U.D. 3

2. - Programmazione in linguaggio C

Compatibilità tra i tipi (semplici)
Precedenze tra gli operatori
Frasi di I/O semplice
-printf
-scanf

Fonti:
A. Antola - POLI_MI Dispense del Corso
A. Ciampolini – UNI_BO – Lucidi del corso

Espressioni Omogenee ed Eterogenee

In C e` possibile combinare tra di loro operandi di tipo diverso:

- espressioni **omogenee**: tutti gli operandi sono dello stesso tipo
- espressioni **eterogenee**: gli operandi sono di tipi diversi.

Regola adottata in C:

- sono eseguibili le espressioni eterogenee in cui tutti i tipi referenziati risultano ***compatibili*** (cioe`: dopo l'applicazione della regola automatica di conversione implicita di tipo del C risultano omogenei).
- non sono eseguibili le espressioni eterogenee se tutti i tipi referenziati risultano non **compatibili** (cioe` restano eterogenei anche dopo l'applicazione della regola automatica di conversione implicita di tipo del C).

Compatibilita` fra tipi di dato

Definizione:

Un tipo di dato T_1 e` **compatibile** con un tipo di dato $T_2$ se il dominio $D_1$ di $T_1$ e` contenuto in D_2, dominio di T_2.

Ad esempio: gli interi sono compatibili con i reali, perche` $Z \subset R$

☞ la relazione di compatibilita` **non e` simmetrica:** se T_1 e` compatibile con $T_2$, non e` detto che T_2 sia compatibile con T_1.

Proprieta`:

- Se T_1 e` compatibile con $T_2$ , un'operatore *Op* definito per $T_2$ puo` essere anche utilizzato con argomenti T_1.

Quindi:

se *Op* e` definito per T2 come: $Op: D2 \times D2 \rightarrow D2$

Allora puo` essere utilizzato anche come:

$Op: D1 \times D2 \rightarrow D2$

$Op: D2 \times D1 \rightarrow D2$

Compatibilita` tra tipi primitivi

In C e` definita la seguente gerarchia tra i tipi primitivi:

char < short < int

int < long < unsigned< unsigned long < float < double < long double

Dove il simbolo **<** indica la relazione di **compatibilita`**.

☞ La gerarchia associa un ***rango*** a ciascun tipo:

Ad esempio:

rango(int)<rango (double)

Regola di Conversione Implicita

Facendo riferimento alla gerarchia tra tipi C primitivi, ad ogni espressione *x op y* viene applicata automaticamente la seguente **regola**:

1. Ogni variabile di tipo **char** o **short** (eventualmente con qualifica **signed** o **unsigned**) viene convertita nel tipo **int**;
2. Se dopo il passo 1 l'espressione e` ancora eterogenea,si converte temporaneamente l'operando di tipo *inferiore* al tipo *superiore* (***promotion***);
3. A questo punto l'espressione e` **omogenea** e viene eseguita l'operazione specificata. Il risultato e` di tipo uguale a quello prodotto dall'operatore effettivamente eseguito. (In caso di overloading, quello di rango piu` alto).

Compatibilita` e conversione implicita di tipo

☞ La compatibilita`, di solito, viene controllata staticamente, applicando le regole di conversione implicita in fase di **compilazione** senza conoscere i valori attribuiti ai simboli (**tipizzazione forte**).

Esempio 1: espressione semplice

	x / y	
3 / 3.0	➠ 1.0	(reale)
3.0 /3	➠ 1.0	(reale)
3 / 3	➠ 1	(intero)

Esempio 2: espressione composta

```
int x;
char y;
double r;

(x+y) / r
```

E` necessario conoscere:

☞ **Priorita`** degli operatori (definita dallo standard)

☞ **Ordine di valutazione** degli operandi (lo standard non lo indica)

Ipotesi: gli operandi vengono valutati da sinistra a destra:

- **passo 1**: (x+y)
  - y viene convertito nell'intero corrispondente
  - viene applicata la somma tra interi

  ➠ **risultato intero *tmp***

- **passo 2:** *tmp* / r
  - tmp viene convertito nel double corrispondente
  - viene applicata la divisione tra reali

  ➠ **risultato reale**

Conversione esplicita:

In C si puo` forzare la conversione di un dato in un tipo specificato, mediante l'operatore di **cast**:

(<nuovo tipo>) <dato>

il <dato> viene convertito esplicitamente nel <nuovo tipo>:

```
int A, B;
float C;

C=A/(float)B;
```

⇒ viene eseguita la divisione tra reali.

Tipi primitivi nel linguaggio C: *Integral* Types e *Floating* Types

I tipi primitivi (scalari) del C possono essere suddivisi in tipi **enumerabili** (Integral Types) e non (Floating Types).

Tipi enumerabili (*Integral Types*):

gli elementi del dominio associato al tipo sono rigidamente ordinati:

- e` possibile far riferimento al primo valore del dominio ed all'ultimo.
- per ogni elemento e' sempre possibile individuare l'elemento *precedente* (se non e` il primo) ed il *successivo* (se non e` l'ultimo).

☞ a ciascun elemento del dominio puo` essere associato un valore intero positivo, che rappresenta il numero d'ordine dell'elemento nella sequenza ordinata dei valori.

Tipi enumerabili in C:

char

int

Tipi non enumerabili (*Floating Types*):

Concettualmente, il dominio R e` un insieme *denso*: dati due elementi x1 ed x2 del dominio distanziati tra loro di un ε piccolo a piacere, esiste sempre un'infinita` di valori di R contenuti nell'intervallo
[x1, x2].

