Fondamenti di Informatica

Ingegneria Clinica

Lezione 22/10/2009

Prof. Raffaele Nicolussi

FUB - Fondazione Ugo Bordoni

Via B. Castiglione 59 - 00142 Roma

Docente	**Raffaele Nicolussi**	*rnicolussi@fub.it* *0654803323*
Lezioni Aula 54 (ex aula 4) Via del Castro Laurenziano, 7	**Lunedì, Giovedì, Venerdì**	**12:00 – 13:30**
Esercitazioni Aula 15 Via Tiburtina, 205	**Giovedì**	**14:00 – ???**
Ricevimento:	**Per appuntamento**	**in FUB, per email, per telefono**
Sito web:	**http://w3.uniroma1.it/IngClinFondinf**	

Problema: Stampare il messaggio “Salve mondo!”

```
#include <stdio.h>

/* Un primo programma in C */

int main(void)
{
    printf(“Salve mondo! \n”);
}
```

Problema: Quadrato di un intero

```
#include <stdio.h>
/* Prodotto di due interi */


int square (int num )
{
      int prodotto;
      prodotto=num*num;
      return prodotto;
}
```

Le funzioni

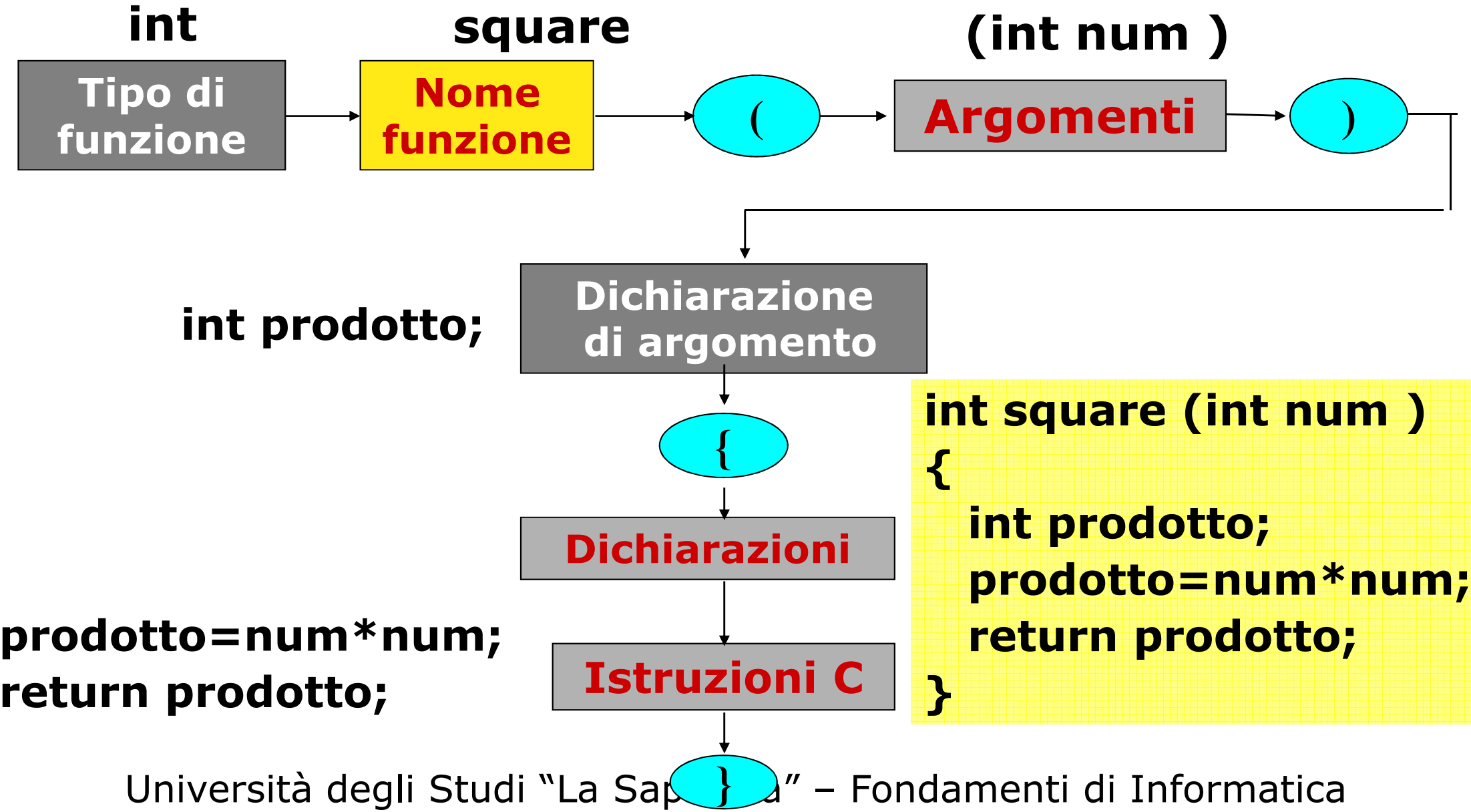
- **main** e **square** sono nomi di funzioni

