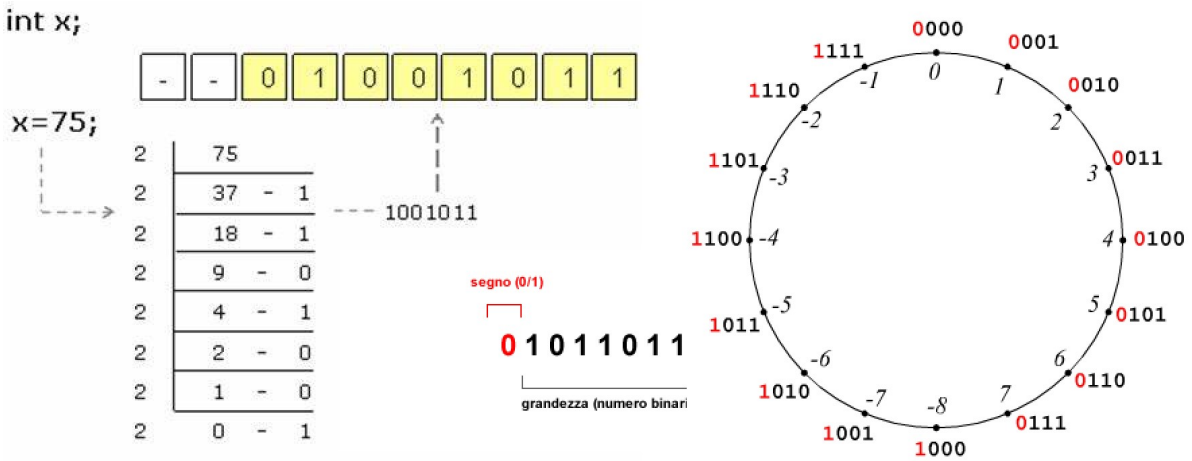
ITIS-LS “Francesco Giordani” Caserta

prof. Ennio Ranucci

a.s. 2020-2021

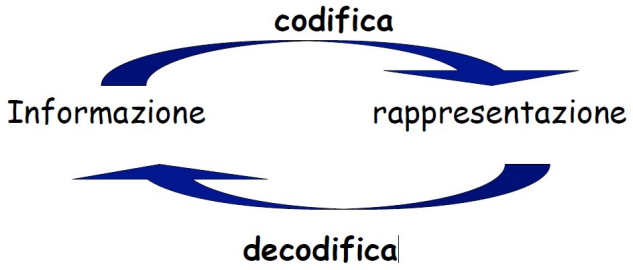
Rappresentazione delle informazioni in memoria e compressione dati

Codifica c++



RAPPRESENTAZIONE DELLE INFORMAZIONI NELLA MEMORIA DEL COMPUTER

Il computer per lavorare sui dati, ha bisogna che questi siano espressi come sequenze di 1 e di 0.
L'operazione di trasformazione dei dati in sequenze di 1 e di 0, cioè in numeri, prende il nome di **procedimento di codifica**.
Le informazioni: numeri, caratteri, immagini, suoni, video. . . devono essere rappresentate in forma binaria.



RAPPRESENTAZIONE DEI CARATTERI

UnO dei primi sistemi che vennero usati per trasformare i caratteri in sequenze di 1 e di 0 è la codifica **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) a 7 bit.
Avendo a disposizione 7 bit per la codifica dei caratteri è possibile codificare 128 combinazioni diverse!
Infatti $2^7 = 128$.
La codifica ASCII è una tabella di corrispondenza tra simboli e numeri, viene detta infatti codifica dei caratteri.
I **primi 32** caratteri del codice ASCII (con codice da 0 a 31) sono **caratteri di controllo,** i caratteri **da 32 a 127** sono **caratteri stampabili**

Byte	Cod.	Char
00000000	0	Null
00000001	1	Start of heading
00000010	2	Start of text
00000011	3	End of text
00000100	4	End of transmit
00000101	5	Enquiry
00000110	6	Acknowledge
00000111	7	Audible bell
00001000	8	Backspace
00001001	9	Horizontal tab
00001010	10	Line feed
00001011	11	Vertical tab
00001100	12	Form Feed
00001101	13	Carriage return
00001110	14	Shift out
00001111	15	Shift in
00010000	16	Data link escape
00010001	17	Device control 1
00010010	18	Device control 2
00010011	19	Device control 3
00010100	20	Device control 4
00010101	21	Neg. acknowledge
00010110	22	Synchronous idle
00010111	23	End trans. block
00011000	24	Cancel
00011001	25	End of medium
00011010	26	Substitution
00011011	27	Escape
00011100	28	File separator
00011101	29	Group separator
00011110	30	Record Separator
00011111	31	Unit separator

Byte	Cod.	Char
00100000	32	Spc
00100001	33	!
00100010	34	"
00100011	35	#
00100100	36	$
00100101	37	%
00100110	38	&
00100111	39	'
00101000	40	(
00101001	41	)
00101010	42	*
00101011	43	+
00101100	44	,
00101101	45	-
00101110	46	.
00101111	47	/
00110000	48	0
00110001	49	1
00110010	50	2
00110011	51	3
00110100	52	4
00110101	53	5
00110110	54	6
00110111	55	7
00111000	56	8
00111001	57	9
00111010	58	:
00111011	59	;
00111100	60	<
00111101	61	=
00111110	62	>
00111111	63	?