Tipi non enumerabili in C:

float

double

{Integral Types, Floating Types}

L'insieme di queste due categorie costituisce i **tipi aritmetici.**

Definizione e Inizializzazione delle variabili di tipo semplice

Definizione di variabili

Tutti gli identificatori di tipo primitivo descritti fin qui possono essere utilizzati per definire variabili.

Ad esempio:

```
char lettera;
int, x, y;
unsigned int P;
float media;
```

Inizializzazione di variabili

E` possibile specificare un valore iniziale di una variablie in fase di definizione.

Ad esempio:

```
int x =10;
char y = 'a';
double r = 3.14*2;
```

☞ Differisce dalla definizione di costanti, perche` i valori delle variabili, durante l'esecuzione del programma, potranno essere modificati.

Istruzioni: classificazione

In C, le istruzioni possono essere classificate in due categorie:

- istruzioni **semplici**
- istruzioni **strutturate**: si esprimono mediante composizione di altre istruzioni (semplici e/o strutturate).

Regola sintattica generale:
In C, ogni istruzione e` terminata da un punto e virgola.
[fa eccezione l'istruzione composta]

Assegnamento

E` l'istruzione con cui si modifica il valore di una variabile.

Sintassi:

<istruzione-assegnamento>::=
<identificatore-variabile> = <espressione>;

Ad esempio:

```
int A, B;

A=20;
B=A*5; /* B=100 */
```

Compatibilita` di tipo ed assegnamento:

In un assegnamento, l'identificatore di variabile e l'espressione devono essere dello stesso tipo (eventualmente, conversione implicita).

Esempio:

```
int x;
char y='a'; /*codice(a)=97*/
double r;

x=y;   /* char -> int: x=97*/
x=y+x; /*x=194*/
r=y+1.33; /* char -> int -> double*/
x=r; /* troncamento: x=98*/
```


Esempio:

```
main()
{
  /*definizioni variabili: */
int          X,Y;
unsigned int Z;
float      SUM;
  /* segue parte istruzioni */
  X=27;
  Y=343;
  Z = X + Y -300;
  X = Z / 10 + 23;
  Y = (X + Z) / 10 * 10;
  /* qui X=30, Y=100, Z=70 */
  X = X + 70;
  Y  = Y % 10;
  Z = Z + X -70;
  SUM = Z * 10;
  /* X=100, Y=0, Z=100 , SUM=1000.0*/
}
```

Assegnamento come operatore

Formalmente, l'istruzione di assegnamento è un'espressione:

- Il simbolo = **e` un operatore**
  - l'istruzione di assegnamento e` una espressione
  - ritorna un valore:
    - il valore ritornato e` quello assegnato alla variabile a sinistra del simbolo =
    - il tipo del valore ritornato e` lo stesso tipo della variabile oggetto dell'assegnamento

Ad esempio:

```
const int valore=122;
int K, M;

K=valore+100; /* K=222;l'espressione
                 produce il
                 risultato 222 */
M=(K=K/2)+1; /* K=111, M=112*/
```

Assegnamento

In C sono disponibili operatori che realizzano particolari forme di assegnamento:

Operatori di incremento e decremento:

Determinano l'incremento/decremento del valore della variabile a cui sono applicati.
Restituiscono come risultato il valore incrementato/decrementato della **variabile** a cui sono applicati.

```
int A=10;

A++; /*equivale a: A=A+1; */
A--; /*equivale a: A=A-1; */
```

Differenza tra notazione prefissa e postfissa:

- **Notazione Prefissa**: (ad esempio, ++**A**) significa che l'incremento viene fatto prima dell'impiego del valore di A nella espressione.
- **Notazione Postfissa:** (ad esempio, **A**++) significa che l'incremento viene effettuato dopo l'impiego del valore di A nella espressione.

Ad esempio:

```
int A=10, B;
char C='a';

B=++A;  /*A e B valgono 11 */
B=A++;     /* A=12, B=11 */
C++;       /* C vale 'b' */

int i, j, k;
k = 5;
i = ++k; /*  i = 6, k = 6 */
j = i + k++; /* j=12, i=6,k=7 */

j = i + k++; /*equivale a:
                 j=i+k; k=k+1;*/
```

☞ In C l'ordine di valutazione degli operandi non e' indicato dallo standard: si possono scrivere espressioni il cui valore e` difficile da predire.

Operatore di assegnamento *abbreviato*:

E` un modo sintetico per modificare il valore di una variabile.
Sia **v** una variabile, ***op*** un'operatore (ad esempio, +,-,/, etc.), ed **e** una espressione.

v ***op***= e

.è ***quasi*** equivalente a:

v = v ***op*** (e)

Ad esempio:

```
k += j          /*  equivale a k = k + j  */
k *= a + b      /*  equivale a k = k * (a + b)  */
```

☞ Le due forme sono **quasi equivalenti** perchè in

v *op*= e

v viene valutato una sola volta, mentre in:

v = v *op* (e)

v viene valutato due volte.

Espressioni sequenziali:

Un'espressione sequenziale si ottiene concatenando tra loro piu` espressioni con l'operatore virgola (,).

- Il risultato prodotto da un'espressione sequenziale e` il risultato ottenuto dall'ultima espressione della sequenza.
- La valutazione dell'espressione avviene valutando nell'ordine testuale le espressioni componenti, da sinistra verso destra.

