3

Il linguaggio C
Istruzione di assegnamento e operatori

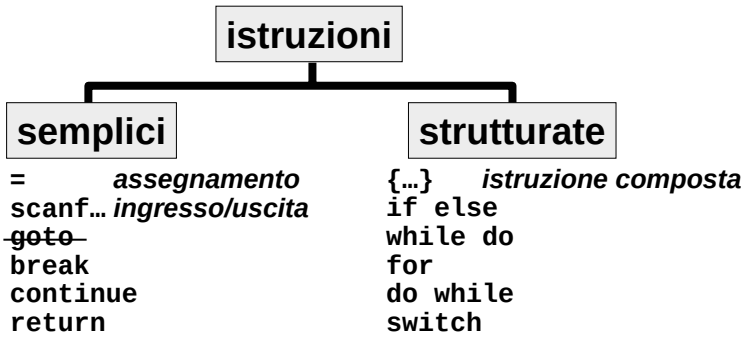
```
main()
{
/*definizioni variabili: */
char y='a'; /*codice(a)=97*/
int x,X,Y;
unsigned int Z;
float SUM;
double r;

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


3

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).



Istruzione di Assegnamento

- È l'istruzione con cui si modifica il valore di una variabile.

Mediante l'assegnamento, si scrive un nuovo valore nella cella di memoria che rappresenta la variabile specificata.

Sintassi:

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

Ad esempio:

```
int A, B;

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

Compatibilità di tipo ed assegnamento:

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

Assegnamento e Coercizione

Facendo riferimento alla gerarchia dei tipi semplici:

`int < long < float < double < long double`

consideriamo l'assegnamento:

V=E;

Quando la variabile V e` di un tipo di *rango inferiore* rispetto al tipo dell'espressione E l'assegnamento prevede la conversione forzata (***coercizione***) di E nel tipo di V.

Ad esempio:

```
float k=1.5;
int x;
x=k;   /*il valore di k viene convertito forzatamente
   nella sua parte intera: x assume il valore 1  ! */
```

Esempio:

```
main()
{
/*definizioni variabili: */
char y='a'; /*codice(a)=97*/
int     x,X,Y;
unsigned int Z;
float SUM;
double r;

/* parte istruzioni: */
X=27;
Y=343;
Z = X + Y -300;
X = Z / 10 + 23;
Y = (X + Z) / 10 * 10;
X = X + 70;
Y  = Y % 10;
Z = Z + X -70;
SUM = Z * 10;
x=y;
x=y+x;
r=y+1.33;
x=r;
}
```


Assegnamento come operatore

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

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

Ad esempio:

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

K=valore+100;

M=(K=K/2)+1;
```

3

Assegnamento abbreviato

In C sono disponibili operatori che realizzano particolari forme di assegnamento:
- operatori di incremento e decremento
- operatori di assegnamento abbreviato
- operatore sequenziale

- **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.

Incremento & decremento: esempi

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

B=++A;
B=A++;
C++;

int i, j, k;
k = 5;
i = ++k;
j = i + k++;

j = i + k++;
```

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

```
k = 5;
j = ++k * k++; /* quale effetto ?*/
```

Operatori 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;
- in ***v = v op (e)*** v viene valutato due volte.

Operatore sequenziale

Un'espressione sequenziale (o di ***concatenazione***) si ottiene concatenando tra loro più espressioni con l'operatore virgola (**,**).

(<espr1>, <espr2>, <espr3>, .. <esprN>)

