Materiale didattico di supporto alle lezioni del corso di

Fondamenti di Programmazione

Docenti:
Marco Cococcioni (titolare del corso)
Carlo Puliafito
Nicola Messina
Jacopo Cecchetti

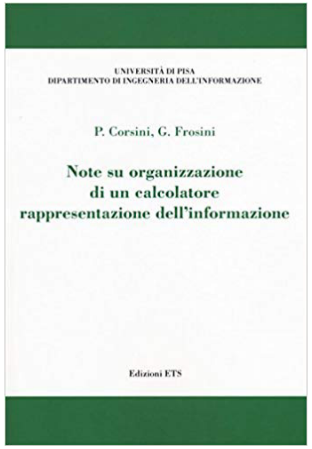
Corso di Laurea in Ingegneria Informatica
Dipartimento di Ingegneria dell'Informazione
Scuola di Ingegneria – Università di Pisa

Anno Accademico 2025-2026

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*)

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

Orario settimanale (**TEO**= teoria, **LAB**= laboratorio)

<table>
<tr><th></th><th>Lu</th><th>Ma</th><th>Me</th><th colspan="2">Gi</th><th>Ve</th></tr>
<tr><td>08:30-09:30</td><td></td><td></td><td rowspan="3">FdP TEO (3h)
F9</td><td colspan="2"></td><td></td></tr>
<tr><td>09:30-10:30</td><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td>10:30-11:30</td><td></td><td rowspan="2">FdP TEO (2h)
F9</td><td colspan="2"></td><td></td></tr>
<tr><td>11:30-12:30</td><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td>12:30-13:30</td><td></td><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td>13:30-14:30</td><td></td><td></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td>14:30-15:30</td><td></td><td></td><td></td><td colspan="2">FdP TEO (1h)
F9</td><td></td></tr>
<tr><td>15:30-16:30</td><td></td><td></td><td></td><td rowspan="2">FdP LAB (2h)
Gruppo A
A13</td><td rowspan="2">FdP LAB (2h)
Gruppo B
F9</td><td></td></tr>
<tr><td>16:30-17:30</td><td></td><td></td><td></td><td></td></tr>
</table>

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

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.

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

Calcolatore elettronico come risolutore di problemi

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.

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.

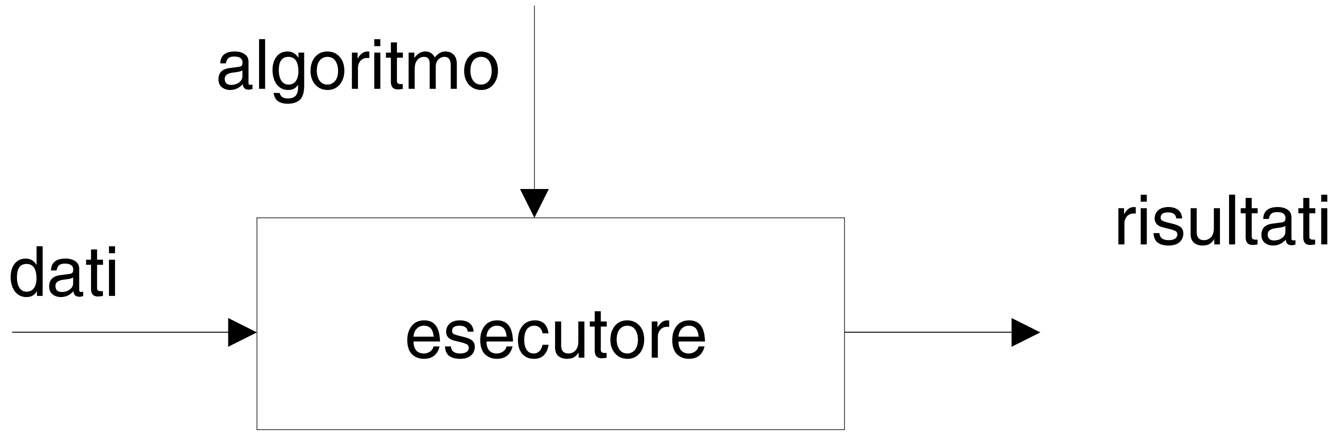
3. **Operazioni iterative**

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

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)

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*

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** togli da *P* e da *Q* la prima lettera
- **se** trovato vale 0 **allora** scrivi falso

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 corrispondeza degli stessi valori del dominio di ingressso 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

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

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: **MCD(m,n) = MCD((m-n),n)**

Algoritmo

- **ripeti finché** (*M != 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)

Algoritmo (8)

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.

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

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.

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 operazionale)
- mediante funzioni matematiche (semantica denotazionale)
- mediante formule logiche (semantica assiomatica)

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**

Grammatiche

Grammatica G = <V, VN, P, S>

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.

Grammatica BNF (1)

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

X

$A_1\ A_2\ ...\ A_n$

dove ***X*** è un simbolo non terminale ed $A_1\ A_2\ ...\ A_n$ 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 *riscrittura*)

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

<u>one of</u> // **<u>one of</u>** è un costrutto del metalinguaggio. Indica che ***X***
$A_1\ A_2\ ...\ A_n$ // può assunere **una unica alternativa** tra $A_1, \ldots, A_n$

Grammatica BNF (2)

G = <V, VN, P, S>

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

VN = {***frase***, ***soggetto***, ***verbo***, ***articolo***, ***nome*** }

S = ***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

<u>one of</u>

il lo

nome

<u>one of</u>

lupo canarino bosco cielo

verbo

<u>one of</u>

mangia vola canta

Esempio: derivazione della frase

"il lupo mangia."