Esempio:

```
int A=1;
char B;
A=(B='k', ++A, A*2); /* A=4 */
```

Precedenza e Associativita` degli Operatori

In ogni espressione, gli operatori sono valutati secondo una **precedenza** stabilita dallo standard, seguendo opportune regole di **associativita`**:

- La **precedenza** indica l'ordine con cui vengono valutati operatori diversi;
- L'**associativita`** indica l'ordine in cui operatori di pari priorita` (cioe`, stessa precedenza) vengono valutati.

☞ E` possibile forzare le regole di precedenza mediante l'uso delle parentesi.

Regole di Precedenza e Associativita` degli Operatori C (in ordine di priorita` decrescente)

Operatore	Associativita`
`() [] ->`	da sinistra a destra
`! ~ ++ -- & sizeof`	da destra a sinistra
`* / %`	da sinistra a destra
`+ -`	da sinistra a destra
`<< >>`	da sinistra a destra
`< <= > >=`	da sinistra a destra
`== !=`	da destra a sinistra
`&`	da sinistra a destra
`^`	da sinistra a destra
`|`	da sinistra a destra
`&&`	da sinistra a destra
`||`	da sinistra a destra
`?:`	da destra a sinistra
`=+ =- =* =/`	da destra a sinistra
`,`	da sinistra a destra

Precedenza e Associativita`

Esempi

3*5 % 2 ➠ equivale a: (3*5) % 2
X+7-A ➠ equivale a: (X+7) - A
3 < 0 && 3 < 10 ➠ (3<0) && (3<10) ➠ 0 && 1
3<(0 && 3) < 10 ➠ (3 < 0) < 10 ➠ 0 < 1
0 == 7 == 3 ➠ 0==(7==3) ➠ 0 == 0

Valutazione a "corto circuito" *(short-cut)*:

nella valutazione di una espressione C, se un risultato intermedio determina a priori il risultato finale della espressione, il resto dell'espressione non viene valutato.

Ad esempio, espressioni logiche:

Hp. Valutazione degli operandi da sin a destra

(3 > 0) && (X < Y) ➠ solo primo operando
falso && vero

☞ Bisognerebbe evitare di scrivere espressioni che dipendono dal metodo di valutazione usato ➠ scarsa portabilita` (ad es., in presenza di funzioni con effetti collaterali).

Esercizi:

Sia V=5, A=17, B=34. Determinare il valore delle seguenti espressioni logiche:

A<=20 || A>=40
! (B=A*2)
A<=B && A<=V
A>=B && A>=V
! (A>=B && A<=V)
! (A>=B) || !(A<=V)

Soluzioni: (Hp: valutazione degli operandi sn->dx)

A<=20 || A>=40 ➠ vero {A<20, short cut}
! (B=A*2) ➠ falso {B=34=17*2}
A<=B && A<=V ➠ falso {A>V}
A>=B && A>=V ➠ falso {A<B}
! (A<=B && A<=V) ➠ vero
! (A>=B) || !(A<=V) ➠ vero

Istruzioni di ingresso ed uscita (input/output)

L'immissione dei dati di un programma e l'uscita dei suoi risultati avvengono attraverso operazioni di lettura e scrittura.

Il C non ha istruzioni predefinite per l'input/output.

Libreria standard di I/O:

In ogni versione ANSI C, esiste una *Libreria Standard* di input/output (**stdio**) che mette a disposizione alcune funzioni (dette *funzioni di libreria*) che realizzano l'ingresso e l'uscita da/verso i dispositivi standard di input/output.

Dispositivi standard di input e di output:

per ogni macchina, sono periferiche predefinite (di solito, tastiera e video).

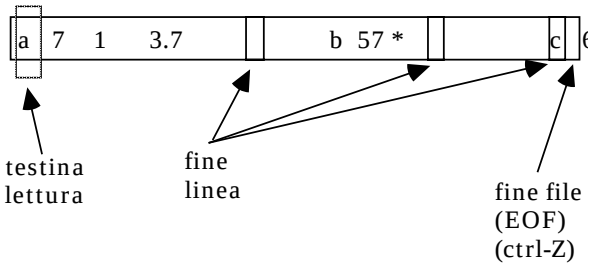
Input/Output

Il C vede le informazioni lette/scritte da/verso i dispositivi standard di I/O come **file *sequenziali***, cioe` **sequenze di caratteri**.

I file standard di input/output possono contenere dei caratteri di controllo:

- **End Of File (EOF)** indica la fine del file
- **End Of Line** indica la fine di una linea

...



Funzioni di libreria per:

- **Input/Output a caratteri**
- **Input/Output a stringhe di caratteri**
- **Input/Output con formato**

I/O con formato

Nell'Input ed Output con formato occorre specificare il formato dei dati che si vogliono leggere oppure stampare.

