Laboratorio di Calcolatori 1

Corso di Laurea in Fisica
A.A. 2007/2008

Dott.Davide Di Ruscio

Dipartimento di Informatica
Università degli Studi di L'Aquila

Lezione del 14/04/08

Sommario (II parte)

Il Linguaggio C

- Caratteristiche generali
- Un linguaggio C semplificato ed esempi di semplici programmi
- Struttura di un programma C
- Direttive del pre-processore
- Parte dichiarativa:
  - tipi
  - definizioni di tipi
  - definizioni di variabili
- Parte esecutiiva
  - istruzione di assegnamento
  - istruzioni (funzioni) di input-output
  - istruzioni di selezione
  - istruzioni iterative
- Vettori mono e multidimensionali
- Funzioni e procedure
- File
- Allocazione dinamica di memoria
- Suddivisione dei programmi in piu' file e compilazione separata
- Algoritmi elementari
  - ricerca sequenziale e binaria
  - ordinamento di un vettore: per selezione, per inserimento, per fusione e a bolle
- Aspetti avanzati di programmazione
  - ricorsione
  - strutture dati dinamiche

RIFERIMENTI

Ceri, Mandrioli, Sbattella

Informatica arte e mestiere

McGraw-Hill

Nota

Questi lucidi sono tratti dal materiale distribuito dalla McGraw-Hill e basati su del materiale fornito dal Prof. Flammini Michele

Il “vero” linguaggio C

- Che cosa manca per poter “far girare i programmi”?
- Primi esempi … “che girano”
- La (nuova) struttura di un programma C
- La parte dichiarativa
- L’ I/O

Primi esempi… “che girano” (1)

```
                    /* PrimoProgrammaC */

#include <stdio.h>

main()
{
   printf("Questo è il mio primo programma in C\n");

}
```

N.B.: niente dichiarazioni!

Primi esempi… “che girano” (2)

```
                                /* Programma SommaDueInteri */
#include <stdio.h>

main()
{
    int  a, b, somma;

    printf (“Inserisci i due addendi: ”);
    scanf("%d%d", &a, &b);
    somma = a + b;
    printf("La somma di a+b è:\n%d \nArrivederci!\n",
    somma);
}
```

- Se vengono inseriti i dati 3 e 5, l’effetto dell’esecuzione del programma sullo Standard Output è il seguente:

  La somma di a+b è:
  8
  Arrivederci!

- Se fossero stati omessi i primi due simboli \n nella stringa di controllo?

Primi esempi… “che girano” (3)

```
                /* Programma SommaSequenza */
#include <stdio.h>
main()
{
    int  numero, somma;

    somma = 0;
    scanf("%d", &numero);
    while (numero != 0)
    {
        somma = somma + numero;
        scanf("%d", &numero);
    }
    printf("La somma dei numeri digitati è: %d\n", somma);
}
```

Struttura di un programma C

[Direttive preprocessore]

[Parte dichiarativa globale]

```
main ()
{
    [Parte dichiarativa locale]

    [Parte esecutiva]

}
```

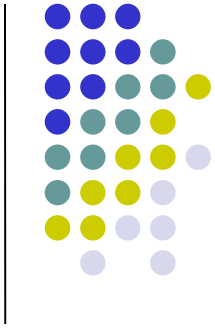
In particolare …

- Un programma C deve contenere, nell'ordine:
  - Una parte contenente **direttive** per il compilatore. Per il momento trascuriamo questa parte.
  - Una **parte dichiarativa globale.**
  - L'identificatore predefinito `main` seguito dalle parentesi `()`
  - Due parti, racchiuse dalle parentesi **`{}`**:
    - la **parte dichiarativa locale**;
    - la **parte esecutiva**.

- Le parti dichiarative elencano tutti gli elementi che fanno parte del programma, con le loro principali caratteristiche. Per il momento considereremo solo quella locale.

- La parte esecutiva consiste in una successione di istruzioni come già descritto in pseudo-C.

La parte dichiarativa



- Tutto ciò che viene usato va dichiarato. In prima istanza:
  - Dichiarazione delle costanti
  - Dichiarazione delle variabili

- Perché questa fatica … inutile?
  - Aiuta la **diagnostica** (ovvero *segnalazione di errori*):

    x = alfa;

    al*b*a = alfa + 1;
  - Senza dichiarazione, alba è una nuova variabile!

