

Introduzione a C

Table of Contents

- [Introduzione a C](#)
 - [Table of Contents](#)
 - [Tipi di dato](#)
 - [Regole di scrittura](#)
 - [Sintassi per dichiarazione delle variabili](#)
 - [Assegnamento variabili](#)
 - [Operatori aritmetici e di casting](#)
 - [Operatori aritmetici](#)
 - [Operatore di casting](#)
 - [Operatori Logici Booleani](#)
 - [Astrazione procedurale](#)
 - [Main function](#)
 - [Scanf](#)
 - [Printf](#)
- [Astrazione sul controllo](#)
 - [If](#)
 - [Caso 1](#)
 - [Caso 2](#)
 - [Caso 3](#)
 - [Loops](#)
 - [For](#)
 - [While](#)
 - [Do-while](#)
 - [Numeri random](#)
 - [Rand](#)
 - [Srand](#)
 - [Vettori](#)
 - [Matrici](#)
- [Puntatori](#)
 - [Dichiarazione di puntatori](#)
 - [Operatore &](#)
 - **[Operatore *](#)**
 - [Esempio](#)
 - [Puntatori con vettori](#)
 - [Stringhe \(Array di caratteri\)](#)
 - [strlen\(\)](#)
 - [gets\(\)](#)
 - [puts\(\)](#)
 - [strcpy\(\)](#)
 - [strncpy\(\)](#)
 - [strcat\(\)](#)

- `strncat()`
- `strcmp()`
- Funzioni
 - Definire e chiamare una funzione
 - Return nelle funzioni
 - Passaggio di parametri
- Esercizi da svolgere:

Tipi di dato

- `int`: Integer type (%d)
- `short int`: Short integer type (%hd)
- `long`: Long integer type (%ld)
- `long long`: Long long integer type (%lld)
- `float`: Single-precision floating-point type (%f)
- `double`: Double-precision floating-point type (%lf)
- `long double`: Extended-precision floating-point type (%Lf)
- `char`: Character type (%c)

Unsigned Types

- `unsigned int`: Unsigned Integer type (%u)
- `unsigned short`: Unsigned Short integer type (%hu)
- `unsigned long`: Unsigned Long integer type (%lu)
- `unsigned long long`: Unsigned Long long integer type (%llu)

Regole di scrittura

- Indentazione
- Nome significativo delle variabili
- Commenti nel caso fossero necessari
- Evitare l'uso di codici superflui
- Utilizzo graffe
- Salto delle righe tra dichiarazione e corpo del codice

Sintassi per dichiarazione delle variabili

Specificatore tipo (`int`, `float`, `char`) + nome variabile (o lista di variabili separate da una virgola) + ;

```
int x, y;
```

Assegnamento variabili

Specificatore tipo (int, float, char) + nome variabile (o lista di variabili separate da una virgola) +

```
int x = 2;  
int y = 5;  
  
y = x;
```

Operatori aritmetici e di casting

Operatori aritmetici

- `+`: somma;
- `-`: differenza;
- `*`: moltiplicazione;
- `/`: diviso;
- `%`: modulo (ritorna il resto di una divisione).

Operatore di casting

Operatore che mi permette di cambiare il tipo di dato in un altro.

```
#include <stdio.h>  
  
int main() {  
    int a = 5, b = 4;  
    float c;  
  
    c = (float) (a+b)/2;  
    printf("La media è : %f", c );  
}
```

Nonostante sia un'operazione tra interi riusciamo ad ottenere i decimali risultanti da quella operazione.

Operatori Logici Booleani

- **AND** &&
- **OR** ||
- **NOT** !

A	B	A AND B	A OR B	NOT A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Esiste anche il tipo di dato bool che permette di memorizzare "vero" o "falso" all'interno delle variabili; per poterlo utilizzare è necessario include la libreria `#include <stdbool.h>`.

Astrazione procedurale

Il programma viene suddiviso in funzioni.

Main function

La funzione main è l'unica funzione che viene eseguita autonomamente e la sua presenza è indispensabile per il funzionamento del programma.

```
#include <stdio.h>

int main(void) {

    return 0;
}
```

- void: indica che nella funzione non sono presenti dei parametri
- return 0;: se viene eseguito tutto correttamente la funzione ritorna il valore 0 (il valore è un intero perchè la funzione è di tipo int).