Il formato stabilisce:

- come interpretare la sequenza dei caratteri immessi dal dispositivo di ingresso (nel caso della **lettura**)
- con quale sequenza di caratteri rappresentare in uscita i valori da stampare (nel caso di **scrittura**)

Il formato viene indicato con opportune direttive del tipo:

%<direttiva>

Formati più comuni:

		short	long
signed int	**%d**	**%hd**	**%ld**
unsigned int	**%u** (decimale)	**%hu**	**%lu**
	%o (ottale)	**%ho**	**%lo**
	%x (esadecimale)	**%hx**	**%lx**
float	**%e**, **%f**, **%g**		
double	**%le**, **%lf**, **%lg**		
carattere singolo	**%c**		
stringa di caratteri	**%s**		
puntatori (indirizzi)	**%p**		

FUNZIONI DI I/O PER VIDEO E TASTIERA

Stampa su video

printf(stringa di controllo, elementi da stampare)

dove:

- printf ( ) è l'identificatore riservato della funzione
- *stringa di controllo* è racchiusa tra " e " e contiene
  - ⇒caratteri alfanumerici da stampare direttamente su video
  - ⇒**caratteri di conversione** e/o di formato preceduti dal simbolo **%** che vengono utilizzati al momento di interpretare per la stampa i valori degli elementi da stampare. Esempi di caratteri di conversione **d, f, c** ....
  - ⇒**caratteri di controllo della stampa** (che sono caratteri ASCII a cui non corrisponde alcun simbolo stampabile e che hanno come effetto quello di linea nuova (\n), tabulazione, salto pagina..
- *elementi da stampare* è una lista di identificatori di variabili, identificatori di costanti, espressioni il cui valore deve essere stampato. La lista è ordinata rispetto ai caratteri di conversione.

Significato e funzionamento:

- l'istruzione

```
printf (stringa dicontrollo, elementi da stampare);
```

  è la chiamata alla funzione e quindi si attiva l'esecuzione del sottoprogramma associato.

- vengono stampati gli eventuali caratteri alfanumerici tra doppi apici e nella posizione del carattere di conversione viene stampato il valore dell'identificatore corrispondente nella lista di elementi da stampare. I caratteri di controllo stampa spostano il cursore in posizione opportuna.

Basic Output

- function format

  printf(“format-control-string”, <arguments>);
- Format control string may contain:

  - ordinary text characters (“This is a string”)

  - escape sequences (\n, \t, etc.)

  - conversion code (%d, %c, %f, %lf)

    begins with % sign

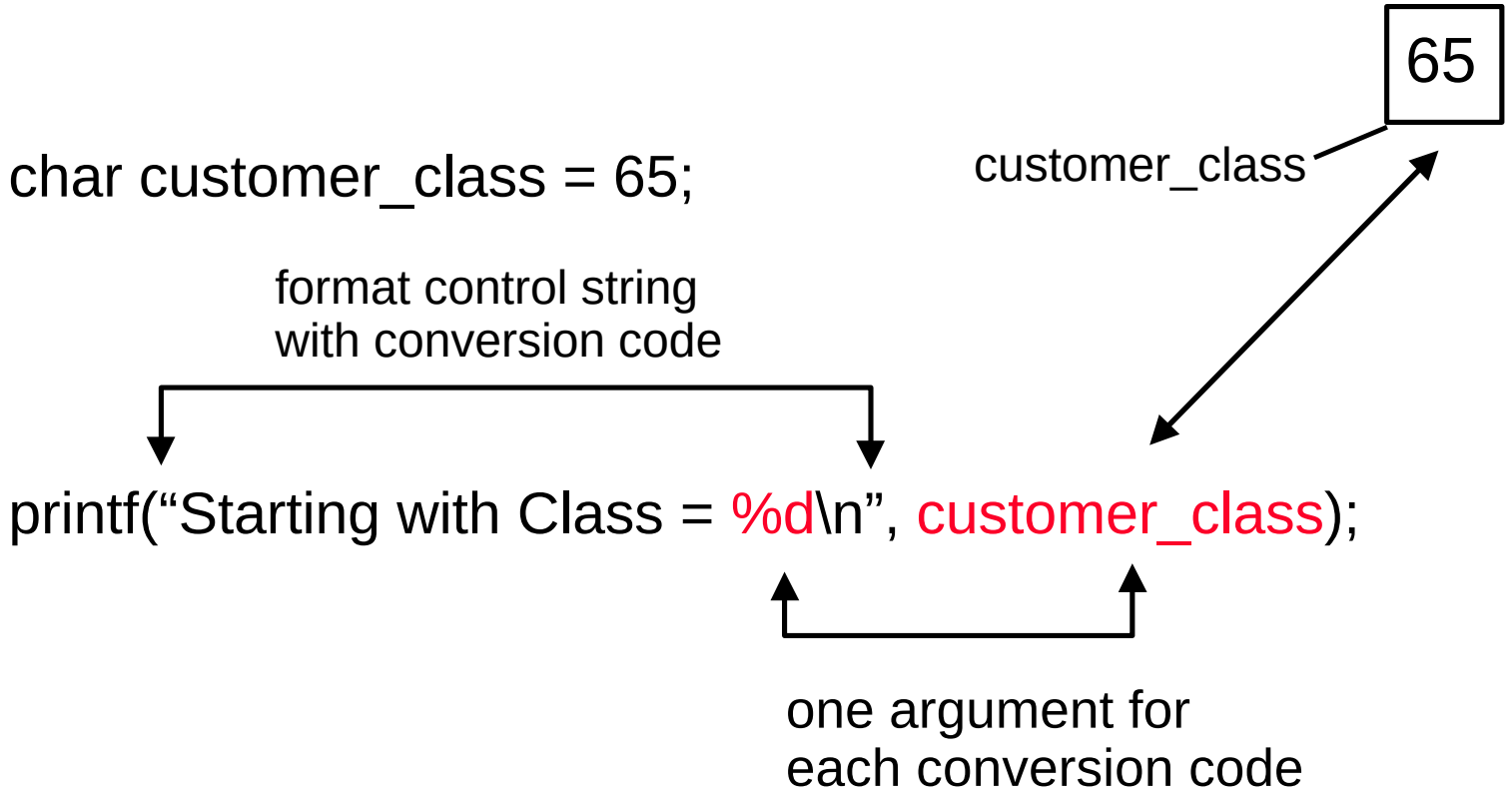
    some optional fields

    d for decimal output; c for character; f for float, lf for double
- argument present only if matching a conversion code

