

Dal Pascal al C

Parte 2^a

Dichiarazioni di dati

Nel C come nel Pascal e come nella maggior parte dei linguaggi di programmazione, prima di utilizzare un identificatore è necessario definirlo.

Nel Pascal il gruppo delle costanti, quello dei tipi e quello delle variabili sono riconoscibili perché preceduti dalle parole riservate **const**, **type**, **var**.

Passiamo ora a vedere come si definiscono costanti, tipi e variabili nel linguaggio C.

Dichiarazione di costanti

Esistono due modi in C per definire una costante:

1. mediante la direttiva **#define** si associa ad un identificatore un dato costante di tipo numerico, carattere o stringa, ecco alcuni esempi:
#define PI 3.14 (definisce una costante di tipo numerico)
#define UNO 1 (definisce una costante di tipo numerico)
#define N 100 (definisce una costante di tipo numerico)
#define PRIMO 'a' (definisce una costante di tipo carattere)
#define NOME "pino" (definisce una costante di tipo stringa)
#define a_capo '\n' (definisce una costante di tipo carattere)
2. mediante la parola chiave **const** del C si dichiarano delle variabili di tipo qualsiasi il cui valore non può cambiare; tali variabili possono essere usate in tutto e per tutto come le altre variabili ma, non si può riassegnare loro un nuovo valore dopo l'inizializzazione

Dichiarazione di tipo

Nel C gli identificatori di tipo sono introdotti utilizzando la parola chiave **typedef**, vediamo degli esempi:

- **typedef int matricola;**
l'identificatore **matricola** è un identificatore che dopo la sua dichiarazione può essere usato al posto all'identificatore di tipo primitivo **int**
- **typedef float pressione, temperatura;**
sia l'identificatore **pressione** che l'identificatore **temperatura** possono essere utilizzati al posto di **float**
- **typedef enum { rosso, blu, verde, giallo, bianco } colore;**
l'identificatore **colore** è un identificatore che corrisponde ad un tipo enumerato

Mentre nel Pascal è necessario usare una definizione per ogni identificatore di tipo, nel C è possibile definire più identificatori di tipo utilizzando una sola definizione (es. **pressione**, **temperatura**). Viceversa nel C per definire tipi strutturalmente diversi occorre usare più volte la parola chiave **typedef**.

Il simbolo ***** è usato in C per definire tipi puntatori.

Le parentesi quadre **[]** sono usate in C per definire tipi array.

Le parentesi tonde **()** non hanno analogie con il Pascal e sono utilizzate nella costruzione di tipi dalla struttura complessa o di tipi funzione.

Dichiarazione di variabili

Nel C è possibile inizializzare le variabili all'atto della loro dichiarazione come nel seguente esempio:

```
typedef float temperatura;  
int somma = 0;  
char ch = ' ';  
temperatura t0 = -273.16;
```

Nell'esempio:

- (1) viene definito il tipo **temperatura**;
- (2) la variabile **somma** non solo è dichiarata come variabile di tipo **int** ma è anche inizializzata con il valore **0**;
- (3) la variabile **ch** è dichiarata di tipo **char** e inizializzata a **' '**;
- (4) La variabile **t0** è dichiarata di tipo **temperatura** e inizializzata a **-273.16**.

Espressioni ed istruzioni di assegnazione (1)

- **a=a+b**; è una istruzione di assegnazione, equivalente ad **a:=a+b** del Pascal
- **a=sqrt(a)**; è ancora una istruzione di assegnazione, equivalente alla istruzione **a:=sqrt(a)**; del Pascal; il C mette a disposizione del programmatore un insieme di funzioni di libreria molto più ricco di quello messo a disposizione dal Pascal
- **a=b=c**; corrisponde a due istruzioni di assegnazione ed ha un comportamento equivalente alla sequenza di istruzioni **b:=c**; **a:=c**; del Pascal; il calcolatore esegue le assegnazioni da destra verso sinistra; **b=c** oltre ad assegnare alla variabile **b** il valore di **c**, restituisce come risultato il valore di **c** che a sua volta viene assegnato ad **a**
- **a++**; aggiunge **1** ad **a** ed il risultato dell'addizione viene assegnato ad **a**; il comportamento è equivalente alla istruzione **a:=a+1** del Pascal; l'operatore postfisso **++** consente di esprimere sinteticamente l'operazione di incremento unitario
- **a+=b**; somma **b** ad **a** ed assegna il risultato della somma ad **a**; il comportamento è equivalente a quello della istruzione **a:=a+b** del Pascal; nel C, oltre all'operatore **+=** esistono gli operatori ***=**, **-=**
- **a+b**; è una istruzione sintatticamente corretta che restituisce il valore **a+b**; l'istruzione non ha alcun effetto sul programma