Scanf

Funzione di input, che serve per inserire dei dati, tramite tastiera, all'interno del programma

```
#include <stdio.h>

main() {
    int a = 0;
```

```
scanf("%identificatore", &a);  
}
```

In questo caso sto prendendo in input un valore da tastiera e lo sto assegnando alla variabile `a`, questa cosa non avviene direttamente, ma mediante l'operatore `&` che serve per indicare l'indirizzo della variabile.

Printf

Questa funzione serve a mostrare a video una frase o una variabile, anche in questi casi si ricorre all'utilizzo degli identificatori.

```
#include <stdio.h>  
  
main() {  
    int a = 7;  
  
    printf("La variabile a, ha valore: %d", a);  
}
```

Output del programma:

La variabile a, ha valore: 7

Astrazione sul controllo

If

Il blocco if serve a eseguire un'operazione nel caso venga verificata o meno una condizione. Letteralmente il blocco if significa "se la condizione è vera, fai questo, altrimenti non farlo o fai qualcos'altro".

Caso 1

```
if(condizione){  
    "fai qualcosa";  
}
```

In questo caso il blocco if esegue solo l'operazione solo se la condizione viene verificata.

Caso 2

```
if(condizione){  
    "fai qualcosa";  
}else{  
    "fai altro";  
}
```

In questa situazione se la condizione non viene rispettata viene svolta un'altra operazione.

Caso 3

```
if(condizione){  
    "fai qualcosa";  
}else if(condizione 2){  
    "fai altro";  
}else{  
    "fai altro ancora";  
}
```

In questo caso se la prima condizione non viene verificata, controlla la seconda, se anche questa non viene verificata esegue l'operazione nel blocco `else`.

Loops

Blocchi che permettono di iterare delle operazioni. Esistono 3 differenti loops:

- for
- while
- do-while

For

Il for è il ciclo con contatore.

```
for (int i = 0; i < 100; i++) {  
    "operazione da eseguire fino a quando i >= 100";  
}
```

Il ciclo for è costituito da un header e un body. All'interno dell'header c'è l'inizializzazione della variabile contatore, la condizione di permanenza e l'incremento (o decremento) della variabile. Nel body vengono eseguite le operazioni da ripetere.

While

```
while (condizione) {  
    "fai qualcosa dopo aver controllato la condizione";  
}
```

Il while controlla sempre prima la condizione prima di eseguire l'operazione al suo interno.

Do-while

Il do-while è un loop a condizione finale, questo garantisce che il codice al suo interno venga eseguito almeno una volta; questo perchè se la condizione viene verificata il loop continua, fino a quando non viene più rispettata.

```
int x = 0;  
do{  
    scanf("%d", &x);  
}while(x<100);
```

In questo do-while l'operazione `scanf` viene ripetuta fino a quando il valore di x non supera 99.

Numeri random

Rand

La funzione `rand` genera un numero pseudo-casuale nell'intervallo da 0 a `RAND_MAX`. Per ottenere numeri in un intervallo specifico, è possibile utilizzare l'operazione modulo (%). Prima di utilizzare `rand`, è necessario inizializzare il generatore di numeri casuali con `srand`.

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

int main() {
    // Generazione di un numero casuale compreso tra 0 e RAND_MAX
    int numeroCasuale = rand();

    // Generazione di un numero casuale compreso tra 1 e 100
    int numeroCasualeLimitato = rand() % 100 + 1;

    printf("Numero casuale: %d\n", numeroCasuale);
    printf("Numero casuale tra 1 e 100: %d\n", numeroCasualeLimitato);

    return 0;
}
```

Srand

La funzione `srand` viene utilizzata per inizializzare il generatore di numeri casuali. Di solito, si utilizza come argomento il valore restituito dalla funzione `time(NULL)`. Ciò consente di generare sequenze di numeri casuali diverse ad ogni esecuzione del programma.

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

int main() {
    // Inizializzazione del generatore di numeri casuali
    srand(time(NULL));

    // Resto del codice

    return 0;
}
```


Vettori

I vettori sono strutture dati che consentono di memorizzare più elementi dello stesso tipo in una singola variabile. La dichiarazione di un vettore avviene specificando il tipo degli elementi e la dimensione del vettore.

```
#include <stdio.h>

int main() {
    // Dichiarazione e inizializzazione di un vettore di interi
    int numeri[5] = {1, 2, 3, 4, 5};

    // Accesso agli elementi del vettore
    printf("Elemento 0: %d\n", numeri[0]);
    printf("Elemento 1: %d\n", numeri[1]);

    return 0;
}
```