Byte	Cod.	Char
01000000	64	@
01000001	65	A
01000010	66	B
01000011	67	C
01000100	68	D
01000101	69	E
01000110	70	F
01000111	71	G
01001000	72	H
01001001	73	I
01001010	74	J
01001011	75	K
01001100	76	L
01001101	77	M
01001110	78	N
01001111	79	O
01010000	80	P
01010001	81	Q
01010010	82	R
01010011	83	S
01010100	84	T
01010101	85	U
01010110	86	V
01010111	87	W
01011000	88	X
01011001	89	Y
01011010	90	Z
01011011	91	[
01011100	92	\
01011101	93	]
01011110	94	^
01011111	95	_

Byte	Cod.	Char
01100000	96	`
01100001	97	a
01100010	98	b
01100011	99	c
01100100	100	d
01100101	101	e
01100110	102	f
01100111	103	g
01101000	104	h
01101001	105	i
01101010	106	j
01101011	107	k
01101100	108	l
01101101	109	m
01101110	110	n
01101111	111	o
01110000	112	p
01110001	113	q
01110010	114	r
01110011	115	s
01110100	116	t
01110101	117	u
01110110	118	v
01110111	119	w
01111000	120	x
01111001	121	y
01111010	122	z
01111011	123	{
01111100	124	|
01111101	125	}
01111110	126	~
01111111	127	Del

In seguito il codice ASCII è stato però esteso ad 8 bit,

2^8=256 caratteri

questo codice viene chiamato **ASCII esteso**. I codici che vanno da 0 a 127 che hanno uno 0 iniziale, sono gli stessi del codice ASCII su 7 bit, gli altri (da 128 a 255) che hanno un 1 iniziale sono utilizzati per codificare caratteri speciali.

Byte	Cod.	Char
10000000	128	Ç
10000001	129	ü
10000010	130	é
10000011	131	â
10000100	132	ä
10000101	133	à
10000110	134	å
10000111	135	ç
10001000	136	ê
10001001	137	ë
10001010	138	è
10001011	139	ï
10001100	140	î
10001101	141	ì
10001110	142	Ä
10001111	143	Å
10010000	144	É
10010001	145	æ
10010010	146	Æ
10010011	147	ô
10010100	148	ö
10010101	149	ò
10010110	150	û
10010111	151	ù
10011000	152	ÿ
10011001	153	Ö
10011010	154	Ü
10011011	155	ø
10011100	156	£
10011101	157	Ø
10011110	158	×
10011111	159	ƒ

Byte	Cod.	Char
10100000	160	á
10100001	161	í
10100010	162	ó
10100011	163	ú
10100100	164	ñ
10100101	165	Ñ
10100110	166	ª
10100111	167	º
10101000	168	¿
10101001	169	®
10101010	170	¬
10101011	171	½
10101100	172	¼
10101101	173	¡
10101110	174	«
10101111	175	»
10110000	176	▒
10110001	177	▒
10110010	178	▒
10110011	179	¦
10110100	180	¦
10110101	181	Á
10110110	182	Â
10110111	183	À
10111000	184	©
10111001	185	¦
10111010	186	¦
10111011	187	+
10111100	188	+
10111101	189	¢
10111110	190	¥
10111111	191	+

Byte	Cod.	Char
11000000	192	+
11000001	193	-
11000010	194	-
11000011	195	+
11000100	196	-
11000101	197	+
11000110	198	ã
11000111	199	Ã
11001000	200	+
11001001	201	+
11001010	202	-
11001011	203	-
11001100	204	¦
11001101	205	-
11001110	206	+
11001111	207	¤
11010000	208	ð
11010001	209	Ð
11010010	210	Ê
11010011	211	Ë
11010100	212	È
11010101	213	i
11010110	214	Í
11010111	215	Î
11011000	216	Ï
11011001	217	+
11011010	218	+
11011011	219	▒
11011100	220	▒
11011101	221	¦
11011110	222	Ì
11011111	223	▒

Byte	Cod.	Char
11100000	224	Ó
11100001	225	ß
11100010	226	Ô
11100011	227	Ò
11100100	228	õ
11100101	229	Õ
11100110	230	µ
11100111	231	þ
11101000	232	Þ
11101001	233	Ú
11101010	234	Û
11101011	235	Ù
11101100	236	ý
11101101	237	Ý
11101110	238	¯
11101111	239	´
11110000	240	-
11110001	241	±
11110010	242	‗
11110011	243	¾
11110100	244	¶
11110101	245	§
11110110	246	÷
11110111	247	¸
11111000	248	º
11111001	249	¨
11111010	250	·
11111011	251	¹
11111100	252	³
11111101	253	²
11111110	254	■
11111111	255	

La tabella ASCII fu creata sostanzialmente per poter scrivere in inglese.
La codifica ASCII non è sufficiente per poter scrivere perfettamente in italiano perché mancano le lettere accentate. Figuriamoci altre lingue come l'arabo, il cinese, il giapponese, ecc...
Per questo motivo si usa un'altra codifica dei caratteri, chiamata **UNICODE** (http://www.unicode.org) che utilizza 16 bit (65536 caratteri).
Contiene tutti i simboli per tutte le scritture del mondo (arabo, ebraico, cinese, giapponese, coreano, tailandese….) e non solo, contiene anche simboli per scrivere la matematica, alfabeti fonetici, latino, greco antico, ecc...
I primi 128 simboli UNICODE sono identici a quelli dell'ASCII per motivi di compatibilità con il passato, i successivi corrispondono ad altri alfabeti. Non riesce in ogni caso a coprire i simboli (oltre 200.000) di tutte le lingue!

