o: 1 p: 1 q: 0 r: 1 s: 0 t: 0
u: 0 v: 1 w: 0 x: 0 y: 0 z: 0

B-217

9.6 Funzioni di libreria sulle stringhe (I)

Dichiarazioni contenute nel file <cstring>

`char *strcpy(char *dest, const char *sorg);`

Copia *sorg* in *dest*, incluso il carattere nullo (terminatore di stringa), e restituisce *dest*;

ATTENZIONE: non viene effettuato nessun controllo per verificare se la dimensione di *dest* è sufficiente per contenere *sorg*.

`char *strcat(char *dest, const char *sorg);`

Concatena *sorg* al termine di *dest* e restituisce *dest* (il carattere nullo compare solo alla fine della stringa risultante);

ATTENZIONE: non viene effettuato nessun controllo per verificare se la dimensione di *dest* è sufficiente per contenere la concatenazione di *sorg* e *dest*.

ATTENZIONE: sia *sorg* che *dest* devono essere delle stringhe.

B-218

9.6 Funzioni di libreria sulle stringhe (II)

int strlen(const char *string);

Restituisce la lunghezza di *string*; il valore restituito è inferiore di 1 al numero di caratteri effettivi, perché il carattere nullo che termina *string* non viene contato.

int strcmp(const char *s1, const char *s2);

Confronta s1 con s2:

- restituisce un valore negativo se s1 è alfabeticamente minore di s2;
- un valore nullo se le due stringhe sono uguali,
- un valore positivo se s1 è alfabeticamente maggiore di s2; (la funzione *distingue* tra *maiuscole* e *minuscole*).

char *strchr(const char *string, char c);

Restituisce il puntatore alla prima occorrenza di c in *string* oppure 0 se c non si trova in *string*.

9.6 Funzioni di libreria sulle stringhe (III)

// ESEMPIO

```
#include <iostream>
#include <cstring>
using namespace std;
int main ()
{
    const int N = 30;
    char s1[] = "Corso ";
    char s2[] = "di ";
    char s3[] = "Informatica\n";
    char s4[N] = "Corso ";

    cout << "Dimensione degli array s1 e s4 " << endl;
    cout << sizeof s1 << " " << sizeof s4 << endl;

    cout << "Dimensione delle stringhe s1 e s4 " << endl;
    cout << strlen(s1) << " " << strlen(s4) << endl;
    if (!strcmp(s1,s4)) cout << "Stringhe uguali " << endl;
    else cout << "Stringhe diverse " << endl;

    if (!strcmp(s1,s2)) cout << "Stringhe uguali " << endl;
    else cout << "Stringhe diverse " << endl << endl;

    char s5[N];
    strcpy(s5,s1);
    strcat(s5,s2);
    strcat(s5,s3);
```

9.6 Funzioni di libreria sulle stringhe (IV)

```
cout << "Concatenazione di s1, s2 e s3 " << endl;  
cout << s5 << endl;  
char *s=strchr(s5,'I');  
cout << "Stringa dalla prima istanza di I " << endl;  
cout << s << endl;  
  
return 0;  
}
```

Dimensione degli array s1 e s4

7 30

Dimensione delle stringhe s1 e s4

6 6

Stringhe uguali

Stringhe diverse

Concatenazione di s1, s2 e s3

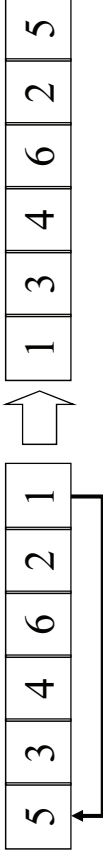
Corso di Informatica

Stringa dalla prima istanza di I
Informatica

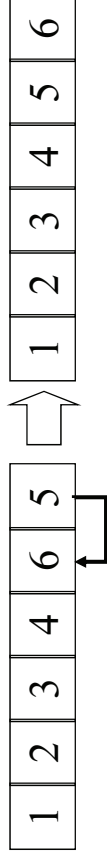
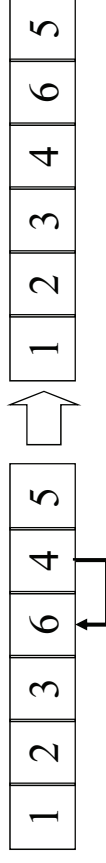
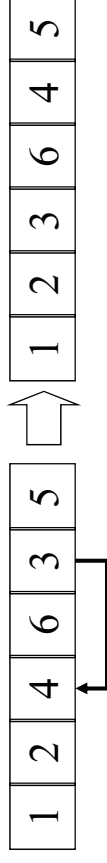
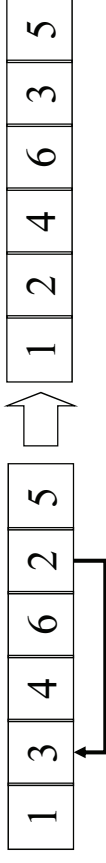
9.7 Ordinamento dei vettori

Ordinamento per selezione (selection-sort)

- Si cerca l'elemento più piccolo e si scambia con l'elemento in posizione $i = 0$



- Si cerca l'elemento più piccolo tra i rimanenti $N-i$ e si scambia con l'elemento in posizione i , per $i = 1..N-1$



9.7 Ordinamento dei vettori (selection-sort)

Ordinamento per selezione (selection-sort)

```
#include <iostream>
using namespace std;
typedef int T;    // introduce un nuovo identificatore
                 // per individuare un tipo pre-esistente
                 // (ossia crea un sinonimo per un tipo)

void stampa(const T v[], int n)
{
    if (n != 0)
    {
        cout << "[" << v[0];
        for (int i = 1; i < n; i++)
            cout << " " << v[i];
        cout << "]" << endl;
    }
}

void scambia(T vettore[], int x, int y)
{
    T lavoro = vettore[x];
    vettore[x] = vettore[y];
    vettore[y] = lavoro;
}
```

B-223

9.7 Ordinamento dei vettori (selection-sort)

Ordinamento per selezione (selection-sort)

```
void selectionSort(T vettore[], int n)
{
    int min;
    for (int i = 0 ; i < n-1; i++)
    {
        min = i;
        for (int j = i + 1; j < n; j++)
            if (vettore[j] < vettore[min]) min = j;
        scambia(vettore,i,min);
    }
}

int main()
{
    T v[] = {2, 26, 8, 2, 23};
    selectionSort(v, 5);
    stampa(v, 5);

    return 0;
}
```

[2 2 8 23 26]

- Complessità dell'algoritmo dell'ordine di n^2 , dove n è il numero di elementi nel vettore.

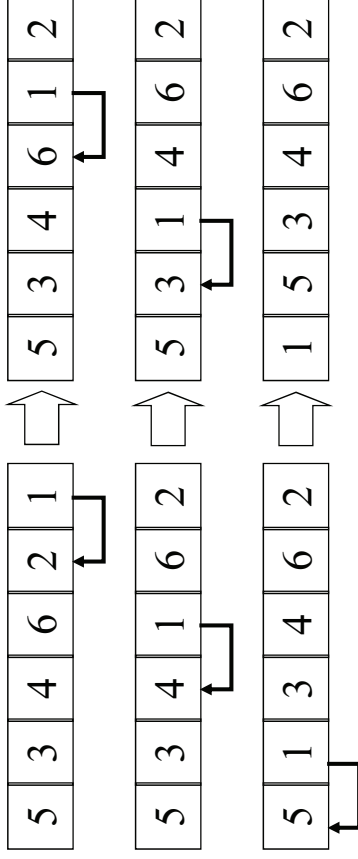
B-224

9.7 Ordinamento dei vettori (bubble-sort)

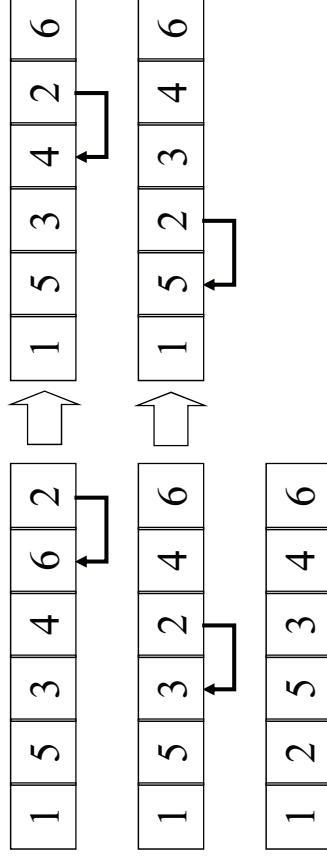
Ordinamento bubble-sort

- Si scorre l'array $n-1$ volte, dove n è il numero di elementi nell'array, da destra a sinistra, scambiando due elementi contigui se non sono nell'ordine giusto.

• Prima passata



• Seconda passata

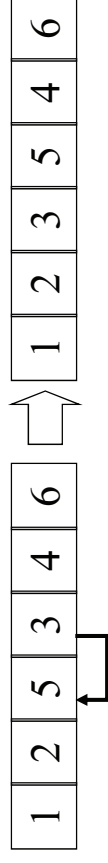


N.B.: i primi due elementi risultano ordinati

9.7 Ordinamento dei vettori (bubble-sort)

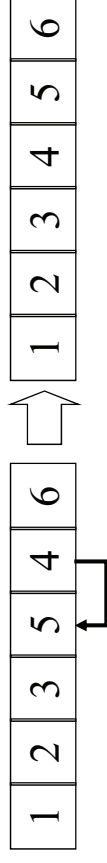
Ordinamento bubble-sort

• Terza passata



N.B.: i primi tre elementi risultano ordinati

• Quarta passata



N.B.: i primi quattro elementi risultano ordinati

- Quinta passata
 - Nessun cambiamento

```
void bubble(T vettore[], int n)
{
    for (int i = 0 ; i < n-1; i++)
        for (int j = n-1; j > i; j--)
            if (vettore[j] < vettore[j-1])
                scambia(vettore, j, j-1);
}
```

- Complessità dell'algoritmo dell'ordine di n^2 , dove n è il numero di elementi nel vettore.

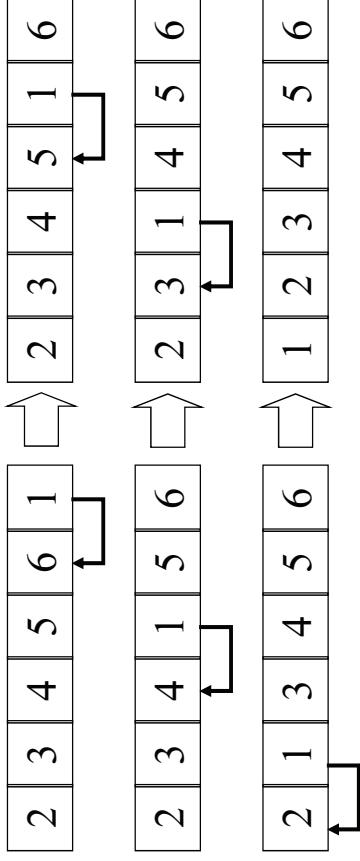
9.7 Ordinamento dei vettori (bubble-sort)

Ordinamento bubble-sort ottimizzato

Supponiamo che il vettore sia questo:

2	3	4	5	6	1
---	---	---	---	---	---

- Prima passata



- Il vettore dopo la prima passata risulta ordinato.
- Me ne accorgo dal fatto che la seconda passata non effettua nessuno scambio.
- Inutile eseguire le passate successive alla seconda.

B-227

9.7 Ordinamento dei vettori (bubble-sort)

Ordinamento bubble-sort ottimizzato

```
void bubble(T vettore[], int n)
{
    bool ordinato = false;
    for (int i = 0 ; i < n-1 && !ordinato; i++)
    {
        ordinato = true;
        for (int j = n-1; j >= i+1; j--)
            if(vettore[j] < vettore[j-1])
            {
                scambia(vettore, j, j-1);
                ordinato = false;
            }
    }
}

int main()
{
    T v[] = {2, 1, 3, 4, 5};
    bubble(v, 5);
    stampa(v, 5);

    return 0;
}
```

[1 2 3 4 5]

- L'algoritmo ottimizzato esegue due sole passate invece delle quattro dell'algoritmo non ottimizzato.

B-228

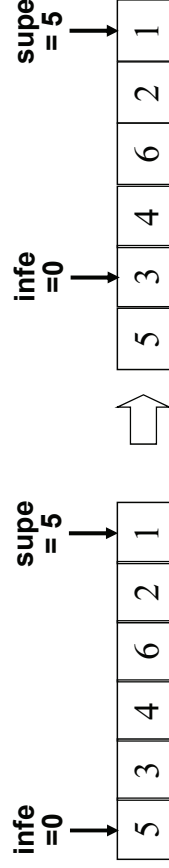
9.7 Ricerca lineare (I)

Problema

- cercare un elemento in un array tra l'elemento in posizione *infe* e quello in posizione *supe*.

Possibile soluzione

- Scorrere il vettore in sequenza a partire dall'elemento in posizione *infe* fino all'elemento cercato oppure all'elemento in posizione *supe*.



```
#include <iostream>
using namespace std;
typedef int T;
```

```
bool ricerca(T vett[], int infe, int supe, T k, int &pos)
```

```
{
    bool trovato = false;
    while (!trovato) && (infe <= supe))
    {
        if (vett[infe] == k)
        {
            pos = infe;
            trovato = true;
        }
        infe++;
    }
    return trovato;
}
```

B-229

9.7 Ricerca lineare (II)

```
int main()
{
    T v[] = {1, 3, 4, 5, 7, 9};
    int i;
    // individua la posizione
    if (!ricerca(v, 0, 5, 9, i))
        cerr << "Elemento cercato non presente " << endl;
    else
        cout << "Posizione elemento cercato: " << i << endl;
    if (!ricerca(v, 0, 4, 8, i))
        cerr << "Elemento cercato non presente " << endl;
    else
        cout << "Posizione elemento cercato: " << i << endl;
    return 0;
}
```

Posizione elemento cercato: 5
Elemento cercato non presente

N.B: Per la ricerca della presenza dell'elemento 9 è necessario esaminare uno ad uno tutti gli elementi.

B-230

9.7 Ricerca binaria (I)

PREREQUISITO: Vettori ordinati!!!!

Ricerca binaria (vettori ordinati in ordine crescente)

Idea:

- Si confronta l'elemento cercato con l'elemento in posizione centrale; se sono uguali la ricerca termina;
- altrimenti:
 - se l'elemento cercato è minore dell'elemento in posizione centrale la ricerca prosegue nella prima metà del vettore; altrimenti prosegue nella seconda metà del vettore.

```
bool ricbin(T ordVett[], int infe, int supe, T k, int &pos)
```

```
{
    while (infe <= supe)
    {
        int medio = (infe + supe) / 2; // calcola l'indice centrale
        if (k > ordVett[medio])
            infe = medio + 1; // ricerca nella metà superiore
        else
            if (k < ordVett[medio])
                supe = medio - 1; // ricerca nella metà superiore
            else
            {
                pos = medio; // trovato
                return true;
            }
        return false;
    }
}
```

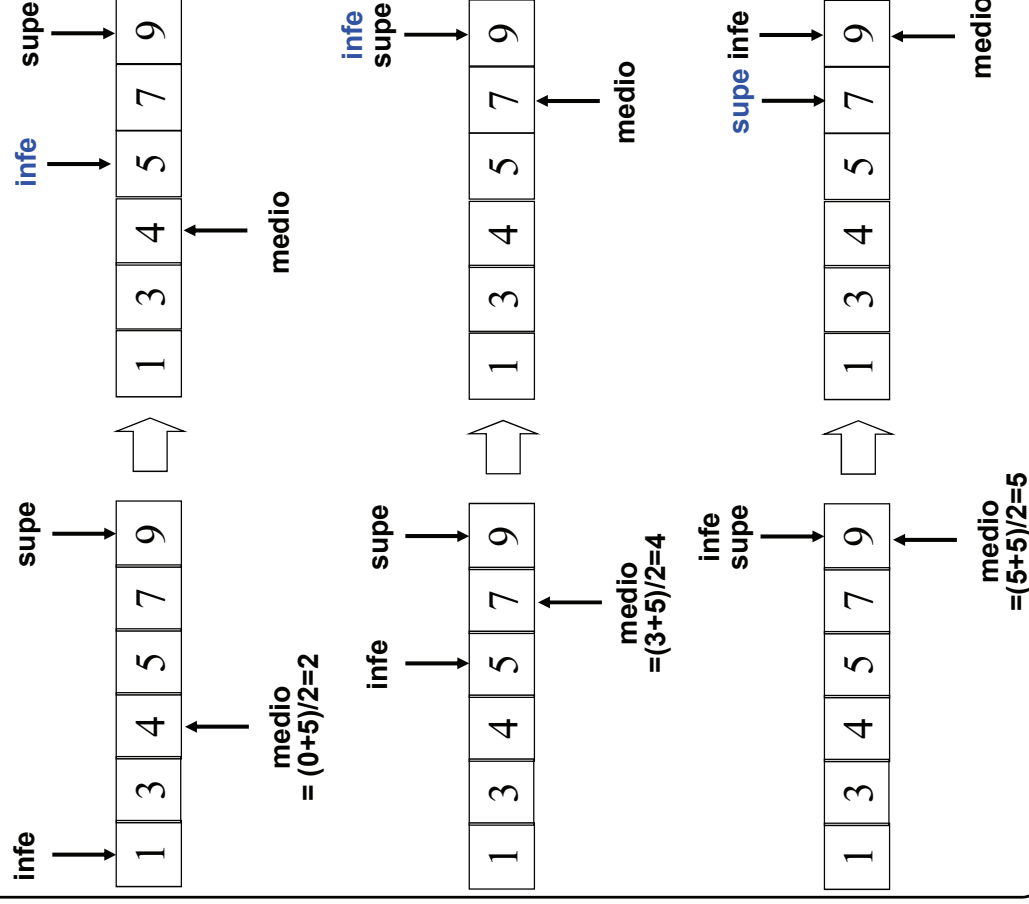
N.B: Per la ricerca dell'elemento 9, esempio precedente, è sufficiente esaminare 3 elementi solamente.

B-231

9.7 Ricerca binaria (II)

Esempio di esecuzione della ricerca binaria.

Supponiamo di cercare l'elemento 2 ($k=8$).