- Principio importante:
  meglio un po' più di fatica nello scrivere un programma che nel leggerlo e capirlo!

Dichiarazioni di variabili

- Una dichiarazione
  - Riserva nomi o “identificatori” per le variabili
  - Dichiara il tipo delle variabili associate a quegli identificatori

- La dichiarazione delle variabili è obbligatoria

- Una dichiarazione di variabile ha la seguente sintassi:

  tipovar nomevar1,nomevar2,…;

- Qundi una dichiarazione di variabile consiste in:
  - Uno **specificatore di tipo**, seguito da
  - Una lista di uno o più nomi o “identificatori” di variabili separati da una virgola
  - Ogni dichiarazione termina con ‘;’

- Il tipo di una variabile può essere uno dei predefiniti del C (intero, reale, carattere, …) o, come vedremo di seguito, definito appositamente dall'utente

- L'identificatore di una variabile deve iniziare per lettera (includendo tra le lettere il simbolo '_') e può essere seguito da una qualsiasi sequenza di lettere o cifre

Esempi di dichiarazioni di variabili

```
float     x,y,z1w,z2w;
int       i,j;
char      simb;
```

Se un identificatore di variabile *x* è dichiarato di tipo ***int***, esso potrà essere usato nella parte esecutiva solo come tale. Di conseguenza, *x* non assumerà mai un valore reale

equivalenti a:

```
float     x,z1w;
int       i,j;
char      simb;
float     y,z2w;
```

Dichiarazioni di costanti

- Alcuni esempi di dichiarazioni di costanti:

```
const    float    PiGreco    = 3.14;
const    float    PiGreco    = 3.1415, e = 2.718;
const    int      N = 100, M = 1000;
const    char     CAR1 = 'A', CAR2 = 'B';
```

- Un eventuale assegnamento a una costante sarebbe segnalato come errore dal compilatore.

```
AreaCerchio = PiGreco*RaggioCerchio*RaggioCerchio;
```

è equivalente a:

```
AreaCerchio = 3.14*RaggioCerchio*RaggioCerchio;
```

(se si fa riferimento alla prima dichiarazione di PiGreco)

- Importanza delle dichiarazioni di costanti:
  - Leggibilità: meglio identificatori simbolici che valori numerici
  - Correttezza: i valori delle costanti vengono specificati in un unico punto invece che in diversi punti del programma
  - Modificabilità: se bisogna migliorare l'accuratezza, ad esempio usando come approssimazione del valore reale 3.1415 invece di 3.14, basta modificare solo la dichiarazione

