Precisazione sui tipi in ANSi C

Tipi primitivi del C

Un tipo è costituito da un insieme di valori ed un insieme di operazioni su questi valori.

Classificazione dei tipi primitivi del C

- **scalari**
  - **aritmetici**:
    - **interi**: con segno, senza segno, caratteri, enumerati
    - reali
  - **puntatori**
- **aggregati**
  - array
  - strutture
- **void** (nessun valore e nessuna operazione)

Precisazione sui tipi in ANSi C

Tipi aritmetici del C. Per ciascun tipo consideriamo i seguenti aspetti:

1. intervallo di definizione

2. notazione per le costanti (nel codice, oppure in input/output)

3. operatori

4. predicati (operatori di confronto)

Precisazione sui tipi in ANSi C: una nota

– Per determinare le dimensioni di una variabile o di un tipo in C abbiamo l'operatore **sizeof**

– **sizeof** restituisce il numero di byte occupati da una variabile di un certo tipo

– Uso: sizeof (tipo) oppure sizeof(variabile)

Precisazione sui tipi in ANSi C: una nota

Esempio di uso di sizeof:
– Esempio:

```
#include <stdio.h>
int main(){
    int a;
    double x;
    printf("Dimensione a: %lu\n",sizeof(a));
    printf("Dimensione a: %lu\n",sizeof(int));
    printf("Dimensione x: %lu\n",sizeof(x));
    printf("Dimensione x: %lu\n",sizeof(double));


return 0;
}
```

Tipi interi

Codificati in binario

Interi con segno
3 tipi:

- --short (oppure short int, signed short, signed short int)
- – int (oppure signed int, signed)
- – long (oppure long int, signed long, signed long int)

Intervallo di definizione: da -2^{n-1} a $2^{n-1}-1$, dove n dipende dal compilatore

Vale che:

- – sizeof(short) ≤ sizeof(int) ≤ sizeof(long)
- – sizeof(short) ≥ 2 (ovvero, almeno 2 byte, 16 bit)
- – sizeof(long) ≥ 4 (ovvero, almeno 4 byte, 16 bit)

In gcc: short: 16 bit, int: 32 bit, long: 32 bit

I valori limite sono contenuti nel file header **limits.h**, che definisce le costanti:

SHRT_MIN, SHRT_MAX, INT_MIN, INT_MAX, LONG_MIN, LONG_MAX

Tipi interi

Notazione per le costanti: in decimale: 0, 10, -10, . . .

Per distinguere long (solo nel codice): 10L (oppure 10l, ma l sembra 1).

Operatori: +, -, *, /, %, ==, !=, <, >, <=, >=, ++,--, +=, -=

Ingresso/uscita: tramite printf e scanf, con i seguenti specificatori di formato (dove d indica “decimale”):

%hd per short

%d per int

%ld per long (con l minuscola

Tipi interi

Interi senza segno. 3 tipi:

- unsigned short (oppure unsigned short int)
- unsigned int (oppure unsigned)
- unsigned long (oppure unsigned long int)

Intervallo di definizione: da 0 a $2^n - 1$, dove n dipende dal compilatore.

Il numero n di bit è come per i corrispondenti interi con segno.

Le costanti definite in limits.h sono:
USHRT_MAX, UINT_MAX, ULONG_MAX
(si noti che il minimo è sempre 0)

Notazione per le costanti :

- decimale: come per interi con segno
- esadecimale: 0xA, 0x2F4B, . . .

Nel codice si può far seguire le cifre dallo specificatore u (ad esempio 10u).

Tipi interi (ancora tipi senza segno)

Ingresso/uscita: tramite printf e scanf, con i seguenti specificatori di formato:

%u per numeri in decimale
%x per numeri in esadecimale con cifre 0, . . . , 9, a, . . . , f
%X per numeri in esadecimale con cifre 0, . . . , 9, A, . . . , F

Per interi short si antepone h
Per interi long si antepone l (minuscola)

Attenzione alla conversione tra signed e unsigned in operazioni miste:
I valori signed vengono **sempre promossi** a unsigned.

Tipi interi: caratteri

Servono per rappresentare caratteri alfanumerici.
Attraverso un opportuno codice: il codice ASCII (American Standard Code for Information Interchange).

Un codice associa ad ogni carattere un intero.

In C i caratteri possone anche essere usati come gli interi.

3 tipi:

- char,
- signed char,
- unsigned char.

I tipi signed char e unsigned char hanno lo stesso tipo di rappresentazione degli interi.

Tabella ASCII

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Tipi interi: caratteri

Intervallo di definizione:

dipende dal compilatore

Vale che: sizeof(char) ≤ sizeof(int)

Tipicamente i caratteri sono rappresentati con 8 bit.

Operatori: sono gli stessi di **int** (operazioni effettuate utilizzando il codice del carattere).

Tipi interi: caratteri

Acluni caratteri necessitano una ‘sequenza speciale per essere scritti;

Nome del carattere	In **C**	Valore intero
alert	`\a`	7
backslash	`\\`	92
backspace	`\b`	8
carriage return	`\r`	13
double quote	`\"`	34
formfeed	`\f`	12
horizontal tab	`\t`	9
newline	`\n`	10
null character	`\0`	0
single quote	`\’`	39
vartical tab	`\v`	11
question mark	`\?`	63

Tipi reali

I reali vengono rappresentati in virgola mobile (floating point).