Matrici

Una matrice è una struttura bidimensionale che può essere vista come un vettore di vettori. La dichiarazione di una matrice avviene specificando il tipo degli elementi e le dimensioni della matrice.

```
#include <stdio.h>

int main() {
    // Dichiarazione e inizializzazione di una matrice di interi
    int matrice[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

    // Accesso agli elementi della matrice
    printf("Elemento in posizione (1, 1): %d\n", matrice[1][1]);

    return 0;
}
```

Puntatori

I puntatori sono variabili che contengono l'indirizzo di memoria di un'altra variabile. Consentono di manipolare direttamente la memoria, migliorando l'efficienza e la flessibilità del codice. I puntatori sono variabili che contengono l'indirizzo di memoria di un'altra variabile. Consentono di manipolare direttamente la memoria, migliorando l'efficienza e la flessibilità del codice.

Dichiarazione di puntatori

I puntatori vengono dichiarati utilizzando il simbolo * seguito dal tipo di variabile a cui puntano. Ad esempio, la seguente dichiarazione dichiara un puntatore a una variabile di tipo int:

```
int *p;
```

Questa dichiarazione alloca spazio per una variabile di tipo int, ma non assegna un indirizzo di memoria a tale variabile. Per assegnare un indirizzo di memoria a un puntatore, è necessario utilizzare l'operatore &.

Operatore &

L'operatore & viene utilizzato per ottenere l'indirizzo di memoria di una variabile. Ad esempio, la seguente istruzione assegna l'indirizzo di memoria della variabile x al puntatore p:

```
int x = 10;  
int *p = &x;
```

Dopo questa istruzione, il puntatore p contiene l'indirizzo di memoria della variabile x. Questo indirizzo può essere utilizzato per accedere al valore di x utilizzando l'operatore *.

Operatore *

L'operatore * viene utilizzato per accedere al valore a cui punta un puntatore. Ad esempio, la seguente istruzione stampa il valore della variabile x utilizzando il puntatore p:

```
int x = 10;  
int *p = &x;  
  
printf("%d\n", *p);
```

Questa istruzione stampa il valore 10.

Esempio

```
#include <stdio.h>

int main() {
    // Dichiarazione di una variabile e di un puntatore
    int x = 10;
    int \*puntatoreX;

    // Inizializzazione del puntatore con l'indirizzo di x
    puntatoreX = &x;

    // Accesso al valore di x tramite il puntatore
    printf("Valore di x: %d\n", *puntatoreX);

    // Modifica del valore di x tramite il puntatore
    *puntatoreX = 20;

    // Stampa del nuovo valore di x
    printf("Nuovo valore di x: %d\n", x);

    return 0;
}
```

I puntatori sono particolarmente utili nell'allocazione dinamica della memoria e nelle funzioni che devono modificare direttamente i valori delle variabili.

Puntatori con vettori

```
#include <stdio.h>
#define DIM 5

int main(){
    // dichiaro vettore
    int v[DIM];

    // dichiaro puntatore
    int* p;

    /* p = &v[0]: il valore del puntatore p è
    inizializzato al primo byte del vettore */
    for(p = &v[0]; p < v[DIM]; p++){
        // inizializzo il vettore tramite il puntatore
        *p = 0;
    }
}
```

Stringhe (Array di caratteri)

In C non esistono le "stringhe" in quanto le parole e le frasi vengono memorizzati come array di caratteri. Gli array vengono dichiarati mediante il tipo di dato `char`.

```
#include <stdio.h>
#define DIM 21

int main(){
    char parola[DIM];
    int len = strlen(parola);

    for(int i = 0; i < len; i++){
        parola[i] = '-';
    }
}
```

In questo esempio vediamo come inizializzare un char array; notiamo che nella dichiarazione inserisco nelle parentesi quadre la dimensione.

La dimensione DIM è 21, questo significa che possiamo accettare delle parole lunghe al massimo 20 caratteri in quanto nei char array l'ultima posizione è riservata per il carattere di fine stringa `\0`.

strlen()

La funzione `strlen` consente di trovare la lunghezza della stringa.

gets()

La funzione `gets()` ha la stessa funzione di `scanf()`, ma prende tutti i caratteri inseriti dall'utente, compresi gli spazi. Questo non avviene utilizzando `scanf()` in quanto lo spazio viene considerato come un "invio".