Tipi di dati

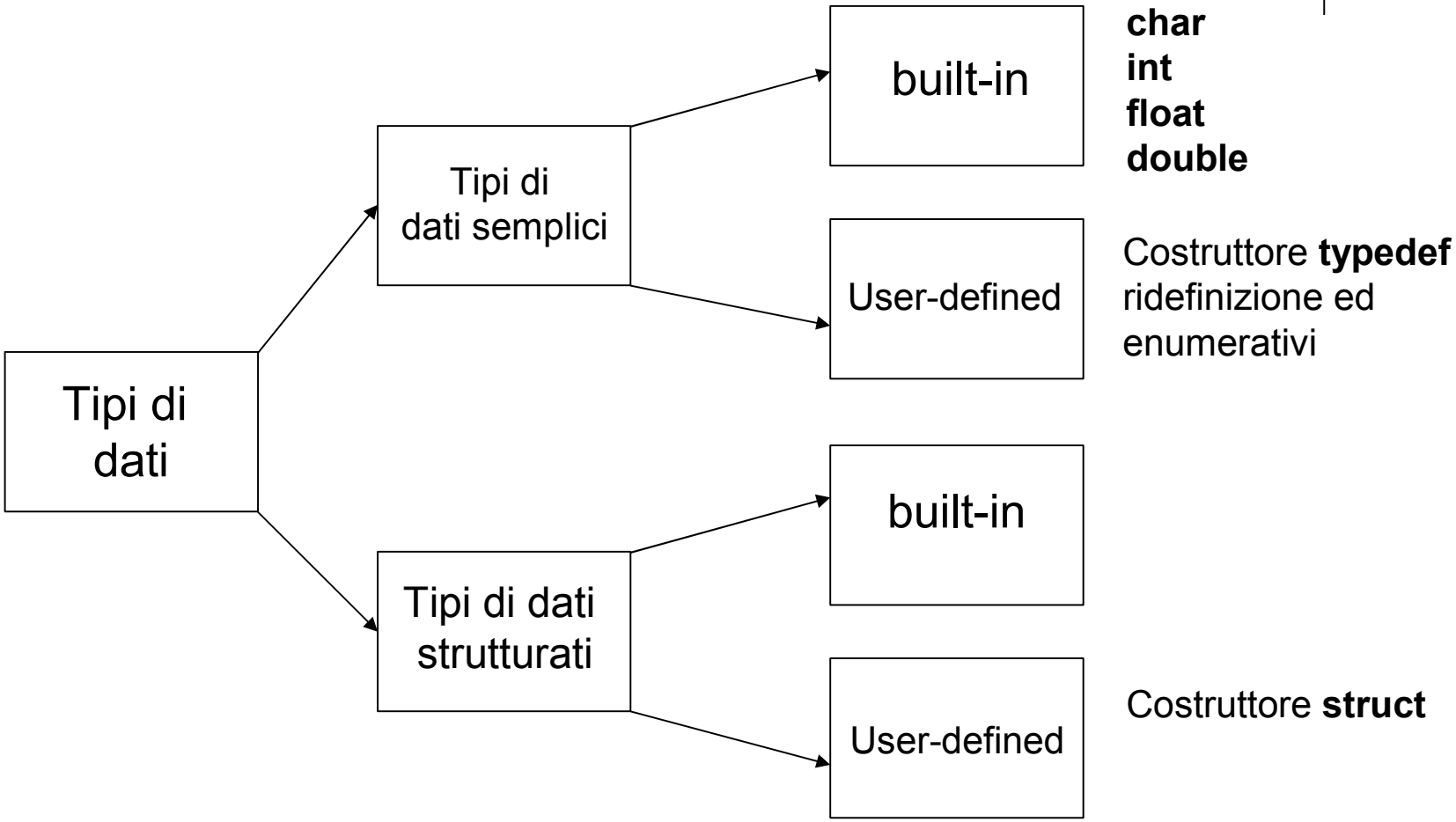
- Definizione del concetto di **Tipo**
  - Il tipo è definito da una collezione di valori ed un insieme di operazioni applicabili su di essi
- Ogni tipo di dato ha una propria rappresentazione in memoria, realizzata tramite un opportuna codifica che sfrutta un certo numero di celle
- In C tutte le variabili hanno un tipo
  - Associato in fase dichiarazione
  - Non modificabile
- Questo sistema di tipizzazione implica che
  - Si determinano i valori ammissibili e le operazioni possibili sulle variabili
  - Si determina a priori l'occupazione di memoria
  - Si rilevano alcuni errori durante la fase di compilazione

- Tipi di dati semplici
  - Rappresentano informazione semplice (numeri, caratteri, …)
  - Occupano poca memoria
- Tipi di dati strutturati
  - Rappresentano informazione costituita dall’aggregazione di varie componenti: (Es. Data, impiegato)
  - Array rappresenta informazione complessa costituita da una sequenza di elementi omogenei
  - Componenti accessibili singolarmente

- Tipi esistenti (built-in)
  - Tipi base messi ha disposizione dal linguaggio C
  - Char, int, float, ….

- Tipi definiti dall'utente (user-defined)
  - Nuovi tipi definibili dall'utente ed affiancabili a quelli già esistenti (Es. data, fattura, impiegato)
  - Costruttore di tipo: **typedef**, **struct**

Schema tipi di dati



Tipi semplici predefiniti (1/2)

- Il linguaggio C prevede 4 tipi base:
  - **Char**
  - **Int**
  - **Float**
  - **Double**

- Dotati dei qualificatori di tipo (da premettere al nome del tipo nelle dichiarazioni)
  - **Signed** e **unsigned** applicabili a **char** ed **int**
  - **Short** e **long** applicabili al tipo **int**
  - **Long** applicabile al tipo **double**

Il tipo int

- Approssimazione del corrispondente tipo matematico: non è infinito
- L'insieme dei valori dipende dalla macchina
  - Esempio
    - **Short int** 16 bit, forse meno di un **int**
    - **Int** 16 o 32 bit dipendentemente dalla macchina
    - **Long int** 32 bit
- Vale comuque sempre che:
  - Bit allocati (**short int**) ≤ Bit allocati (**int**) ≤ Bit allocati (**long int**)

