

C

Manuale per le ripetizioni di Informatica

Marco Lampis

14 marzo 2023

Indice

0	Informazioni generali	1
0.1	Come svolgo le lezioni	1
1	introduzione	3
1.1	Caratteristiche del linguaggio C	3
1.2	Case sensitive	4
2	main	5
3	variabili	7
4	if	9
5	switch	11
6	for	13
7	while	15
8	do-while	17
9	array	19
10	stringhe	21
11	struct	23
12	funzioni	25
13	Ricorsione	27
14	file	29
14.1	Aprire un file	29
14.2	Leggere un file	29
14.3	Scrivere su un file	30

14.4 Chiudere un file	31
---------------------------------	----

0 Informazioni generali

Ciao! Sono **Marco**, sono uno studente magistrale in *Software Engineering* al Politecnico di Torino e mi sono laureato in Ingegneria Informatica all'università di Pisa. Nella vita sono un programmatore, uno smanettone e amante di videogiochi! Amo quello che studio, e per questo motivo fornisco ripetizioni di informatica con particolare attenzione a:

- programmazione (Java, C++, C, Python, C#, Javascript, PHP)
- algoritmi e strutture dati
- basi di Dati
- più o meno tutto quello che riguarda l'informatica!

Sia per studenti delle scuole superiori che per l'università.

0.1 Come svolgo le lezioni

La mia metodologia è **innovativa** e **interattiva**: utilizzo una piattaforma apposita per lavorare contemporaneamente sullo stesso file con gli studenti, come se fossimo accanto. In questo modo, le lezioni sono più coinvolgenti e divertenti, e gli studenti possono imparare in modo facile ed efficiente. Inoltre, le lezioni includono sia esercitazioni che approfondimenti teorici, aiuto compiti e revisione.

Per aiutare gli studenti a prepararsi al meglio e a esercitarsi anche a casa, ho creato un sito web dedicato con risorse utili come teoria, esercizi e slide. Su questo sito, gli studenti possono trovare tutto il materiale di cui hanno bisogno, e il materiale è sempre disponibile gratuitamente per i miei studenti. Inoltre, uso un iPad per prendere appunti durante le lezioni e poi condividerli con gli studenti, in modo che possano rivedere tutto ciò che è stato trattato.

Se volete migliorare le vostre conoscenze in informatica e avere un supporto personalizzato e professionale, non esitate a contattarmi. Sarò felice di fornirvi maggiori informazioni e di fissare una lezione con voi.

1 introduzione

Il C è un linguaggio di programmazione procedurale a basso livello e general-purpose sviluppato da *Dennis Ritchie* negli anni '70 per sistemi operativi di tipo UNIX. È stato successivamente adottato da molti altri sistemi operativi e piattaforme hardware e, a partire dagli anni '80, è stato utilizzato come linguaggio di riferimento per lo sviluppo di altri linguaggi, come C++ e Java.

Uno dei principali punti di forza di C è la sua efficienza e la sua capacità di controllo diretto delle risorse hardware del sistema. Queste caratteristiche lo rendono un linguaggio adatto allo sviluppo di sistemi operativi, driver di dispositivi e altri tipi di codice che richiedono prestazioni elevate e un livello di controllo accurato.

1.1 Caratteristiche del linguaggio C

C è un linguaggio di programmazione procedurale, il che significa che i programmi sono suddivisi in funzioni che vengono eseguite in sequenza per completare un compito specifico. C è un linguaggio a basso livello, il che significa che fornisce un livello di controllo diretto sulla memoria e sulle risorse hardware del sistema. C è un linguaggio general-purpose, il che significa che può essere utilizzato per sviluppare una vasta gamma di applicazioni, dai sistemi operativi ai giochi. C è un linguaggio compilato, il che significa che il codice sorgente viene convertito in codice binario eseguibile dal compilatore prima di essere eseguito dal sistema. Esempio di codice in C Ecco un semplice esempio di codice in C che stampa una stringa di testo a schermo:

```
1 #include <stdio.h> // Importa la libreria standard di input/output
2
3 int main() { // Dichiarazione della funzione main, che rappresenta il punto di
    ingresso del programma
4     printf("Ciao, mondo!\n"); // Stampa la stringa "Ciao, mondo!" a
        schermo
5     return 0; // Restituisce 0 come codice di uscita del programma
6 }
```

Questo codice stamperà:

```
1 Ciao, mondo!
```

Come puoi vedere, il codice include una libreria standard di input/output (stdio.h) per poter utilizzare la funzione printf(), che viene usata per stampare la stringa “Ciao, mondo!” a schermo.

La sintassi della funzione main() è la seguente:

```
1 int main() {  
2     // Codice del programma  
3     return 0;  
4 }
```

Come puoi vedere, la funzione main() ha il tipo di dati int (che significa che restituisce un valore intero) e non accetta argomenti. Il codice del programma viene inserito all'interno delle parentesi graffe {} e, alla fine del codice, viene restituito il valore 0 per indicare che il programma è terminato correttamente.