B-232

9.7 Ricerca binaria (III)

Esempio: ricerca in un vettore di caratteri

```
#include <iostream>
using namespace std;
typedef char T;

bool ricbin(T ordVett[], int infe, int supe, T k, int &pos)
{
    while (infe <= supe)
    {
        int medio = (infe + supe) / 2; //calcola l'indice centrale
        if (k > ordVett[medio])
            infe = medio + 1; // ricerca nella meta' superiore
        else if (k < ordVett[medio])
            supe = medio - 1; // ricerca nella meta' superiore
        else
        {
            pos = medio; // trovato
            return true;
        }
    }
    return false;
}

int main()
{
    T v[] = {'a', 'b', 'c', 'r', 's', 't'}; // individua la posizione
    int i;
    if (!ricbin(v, 0, 5, 'c', i))
        cerr << "Elemento cercato non presente " << endl;
    else
        cout << "Posizione elemento cercato " << i << endl;
    // 2
    return 0;
}
```

B-233

9.7 Esempio (I)

Esempio: ordinamento e ricerca in un vettore di stringhe

```
#include <iostream>
using namespace std;
const int C=20;
typedef char T[C];

void scambia(T vettore[], int x, int y)
{
    T lavoro;
    strcpy(lavoro,vettore[x]);
    strcpy(vettore[x],vettore[y]);
    strcpy(vettore[y],lavoro);
}

void stampa(const T vettore[], int n)
{
    if (n != 0)
    {
        cout << '[' << vettore[0];
        for (int i = 1; i < n; i++)
            cout << ',' << vettore[i];
        cout << ']' << endl;
    }
}
```

B-234

9.7 Esempio (II)

```

void bubble(T vettore[], int n)
{
    bool ordinato = false;
    for (int i = 0 ; i < n-1 && !ordinato; i++)
    {
        ordinato = true;
        for (int j = n-1; j >= i+1; j--)
            if(strcmp(vettore[j],vettore[j-1])<0)
            {
                scambia(vettore, j, j-1);
                ordinato = false;
            }
    }
}

bool ricbin(T ordVett[], int infe, int supe, T k, int &pos)
{
    while (infe <= supe)
    {
        int medio = (infe + supe) / 2;
        if (strcmp(k,ordVett[medio])>0)
            infe = medio + 1;
        else
            if (strcmp(k,ordVett[medio])<0)
                supe = medio - 1;
            else
            {
                pos = medio;
                return true;
            }
    }
    return false;
}

```

B-235

9.7 Esempio (III)

```

int main ()
{
    T s[4];
    for (int i=0; i<4; i++)
    {
        cout << '?' << endl;
        cin >> s[i];
    }
    stampa(s,4);
    bubble(s,4);
    stampa(s,4);
    int i; T m;
    cout << "Ricerca ?" << endl;
    cin >> m;
    if (ricbin(s,0,4,m,i))
        cout << "Trovato in posizione " << i << endl;
    else cout << "Non trovato" << endl;
    cout << "Ricerca ?" << endl;
    cin >> m;
    if (ricbin(s,0,4,m,i))
        cout << "Trovato in posizione " << i << endl;
    else cout << "Non trovato" << endl;

    return 0;
}

```

```

? mucca
?
anatra
?
zebra
?
cavallo
[mucca anatra zebra cavallo]
[anatra cavallo mucca zebra]
Ricerca ?
cavallo
Trovato in posizione 1
Ricerca ?
bue
Non trovato

```

B-236

10.1 Strutture (I)

Struttura:

- n-upla ordinata di elementi, detti membri (o campi), ciascuno dei quali ha uno specifico tipo ed uno specifico nome, e contiene una data informazione;
- rappresenta una collezione di informazioni su un dato oggetto.

```
basic-structure-type-declaration  
structure-type-specifier ;  
structure-type-specifier  
struct identifieropt  
    { structure-member-section-seq }
```

Sezione:

- membri di un certo tipo, ciascuno destinato a contenere un dato (campi dati);
- forma sintatticamente equivalente alla definizione di oggetti non costanti e non inizializzati.

Esempio:

```
struct persona  
{  
    char nome[20];  
    char cognome[20];  
    int g_nascita, m_nascita, a_nascita;  
};
```

B-237

10.1 Strutture (II)

```
// Punto  
#include <iostream>  
using namespace std;  
struct punto  
{  
    double x;  
    double y;  
};  
int main ()  
{  
    punto r, s;  
    r.x = 3;  
    r.y = 10.5;  
    s.x = r.x;  
    s.y = r.y + 10.0;  
    cout << 'r' << r.x << " " << r.y << ">\n";  
    cout << 's' << s.x << " " << s.y << ">\n";  
    punto *p = &r;  
    cout << 'p' << p->x << " " << (*p).x  
    cout << p->y << ">\n";  
    cout << (*p).y  
    punto t = {1.0, 2.0};  
    cout << 't' << t.x << " " << t.y << ">\n";  
}
```

```
<3, 10.5>  
<3, 20.5>  
<3, 10.5>  
<1, 2>
```

// selettore di membro

B-238

10.1 Strutture (III)

```
struct punto
{
    /* ... */
    punto s;
};

//~~~~~
// Struttura contenente un riferimento a se stessa
//~~~~~

struct punto {
    /* ... */
    punto* p;
};

//~~~~~
// Strutture con riferimenti intrecciati.
// Dichiarazioni incomplete
//~~~~~

struct Parte;

struct Componente {
    /* ... */
    Parte* p;
};

struct Parte {
    /* ... */
    Componente* c;
};
```

B-239

10.1 Strutture (IV)

```
// Array di strutture

#include <iostream>
using namespace std;

struct punto
{
    double x; double y;};

const int MAX = 30;

int main ()
{
    struct poligono
    {
        int quanti;
        punto p[MAX];
    } p;

    p.quanti = 3;
    p.p[0].x = 3.0;
    p.p[0].y = 1.0;
    p.p[1].x = 4.0;
    p.p[1].y = 10.0;
    p.p[2].x = 3.0;
    p.p[2].y = 100.0;

    cout << "Stampa del poligono " << endl;
    for (int i = 0; i < p.quanti; i++)
        cout << ' ' << p.p[i].x << " " << p.p[i].y << ">\n";
}
```

```
Stampa del poligono
<3, 1>
<4, 10>
<3, 100>
```

B-240

10.1.1 Operazioni sulle strutture (I)

```
// Assegnamento tra strutture

#include <iostream>
using namespace std;

struct punto
{
    double x;
    double y;
};

int main ()
{
    punto r1 = {3.0, 10.0}; // inizializzazione
    punto r2;

    r2 = r1; // copia membro a membro

    cout << "r1 = <" << r1.x << ", " << r1.y << ">\n";
    // <3, 10>
    cout << "r2 = <" << r2.x << ", " << r2.y << ">\n";
    // <3, 10>

    // if (r2 != r1) ERRORE

}
```

```
r1 = <3, 10>
r2 = <3, 10>
```

N.B.: Non sono definite operazioni di confronto sulle strutture.

B-241

10.1.1 Operazioni sulle strutture (II)

```
// Strutture come argomenti di funzioni e restituite da
// funzioni

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

struct punto
{
    double x; double y; };

void test(punto p)
{
    cout << "Dimensione argomento " << sizeof p << endl;
}

void test(const punto* p) // Overloading
{
    cout << "Dimensione argomento " << sizeof p << endl;
}

double dist(const punto* p1, const punto* p2)
{
    return sqrt((p1->x - p2->x) * (p1->x - p2->x) +
                (p1->y - p2->y) * (p1->y - p2->y));
}

punto vicino(punto ins[], int n, const punto* p)
{
    double min = dist(&ins[0], p), t;
    int index = 0;
    for (int i = 1; i < n; i++)
    {
        t = dist(&ins[i], p);
        if (min > t)
        {
            index = i; min = t;
        }
    }
    return ins[index];
}
```

120	3.5	p.x
124		
128	10.0	p.y
132		

B-242

10.1.1.1 Operazioni sulle strutture (III)

```
// Strutture come argomenti di funzioni e restituite da
// funzioni

int main ()
{
    punto ins[] = {{3.0, 10.0}, {2.0, 9.0}, {1.0, 1.0}};
    punto r = {3.5, 10.0};
    test(r);
    test(&r);
    cout << "Distanza: " << dist(&r, &ins[0]) << endl;
    punto s = vicino(ins, sizeof ins/sizeof(punto), &r);
    cout << "Punto piu' vicino: ";
    cout << 'c' << s.x << ", " << s.y << ">\n";

    return 0;
}
```

Dimensione argomento 16
Dimensione argomento 4
Distanza: 0.5
Punto piu' vicino: <3, 10>

B-243

10.1.1.1 Operazioni sulle strutture (IV)

```
// Copia di vettori

#include <iostream>
using namespace std;
const int N = 3;
struct vettore
{
    int vv[N];
};

void stampa(const vettore& v, int n)
{
    cout << '[' << v.vv[0];
    for (int i = 1; i < n; i++)
        cout << ' ' << v.vv[i];
    cout << ']' << endl;
}

int main ()
{
    vettore v1 = {1, 2, 3}, v2;
    v2 = v1;
    cout << "Stampa del vettore v1 " << endl;
    stampa(v1, N);
    cout << "Stampa del vettore v2 " << endl;
    stampa(v2, N);

    return 0;
}
```

Stampa del vettore v1
[1 2 3]
Stampa del vettore v2
[1 2 3]

B-244

10.1.1.1 Operazioni sulle strutture (V)

```
// Trasmissione di vettori per valore

#include <iostream>
using namespace std;
const int N = 3;
struct vettore
{   int vv[N];   };

void stampa(const vettore& v, int n)
{
    cout << '[' << v.vv[0];
    for (int i = 1; i < n; i++)
        cout << ' ' << v.vv[i];
    cout << ']' << endl;
}

void incrementa(vettore v, int n)
{   for (int i = 0; i < n; i++) v.vv[i]++;}

int main ()
{
    vettore v1 = {1, 2, 3};
    cout << "Stampa del vettore v1 " << endl;
    stampa(v1, N);
    incrementa(v1, N);           // Incrementa la copia
    cout << "Stampa del vettore v1 " << endl;
    stampa(v1, N);              // [1 2 3]

    return 0;
}
```

```
Stampa del vettore v1
[1 2 3]
Stampa del vettore v2
[1 2 3]
```

B-245

10.2 Unioni (I)

Sono dichiarate e usate con la stessa sintassi delle strutture:

- si utilizza la parola chiave *union* al posto di *struct*.

Rappresentano un'area di memoria che in tempi diversi può contenere dati di tipo differente:

- i membri di un'unione corrispondono a diverse "interpretazioni" di un'unica area di memoria.

Membri di una unione non della stessa dimensione:

- viene riservato spazio per il più grande.

Esempio:

```
struct { int i; double d; } x;
union { int i; double d; } y;
```

- la struttura *x* occupa 96 bit di memoria (32 per *i* e 64 per *d*);
- l'unione *y* occupa 64 bit di memoria, che possono essere dedicati ad un valore intero (lasciandone 32 inutilizzati) o ad un valore reale.

Operazioni:

- quelle viste per le strutture.

Valori iniziali delle unioni:

- solo per il primo membro;
- esempio:
union {char c; int i; double f; } a = { 'X' };

B-246

10.2 Unioni (II)

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

union Uni
{ char c; int i; };

struct Str
{ char c; int i; };

int main()
{
    cout << sizeof(char) << '\t' << sizeof(int) << endl;
    // 1 4
    cout << sizeof(Uni) << '\t' << sizeof(Str) << endl;
    // 4 8

    Uni u = {'a'};
    // Uni u1 = {'a', 10000};          ERRATO

    u.i = 0xFFFF7A;                  // 7A e' la codifica ASCII di z
    cout << u.c << '\t' << u.i << endl;
    // z 65402

    Str s = {'a', 0xFFFF7A};
    cout << s.c << '\t' << s.i << endl;
    // a 65402

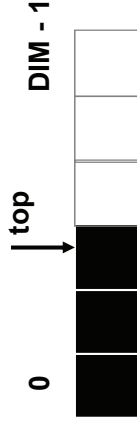
    return 0;
}
```

1	4
4	8
z	65402
a	65402

B-247

10.3.1 Pila (I)

- Insieme ordinato di dati di tipo uguale, in cui è possibile effettuare operazioni di inserimento e di estrazione secondo la seguente regola di accesso: l'ultimo dato inserito è il primo ad essere estratto (LIFO: Last In First Out).



- Se `top == -1`, la pila è vuota. Se `top == DIM - 1`, dove `DIM` è il numero massimo di elementi nella pila, la pila è piena.

```
#include <iostream>
using namespace std;
typedef int T;
const int DIM = 5;

struct pila
{
    int top;
    T stack[DIM];
};

//inizializzazione della pila
void inip(pila& pp)
{
    pp.top = -1;
}
```

B-248

10.3.1 Pila (II)

```
bool empty(const pila& pp)    // pila vuota?
{
    if (pp.top == -1) return true;
    return false;
}

bool full(const pila& pp)    // pila piena?
{
    if (pp.top == DIM - 1) return true;
    return false;
}

bool push(pila& pp, T s)    // inserisce un elemento in pila
{
    if (full(pp)) return false;
    pp.stack[++(pp.top)] = s;
    return true;
}

bool pop(pila& pp, T& s)    // estrae un elemento dalla pila
{
    if (empty(pp)) return false;
    s = pp.stack[(pp.top)--];
    return true;
}

void stampa(const pila& pp)    // stampa gli elementi
{
    cout << "Elementi contenuti nella pila: " << endl;
    for (int i = pp.top; i >= 0; i--)
        cout << '[' << i << "]" << pp.stack[i] << endl;
}
```

B-249

10.3.1 Pila (III)

```
int main()
{
    pila st;
    inip(st);
    T num;
    if (empty(st)) cout << "Pila vuota" << endl;
    for (int i = 0; i < DIM; i++)
        if (push(st, DIM - i))
            cout << "Inserito " << DIM - i <<
                ". Valore di top: " << st.top << endl;
        else cerr << "Inserimento di " << i << " fallito" << endl;
    if (full(st)) cout << "Pila piena" << endl;
    stampa(st);
    for (int i = 0; i < DIM - 2; i++)
        if (pop(st, num))
            cout << "Estratto " << num << ". Valore di top: "
                << st.top << endl;
        else cerr << "Estrazione fallita" << endl;
    for (int i = 0; i < DIM; i++)
        if (push(st, i))
            cout << "Inserito " << i << ". Valore di top: "
                << st.top << endl;
        else cerr << "Inserimento di " << i << " fallito" << endl;
    stampa(st);
    for (int i = 0; i < 2; i++)
        if (pop(st, num))
            cout << "Estratto " << num << ". Valore di top: "
                << st.top << endl;
        else cerr << "Estrazione fallita" << endl;
    stampa(st);
    return 0;
}
```

B-250

10.3.1 Pila (IV)

Pila vuota

Inserito 5. Valore di top: 0
Inserito 4. Valore di top: 1
Inserito 3. Valore di top: 2
Inserito 2. Valore di top: 3
Inserito 1. Valore di top: 4

Pila piena

Elementi contenuti nella pila:

[4] 1
[3] 2
[2] 3
[1] 4
[0] 5

Estratto 1. Valore di top: 3
Estratto 2. Valore di top: 2
Estratto 3. Valore di top: 1
Inserito 0. Valore di top: 2
Inserito 1. Valore di top: 3
Inserito 2. Valore di top: 4
Inserimento di 3 fallito

Inserimento di 4 fallito

Elementi contenuti nella pila:

[4] 2
[3] 1
[2] 0
[1] 4
[0] 5

Estratto 2. Valore di top: 3
Estratto 1. Valore di top: 2

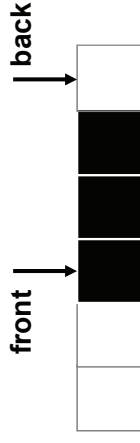
Elementi contenuti nella pila:

[2] 0
[1] 4
[0] 5

B-251

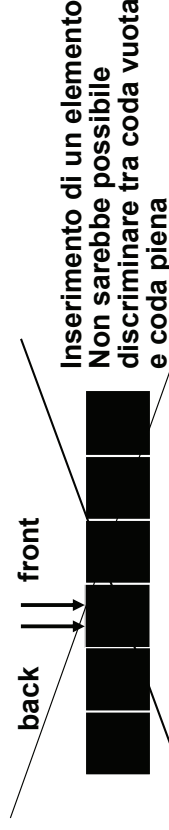
10.3.2 Coda (I)

- Insieme ordinato di dati di tipo uguale, in cui è possibile effettuare operazioni di inserimento e di estrazione secondo la seguente regola di accesso: il primo dato inserito è il primo ad essere estratto (FIFO: First In First Out).



- Realizzata con un array circolare e due puntatori:

- front – posizione da cui avviene l'estrazione;
- back – posizione in cui avviene l'inserimento.



- front == back coda vuota
- front == (back + 1) % DIM coda piena
- (ATTENZIONE al massimo DIM - 1 elementi)

B-252

10.3.2 Coda (II)

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

typedef int T;
const int DIM = 5;

struct coda
{
    int front, back;
    T queue[DIM];
};

void inic(coda& cc) // inizializzazione della coda
{
    cc.front = cc.back = 0;
}

bool empty(const coda& cc) // coda vuota?
{
    if (cc.front == cc.back) return true;
    return false;
}

bool full(const coda& cc) // coda piena?
{
    if (cc.front == (cc.back + 1)%DIM) return true;
    return false;
}
```

B-253

10.3.2 Coda (III)

```
bool insqueue(coda& cc, T s) // inserisce un elemento
{
    if (full(cc)) return false;
    cc.queue[cc.back] = s;
    cc.back = (cc.back+1)%DIM;
    return true;
}

bool esqueue(coda& cc, T& s) // estrae un elemento
{
    if (empty(cc)) return false;
    s = cc.queue[cc.front];
    cc.front = (cc.front + 1)%DIM;
    return true;
}

void stampa(const coda& cc) // stampa gli elementi
{
    for (int i = cc.front; i%DIM != cc.back; i++)
        cout << cc.queue[i%DIM] << endl;
}
```

B-254

10.3.2 Coda (IV)

```
int main()
{
    coda qu; T num;
    inic(qu);
    if (empty(qu)) cout << "Coda vuota" << endl;
    for (int i = 0; i < DIM; i++)
        if (insqueue(qu, i))
            cout << "Inserito l'elemento " << i << " in
                posizione " << (qu.back + DIM - 1)%DIM << endl;
            else cerr << "Coda piena" << endl;
    if (full(qu)) cout << "Coda piena" << endl;
    stampa(qu);
    for (int i = 0; i < DIM - 2; i++)
        if (esqueue(qu, num))
            cout << "Estratto l'elemento " << num << " in
                posizione " << (qu.front + DIM - 1)%DIM << endl;
            else cout << "Coda vuota" << endl;
    for (int i = 0; i < DIM; i++)
        if (insqueue(qu, i))
            cout << "Inserito l'elemento " << i << " in posizione "
                << (qu.back + DIM - 1)%DIM << endl;
            else cerr << "Coda piena" << endl;
    stampa(qu);
    for (int i = 0; i < 2; i++)
        if (esqueue(qu, num))
            cout << "Estratto l'elemento " << num << " in
                posizione " << (qu.front + DIM - 1)%DIM << endl;
            else cout << "Coda vuota" << endl;
    stampa(qu);
    return 0;
}
```

B-255

10.3.2 Coda (V)

```
Coda vuota
Inserito l'elemento 0 in posizione 0
Inserito l'elemento 1 in posizione 1
Inserito l'elemento 2 in posizione 2
Inserito l'elemento 3 in posizione 3
Coda piena
Coda piena
0
1
2
3
Estratto l'elemento 0 in posizione 0
Estratto l'elemento 1 in posizione 1
Estratto l'elemento 2 in posizione 2
Inserito l'elemento 0 in posizione 4
Inserito l'elemento 1 in posizione 0
Inserito l'elemento 2 in posizione 1
Coda piena
Coda piena
3
0
1
2
Estratto l'elemento 3 in posizione 3
Estratto l'elemento 0 in posizione 4
1
2
```

B-256

11.1 Tipi funzione

Dichiarazione di una funzione di n argomenti:

- associa ad un identificatore un tipo, determinato dalla n-upla ordinata dei tipi degli argomenti e dal tipo del risultato.

Valori associati ai tipi funzione:

- tutte le funzioni corrispondenti (non valgono le conversioni implicite).

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

```
int quadrato(int n)           // Istanza di tipo funzione int(int)
{ return n*n; }
```

```
int cubo(int n)              // Istanza di tipo funzione int(int)
{ return n*n*n; }
```

```
double media(int fp(int), int a, int b)    // funzione arg.
```

```
{ int s = 0;
  for (int n = a; n <= b; n++)
    s += fp(n);
  return static_cast<double>(s) / (b-a+1);
}
```

```
int main()
{
  cout << media(quadrato, 1, 2) << endl;
  cout << media(cubo, 1, 2) << endl;

  return 0;
}
```

2.5
4.5

B-257

11.2 Puntatori a funzione (I)

I puntatori possono contenere anche indirizzi di funzione.