NB! `fgets()` migliore.

puts()

La funzione `puts()` ha la stessa funzione di `printf()`, ma solo per le stringhe.

strcpy()

```
char* strcpy(str2, str1);
```

Serve a copiare una stringa all'interno di un'altra. La funzione ritorna il puntatore della stringa in cui viene copiata la prima.

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

int main() {
    char str1[20] = "C programming";
    char str2[20];

    // copia str1 in str2
    strcpy(str2, str1);

    puts(str2);

    return 0;
}
```

strncpy()

Funziona come `strcpy()`, ma ammette un altro parametro di tipo intero che permette di specificare quanti char copiare.

```
char* strncpy(char*, char*, int);
```

strcat()

```
char* strcat(char*, char*);
```

Funzione che permette di concatenare due stringhe (quindi di "sommarle"). Il primo parametro corrisponde alla stringa di destinazione e il secondo alla stringa sorgente. Il valore che ritorna è la stringa di destinazione (posso concatenare la funzione).

```
char str1[4] = "abc";
char str2[7] = "def";
strcat(str2, str1);
```

```
risultato: "d e f a b c \0"
```

strncat()

```
char* strncat(char*, char*, int);
```

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

int main () {
    char src[50], dest[50];

    strcpy(src, "This is source");
    strcpy(dest, "This is destination");

    strncat(dest, src, 15);

    printf("Final destination string : %s", dest);

    return(0);
}
```

```
Final destination string : This is destinationThis is source
```

strcmp()

Funzione che viene utilizzata per comparare le due stringhe.

- **ritorna 0** se le stringhe sono identiche
- **ritorna un numero > 0** se il primo carattere non identico ha un valore ASCII maggiore del secondo
- **ritorna un numero < 0** se il primo carattere non identico ha un valore ASCII minore del secondo

Funzioni

Fino ad ora abbiamo scritto i nostri programmi utilizzando solo la funzione main: si può (e si deve), invece, suddividere il programma in blocchi di codice che risolvono specifiche task. Dividendo il programma in blocchi rende più facile la comprensione e la manutenzione del programma. Ci sono due tipi di funzioni, quelle standard, definite "dal linguaggio di programmazione" che già conosciamo, `scanf()`, `printf()` etc. e **quelle definite dall'utente**.

Definire e chiamare una funzione

```
#include <stdio.h>
int nomeFunzione("eventuali parametri")
{
```

```
    ... ..  
    ... ..  
}  
  
int main()  
{  
    ... ..  
    ... ..  
    // chiamata della funzione  
    nomeFunzione();  
    ... ..  
    ... ..  
}
```

Si può definire anche in modo diverso, creando prima un prototipo della funzione:

```
#include <stdio.h>  
int nomeFunzione("eventuali parametri");  
  
int main()  
{  
    ... ..  
    ... ..  
    nomeFunzione();  
    ... ..  
    ... ..  
}  
  
int nomeFunzione("eventuali parametri"){  
    // operazioni da svolgere  
}
```

Return nelle funzioni

Una volta che la funzione viene eseguita è previsto un **return** che permette di ritornare il risultato della funzione, appena eseguita, alla funzione main, per magari memorizzarla all'interno di una variabile. Il **tipo di dato** ritornato dalla funzione dipende dal **tipo della funzione**, se una funzione è di tipo **int** allora dovrà ritornare una variabile di tipo int e così via.

Passaggio di parametri

Per fare in modo che le funzioni siano effettivamente utili dobbiamo passare dei **parametri** in modo che la funzione possa lavorare su qualcosa:

```
#include <stdio.h>  
int somma(int a, int b);           // prototipo funzione  
  
int main()
```

```

{
    int n1, n2, risultato;

    printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);

    // chiamata funzione e memorizzazione del valore nella variabile
    risultato
    risultato = somma(n1, n2);
    printf("somma = %d", risultato);

    return 0;
}

// i parametri n1 e n2 prendono il nome di a e b
int somma(int a, int b)          // definizione funzione
{
    int risultato;
    risultato = a+b;

    return risultato;            // return
}

```

How to pass arguments to a function?

```

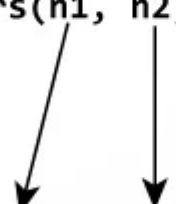
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... ..
    sum = addNumbers(n1, n2);
    ... ..
}

int addNumbers(int a, int b)
{
    ... ..
    ... ..
}

```



Esercizi da svolgere:

[Pdf esercizi Università della Calabria](#)