Espressioni ed istruzioni di assegnazione (2)

Può sembrare strano che il C consenta di scrivere istruzioni che contengono solo delle espressioni senza alcun effetto sulle variabili. La spiegazione sta nel fatto che nel C non esistono le procedure e che l'idea di sottoprogramma è esprimibile solo attraverso l'uso delle funzioni. Una espressione può contenere la chiamata ad una funzione e la funzione può avere degli effetti collaterali; è attraverso questi effetti collaterali che in C una funzione può essere utilizzata per simulare il comportamento di una procedura. Vediamo due esempi a tal proposito.

- **printf("ciao");** è una espressione che contiene una chiamata alla funzione del C **printf**, il risultato della funzione non viene utilizzato e la funzione ha come effetto solo l'effetto collaterale di stampa sul dispositivo di uscita
- **scanf("%d",&num);** è una espressione che contiene una chiamata alla funzione del C **scanf**, che ha come effetto collaterale quello di leggere dalla tastiera un intero e di modificare il valore della variabile **num**

Operatori di assegnamento

Operatore	Utilizzo	Esempio	Istruzione equivalente
++	Incremento unitario	x++;	x = x + 1;
--	Decremento unitario	x--;	x = x - 1;
+=	Incremento	x+=y;	x = x + y;
-=	Decremento	x-=y;	x = x - y;
=	Moltiplicazione	x=y;	x = x * y;
/=	Divisione	x/=y;	x = x / y;
%=	Resto della divisione intera	x%=y;	x = x % y;

Alcune funzioni matematiche

Per poter utilizzare le funzioni matematiche è necessario inserire la libreria **math.h** all'inizio del programma con la direttiva **#include <math.h>**.

- **pow(x,y)** calcola la potenza di **x** con esponente **y** (**x** e **y** devono essere di tipo **double** così come lo è il risultato del calcolo)
- **sqrt(x)** calcola la radice quadrata del numero **x**
(ad es. per calcolare l'ipotenusa di un triangolo rettangolo:
ipotenusa = sqrt (pow(cat1, 2.0) + pow(cat2 , 2.0)) ;
dove l'esponente dei cateti è **2.0** perché gli argomenti di **pow** sono **double**)
- **ceil(x)** arrotonda un numero all'intero superiore (**ceil(3.7)** produce **4**)
- **floor(x)** fornisce un numero senza parte decimale (**floor(3.7)** produce **3**)
- **sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), log(x), log10(x), exp(x)**, sono tutte funzioni di numeri **x** di tipo **double** che restituiscono valori **double**

Il casting per la conversione di tipo (1)

Per **conversione di tipo** si intende un'operazione volta a trasformare un valore di un certo tipo in un valore di altro tipo.

Quando in una espressione si utilizzano dati dello stesso tipo, il tipo del risultato di calcolo è ben definito ed è lo stesso degli operandi. Il problema nasce quando in una operazione gli operandi sono di diverso tipo; per esempio in:

int base;

float altezza;

area = base * altezza;

si tratta di decidere di che tipo sia il valore della variabile **area**.

Non sempre una conversione di tipo preserva il valore; per esempio nella conversione da **float** a **int** si ha una perdita di precisione: il compilatore non fa altro che scartare la parte frazionaria; se poi il valore non è rappresentabile dal tipo **int** il risultato è indefinito. È da notare che le operazioni di conversione riguardano il valore della variabile in quel particolare contesto e non il tipo della variabile stessa.

Il casting per la conversione di tipo (2)

Le conversioni eseguite direttamente dal compilatore si chiamano **conversioni implicite**. Nel seguente esempio un valore **non intero** viene assegnato ad una variabile di tipo **int**.

```
#include <stdio.h>
main() {
    int a = 5;
    float b = 3.56;
    a=b;
    printf(“%d”, a);
}
```

L'output del programma è **3**.