Definizione di un puntatore a funzione:

result-type (* identifier) (argument-portion|opt) ;

Chiamata di funzione attraverso il puntatore:

identifier (expression-list|opt)
(* identifier) (expression-list|opt)

```
#include <iostream>
```

```
using namespace std;
```

```
int quadrato(int n) { return n*n; }
```

```
int cubo(int n) { return n*n*n; }
```

```
double media(int (*pf)(int), int a, int b)
```

```
{int s = 0;
```

```
  for (int n = a; n <= b; n++)
```

```
    s += (*pf)(n);           // forma alternativa s += pf(n)
```

```
  return static_cast<double>(s) / (b-a+1);
```

```
}
```

```
int main()
```

```
{ cout << media(quadrato, 1, 2) << endl;
```

```
  cout << media(cubo, 1, 2) << endl;
```

```
  return 0;
```

```
}
```

B-258

11.2 Puntatori a funzione (II)

```
#include <iostream>
using namespace std;
void f1()
{   cout << "uno" << endl; }
void f2()
{   cout << "due" << endl; }

void f3(void (*&pf)(void))
{   int a;
    cin >> a;
    if (a==0) pf = f1; else pf = f2;
}

int main()
{
    void(*p1)(void);
    f3(p1);
    (*p1)();
    f3(p1);
    (*p1)();

    return 0;
}
```

```
5
due
0
uno
```

B-259

11.3 Argomenti default (I)

Argomenti formali di una funzione:

- possono avere inizializzatori;
- costituiscono il valore default degli argomenti attuali (vengono inseriti dal compilatore nelle chiamate della funzione in cui gli argomenti attuali sono omessi);
- se il valore default viene indicato solo per alcuni argomenti, questi devono essere gli ultimi;
- gli inizializzatori non possono contenere né variabili locali né argomenti formali della funzione.

Chiamata di funzione:

- possono essere omessi tutti o solo alcuni argomenti default: in ogni caso gli argomenti omessi devono essere gli ultimi.

Esempio (peso di un cilindro):

```
double peso(double lung, double diam = 10, double
dens = 15)
{
    diam /= 2;
    return (diam*diam * 3.14 * lung * dens);
}

int main()
{ // ...
    p = peso(125); // equivale a peso(125, 10, 15)
    p = peso(35, 5); // equivale a peso(35, 5, 15)
    // ...
}
```

B-260

11.3 Argomenti default (II)

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

// prima dichiarazione di perimetro
double perimetro(int nLati, double lungHato = 1.0);

void f() {
    // double p = perimetro();
    // cout << p << endl;
}

// con questa dichiarazione alternativa, tutto funziona
double perimetro(int nLati = 3, double lungHato = 1.0);

void g(){
    double p = perimetro();
    cout << p << endl;
}

int main (){
    f();
    g();

    return 0;
}

double perimetro(int nLati, double lungHato){
    return nLati * lungHato;
}
```

3

B-261

11.4 Overloading (I)

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

int massimo(int a, int b)
{
    cout << "Massimo per interi " << endl;
    return a > b ? a : b;
}

double massimo(double a, double b)
{
    cout << "Massimo per double" << endl;
    return a > b ? a : b;
}

/* int massimo(double a, double b)      ERRORE!
{   return int(a > b ? a : b);}*/

int main ()
{
    cout << massimo(10, 15) << endl;
    cout << massimo(12.3, 13.5) << endl;
    cout << massimo(12.3, 13) << endl;
    // ERRORE: ambiguo
    cout << massimo('a','r') << endl;

    return 0;
}
```

Massimo per interi 15
Massimo per double 13.5
Massimo per interi 114

B-262

11.4 Overloading (II)

// Sovrapposizione const - non const

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

int massimo(const int v[], int n)
{
    cout << "Array const ";
    int m = v[0];
    for (int i = 1; i < n; i++) m = m >= v[i] ? m : v[i];
    return m;
}

int massimo(int v[], int n)
{
    cout << "Array non const ";
    int m = v[0];
    for (int i = 1; i < n; i++) m = m >= v[i] ? m : v[i];
    return m;
}

int main ()
{
    const int N = 5;
    const int cv[N] = {1, 10, 100, 10, 1};
    cout << massimo(cv, N) << endl;
    int v[N] = {1, 10, 100, 10, 1};
    cout << massimo(v, N) << endl;

    return 0;
}
```

Array const 100
Array non const 100

B-263

Overloading (III) – Argomenti default

```
#include <iostream>
using namespace std;
double perimetro(int nLati, double lunghLato){
    return nLati * lunghLato;
}

int perimetro(int lato){ // perimetro del quadrato
    return 4 * lato;
}

int main() {
    cout << perimetro(5); // OK: stampa 20, il perimetro
                        // del quadrato di lato 5
    return 0;
}
```

NB: L'aggiunta di un argomento di default alla versione 1 di perimetro impedirebbe invece la compilazione, per ambiguità nella chiamata:

```
#include <iostream>
using namespace std;
double perimetro(int nLati, double lunghLato = 2.5){
    return nLati * lunghLato;
}

int perimetro(int lato){ // perimetro del quadrato
    return 4 * lato;
}

int main() {
    cout << perimetro(5); // ERRORE! Chiamata ambigua

    return 0;
}
```

B-264

Overloading (IV) – Argomenti default

Questa ulteriore variante, invece, torna a funzionare correttamente:

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

double perimetro(int nLati, double lungH lato = 2.5){
    return nLati * lungH lato;
}

int perimetro(double lato){ // perimetro del quadrato
    return 4 * lato;
}

int main() {
    cout<<perimetro(5.7); // OK: stampa 22, il perimetro
    // del quadrato di lato 5.7
    // (che varrebbe 22.8, ma il
    // risultato viene convertito
    // ad intero)

    return 0;
}
```

NB: Questa versione del codice funziona per lo stesso motivo per cui possono coesistere due funzioni con singolo argomento, una aevento argomento intero e l'altra argomento double (overloading delle funzioni)

B-265

12.4 Dichiarazioni typedef (I)

Parola chiave typedef:

- definisce degli identificatori (detti *nomi typedef*) che vengono usati per riferirsi a tipi nelle dichiarazioni.

Le dichiarazioni typedef non creano nuovi tipi

```
#include <iostream>
using namespace std;
int main ()
{
    int i = 1;
    typedef int* intP;
    intP p = &i;
    cout << *p << endl;           // 1

    typedef int vett[5];
    vett v = {1, 10, 100, 10, 1};
    cout << "v = [" << v[0];
    for (int j = 1; j < 5; j++)
        cout << ", " << v[j];
    cout << "]" << endl;

    typedef int intero;
    int a = 4;
    intero b = a; // OK, typedef non introduce un nuovo tipo
    cout << a << "\t" << b << endl; // 4 4

    return 0;
}
```

```
1
v = [1 10 100 10 1]
4      4
```

B-266

13.1 Memoria dinamica (I)

• Programmi precedenti:

- il programmatore specifica, utilizzando definizioni, numero e tipo delle variabili utilizzate.

• Situazioni comuni:

- il programmatore non è in grado di stabilire a priori il numero di variabili di un certo tipo che serviranno durante l'esecuzione del programma.
- Per variabili di tipo array, per esempio, dover specificare le dimensioni (costanti) è limitativo.
- Vorremmo poter dimensionare un array dopo aver scoperto durante l'esecuzione del programma, quanto deve essere grande.
- Per esempio, somma di N numeri inseriti da tastiera, con N letto da tastiera.

• Meccanismo della memoria libera (o memoria dinamica):

- risulta possibile allocare delle aree di memoria durante l'esecuzione del programma, ed accedere a tali aree mediante puntatori;
- gli oggetti così ottenuti sono detti *dinamici*, ed *allocati nella memoria libera*.

B-267

13.1 Memoria dinamica (II)

• Allocazione di oggetti dinamici:

- operatore prefisso *new*:
 - » ha come argomento il tipo dell'oggetto da allocare;
 - » restituisce l'indirizzo della memoria ottenuta, che può essere assegnato a un puntatore;
 - » se non è possibile ottenere la memoria richiesta, restituisce l'indirizzo 0.

```
#include <iostream>
using namespace std;
int main ()
{
    int* q;
    q = new int;
    *q = 10;
    cout << *q << endl;    // 10

    int * p;
    int n;
    cin >> n;
    p = new int [n];
    for (int i = 0; i < n; i++)
        p[i] = i;
    // anche *(p+i) = i;

    return 0;
}
```

B-268

13.1 Memoria dinamica (III)

- Buon esito dell'operatore *new*:
 - può essere controllato usando la funzione di libreria *set_new_handler()*, dichiarata nel file <new>:
 - » questa funzione ha come argomento una funzione *void* senza argomenti, che viene eseguita se l'operatore *new* fallisce (se l'allocazione non è possibile).

```
#include <cstdlib>
#include <iostream>
#include <new>
using namespace std;
void myhandler()
{
    cerr << "Memoria libera non disponibile" << endl;
    exit(1);
}
int main()
{
    int n;
    set_new_handler(myhandler);
    cout << "Inserisci la dimensione " << endl;
    cin >> n;
    int** m = new int* [n];
    for (int i = 0; i<n; i++)
        m[i] = new int[n];
    return 0;
}
```

B-269

Memoria dinamica (IV)

- Oggetti allocati nella memoria libera:
 - esistono finché non vengono distrutti dall'operatore prefisso *delete*:
 - » esso ha come argomento un puntatore all'oggetto da distruggere;
 - » può essere applicato solo ad un puntatore che indirizza un oggetto allocato mediante l'operatore *new* (in caso contrario si commette un errore).
- Se l'operatore *delete* non viene utilizzato:
 - gli oggetti allocati vengono distrutti al termine del programma.

```
int main()
{
    int n = 12;
    int* p = new int(10);
    cout << *p << endl;
    delete p;
    // cout << *p << endl;
    // SBAGLIATO, NON SEGNALE ERRORE
    p = 0;
    // cout << *p << endl;
    // SEGNALE ERRORE A TEMPO DI ESECUZIONE
    int* m = new int [n];
    delete[] m;
    // delete m;
    // ERRORE - oggetto non allocato dinamicamente
    return 0;
}
```

B-270

13.2 Liste (I)

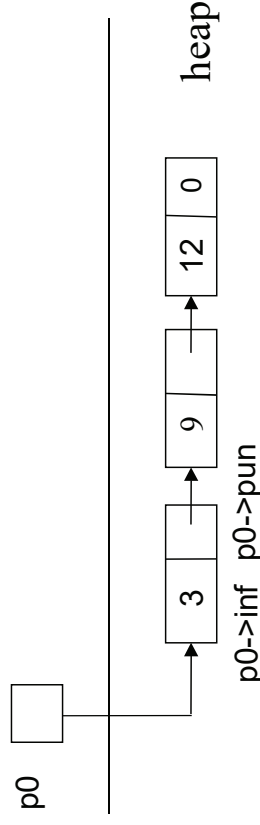
Problema: memorizzare numeri inseriti da tastiera finchè non viene inserito il carattere '.'.

Struttura dati, formata da elementi dello stesso tipo collegati in catena, la cui lunghezza varia dinamicamente.

•Lista:

- ogni elemento è una struttura, costituita da uno o più campi contenenti informazioni, e da un campo puntatore contenente l'indirizzo dell'elemento successivo;
- il primo elemento è indirizzato da un puntatore (puntatore della lista);
- il campo puntatore dell'ultimo elemento contiene il puntatore nullo.

```
typedef int T;  
struct elem  
{  
    T inf;  
    elem* pun;  
};
```



B-271

13.2 Liste (II)

Creazione di una lista

1. Leggere l'informazione
2. Allocare un nuovo elemento con l'informazione da inserire
3. Collegare il nuovo elemento al primo elemento della lista
4. Aggiornare il puntatore di testa della lista a puntare al nuovo elemento

```
typedef elem* lista; // tipo lista  
lista crealista(int n)  
{  
    lista p0 = 0; elem* p;  
    for (int i = 0; i < n; i++)  
    {  
        p = new elem;  
        cin >> p->inf;  
        p->pun = p0; p0 = p;  
    }  
    return p0;  
}
```

B-272

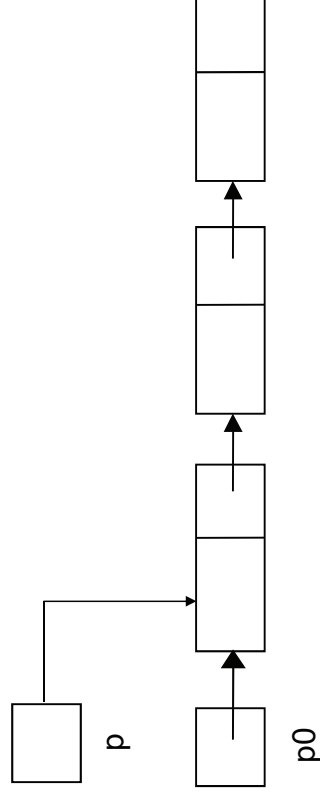
13.2 Liste (III)

Stampa lista

1. Scandire la lista dall'inizio alla fine e per ogni elemento stampare su video il campo informazione

```
void stampalista(lista p0)
{
    elem* p = p0;
    while (p != 0){
        cout << p->inf << " ";
        p = p->pun;
    }
}
```

La condizione nel while poteva essere scritta anche come:
`p != nullptr`
oppure come
`p != NULL`
(quest'ultima però a patto di includere una libreria che definisce il simbolo NULL a zero, mediante `define`)



B-273

13.2 Liste (IV)

Dealloca lista

Ci sono diversi modi per deallocare dalla memoria ogni singolo elemento della lista. Ad esempio si può fare mediante funzione ricorsiva, sfruttando la ricorsione in coda.

Qui di seguito invece viene mostrata la soluzione iterativa.

```
void dealloca(elem* p0){
    elem* p = p0, *q;
    while (p != nullptr){
        q = p;
        p = p->pun;
        delete q;
    }
}
```

ATTENZIONE! L'implementazione sottostante è invece ERRATA! Ed è uno degli errori più frequenti!

```
void dealloca_errata(elem* p0){
    elem* p = p0;
    while (p != nullptr){
        delete p;
        p = p->pun;
    }
}
```

Qui starei provando ad accedere ad un oggetto senza nome di tipo elem, allocato sullo heap (l'oggetto elem puntato da p), che però ora non esiste più. Infatti esso è stato deallocato alla riga precedente!

B-274

13.2 Liste(V)

Inserimento in testa

1. Allocare un nuovo elemento con l'informazione da inserire
2. Collegare il nuovo elemento al primo elemento della lista
3. Aggiornare il puntatore di testa della lista

```
void instesta(lista& p0, T a)
{
    elem* p = new elem;
    p->inf = a;
    p->pun = p0;
    p0 = p;
}
```

B-275

13.2 Liste(VI)

Estrazione dalla testa

Se la lista non è vuota

1. Aggiornare il puntatore di testa della lista
2. Deallocare l'elemento

```
bool esttesta(lista& p0, T& a)
{
    elem* p = p0;
    if (p0 == 0)
        return false;
    a = p0->inf;
    p0 = p0->pun;
    delete p;
    return true;
}
```

B-276

13.2 Liste(VII)

Inserimento in fondo

1. Scandire la lista fino all'ultimo elemento (membro $\text{pun} = 0$)
2. Allocare un nuovo elemento con l'informazione da inserire
3. Collegare l'ultimo elemento al nuovo elemento

```
void insfondo(lista& p0, T a)
{
    elem* p;
    elem* q;
    for (q = p0; q != 0; q = q->pun)
        p = q;
    q = new elem;
    q->inf = a;
    q->pun = 0;
    if (p0 == 0)
        p0 = q;
    else
        p->pun = q;
}
```

B-277

13.2 Liste(VIII)

Estrazione dal fondo

ATTENZIONE: necessita di due puntatori per scandire la lista

```
bool estfondo(lista& p0, T& a)
{
    elem* p = 0;
    elem* q;
    if (p0 == 0)
        return false;
    for (q = p0; q->pun != 0; q = q->pun)
        p = q;
    a = q->inf;
    // controlla se si estrae il primo elemento
    if (q == p0)
        p0 = 0;
    else
        p->pun = 0;
    delete q;
    return true;
}
```

B-278

Regola del cortocircuito (1)

Richiami alla regola del cortocircuito per l'operatore di AND logico

Supponiamo che l'AND venga chiamato su due espressioni, quella di sinistra e quella di destra:

expr1 && expr2

La regola del cortocircuito dice due cose:

1. Verrà valutata sicuramente per prima la prima espressione (quella di sinistra)
2. Abbiamo la certezza che la seconda espressione non venga valutata, qualora la prima abbia restituito un risultato falso

Esempio:

```
int main(){  
    int a = -1, b = 2;  
  
    if (++a && --b)    // la condizione è falsa perchè lo è expr1  
        cout<<b<<endl;    // cout non eseguita  
  
    cout<<a<<' '<<b<<endl;    // stampa 0 2  
  
    if (++a && --b)    // questa volta la condizione è vera  
        cout<<a<<' '<<b<<endl;    // stampa 1 1  
  
    return 0;  
}
```

B-279

Regola del cortocircuito (2)

Altri esempi di utilizzo della regola del cortocircuito nell'AND logico:

```
int main(){  
  
    int num, den;  
    cin>>num<<den;  
  
    // AND logico: Esempio 2  
    if (den != 0 && num/den > 5)  
        cout<<"Il rapporto e' maggiore di cinque"<<endl;  
  
    // AND logico: Esempio 3 (questo lo sfrutteremo nelle liste)  
    int *p = nullptr;  
    allocaVettoreMemDin(p,10);  
  
    if (p!= nullptr && *p < 0)  
        cout<<"Il primo elemento è negativo"<<endl;  
  
    return 0;  
}
```

B-280

Regola del cortocircuito (3)

Richiami alla regola del cortocircuito per l'operatore di OR logico

Supponiamo che l'OR logico venga chiamato su due espressioni, quella di sinistra e quella di destra:

expr1 || expr2

La regola del cortocircuito dice due cose:

1. Verrà valutata sicuramente per prima la prima espressione (quella di sinistra)
2. Abbiamo la certezza che la seconda espressione non venga valutata, qualora la prima abbia restituito un risultato vero

Esempio:

```
int main(){
    int a = -1, b = 2;

    if (++a || --b) // la condizione è vera dopo valutazione expr2
        cout<<b<<endl; // stampa 1

    cout<<a<<' '<<b<<endl; // stampa 0 1

    if (++a || --b) // condizione vera (non decrementa b)
        cout<<a<<' '<<b<<endl; // stampa 1 1

    return 0;
}
```

B-281

Regola del cortocircuito (4)

Altri esempi di utilizzo della regola del cortocircuito nell'OR logico:

```
int main(){
    // OR logico: Esempio 2
    int* p = nullptr;
    // ...
    if (vettoreAllocato(p) || allocaVettoreMemDin(p,10))
        cout<<"Il vettore risulta allocato"<<endl;

    // OR logico: Esempio 3
    // (definire la variabile indice, il vettore vett[]
    // e la funzione indiceInvalido())
    if (indiceInvalido(indice) || vett[indice] <= 0)
        cout<<"Operazione di decremento fallita";
    else
        vett[indice]--; // decremento solo se l'indice è valido
                        // e vett[indice] > 0

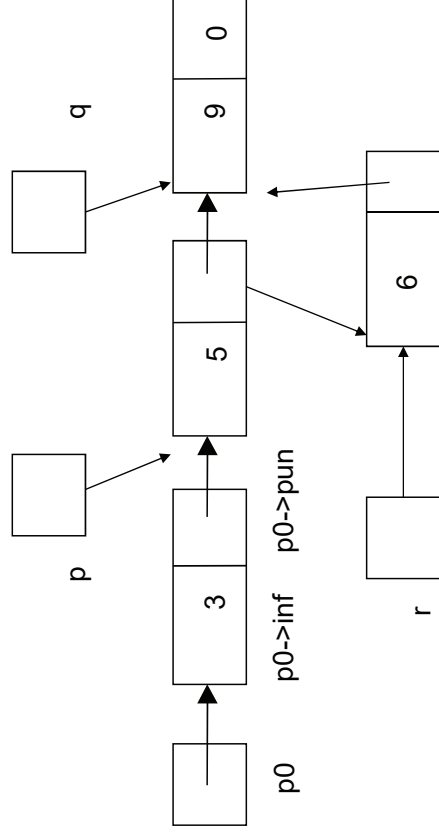
    return 0;
}
```