- 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);
```

Precedenza e Associatività degli Operatori

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

- La **precedenza** (o priorità) indica l'ordine con cui vengono valutati operatori diversi;
- L'**associatività** indica l'ordine in cui operatori di pari priorità (cioè, stessa precedenza) vengono valutati.

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

Precedenza & associatività degli operatori C

Precedenza	Operatore	Simbolo	Associatività
1 (max)	chiamate a funzione selezioni	`()` `[]` `->` `.`	a sinistra

2	operatori unari:
	op. negazione
	op. aritmetici unari
	op. incr. / decr.
	op. indir. e deref.
	op. sizeof
		
		`!` `~`
		`+` `-`
		`++` `--`
		`&` `*`
		`sizeof`
			a destra

3	op. moltiplicativi	`*` `/` `%`	a sinistra
4	op. additivi	`+` `-`	a sinistra

3

Precedenza & associatività degli operatori C (continua)

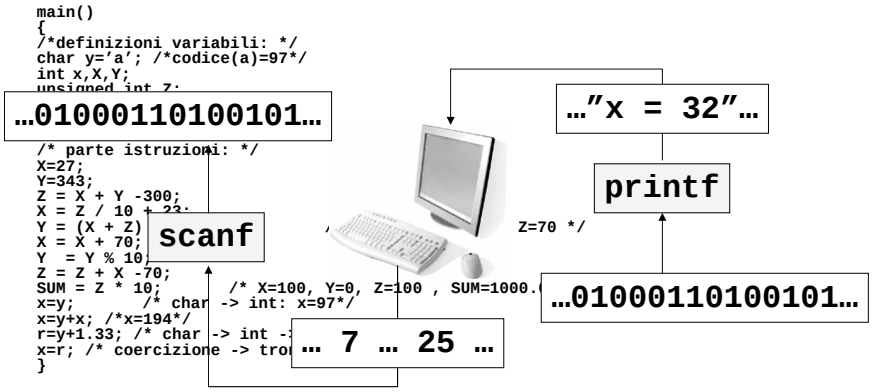
Precedenza	Operatore	Simbolo	Associatività
5	op. di shift	`>>` `<<`	a sinistra
6	op. relazionali	`<` `<=` `>` `>=`	a sinistra
7	op. uguaglianza	`==` `!=`	a sinistra
8	op. di AND bit a bit	`&`	a sinistra
9	op. di XOR bit a bit	`^`	a sinistra
10	op. di OR bit a bit	`|`	a sinistra
11	op. di AND logico	`&&`	a sinistra
12	op. di OR logico	`||`	a sinistra
13	op. condizionale	`?...:`	a destra
14	op. assegnamento e sue varianti	`=` `+=` `-=` `*=` `/=` `%=` `&=` `^=` `|=` `<<=` `>>=`	a destra
15 (min)	op. concatenazione	`,`	a sinistra

Esempi

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

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

Il linguaggio C
Istruzioni di input/output



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.
- In ogni versione ANSI C, esiste una *Libreria Standard* (**stdio**) che mette a disposizione alcune funzioni (dette *funzioni di libreria*) per effettuare l'input e l'output da e verso dispositivi.
- **Dispositivi standard di input e di output:**
  - per ogni macchina, sono periferiche predefinite (generalmente tastiera e video).

INPUT/OUTPUT

Le dichiarazioni delle funzioni messe a disposizione da tale libreria devono essere *incluse* nel programma:

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

- **`#include`** e` una direttiva per il **preprocessore** C:
- nella fase precedente alla compilazione del programma ogni direttiva "**#**..." viene eseguita, provocando delle modifiche testuali al programma sorgente.
- Nel caso di **`#include <nomefile>`**:

  viene sostituita l'istruzione stessa con il contenuto del file specificato.

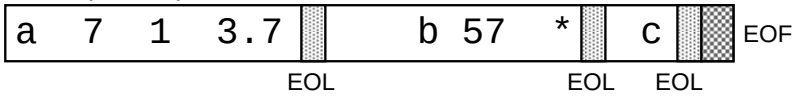
INPUT/OUTPUT

Il C vede le informazioni lette/scritte da/verso i dispositivi standard di I/O come file *sequenziali*, cioè **sequenze di caratteri** (o *stream*).

- Gli *stream* di input/output possono contenere dei caratteri di controllo:
  - End Of File (EOF)
  - End Of Line (EOL)

Sono disponibili funzioni di libreria per:

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



INPUT/OUTPUT CON FORMATO

- Nell'I/O con formato occorre specificare il formato (*tipo*) 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 specificato mediante apposite ***direttive di formato***; ad esempio **`%d`**, **`%f`**, **`%s`** ecc.

LETTURA CON FORMATO: `scanf`

E' una particolare forma di assegnamento: la scanf assegna i valori letti alle variabili specificate come argomenti (nell'ordine di lettura).

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

Ad esempio:

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

LETTURA CON FORMATO: scanf

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

- **scanf** legge una serie di valori in base alle specifiche contenute in **<stringa-formato>** e memorizza i valori letti nelle variabili specificate in **<sequenza-variabili>.**
- Se la **<stringa-formato>** contiene N direttive (del tipo %..), è necessario che le variabili specificate nella **<sequenza-variabili>** siano esattamente N.

- restituisce il numero di valori letti e memorizzati, oppure EOF in caso di end of file :