frase -> ***soggetto verbo.***

-> ***articolo nome verbo.***

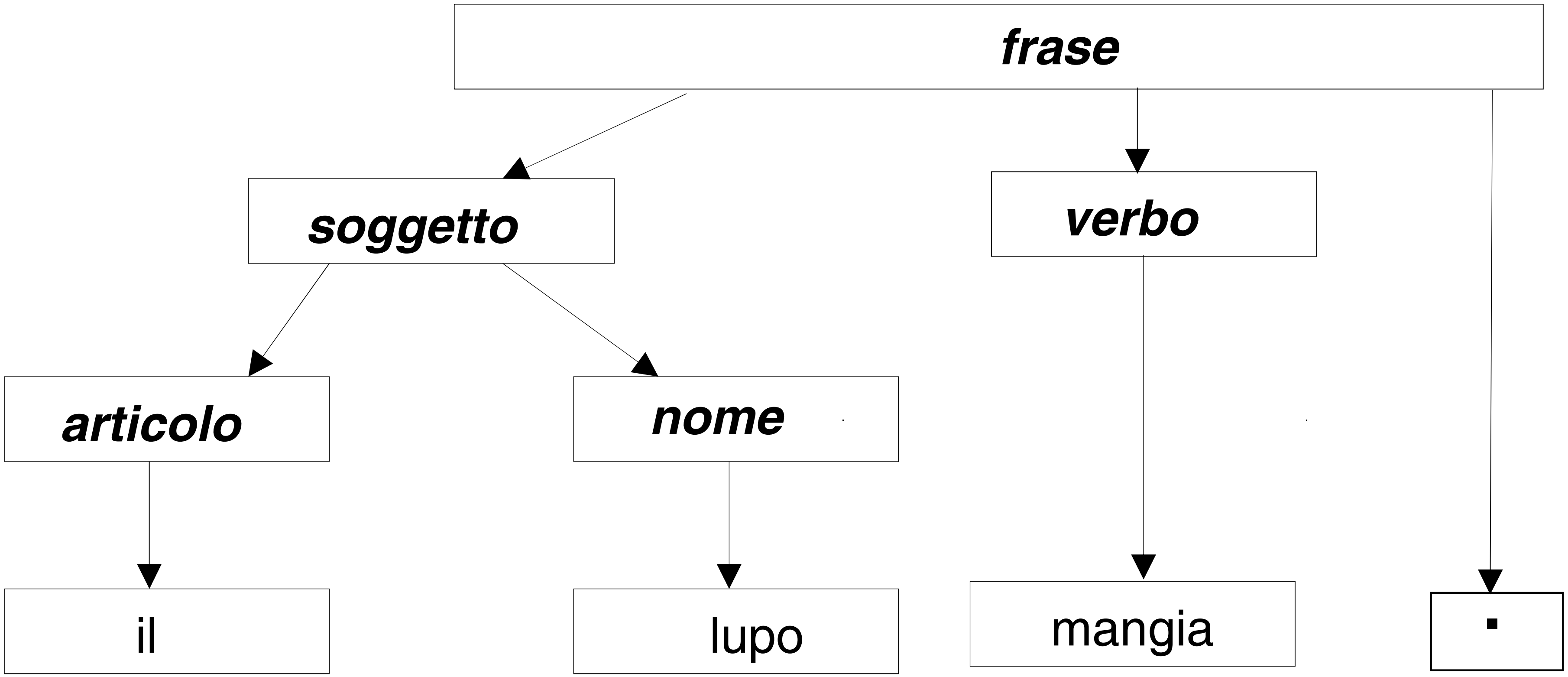
-> il ***nome verbo.***

-> il lupo ***verbo.***

-> il lupo mangia.

derivazione left-most

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.

Altri costrutti del matalinguaggio: 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**opt** 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**opt**

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

A_1 A_2

che come

A_1

Altri costrutti del matalinguaggio: 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

Pertanto

lupo canarino lupo lupo

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

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

nome-seq
nome nome-seq<u>**opt**</u>

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!

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

<u>one of</u>

0 1 2 3 4 5 6 7 8 9

cifra-seq

cifra cifra-seq**<u>opt</u>**

intero

-**<u>opt</u>** ***cifra-seq***

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

Esempi:

-23

4506

0023 // equivale a 23

-067 // equivale a -67

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

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

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!*

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
}
```

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++

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;
}
```

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, …

Approccio Interpretato

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

ESEMPI: Python, Matlab, Javascript, …

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

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 contentente 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)

Laboratorio di C++ basato su CLion

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 obbiamente 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**.

Rappresentazione dell'informazione

caratteri, naturali, interi e reali

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.

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 $\mathbf{2^n}$

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
- ...

dispositivi di conversione

testo
disegni
immagini
numeri
musica
...