- **Signed int**
  - Un bit viene usato per rappresentare il segno
- **Unsigned int**
  - Tutti i bit sono usati per rappresentare il numero positivo
- Vale comunque:
  - Bit allocati (**unsigned int**) = bit allocati (**signed int**)
  - Se non specificato il tipo è **signed**
- Rappresentazione del minimo e del massimo simbolica:
  - **INT_MIN** e **INT_MAX:** identificatori di costanti predefiniti dall'implementazione del linguaggio inclusi nell'header <limits.h>

Il tipo int

- Esempio:
  - **Signed short int**
    - $\{-2^{15}, \ldots, 2^{15}-1\}$
  - **Signed int**
    - $\{-2^{15}, \ldots, 2^{15}-1\}$
  - **Signed long int**
    - $\{-2^{31}, \ldots, 2^{31}-1\}$
  - **Unsigned short int**
    - $\{0, \ldots, 2^{16}-1\}$
  - **Unsigned int**
    - $\{0, \ldots, 2^{16}-1\}$
  - **Unsigned long int**
    - $\{0, \ldots, 2^{32}-1\}$

- Esempi di operatori
  - = assegnamento
  - + , - , * , /
  - % Resto della divisione intera
  - == relazione di uguaglianza
  - != relazione di diversità
  - < relazione di minore
  - > relazione di maggiore
  - <= relazione di minore uguale
  - >= relazione di maggiore uguale
  - & and bit a bit
  - | or bit a bit
  - ...

I tipi float e double

- Approssimazione dei numeri reali della matematica
  - Limite e precisione

- Virgola mobile:
  - La rappresentazione in virgola mobile consiste in due parti: **mantissa** ed **esponente** (della base 10) separate dal carattere E
  - Es:
    - 1345000 E1
    - 0,0008768 E-2

- Limite e precisione dipende dallo spazio allocato
  - Il tipo double rappresenta i reali ma in generale ha precisione e limiti “migliori”

  - Bit allocati (**float**) ≤ bit allocati (**double**) ≤ bit allocati (long **double**)

- Esempio:
  - **Float**
    - **4 byte**
    - Accuratezza 6 cifre
    - Range 10^{-38} 10^{+38}
  - **double**
    - **8 byte**
    - Accuratezza 15 cifre
    - Range 10^{-308} 10^{+308}
  - **Long double**
    - Spesso ha stesso spazio di un double

- Esempi di operatori
  - = assegnamento
  - + , - , * , /
  - == relazione di uguaglianza
  - != relazione di diversità
  - < relazione di minore
  - > relazione di maggiore
  - <= relazione di minore uguale
  - >= relazione di maggiore uguale
  - Sin(x) seno
  - Cos(x) coseno
  - Sqrt(x) radice quadrata di x
  - Pow(x,y) x elevato alla y
  - ……

Tutte incluse
Nella libreria
#include <Math.h>

Il tipo char

- Generalmente è l'insieme dei caratteri ASCII
  - Lettere maiuscole, minuscole
  - Simboli #@*+?!£"$%
  - Cifre 1234567890
  - Caratteri di controllo

- Nella codifica ASCII ogni carattere ha associato un numero
  - Definisce un ordinamento

- Particolari caratteri detti di controllo
  - Non stampano simboli sul video o sulla carta
  - Esempio '\n' controllo che consente di effettuare un new line o a capo
  - \b = "backspace", \t="horizontal tab",…