7 bit (ASCII standard)

8 bit [1byte] (ASCII esteso)

16 bit [2byte] (UNICODE)

RAPPRESENTAZIONE DELLE PAROLE

Le **parole (stringhe)** sono sequenze di caratteri e come tali sequenze bit.
Una stringa di caratteri sarà rappresentata dal computer come una successione di gruppi di 8 bit.
Esempi:
"Ciao" = 01000011 01101001 01100001 01101111
"24" = 00110001 00110011
"cane" = 01100011 01100001 01101110 01100101

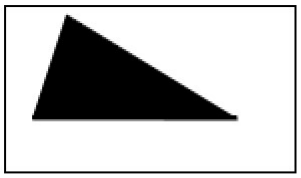
O	G	G	I		P	I	O	V	E
01001111	01000111	01000111	01001001	00100000	01010000	01001001	01001111	01010110	01000101

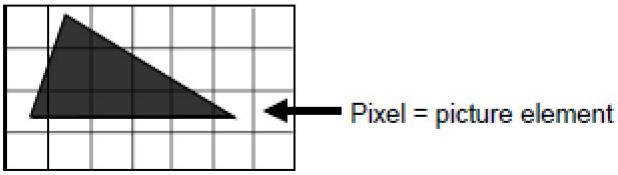
Esercizi
1. Nell'alfabeto di Marte sono previsti 300 simboli; quanti bit si devono utilizzare per rappresentarli tutti?
2.Quanti byte occupa la frase "**biologia marina**" se la si codifica utilizzando il codice ASCII?
3. Quanti byte occupa la stessa frase scritta in codice UNICODE?
4. Dati 12 bit per la codifica, quante informazioni distinte si possono rappresentare?

RAPPRESENTAZIONE DELLE IMMAGINI

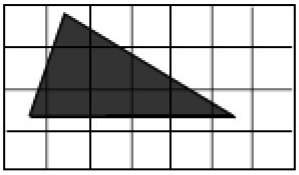
Anche le immagini possono essere codificate mediante una sequenza di 0 e 1., questa operazione si chiama **digitalizzazione**.
La prima cosa da fare è dividere l'immagine in tanti quadratini, che prendono il nome di PIXEL (Picture Element).
Una volta specificata la dimensione in pixel dell'immagine è sufficiente elencare in ordine il colore di ogni pixel.
Per le immagini in bianco e nero si assegna ad ogni pixel un bit
Per le immagini a colori si assegna ad ogni pixel una sequenza di bit

Consideriamo un'immagine in bianco e nero, senza ombreggiature o livelli di chiaroscuro.
L'immagine viene suddivisa mediante una griglia formata da righe orizzontali e verticali a distanza costante.





Il simbolo "0" viene utilizzato per la codifica di un pixel corrispondente ad un quadratino bianco (in cui il bianco è predominante), il simbolo "1" viene utilizzato per la codifica di un pixel corrispondente ad un quadratino nero (in cui il nero è predominante).
Si deve definire una convenzione per ordinare i pixel della griglia ipotesi: assumiamo che i pixel siano ordinati dal basso verso l'alto e da sinistra verso destra.

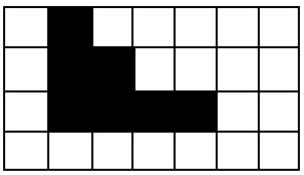


0_{22}	1_{23}	0_{24}	0_{25}	0_{26}	0_{27}	0_{28}
0_{15}	1_{16}	1_{17}	0_{18}	0_{19}	0_{20}	0_{21}
0_{8}	1_{9}	1_{10}	1_{11}	1_{12}	0_{13}	0_{14}
0_{1}	0_{2}	0_{3}	0_{4}	0_{5}	0_{6}	0_{7}

La rappresentazione della figura è data dalla stringa binaria:

0000000 0111100 0110000 0100000

Dato che il contorno della figura non sempre coincide con la griglia si ottiene un'approssimazione della figura originaria. Riconvertendo la stringa si avrà:



La rappresentazione delle immagini utilizzando la suddivisione in pixel si chiama grafica **bitmap (o raster).**
Le immagini di questo tipo sono anche dette bitmap perché i pixel vengono memorizzati all'interno del file mediante una mappa di bit.
Il formato bitmap (**.bmp**) è il formato standard di windows, semplice e con buona velocità di lettura e scrittura su disco, ma occupa molto spazio.
Le immagini bitmap occupano parecchio spazio, esistono quindi delle tecniche di compressione che permettono di ridurne le dimensioni eliminando i pixel ripetitivi. Ad esempio, se k pixel lungo la stessa riga hanno lo stesso colore, si memorizza il colore una volta sola e il numero k. I file che usano queste tecniche di compressione hanno le estensioni GIF (.**gif**), JPEG (.**jpg**), TIFF (.**tif**).