  MUST have ONEargument per conversion code

  You MAY print just text and escape sequences, then no arguments

printf()


65
customer_class
char customer_class = 65;
format control string
with conversion code
printf("Starting with Class = %d\n", customer_class);
one argument for
each conversion code

Special Characters

- Escape sequences are used to represent many special characters in C

\n	newline	each escape sequence represents only one character
\t	tab	
\b	backspace	
\f	form feed	
\a	audible bell	
\0	null	

- Can be mixed freely with other characters

  printf("\nA\nB\tC\nDE\aF\n"); /* what does this print? */

  do NOT try to print the null ('\0') character

Constants

Integer vs Floating-point Constants

10 33 3.333

Decimal Constants

10 33L

Octal

017

Hex

0X1A 0x1A 0x1a

Character

‘A’ ‘a’ ‘2’

String

“A string has more than one character”

Constant Values

- int:

  20, -123 // decimal

  024, 0177 // octal

  0X14, 0xffff // hexadecimal

  24U, 35u // unsigned

- long:

  -32L, 123456789L // decimal (avoid using lower case ‘L’)

  1111UL, 4321uL // unsigned

- float:

  3.14F, -0.1234f

- double:

  3.1415, -12345.6789

  0.31415E1, -1.e-10 // exponential notation

- long double:

  3.1415926L

- char:

  ‘a’, ‘A’, ‘3’,’%’ // printable

  ‘\n’, ‘\t’, ‘\7’ // non-printable

  “hello world!\n” // character string

(constants == $\rightarrow$ nonaddressable)

Irwin Sheer

Superconducting Super Collider Laboratory

MS 2300, 2550 Beckleymeade Ave., Dallas, TX 75237

Tel: (214) 708-1050; Fax: (214) 708-6354

e-mail: Irwin_Sheer@ssc.gov

Sample C Code

octal constant

```
int x, y;

x = 011;
y = 12;

if(x == 11)
   printf("%d\n", x);
else
   printf("%d\n", y);
```

what is the output?

Sample C Code

octal constant

```
int x, y;

x = 011;     ottale 011 è uguale a decimale   9
y = 12;

if(x == 11)   questo è il decimale 11
   printf("%d\n", x);
else
   printf("%d\n", y);
```

what is the output?

Basic Input

- General Format

  scanf(“format-control-string”, arguments);

  looks like printf(), but very different

  not same action as printf()

  does input, not output; so do NOT use ordinary text, escape codes

  use ONLY conversion codes in format control string; no spaces

  argument list is **address** list, not variable names

  argument list is NOT optional; 1 per conversion code

Lettura con formato: scanf

La scanf assegna i valori letti dal file standard di input alle variabili specificate come argomenti.

Sintassi:

scanf(<stringa-formato>, <sequenza-variabili>);

Ad esempio:

```
int X;
float Y;
scanf("%d%f", &X, &Y);
```

La scanf:

- legge una serie di valori in base alle specifiche contenute in <*stringa-formato*>: in questo caso "%d%f" indica che i due dati letti dallo standard input devono essere interpretati rispettivaente come un valore intero decimale (%d) ed uno reale (%f)
- memorizza i valori letti nelle variabili specificate come argomenti nella <*sequenza_variabili*> (X, Y nell'esempio)
- e` una *funzione* che restituisce il numero di valori letti e memorizzati, oppure EOF in caso di *end of file*.

scanf

Separatori:

ogni direttiva di formato prevede dei separatori specifici:

Tipo di dato	Direttive di formato	Separatori
Intero	%d, %x, %u, etc.	Spazio, EOL, EOF.
Carattere	%c	Nessuno
Stringa	%s	Spazio, EOL, EOF

Nota bene:

- ☞ Se la stringa di formato contiene N direttive, è necessario che le variabili specificate nella sequenza siano esattamente N.
- ☞ Gli identificatori delle variabili a cui assegnare i valori sono sempre preceduti dal simbolo &.
  [Infatti, le variabili devono essere specificate attraverso il loro indirizzo➠ operatore & (v. puntatori e funzioni)]
- ☞ La <stringa_formato> puo` contenere dei caratteri qualsiasi (che vengono scartati, durante la lettura), che rappresentano separatori aggiuntivi rispetto a quelli standard.

Ad esempio:

```
scanf("%d:%d:%d", &A, &B, &C);
```

=> richiede che i tre dati da leggere vengano immessi separati dal carattere ":".

Ingresso da tastiera

scanf(stringa di controllo, variabili a cui associare il valore letto)

dove:

- scanf( ) è l'identificatore riservato della funzione
- *stringa di controllo* è racchiusa tra « e « e contiene
  ⇒**caratteri di conversione** e/o di formato preceduti dal simbolo **%** che vengono utilizzati al momento di interpretare il codice associato alla pressione dei tasti della tastiera per la memorizzazione dei valori nelle variabili (con la codifica adeguata). Esempi di caratteri di conversione **d, f, c** ....
- *variabili a cui associare il valore letto* è una lista di identificatori di variabili. Le variabili devono essere indicate tramite il loro indirizzo: **&nome_var**. La lista è ordinata rispetto ai caratteri di conversione.