IN →

OUT ←

sequenze di bit

Calcolatore

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   Caro amico,
01101001
01100011
01101111
00101100
```

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 catteri 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).

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

Data una base $\beta \geq$ **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).

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:

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

allora N può essere espresso come segue:

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

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

Chiamiamo questa formula «*<u>formula della sommatoria</u>*».

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

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:

$$2^p \leftrightarrow (1\underbrace{00\ldots00}_{p})_{due}$$

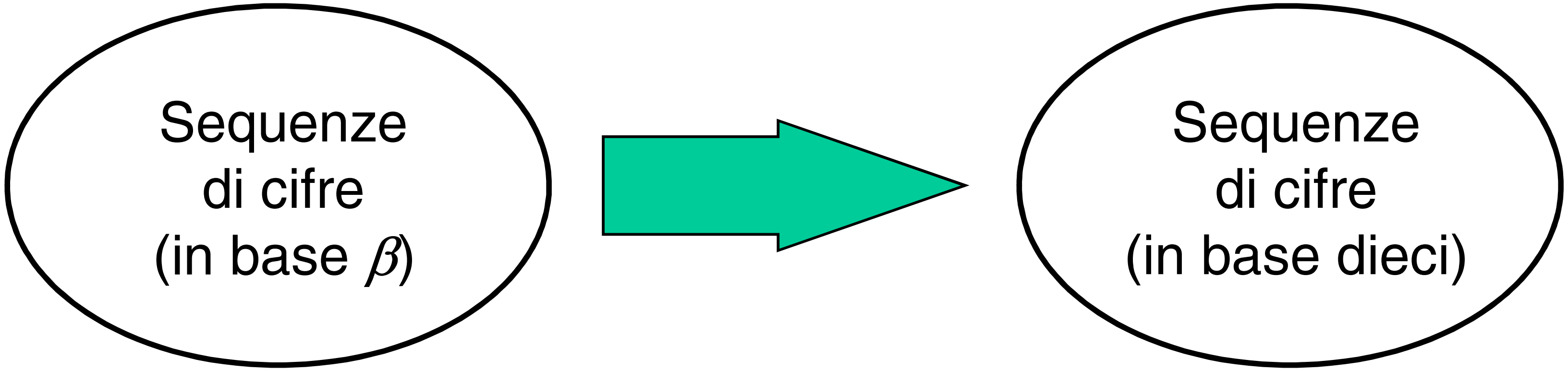
Esempio:

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*)

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



$$a_{p-1}\,a_{p-2}\,\ldots\,a_1\,a_0 \qquad\qquad N\,?$$

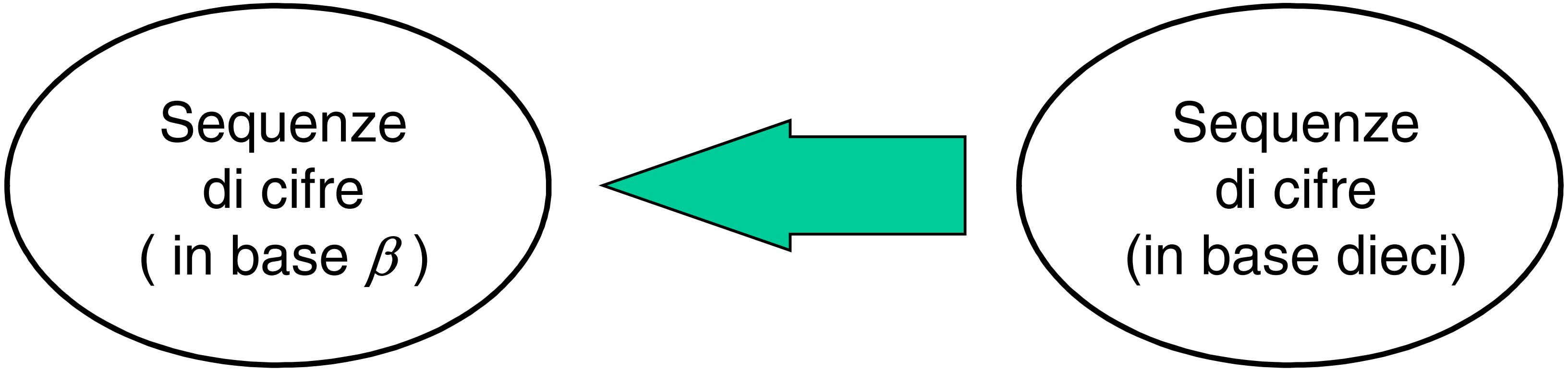
$$(656)_{otto} \rightarrow 6\cdot\boldsymbol{8}^2+5\cdot\boldsymbol{8}^1+6\cdot\boldsymbol{8}^0 \rightarrow (430)_{dieci}$$

$$(A03)_{sedici} \rightarrow 10\cdot\boldsymbol{16}^2+0\cdot\boldsymbol{16}^1+3\cdot\boldsymbol{16}^0 \rightarrow (2563)_{dieci}$$

$$(1101)_{due} \rightarrow 1\cdot\boldsymbol{2}^3+1\cdot\boldsymbol{2}^2+0\cdot\boldsymbol{2}^1+1\cdot\boldsymbol{2}^0 \rightarrow (13)_{dieci}$$

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



$a_{p-1} a_{p-2} \ldots a_1 a_0$ $\qquad$ N

Esempio: da base 10 a base 2

$N = 23$

inizio

div 2
div 2
div 2
div 2

QUOZIENTE	RESTO	Rappresentazione
23	-	
11	1	11*2+1
5	1	((5*2)+1)*2+1
2	1	((((2*2)+1)*2)+1)*2+1
1	0	(((((1*2)+0)*2)+1)*2)+1)*2+1
0	1	(((((0*2+1)*2+0)*2+1)*2+1)*2)+1

fine

$$(10111)_{due}$$

$a_4a_3a_2a_1a_0$

che infatti in base dieci vale proprio $1*2^4+\ldots+1=(23)_{dieci}$

(controprova, effettuata mediante algoritmo della sommatoria)

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

Esempio:
23 **div** 2 = 11
23 **mod** 2 = 1

Procedimento "Div & Mod"

Se $N = 0$ porre $a_0 = 0$;   => fine

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

$$q_1 = q_0 \text{ div } \beta \qquad a_0 = q_0 \text{ mod } \beta$$
$$q_2 = q_1 \text{ div } \beta \qquad a_1 = q_1 \text{ mod } \beta$$
$$\dots$$
$$q_p = q_{p-1} \text{ div } \beta \qquad a_{p-1} = q_{p-1} \text{ mod } \beta$$

NB: Il risultato della **mod** è sempre una cifra valida in base β, perché restituisce sempre un numero fra 0 e β-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

Inizio $q_0 = N$

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

div β

div β

div β

div β

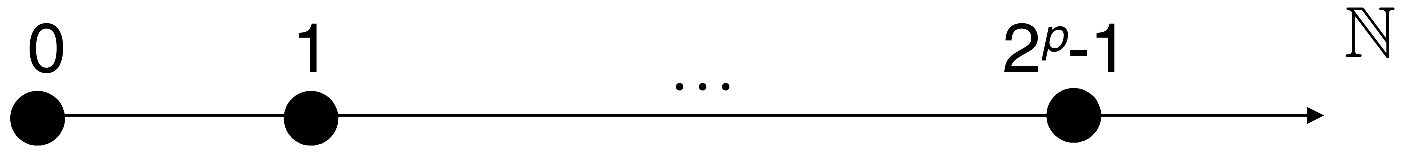
div β

div β

fine

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

Intervallo di rappresentabilità con *p* bit



$\underbrace{00\ldots00}_{p}$ $\underbrace{00\ldots01}_{p}$ $\underbrace{11\ldots11}_{p}$ $(= 2^p\text{-}1)$

p	**Intervallo** $[0, 2^p\text{-}1]$
8	[0, 255]
16	[0, 65535]
32	[0, 4294967295]

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

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)

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:

0 + 0 =	0 + 1 =	1 + 0 =	1 + 1 =	1 + 1 =
0	1	1	0	1
			1 (genera un riporto)	11 (genera un riporto; eventuale riporto generatosi precedentemente)

genera un riporto

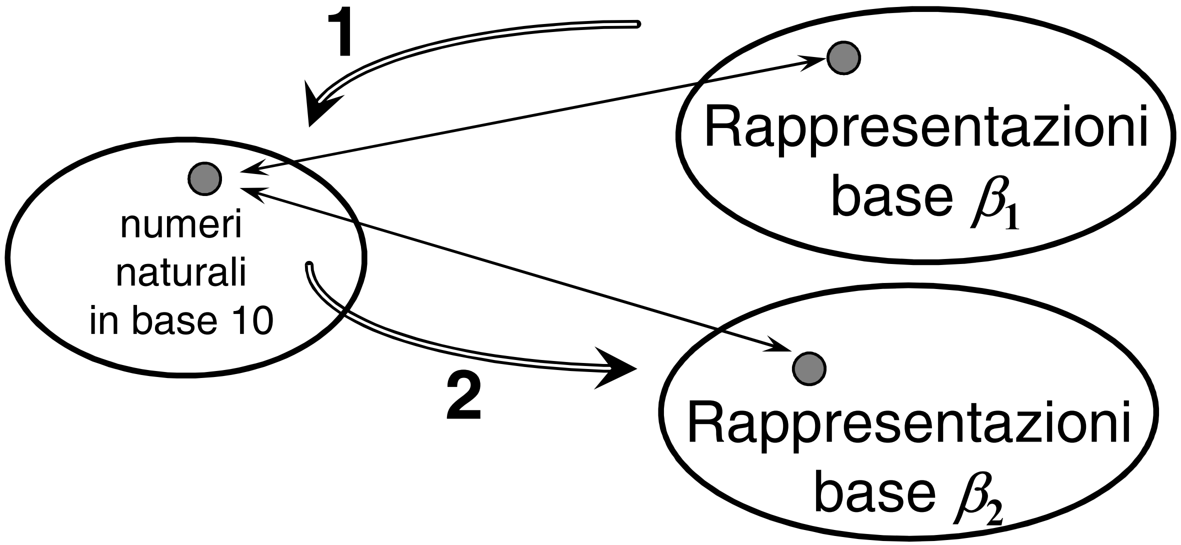
genera un riporto

eventuale riporto generatosi precedentemente

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

Esempio: calcolare la somma di 101100 e 111010