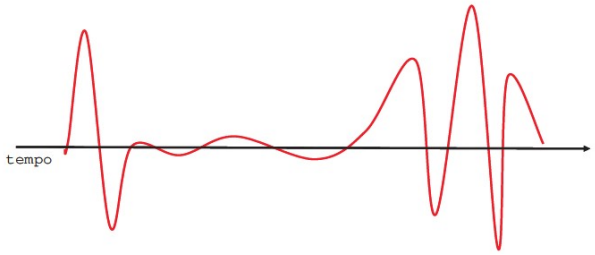
GRAFICA VETTORIALE
Esiste anche un altro metodo di rappresentare le immagini chiamato **vettoriale**. In questo metodo ogni elemento geometrico viene specificato individualmente.
La codifica vettoriale è utilizzata se le immagini sono regolari in cui non si specificano le informazioni di colore di ogni singolo pixel ma si specificano gli elementi geometrici (punti, segmenti, poligoni,…) che le compongono. Questi elementi geometrici sono rappresentati da formule matematiche (un rettangolo è definito da due punti, un cerchio da un centro e un raggio, una curva da più punti e un'equazione).
Le dimensioni dei file sono ridotte rispetto alle immagini bitmap. Sono immagini tipiche della progettazione meccanica (CAD).

Le immagini **bitmap** supportano in genere molti colori e trovano quindi applicazione in tutti quei casi in cui è richiesto un effetto pittorico o fotografico.
Purtroppo l'ingrandimento di un'immagine bitmap, porta inevitabilmente ad un decadimento della qualità dell'immagine stessa (vengono evidenziate in modo più marcato le differenze fra pixel e pixel). I file occupano molto spazio.
Un'immagine **vettoriale** ha il vantaggio di non risentire della "sgranatura" negli ingrandimenti. I file vettoriali possono infatti essere ingranditi, rimpiccioliti e deformati a piacimento senza avere nessuna perdita di definizione in quanto una volta noti i dati di base (ad esempio le coordinate del centro e raggio per un cerchio) le funzioni matematiche implementate nel programma permettono di riprodurre con precisione l'oggetto.

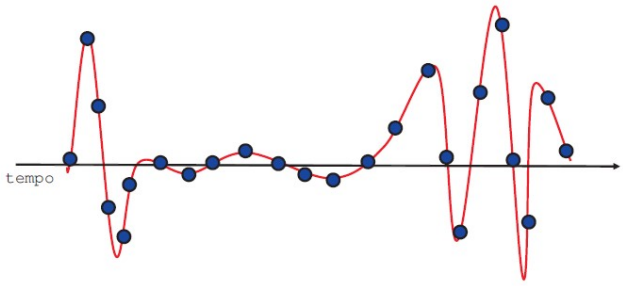
I VIDEO
Un filmato è una sequenza di immagini (dette fotogrammi o frames) che si susseguono rapidamente. Il movimento appare fluido e senza scatti se le immagini si susseguono alla velocità di almeno 25 fotogrammi al secondo.
Per codificare un filmato basta quindi digitalizzare i suoi fotogrammi, il che ci riconduce alle problematiche del paragrafo precedente.
In genere si tratta di sequenze compresse di immagini dove vengono registrate solo le variazioni tra un fotogramma e l'altro.

Vari formati (comprendente il sonoro):
• .avi (Audio Video Interleave, Microsoft)
• .mov
• .mpeg (il più usato) compresso
• Quicktime (Apple)

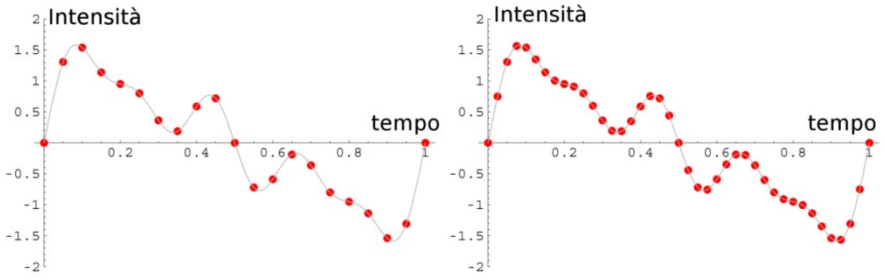
I SUONI
Il suono, come è noto, è una vibrazione. Fisicamente un suono è rappresentato come un'onda che descrive la variazione della intensità nel tempo (onda sonora). Graficamente un'onda sonora può essere rappresentata riportando sull'asse delle x il tempo e sull'asse delle y il valore dell'intensità:



Per codificare i suoni si effettuano dei **campionamenti** sull'onda cioè si misura l'ampiezza dell'onda a intervalli di tempo costanti e si codificano in forma digitale le informazioni estratte da tali campionamenti:



La sequenza dei valori numerici così ottenuta può essere facilmente codificata.
Minore è l'intervallo che intercorre tra una misurazione e la successiva e maggiore sarà la qualità del suono. La **frequenza di campionamento** indica il numero di rilevazioni effettuate in un secondo. L'unità di misura della frequenza è l'Hertz (Hz) e in genere si utilizzano frequenze da 8 a 48 kHz (kilohertz = 1000 Hz). Se un suono è campionato a 200 Hz significa che in un secondo di suono vengono presi 200 campioni.



A destra la frequenza di campionamento è più alta rispetto a sinistra e questo farà sì che il suono digitalizzato sarà più simile al suono originale.

RAPPRESENTAZIONE DEI NUMERI

Quanti oggetti posso codificare con k bit?
1 bit -> (0, 1) -> 2 oggetti 2^1
2 bit -> (00, 01, 10, 11) -> 4 oggetti 2^2
3 bit -> (000, 001, 010, …, 111) -> 8 oggetti 2^3
…
k bit -> (...)-> 2^k oggetti

Domanda: quanti diversi valori posso rappresentare con 2 byte?
Risposta: 2 byte = 16 bit, quindi posso rappresentare 2^{16} = 65536 diversi valori.
Quanti bit mi servono per codificare N oggetti?
Devo trovare quel numero K tale che……N <= 2^K
Domanda: quanti bit mi servono per rappresentare 112 diversi valori?
Risposta: 7 bit (27 = 128). 6 bit sarebbero stati pochi, mentre 8 bit sarebbero stati troppi!