Significato e funzionamento:

- l'istruzione

```
scanf (stringa dicontrollo, lista di variabili);
```

  è la chiamata alla funzione e quindi si attiva l'esecuzione del sottoprogramma associato.
- ad ogni pressione di tasto la funzione fa eco su video, visualizzando il carattere alfanumerico premuto
- la sequenza di tasti premuti deve terminare con la pressione del tasto **ENTER**.
- la funzione assegna, con l'opportuna codifica binaria, il valore alle variabili, fin quando possibile.

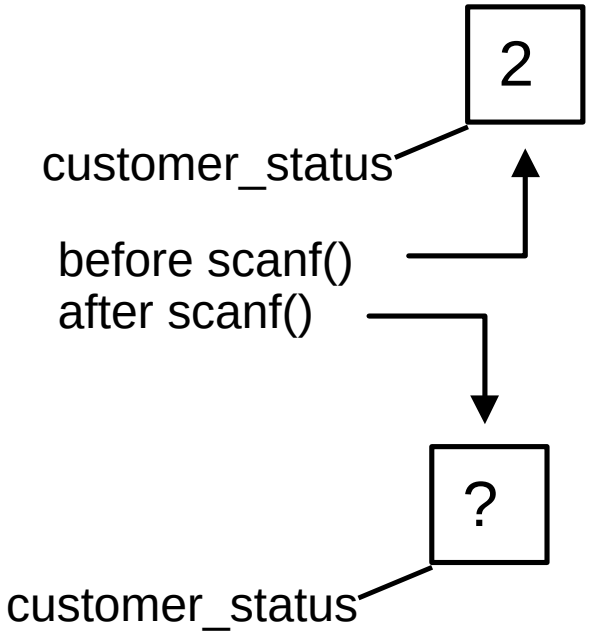
E' da usare con attenzione. Carattere di conversione **%d**: la sequenza di cifre è interpretata come un valore intero da assegnare. Carattere di conversione **%f**: la sequenza di cifre con il punto è interpretata come un valore float da assegnare. Carattere di conversione **%c**: il singolo carattere alfanumerico è interpretato come un carattere ASCII da assegnare.

scanf()

```
int customer_status = 2;
scanf("%d", &customer_status);
```

address operator

only conversion code(s)
no plain text



new value is whatever
was keyed by user

Sample Code

```
#include <stdio.h>

main()
{
   int customer_status = 2;

   printf("Starting with Status = %d\n", customer_status);

   printf("Enter new status: ");          /* prompt user */
   scanf("%d", &customer_status);

   printf("\nNew status value is %d\n", customer_status);
}
```

output would be as follows:

```
Starting with Status = 2
Enter new status: 5          (value entered by user)

New status value is 5
```

printf/scanf

Esempio:

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

Se in ingresso vengono dati:

```
ABC 3 7.345
```

le variabili assumono i seguenti valori:

char c1	'A'
char c2	'B'
char c3	'C'
int i	3
float x	7.345

Esempio: stampa della codifica (decimale, ottale e esadecimale) di un carattere dato da input.

```
#include <stdio.h>

main()
{
int a;
printf("Dai un carattere e ottieni il
valore \
     decimale, ottale e hex  ");
scanf("%c",&a);
printf("\n%c vale %d in decimale, %o in
ottale \
     e %x in hex.\n",a, a, a, a);
}
```

ESEMPIO (prematuro)

Programma che riceve in ingressso nome e cognome separati da uno spazio e visualizza cognome e nome separati da uno spazio

```
#include <stdio.h>
#define N 30

main ()
{
 char Cognome[N], Nome[N];
 int indiceC, indiceN,i;
 char carattere, proseguire,tappo;

 do
 {
  printf("\nInserire Nome e Cognome separati da uno
spazio\n");
  indiceN=0;

 /*legge un carattere del nome fa eco e lo inserisce in
Nome */

  scanf("%c", &carattere);

  while (carattere != ' ')
   {
   Nome[indiceN]=carattere;
   indiceN=indiceN+1;
   scanf("%c", &carattere);
   }
```

```
  indiceC=0;
  /*legge un carattere del cognome fa eco e lo inserisce
in Cognome */

  scanf("%c", &carattere);
  while (carattere != '\n')
   {
   Cognome[indiceC]=carattere;
   indiceC=indiceC+1;
   scanf("%c", &carattere);
   }


/*ciclo per la scrittura del Cognome*/

   for (i=0; i<indiceC; i++)
   /*scrive un carattere del Cognome*/
   printf("%c", Cognome[i]);

   printf(" ");



/*ciclo per la scrittura del Nome*/

   for (i=0; i<indiceN; i++)
   /*scrive un carattere del Nome*/
    printf("%c", Nome[i]);

   printf("\nVuoi inserire un altro nome (S\N)? ");
   scanf("%c", &proseguire);
   scanf("%c", &tappo);
 }
 while(proseguire == 'S');

}  /* fine main */
```

Examples

scanf(“%2d%4s%4f”, &integer, string, &real);

if input is **44mice2.97**

scanf(“%6f%6f”, &x, &y);