```
111 1     ← riporti generati
 101011 +
 111010 =
--------
1100101
```



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 $(163)_{nove}$

Casi particolari (potenze di 2)

$\beta = 8$

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

$$\beta=8 \rightleftarrows \beta=2 \rightleftarrows \beta=16$$

$\beta = 16$

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

$(657)_{otto}$

$(\ 110\ 101\ 111\ )_{due}$

$(A03)_{sedici}$

$(1010\ 0000\ 0011)_{due}$

$(FFE4)_{sedici}$

$(1111\ 1111\ 1110\ 0100)_{due}$

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 nature $2^3+2^7+2^5$ e sono interessato a conoscerne la rappresentazione proprio in base due. Certamente potrei utilizzare la tecnica generale, e calolare innanzitutto $N = 8+128+32 = (168)_{dieci}$
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 $2^{\mathbf{3}}+2^7+2^5+2^{\mathbf{3}}$, non avrei altra scelta che applicare il metodo generale!

Rappr. Naturali – Nel linguaggio C/C++ (16/16)

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).

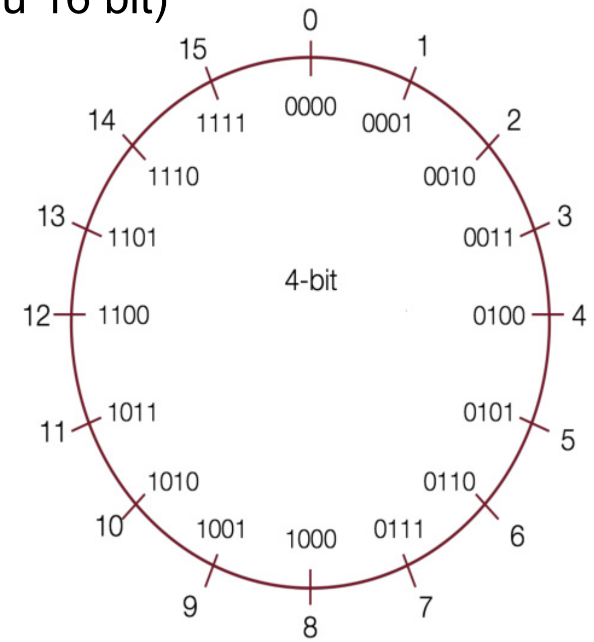
```
int main(){