Rappresentazione dell'informazione numerica

• Numeri naturali (insieme N)
• Numeri interi (insieme Z)
• Numeri razionali (insieme Q)
……..

La precisione con cui i numeri possono essere espressi è finita e predeterminata poiché questi devono essere memorizzati entro un limitato spazio di memoria.
I numeri a precisione finita sono quelli rappresentati con un numero finito di cifre.
Nei sistemi di numerazione posizionali i dieci simboli/cifre:
0 1 2 3 4 5 6 7 8 9
acquistano un valore/peso diverso a seconda della posizione che occupano.
$73093 = 7 \cdot 10^4 + 3 \cdot 10^3 + 0 \cdot 10^2 + 9 \cdot 10^1 + 3 \cdot 10^0 =$
$= 7 \cdot 10000 + 3 \cdot 1000 + 0 \cdot 100 + 9 \cdot 10 + 3 \cdot 1$
Anche nel sistema binario:
$1000111 = 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 =$
$= 1 \cdot 64 + 0 \cdot 32 + 0 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 71$
$10001112 = 1 \cdot 26 + 0 \cdot 25 + 0 \cdot 24 + 0 \cdot 23 + 1 \cdot 22 + 1 \cdot 21 + 1 \cdot 20 =$
$= 1 \cdot 64 + 0 \cdot 32 + 0 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 71$
Viceversa, per convertire un numero decimale nel suo equivalente binario bisogna effettuare la divisione intera del numero decimale per 2 finché non si ha come risultato zero: i resti ottenuti formano il numero binario. I resti devono essere presi in ordine inverso: l'ultimo resto ottenuto è la *cifra binaria più significativa*, il primo è la *cifra meno significativa*. Per esempio:
71 : 2 = 35 resto 1
35 : 2 = 17 resto 1
17 : 2 = 8 resto 1
8 : 2 = 4 resto 0
4 : 2 = 2 resto 0
2 : 2 = 1 resto 0
1 : 2 = 0 resto 1
La rappresentazione binaria di 71 è dunque 1000111.

NUMERI NATURALI

Per i numeri naturali si usa la **rappresentazione binaria posizionale:**
$(101100)_2 = (44)_{10}$
Con n bit si possono rappresentare $\mathbf{2^n}$ diversi numeri naturali. Quali sono?
I numeri rappresentabili appartengono all'intervallo:
$(0; 2^n - 1)$
n è la dimensione (in bit) della cella di memoria che contiene il numero
Insieme dei valori rappresentabili:
con 1 byte si possono rappresentare i num tra **$(0; 2^8 - 1)$ cioè tra (0; 255)**
con 2 byte **$(0; 2^{16} - 1)$…..(0; 65535)**
con 4 byte **$(0; 2^{32} - 1)$…..$(0; 4*10^9)$**
con 8 byte **$(0; 2^{64} - 1)$…….(0; ????)**
Dato che lo spazio disponibile è finito, vi sono dei limiti nella dimensione dei numeri memorizzabili ….
Le operazioni con i numeri a precisione finita causano errori quando il loro risultato non appartiene all'insieme dei valori rappresentabili:
Underflow: si verifica quando il risultato dell'operazione è minore del più piccolo valore rappresentabile
Overflow: si verifica quando il risultato dell'operazione è maggiore del più grande valore rappresentabile
Non appartenenza all'insieme: si verifica quando il risultato dell'operazione, pur non essendo troppo grande o troppo piccolo, non appartiene all'insieme dei valori rappresentabili
Esempio: si considerino i numeri naturali di tre cifre, non possono essere rappresentati:
Numeri superiori a 999
Numeri negativi
Frazioni e numeri irrazionali
Alcuni errori possibili in operazioni fra tali numeri:
600+600 = 1200 Overflow
300-600 = -300 Underflow
007/002 = 3.5 Non appartenenza all'insieme
Valore minimo di una sequenza di n cifre binarie: 000 … 0 (n volte)
Valore massimo di una sequenza di n cifre binarie: 1111…111 (n volte)

NUMERI INTERI

Modulo e segno

E' indispensabile indicare il numero K di bit utilizzati per la rappresentazione
Il bit più a sinistra (il + significativo) rappresenta il segno del numero: 0 = '+' , 1 = '-'
I rimanenti k-1 bit rappresentano il modulo
Come si converte da decimale a Modulo e Segno?
Si calcola la rappresentazione binaria del valore assoluto del numero senza il bit del segno poi se il numero è negativo si pone = a 1 il bit del segno se il numero è positivo si pone = a 0 il bit del segno.
NB: il bit del segno non ha significato numerico
Esempi:
se utilizzo 4 bit…..
+7 = (0111) -7 = (1111)
+6 = (0110) -6 = (1110)
se utilizzo 8 bit…..
-5 = (10000101) +5 = (00000101)

Come si converte da Modulo e Segno a decimale?
Si elimina il bit del segno e si converte il valore assoluto in notazione decimale. Il risultato sarà il valore assoluto se il bit di segno è 0, oppure il corrispondente numero negativo se il bit di segno è 1.