Un programma C è formato da funzioni

- Una funzione è formata da **un blocco di istruzioni**, strutturate in base ad una precisa sintassi
- Una funzione può:
  - produrre un risultato a fronte di elaborazioni sui dati di ingresso
  - seguire delle operazioni che non producono output

- Ogni programma eseguibile deve contenere la funzione speciale main
  - indica il punto da cui inizia l'esecuzione del programma
  - La funzione main determina il flusso del programma, ossia l'ordine di esecuzione di istruzioni e funzioni

Struttura di una funzione C



printf("Salve, mondo!\n");

- **Istruzione**
- **Notare che termina con ;**
- **printf è funzione di stampa**
  - **→Proviene da una libreria del C, detta stdio.h**
- **visualizza sullo schermo la stringa di caratteri tra virgolette**
  - **→Le virgolette non sono stampate, servono solo per delimitare la stringa**

- **\n :carattere di escape che indica un ritorno a capo (newline)**

Altri caratteri di escape

\n	newline
\t	tabulazione
\r	posiziona il cursore all'inizio riga corrente
\a	emissione suono
\\	visualizzazione del carattere \
\'	visualizzazione apice
\"	visualizzazione virgolette

include <stdio.h>

- direttiva C, sempre preceduta dal simbolo #
- in C ci sono altre direttive, che sono "istruzioni" al di fuori dal linguaggio
- sono comandi per il pre-processore:
  - le righe che iniziano con # sono elaborate dal pre-processore prima che il programma sia compilato
- questa direttiva *include* un file di libreria

include

- fa usare al compilatore un file diverso da quello sorgente che sta compilando
- consente di inserire i contenuti di altri file
- i file sono "inclusi" prima della compilazione
- i file da includere possono essere libreria di sistema oppure file scritti da noi
- sono infatti presenti due formati
  - #include <*nomefile*> : il pre-processore cerca il file in una directory speciale (definita dal sistema operativo)
  - #include "*nomefile*" : il pre-processore cerca il file nella directory dove è il file sorgente: si può quindi specificare un *path*. Se non viene trovato, cerca nella directory speciale

Libreria di run-time

- Tipicamente i file inclusi sono file di libreria
- I file di libreria contengono funzioni che svolgono operazioni fondamentali
- Queste funzionalità sono ottimizzate rispetto alle istruzioni del linguaggio
- I file da includere sono file oggetto, ossia già compilati
- Contengono classi di funzioni relative a uno o più servizi
  - I/O (Input/Output)
  - gestione della memoria
  - operazioni matematiche
  - manipolazione di stringhe
- Per ogni classe di funzioni esiste un file sorgente, detto file header
- I file header hanno l'estensione .h (come stdio.h)
  - Contengono le informazioni su come utilizzare le funzioni

<stdio.h>

- è un file di intestazione (estensione .h) per la gestione standard dell'input/ouyput
- printf è una funzione appartenente a questa libreria

Come è fatto un programma C

- Un programma C è costituito da una o più funzioni, ciascuna dedicata alla risoluzione di una parte del problema complessivo
- Una funzione viene progettata a partire dalla decomposizione del problema iniziale in sottoproblemi (progettazione top-down)
- Un programma C contiene sempre la funzione main
  - L'esecuzione parte sempre da questa funzione
  - main determina il flusso del programma
  - può essere anche la sola funzione del programma
- Tutte le funzioni in un programma C, compresa la funzione main, seguono la stessa sintassi (stesse regole di formazione)