- ANSI C alloca lo spazio di **1 byte** per i caratteri

- In C i caratteri sono visti come interi

0		32		64	@	96	`	128	Ç	160	á	192	└	224	Ó
1	☺	33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
2	☻	34	"	66	B	98	b	130	é	162	ó	194	┬	226	Ô
3	♥	35	#	67	C	99	c	131	â	163	ú	195	├	227	Ò
4	♦	36	$	68	D	100	d	132	ä	164	ñ	196	─	228	õ
5	♣	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	Õ
6	♠	38	&	70	F	102	f	134	å	166	ª	198	ã	230	µ
7	•	39	'	71	G	103	g	135	ç	167	º	199	Ã	231	þ
8	◘	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Þ
9	○	41	)	73	I	105	i	137	ë	169	®	201	╔	233	Ú
10	◙	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Û
11	♂	43	+	75	K	107	k	139	ï	171	½	203	╦	235	Ù
12	♀	44	,	76	L	108	l	140	î	172	¼	204	╠	236	ý
13	♪	45	-	77	M	109	m	141	ì	173	¡	205	═	237	Ý
14	♫	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	¯
15	☼	47	/	79	O	111	o	143	Å	175	»	207	¤	239	´
16	►	48	0	80	P	112	p	144	É	176	░	208	ð	240	-
17	◄	49	1	81	Q	113	q	145	æ	177	▒	209	Ð	241	±
18	↕	50	2	82	R	114	r	146	Æ	178	▓	210	Ê	242	‗
19	‼	51	3	83	S	115	s	147	ô	179	│	211	Ë	243	¾
20	¶	52	4	84	T	116	t	148	ö	180	┤	212	È	244	¶
21	§	53	5	85	U	117	u	149	ò	181	Á	213	ı	245	§
22	▬	54	6	86	V	118	v	150	û	182	Â	214	Í	246	÷
23	↨	55	7	87	W	119	w	151	ù	183	À	215	Î	247	¸
24	↑	56	8	88	X	120	x	152	ÿ	184	©	216	Ï	248	°
25	↓	57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	¨
26	→	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
27	←	59	;	91	[	123	{	155	ø	187	╗	219	█	251	¹
28	∟	60	<	92	\	124	|	156	£	188	╝	220	▄	252	³
29	↔	61	=	93	]	125	}	157	Ø	189	¢	221	¦	253	²
30	▲	62	>	94	^	126	~	158	×	190	¥	222	Ì	254	■
31	▼	63	?	95	_	127	⌂	159	ƒ	191	┐	223	▀	255	

- Esempio:
- Signed char
  - 1 byte
  - 256 simboli diversi
  - Da -128 a -127
- Unsigned char
  - 1 byte
  - 256 simboli diversi
  - Da 0 a 255
- Esempio

  char c;

  …….

  c = ‘a’;

  c = 97;

- Esempi di operatori
  - Essendo un int ha le stesse operazioni
  - = assegnamento
  - + , - , * , /
  - % Resto della divisione intera
  - == relazione di uguaglianza
  - != relazione di diversità
  - < relazione di minore
  - > relazione di maggiore
  - <= relazione di minore uguale
  - >= relazione di maggiore uguale
  - & and bit a bit
  - | or bit a bit
  - …

Schema riassuntivo

- Tipo **char**
  - **Char**, **unsigned char**, **signed char** (1 byte)
  - Rappresentati come un intero
  - Printf con opzione %d %c

- Tipo **int**
  - **Short, unsigned**, **signed**, **long**

- Tipo **float** e **double**
  - **Long** per **double**

- Char **int float double** (tipi aritmetici)
  - Totalmente ordinati
  - Sono dotati di limite
  - Solo **char** ed **int** utilizzabili come indici

Semplificando…

- I principali tipi numerici utilizzati sono:
  - char
  - short
  - int
  - long
  - float
  - double
  - long double

interi

reali

Esempi di spazio allocato

Tipo	Occupazione (in bits)		
	VAX	PDP	UX-16
char	8	8	8
int	32	16	16
short	16	16	16
long	32	32	32
float	32	32	32
double	64	64	64

C e tipizzazione forte

- Assegnare tipi alle variabili consente di verificare che le operazioni sui relativi dati siano usate correttamente
- Le regole di uso dei tipi del C sono tutte verificabili “a compile-time”
- Questa caratteristica viene anche indicata come *tipizzazione forte*; non tutti i linguaggi ne sono dotati
- Altri errori “a compile-time”:
  - Errato annidamento di parentesi
  - Mancata o errata dichiarazione di variabile
  - …
- Errori a run-time:
  - Divisione per 0
  - Indice di un array fuori dai limiti
  - Accesso ad una variabile non inizializzata
  - …

Parte esecutiva

- Espressioni e condizioni

- Istruzioni semplici:
  - Assegnamento
  - Input/Output (in realtà in C vengono realizzate tramite *funzioni*)

- Istruzioni composte:
  - Condizionale
    - if
  - Iterative
    - while
    - do while
    - for

Espressioni e condizioni

- Un'espressione è data dalla composizione di variabili, costanti e numeri mediante operatori aritmetici e parentesi
  - Esempio: `3*x+(2+y)/z`

- Una condizione può essere:
  - La composizione di due espressioni tramite un operatore relazionale (es: `x+1!=4*i`)
  - La composizione di una o più condizioni tramite operatori logici (AND, OR, NOT)

Gli operatori logici o booleani: tabelle di verità

- Operatore binario AND (&&)

A	B	$A \wedge B$
F	F	F
F	V	F
V	F	F
V	V	V

- Operatore binario OR (||)

A	B	$A \vee B$
F	F	F
F	V	V
V	F	V
V	V	V

- Operatore unario NOT (!)