1.2 Case sensitive

Il linguaggio C è case sensitive, il che significa che le lettere minuscole e maiuscole sono considerate diverse. Per esempio, le variabili `a` e `A` sono considerate due variabili diverse.

ASuggerimento: è possibile utilizzare le lettere maiuscole e minuscole per identificare le variabili, ma è una buona pratica utilizzare solo lettere minuscole per identificare le variabili.

Attenzione: possono esistere variabili con stesso nome ma che differiscono per le lettere maiuscole e minuscole. Per esempio, `ciao` e `CIAO` sono due file diversi.

2 main

In C, la funzione main è la funzione principale di un programma e viene eseguita quando il programma viene avviato.

Esempio:

```
1 int main() {  
2     // codice del programma  
3     return 0;  
4 }
```

In questo esempio, abbiamo definito la funzione main che non accetta alcun argomento e restituisce un valore di tipo int. All'interno del corpo della funzione, possiamo inserire il codice del nostro programma. La funzione main termina con l'istruzione return 0, che indica che il programma è stato eseguito con successo.

La funzione main può anche accettare argomenti dalla riga di comando:

```
1 int main(int argc, char *argv[]) {  
2     // codice del programma  
3     return 0;  
4 }
```

In questo esempio, la funzione main accetta due argomenti: argc e argv. Il primo argomento, argc, è un intero che indica il numero di argomenti passati alla riga di comando. Il secondo argomento, argv, è un array di stringhe che contiene i valori degli argomenti passati alla riga di comando.

Possiamo quindi utilizzare gli argomenti argc e argv all'interno del nostro codice per manipolare e utilizzare gli argomenti passati alla riga di comando. Ad esempio, possiamo utilizzare il valore di argc per determinare se sono stati passati argomenti alla riga di comando:

```
1 if (argc > 1) {  
2     // codice da eseguire se sono stati passati argomenti alla riga di  
    comando  
3 } else {  
4     // codice da eseguire se non sono stati passati argomenti alla riga  
    di comando  
5 }
```


In questo caso, il codice all'interno del blocco dell'istruzione `if` verrà eseguito se sono stati passati argomenti alla riga di comando, altrimenti verrà eseguito il codice all'interno del blocco dell'istruzione `else`.

La funzione `main` è una parte fondamentale di ogni programma in C poiché viene eseguita quando il programma viene avviato. È importante assicurarsi di definire la funzione `main` nel nostro codice e di utilizzarla per inserire il codice principale del nostro programma.

3 variabili

In C, una variabile è un contenitore utilizzato per memorizzare un valore. Ogni variabile ha un nome univoco e un tipo di dati che determina il tipo di valori che può contenere.

Esempio:

```
1 int x = 5;
```

In questo esempio, abbiamo dichiarato una variabile chiamata `x` di tipo `int`, che può contenere valori interi. Abbiamo quindi assegnato alla variabile `x` il valore 5.

È possibile utilizzare le variabili all'interno del nostro codice per manipolare e utilizzare i valori che contengono. Ad esempio, possiamo utilizzare le variabili all'interno di espressioni matematiche:

```
1 int y = 7;  
2 int z = x + y; // z conterrà 12
```

In questo caso, abbiamo dichiarato una nuova variabile chiamata `y` e l'abbiamo inizializzata con il valore 7. Abbiamo quindi utilizzato le variabili `x` e `y` all'interno di un'espressione matematica per calcolare il valore della variabile `z`. La variabile `z` conterrà quindi il valore 12, poiché $5 + 7 = 12$.

È anche possibile utilizzare le variabili all'interno delle istruzioni di controllo come `if` e `switch` per eseguire codice in base al valore di una variabile:

```
1 if (x > 3) {  
2     printf("x è maggiore di 3\n");  
3 } else {  
4     printf("x è minore o uguale a 3\n");  
5 }
```

In questo caso, stiamo utilizzando il valore della variabile `x` all'interno dell'istruzione `if` per determinare se il codice all'interno del blocco dell'istruzione `if` verrà eseguito o meno. Poiché il valore della variabile `x` è maggiore di 3, il codice all'interno del blocco dell'istruzione `if` verrà eseguito e verrà stampato il messaggio "x è maggiore di 3".

Le variabili sono una parte fondamentale della programmazione in C, poiché permettono di memorizzare e manipolare i valori all'interno del nostro codice. È importante assicurarsi di utilizzare nomi di variabili significativi e di assegnare loro valori appropriati per garantire che il nostro codice funzioni correttamente.

4 if

In C, l'istruzione `if` è utilizzata per eseguire un blocco di codice solo se una determinata condizione è vera.