    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****.*

Prima rappresentazione: *rappresentazione in modulo e segno*

Trasformazione a => A (codifica)

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

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

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 $a_{p-2},\ldots,a_0$),
e *segno_a* è un unico bit che vale 0 se **a** >= 0 e 1 se **a** <= 0).
NB: ad **a**=0 corrispondono due rappresentazioni: +*zero* (0,00...0) e -*zero* (1,00...0)

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

a=+3 => *segno_a* = 0, *ABS_a* = 011 (naturale 3 rappresentato su 3 cifre) => **A**=0011
a= -3 => *segno_a* = 1, *ABS_a* = 011 (naturale 3 rappresentato su 3 cifre) => **A**=1011

Prima rappresentazione: *rappresentazione in modulo e segno*

Trasformazione A => 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**=$(\underbrace{a_{p-1}}, \underbrace{a_{p-2}\ldots a_0})$ dopodiché:

$$\mathbf{a} = (a_{p-1} == 0)\ ?\ +ABS\_a : -ABS\_a$$

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

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

A=0011 => viene visto come (0,011) => **a** = +011 (+*tre*)
A=1011 => viene visto come (1,011) => **a** = -011 (-*tre*)

NB: 0000 rappresenta +*zero*, mentre 1000 rappresenta –*zero*.

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 ⇔ bit più significativo della rappresentazione uguale a 1

(*zero* rappresentato due volte)

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**=>**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 => 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:

$$\mathbf{A} = a_{p-1} \ldots a_0 = (\mathbf{a} >= 0)\ ?\ ABS\_a : (due^p - ABS\_a)$$

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

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

a = 0 => *ABS_a* = 0, e il naturale 0 ha rappr. 0000 su 4 bit => **A** = 0000
a = 1 => *ABS_a* = 1, e il naturale 1 ha rappr. 0001 su 4 bit => **A** = 0001
a = 7 => *ABS_a* = 7, e il naturale 7 ha rappr. 0111 su 4 bit => **A** = 0111
a = -1 => *ABS_a* = 1, 16-1=15, dove il naturale 15 ha rappr. 1111 su 4 bit => **A** = 1111
a = -2 => *ABS_a* = 2, 16-2=14, dove il naturale 14 ha rappr. 1110 su 4 bit => **A** = 1110
a = -8 => *ABS_a* = 8, 16-8=8, dove il naturale 8 ha rappr. 1000 su 4 bit => **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 => a (decodifica)

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

$$\mathbf{a} = (a_{p-1} == 0)\ ?\ +\mathbf{A} : -(due^{p}\text{-}\mathbf{A})$$

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

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

A = 0000 => **a** = 0 (*zero*)
A = 0001 => **a** = +1 (+*uno*)
A = 0111 => **a** = +7 (+*sette*)
A = 1000 => 16-8 = 8 => **a** = -8 (-*otto*)
A = 1001 => 16-9 = 7 => **a** = -7 (-*sette*)
A = 1111 => 16-15=1 => **a** = -1 (-*uno*)

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 ⇔ 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)

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**=>**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=(due^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 (due^5-9) = 23 $\Rightarrow$ **A**=10111

Numeri Interi (9/17)

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

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

A = 1111111111111111

Mentre -1 su *p* = 32 bit avrà rappresentazione **A** = $\underbrace{111111111 \ \ldots \ 1111111111}_{\text{32 bit ad uno}}$

Supponiamo di dover rappresentare **a** = -14 in complemento a due su cinque bit. Osservo che è rientra nell'intervallo di rappresentabilità ed applico l'algorimo di codifica: due^5-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).

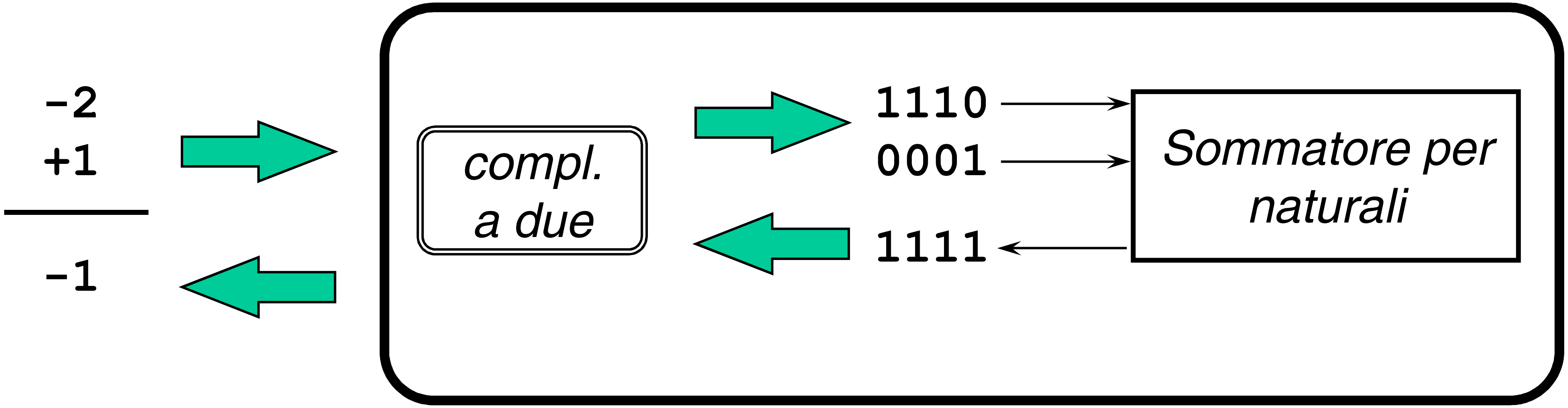
Esempio:
Passo 1: Applico Div&Mod a +14 e ottengo 1110. Estendo su cinque bit: 01110
Passo 2: Inverto i bit e ottengo 10001
Passo 3: Ci sommo uno:

```
10001+
    1=
----------
10010
```

Si può dimostrare che i due metodi sono equivalenti.