A	$\neg A$
F	V
V	F

Più precisamente …

- Mentre le espressioni assumono valori numerici, abbiamo visto che le condizioni assumono valori booleani (VERO o FALSO)

- In realtà, in C le condizioni sono considerate come espressioni:
  - al valore logico FALSO è associato il valore numerico 0
  - al valore logico VERO è associato il valore numerico 1 o diverso da 0

Regole di precedenza tra operatori

- In una condizione gli operatori aritmetici hanno priorità maggiore rispetto a quelli relazionali (ossia vengono valutati prima) e quelli relazionali priorità maggiore rispetto ai logici

- Regole di precedenza operatori logici:
  - NOT ha la massima precedenza
  - poi segue AND
  - infine OR

- Regole di precedenza operatori aritmetici:
  - *,/,% hanno la massima precedenza
  - poi seguono +,-

- Se voglio alterare queste precedenze devo usare le parentesi (a volte usate solo per maggior chiarezza)

- Per valutare un espressione booleana si usa la *tabella della verità*

- Due espressioni booleane sono equivalenti se e solo se le tabelle della verità sono identiche

Valutazione delle espressioni

- Un’espressione aritmetica come x + y è caratterizzata dal valore e dal tipo del risultato.

- Il tipo degli operandi condiziona l’operazione che deve essere eseguita. (A operandi di tipo **int** si applica l’operazione di somma propria di tale tipo, diversa è l’operazione di somma che si applica a operandi di tipo **float** ecc.)

- Se x è di tipo **short** e y di tipo **int** è necessario convertire una delle due variabili per rendere omogenea l’espressione e applicare la corretta operazione. x viene temporaneamente convertita in **int** e la somma tra interi restituisce un risultato intero.

- In generale:

- Regole di **conversione implicita**:
  1. ogni variabile di tipo **char** o **short** (incluse le rispettive versioni **signed** o **unsigned**) viene convertita in variabile di tipo **int**;

  2. se dopo l'esecuzione del passo 1 l'espressione risulta ancora eterogenea rispetto al tipo degli operandi coinvolti, secondo la seguente gerarchia

     int < **long** < **float** < **double** < **long double**

     si converte temporaneamente l'operando di tipo inferiore facendolo divenire di tipo superiore;

  3. Il risultato dell'espressione avrà tipo uguale a quello di più alto livello gerarchico.

- Le regole di conversione implicita esposte vengono utilizzate anche per la valutazione di assegnamenti tra variabili eterogenee in tipo:

  Double d;
  int i;

- L'istruzione

  d = i;

  provoca una temporanea conversione del valore dell'intero i a **double** e successivamente l'assegnamento di tale valore **double** a d.

- L'istruzione

  i = d;

  comporta invece, normalmente, una *perdita di informazione*. Il valore di d subisce infatti un *troncamento* alla parte intera con perdita della parte decimale.

- **...**

Assegnamento

- Nome_var = espressione ;
  - Viene valutata l'espressione a destra dell'uguale
  - Il risultato della valutazione viene assegnato alla variabile a sinistra dell'uguale.

- Esempio:

```
int i;
i=2;
i=3*i;
```

Dopo l'esecuzione del codice, la variabile i è associata all'intero 6.

Istruzioni (funzioni!) di Input / Output

- In realtà sono funzioni fornite dalla libreria standard del C
- Introduzione alle funzioni di Input/Output:
  - `printf` (per l'output)
  - `scanf` (per l'input)
- L'utilizzo delle funzioni richiede l'inclusione della libreria che le contiene:

```
#include <stdio.h>
main () {

/* utilizzo delle funzioni di libreria */

}
```

Funzione di output printf

- Sintassi:

  `printf(`stringa di controllo`,`expr_1`,`expr_2`,`… `,`expr_n`);`

- Stringa di controllo
  - stringa di caratteri da stampare contenente
  - caratteri di conversione: iniziano per % ed indicano nell'ordine la posizione ed il tipo delle espressioni seguenti, o meglio la forma nella quale devono essere stampate (es. %d per intero, %f per reale, …)
  - caratteri di formato: iniziano per "\" servono per la formattazione del testo:
    - \n = "a capo",
    - \b = "backspace",
    - \t = "horizontal tab",
    - \r = "carriage return",
    - ...