Vediamo la rappresentazione dei numeri da -7 a +7 in ModuloSegno:

Codice	Nat	MS
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7

Codice	Nat	MS
1000	8	-0
1001	9	-1
1010	10	-2
1011	11	-3
1100	12	-4
1101	13	-5
1110	14	-6
1111	15	-7

:

Attenzione…….. ci sono due zeri: +0=0000modulosegno e -0=1000modulosegno
In modulo e segno, con n bit, possiamo rappresentare gli interi nell'intervallo da:
$(-(2^{n-1}-1); 2^{n-1}-1)$
Con n=4 bit i valori rappresentabili vanno da $-2^3+1=-7$ a $2^3-1=+7$
Con n=8 bit i valori rappresentabili vanno da $-2^7+1=-127$ a $2^7-1=+127$
Con n=16 bit i valori rappresentabili vanno da -32767 a +32767

Complemento a 2
E' indispensabile indicare il numero **K** di bit utilizzati per la rappresentazione
Il bit più a sinistra rappresenta il segno del numero: 0 = '+' , 1 = '-'
La parte restante della rappresentazione NON è il valore assoluto del numero, lo è soltanto per i numeri positivi
C'è una sola rappresentazione dello 0. Non ci sono più configurazioni "sprecate":
Con 4 bit $0000_2 = 0_{10}$ mentre $1000_2 = -8_{10}$
Come si converte da Decimale a C2?
Numeri interi positivi (compreso lo 0): un numero positivo è rappresentato in modo standard su k bit (come abbiamo visto per modulo e segno)
Numeri interi negativi: si trova la rappresentazione di **-X** a partire da quella di **X.** Effettuare il complemento di ogni bit di X e aggiungere 1

I tre passi da compiere:
1) rappresentare X in modo standard su k bit
2)complementare tutti i bit (1 ► 0, 0 ► 1)
3)sommare 1 al risultato
Esempio: dati 4 bit trovare la rappresentazione di -6 in C2
rappresentazione di +6 = 0110
complemento tutti i bit = 1001
Aggiungo 1 a 1001 ottenendo 1010
Risultato -6 = 1010

Vediamo la rappresentazione dei numeri da -7 a +7 in C2:

Codice	Nat	MS	C2
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7

Codice	Nat	MS	C2
1000	8	-0	-8
1001	9	-1	-7
1010	10	-2	-6
1011	11	-3	-5
1100	12	-4	-4
1101	13	-5	-3
1110	14	-6	-2
1111	15	-7	-1

Come si converte da C2 a Decimale?
Numeri interi positivi (compreso lo 0): stesso procedimento della codifica modulo e segno
Numeri interi negativi:

Prendo in considerazione tutti i bit compreso anche il bit del segno.
• si sottrae 1 al numero rappresentato in C2
• si complementano tutti i bit (1 ► 0, 0 ► 1)
• si converte da binario a decimale e si aggiunge il segno

In complemento a 2, con n bit, possiamo rappresentare gli interi nell'intervallo da:
$(-2^{n-1}; 2^{n-1}-1)$
Con k=4 bit i valori rappresentabili vanno da **-8 a +7**
Con k=8 bit i valori rappresentabili vanno da **-128 a +127**
Con k=16 bit i valori rappresentabili vanno da **-32768 a + 32767**

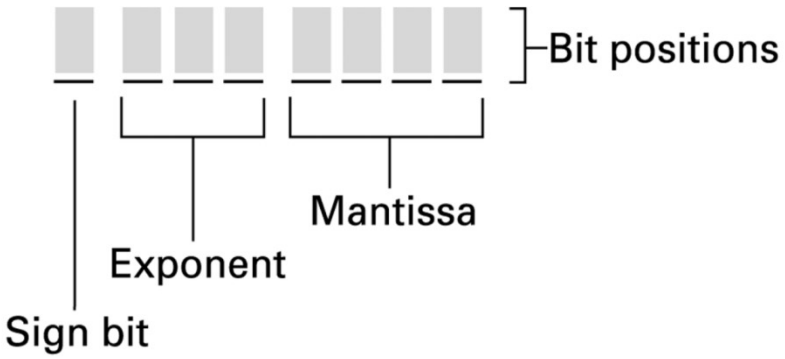
RAPPRESENTAZIONE DEI NUMERI RAZIONALI: VIRGOLA FISSA

- Un numero razionale ha una **parte intera** (prima della virgola) e una **parte frazionaria** (dopo la virgola), es: `3.12`, `0.00074`, `5000.7`, ... (in base 10)
- rappresentazione binaria solitamente su 4/8 byte
- **Rappresentazione in virgola fissa**: riservo un numero fisso di bit per parte intera e parte frazionaria;
- per semplicità consideriamo solo numeri positivi

es: con **3 bit** per la **parte intera** e **2 bit** per quella **frazionaria** posso rappresentare numeri come:
`011.11` `101.01` `000.01`

RAPPRESENTAZIONE DEI NUMERI RAZIONALI: VIRGOLA MOBILE

- Per rappresentare numeri in virgola mobile nel computer, dobbiamo fissare un numero di bit N_m per **il valore assoluto della mantissa**, e un numero di bit N_e per l'esponente **in complemento a 2**
- **Numeri negativi**: rappresentiamo il valore assoluto, mettendo **1** nel **bit del segno**.
- Se $N_m = 4$ e $N_e = 3$ abbiamo una rappr. su 8 bit come a destra.