Come è fatto un programma C (2)

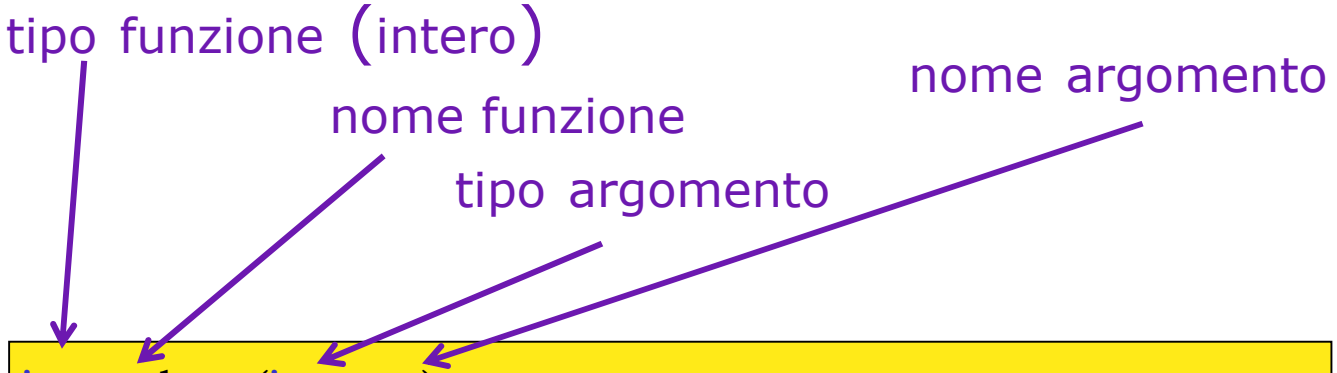
- Una funzione C è un blocco d'istruzioni nel linguaggio C
  - Svolge un'operazione complessa (rispetto alle istruzioni semplici) che può essere attivata tramite il nome della funzione
- La funzione main non può essere chiamata

Problema: Stampare il quadrato di 789

```
#include <stdio.h>
/* Programma che calcola il quadrato di 789 */
int main(void)
{
      int q;
      int c=789;
      q= quadrato(c);
      printf("Il quadrato di %d e\' pari a %d \n", c, q);
      return 0;
}

/* Funzione per il quadrato di un intero */
int quadrato (int num )
{
      int prodotto;
      prodotto=num*num;
      return prodotto;
}
```

Analisi funzione square




```
int quadrato (int num )
    {
        int prodotto;
        prodotto=num*num;
        return prodotto;
    }
```

corpo funzione

prodotto=num*num;

- istruzione di assegnazione
- espressione nella parte destra
- variabile nella parte sinistra

return prodotto;

- restituisce il valore assegnato a prodotto
- ritorna il controllo alla funzione che ha effettuato la chiamata

Programma: sommare due numeri letti da tastiera


```
/* Programma di addizione */
#include <stdio.h>
int main (void)
{
    int num1, num2, somma;          /*dichiarazioni*/

    printf("Primo intero\n");       /* prompt */
    scanf("%d", &num1);     /* lettura intero*/
    printf("Secondo intero\n");     /* prompt */
    scanf("%d", &num2);     /* lettura intero*/
    somma=num1+num2;
                                    /* somma e assegnamento*/
    printf("Somma: %d\n", somma);
                                            /* stampa somma*/

    return 0;
            /* programma terminato con successo*/
}
```

- notare indentazione e spazi

```
int num1, num2, somma;
```

- dichiarazioni variabili
  - **Variabile:** locazione di memoria per memorizzare valori
- le variabili vanno sempre dichiarate con
  - nome
  - tipo di dato
- nome di variabile deve essere un identificatore valido per il linguaggio
  - **identificatore:** sequenza di caratteri (lettere, numeri e segno di underscore _) che non deve iniziare con un numero
  - può avere lunghezza qualsiasi (max di 31 caratteri per lo standard ANSI)
- il C è case-sensitive (A diverso da a)
  - Somma, somma, SommA, SOMMA, etc.
- le dichiarazioni vanno dopo la parentesi graffa e prima di qualsiasi istruzione eseguibile

```
printf("Primo intero\n");
printf("Secondo intero\n");
printf("Somma: %d\n", somma);
```