B-282

13.2 Liste(IX)

Inserimento in una lista ordinata

1. Scandire la lista finchè si incontra un elemento contenente nel campo inf un valore maggiore di quello da inserire oppure fine lista
2. Allocare un nuovo elemento con l'informazione da inserire
3. Inserire il nuovo elemento



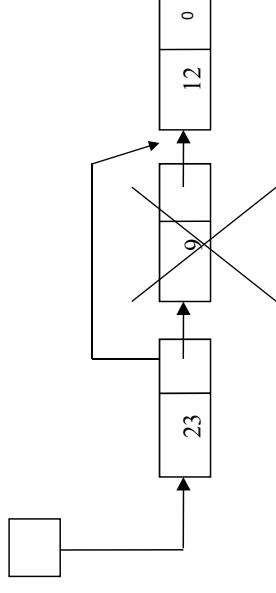
```
void insord(lista& p0, T a)
{
    elem* p = 0; elem* q; elem* r;
    for (q = p0; q != 0 && q->inf < a; q = q->pun)
        p = q;
    r = new elem;
    r->inf = a; r->pun = q;
    // controlla se si deve inserire in testa
    if (q == p0) p0 = r; else p->pun = r;
}
```

B-283

13.2 Liste(X)

Estrazione di un elemento da una lista

1. Scandire la lista finchè si incontra un elemento contenente l'informazione cercata
2. Se trovato, collegare i due nodi adiacenti
3. Deallocare l'elemento



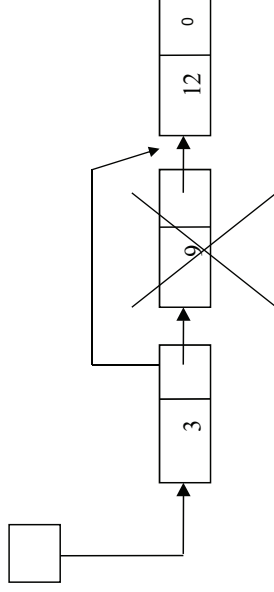
```
bool estrazione(lista& p0, T a)
{
    elem* p = 0; elem* q;
    for (q = p0; q != 0 && q->inf != a; q = q->pun)
        p = q;
    if (q == 0) return false;
    if (q == p0)
        p0 = q->pun;
    else
        p->pun = q->pun;
    delete q;
    return true;
}
```

B-284

13.2 Liste(XI)

Estrazione di un elemento da una lista ordinata

1. Scandire la lista finchè si incontra un elemento contenente l'informazione cercata o maggiore
2. Se trovato, collegare i due nodi adiacenti
3. Deallocare l'elemento



```
bool estrordinata(lista& p0, T a)
```

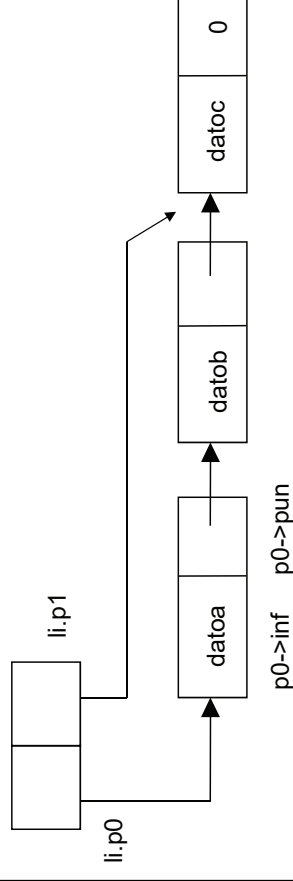
```
{
    elem* p = 0; elem* q;
    for (q = p0; q != 0 && q->inf < a; q = q->pun)
        p = q;
    if ((q == 0)||((q->info>a)) return false;
    if (q == p0)
        p0 = q->pun;
    else
        p->pun = q->pun;
    delete q;
    return true;
}
```

B-285

13.2 Liste con puntatore ausiliario (I)

```
struct lista_n
```

```
{
    elem* p0;
    elem* p1;
};
```



```
lista_n crealista1(int n)
```

```
{
    elem* p; lista_n li = {0, 0};
    if (n >= 1)
    {
        p = new elem;
        cin >> p->inf; p->pun = 0;
        li.p0 = p; li.p1 = p;
        for (int i = 2; i <= n; i++)
        {
            p = new elem;
            cin >> p->inf;
            p->pun = li.p0; li.p0 = p;
        }
        return li;
    }
}
```

B-286

13.2 Liste con puntatore ausiliario (II)

```

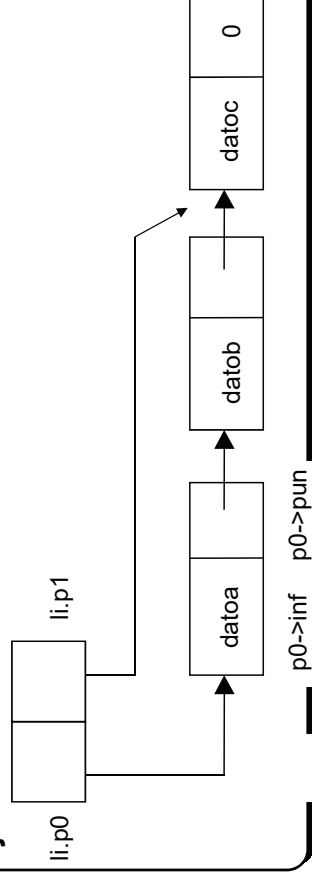
bool esttesta1(lista_n& li, T& a)
{
    elem* p = li.p0;
    if (li.p0 == 0)
        return false;
    a = li.p0->inf; li.p0 = li.p0->pun;
    delete p;
    if (li.p0 == 0)
        li.p1 = 0;
    return true;
}

```

```

void insfondo1(lista_n& li, T a)
{
    elem* p = new elem;
    p->inf = a; p->pun = 0;
    if (li.p0 == 0)
    {
        li.p0 = p; li.p1 = p;
    }
    else
    {
        li.p1->pun = p;
        li.p1 = p;
    }
}

```



B-287

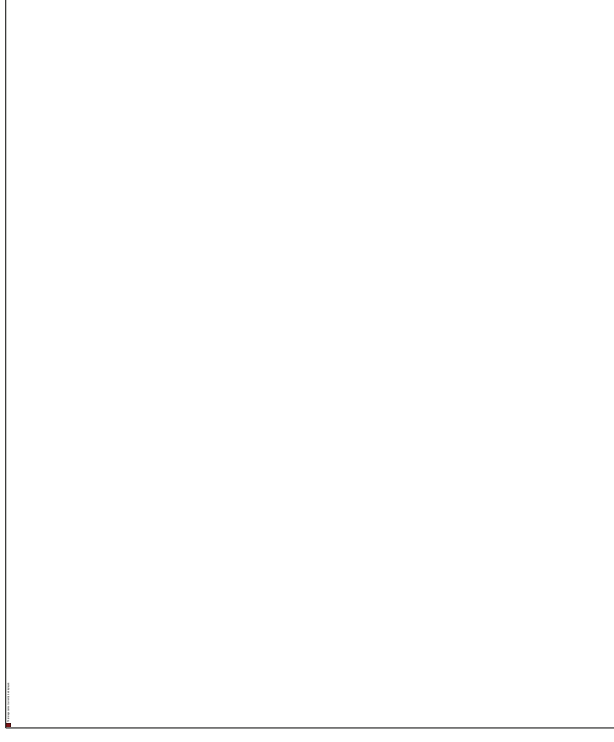
13.3 Liste complesse

```

struct nu_elem
{
    T inf;
    elem* prec;
    elem* succ;
};

struct lista_c
{
    nu_elem* p0;
    nu_elem* p1;
};

```



B-288

14.2.1 Visibilità

Programmi semplici:

- formati da poche funzioni, tutte contenute in un unico file, che si scambiano informazioni attraverso argomenti e risultati.

Programmi più complessi:

- si utilizzano tecniche di programmazione modulare:
 - suddivisione di un programma in diverse parti che vengono scritte, compilate (verificate e modificate) separatamente;
 - scambio di informazioni fra funzioni utilizzando oggetti comuni.

Visibilità (scope):

- campo di visibilità di un identificatore (parte di programma in cui l'identificatore può essere usato);

Regole che definiscono la visibilità degli identificatori (regole di visibilità):

- servono a controllare la condivisione delle informazioni fra i vari componenti di un programma:
 - permettono a più parti del programma di riferirsi ad una stessa entità (il nome dell'entità deve essere visibile alle parti del programma interessate);
 - impediscono ad alcune parti di un programma di riferirsi ad una entità (il nome dell'entità non deve essere visibile a tali parti).

B-289

14.3 Blocchi

// Sequenza di istruzioni racchiuse tra parentesi graffe

```
#include <iostream>
using namespace std;
void f(){
    int i = 2;
    cout << i << endl;
}

int main(){
    // cout << i << endl;          ERRORE!

    int i = 1, j = 5;
    cout << i << '\t' << j << endl;    // 1 5
    {
        cout << i << '\t' << j << endl;    // 1 5
        int i = 10;    // nasconde l'oggetto i del blocco super.
        cout << i << endl;    // 10
    }
    cout << i << endl;    // 1

    f();
    cout << i << endl;    // 1

    for (int a = 0; a < 2; a++)
    {
        int b = 2 * a;
        cout << a << '\t' << b << endl;    // 0 0 1 2
    }

    // cout << a << '\t' << b << endl;    ERRORE!

    return 0;
}
```

B-290

14.4 Unità di compilazione (I)

Unità di compilazione:

- costituita da un file sorgente e dai file inclusi mediante direttive `#include`;
- se il file da includere non è di libreria, il suo nome va racchiuso tra virgolette (e non fra parentesi angolari).

Esempio:

```
// file header.h
int f1(int); int f2(int);

// file main.cpp
#include "header.h"
int main()
{
    f1(3);
    f2(5);
    return 0;
}

// Unità di compilazione risultante
int f1(int); int f2(int);
int main()
{
    f1(3);
    f2(5);
    return 0;
}
```

B-291

14.4 Unità di compilazione (II)

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

int i;           // visibilità a livello di unità di compilazione

void leggi()     // visibilità a livello di unità di compilazione
{
    cout << "Inserisci un numero intero " << endl;
    cin >> i;
}

void scrivi()    // visibilità a livello di unità di compilazione
{
    cout << i << endl;
}

int main()
{
    leggi();
    scrivi();
}
```

Inserisci un numero intero

2
2

Identificatori di oggetti con visibilità a livello di unità di compilazione individuano oggetti condivisi da tutte le funzioni definite nell'unità di compilazione stessa.

B-292

14.4 Unità di compilazione (III)

// Operatore :: unario (risoluzione di visibilità)

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

```
int i = 1;      // visibilità a livello di unità di compilazione

int main()
{
    cout << i << endl;           // 1
    {
        int i = 5;              // visibilità locale
        cout << ::i << '\t' << i << endl; // 1 5
    }
    {
        int i = 10;             // visibilità locale
        cout << ::i << '\t' << i << endl; // 1 10
    }
    cout << ::i << endl;         // 1
    return 0;
}
```

1	
1	5
1	10
1	

B-293

14.5 Spazio di nomi (I)

Spazio di nomi:

- Insieme di dichiarazioni e definizioni racchiuse tra parentesi graffe, ognuna delle quali introduce determinate entità dette *membri*.
- Può essere dichiarato solo a livello di unità di compilazione o all'interno di un altro spazio dei nomi.
- Gli identificatori relativi ad uno spazio dei nomi sono visibili dal punto in cui sono dichiarati fino alla fine dello spazio dei nomi.

```
namespace uno
{
    struct st {int a; double d; };
    int n;
    void ff(int a)
    { /* ... */ }
    // ...
}

namespace due
{
    struct st {int a; double d; };
}

int main()
{
    uno::st ss1;
    using namespace due; // direttiva
    st ss2;
}
```

B-294

14.5 Spazio di nomi (II)

```
namespace uno
{
    struct st {int a; double d; };
    int n;
    void ff(int a)
    { /* ... */ }
    // ...
}

namespace due
{
    struct st {int a; double d; };
}

int main()
{
    uno::st ss1;
    using namespace uno;
    using namespace due;
    st ss2;
    due::st ss2;
}

// ERROR
```

L'uso della direttiva using namespace può generare delle ambiguità.

14.5 Spazio di nomi (III)

Lo spazio dei nomi è aperto, cioè è possibile usare più volte lo stesso identificatore di spazio dei nomi in successive dichiarazioni, per includervi nuovi membri.

Spazio dei nomi globali: costituito dalle definizioni e dichiarazioni a livello di unità di compilazione.
Per usare uno specifico identificatore si può usare la dichiarazione using.

```
#include <iostream>
using namespace std;
namespace uno
{
    int i = 2, j = 3;
}

int i = 4, j = 5; //Spazio globale
int main()
{
    using :: i;
    using uno :: j;
    cout << i << '\t' << j << endl;
    using uno::i;
    //
    {
        using uno::i;
        cout << i << '\t' << j << endl;
    }
}
```

3	5
4	2

14.6 Collegamento (I)

Programma:

- può essere formato da più unità di compilazione, che vengono sviluppate separatamente e successivamente collegate per formare un file eseguibile.

Collegamento:

- un identificatore ha *collegamento interno* se si riferisce a una entità accessibile solo da quella unità di compilazione;
 - uno stesso identificatore che ha collegamento interno in più unità di compilazione si riferisce in ognuna a una entità diversa;
- in una unità di compilazione, un identificatore ha *collegamento esterno* se si riferisce a una entità accessibile anche ad altre unità di compilazione;
 - tale entità deve essere unica in tutto il programma.

Regola default:

- gli identificatori con visibilità locale hanno collegamento interno;
- gli identificatori con visibilità a livello di unità di compilazione hanno collegamento esterno (a meno che non siano dichiarati con la parola chiave *const*).

14.6 Collegamento (II)

Oggetti e funzioni con collegamento esterno:

- possono essere utilizzati in altre unità di compilazione;
- in ciascuna unità in cui vengono utilizzati devono essere *dichiarati* (anche più volte).

Oggetto:

- viene solo dichiarato se si usa la parola chiave *extern* (e se non viene specificato nessun valore iniziale);
- viene anche definito se non viene usata la parola chiave *extern* (o se viene specificato un valore iniziale).

Funzione:

- viene solo dichiarata se si specifica solo l'intestazione (si può anche utilizzare la parola chiave *extern*, nel caso in cui la definizione si trovi in un altro file);
- viene anche definita se si specifica anche il corpo.

Osservazione:

- analogamente agli oggetti con visibilità a livello di unità di collegamento (oggetti *condivisi*), anche gli oggetti con collegamento esterno (oggetti *globali*) permettono la condivisione di informazioni fra funzioni.

14.6 Collegamento (III)

```
// ----- file file1.cpp----- //

int a = 1;           // collegamento esterno

const int N = 0;     // const, collegamento interno

static int b = 10;   // static, collegamento interno

// collegamento esterno
void f1(int a)
{
    int k;
    /* ... */
}

// static, collegamento interno
static void f2()
{
    /* ... */
}

struct punto        // collegamento interno (dichiarazione)
{
    double x;
    double y;
};

punto p1;           // collegamento esterno

                        (continua ...)
```

B-299

14.6 Collegamento (IV)

```
// ----- file file2.cpp----- //

#include <iostream>
using namespace std;

extern int a;         // solo dichiarazione
void f1(int);         // solo dichiarazione
void f2();            // solo dichiarazione
void f3();            // solo dichiarazione
double f4(double, double); // definizione mancante
                        // OK, non utilizzata

int main()
{
    cout << a << endl;           // OK, 1
    extern int b;                // dichiarazione
    cout << b << endl;           // ERRORE!
    f1(a);                       // OK
    f2();                        // ERRORE!
    f3();                        // ERRORE!
    punto p2;                    // ERRORE! punto non dichiarato
    p1.x = 10;                   // ERRORE! P1 non dichiarato

    return 0;
}
```

Stesso tipo in più unità di compilazione:

- viene verificata solo l'uguaglianza tra gli identificatori del tipo;
- se l'organizzazione interna non è la stessa, si hanno errori logici a tempo di esecuzione.

B-300

14.8 Classi di memorizzazioni

```
// Oggetti di classe automatica e di classe statica

#include <iostream>
using namespace std;

static int m;           // inizializzato a zero
int contaChiamateErrata()
{
    int n = 0;          // ERRATA!
    return ++n;         // di classe automatica
}

int contaChiamate()
{
    static int n = 0;    // di classe statica
    ++m;
    return ++n;
}

int main()
{
    for (int i = 0; i < 3; i++)
        cout << contaChiamateErrata() << endl;
    for (int i = 0; i < 3; i++){
        cout << contaChiamate() << "t";
        cout << m << endl;
    }
}
```

```
1
1
1
1
2
3
    1
    2
    3
```

B-301

14.8 Classi di memorizzazioni

```
// Attenzione: ordine (punti di sequenza)

#include <iostream>
using namespace std;

static int m;           // inizializzato a zero
int contaChiamateErrata()
{
    int n = 0;          // ERRATA!
    return ++n;         // di classe automatica
}

int contaChiamate()
{
    static int n = 0;    // di classe statica
    ++m;
    return ++n;
}

int main()
{
    for (int i = 0; i < 3; i++)
        cout << contaChiamateErrata() << endl;
    for (int i = 0; i < 3; i++)
        cout << m << "t" << contaChiamate() << endl;
}
```

```
1
1
1
1
2
3
    1
    2
    3
```

B-302

14.8 Classi di memorizzazioni

Facciamo un riassunto delle classi di memorizzazione:

Classe di memorizzazione	Esempio
Statica	<pre>int n = 10; // fuori definita fuori dai blocchi int main(){ ... }</pre>
Automatica	<pre>int main(){ int a = 3; // variabile di blocco int *p = nullptr; // variabile puntatore // def. in un blocco return 0; }</pre>
Dinamica	<pre>int main(){ int *p; p = new int; // in questo caso la variabile // intera senza nome (e // puntata da p) ha classe di // memorizz. dinamica // NB: ovviamente il puntatore p // ha classe di mem. automatica }</pre>

B-303

14.10 Effetti collaterali (I)

In generale una funzione può avere due effetti:

- l'effetto principale, che riguarda il calcolo di un valore da restituire mediante l'istruzione `return`;
- uno più effetti secondario, chiamati anche *effetti collaterale*, che raggruppa tutti gli effetti che una funzione può avere, escluso il suo effetto principale.

Per effetto collaterale si intende la modifica di una qualunque variabile non locale alla funzione.

Essenzialmente un effetto collaterale può essere generato in uno dei seguenti modi:

- modifica, all'interno della funzione, di una variabile globale
- assegnamento di un nuovo valore ad uno degli argomenti formali, qualora il passaggio sia per riferimento (in tal caso si modifica il valore della corrispondente variabile nel chiamante)
- modifica, per deferenza, della variabile del chiamante puntata da un puntatore. Analogamente definiremo effetto collaterale ottenuto tramite puntatore la modifica del contenuto di un elemento di un vettore, qualora quest'ultimo sia stato passato alla funzione mediante una variabile puntatore.
- modifica di una variabile locale statica di una funzione (cosa si intende per variabile locale statica lo vedremo però più avanti)

B-304

14.10 Effetti collaterali (II)

```
// Variabili globali

#include <iostream>
using namespace std;

int n = 5;

void incrementa1(){
    ++n;
}

int main(){
    incrementa1()<<endl;
    cout <<n<< endl;
    return 0;
}
```

6

B-305

14.10 Effetti collaterali (III)

```
// Modifica di una variabile passata per riferimento

#include <iostream>
using namespace std;

void incrementa2(int &n){
    ++n;
}

int main(){
    int a = 7;
    incrementa2(a)<<endl;
    cout <<a<< endl;
    return 0;
}
```

8

B-306

14.10 Effetti collaterali (IV)

// Modifica di una variabile del chiamante, mediante
argomento puntatore

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

void incrementa3(int *p){
    *p = (*p + 1);
}

int main(){
    int a = 9;
    incrementa3(&a)<<endl;
    cout <<a<< endl;
    return 0;
}
```

10

B-307

14.10 Effetti collaterali (V)

OSSERVAZIONE: Una funzione a ritorno void, che non abbia argomenti, e non produca effetti collaterali non serve a nulla (al massimo può servire per perdere tempo).