Tre tipi:

- **–** float
- **–** double
- **–** long **double**

Intervallo di definizione:

sizeof(float) < sizeof(double) < sizeof(long double)
4 8 16

Costanti: con punto decimale o notazione esponenziale

Esempio: double x, y, z, w;
x = 123.45;
y = 0.0034;
y = .0034;
z = 34.5e+20;
z = 34.5E+20;
w = 5.3e-12;

Tipi reali

Nei programmi, per denotare una costante di tipo

– float, si può aggiungere f o F finale

Esempio: float x = 2.3e5f;

– long double, si può aggiungere L o l finale

Esempio: long double x = 2.34567e520L;

Operatori: stessi che per gli interi (tranne “%”)

Tipi reali

Ingresso/uscita: tramite printf e scanf, con diversi specificatori di formato

Output con printf (per float):

– %f

– %m.nf m cifre complessive, di cui n cifre decimali (Esempio: %8.3f)

– %e (oppure %E) notazione esponenziale

– %m.ne m cifre complessive, di cui n cifre decimali (Esempio: %10.3f)

– %g (oppure %G) forma ottimizzata
come %e se l'esponente è < −4 oppure il numero è ≥ precisione specificata
altrimenti come %f

Input con scanf (per float):

– si può usare indifferentemente %f, %e, o %g.

Tipi reali

Riassunto degli specificatori di formato per i tipi reali:

	float	**double**	**long double**
printf	%f,%e,%g	%f,%e,%g	%Lf,%Le,%Lg
scanf	%f,%e,%g	%lf,%le,%lg	%Lf,%Le,%Lg

.

Tipi reali: rappresentazione a virgola mobile

I valori che una variabile a virgola mobile può assumere sono descritti in termini di

Precision: il numero di decimali significativi che possono essere rappresentati

Range: il più grande e il più piccolo valore positivo che una variabile di quel tipo può assumere

Tipi reali: rappresentazione a virgola mobile

Su molte macchine un **float** ha:

precision di 6 cifre significative

range approssimativo da 10^{-38} a 10^{38}

Un float positivo è rappresentato approssimativamente (nel sistema decimale) come

$0.d_1\ d_2\ d_3\ d_4\ d_5\ d_6 \times 10^n$

d_i sono digit decimali

$-38 \leq n \leq 38$

Tipi reali: rappresentazione a virgola mobile

Su molte macchine un **double** ha:

precision di 15 cifre significative

range approssimativo da 10^{-308} a 10^{308}

Un double positivo è rappresentato approssimativamente (nel sistema decimale) come

$0.d_1\ d_2\ d_3\ d_4\ d_5\ d_6 \times d_7 ... \times d_{15}\ x10^{n}$

d_i sono digit decimali

$-308 \leq n \leq 308$

Tipi reali: rappresentazione a virgola mobile

ATTENZIONE

– **Non tutti i numeri reali sono rappresentabili**

– Operazioni fra valori a virgola mobile non sono esatte, a differenza delle operazioni fra interi

– Non un grosso problema per calcoli di piccola taglia

– Bisogna stare attenti per problemi numerici:
- risoluzione di grossi sistemi di equazioni differenziali
- minimizzazione di funzionali complessi

– **Dominio dell'analisi numerica**

Conversione tra tipi: type-casting

Situazioni in cui si hanno conversioni di tipo

In maniera implicita:
– quando in un’espressione compaiono operandi di tipo diverso

– durante un’assegnazione x = y, quando il tipo di y è diverso da quello di x

– nel passaggio dei parametri a funzione
– attraverso il valore restituito da una funzione

In maniera esplicita:

– tramite l’operatore di cast

Conversione tra tipi: type-casting

Conversioni implicite tra operandi di tipo diverso nelle espressioni

short → int → long → float → double → long double
char → int

Conversione da sinistra verso destra.

Esempio: int x; double y;

Nel calcolo di (x+y):

1. x viene convertito in double
2. viene effettuata la somma tra valori di tipo double
3. il risultato è di tipo double

Inoltre: signed (short/long) viene convertito in unsigned (short/long)
N.B. Se il valore signed è negativo, il valore unsigned risultante sarà un numeropositivo con bit più significativo pari a 1 .

Conversione tra tipi: type-casting

Conversioni sicure e non

– Le conversioni viste prime sono sicure: **non si perde il valore**.

Eccezione: da long a double ci può essere perdita di precisione.

– Se invertiamo le conversioni sicure si può perdere il valore:

short ← int ← long ← float ← double ← long double

char ← int

– ci può essere overflow

– da reali a interi si ha troncamento della parte frazionaria

Conversioni in assegnazione

– Il risultato viene sempre convertito nel tipo della variabile a sinistra (con eventuale overflow o troncamento).

• Se la conversione non è possibile si ha errore.

Esempio:

```
int i; float x = 2.3, y = 4.5;
i = x + y;  printf("%d", i);
/* stampa 6 */
```

Conversione tra tipi: type-casting

Conversioni esplicite (operatore di cast)

Sintassi:

(tipo)espressione

Converte il valore di espressione nel corrispondente valore del tipo specificato.

Esempio:

```
int somma=205, n=12;
float media;

media = somma / n; /* divisione tra interi */
printf(“media:%f\n”,media);
media = (float)somma / n; /* divisione tra reali */
printf(“media:%f\n”,media);
```

Indirizzi di memoria

Posso specificare una variabile che contiene un indirizzo di memoria.

Sintassi:

*(tipo *)nomeVariabile, oppure tipo * nomeVariabile*

Tipo:

- char, int, float, double … qualunque tipo
- nomeVariabile: il nome di una variabile

Posso accedere all'indirizzo di memoria di una:

Sintassi:

&nomeVariabile

Uscita (printf)*: %p*

Indirizzi di memoria

ESEMPIO:

```
int main() {
        int x=10;

        printf(“Valore:%d Dimensioni:%lu Indirizzo:%p”,x,sizeof(x),&x);

        return 0;
}
```