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

(due^p-1)-ABS_a



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!***

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)
------------------------------
  1 1 0 0    (-4)  overflow!
```

-6 -8 = -14

```
  1 0 1 0 +  (-6)
  1 0 0 0 =  (-8)
------------------------------
1 0 0 1 0    (+2)  overflow!
```

7-1=6

```
  0 1 1 1 +  (+7)
  1 1 1 1 =  (-1)
------------------------------
1 0 1 1 0    (+6) corretto!
```

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

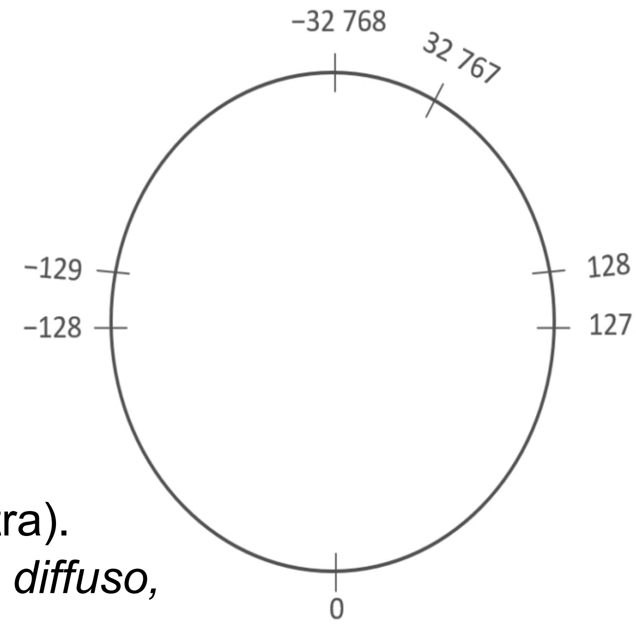
Numeri Interi (13/17)

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****.*

Terza Rappresentazione: ***rappresentazione con bias***

Trasformazione a => 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:

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

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 +** ***bias***

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

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

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

Terza Rappresentazione: ***rappresentazione con bias***

Trasformazione A => 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:

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

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

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

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

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

Intervallo di rappresentabilità nella *rappresentazione con bias*

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

- ***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** ⇒ **A** occorre verificare che **a** sia rappresentabile su *p* bit, altrimenti l'algoritmo conduce a risultati sbagliati.

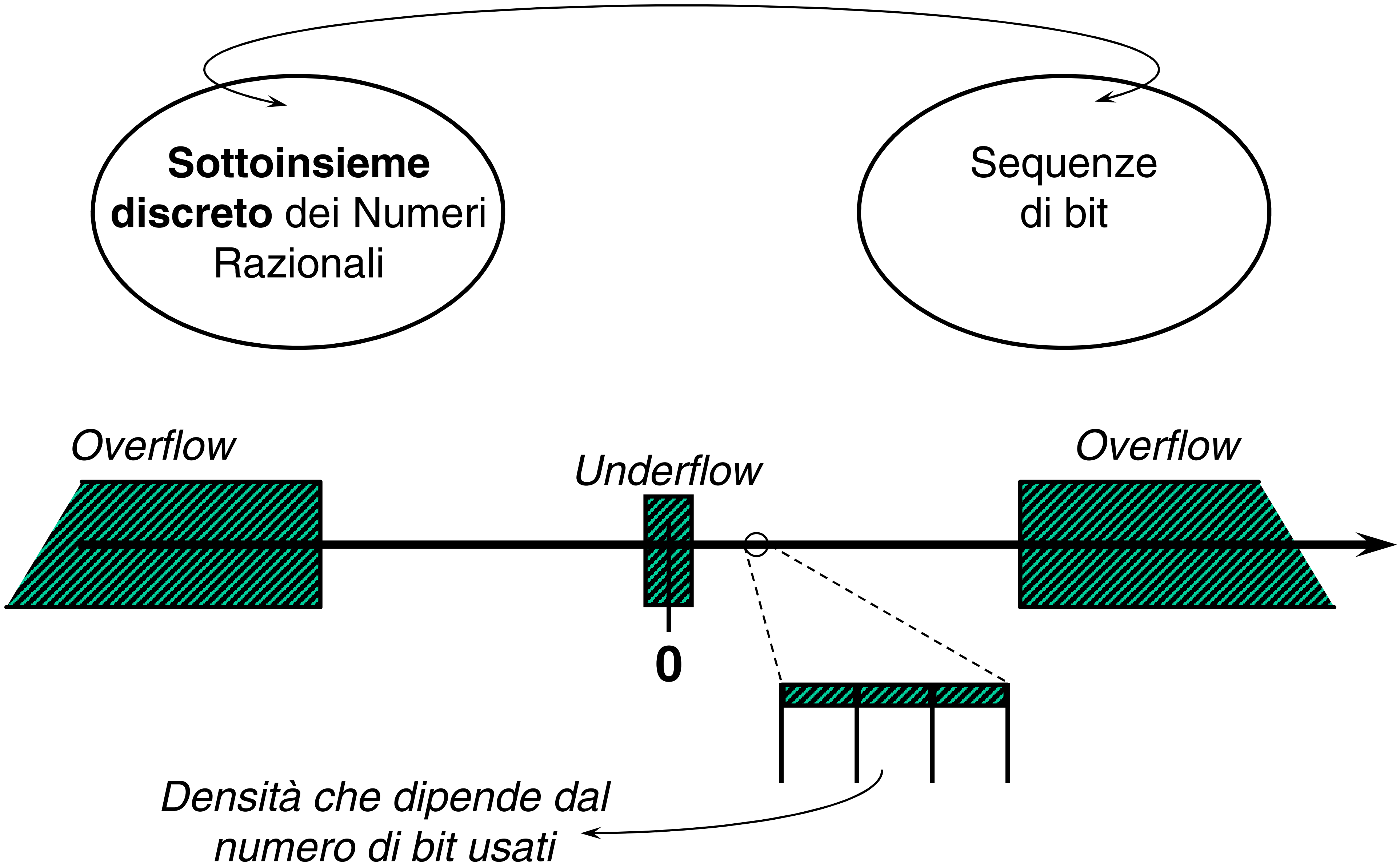
Ad esempio, tentando di rappresentare **a**=-9 su *p*=4 bit, si ottiene:
A=$-9+(2^{4-1}-1)=-9+7 = -2$, che non è un numero naturale!

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), invence, iniziano per 0.

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

Numeri Reali



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:

$$R = a_{p-f-1} \ldots a_0 \, a_{-1} \ldots a_{-f} \qquad r \cong \sum_{i=-f}^{p-f-1} a_i \beta^i = a_{p-f-1} \beta^{p-f-1} + \ldots + a_0 \beta^0 + \underbrace{a_{-1} \beta^{-1} + \ldots + a_{-f} \beta^{-f}}_{\textit{parte frazionaria}}$$

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:

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

$f_0 = F(r)$

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

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

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

...

fino a che f_{-j} è uguale a zero oppure si è raggiunta la precisione desiderata

Esempio:

$p = 16$ e $f = 5$

$r = 331.6875$