La prossima funzione sembra eccepire a questa regola:

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

void stampaCorniceDiAsterischi(){
    cout<<"*****",
}

int main(){
    stampaCorniceDiAsterischi();
    return 0;
}
```

```
*****
```

in quanto la chiamata della funzione provoca un effetto concreto e visibile. Invece l'osservazione fatta sopra rimane vera. Infatti, la funzione stampaCorniceDiAsterischi() **provoca un effetto collaterale**, in quanto modifica variabile globale cout. Ma questo fatto lo capiremo meglio più avanti, quando introdurremo la classe ostream.

B-308

14.10 Effetti collaterali (VI)

```
// Argomenti di tipo puntatore

#include <iostream>
using namespace std;

// Incrementa il maggiore. Restituisce il maggiore non
// incrementato.
int incremMag(int *pa, int *pb)
{
    if (*pa > *pb)
        return (*pa)++;
    else
        return (*pb)++;
}

int main()
{
    int i = 10, j = 100;
    cout << incremMag(&i, &j) << endl;
    cout << i << 't' << j << endl;
}
```

```
100
10  101
```

B-309

14.10 Effetti collaterali (VII)

```
// Problema: proprieta' associativa

#include <iostream>
using namespace std;

// Incrementa il maggiore. Restituisce il maggiore non
// incrementato.
int incremMag(int *pa, int *pb)
{
    if (*pa > *pb)
        return (*pa)++;
    else
        return (*pb)++;
}

int main()
{
    int i = 10, j = 100;
    cout << incremMag(&i, &j) + incremMag(&i, &j);
    cout << endl;
    cout << i << 't' << j << endl;
    i = 10, j = 100;
    cout << incremMag(&i, &j) - incremMag(&i, &j);
    cout << endl;
    cout << i << 't' << j << endl;
}
```

```
201
10  102
-1
10  102
```

B-310

14.10 Effetti collaterali (VIII)

// Problema: ordine di valutazione degli operandi

```
#include <iostream>
using namespace std;
// Incrementa il maggiore. Restituisce il maggiore non
// incrementato.
int incremMag(int *pa, int *pb)
{
    if (*pa > *pb)
        return (*pa)++;
    else
        return (*pb)++;
}

int main()
{
    int i = 10, j = 100;
    cout << i + j + incremMag(&i, &j) << endl;
    cout << i << '\t' << j << endl;
    i = 10, j = 100;
    cout << incremMag(&i, &j) + i + j << endl;
    cout << i << '\t' << j << endl;
}
```

210	
10	101
211	
10	101

B-311

14.10 Effetti collaterali (IX)

// Problema: ottimizzazione

```
#include <iostream>
using namespace std;
// Incrementa il maggiore. Restituisce il maggiore non
// incrementato.
int incremMag(int *pa, int *pb)
{
    if (*pa > *pb)
        return (*pa)++;
    else
        return (*pb)++;
}

int main()
{
    int i = 10, j = 100, r1, r2;
    r1 = i + j + incremMag(&i, &j); // 210
    r2 = i + j + incremMag(&i, &j); // 212
    cout << r1 << '\t' << r2 << endl;
    i = 10, j = 100;
    r1 = i + j + incremMag(&i, &j); // 210 210
    r2 = r1;
    cout << r1 << '\t' << r2 << endl;
}
```

210	212
210	210

B-312

14.11 Moduli (I)

Modulo:

- parte di programma che svolge una particolare funzionalità e che risiede su uno o più file;
- moduli *servitori* e moduli *clienti*.

Modulo servitore:

- offre (esporta) risorse di varia natura, come funzioni, variabili (globali) e tipi.
- costituito normalmente da due file (con estensione *h* e *cpp*, rispettivamente):
 - intestazione o interfaccia (dichiarazione dei servizi offerti);
 - realizzazione.

Separazione fra interfaccia e realizzazione:

- ha per scopo l'occultamento dell'informazione (*information hiding*);
 - semplifica le dipendenze fra i moduli;
 - permette di modificare la realizzazione di un modulo senza influenzare il funzionamento dei suoi clienti.

Modulo cliente:

- utilizza (importa) risorse offerte dai moduli servitori (include il file di intestazione di questi);
- viene scritto senza conoscere i dettagli relativi alla realizzazione dei moduli servitori.

B-313

14.11 Moduli (II)

```
// ESEMPIO PILA
// MODULO SERVER

// file pila.h
typedef int T;
const int DIM = 5;
struct pila
{
    int top;
    T stack[DIM];
};

void inip(pila& pp);
bool empty(const pila& pp);
bool full(const pila& pp);
bool push(pila& pp, T s);
bool pop(pila& pp, T& s);
void stampa(const pila& pp);

// file pila.cpp

#include <iostream>
#include "pila.h"
using namespace std;

// inizializzazione della pila
void inip(pila& pp)
{
    pp.top = -1;
}
....
```

B-314

14.11 Moduli (III)

```
// ESEMPIO PILA
// MODULO CLIENT

// file pilaMain.cpp

#include <iostream>
#include "pila.h"
using namespace std;

int main()
{
    pila st;
    inip(st);
    T num;
    if (empty(st)) cout << "Pila vuota" << endl;
    for (int i = 0; i < DIM; i++)
        if (push(st,DIM - i))
            cout << "Inserito " << DIM - i <<
                ". Valore di top: " << st.top << endl;
        else
            cerr << "Inserimento di " << i << " fallito" <<
                endl;
    if (full(st)) cout << "Pila piena" << endl;
}
```

B-315

14.11 Moduli (IV)

Comandi per la compilazione ed il linking

FILE SINGOLO

Compilazione a riga di comando:

```
>> g++ -c main.cpp -o main.obj // genera il file oggetto
// main.obj
```

Linking a riga di comando:

```
>> g++ main.obj -o main.exe // genera il file eseguibile
// main.exe
```

FILE MULTIPLI (programmazione a moduli)

Compilazione a riga di comando:

```
>> g++ -c main.cpp -o main.obj // genera main.obj
>> g++ -c pila.cpp -o pila.obj // genera pila.obj
```

Linking a riga di comando:

```
>> g++ main.obj pila.obj -o main.exe // gen. main.exe
```

COMPILAZIONE E LINKING INSIEME

```
>> g++ main.cpp pila.cpp -o main.exe
```

COMPILAZIONE E LINKING INSIEME SOTTO LINUX

```
$ g++ main.cpp pila.cpp -o main
```

In quest'ultimo caso il comando per l'esecuzione è:
\$./main
mentre sotto windows basta scrivere
>> main

B-316

14.11.1 Astrazioni procedurali

Astrazioni procedurali.

- i moduli servitori mettono a disposizione dei moduli clienti un *insieme di funzioni*;
- le dichiarazioni di tali funzioni si trovano in un file di intestazione che viene incluso dai moduli clienti;
- la realizzazione di tali funzioni si trova in un file diverso (che non viene incluso).
- tali funzioni vengono usate senza che sia necessaria alcuna conoscenza della loro struttura interna.

Esempio:

- le funzioni di libreria per l'elaborazione delle stringhe sono contenute in un modulo il cui file di intestazione è *<cstring>*, e il loro utilizzo non richiede alcuna conoscenza sulla loro realizzazione.

Parte III

(le classi nel linguaggio C++)

C-1

I TIPI CLASSE

C-2

16.2 Tipi classe (I)

Nell'esempio precedente, la struttura interna della pila è visibile ai moduli che utilizzano istanze della pila. In questo modo non si riesce a realizzare compiutamente il tipo di dato astratto.

Per ovviare a questo problema, il C++ mette a disposizione le classi.

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

```
typedef int T;
const int DIM = 5;
class pila
{
    int top;
    T stack[DIM];
public:
    void inip();
    bool empty();
    bool full();
    bool push(T s);
    bool pop(T& s);
    void stampa();
};
```

```
int main()
{
    pila st;
    st.inip();
    st.push(1);
    st.top = 10;
    // ERRORE!
    // 'int pila::top' is private within this context
}
```

C-3

16.2 Tipi classe (II)

```
basic-class-type-declaration
    class-type-specifier ;
class-type-specifier
    class identifier[opt { class-element-seq } ]
class-element
    access-indicator[opt class-member-section]
access-indicator
    access-specifier :
access-specifier
    private
    protected
    public
```

In genere utilizzeremo la forma semplificata seguente:

```
class nome
{
    parte privata
protected:
    parte protetta
public:
    parte pubblica
};
```

Un membro di una classe può essere:

- un tipo (enumerazione o struttura);
- un campo dati (oggetto non inizializzato);
- una funzione (dichiarazione o definizione);
- una classe (diversa da quella della classe a cui appartiene).

C-4

16.2 Tipi classe (III)

```
// Numeri Complessi (parte reale, parte immaginaria)
#include<iostream>
using namespace std;
class complesso
{
    double re, im;
public:
    void iniz_compl(double r, double i) {re = r; im = i;}
    double reale() {return re;}
    double immag() {return im;}
    /* ... */
    void scrivi() {cout << '(' << re << ", " << im << ')';}
};

int main()
{
    complesso c1, c2;
    c1.iniz_compl(1.0, -1.0);
    c1.scrivi(); cout << endl;
    //inizializzazione
    // (1, -1)

    c2.iniz_compl(10.0, -10.0);
    c2.scrivi(); cout << endl;
    // (10, -10)

    complesso* cp = &c1;
    cp->scrivi(); cout << endl;
    // (1, -1)

    return 0;
}
```

```
(1, -1)
(10, -10)
(1, -1)
```

C-5

16.2 Tipi classe (IV)

```
// Esempio Numeri Complessi

#include<iostream>
using namespace std;
class complesso
{
public:
    void iniz_compl(double r, double i) {re = r; im = i;}
    double reale() {return re;}
    double immag() {return im;}
    /* ... */
    void scrivi() {cout << '(' << re << ", " << im << ')';}
private:
    double re, im;
};

int main(){
    complesso c1, c2;
    c1.iniz_compl(1.0, -1.0);
    c1.scrivi(); cout << endl;
    // (1, -1)

    c2.iniz_compl(10.0, -10.0);
    c2.scrivi(); cout << endl;
    // (10, -10)

    complesso* cp = &c1;
    cp->scrivi(); cout << endl;
    // (1, -1)

    return 0;
}
```

```
(1, -1)
(10, -10)
(1, -1)
```

C-6

16.2 Tipi classe (V)

```
// Numeri Complessi (rappresentazione polare)
#include<iostream>
#include<cmath>
using namespace std;
class complesso
{ double mod, arg;
public:
    void iniz_compl(double r, double i)
        {mod = sqrt(r*r + i*i); arg = atan(i/r);}
    double reale() {return mod*cos(arg);}
    double immag() {return mod*sin(arg);}
    /* ... */
    void scrivi()
        {cout << '(' << reale() << ", " << immag() << ')';}
};
int main(){
    complesso c1, c2;
    c1.iniz_compl(1.0, -1.0);           // (1, -1)
    c1.scrivi(); cout << endl;

    c2.iniz_compl(10.0, -10.0);
    c2.scrivi(); cout << endl;         // (10, -10)

    complesso* cp = &c1;
    cp->scrivi(); cout << endl;        // (1, -1)

    return 0;
}
```

```
(1, -1)
(10, -10)
(1, -1)
```

C-7

16.2 Tipi classe (VI)

Le funzioni membro definite nella dichiarazione di una classe sono funzioni inline.
Le funzioni membro possono anche essere definite esternamente utilizzando l'operatore di risoluzione di visibilità'.

```
#include<iostream>
using namespace std;
class complesso
{
public:
    void iniz_compl(double r, double i);
    double reale();
    double immag();
    /* ... */
    void scrivi();
private:
    double re, im;
};

void complesso::iniz_compl(double r, double i)
{re = r; im = i;}

double complesso::reale()
{return re;}

double complesso::immag()
{return im;}

void complesso::scrivi()
{cout << '(' << re << ", " << im << ')';}
```

C-8

16.3 Operazioni su oggetti classe

Un oggetto appartenente ad una classe si chiama oggetto classe o istanza della classe

```
class complesso
{ /* ... */ };

int main()
{
    complesso c1;
    c1.iniz_compl(1.0, -1.0);
    complesso c2 = c1, c3(c2);
    //Inizializzazione - ricopiatura membro a membro
    c1.scrivi(); cout << endl; // (1, -1)
    c2.scrivi(); cout << endl; // (1, -1)
    c3.scrivi(); cout << endl; // (1, -1)

    complesso *pc1 = new complesso(c1);
    pc1->scrivi(); cout << endl; // (1, -1)

    complesso* pc2 = &c1;
    pc2->scrivi(); cout << endl; // (1, -1)

    return 0;
}
```

```
(1, -1)
(1, -1)
(1, -1)
(1, -1)
(1, -1)
```

C-9

16.3 Operazioni su oggetti classe

```
class complesso
{ /* ... */ };

complesso somma(complesso a, complesso b)
{
    complesso s;
    s.iniz_compl(a.reale()+b.reale(),
                a.immag() + b.immag());
    return s;
}

int main()
{
    complesso c1, c2, c3;
    c1.iniz_compl(1.0, -1.0);
    c2 = c1; //Assegnamento (ricopiatura membro a membro)
    c1.scrivi(); cout << endl; // (1, -1)
    c2.scrivi(); cout << endl; // (1, -1)

    c3 = somma(c1, c2); // oggetti argomento di funzioni
                        // e restituiti da funzioni
    c3.scrivi(); cout << endl; // (2, -2)

    return 0;
}
```

```
(1, -1)
(1, -1)
(2, -2)
```

ATTENZIONE: non esistono altre operazioni predefinite

C-10

16.4 Puntatore this (I)

Nella definizione di una funzione membro, il generico oggetto a cui la funzione viene applicata può essere riferito tramite il puntatore costante predefinito `this`

```
class complesso
{ /* .../
    complesso scala(double s) // restituisce un valore
    {
        re *= s;
        im *= s;
        return *this;
    }
};

int main()
{
    complesso c1;
    c1.iniz_compl(1.0, -1.0);
    c1.scale(2);
    c1.scrivi(); cout << endl;           // (2, -2)

    complesso c2;
    c2.iniz_compl(1.0, -1.0);
    c2.scale(2).scale(2);
    c2.scrivi(); cout << endl;         // (2, -2)

    return 0;
}
```

(2, -2)
(2, -2)

C-11

16.4 Puntatore this (II)

```
class complesso
{ /* .../
    complesso& scala(double s) // restituisce un riferimento
    {
        re *= s;
        im *= s;
        return *this;
    }
};

int main()
{
    complesso c1;
    c1.iniz_compl(1.0, -1.0);
    c1.scale(2);
    c1.scrivi(); cout << endl;           // (2, -2)

    complesso c2;
    c2.iniz_compl(1.0, -1.0);
    c2.scale(2).scale(2);
    c2.scrivi(); cout << endl;         // (4, -4)

    return 0;
}
```

(2, -2)
(4, -4)

C-12

16.5 Visibilità a livello di classe (I)

Una classe individua un *campo di visibilità*.

- Gli identificatori dichiarati all'interno di una classe sono visibili dal punto della loro dichiarazione fino alla fine della classe stessa.
- Nel corpo delle funzioni membro sono visibili tutti gli identificatori presenti nella classe (anche quelli non ancora dichiarati).
- Se nella classe viene riutilizzato un identificatore dichiarato all'esterno della classe, la dichiarazione fatta nella classe nasconde quella più esterna.
- All'esterno della classe possono essere resi visibili mediante l'operatore di risoluzione di visibilità :: applicato al nome della classe:
 - Le funzioni membro, quando vengono definite;
 - Un tipo o un enumeratore, se dichiarati nella parte pubblica della classe
 - Membri statici
- L'operatore di visibilità non può essere utilizzato per i campi dati non statici.

C-13

16.5 Visibilità a livello di classe (II)

```
class grafica
{ // ...
    public:
        enum colore { rosso, verde, blu };
};

class traffico
{ // ...
    public:
        enum colore { rosso, giallo, verde };
        colore stato();
// ...
};

traffico::colore traffico::stato()
{ /* ... */ }

// ...
grafica::colore punto;
traffico::colore semaforo;

int main()
{
    // ...
    punto = grafica::rosso;
    semaforo = traffico::rosso;
    //...
}
```

C-14

16.5 Visibilità a livello di classe (III)

Nella dichiarazione di una classe si può dichiarare un'altra classe (classe annidata). In questo caso, per riferire membri della classe annidata esternamente alla classe che la contiene devono essere usati due operatori di risoluzione di visibilità.

Nessuna delle due classi ha diritti di accesso alla parte privata dell'altra.

```
class A
{
    class B
    {
        int x;
    public:
        void iniz_B(int);
    };
    int y;
    public:
        void iniz_A();
};

void A::iniz_A(){ y = 0; }
void A::B::iniz_B(int n) { x = n; }

// void A::iniz_A(){ x = 3; } ERRORE: A non può accedere
// alla parte privata di B

// void A::B::iniz_B(int n){ y = n; } // ERRORE: B non può
// accedere alla parte privata
// di A
```

C-15

16.6 Modularità e ricompilazione (I)

L'uso delle classi permette di scrivere programmi in cui l'interazione tra moduli è limitata alle interfacce.

Quando si modifica la parte dati di una classe, anche se gli altri moduli utilizzano solo l'interfaccia, si rende necessaria la ricompilazione dei moduli. Infatti, se in un modulo cliente viene definito un oggetto classe, il compilatore ha bisogno di sapere la dimensione dell'oggetto.

Ai fini della ricompilazione non è appropriato separare una classe in interfaccia e struttura interna, ma piuttosto effettuare una suddivisione in file.

Un file di intestazione che contiene la dichiarazione della classe e deve essere incluso in tutti i moduli cliente.

Un file di realizzazione che contiene tutte le definizioni delle funzioni membro.

La modifica di un file di intestazione richiede la ricompilazione di tutti i file che lo includono.

La modifica di un file di realizzazione richiede solo la ricompilazione di questo file.

C-16

16.6 Modularità e ricompilazione (II)

```
// file complesso.h
class complesso
{
    double re, im;
public:
    void iniz_compl(double r, double i);
    double reale();
    double immag();
    /* ... */
    void scrivi();
    complesso& scala(double s);
};

// file complesso.cpp
#include<iostream>
#include "complesso.h"
using namespace std;

void complesso::iniz_compl(double r, double i)
{re = r; im = i;}

double complesso::reale() {return re;}

double complesso::immag() {return im;}

void complesso::scrivi()
{cout << '(' << re << ", " << im << ')';}

complesso& complesso::scala(double s)
{
    re *= s; im *= s; return *this;}
}
```

C-17

16.6 Modularità e ricompilazione (III)

```
// file main.cpp
#include<iostream>
#include "complesso.h"
using namespace std;
int main()
{
    complesso c1;
    c1.iniz_compl(1.0, -1.0);
    c1.scala(2);
    c1.scrivi(); cout << endl; // (2, -2)

    complesso c2;
    c2.iniz_compl(1.0, -1.0);
    c2.scala(2).scala(2);
    c2.scrivi(); cout << endl; // (4, -4)

    return 0;
}
```

C-18

16.7 Funzione globali

Funzioni globali: funzioni che non sono membro di alcuna classe.
Non possono accedere alla parte privata delle classi e non possono usare il puntatore this.