Il casting per la conversione di tipo (3)

Nel seguente programma un valore **intero** viene assegnato ad una variabile di tipo **float**.

```
#include <stdio.h>
main() {
    int a = 5;
    float b = 3.56;
    b=a;
    printf(“%f”, b);
}
```

L'output del programma è **5.0**. I caratteri “%f” specificano il formato per i numeri reali (**float**).

Il casting per la conversione di tipo (4)

Il linguaggio C consente al programmatore di richiedere la conversione di tipo e in questo caso si parla di **conversioni esplicite** (cioè decise dal programmatore) o **cast**.

Il **casting** è l'azione che consente di trasformare la rappresentazione di un dato dal suo tipo originale a un altro.

L'operatore **cast**, rappresentato da una coppia di parentesi tonde che racchiudono il tipo di dato, è un operatore a tutti gli effetti e segue la seguente sintassi:

(tipo) espressione;

Ad esempio: **b = (int) a;**

Il casting per la conversione di tipo (5)

```
#include <stdio.h>
main() {
    int a = 5;
    int b = 2;
    float c = 3.56;
    float p;
    p = a/b + c;
    printf("%f", p);
}
```

```
#include <stdio.h>
main() {
    int a = 5;
    int b = 2;
    float c = 3.56;
    float p;
    p = (float) a/b + c;
    printf("%f", p);
}
```

Con il programma di sinistra si ottiene in output **5.56**, perché la divisione intera **a/b** produce il valore **2** che viene poi sommato a **3.56**.

Con il programma di destra si ottiene in output **6.06**, perché il **casting** applicato alla divisione **a/b**, produce correttamente il valore **2.5** che viene poi sommato a **3.56**.

Gli operatori di relazione

Nel linguaggio C gli **operatori di confronto** si indicano con i simboli:

- **==** per uguale
- **>** per maggiore
- **<** per minore
- **<=** per minore uguale
- **>=** per maggiore uguale
- **!=** per diverso

Esempi di espressioni booleane:

a >= c

b < d

f != g

Gli operatori logici

I **connettivi logici** che possono essere usati nel C sono:

- **&&** indica l'operazione di congiunzione (**And**)
- **||** indica l'operazione di disgiunzione (**Or**)
- **!** indica l'operazione di negazione (**Not**)

A causa della diversa priorità che hanno gli operatori logici rispetto agli operatori relazionali, con il linguaggio C si possono scrivere programmi più sintetici che con il Pascal: mentre nel Pascal gli operatori logici hanno la precedenza su quelli relazionali, nel C vale esattamente il contrario.

L'espressione del Pascal **(a>=0) and (b<>10)** può essere scritta in C, a causa della diversa priorità tra gli operatori, senza l'uso delle parentesi: **a>=0 && b!=10**.

Operatori logici binari

- ~ **operatore not bit a bit**, inverte tutti i bit dell'operando. Se ad esempio l'operando ha una codifica binaria **00100110**, la codifica binaria del risultato è **11011001**. Opera solo su interi o caratteri.
- & **operatore and bit a bit**. Gli operandi devono essere di tipo intero o carattere. Il risultato è l'**and** bit a bit dei due operandi. Ad esempio, se la rappresentazione binaria del contenuto delle variabili a e b è rispettivamente 00001111 e 01010101, il risultato dell'espressione **a&b** vale **00000101**.
- | **operatore or bit a bit**. Gli operandi devono essere di tipo intero o carattere. Il risultato è l'**or** bit a bit dei due operandi. Ad esempio, se la rappresentazione binaria del contenuto delle variabili a e b è rispettivamente **00001111** e **01010101**, il risultato dell'espressione **a|b** vale **01011111**.
- ^ **operatore xor bit a bit**. Gli operandi devono essere di tipo intero o carattere. Il risultato è l'**or esclusivo** bit a bit dei due operandi. Ad esempio, se la rappresentazione binaria del contenuto delle variabili a e b è rispettivamente **00001111** e **01010101**, il risultato dell'espressione **a^b** vale **01011010**.