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;**

Numeri Reali – Virgola fissa(4/5)

F	**I**
$f_0 = 0.6875$	
$f_{-1}=F(0.6875*2=\mathbf{1.375})=0.375$	$a_{-1}=I(\mathbf{1.375})=\mathbf{1}$
$f_{-2}=F(0.375*2=\mathbf{0.75})=0.75$	$a_{-2}=I(\mathbf{0.75})=\mathbf{0}$
$f_{-3}=F(0.75*2=\mathbf{1.5})=0.5$	$a_{-3}=I(\mathbf{1.5})=\mathbf{1}$
$f_{-4}=F(0.5*2=\mathbf{1.0})=0$	$a_{-4}=I(\mathbf{1.0})=\mathbf{1}$

- 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 = (00101001011.10110) base *due*

- Controprova riguardo alla parte frazionaria:

$$1*2^{-1} + 0*2^{-2} + 1*2^{-3} + 1*2^{-4} \Rightarrow 0.5 + 0.125 + 0.0625 \Rightarrow 0.6875$$

- In altre parole abbiamo verificato che $(0.6875)_{\text{base } dieci} \Leftrightarrow (0.1011)_{\text{base } due}$

ERRORE DI TRONCAMENTO

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

F	I
$f_0 = 0.3$	
f_{-1}=F(0.3*2=**0.6**)=0.6	a_{-1}=I(**0.6**)=**0**
f_{-2}=F(0.6*2=**1.2**)=0.2	a_{-2}=I(**1.2**)=**1**
f_{-3}=F(0.2*2=**0.4**)=0.4	a_{-3}=I(**0.4**)=**0**
f_{-4}=F(0.4*2=**0.8**)=0.8	a_{-4}=I(**0.8**)=**0**
f_{-5}=F(0.8*2=**1.6**)=0.6	a_{-5}=I(**1.6**)=**1**
f_{-6}=F(0.6*2=**1.2**)=0.2	a_{-6}=I(**1.2**)=**1**

Per esprimere r sono necessarie un numero infinito di cifre!

(10.0 1001 1001 1001… = 10.0$\underline{1001}$ dove $\underline{1001}$ è la parte periodica)

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

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

r' = 10.010011

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

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

CALCOLO DI FISICA ASTRONOMICA

$m_e = 9 \times 10^{-28}$ gr; $m_{sole} = 2 \times 10^{+33}$ gr $\Rightarrow$ 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

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

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 $\quad 0.314 \ 10^{+1} \quad 31.4 \ 10^{-1}$

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

Numeri Reali – Virgola mobile(2/11)

Rappresentazione di un numero binario in virgola mobile:

$+1110.01$ numero reale binario in virgola fissa

esponenti espressi in binario

$+1.11001 \cdot due^{+11}$ … … $+111001 \cdot due^{-10}$

(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

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

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+$F/2^G$)

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

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

Half precision: 16 bit, *K* = 5 e *G* = 10. Esempi di decodofica R => r

Esempio 1

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

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

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

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

r = -30.578125 in base *dieci*

Esempio 2

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

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