Unità di misura più diffuse

Simbolo	Unità di misura	Descrizione
Bd	Baud	Equivale a 1 bps (Bit Per Secondo)
bps	Bit Per Secondo	Misura il trasferimento dei dati tra computer.
MIPS	Milioni di Istruzioni Per Secondo	Indicano la quantità di istruzioni elementari del processore che vengono elaborati in un secondo all'interno della CPU[1].
Hz	hertz	Misura la velocità del clock[2] interno.
dpi	Dots Per Inch	Stabilisce la quantità di punti per pollice quadrato (2,54 x 2,54 cm) in una immagine e in una stampa.
px	Pixel	Minimo puntino luminoso, la cui quantità o densità identifica la risoluzione di un monitor.

1) CPU - Central Processing Unit - è l'unità centrale di elaborazione del computer e comprende: Microprocessore (unità di controllo e unità aritmetico logica), Memoria d'uso (RAM e ROM).
2) Il clock sincronizza i vari componenti del sistema.

Unità di misura per la memorizzazione delle informazioni

Simbolo	in Bit	in Byte	in potenze di 2
1 b (bit)	1	1/8	2^1 = 2 stati (acceso - spento)
1 B (byte)	8	1	2^8 = 256 caratteri
1 KB (kilobyte)	8.192	1.024	2^{10} byte
1 MB (megabyte)	8.388.608	1.048.576	2^{20} byte
1 GB (gigabyte)	8.589.934.592	1.073.741.824	2^{30} byte
1 TB (terabyte)	8.796.093.302.400	1.099.511.628.000	2^{40} byte

Tempo di trasferimento (download) di 1 MB

- a 14.400 bps (1.8 KB/s) = 582 sec.
- a 28.800 bps (3.6 KB/s) = 291 sec.
- a 33.600 bps (4.2 KB/s) = 250 sec.
- a 56.000 bps (7.0 KB/s) = 150 sec.
- a 64.000 bps (8.0 KB/s) = 131 sec. (ISDN)
- a 128.000 bps (16.0 KB/s) = 65 sec.
- a 256.000 bps (32.0 KB/s) = 33 sec.
- a 382.000 bps (47.7 KB/s) = 22 sec.
- a 640.000 bps (75.0 KB/s) = 14 sec. (ADSL)
- a 1.000.000 bps (125.0 KB/s) = 8 sec.

Molto importante:

La velocità di trasferimento dei modem viene espressa in bps (bit per secondo);
si parla ad es. di modem a 56 Kbps o 56.000 bps.

Poiché le dimensioni dei files vengono espresse in byte (B) e multipli (KB, MB, GB, TB), per ottenere la velocità espressa in KB dobbiamo dividere per 8;
ad es: un modem a 33.600 bps trasferisce 33.600/8 = 4.200 byte/sec = 4,2 KB/sec.

Il **petabyte** (simbolo **PB**) è un'unità di misura informatica del Sistema Internazionale definita come un multiplo del byte. Per definizione 1 petabyte equivale a mille terabyte, a un milione di gigabyte e a un biliardo di byte: 1 PB = 10^3 TB = 10^6 GB = 10^{15} B.

Il **petabyte** è una misura informatica ancora poco conosciuta e diffusa nelle nostre vite quotidiane.

La **compressione dati senza perdita** (o **compressione dati *lossless***),

In informatica e telecomunicazioni, è una classe di algoritmi di compressione dati che non porta alla perdita di alcuna parte dell'informazione originale durante la fase di compressione/decompressione dei dati stessi.
Gli algoritmi di compressione *lossless* non possono sempre garantire che ogni insieme di dati in input diminuisca di dimensione.
Molti applicativi che utilizzano la compressione lossless prevedono di lasciare invariati gli insiemi di dati la cui dimensione sia aumentata dopo la compressione.
Un algoritmo di compressione lossless di un insieme di cifre potrebbe essere "cifra-numero di ripetizioni".

Implementazione che utlizza un vettore

```
#include <iostream>

#include <string>

using namespace std;

int vet[100] = {10,4,5,5,5,18,18,15,15,15,15,4,4,4,4};

int vet2[100]= {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

int dimLog=14;

int dimLogZip;

int vetZip[200];

int vetUnZip[100];

void zip(int vetPar[100],int dimLogPar);

void stampaVet(int vetPar[100],int dimLogPar);

void stampaVetZip(int vetPar[100],int dimLogPar);

void unZip(int vetPar[],int dimLogPar);

int main ()

 {

   zip(vet,dimLog);

   stampaVet(vet,dimLog);

   cout<<endl<<"dimensione vettore prima della compressione: "<<dimLog+1<<endl;

   stampaVetZip(vetZip,dimLogZip);

   cout <<endl<< "dimensione vettore compresso: "<<dimLogZip<<endl;
```

```
  unZip(vetZip,dimLogZip);
  cout <<"Vettore decompresso: "<<endl;
  stampaVet(vetUnZip,dimLog);
  return 0;
}
void stampaVet(int vetPar[100],int dimLogPar)
{
  for (int i=0; i<=dimLogPar;i++)
  {
    cout<<vetPar[i]<<" ";
  }
}
void stampaVetZip(int vetPar[100],int dimLogPar)
{
   for (int i=0; i<dimLogPar;i=i+2)
  {
     cout<<vetPar[i]<<"/"<<vetPar[i+1]<<" ";
  }
}
void zip(int vetPar[],int dimLogPar)
{
 int contatore,j=0,k=0;
  for (int i=0; i<=dimLogPar;i++)
    {
     contatore=0;
     j=i+1;
     while (vetPar[i]==vetPar[j])
      {
```

```
        contatore++;
        j++;
       }
      vetZip[k]=vetPar[i];
      vetZip[k+1]=contatore+1;
      k=k+2;
      i=j-1;
    }
  dimLogZip=k;
}
void unZip(int vetPar[],int dimLogPar)
{
  int k=0;
  for (int i=0; i<=dimLogPar;i=i+2)
    {
     for (int j=1;j<=vetPar[i+1];j++)
        {
         vetUnZip[k]=vetPar[i];
         k++;
        }
    }
}
```