Alcuni operatori particolari

- **operatore meno unario**, cambia segno all'espressione che lo segue.
- % operatore modulo**, come l'operatore **mod** del Pascal restituisce il resto della divisione del primo operando per il secondo. Gli operandi devono essere di tipo intero o carattere ed il secondo operando deve essere diverso da zero.
- <<, >>** sono gli **operatori di scorrimento**; accettano operandi di tipo intero o carattere. Operano sulla rappresentazione binaria del primo operando spostando, rispettivamente a **sinistra** o a **destra**, i bit di tale operando del numero di posizioni indicato dal secondo operando.
- , **operatore virgola** non ha alcuna analogia con gli operatori del Pascal. I suoi operandi sono espressioni. Viene prima valutato l'operando sinistro che influisce nell'operazione solo per i suoi effetti collaterali. Il risultato dell'espressione infatti è il risultato dell'operando destro. Ad esempio l'espressione **a=b,c** viene valutata da sinistra verso destra; prima il valore di **b** viene assegnato ad **a** e l'espressione **a=b** restituisce il valore di **b**; poi viene valutata l'espressione **c**, che restituisce il valore di **c**; quindi l'operatore virgola applicato ai valori di **b** e di **c** restituisce il valore di **c**.
- sizeof** questo operatore restituisce la dimensione dell'operando (numero di byte). La valutazione avviene a tempo di compilazione e non di esecuzione.

Esempi di espressioni sintatticamente corrette (1)

- Supponiamo che la variabile **ch** sia di tipo **char** e consideriamo l'espressione **ch<<2**. Se la rappresentazione binaria del valore di **ch** è **11001010**, l'espressione restituisce come risultato il carattere la cui rappresentazione binaria è **00101000**. Si noti che i bit che entrano da destra (da sinistra se l'operatore è **>>**) hanno valore **0**. In alcune implementazioni se l'operatore è **>>**, i bit che entrano da sinistra hanno valore uguale a quello del bit più significativo. (Scrivere un programma che verifichi il comportamento del compilatore in uso)
- L'espressione **a++ * ++b** viene valutata nel modo seguente: (1) vengono valutate le due espressioni **a++** e **++b**; la valutazione di **a++** restituisce **a**, la valutazione di **++b** restituisce **b+1**; contemporaneamente alla valutazione si manifestano gli effetti collaterali di incremento di **a** e **b**; (2) viene eseguito il prodotto. Se ad esempio prima della valutazione della espressione **a=5** e **b=7**, il risultato dell'espressione è **40** e dopo la valutazione **a** vale **6** e **b** vale **8**. (Scrivere un programma che dimostri quanto detto.)
- L'espressione **a++ * b++** viene valutata nel modo seguente: (1) vengono valutate le due espressioni **a++** e **b++**; la valutazione di **a++** restituisce **a**, la valutazione di **b++** restituisce **b**; contemporaneamente alla valutazione si manifestano gli effetti collaterali di incremento di **a** e **b**; (2) viene eseguito il prodotto. Se ad esempio prima della valutazione della espressione **a=5** e **b=7**, il risultato dell'espressione è **35** e dopo la valutazione **a** vale **6** e **b** vale **8**. (Scrivere un programma che dimostri quanto detto.)

Esempi di espressioni sintatticamente corrette (2)

- Al termine della valutazione dell'espressione **c=a=7,b=a+1** **a** vale **7**, **b** vale **8** e **c** vale **7**; l'espressione restituisce il valore **8**. Invece al termine della valutazione dell'espressione **c=(a=7,b=a+1)** **a** vale **7**, **b** vale **8** e **c** vale **8** e l'espressione restituisce **8**. (Scrivere un programma che dimostri quanto detto.)
- Si osservi come invertire tra loro due operandi possa portare ad effetti difficilmente controllabili. Si confrontino l'espressione **(++b==3) || (b>=6)** e l'espressione **(b>=6) || (++b==3)**. Supponiamo che prima della valutazione dell'espressione la variabile **b** valga **5**. se valutiamo la prima espressione il risultato è **1** (espressione **vera**), se invece valutiamo la seconda il risultato è **0** (espressione **falsa**). Entrambe le espressioni come effetto collaterale portano **b** a **6**. supponiamo invece che, prima della valutazione dell'espressione la variabile **b** valga **6**. entrambe le espressioni restituiscono il valore **1**, ma nel primo caso la variabile **b** assume il valore **7** mentre nel secondo caso il suo valore rimane invariato. Infatti, dato che **(b>=6)** è vera il C non valuta l'altro operando dell'**or**.
- L'espressione **a<b<c** è interpretata da sinistra verso destra, e cioè come **(a<b)<c**. Quindi la parte **(a<b)** restituisce il valore **1** (se vera) o **0** (se falsa) ed è questo valore che viene confrontato con **c**. Il risultato è molto diverso da quello del confronto contemporaneo di **b** con **a** e **c** che ci sarebbe potuto aspettare.