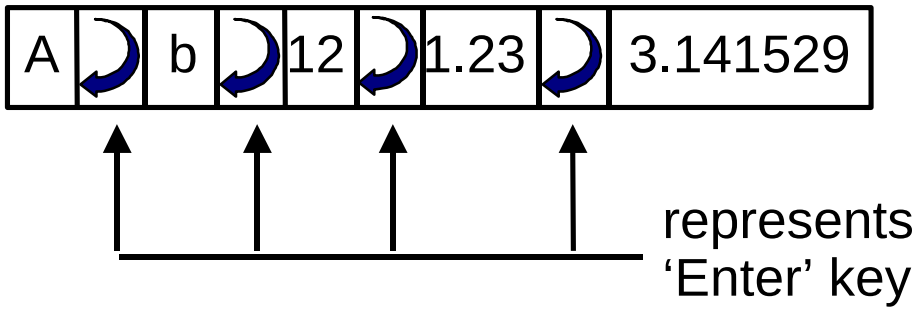
if input is **123.5946.482**

scanf(“%3d%2d”, &x, &y);

Data Stream with scanf()

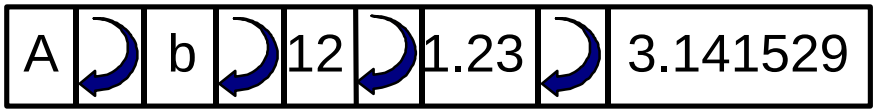
```
main()
{
      char letter1, letter2, letter3;
      int num1, num2;
      float fpnum;
      double dblnum;

      scanf("%c", &letter1);
      scanf("%c", &letter2);
      scanf("%c", &letter3);
      scanf("%d", &num1);
      scanf("%f", &fpnum);
      scanf("%lf", &dblnum);

      printf("int num is %d\n", num1);
      printf("fpnum is %f", fpnum);
      printf("letter1 is %c, value %d\n", letter1, letter1);
}
```

How would this program work with the following input data? (The box represents the input data stream, or buffer)



Data Stream with scanf()

input data stream



Why is this needed?

```
scanf("%c", &letter1);
scanf("%c", &letter2);
scanf("%c", &letter3);
scanf("%d", &num1);
scanf("%f", &fpnum);
scanf("%lf", &dblnum);

printf("int num is %d\n", num1);
printf("fpnum is %f\n", fpnum);
printf("letter1 is %c, value %d\n", letter1, letter1);
```

Dichiarazioni e Definizioni

Nella parti dichiarative di un programma C possiamo incontrare:

- ***definizioni*** (di variabile, o di funzione)
- ***dichiarazioni*** (di tipo o di funzione)

Definizione:

Descrive le proprieta` dell'oggetto definito e ne determina l'esistenza.

Ad esempio, *definizione* di una variabile:

```
int V; /* la variabile intera V
          viene creata (allocata in
          memoria) */
```

Dichiarazione:

Descrive soltanto delle proprieta` di oggetti, che verranno (eventualmente) creati mediante definizione.

Ad esempio, *dichiarazione* di un tipo non primitivo.

Dichiarazione di tipo

La dichiarazione di tipo serve per introdurre **tipi non primitivi.**

Associa ad un **tipo di dato** non primitivo un **identificatore** (scelto arbitrariamente dal programmatore).

Aumenta la leggibilita` e modificabilita` del programma.

In C, per dichiarare un nuovo tipo, si utilizza la parola chiave **typedef.**

Tipi scalari non primitivi

In C sono possibili dichiarazioni di tipi scalari non primitivi:

- tipi ***ridefiniti***
- tipi ***enumerati***

Tipo ridefinito

Un nuovo identificatore di tipo viene associato ad un tipo gia` esistente (primitivo o non).

Sintassi:

```
typedef    TipoEsistente  NuovoTipo;
```

Esempio:

```
typedef int MioIntero;
MioIntero X,Y,Z;
int W;
```


Tipo enumerato

Un **tipo enumerato** viene specificato attraverso l'esplicitazione di un **elenco di valori** che rappresenta il suo **dominio**.

Sintassi:

```
typedef enum {a1, a2, a3, ... , an} EnumType;
```

Una volta dichiarato, un tipo enumerato puo` essere utilizzato per definire variabili.

Ad esempio:

```
typedef enum{nord,sud,est,ovest} direzione;
direzione D=nord;
```

Proprieta`:

Dati di tipo enumerato sono **enumerabili**: il dominio e` strettamente ordinato, in base all'ordine testuale con cui si indicano gli elementi del dominio nella dichiarazione.

Ad esempio:

```
typedef enum{nord,sud,est,ovest} direzione;
```

- nord *precede* sud
  est *segue* nord e sud
  ecc.

TIPI ENUMERATIVI

Sono tipi **semplici user-defined** che implicano l'enumerazione esplicita dei valori che la variabile potrà assumere.

Sintassi C

```
enum {v1, v2, v3, ... vn} nome_var;
```

v1, v2, ...vn sono tutti e soli i valori che la variabile potrà assumere.

- i valori sono di solito espressi tramite nomi simbolici
- l'**ordine** di enumerazione definisce le relazioni tra i valori (v1<v2<v3 ....< vn)
- ad ogni valore enumerativo viene associato un valore di tipo integral

```
enum {lu, ma, me, gio, ve, sa, do} giorno;

enum {verde, giallo, rosso} semaforo;
```

Tipo enumerato in C

In C il tipo enumerato equivale a una ridefinizione del tipo int:

Ad ogni elemento del dominio viene rappresentato come un **intero**, che viene utilizzato nella valutazione di espressioni, relazioni ed assegnamenti.

- ☞ **enum** in C ha stessa occupazione, stesso range e **stesso utilizzo** di **int**

Convenzione (default):

Il primo elemento del dominio viene rappresentato con il valore 0, il secondo con 1, etc.