Implementazione che utlizza una stringa

```
#include <iostream>
#include <string>
#include <sstream>
#include <stdlib.h>
using namespace std;
string str = "10000111111111111111111111110010011111";
string strZip="";
string strUnZip;
int dimStr;
string numeroRipetizioni;
void zip(string strPar);
void unZip(string strPar);
string toString(int n)
{
   ostringstream temp;
   temp<<n;
   return temp.str();
}
int toInt(string strPar)
{
   int dim=strPar.length();
   int result=0;
   for(int i=0; i<dim; i++)
     {
       result = result * 10 + ( strPar[i] - '0' );
     }
```

```
    return result;
}
int main ()
 {
   dimStr=str.length();
   cout<<str<<endl;
   cout<<"dimensione vettore prima della compressione: "<<dimStr<<endl;
   zip(str);
   cout<<strZip<<endl;
   int dimStrZip=strZip.length();
   cout <<"dimensione vettore compresso: "<<dimStrZip<<endl;
   unZip(strZip);
   cout <<"Stringa decompressa: "<<strUnZip<<endl;
   return 0;
}

void zip(string strPar)
{
  int contatore,j=0;
   for (int i=0; i<=dimStr-1;i++)
     {
       contatore=0;
       j=i+1;
       while (strPar[i]==strPar[j])
        {
          contatore++;
          j++;
        }
```

```
      contatore++;
      numeroRipetizioni= toString(contatore);
      strZip=strZip+strPar[i]+"#"+numeroRipetizioni+'#';
      i=j-1;
     }

}

void unZip(string strPar)
{
  strUnZip="";
  int dim=strPar.length()-1;
  int numeroRipetizioni;
  string ripetizioni;
  string elemento;
  for (int i=0;i<dim;i++)
    {
      elemento="";
      while (strPar[i]!='#')
        {
          elemento=elemento+strPar[i];
          i++;
        }
      i++;
      ripetizioni="";
      while (strPar[i]!='#')
        {
          ripetizioni=ripetizioni+strPar[i];
```

```
        i++;
      }
    numeroRipetizioni=toInt(ripetizioni);
    for(int k=1;k<=numeroRipetizioni;k++)
    {
      strUnZip=strUnZip+elemento;
    }
  }
}
```

Implementazione che trasforma la stringa di caratteri in una stringa binaria

```
#include <iostream>
#include <string>
#include <sstream>
#include <stdlib.h>
using namespace std;
string str; //= "10000111111111111111111111111110010011111";
string strZip="";
string strUnZip;
int dimStr;
string numeroRipetizioni;
string convertiStrBin(string strPar);
void zip(string strPar);
void unZip(string strPar);
string toString(int n)
{
   ostringstream temp;
   temp<<n;
   return temp.str();
}
int toInt(string strPar)
{
   int dim=strPar.length();
   int result=0;
   for(int i=0; i<dim; i++)
     {
       result = result * 10 + ( strPar[i] - '0' );
     }
```

```
    return result;
}
int main ()
 {
   str=convertiStrBin("abbbbbccccccc");
   //str="abbbbbccccccc";
   dimStr=str.length();
   cout<<str<<endl;
   cout<<"dimensione vettore prima della compressione: "<<dimStr<<endl;
   zip(str);
   cout<<strZip<<endl;
   int dimStrZip=strZip.length();
   cout <<"dimensione vettore compresso: "<<dimStrZip<<endl;
   unZip(strZip);
   cout <<"Stringa decompressa: "<<strUnZip<<endl;
   return 0;
}
string convertiStrBin(string strPar)
{
    string strBin="";
    char car;
    string carBin;
    int n;
    char buffer[255];
    int dim=strPar.length();
    for(int i=0; i<dim; i++)
      {
        car=strPar[i];
```

```
        n=(int)car;
        carBin=itoa(n,buffer,2);
        strBin=strBin+carBin;
      }
     return strBin;
}
void zip(string strPar)
{
  int contatore,j=0;
   for (int i=0; i<=dimStr-1;i++)
     {
       contatore=0;
       j=i+1;
       while (strPar[i]==strPar[j])
        {
          contatore++;
          j++;
        }
       contatore++;
       numeroRipetizioni= toString(contatore);
       strZip=strZip+strPar[i]+"#"+numeroRipetizioni+'#';
       i=j-1;
     }
}
void unZip(string strPar)
{
   strUnZip="";
   int dim=strPar.length()-1;
```

```
  int numeroRipetizioni;
  string ripetizioni;
  string elemento;
  for (int i=0;i<dim;i++)
    {
      elemento="";
      while (strPar[i]!='#')
        {
          elemento=elemento+strPar[i];
          i++;
        }
      i++;
      ripetizioni="";
      while (strPar[i]!='#')
        {
          ripetizioni=ripetizioni+strPar[i];
          i++;
        }
      numeroRipetizioni=toInt(ripetizioni);
      for(int k=1;k<=numeroRipetizioni;k++)
      {
        strUnZip=strUnZip+elemento;
      }
    }
}
```