Input e output

Tutte le funzioni di ingresso/uscita possono essere attivate solo dopo aver utilizzato la direttiva **#include<stdio.h>** cioè dopo aver aggiunto al programma di cui si sta scrivendo il sorgente la libreria per le operazioni di **input/output standard**.

La funzione **scanf** è la funzione di lettura (acquisizione dati da tastiera) più usata in C.

La lettera finale **f** indica che l'acquisizione dei dati è fatta secondo un formato (**input formattato**).

La funzione **printf** è la funzione di scrittura più usata in C.

La lettera finale **f** indica che la visualizzazione dei dati è ottenuta specificando un formato (**output formattato**).

Input: la funzione scanf (1)

L'istruzione **scanf** indica una operazione sullo **standard input**, cioè l'unità standard del computer per acquisire dati dall'esterno (la tastiera).

Ad esempio **scanf("%d", &a);** fa entrare da tastiera un numero intero (%d) e lo assegna alla variabile **a**.

L'istruzione ha due argomenti, scritti tra parentesi tonde e separati dalla virgola: il primo, tra virgolette, indica il formato; il secondo contiene la variabile il cui valore viene acquisito dalla tastiera.

Il nome della variabile è preceduto dal carattere **&**, perché la funzione **scanf** richiede come argomento l'**indirizzo della variabile** e non il nome della variabile. Questo vale per i dati di tipo numerico, intero e floating point, e per i caratteri; invece per le stringhe formate da più di un carattere non si deve usare il carattere **&**.

I formati sono rappresentati con sequenze di caratteri, dette **specificatori di formato** o **formattatori**: essi iniziano con il carattere **%**, seguito da uno o più caratteri.

Input: la funzione scanf (2)

Gli specificatori di uso più comune sono:

- %d** (dati numerici interi)
- %f** (numeri float e double)
- %c** (dati formati da un solo carattere)
- %s** (dati stringa)

L'operazione di lettura con **scanf** permette di leggere con un'unica operazione tutti i caratteri digitati fino alla pressione del tasto **invio**. Se si vogliono leggere più dati, si devono indicare, nelle parentesi di **scanf**, tanti specificatori di formato e tante variabili quanti sono i dati da inserire:

```
scanf("%d %d", &a, &b);
```

L'utente del programma può inserire i dati richiesti o sulla stessa riga, separandoli con la barra spaziatrice, o con il ritorno a capo.

Output: la funzione printf

L'istruzione per visualizzare dati e messaggi sul video (**output**) si indica in C con **printf**; essa indica una operazione sullo **standard output**, cioè l'unità standard del computer per comunicare i risultati (il video).

printf riceve un numero variabile di parametri di cui il primo è una stringa.

Tale stringa specifica il formato della stampa che deve essere eseguita.

All'interno della stringa è possibile inserire, mescolati con i caratteri, un numero qualunque di **formattatori**. Vediamo un esempio:

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

```
main() {  
    int x = 5;  
    int y = 6;  
    printf("x = %d y =%d \n", x, y);  
}
```

Il programma stampa **x=5** e **y=6** e poi salta a riga nuova. Le due sottostringhe **%d** sono esempi di formattatori; esse specificano il formato dei due dati che devono essere stampati (in questo caso due interi).

La funzione **printf** scandisce da sinistra verso destra la stringa che specifica il formato; i caratteri non formattatori vengono stampati ed i formattatori vengono associati, con una corrispondenza posizionale, ai parametri successivi alla stringa..Quando nella scansione della stringa viene incontrato un formattatore, il programma stampa il corrispondente parametro.

Output: formattatori

%c per stampare un carattere

%s per stampare una stringa; il parametro corrispondente deve essere un puntatore a carattere.