```
#include<iostream>
#include "complesso.h"

complesso somma(const complesso& a, const
                 complesso& b)
{
    complesso s;
    s.iniz_compl(a.reale()+b.reale(),
                 a.immag() + b.immag());
    return s;
}

using namespace std;
int main()
{
    complesso c1, c2, c3;
    c1.iniz_compl(1.0, -1.0);
    c2.iniz_compl(2.0, -2.0);

    c3 = somma(c1,c2);
    c3.scrivi(); cout << endl;           // (3, -3)

    return 0;
}
```

(3, -3)

C-19

16.8 Costruttori (I)

Costruttore: funzione membro il cui nome è il nome della classe. Se definita viene invocata automaticamente tutte le volte che viene creata un'istanza di classe (subito dopo che è stata riservata la memoria per i campi dati)

```
// file complesso.h
class complesso
{
    double re, im;
public:
    complesso(double r, double i);
    double reale();
    double immag();
    /* ... */
    void scrivi();
};

// file complesso.cpp
/* .... */
complezzo::complezzo(double r, double i)
{re = r; im = i;}

// file main.cpp
/* .... */
int main()
{
    complesso c1(1.0, -1.0);
    c1.scrivi(); cout << endl;           // (1, -1)

    complesso c2;           // ERRORE: non esiste complesso()
    complesso c3(3);        // ERRORE: non esiste complesso(int)
}
```

C-20

16.8 Costruttori default(I)

Per definire oggetti senza inizializzatori occorre definire un costruttore di default che può essere chiamato senza argomenti. Due meccanismi.

1) Meccanismo dell'overloading

```
// file complesso.h
class complesso
{
    double re, im;
public:
    complesso();
    complesso(double r, double i);
    double reale();
    double immag();
    /* ... */
    void scrivi();
};

// file complesso.cpp
/* .... */
complesso :: complesso() // costruttore di default
{re = 0; im = 0;}

complesso :: complesso(double r, double i)
{re = r; im = i;}

// file main.cpp
/* .... */
int main()
{
    complesso c1(1.0, -1.0);
    c1.scrivi(); cout << endl;
    complesso c2;
    complesso c3(3.0);
}
```

C-21

16.8 Costruttori default(II)

2) Meccanismo degli argomenti default

```
// file complesso.h
class complesso
{
    double re, im;
public:
    complesso(double r = 0, double i = 0);
    double reale();
    double immag();
    /* ... */
    void scrivi();
};

// file complesso.cpp
/* .... */
complesso :: complesso(double r, double i)
{re = r; im = i;}

// file main.cpp
/* .... */
int main()
{
    complesso c1(1.0, -1.0); // (1, -1)
    complesso c2;           // (0, 0)
    complesso c3(3.0);      // (3, 0)
    complesso c4 = 3.0;      // (3, 0)
    // possibile quando il costruttore può essere
    // invocato con un solo argomento
}
```

ATTENZIONE: I due meccanismi non possono essere utilizzati contemporaneamente.

C-22

16.8 Costruttori per allocare memoria

Spesso un costruttore richiede l'allocazione di memoria libera per alcuni membri dell'oggetto da costruire

```
// file stringa.h
class stringa
{
    char* str;
public:
    stringa(const char s[]);
    // ...
};

// file stringa.cpp
stringa::stringa(const char s[])
{
    str = new char[strlen(s)+1];
    strcpy(str, s); // copia la stringa s nella stringa str
}
// ...

// file main.cpp
int main()
{
    stringa inf("Fondamenti di Progr.");
    /* ... */
}
```

Quando definiamo una variabile di tipo `stringa`, viene riservata memoria per `str` e quindi viene richiamato il costruttore che alloca un array della dimensione richiesta.

C-23

16.8 Costruttori per oggetti dinamici

I costruttori definiti per una classe vengono chiamati implicitamente anche quando un oggetto viene allocato dinamicamente

```
// file main.cpp

#include<iostream>
#include "complesso.h"
int main()
{
    complesso* pc1 = new complesso(3.0, 4.0);
    complesso* pc2 = new complesso(3.0);
    complesso* pc3 = new complesso;
    /* ... */
}
```

C-24

16.9 Distruttori

Distruttore: funzione membro che viene invocata automaticamente quando un oggetto termina il suo tempo di vita.

Un distruttore non ha argomenti.

Obbligatori quando il costruttore alloca memoria dinamica.

```
// file stringa.h
class stringa
{
    char* str;
public:
    stringa(const char s[]);           // costruttore
    ~stringa();                       // distruttore
    // ...
};

// file stringa.cpp
stringa::stringa(const char s[])
{
    str = new char[strlen(s)+1];
    strcpy(str, s); // copia la stringa s nella stringa str
}

stringa::~~stringa()
{
    delete[] str;
} /* */

// file main.cpp
/* */
int main()
{
    stringa* ps = new stringa("Fondamenti di Progr.");
    /* */
    delete ps;
}
```

C-25

16.9 Regole di chiamata

I costruttori vengono richiamati con le regole seguenti:

1. Per gli oggetti statici, all'inizio del programma;
2. Per gli oggetti automatici, quando viene incontrata la definizione;
3. Per gli oggetti dinamici, quando viene incontrato l'operatore new;
4. Per gli oggetti membri di altri oggetti, quando questi ultimi vengono costruiti.

I distruttori vengono richiamati con le regole seguenti:

1. Per gli oggetti statici, al termine del programma;
2. Per gli oggetti automatici, all'uscita dal blocco in cui sono definiti;
3. Per gli oggetti dinamici, quando viene incontrato l'operatore delete;
4. Per gli oggetti membri di altri oggetti, quando questi ultimi vengono distrutti.

Gli oggetti con lo stesso tempo di vita vengono distrutti nell'ordine inverso a quello in cui sono definiti.

C-26

16.9 Costruttori/Distruttori Esempio

```
// file stringa.cpp
/*...*/
stringa::~stringa(const char s[])
{
    str = new char[strlen(s)+1];
    strcpy(str, s);    // copia la stringa s nella stringa str
    cout << "C " << str << endl;
}

stringa::~~stringa()
{
    cout << "D " << str << endl;
    delete[] str;
}

// file main.cpp
#include <cstdlib>
#include "stringa.h"
int main()
{
    stringa* ps = new stringa("Fondamenti di Progr.");
    {
        stringa s("Reti Logiche");
    }
    delete ps;
    /*...*/
    return 0;
}
```

C Fondamenti di Progr.
C Reti Logiche
D Reti Logiche
D Fondamenti di Progr.

C-27

16.10 Costruttori di copia (I)

Costruttore di copia predefinito: costruttore che agisce fra due oggetti della stessa classe effettuando una ricopiatura membro a membro dei campi dati.

Viene applicato:

1. Quando un oggetto classe viene inizializzato con un altro oggetto della stessa classe
2. Quando un oggetto classe viene passato ad una funzione come argomento valore;
3. Quando una funzione restituisce come valore un oggetto classe (mediante l'istruzione return)

Nel caso della classe complesso il costruttore di copia predefinito ha la seguente forma:
complesso(const complesso& c) { re = c.re; im = c.im;;};

C-28

16.10 Costruttori di copia (II)

```
// file main.cpp
#include "complesso.h"
complesso fun (complesso c)
{
    /* .... */
    return c;
}

int main()
{
    complesso c1 (3.2, 4);           // <3.2, 4>
    c1.scrivi(); cout << endl;

    complesso c2 (c1);              // costruttore di copia
    c2.scrivi(); cout << endl;

    complesso c3 = c1;              // costruttore di copia
    c3.scrivi(); cout << endl;

    c3 = fun(c1);                   // istanza di funzione
                                    // costruttore di copia applicato 2 volte:
                                    // chiamata di funzione
                                    // restituzione del valore
    c3.scrivi(); cout << endl;      // <3.2, 4>

    return 0;
}
```

C-29

16.10 Costruttori di copia (III)

Per osservare quando il costruttore di copia viene effettivamente richiamato, ridefiniamo il costruttore di copia nella classe complesso ed eseguiamo lo stesso programma principale

```
// file complesso.h
class complesso
{
    double re, im;
public:
    complesso(const complesso&); // costruttore di copia
    /* ... */
};

// file complesso.cpp
/* .... */
complesso :: complesso(const complesso& c)
{
    re = c.re; im = c.im;
    cout << "Costruttore di copia " << endl;
}
```

```
(3.2, 4)
Costruttore di copia
(3.2, 4)
Costruttore di copia
(3.2, 4)
Costruttore di copia
Costruttore di copia
(3.2, 4)
```

C-30

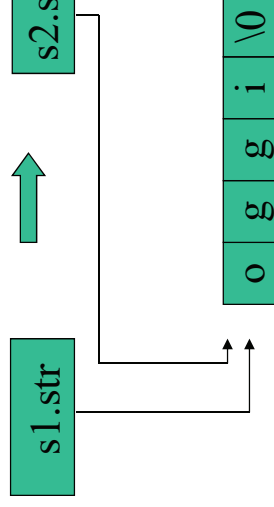
16.10 Costruttori di copia (IV)

Il costruttore di copia deve essere ridefinito per quelle classi che utilizzano memoria libera e prevedono un distruttore.

```
// file stringa.h
class stringa
{
    char* str;
public:
    stringa(const char s[]);           // distruttore
    ~stringa();                       // ...
};

// file main.cpp
#include <cstdlib>
#include "stringa.h"
int main()
{
    stringa s1("oggi");
    stringa s2(s1);                  // costruttore di copia predefinito

    return 0;
}
```



C-31

16.10 Costruttori di copia (V)

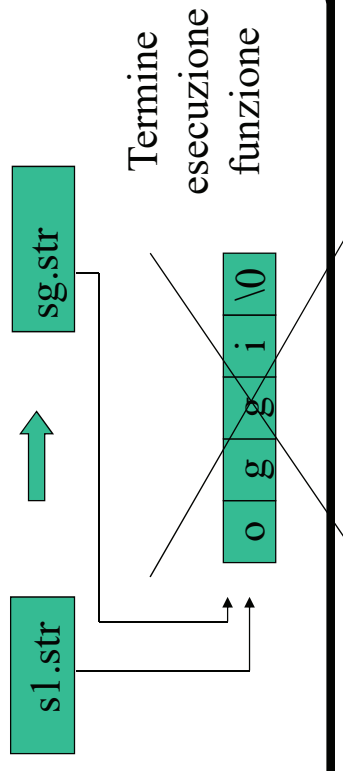
Consideriamo l'esempio seguente.

```
// file main.cpp
#include <cstdlib>
#include "stringa.h"

void ff(stringa sg)
{
    // ...
}

int main()
{
    stringa s1("oggi");
    ff(s1);

    return 0;
}
```



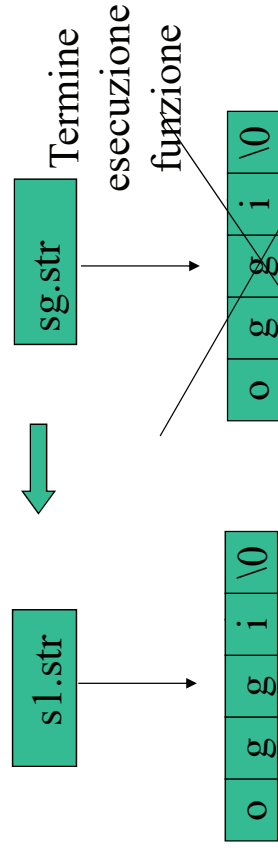
C-32

16.10 Costruttori di copia (VI)

```
// file stringa.h
class stringa
{
    char* str;
public:
    stringa(const stringa &);
    // ...
};

// file stringa.cpp
/* ... */
stringa::stringa(const stringa& s)
{
    str = new char[strlen(s.str) + 1];
    strcpy(str, s.str);
}

// file main.cpp
/* ... */
int main()
{
    stringa s1("oggi");
    ff(s1);
    /* ... */
}
```



C-33

16.10 Costruttori di copia (VI)

Se il costruttore di copia non viene ridefinito, viene usato il costruttore di copia prefinito.

Per impedire l'utilizzo, occorre inserire la sua dichiarazione nella parte privata della classe stessa senza alcuna ridefinizione.

Nel caso in cui il costruttore di copia venga nascosto, non si possono avere funzioni che abbiano argomenti valore del tipo della classe o un risultato valore del tipo della classe.

```
// file stringa.h
class stringa
{
    char* str;
    stringa(const stringa &);
public:
    // ...
};

// file main.cpp
/* ... */
void ff(stringa sg)           // ERRORE
{
    // ...
}

void ff(stringa& sg)          // OK
{
    // ...
}
```

C-34

16.11 Funzioni friend (I)

Una funzione è *friend* (*amica*) di una classe se una sua dichiarazione, preceduta dalla parola chiave *friend*, appare nella dichiarazione di tale classe. Una funzione *friend* può accedere ai membri pubblici e privati della classe, usando i selettori di membro.

```
// file complesso.h
class complesso
{
    double re, im;
public:
    double reale();
    double immag();
    /* ... */
};

// file main.cpp
complesso somma(const complesso& a, const
                complesso& b)
{
    complesso s(a.reale()+b.reale(), a.immag() + b.immag());
    return s;
    // Non si può accedere ai
    // membri privati della classe
}

int main()
{
    complesso c1 (3.2, 4);
    complesso c2 (c1);
    complesso c3 = somma(c1,c2);
    c3.scrivi(); cout << endl;           // (6.4, 8)

    return 0;
}
```

C-35

16.11 Funzioni friend (II)

```
// file complesso.h
class complesso
{
    double re, im;
public:
    double reale();
    double immag();
    friend complesso somma(const complesso& a,
                          const complesso& b);
    /* ... */
};

// file main.cpp
complesso somma(const complesso& a, const
                complesso& b)
{
    complesso s(a.re+b.re, a.im + b.im);
    return s;
    // Si può accedere ai
    // membri privati della classe
}

int main()
{
    complesso c1 (3.2, 4);
    complesso c2 (c1);
    complesso c3 = somma(c1,c2);
    c3.scrivi(); cout << endl;           // (6.4, 8)

    return 0;
}
```

C-36

16.11 Funzioni friend (III)

```
// Esempio di utilizzo della FRIEND nel caso di una classe
// che utilizza un'altra classe:
// CASO A) classe utilizzatrice FRIEND (per intero) della classe utilizzata
#include<iostream>
using namespace std;

class complesso{      // dichiarazione della classe utilizzata
double re, im;
public:
    complesso(double r = 0, double i = 0){re=r; im=i;}
/* seguono le altre funzioni membro della classe complesso... */
friend class vettoreComplesso; // FRIEND tutta la classe utilizzatrice
};

class vettoreComplesso // dichiarazione della classe utilizzatrice
{ complesso *vett;  int size;
public:
    vettoreComplesso(int sz){
        (sz <= 0) ? size = 10 : size = sz ;
        vett = new complesso[size]; // chiamata al costruttore di default
        // su ogni complesso del vettore
    }
    double moduloQuadro(){
        double mQ = 0;
        for (int i=0; i<size; i++){
            // essendo friend vettoreComplesso puo' accedere a .re e .im
            // senza dover chiamare le funzioni di accesso reale() e immag()
            mQ += vett[i].re*vett[i].re + vett[i].im*vett[i].im;
        }
        return mQ;
    }
}; // fine classe vettorecomplesso

int main(){
    vettoreComplesso v(5);
    cout<<"Modulo quadro di v: ";
    cout<<v.moduloQuadro()<<endl;
    return 0;
}
```

C-37

16.11 Funzioni friend (IV)

```
// Esempio di utilizzo della FRIEND nel caso di una classe
// che utilizza un'altra classe:
// CASO B) solo alcuni metodi della classe utilizzatrice
// sono FRIEND di quella utilizzata
#include<iostream>
using namespace std;

class complesso;      // dichiarazione incompleta della classe utilizzata
class vettoreComplesso // dichiarazione (completa) della classe utilizzatrice
{
    complesso *vett;    int size;
public:
    vettoreComplesso(int);
    double moduloQuadro();
};

class complesso{ // dichiarazione (completa) della classe utilizzata
double re, im;
public:
    complesso(double r = 0, double i = 0){re=r; im=i;}
    friend double vettoreComplesso::moduloQuadro(); // FRIEND un metodo solo
};

// definizione delle funzioni membro della classe utilizzatrice
vettoreComplesso::vettoreComplesso(int sz){
    size = sz;  vett = new complesso[size]; // chiamata al costruttore di default
}

double vettoreComplesso::moduloQuadro(){
    double mQ = 0;
    for (int i=0; i<size; i++){
        mQ += vett[i].re*vett[i].re + vett[i].im*vett[i].im;
    }
    return mQ;
}

int main(){
    vettoreComplesso v(5);  cout<<"Modulo quadro di v: ";
    cout<<v.moduloQuadro()<<endl;
    return 0;
}
```

C-38

17.1 Overloading di operatori (I)

- Tipo di dato astratto: può richiedere che vengano definite delle operazioni tipo somma, prodotto, etc.
- Opportuno utilizzare gli stessi operatori usati per le operazioni analoghe sui tipi fondamentali.
- Necessaria la ridefinizione degli operatori
- La ridefinizione di un operatore ha la forma di una definizione di funzione, il cui identificatore è costituito dalla parola chiave *operator* seguita dall'operatore che si vuole ridefinire, e ogni occorrenza dell'operatore equivale ad una chiamata alla funzione.
- Un operatore può designare una funzione membro o una funzione globale.

Se \odot è un operatore unario

1. Funzione membro *operator \odot ()*
2. Funzione globale *operator \odot (x)*

Se \odot è un operatore binario

1. Funzione membro *operator \odot (x)*
2. Funzione globale *operator \odot (x,y)*

ATTENZIONE: si possono usare solo operatori già definiti nel linguaggio e non se ne possono cambiare le proprietà

ATTENZIONE: una nuova definizione di un operatore deve avere fra gli argomenti almeno un argomento di un tipo definito dall'utente

17.1 Overloading di operatori (II)

Ridefinizione dell'operatore somma per i numeri complessi come funzione membro

```
//file complesso.h
class complesso
{
    double re, im;
public:
    complesso(double r = 0, double i = 0);
    complesso operator+(const complesso& );
    /*...*/
};

// file complesso.cpp
/*...*/
complesso complesso::operator+(const complesso& x)
{
    complesso z(re+x.re,im+x.im);
    return z;
}

// file main.cpp
#include<iostream>
#include "complesso.h"
int main()
{
    complesso c1 (3.2, 4);
    complesso c2 (c1);
    complesso c3 = c1 + c2;
    c3.scrivi(); cout << endl;
    return 0;
} // (6.4, 8)
```

17.1 Overloading di operatori (III)

Ridefinizione dell'operatore somma per i numeri complessi come funzione globale

```
//file complesso.h
class complesso
{
    double re, im;
public:
    complesso(double r = 0, double i = 0);
    friend complesso operator+(const complesso&,
                               const complesso&);
    /* ... */
};
// file complesso.cpp
/* ... */
complesso operator+(const complesso& x, const
                    complesso& y)
{
    complesso z(x.re+y.re, x.im+y.im);
    return z;
}
// file main.cpp
#include<iostream>
#include "complesso.h"
int main()
{
    complesso c1 (3.2, 4);
    complesso c2 (c1);
    complesso c3 = c1 + c2;
    c3.scrivi(); cout << endl;
    return 0;
}
// (6.4, 8)
```

C-41

17.1 Overloading di operatori (IV)

Operatori incremento

```
//file complesso.h
class complesso
{
    double re, im;
public:
    complesso& operator++();
    complesso operator++(int);
    //argomento fittizio che individua l'operatore postfisso
    /* ... */
};
// file complesso.cpp
/* ... */
complesso& complesso::operator++()
{
    re++; im++;
    return *this;
}
complesso complesso::operator++(int)
{
    complesso app = *this;
    re++; im++;
    return app;
}
```

C-42

17.1 Overloading di operatori (V)

Operatori incremento

```
// file main.cpp
#include<iostream>
#include "complesso.h"
int main()
{
    complesso c1 (3.2, 4);
    complesso c2 (c1);
    complesso c3 = c1 + c2;
    c3.scrivi(); cout << endl;
    c1++;
    c1.scrivi(); cout << endl;
    ++c2;
    c2.scrivi(); cout << endl;
    c1+++++++;
    c1.scrivi(); cout << endl;
    ++++c2;
    c2.scrivi(); cout << endl;

    return 0;
}
```

// (6.4, 8)

// (4.2, 5)

// (4.2, 5)

// (5.2, 6)

// (7.2, 8)

17.1 Overloading di operatori (VI)

Tutti gli operatori di assegnamento si ridefiniscono come funzioni membro e restituiscono un riferimento all'oggetto

Nella scelta fra le diverse versioni di una funzione sovrapposta (o di un operatore sovrapposto), il compilatore tiene conto delle possibili conversioni di tipo: queste possono essere conversioni fra tipi fondamentali (implicite o indicate dal programmatore), o conversioni definite dal programmatore che coinvolgono tipi classe.

Non ci sono priorità fra le possibili conversioni. Se esistono diverse possibilità, il compilatore segnala questa ambiguità.

ATTENZIONE: non assegnate ad un operatore un significato non intuitivo

ATTENZIONE: le versioni predefinite degli operatori assicurano alcune equivalenze. Per esempio, `++x` è equivalente a `x+=1`. Quando si ridefinisce un operatore, le equivalenze tra gli operatori non valgono automaticamente. Sarà compito del programmatore fare in modo che tali equivalenze vengano rispettate.

17.1 Overloading di operatori (VII)

Ambiguità non risolta dal compilatore.