- I caratteri di conversione possono essere:
  - %d: l’espressione è convertita in decimale
  - %o: l’espressione è convertita in ottale
  - %x: l’espressione è convertita in esadecimale
  - %c: l’espressione è un carattere
  - %s: l’espressione è una stringa
  - %f: l’espressione è un float o un double e viene convertita nella forma con punto decimale.
  - %e: l’espressione è un float o un double e viene convertita nella forma esponenziale.

- Nel caso delle stringhe vengono stampati i caratteri finché non si incontra il carattere di fine riga, o non si esaurisce il numero di elementi indicati nel numero di conversione.

- Esempi:

```
printf ("Lo stipendio annuo dei dipendenti di categoria %d è pari a L. %f", cat_dipend, stip_medio);
```

```
printf("%s\n%c%c\n\n%s\n", "Questo programma è stato scritto da", iniz_nome, iniz_cognome, "Buon lavoro!");
```

Funzione di input scanf

- Sintassi:

  `scanf(`stringa di controllo`,`&var_1`,`&var_2`,… ,`&var_n`);`

- Legge caratteri dallo standard input, li interpreta, e memorizza nell'ordine il risultato nelle variabili elencate, var_1,var_2,…,var_n, il cui nome è preceduto da &.

- La stringa di controllo è simile a quella di printf e specifica il modo in cui devono essere interpretati i caratteri immessi nello standard input.

- Esempio:

  scanf("%c%c%c%d%f", &c1, &c2, &c3, &i, &x);

La direttiva #include

- Ogni programma che utilizza al suo interno le funzioni *printf* e *scanf* deve dichiarare l'uso di tali funzioni nella parte direttiva che precede il programma principale:

```
#include <stdio.h>
```

- È una direttiva data a una parte del compilatore, chiamata *preprocessore*

Esempio

```
                                /* Programma Fahrenheit-Celsius */
#include <stdio.h>
main()
{

   int Ftemp;
   float Ctemp;

   printf("Inserire la temperatura in gradi Fahrenheit da
   convertire in gradi Celsius\n");

   scanf("%d",&Ftemp);
   Ctemp=(5.0/8.0)*(Ftemp - 32);

   printf("%d gradi Fahrenheit corrispondono a %f  in gradi
   Celsius\n", Ftemp, Ctemp);
}
```

Blocchi di istruzioni

- Una sequenza di istruzioni è:
  - istruzione;

    oppure
  - {istruzione_1; istruzione_2; … istruzione_k;}

- Le sequenze di istruzioni ci sono utili per definire le istruzioni composte del C, ossia istruzioni che combinano altre istruzioni tramite condizioni

Istruzione condizionale

- L'istruzione condizionale permette di modificare il flusso di esecuzione a seconda del valore di verità di una condizione

- Sintassi:

```
if condizione
    sequenza_istruzioni_1
else
    sequenza_istruzioni_2
```

  - Viene valutata la condizione
  - Se la condizione risulta vera vengono eseguite le istruzioni in sequenza_istruzioni_1
  - Se la condizione risulta falsa vengono eseguite le istruzioni in sequenza_istruzioni_2

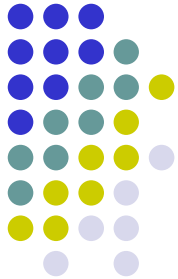
- Esempio:

```
if (x % 2 == 0)
 printf (“%d è pari”,x);
else
 {
 printf (“%d è dispari”,x);
 x=x+1;
 }
```

- La parte dell’else è facoltativa; ad esempio:

```
if (x != 0)
 {
 x=x/2;
 a=5/x;
 }
```

Istruzioni iterative while e do…while



- Le istruzioni iterative while e do…while permettono di ripetere una sequenza di istruzioni mentre una condizione è vera.

- Sintassi:

```
while condizione
    sequenza_istruzioni
```

1. Viene valutata la condizione
2. Se è vera si eseguono le istruzioni in sequenza_istruzioni e si torna al passo 1
3. Se è falsa si esce dal ciclo

```
do
    sequenza_istruzioni
while condizione;
```

1. si eseguono le istruzioni in sequenza_istruzioni
2. Viene valutata la condizione
3. Se è vera si torna al passo 1
4. Se è falsa si esce dal ciclo

- Nel while la condizione viene controllata all'inizio di ogni iterazione, mentre nel do…while alla fine

- Nel do…while le istruzioni del ciclo vengono eseguite almeno una volta, mentre nel while potrebbero non essere eseguite mai.