- funzione di *stdio*
- manda i dati dallo standard output (video)
  - Lo *standard output* è il device standard su cui viene inviata l'uscita
- "%d", stringa di controllo del formato: tipo di dato da stampare (%d indica un intero)
- La stringa tra virgolette è l'oggetto da stampare
  - Al suo interno possono essere presenti caratteri di controllo, come la stringa di controllo %d o caratteri speciali che servono per formattare l'output
  - Consiste di due parti
    - La prima parte indica la forma dell'input e il tipo degli elementi
      - "Somma: %d\n"
    - La seconda contiene le variabili
      - somma

```
scanf("%d", &num1);
scanf("%d", &num2);
```

- funzione di *stdio*
- prende i dati dallo standard input (tastiera)
  - Lo *standard input* è il device standard da cui viene prelevato l'ingresso
- "%d", stringa di controllo del formato: tipo di dato da immettere (%d indica un intero)
- &num1, &num2: la variabile da leggere deve essere preceduta dal carattere & (ampersand).
- &num1, &num2: indica la locazione di memoria dove memorizzare la variabile (*num1* e *num2*)
- Due parti
  - La prima parte indica la forma dell'input e il tipo degli elementi
    - "%d"
  - La seconda le variabili
    - &num1

NOTA: nella scanf le variabili sono sempre precedute da &

Tipi di dati

- **Tipo di dato è definito da**
  - Insieme dei valori che può assumere
  - Operazioni ammissibili

Più formalmente

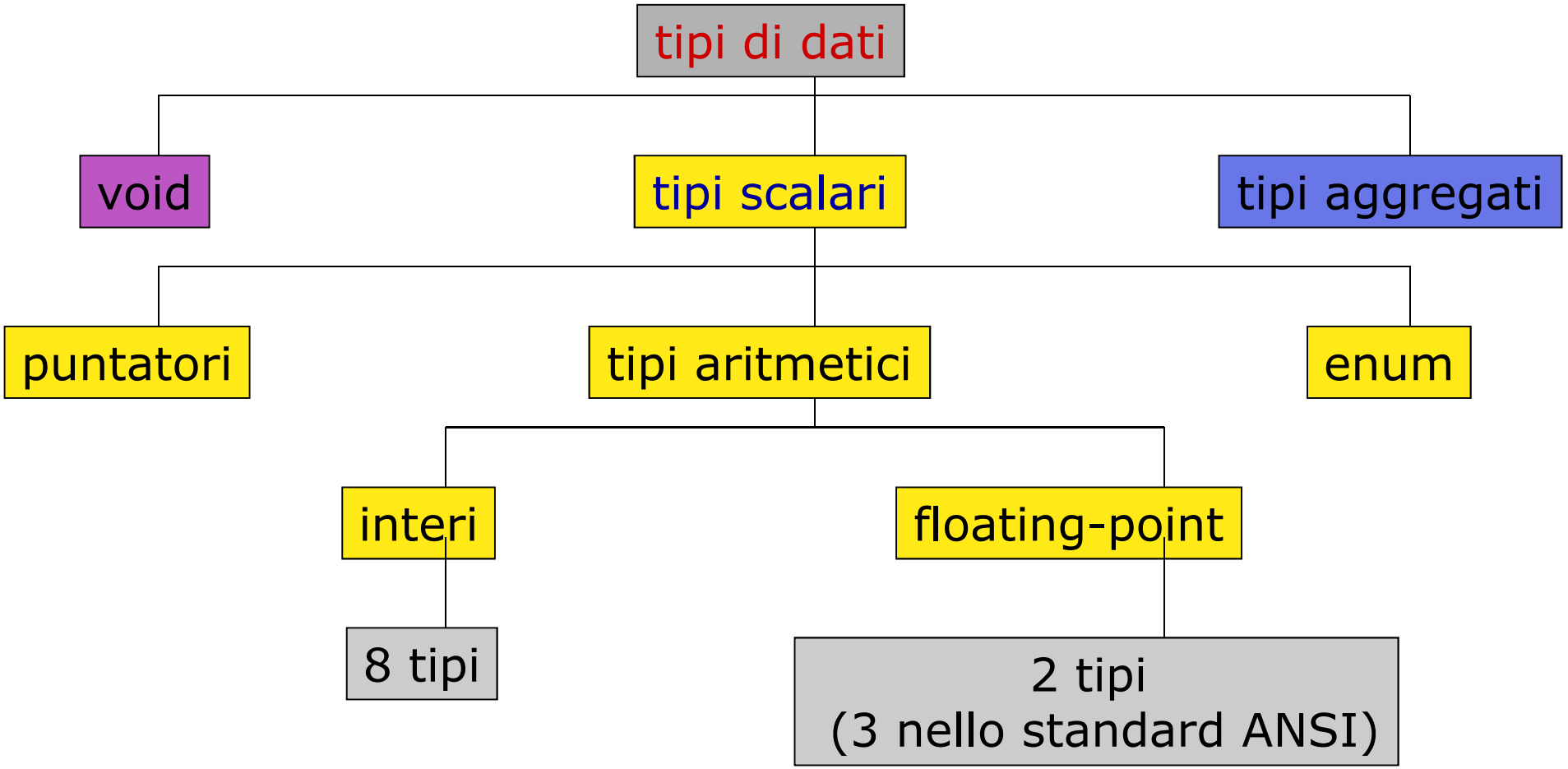
Un **tipo di dato T** è definito da:

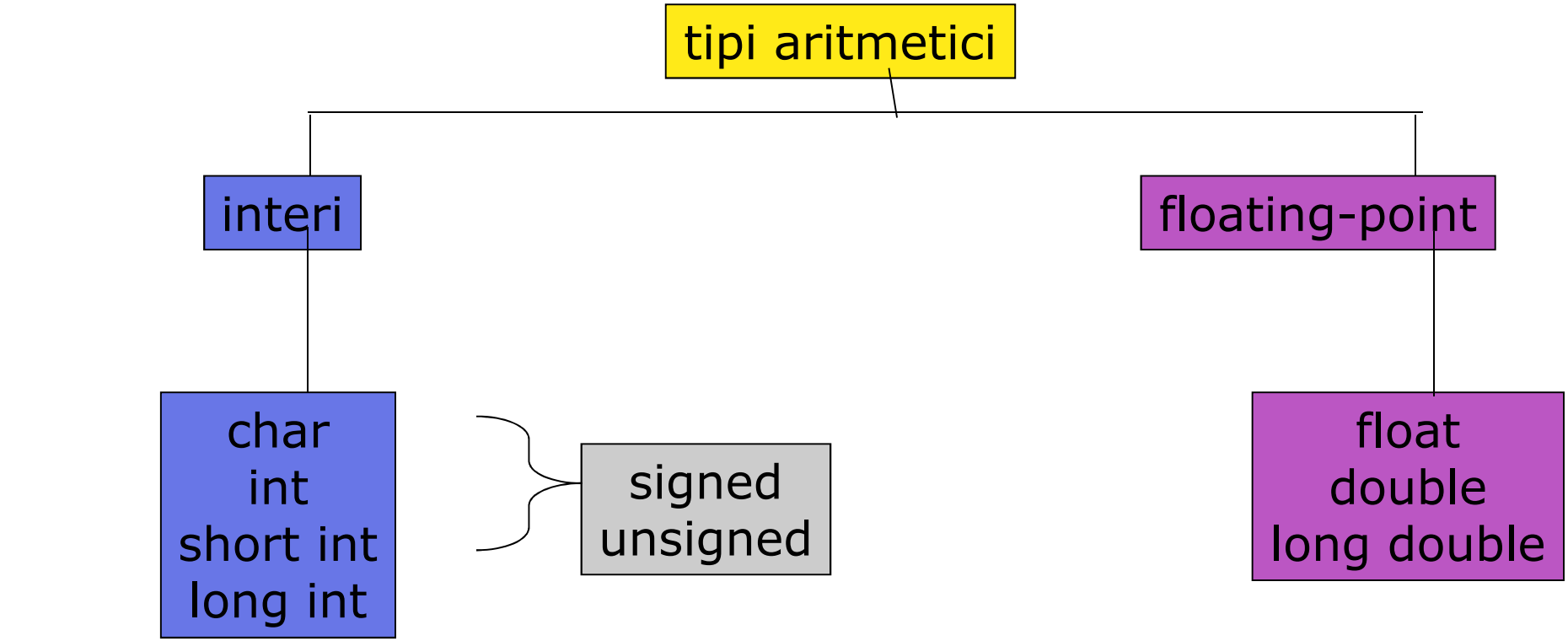
- Un dominio di valori **D**
- Funzioni $\mathbf{f_1, \ldots, f_n}$ e predicati $\mathbf{p_1, \ldots, p_m}$ (detti globalmente *operatori*)
- Alcune costanti $\mathbf{c_1, \ldots, ck}$
- **Le funzioni agiscono su uno o più valori del dominio D per produrre un valore (eventualmente non appartenente a D), mentre i predicati agiscono su uno o più valori del dominio D per produrre un valore booleano (vero o falso)**