```
//file complesso.h
class complesso
{ double re, im;
public:
    complesso(double r = 0, double i = 0);
    complesso operator+(const complesso& );
    operator double(){ return re+im;}
/*...*/
};

// file complesso.cpp
/*...*/
complesso complesso::operator+(const complesso& x)
{
    complesso z(re+x.re,im+x.im);
    return z;
}

// file main.cpp
#include<iostream>
#include "complesso.h"
int main()
{
    complesso c1 (3.2, 4), c2;
    cout << double(c1) << endl;    // 7.2
    c2 = c1 + 3.0;
    // ERRORE: ambiguous overload for 'operator+' in 'c1 +
    // 3.0e+0' candidates are:
    // operator+(double, double) <built-in>
    // complesso complesso::operator+(const complesso&)

    return 0;
}
```

C-45

17.2 Simmetria tra gli operatori (I)

Gli operatori ridefiniti come funzioni membro non sono simmetrici.

```
//file complesso.h
class complesso
{ double re, im;
public:
    complesso(double r = 0, double i = 0);
    complesso operator+(const complesso& );
/*...*/
};

// file complesso.cpp
/*...*/
complesso complesso::operator+(const complesso& x)
{
    complesso z(re+x.re,im+x.im);
    return z;
}

// file main.cpp
#include<iostream>
#include "complesso.h"
int main()
{
    complesso c1 (3.2, 4), c2;
    c2 = c1 + 3.0;
    // chiamata implicita al costruttore prefinito con
    // argomento attuale 3. Somma tra numeri complessi
    c2 = 3.0 + c1;    // ERRORE

    return 0;
}
```

C-46

17.2 Simmetria tra gli operatori (II)

Per mantenere la simmetria degli operatori (il primo operando possa essere qualsivoglia e non l'oggetto a cui si applica l'operatore) si devono utilizzare funzioni globali.

```
//file complesso.h
class complesso
{
    double re, im;
public:
    complesso(double r = 0, double i = 0);
    friend complesso operator+(const complesso& x,
                              const complesso& y);
    friend complesso operator+(const complesso& x,
                              double d);
    friend complesso operator+(double d, const
                              complesso& y);
}

/* .../
};

// file complesso.cpp
/* .../
complesso operator+(const complesso& x,
                    const complesso& y)
{
    complesso z(x.re+y.re,x.im+y.im);
    return z;
}

complesso operator+(const complesso& x,double d)
{
    complesso z(x.re+d,x.im);
    return z;
}
```

C-47

17.2 Simmetria tra gli operatori (III)

```
// file complesso.cpp
complesso operator+(double d, const complesso& x)
{
    complesso z(x.re+d,x.im+d);
    return z;
}

// file main.cpp
#include<iostream>
#include "complesso.h"
int main()
{
    complesso c1 (3.2, 4);
    complesso c2 = c1 + 4.0;
    c2.scrivi(); cout << endl;
    c2 = 3.0 + c1;
    c2.scrivi(); cout << endl;
    c2 = 4.0 + 3.0;
    c2.scrivi(); cout << endl;

    return 0;
}

// (7.2, 8)
// (6.2, 7)
// Costruttore prefinito
// (7, 0)
```

C-48

17.3 Operatore di assegnamento (I)

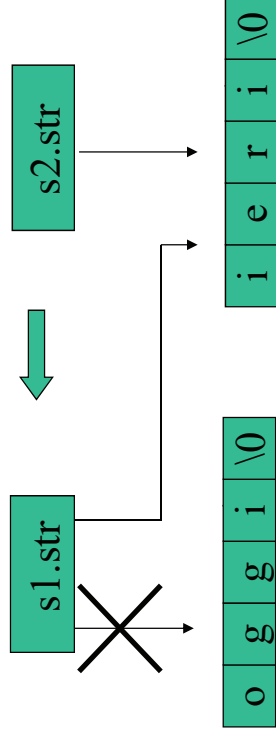
Operatore di assegnamento: predefinito nel linguaggio. Esegue una copia membro a membro. Deve essere ridefinito in presenza di allocazione dinamica della memoria.

```
// file stringa.h
class stringa
{ char* str;
public:
    stringa(const char s[]);
    ~stringa();
    stringa(const stringa &); // costruttore di copia
// ...
};

// file main.cpp
void fun()
{
    stringa s1("oggi");
    stringa s2("ieri");
    s1 = s2;
// ...
}

// operatore di assegnamento
```

// Al termine della funzione viene richiamato il distruttore due volte. Comportamento dell'operatore delete quando applicato ad un'area già liberata non definito!!!



C-49

17.3 Operatore di assegnamento (II)

Operatore di assegnamento deve:

1. deallocare la memoria dinamica dell'operando a sinistra
2. allocare la memoria della dimensione uguale all'operando destro
3. copiare i membri dato e gli elementi dello heap

PROBLEMA DI ALIASING: argomento implicito uguale all'argomento esplicito
s1 = s1

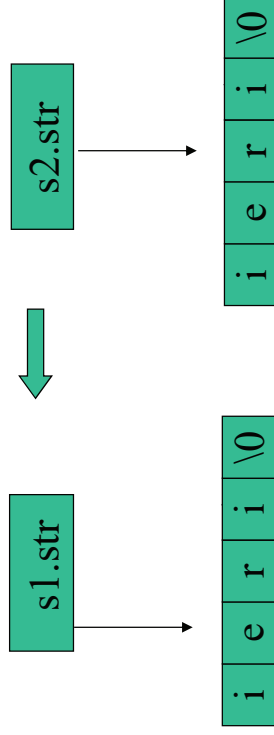
```
// file stringa.h
class stringa
{ char* str;
public:
    stringa(const char s[]);
    ~stringa(); // distruttore
    stringa(const stringa &); // costruttore di copia
    stringa& operator=(const stringa&); // oper. ass
// ...
};
```

```
// file stringa.cpp NB: l'oggetto di sinistra (sx) è l'oggetto (*this)
stringa& stringa::operator=(const stringa& dx)
{ if (this != &dx) // CONTROLLO ALIASING
{
    delete[] str;
    str = new char[strlen(dx.str)+1];
    strcpy(str, dx.str);
}
return *this;
}

// NB: il test poteva anche essere scritto: if ( &(*this) != &dx ) ...
```

C-50

17.3 Operatore di assegnamento (III)



OTTIMIZZAZIONE:

Se la memoria dinamica ha la giusta dimensione possiamo evitare di deallocare e riallocare la memoria

```
// file stringa.cpp
stringa& stringa::operator=(const stringa& dx)
{ if (this != &dx)
  {
    if (strlen(str) != strlen(dx.str))
    {
      delete[] str;
      str = new char[strlen(dx.str)+1];
    }
    strcpy(str, dx.str);
  }
  return *this;
}
```

C-51

17.3 Operatori che si possono ridefinire (I)

Si possono ridefinire tutti gli operatori tranne:

- L'operatore risoluzione di visibilità (::)
- L'operatore selezione di membro (.)
- L'operatore selezione di membro attraverso un puntatore a membro (.*)

ATTENZIONE: gli operatori di assegnamento ('='), di indicizzazione ('[]'), di chiamata di funzione ('()') e di selezione di membro tramite un puntatore ('->') devono essere ridefiniti sempre come funzioni membro.

ATTENZIONE: oltre all'operatore di assegnamento, sono predefiniti anche quello di indirizzo ('&') e di sequenza (';').

C-52

18.1 Costanti e riferimenti nelle classi (I)

Il campo dati di una classe può avere l'attributo `const`: in questo caso il campo deve essere inizializzato nel momento in cui viene definito un oggetto appartenente alla classe stessa

INIZIALIZZAZIONE: avviene tramite la lista di inizializzazione del costruttore.

```
// file complesso.h
class complesso
{   const double re;
    double im;
public:
    complesso(double r = 0, double i = 0);
    complesso operator+(const complesso&);
    /* ... */
};
```

```
// file complesso.cpp
complesso::complesso(double r, double i) : re(r)
{ im = i;}
// anche permessa la scrittura seguente
// complesso::complesso(double r, double i) : re(r), im(i) {}

complesso :: complesso(const complesso& c) : re(c.re)
{ im = c.im;}
```

ATTENZIONE: in una classe si possono dichiarare riferimenti non inizializzati, purché siano inizializzati con la lista di inizializzazione

C-53

18.2 Membro classe all'interno di classi (I)

In una classe (classe principale) possono essere presenti membri di tipo classe (classe secondaria) diversa dalla classe principale.

```
// file record.h
class record           // classe principale
{
    stringa nome, cognome;
public:
    //....
};
```

Quando viene dichiarato un oggetto appartenente alla classe principale:

1. vengono richiamati i costruttori delle classi secondarie, se definiti, nell'ordine in cui queste compaiono nella dichiarazione dei membri della classe principale;
2. quindi, viene eseguito il corpo del costruttore della classe principale, se definito.

Quando un oggetto appartenente alla classe principale viene distrutto:

1. viene eseguito il corpo del distruttore della classe principale, se definito;
2. Quindi, vengono richiamati i distruttori delle classi secondarie, se definiti, nell'ordine inverso in cui queste compaiono nella dichiarazione dei membri della classe principale.

C-54

18.2 Membro classe all'interno di classi (II)

Se alcune classi secondarie possiedono costruttori con argomenti formali, anche per la classe principale deve essere definito un costruttore e questo deve prevedere una lista di inizializzazione

```
// file record.h
#include "stringa.h"
class record // classe principale
{
    stringa nome, cognome;
public:
    record(const char n[], const char c[]);
    // ...
};

// file record.cpp
record::record(const char n[], const char c[])
: nome(n), cognome(c)
{/*....*/}
//...

// file main.cpp
#include "record.h"
int main()
{
    record pers("Mario", "Rossi");
    //...
}
```

C-55

18.2 Membro classe all'interno di classi (III)

Se alcune classi secondarie prevedono costruttori default, questi vengono richiamati.

Se tutte le classi secondarie hanno costruttori default, anche quello della classe principale può essere un costruttore default o mancare (viene usato il costruttore predefinito).

```
// file stringa.h
class stringa
{ char* str;
public:
    stringa(const char s[]="");
    //...
};

// file record.h
#include "stringa.h"
class record // classe principale
{
    stringa nome, cognome;
public:
    // ...
};

// file main.cpp
#include "record.h"
int main()
{
    record pers;
    //...
}
```

C-56

18.3 Array di oggetti classe (I)

Un array può avere come elementi oggetti classe: se nella classe sono definiti costruttore e distruttore, questi vengono richiamati per ogni elemento dell'array

```
//file complesso.h
class complesso
{
    double re, im;
public:
    complesso(double r = 0, double i = 0);
    //....
};

// file complesso.cpp
complesso::complesso(double r, double i)
{re = r; im = i;}
/* */

// file main.cpp
#include<iostream>
#include "complesso.h"
using namespace std;
int main()
{
    complesso vc[3];
    for (int i=0; i<3; i++)
        vc[i].scrivi(); cout << endl;
    return 0;
}
```

```
(0, 0)
(0, 0)
(0, 0)
```

C-57

18.3 Array di oggetti classe (II)

Inizializzazione esplicita

```
// file main.cpp
#include<iostream>
#include "complesso.h"
using namespace std;
int main(){
    complesso vc[5] = {complesso(1.1,2.2),
        complesso(1.5,1.5), complesso(4.1,2.5)};
    for (int i=0; i<5; i++)
        { vc[i].scrivi(); cout << endl;}

    return 0;
}
```

```
(1.1, 2.2)
(1.5, 1.5)
(4.1, 2.5)
(0, 0)
(0, 0)
```

ATTENZIONE: se la lista di costruttori non è completa, nella classe deve essere definito un costruttore default, che viene richiamato per gli elementi non esplicitamente inizializzati.

A partire da C++14 è ammessa anche questa sintassi:

```
string* vettDinamicoDiStringhe = new string[3]{
    stringa("Fondamenti "), stringa("di "),
    stringa("Programmazione") } ;
```

C-58

18.4 Membri statici (I)

Membri dati statici: modellano informazioni globali, appartenenti alla classe nel suo complesso e non alle singole istanze.

```
// file entity.h
class entity{
    int dato;
public:
    // ...
    entity(int n);           //dichiarazione della var. conto
    static int conto;
};

// file entity.cpp
#include "entity.h"
int entity::conto = 0;      //definizione della var. conto

entity::entity(int n)
{ conto++; dato = n; }

// file main.cpp

#include<iostream>
#include "entity.h"
using namespace std;
int main(){
    entity e1(1), e2(2);
    cout << "Numero istanze: " << entity::conto << endl;
    return 0;
}
```

Numero istanze: 2

C-59

18.4 Membri statici (II)

Funzione membro statica: operazione che non viene applicata ad una singola istanza della classe, ma svolge elaborazioni sulla classe in quanto tale.

```
// file entity1.h
class entity1
{ int dato;
  static int conto;
public:
    // ...
    entity1(int n);
    static int numero();    // Funzione membro statica
};

// file entity1.cpp
#include "entity1.h"
int entity1::conto = 0;

entity1::entity1(int n)
{ conto++; dato = n; }

int entity1::numero()
{ return conto;}

// file main.cpp
// ...
int main(){
    entity1 e1(1), e2(2);
    cout << "Num. istanze: " << entity1::numero() << endl;
    cout << "Num. istanze: " << e1.numero() << endl;
    return 0;
}
```

Num. istanze: 2
Num. istanze: 2

C-60

18.4 Membri statici (III)

Una funzione membro statica non può accedere implicitamente a nessun membro non statico né usare il puntatore this.

```
// file entity2.h
class entity2
{ int dato;
  int ident;
  static int conto;
public:
  // ...
  entity2(int n);
  static int numero(); // Funzione membro statica
  static void avanzaConto(int n);
};

// file entity2.cpp
#include "entity2.h"
int entity2::conto = 0;

entity2::entity2(int n)
{ conto++; dato = n; ident = conto;}

int entity2::numero()
{ return conto;}

void entity2::avanzaConto(int n)
{
  conto += n;
  ident = conto++;
  // ERRORE: invalid use of member `entity2::ident' in
  // static member function
}
```

C-61

18.5 Funzioni const (I)

Funzione membro const: funzioni che non possono modificare l'oggetto a cui sono applicate. Importanti perché ad oggetti classe costanti possono essere applicate solo funzioni membro costanti.

```
// file complesso.h
class complesso
{
  double re, im;
public:
  complesso(double r = 0, double i = 0);
  double reale();
  double immag();
  //...
};

// file main.cpp
#include <iostream>
#include "complesso.h"
using namespace std;
int main()
{
  const complesso c1 (3.2, 4);
  cout << "Parte Reale " << c1.reale() << endl;
  //ERRORE: c1 e' costante

  return 0;
}
```

C-62

18.5 Funzioni const (II)

```
// file complesso.h
class complesso
{
    double re, im;
public:
    complesso(double r = 0, double i = 0);
    double reale() const;
    double immag() const;
    //...
};

// file complesso.cpp
double complesso::reale() const {return re;}
double complesso::immag() const {return im;}
//...

// file main.cpp
#include<iostream>
#include "complesso.h"
using namespace std;
int main()
{
    const complesso c1 (3.2, 4);
    cout << "Parte Reale " << c1.reale() << endl;
    return 0;
}
```

Parte Reale 3.2

C-63

18.5 Funzioni const (III)

Il meccanismo di overloading distingue due versioni di una funzione che differiscono solo per la proprietà di essere costanti

```
class complesso
{
    double re, im;
public:
    complesso(double r = 0, double i = 0):re(r),im(i){};
    double reale() const;
    double reale();
    //...
};

// file complesso.cpp
double complesso::reale() const
{cout << "[ chiamata a complesso::reale()const ]" << endl;
return re;}

double complesso::reale()
{cout << "[ chiamata a complesso::reale() ]" << endl; return re;}

// file main.cpp
int main()
{
    const complesso c1 (3.2, 4);
    complesso c2(7.8, 5);
    cout<< c1.reale() << endl;
    cout<< c2.reale() << endl;
    return 0;
}
//...    NOTARE L'USCITA
```

C-64

18.7 Espressioni letterali

Non è possibile definire espressioni letterali per i tipi definiti dall'utente. I costruttori possono comunque svolgerne la funzione

```
// file complesso.h
class complesso
{
    double re, im;
public:
    complesso(double r = 0, double i = 0);
    complesso operator+(const complesso& x); //...
};

// file complesso.cpp
complesso complesso::operator+(const complesso& x)
{
    complesso z(re+x.re,im+x.im);
    return z;
}
//....

// file main.cpp
//...
int main()
{
    complesso c1 (3.2, 4), c2;
    c2 = complesso(0.3,3.0) + c1;
    c2.scrivi(); cout << endl;
    //...
}
```

(3.5, 7)

C-65

18.8 Conversioni mediante costruttori

Un costruttore che può essere chiamato con un solo argomento viene usato per convertire valori del tipo dell'argomento nel tipo della classe a cui appartiene il costruttore.