```
int X, Y, K;
K = scanf("%d%d", &X, &Y);
```

➔ se vengono immessi da input i due valori 100 e -25, le variabili X,Y e K assumeranno i seguenti valori:

X=100 Y=-25 K=2

scanf & formato

Ogni direttiva di formato prevede dei separatori specifici:

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

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

Osservazioni:

- 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 "**:**".

SCRITTURA CON FORMATO: printf

La **printf** viene utilizzata per fornire in uscita il valore di una variabile, o, più in generale, il risultato di una espressione:

```
printf(<stringa-formato>[,<sequenza-elementi>]);
```

- Anche in scrittura è necessario specificare (mediante una **<stringa-formato>**) il formato dei dati che si vogliono stampare.
- **<sequenza-elementi>** è una lista di ***espressioni*** (tante quante le direttive di formato contenute nella **<stringa-formato>).**

Ad esempio:

```
int X=19;
float Y=2.5;
printf("%d%f", X, X+Y);
```


printf

```
printf(<stringa-formato>[,<sequenza-elementi>]);
```

- **printf** scrive una serie di valori in base alle specifiche contenute in **<stringa-formato>.**
- I valori visualizzati sono i risultati delle espressioni indicate nella **<sequenza-elementi>.**
- La **printf** restituisce il numero di caratteri scritti.
- La stringa di formato della **printf** può contenere sequenze costanti di caratteri da stampare (nell'ordine indicato).

Ad esempio:

```
int X=19, K;
float Y=2.5;
K=printf("Risultato: %d%f\n", X, X+Y);
```

Effetti:

```
int X=19, K;
float Y=2.5
```

3

FORMATI COMUNI

- Formati più comuni:

int	**%d**
float	**%f**
carattere singolo	**%c**
stringa di caratteri	**%s**

- Caratteri di controllo:

newline	**\n**
tab	**\t**
backspace	**\b**
form feed	**\f**
carriage return	**\r**

- Per la stampa del carattere '**%**' si usa: **%%**

ESEMPIO

```
#include <stdio.h>
main()
{int   k;
scanf("%d",&k);
printf("Quadrato di %d: %d\n",k,k*k);
}
```

Esempio:
Se in ingresso viene immesso il dato: **3**
La **printf** stampa:

ESEMPIO

Rivediamo l'esempio visto inizialmente:

```
/*programma che, letti due numeri a terminale, ne stampa
  la somma*/

#include <stdio.h>

main()
{ int X,Y;  /* p. dichiarativa */

  scanf("%d%d",&X,&Y);/*lettura dei due dati*/
  printf("%d",X+Y);/* stampa della loro somma */
}
```

Dati da input i due valori 26 e -32, il programma stampa:

ESEMPIO

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

- Se in ingresso vengono dati:

```
ABC 3 7.345
```

- la **scanf** effettua i seguenti assegnamenti:

```
char c1
char c2
char c3
int i
float x
```

ESEMPIO

```
#include <stdio.h>
main()
{char Nome='A';
char Cognome='C';
printf("%s\n%c. %c. \n%s\n",
         "Programma scritto da:",
               Nome, Cognome,"Fine");
}
```

Stampa:

Esempio

Esempio:

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

```
#include <stdio.h>

main()
{ char a;

printf("Inserire un carattere:  ");
scanf("%c",&a);
printf("\n%c vale %d in decimale, %o in ottale \
       e %x in hex.\n",a, a, a, a);
}
```

Effetti dell'esecuzione:

```
Inserire un carattere:  A

A vale    in decimale,    in ottale e    in hex.
```

Esercizio

Calcolo dell'orario previsto di arrivo.

Scrivere un programma che legga tre interi positivi da terminale, rappresentanti l'orario di partenza (ore, minuti, secondi) di un vettore aereo, legga un quarto intero positivo rappresentante il tempo di volo in secondi e calcoli quindi l'orario di arrivo.

Prima specifica:

```
main()
{ /*dichiarazione variabili: occorrono tre
  variabili intere per l'orario di partenza ed
  una variabile intera per i secondi di volo.  */
  /*leggi i dati di ingresso */
  /*calcola l'orario di arrivo */
  /*stampa l'orario di arrivo */
}
```


Soluzione:

```
#include <stdio.h>
main()
{ /* dichiarazione dati */

/* leggi i dati di ingresso*/



/* calcola l'orario di arrivo*/



/* stampa l'orario di arrivo*/

                                                    ;
}
```