- Esempi:

```
while (x > 0) {
   printf ("%d\n",x);
   x=x-1;
   }
```

Stampa in ordine decrescente i numeri da x a 1

```
do {
   scanf ("%d",&x);
} while (x < 100);
```

Prende in input valori numerici fino a che viene immesso un numero almeno pari a 100

Istruzione iterativa for

- L'istruzione for è utile quando bisogna ripetere una sequenza di istruzioni un numero determinato di volte per mezzo di una variabile contatore

- Sintassi:

  `for (`inizializzazione`;` condizione`;` aggiornamento`)`
  sequenza_istruzioni

  Viene inizializzata la variabile contatore tramite un comando di assegnamento presente in inizializzazione

  Viene valutata la condizione

  - Se risulta vera vengono eseguite le istruzioni del ciclo, viene aggiornata la variabile contatore tramite l'incremento o il decremento presente in aggiornamento e si torna al passo 2
  - Se risulta falsa si esce dal ciclo

- La condizione viene verificata all’inizio di ogni iterazione
- L’incremento viene effettuato alla fine di ogni iterazione
- Esempio:

```
for (i=0;i<=k;i++)
    printf (“%d\n”,i);
```

Vengono stampati i numeri interi da 0 a k

Ancora sui tipi: definizione di tipo

- Dichiarazione di un nuovo tipo
  - Effettuata normalmente all'inizio del programma
  - Si utilizza il costruttore di tipo **typedef,** eventualmente con altri costruttori tipi **struct, array**

- Il costruttore **typedef** consente di dichiarare nuovi tipi elencando nell'ordine:
  - La rappresentazione del nuovo tipo
  - Il nome del nuovo tipo
  - Termina sempre con un ;

- L'utente può definire nuovi tipi mediante:
  - Ridefinizione
  - Applicazione del costrutto di enumerazione esplicita

Ridefinizione di tipi

- Sintattica:

  typedef TipoEsistente NuovoTipo;

- Dove:
  - TipoEsistente può essere sia un tipo *built-in* (predefinito), sia un tipo precedentemente definito
  - NuovoTipo è nome che identifica il nuovo tipo

- Esempi:

  -
    ```
    typedef int anno;
    typedef char tipo1;
    typedef anno tipo2;
    typedef tipo2tipo3;
    ```

Una volta definito e identificato un nuovo tipo ogni variabile può essere dichiarata di quel tipo come di ogni altro tipo già esistente:

```
char x;
anno y;
```

Enumerazione esplicita di valori

- Un nuovo tipo dichiarato con la parola chiave **typedef,** seguito da
  - la parola chiave **enum**
  - la lista dei possibili valori fra parentisi graffe (separati da virgole)
  - il nome del nuovo tipo
  - termina con un punto e virgola

- Esempio:

```
typedef enum    {lun, mar, mer, gio, ven, sab, dom}  GiornoDellaSettimana;
typedef enum    {rosso, verde, giallo, arancio, marrone, nero}  colore;
typedef enum    {Giovanni, Claudia, Carla, Simone, Serafino}  persone;
typedef enum    {gen, feb, mar, apr, mag, giu, lug, ago, set, ott, nov, dic} mese;
```

```
persone individuo, individuo1, individuo2;

individuo = Giovanni;
if (individuo1 == individuo2) individuo = Claudia;
```

- Alcune osservazioni:
  - Spesso i valori del nuovo tipo sono rappresentati da nomi; però il compilatore associa a tali nomi nell'ordine un valore intero progressivo a partire da 0
  - Per esempio, per x di tipo mese:
    - gen è in realtà 0, apr è in realtà 3, ecc.

- Operazioni applicabili: le stesse degli interi

  Le seguenti operazioni:

  apr < giu

  rosso < arancio

  Producono come risultato un intero diverso da 0 (valore logico "vero"); mentre:

  dom < lun

  Simone < Giovanni

  Producono 0 (valore "false").

- Un importante caso particolare:

```
typedef   enum {false, true} boolean;
boolean    flag, ok;
```

- flag e ok possono così essere definite come variabili in grado di assumere valore vero (true) o falso (false) durante l'esecuzione di un programma che le usi.

- **NB: non invertire l'ordine!**