- L'associazione **tipo-dato** limita gli errori perché protegge il programmatore da associazioni illogiche tra dati e operatori (non presente a livello macchina)

Tipi di dati scalari

- Due classi di tipi
  - tipi scalari (è definito un ordinamento)
  - tipi aggregati (si ottengono dagli scalari)





Parole chiave

char
int
float
double
enum

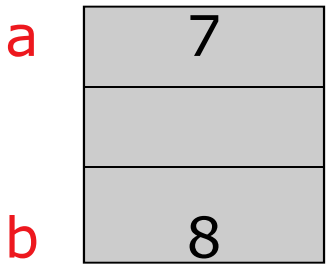
Qualificatori

short
long
signed
unsigned

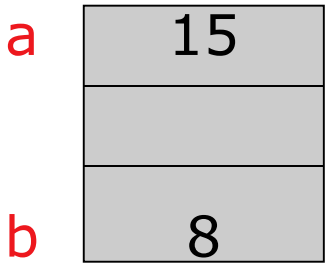
Rappresentazione in memoria

- I nomi di variabile (qualunque sia il loro tipo) fanno riferimento a locazioni (posizioni) di memoria
- Una variabile è caratterizzata da
  - Nome
  - Tipo
  - Valore
- Una variabile corrisponde ad una cella di memoria indirizzata simbolicamente dal suo identificatore
- La cella contiene il valore della variabile
- La modalità di rappresentazione del valore corrisponde al suo tipo

int a=7, b=8;



a= a+b;



Tipi interi (2)

int

- è il tipo intero di base
- la sua dimensione dipende dal calcolatore (1, 2, 3, 4 byte), lo standard ANSI richiede che sia almeno 16 bit (2 byte)

short int (o semplicemente short)

- memorizzazione degli interi generalmente con 2 byte

long int (o semplicemente long)

- memorizzazione degli interi generalmente con 4 byte

- Ad un diverso numero di bit allocati corrisponde un diverso intervallo di valori memorizzabili:
  - 4 byte da -2^{31} a $2^{31}-1$
  - 2 byte da -2^{15} a $2^{15}-1$
  - Un bit è utilizzato per il segno

Tipi interi (2)

Interi senza segno: specificatore unsigned

- si può specificare un tipo intero con solo valori non negativi
- viene raddoppiato l'intervallo di valori dato che il bit più significativo non rappresenta il segno
  - 4 byte da 0 a $2^{32}-1$
- se non specificato, l'intero è con segno (signed)

Tipi interi (3)

char

- **i caratteri fanno parte del tipo intero**
- **sono memorizzati come codici numerici secondo codifiche definite**
  - ASCII (American Standard Code for Information Interchange)
  - EBCDIC (usato IBM)
- **in genere sono usati 8 bit (da 0 a 2^8 -1 unsigned, da -2^7 a 2^7 -1 signed)**
  - **il default dipende dal compilatore, in genere è signed**
- **char c;**
  - **c='A';**
  - **c=65;**
    - **in entrambi i casi a c corrisponde il valore 65**
- **char a, b;**
  - **a=5;** **(assegnato il valore 5)**
  - **a='5';** **(assegnato il valore 53, codice ASCII del carattere "5")**
- **int j;**
  - **j='A' + 'B';** 66, 131, 65

Tavola ASCII

http://www.asciitable.com/

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com