```
// file complesso.h
class complesso
{
    double re, im;
public:
    complesso(double r = 0, double i = 0);
    complesso operator+(const complesso& x); //...
};

// file main.cpp
//...
int main()
{
    complesso c1 (3.2, 4), c2;
    c2 = c1 + 2.5;
    // Il costruttore trasforma il reale 2.5 nel complesso (2.5,0)
    c2.scrivi(); cout << endl;
    //...
}
```

(5.7, 4)

C-66

18.8 Operatori di conversione

Operatore di conversione: converte un oggetto classe in un valore di un altro tipo.

```
// file complesso.h
class complesso
{
    double re, im;
public:
    operator double();
    operator int();
    //...
};

// file complesso.cpp
complesso::operator double()
{ return re+im;}

complesso::operator int()
{ return static_cast<int>(re+im);}

// file main.cpp
//...
int main()
{
    complesso c1 (3.2, 4);
    cout << (double)c1 << endl;
    cout << (int)c1 << endl;
    //...
}
```

7.2
7

C-67

20. Classi per l'ingresso e l'uscita (I)

Le librerie di ingresso/uscita forniscono classi per elaborare varie categorie di stream

Classe per l'ingresso dati

```
class istream
{
    // stato: configurazione di bit
public:
    istream(){...};
    istream& operator>>(int&);
    istream& operator>>(double&);
    istream& operator>>(char&);
    istream& operator>>(char* );
    int get();
    istream& get(char& c);
    istream& get(char* buf, int dim , char delim = '\n');
    istream& read(char* s, int n)
    //...
};
```

L'oggetto cin e un'istanza predefinita della classe istream.

- (1) La funzione *get()* preleva un carattere e lo restituisce convertito in intero
- (2) La funzione *get(c)* preleva un carattere e lo assegna alla variabile *c*
- (3) La funzione *get(buf,dim,delim)* legge caratteri dallo stream di ingresso e li trasferisce in *buf* aggiungendo il carattere nullo finale finché nel buffer non vi sono *dim* caratteri o non si incontra il carattere *delim* (che non viene letto).
- (4) La funzione *read(s,n)* legge *n* byte e li memorizza a partire dall'indirizzo contenuto in *s*.

C-68

20. Classi per l'ingresso e l'uscita (II)

Classe per l'uscita dati

```
class ostream
{ // stato: configurazione di bit
  // ...
public:
  ostream(){...};
  ostream& operator<<(int);
  ostream& operator<<(double);
  ostream& operator<<(char);
  ostream& operator<<(const char* );
  ostream& put(char c)
  ostream& write(const char* s, int n)
  ostream& operator<< ( ostream& ( *pf )(ostream&));
  //...
};
```

Gli oggetti `cout` e `cerr` sono istanze predefinite della classe `ostream`.

La funzione `put()` trasferisce il carattere `c` sullo stream di uscita

La funzione `write()` trasferisce la sequenza di n caratteri contenuti in `s` sullo stream di uscita.

20. Controllo del formato (I)

Controllo del formato: può avvenire attraverso i manipolatori

Manipolatori: puntatori a funzione.

Gli operatori di lettura e scrittura sono ridefiniti per `overloading` in modo da accettare tali puntatori.

```
ostream
endl                // ostream& endl ( ostream& os );
                    inserisce '\n' e svuota il buffer
cout << x << endl;
dec, hex, oct
                    selezionano la base di numerazione
cout << 16 << " " << oct << 16 << endl;
boolalpha
```

`insert` o `extract` oggetti di tipo booleano come nomi (`true` o `false`) piuttosto che come valori numerici

```
cout << boolalpha << true << endl;
```

`ios manip`

`setw(int)`

numero minimo di caratteri usati per la stampa di alcune rappresentazioni (w sta per *width*)

`setfill(char)`

specifica quale carattere prendere per arrivare alla lunghezza desiderata (default: spazio bianco).

`setprecision(int)`

numero di cifre significative (parte intera + parte decimale) per la stampa di numeri reali

I manipolatori hanno effetto fino alla nuova occorrenza del manipolatore; eccetto `setw()` che ha effetto solo sull'istruzione di scrittura immediatamente successiva

20. Controllo del formato (II)

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

int main(){
    double d = 1.564e-2;    int i;    bool b;
    cout << d << endl;
    cout << setprecision(2) << d << endl;
    cout << setw(10) << d << endl;
    cout << setfill('0') << setw(10) << d << endl;
    cin >> hex >> i; // supponendo di inserire 'b'
    cout << hex << i << 't' << dec << i << endl;
    cin >> oct >> i; // supponendo di inserire "12"
    cout << oct << i << 't' << dec << i << endl;
    cin >> boolalpha >> b; // supponendo di ins. "false"
    cout << b << boolalpha << 't' << b << endl;
    cout << true << endl;
    return 0;
}
```

```
0.01564
0.016
0.016
000000.016
b
b 11
12
12 10
false
0 false
true
```

C-71

20.2 Controlli sugli stream

Stato di uno stream: configurazione di bit, detti *bit di stato*. Lo stato è corretto (*good*) se tutti i bit di stato valgono 0.

- Bit *fail* – posto ad 1 ogniqualvolta si verifica un errore recuperabile
- Bit *bad* – discrimina se l'errore è recuperabile (0) o meno (1)
- Bit *eof* – posto ad 1 quando viene letta la marca di fine stream

I bit possono essere esaminati con le funzioni seguenti, che restituiscono valori booleani:

- *fail()* – restituisce *true* se almeno uno dei due bit *fail* e *bad* sono ad 1
- *bad()* – restituisce *true* se il bit *bad* è ad 1
- *eof()* – restituisce *true* se il bit *eof* è ad 1
- *good()* – restituisce *true* se tutti i bit sono a 0

I bit possono essere manipolati via software con la funzione *clear()*, che ha per argomento (di default è 0) un valore intero che rappresenta il nuovo valore dello stato.

Possono essere usati i seguenti enumeratori:

- *ios::failbit* – corrisponde al bit *fail* ad 1
- *ios::badbit* – corrisponde al bit *bad* ad 1
- *ios::eofbit* – corrisponde al bit *eof* ad 1

Ogni valore dello stato può essere ottenuto applicando l'operatore OR bit a bit con operandi tali enumeratori.

NOTA BENE: Quando uno stream viene posto in una condizione, viene restituito il complemento del risultato della funzione *fail()*.

C-72

20.3 Uso dei file

Le classi *fstream*, *ifstream* e *ofstream* hanno una struttura simile a quella vista per gli stream di ingresso/uscita.

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
    ifstream input("nonEsiste.txt"); char in;
    cout << input.fail() << " " << input.bad() << " "
        << input.eof() << " " << input.good() << endl;
    input.clear();
    cout << input.fail() << " " << input.bad() << " "
        << input.eof() << " " << input.good() << endl;
    input.clear(ios::badbit);
    cout << input.fail() << " " << input.bad() << " "
        << input.eof() << " " << input.good() << endl;
    input.clear();
    input.open("Esiste.txt");
    while (input>in);
    cout << input.fail() << " " << input.bad() << " "
        << input.eof() << " " << input.good() << endl;

    return 0;
}
```

```
1 0 0 0
0 0 0 1
1 1 0 0
1 0 1 0
```

C-73

20.4 Ridef. operatori ingresso/uscita(I)

Il meccanismo di overloading permette di ridefinire gli operatori di lettura e di scrittura per i tipi definiti dall'utente.

ATTENZIONE: Tali operatori devono essere ridefiniti come funzioni globali.

```
// file complesso.h

#include<iostream>
class complesso
{
    double re, im;
public:
    complesso(double r = 0, double i = 0){re=r;im=i;};
    double reale() const {return re;}
    double immag() const {return im;}
    //...
};

using namespace std;

ostream& operator<<(ostream& os, complesso z);
istream& operator>>(istream& is, complesso& z);
```

C-74

20.5 Ridef. operatori ingresso/uscita(II)

```
// file complesso.cpp
#include "complesso.h"

ostream& operator<<(ostream& os, const complesso& z)
{
    os << '(' << z.reale() << ',' << z.immag() << ')';
    return os;
}

istream& operator>>(istream& is, complesso& z)
{
    double re = 0, im = 0;
    char c = 0;
    is >> c;
    if (c != '(') is.clear(ios::failbit);
    else
    {
        is >> re >> c;
        if (c != ',') is.clear(ios::failbit);
        else
        {
            is >> im >> c;
            if (c != ')') is.clear(ios::failbit);
        }
        z = complesso(re, im);
        return is;
    }
}
```

C-75

20.5 Ridef. operatori ingresso/uscita(III)

```
#include <iostream>
#include "complesso.h"
using namespace std;

int main()
{
    complesso c1(1.0, 10.0);
    complesso c2(1.1, 10.1);
    cout << c1 << '\t' << c2 << endl;
    if (!(cin >> c2))
    {
        cerr << "Errore nella lettura di un numero
                complesso" << endl;
        exit(1);
    }
    cout << c1 << '\t' << c2 << endl;

    return 0;
}
```

```
(1,10) (1.1,10.1)
(2,12)
(1,10) (2,12)
```

C-76

21 Preprocessore (I)

Preprocessore: parte di compilatore che elabora il testo del programma prima dell'analisi lessicale e sintattica

- Può includere nel testo altri file
- Espande i simboli definiti dall'utente secondo le loro definizioni
- Include o esclude parti di codice dal testo che verrà compilato

Queste operazioni sono controllate dalle *direttive per il preprocessore* (il primo carattere è #)

Inclusione di file header. Due possibilità:

- Percorso a partire da cartelle standard
#include <file.h>
- Percorso a partire dalla cartella in cui avviene la compilazione o percorso assoluto
#include "file.h"

Esempio:

```
#include "subdir\file.h"  
#include "c:\subdir\file.h"
```

21 Preprocessore (II)

Definizione di simboli con associato un valore

```
#define KONST 42  
int main()  
{  
    int x = KONST;           // come scrivere int x = 42;  
    cout<<x<<endl;         // stampa 42  
}
```

NB: Non bisogna abusare di questa tecnica. In questo caso specifico, avrebbe avuto più senso utilizzare la seguente soluzione:

```
const int KONST = 42;
```

```
int main()  
{  
    int x = KONST;  
    cout<<x<<endl;  
}
```

in quanto in questo secondo caso KONST ha anche un tipo (tipo intero), e pertanto il compilatore può fare tutti i controlli sui tipi nelle espressioni in cui compare KONST.

Pertanto la define per la definizione di costanti va utilizzata con parsimonia e solo in presenza di ulteriori considerazioni che ne giustificano l'utilizzo.

21 Preprocessore (III)

Macro: Simbolo che viene sostituito con la sequenza di elementi lessicali corrispondenti alla sua definizione

```
#define MAX(A,B) ((A)>(B) ? (A) : (B))
int main(){
    int y;
    y = 9 + MAX(2, 5);    // y = 9 + ( 2) > (5) ? (2) : (5) );
    cout<<y<<endl;      // stampa 14, come ci si aspetta
}
```

ATTENZIONE però alle parentesi!!!!

Infatti, la macro definita come segue, senza parentesi:

```
#define MAX(A,B) (A)>(B) ? (A) : (B)
```

```
int main(){
    int y;
    y = 9 + MAX(2, 5);    // y = 9 + (2) > (5) ? (2) : (5)
    cout<<y<<endl;      // y = 2, non 14!!!!
}
```

farebbe sì che y alla fine valga 2 invece che 14:

```
int main(){
    int y;
    y = 9 + (2) > (5) ? (2) : (5);
    // di conseguenza y alla fine vale 2 perchè l'operatore +
    // ha priorità maggiore dell'operatore ternario, che a sua
    // volta ha priorità minore dell'assegnamento:
    // y = (9 + (2)) > (5) ? (2) : (5);
    // y = ( 11 > 5 ? 2 : 5 );
    // y = 2;
    cout<<y<<endl;      // stampa 2, non 14
}
```

C-79

21 Compilazione condizionale (I)

Compilazione condizionale **#if**, **#elif**, **#else**, **#endif**

```
#if    constant-expression
        (code if-part)
#elif constant-expression
        (code elif-parts)
#elif constant-expression
        (code elif-parts)
#else
        (code else-part)
#endif
```

I valori delle espressioni costanti vengono interpretati come valori logici ed in base a essi vengono compilati o no i frammenti testo (text) seguenti.

Esempio:

```
#define DEBUG_LEVEL 1 // all'inizio del file

int main(){
    #if DEBUG_LEVEL == 2
        cout<<i<<j; // debug di i e j
    #elif DEBUG_LEVEL == 1
        cout<<i;    // debug della sola variabile i
    #else
        // DEBUG_LEVEL == 0
        (nessuna cout)
    #endif
    return 0;
}
```

C-80

21 Compilazione condizionale (II)

FORME CONCISE

```
#ifdef identifier      per #if defined identifier
#endif identifier      per #if !defined identifier
```

ESEMPIO

```
// file main.cpp

#define LINUX          // commentare questa riga e
// #define WINDOWS    // scommentare questa
// per compilare sotto Windows
```

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

```
int main()
#ifdef LINUX    // equivale a '#if defined LINUX'
    system("CLEAR");
#elif defined WINDOWS
    system("CLS");
#else
    cout<<"Sistema operativo non supportato"<<endl;
    exit(1);
#endif
    return 0;
}
```

DEFINE A RIGA DI COMANDO

Alternativamente, gli identificatori per il processore si possono definire invece che in main.cpp direttamente alla chiamata del compilatore:

```
// Per compilare sotto LINUX
g++ -DLINUX main.cpp -o main.exe
// Per compilare sotto WINDOWS
g++ -DWINDOWS main.cpp -o main.exe
```

C-81

21 Compilazione condizionale (III)

Altro utilizzo della compilazione condizionale:

evitare che uno stesso file venga incluso più volte in una unità di compilazione. Ogni file di intestazione, per esempio *complesso.h*, dovrebbe iniziare con le seguenti direttive

```
-----
// file complesso.h
#ifndef COMPLESSO_H
#define COMPLESSO_H
// qui va il contenuto vero e proprio del file
class complesso{
    double re, im;
public:
    // funzioni della classe complesso
};
#endif
-----
```

```
// file main.cpp
#include "complesso.h"
#include <iostream>
#include "complesso.h"
```

```
int main(){
    complesso c;
    return 0;
}
```

l'aggiunta erronea di questo secondo include ora non causa più il problema di ridefinizione della classe *complesso*, perché la definizione della classe *complesso* viene inclusa solo la prima volta

C-82

Appendice A – Classe di mem. dei vettori

Caso 1: Vettore statico, avente lunghezza nota a

TEMPO DI COMPILAZIONE.

```
int vett1[10]; // vett1 deve essere def. fuori dai blocchi
```

Caso 2: Vettore automatico, avente lunghezza nota a

TEMPO DI COMPILAZIONE.

```
const int DIM = 10;
int main(){
    int vett2[D]; // vettore automatico (quello «classico»)
}
```

Caso 3: Vettore dinamico, avente lunghezza nota a

TEMPO DI ESECUZIONE.

```
int main(){
    int lun;
    cin>>lun;
    int *vett3 = new int[lun]; // vettore dinamico vero e proprio
}
```

Caso 4: Vettore automatico, avente lunghezza nota a

TEMPO DI ESECUZIONE.

```
int main(){
    int lun;
    cin>>lun;
    int vett4[lun];
}
```

Come potremmo chiamare vett4? Vettore dinamico? No! In quanto esso ha classe di memorizzazione automatica (ossia viene allocato sullo stack). Qualcuno chiama vett4 vettore «**semi-dinamico**», per distinguerlo sia da quello automatico classico, che da quello dinamico vero e proprio.

Appendice A – Classe di mem. dei vettori

NB: Dentro una classe posso definire solo vettori dinamici veri e propri, oppure vettori automatici PURCHE' AVENTI LUNGHEZZA NOTA A TEMPO DI COMPILAZIONE.

```
class Esempio1{
    int lun;
    int *vett; // questo è consentito, purchè ovviamente
              // il costruttore allochi memoria nello heap
public:
    ...
};
```

```
class Esempio2{
    int vett[10]; // questo è consentito
public:
    ...
};
```

Quello che non si può fare è usare, dentro una classe, un vettore semi-dinamico, ossia con lunghezza nota a tempo di esecuzione!

```
class Esempio3{
    int n;
    int vett[n]; // questo non è consentito
               // ed, infatti, non compila!
};
```

Appendice B – Priorità degli operatori (1/2)

Operatore (in grassetto)	Nome	Ass.
<code>class-name :: member</code> <code>namespace-name :: member</code>	risolutore di visibilità	s
<code>:: member</code>	risolutore globale di visibilità	d
<code>object . member</code>	selezione	s
<code>pointer -> member</code>	dereferenziazione e selezione	s
<code>array [rvalue]</code>	indicizzazione	s
<code>function (actual-argument-list)</code>	chiamata di funzione	s
<code>lvalue ++</code>	postincremento	d
<code>lvalue --</code>	postdecremento	d
<code>static_cast type <rvalue></code>	conversione di tipo	s
<code>const_cast type <rvalue></code>	conversione const	s
<code>sizeof object / sizeof (type)</code>	dimensione di oggetto o di tipo	d
<code>++ lvalue</code>	preincremento	d
<code>-- lvalue</code>	predecremento	d
<code>~ rvalue</code>	complemento bit a bit	d
<code>! rvalue</code>	negazione	d
<code>- rvalue</code>	meno unario	d
<code>+ rvalue</code>	più unario	d
<code>& lvalue</code>	indirizzo	d
<code>* rvalue</code>	dereferenziazione	d
<code>new type</code>	allocazione	d
<code>delete pointer</code>	deallocazione	d
<code>delete[] pointer</code>	deallocazione di array	d

C-85

Appendice B – Priorità degli operatori (2/2)

Operatore	Nome	Ass.
<code>object . * pointer-to-member</code>	selezione con punt. a membro	s
<code>pointer -> * pointer-to-member</code>	deref. e sel. con punt. a membro	s
<code>rvalue * rvalue</code>	moltiplicazione	s
<code>rvalue / rvalue</code>	quoziente	s
<code>rvalue % rvalue</code>	resto	s
<code>rvalue + rvalue</code>	somma	s
<code>rvalue - rvalue</code>	sottrazione	s
<code>rvalue << rvalue</code>	traslazione sinistra	s
<code>rvalue >> rvalue</code>	traslazione destra	s
<code>rvalue < rvalue</code>	minore	s
<code>rvalue <= rvalue</code>	minore o uguale	s
<code>rvalue > rvalue</code>	maggiore	s
<code>rvalue >= rvalue</code>	maggiore o uguale	s
<code>rvalue == rvalue</code>	uguale	s
<code>rvalue != rvalue</code>	diverso	s
<code>rvalue & rvalue</code>	AND bit a bit	s
<code>rvalue ^ rvalue</code>	OR esclusivo bit a bit	s
<code>rvalue rvalue</code>	OR bit a bit	s
<code>rvalue && rvalue</code>	AND logico	s
<code>rvalue rvalue</code>	OR logico	s
<code>rvalue ? rvalue : rvalue</code>	espressione condizionale	s
<code>lvalue = rvalue</code>	assegnamento	d
<code>lvalue += rvalue</code>	somma e assegnamento	d
<code>lvalue -= rvalue</code>	sottrazione e assegnamento	d
<code>lvalue *= rvalue</code>	moltiplicazione e assegnamento	d
<code>lvalue /= rvalue</code>	divisione e assegnamento	d
<code>lvalue %= rvalue</code>	resto e assegnamento	d
<code>lvalue &= rvalue</code>	AND bit a bit e assegnamento	d
<code>lvalue = rvalue</code>	OR bit a bit e assegnamento	d
<code>lvalue ^= rvalue</code>	OR esclusivo bit a bit e assegnamento	d
<code>lvalue <<= rvalue</code>	traslazione a sinistra e assegnamento	d
<code>lvalue >>= rvalue</code>	traslazione a destra e assegnamento	d
<code>expression , expression</code>	virgola	s

C-86

(fine delle slide)