Esempio:

```
typedef enum{lu,ma,me,g,v,s,d}Giorno;
/*lu=0, ma=1, me=2,..d=6*/
typedef enum{cuori,picche,quadri,
             fiori} Carta;
/*cuori=0, picche=1,…*/
Carta C1, C2, C3, C4, C5;
Giorno  G;

G=lu;   ➠ G=0
G=ma    ➠ G=1
...
C1=cuori   ➠ 0
  ...
```

- ☞ L'utilizzo di tipi ottenuti per enumerazione rende piu` leggibile il codice.
- ☞ Un identificatore di un valore scalare definito dall'utente (ad es., l) deve comparire nella definizione di **un solo** tipo enumerato.

typedef enum{lu, ma, me, g, v, s, d} Giorno;
typedef enum{lu,ma,me} PrimiGiorni; /***scorretto** */

Operatori sul Tipo Enumerato

Il tipo **enum**, e` un tipo **totalmente ordinato** (e` inoltre un *Integral Type*).

Essendo i dati di tipo enumerato mappati su interi, su di essi sono disponibili gli stessi operatori visti per gli interi. In particolare:

Operatori relazionali:

lu < ma ➠ 0<1 ➠ *vero*
lu <= v ➠ 0<5 ➠ *vero*
lu >= s ➠ 0>6 ➠ *falso*
cuori <=quadri ➠ 0<2 ➠ *vero* ...

☞ Valori di tipo enumerato non sono stampabili ne` leggibili (non esiste, cioe` una direttiva di formato specifica per gli enumerati)

```
typedef
enum{cuori,picche,quadri,fiori}seme;
seme  S;
  ...
 if(S==cuori)
     printf("%s","cuori");
```


Tipi enumerati in C

E` possibile forzare il meccanismo di associazione automatico di interi a valori del dominio di tipi enum.

```
typedef enum{Nord=90,Sud,Est,
                Ovest}direzione;
/* Nord=90,Sud=91,Est=92,Ovest=93 */
direzione D;
```

Attenzione:

d = 0; /* **Viene accettato ed eseguito!** */

☞ Non c'e` controllo sugli estremi dell'intervallo di interi utilizzato!

Tipi enumerati

Esempio:

Vogliamo realizzare il tipo **boolean**: D={*falso*, *vero*}

- E` un tipo predefinito in altri linguaggi di programmazione (ad esempio, in Pascal).
- **Non e` previsto in C**, dove pero`:
    - il valore 0 (zero) indica ***FALSO***
    - ogni valore diverso da 0 indica ***VERO***

Prima soluzione:

Per definire le due costanti booleane:

```
#define FALSE      0
#define TRUE  1
```

☞ **#define** e` una direttiva del *preprocessore* C: provoca una sostituzione nel testo:
- dove c'e` "FALSE" -> 0
- dove c'e` "TRUE" -> 1

(non si alloca spazio in memoria)

Seconda soluzione:

Uso del tipo enumerato:

```
typedef enum {false, true} Boolean;
Boolean flag1,flag2;

 flag1 = true;
 if (flag1) .../* flag vale 1 */

 flag2 = -37 /*  !!!  */
 if (flag2).../*funziona lo stesso!*/
```

Equivalenza tra tipi di dato

Quando due oggetti hanno lo stesso tipo? Dipende dalla realizzazione del linguaggio. Due possibilita`:

- equivalenza **strutturale**
- equivalenza **nominale**

Equivalenza strutturale:

due dati sono considerati di tipo equivalente se hanno la stessa struttura.

Ad esempio:

```
typedef int     MioIntero;
typedef int     NuovoIntero;
MioIntero    A;
NuovoIntero  B;
```

A e B hanno lo stesso tipo.

Equivalenza nominale:

due dati sono considerati di tipo equivalente solo l'identificatore di tipo che compare nella loro definizione e` lo stesso.

Ad esempio:

```
typedef int     MioIntero;
typedef int     NuovoIntero;
MioIntero    A;
NuovoIntero  B;
```

A e B vengono considerati di tipo diverso.

Equivalenza di tipo in C:

Lo standard non stabilisce il tipo di equivalenza da adottare.

☞ Per garantire la portabilita`, e` necessario sviluppare programmi che presuppongano l'equivalenza nominale.

Summary

- Primary Data Types

char	holds an integer (whole number) decimal value
	uses one byte of storage space; one character
int	holds an integer (whole number) decimal value
	uses one word of storage; machine dependent
float	single precision floating point value (decimal fraction)
	storage is machine dependent
double	double precision floating point value (decimal fraction)
	storage is machine dependent

- Escape Characters

  single characters preceded by backslash; cause special action

- printf() and scanf() functions

  basic functions for producing formatted program input and output

Assignment

- Simple, with constant or other variables:

```
float salary, initial_salary, raise;
salary = 9500.f;    // lhs: non-const, with lvalue
initial_salary = salary;
```

- Pre/post increment/decrement: `++`, `--`

```
int num_objects = 0, grades [50], student = 6;
num_objects ++;    // increments num_objects by one
grades [++ student] = 0;    // student == 7 and
                            // grades [7] == 0
```

- Compound: `+=`, `-=`, `*=`, `/=`, `<<=`, `>>=`, `&=`, `|=`, `^=`

```
salary += raise;    // same as: salary = salary + raise;
```

- Quick to get used to and easy to read

Often used with pointers and indices.