%d , **%i** per stampare un intero con segno

%u per stampare un intero senza segno

%o per stampare un intero in notazione ottale

%x , **%X** per stampare un intero in notazione esadecimale; i caratteri a, b, c, d, e, f sono stampati in minuscolo (**%x**) o in maiuscolo (**%X**)

%f , **%lf** per stampare un float (**%f**) o un double (**%lf**)

%e , **%E** per stampare in notazione scientifica con mantissa ed esponente, l'esponente è preceduto da una **e** minuscola (**%e**) o da una **E** maiuscola (**%E**) ed è sempre composto da **3** cifre

%g , **%G** per stampare un float o un double; se l'esponente è minore di **-4** o maggiore della precisione selezionata viene utilizzata la notazione scientifica (formato **e** o formato **E**, a seconda, rispettivamente, di **%g** o **%G**)

%p per stampare l'indirizzo contenuto in una variabile di qualunque tipo puntatore

%% per stampare il carattere **%**

Output: opzioni di formato

Nella specificazione del formato, dopo il carattere % e prima della lettera che identifica il formato, si possono aggiungere altre opzioni di formato; nell'ordine:

– il dato viene allineato a sinistra, il default è l'allineamento a destra

0 (zero) in alternativa al meno (–) per indicare che le posizioni a sinistra, occupate dal numero da stampare devono essere riempite con zeri

un numero intero per impostare il numero di posizioni all'interno delle quali il valore deve essere visualizzato; oppure

due numeri, separati dal punto, per indicare il numero di posizioni e il numero di cifre decimali della rappresentazione

Ad esempio se la variabile **a** assume il valore **5**, l'istruzione:

```
printf("Risultato del calcolo = %–7.2f", a);
```

produce il seguente output a video con il numero **allineato a sinistra**, all'interno di **7** posizioni, con **2** cifre decimali:

```
Risultato del calcolo = 5.00
```

Esempi di specificazione di formato (1)

Consideriamo le seguenti dichiarazioni di variabili e le uscite (colonna a destra della tabella seguente) delle corrispondenti istruzioni **printf** (colonna a sinistra sempre della tabella seguente):

```
int a = 5;  
float b = 3.4567;  
char nazione[7] = "Italia";
```

(il numero indicante la stringa **nazione** è aumentato di 1, perché occorre considerare il carattere **terminatore di stringa** **\0**, che viene aggiunto automaticamente.

Esempi di specificazione di formato (2)

Istruzioni	Output
<code>printf("%d \n", a);</code>	5
<code>printf("%15d \n", a);</code>	5
<code>printf("%015d \n", a);</code>	0000000000000005
<code>printf("%f \n", b);</code>	3.456700
<code>printf("%15.2f \n", b);</code>	3.46
<code>printf("%10d %15.2f \n", a, b);</code>	5 3.46
<code>printf("%-10d %-15.2f \n", a, b);</code>	5 3.46
<code>printf("%s \n", nazione);</code>	Italia
<code>printf("%20s \n", nazione);</code>	Italia
<code>printf("%-20s \n", nazione);</code>	Italia
<code>printf("%-20s %15.2f \n", nazione, b);</code>	Italia 3.46

Altre funzioni di input/output del C

Se si vuole stampare sulla carta della **stampante** anziché sul video, si deve utilizzare, anziché l'istruzione **printf**, l'analogo **fprintf** (stampa formattata su file), che funziona con le stesse modalità di **printf**; occorre solo specificare il nome della stampante, **stdprn** (**standard printer**, stampante standard o predefinita), per indicare la ridirezione dell'output dal video alla stampante (da **stdout**, standard output o video, a stampante); ad esempio:

```
fprintf(stdprn, "Totale importi = %f \n", Tot);
```

Nella fase di testing di un programma che richiede l'output su stampante oppure nel caso in cui la stampante non fosse disponibile, si può sostituire **stdprn** con **stdout**, per ottenere i risultati sul video, invece che sulla carta della stampante.

Per l'input e l'output standard di singoli caratteri, il linguaggio C possiede anche le funzioni **getchar** per l'acquisizione e **putchar** per la visualizzazione.

Per le stringhe le funzioni sono **gets** e **puts**.