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

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

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

Half precision, esempi di codifica r => 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:

$f = F/2^G = 0$

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

$\mathbf{r} = +1.0000000000 \cdot due^{+1} = 2$ 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:

$f = F/2^G = 0$

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

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

Half precision, esempi di codifica r => 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 $10.0\underline{1001}$, *iniziamo a portarlo in forma normalizzata:* $1.00\underline{1001} \cdot due^{+1}$

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

$1.\underbrace{00\ 1001\ 1001}_{F}\ \cancel{1001\ 1001}\ldots \cdot due^{+1}$

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}

Numero massimo e minimo: +max_pos, -max_pos

R={*0* 11111 1111111111} (massimo numero rappresentabile, +max_pos)
R={*1* 11111 1111111111} (minimo numero rappresentabile, -max_pos)

$f = F/2^G = F/2^{dieci} = 0.1111111111$
$e = + E - (2^{K-1} - 1) = +11111 - (+01111) = +10000\ (+sedici)$

$|\mathbf{r}| = 1.1111111111 \cdot due^{+10000} < 2^{(2^{(K-1)}+1)} = \mathbf{2^{bias+2}} = 2^{17} = 131072$
$|\mathbf{r}| = 131008 \cong 1.3 \cdot 10^{+5}$ in base *dieci*

Riassumendo, nella rappresentazione *half precision*:
max_pos corrisponde a circa $+2^{+17}$

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

Numeri con modulo minimo: +min_pos e –min_pos

R={*0*, 00000,0000000000} (minimo numero positivo, 0^+)
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:

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

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

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

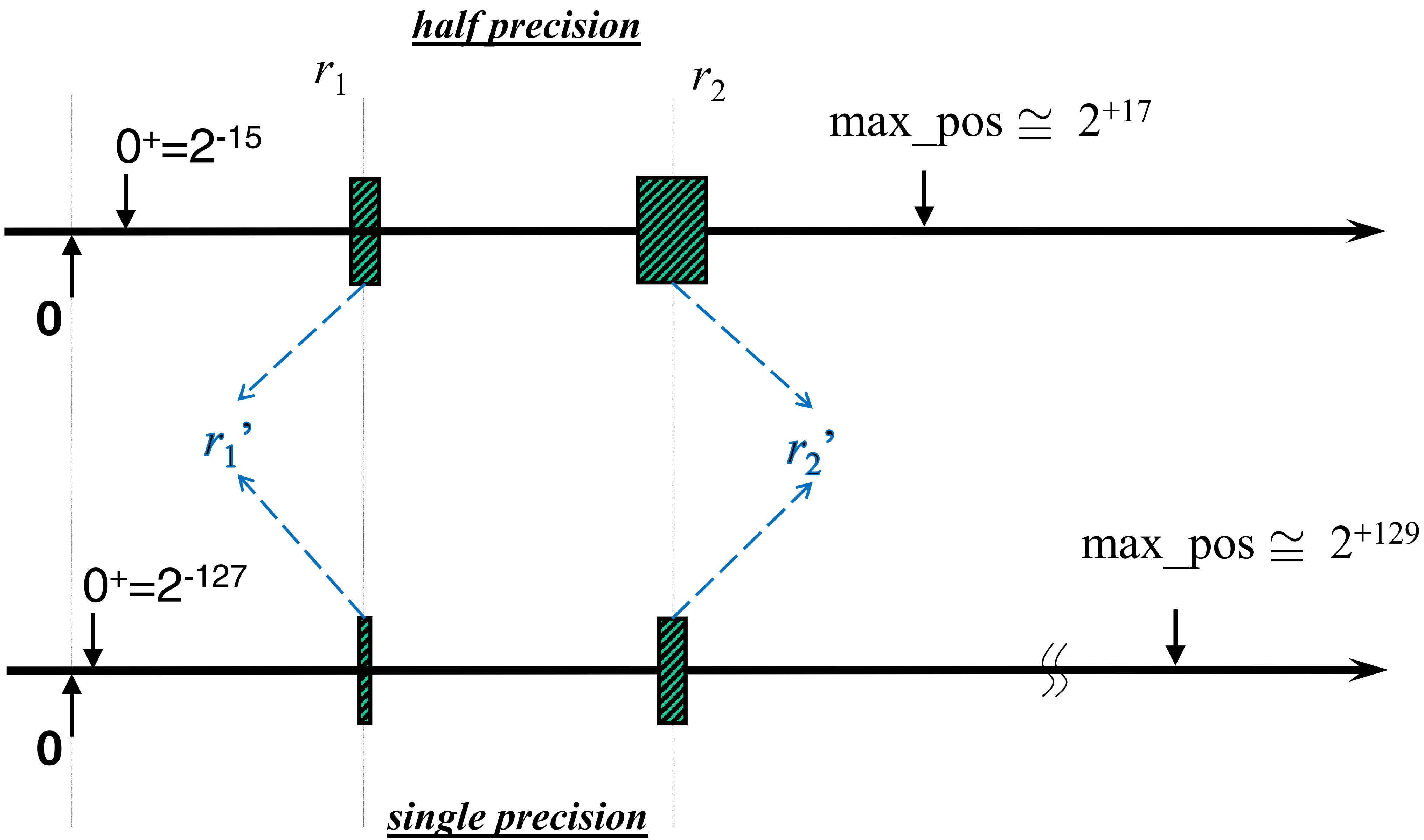
massimo modulo $\cong 2^{(bias+2)} = 2^{+129} \cong 6.8 \cdot 10^{+38}$
minimo modulo $= 2^{-bias} = 2^{-127} \cong 0.58 \cdot 10^{-38}$

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

massimo modulo $\cong 2^{(bias+2)} = 2^{+1025} \cong 3.6 \cdot 10^{+308}$
minimo modulo $= 2^{-bias} = 2^{-1023} \cong 1.1 \cdot 10^{-308}$

IEEE 754-2008 aggiunge:
rappresentazione in ***quadruple precision*** su **128 bit** con ***K* = 15** e ***G* = 112**

massimo modulo $\cong 2^{(bias+2)} = 2^{+16385} \cong 1.2 \cdot 10^{+4932}$
minimo modulo $= 2^{-bias} = 2^{-16383} \cong 1.6 \cdot 10^{-4932}$


half precision
r1
r2
0+=2-15
max_pos ≅ 2+17
0
r1'
r2'
0+=2-127
max_pos ≅ 2+129
0
single precision

Numeri Reali - Virgola mobile (11/11)

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 versione, 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.

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