Esempio:

```
1 if (condizione) {  
2     // blocco di codice da eseguire se la condizione è vera  
3 }
```

In questo esempio, il blocco di codice all'interno delle parentesi graffe verrà eseguito solo se la condizione specificata tra parentesi è vera. Ad esempio:

```
1 int x = 5;  
2 if (x > 3) {  
3     printf("x è maggiore di 3\n");  
4 }
```

In questo caso, il codice all'interno del blocco `if` verrà eseguito poiché la variabile `x` è effettivamente maggiore di 3.

È anche possibile utilizzare l'istruzione `else` per eseguire un blocco di codice diverso se la condizione non è vera:

```
1 int x = 5;  
2 if (x > 3) {  
3     printf("x è maggiore di 3\n");  
4 } else {  
5     printf("x è minore o uguale a 3\n");  
6 }
```

In questo caso, il codice all'interno del blocco `else` non verrà eseguito poiché la variabile `x` è effettivamente maggiore di 3. Tuttavia, se la variabile `x` fosse stata minore o uguale a 3, il codice all'interno del blocco `else` sarebbe stato eseguito.

È inoltre possibile utilizzare più istruzioni `if` e `else` per creare una catena di condizioni:

```
1 int x = 5;  
2 if (x > 5) {  
3     printf("x è maggiore di 5\n");
```

```
4 } else if (x > 3) {  
5     printf("x è maggiore di 3 ma minore o uguale a 5\n");  
6 } else {  
7     printf("x è minore o uguale a 3\n");  
8 }
```

In questo caso, il codice all'interno del primo blocco if non verrà eseguito poiché la variabile x non è maggiore di 5. Tuttavia, il codice all'interno del secondo blocco else if verrà eseguito poiché la variabile x è effettivamente maggiore di 3 ma minore o uguale a 5.

5 switch

In C, l'istruzione switch è utilizzata per eseguire un blocco di codice diverso in base al valore di una variabile.

Esempio:

```
1  int x = 2;
2  switch (x) {
3      case 1:
4          // blocco di codice da eseguire se x è uguale a 1
5          break;
6      case 2:
7          // blocco di codice da eseguire se x è uguale a 2
8          break;
9      default:
10         // blocco di codice da eseguire se x non è uguale a nessuno dei
            valori specificati nei casi precedenti
11         break;
12 }
```

In questo esempio, il blocco di codice all'interno del caso case 2 verrà eseguito poiché la variabile x è effettivamente uguale a 2. È importante notare che ogni caso deve terminare con l'istruzione break per interrompere l'esecuzione del codice e passare al codice successivo dopo l'istruzione switch. Inoltre, il blocco default viene eseguito se nessun altro caso è vero.

È anche possibile utilizzare l'istruzione switch per confrontare stringhe:

```
1  char nome[50] = "Mario Rossi";
2  switch (nome) {
3      case "Mario Rossi":
4          // blocco di codice da eseguire se nome è uguale a "Mario Rossi"
5          break;
6      case "Luca Bianchi":
7          // blocco di codice da eseguire se nome è uguale a "Luca Bianchi"
8          break;
9      default:
10         // blocco di codice da eseguire se nome non è uguale a nessuno dei
            valori specificati nei casi precedenti
11         break;
12 }
```

In questo caso, il blocco di codice all'interno del caso case "Mario Rossi" verrà eseguito poiché la stringa contenuta nella variabile nome è effettivamente uguale a "Mario Rossi".

L'istruzione switch può essere uno strumento molto utile in C per eseguire codice in base al valore di una variabile. Tuttavia, è importante assicurarsi di utilizzare l'istruzione break in ogni caso per evitare che il codice venga eseguito in modo imprevisto.

6 for

La costruzione for in C è un altro tipo di ciclo che viene utilizzato per eseguire un blocco di codice ripetutamente. A differenza del ciclo while, che verifica una determinata condizione prima di ogni iterazione del ciclo, il ciclo for include una serie di espressioni all'interno della sua sintassi che controllano il flusso del ciclo.

La sintassi di un ciclo for in C è la seguente:

```
1 for (espressione_iniziale; espressione_condizione;  
    espressione_iterazione) {  
2     // Codice da eseguire ripetutamente  
3 }
```

Le espressioni all'interno delle parentesi del ciclo for controllano il flusso del ciclo come segue:

- L'espressione iniziale viene eseguita una volta all'inizio del ciclo, prima della prima iterazione.
- L'espressione condizione viene verificata prima di ogni iterazione del ciclo. Se la condizione è verificata, il ciclo viene eseguito; altrimenti, il ciclo viene interrotto e il programma prosegue con l'esecuzione del codice successivo.
- L'espressione iterazione viene eseguita alla fine di ogni iterazione del ciclo.

Ecco un esempio di un ciclo for in C:

```
1 for (int i = 0; i < 5; i++) {  
2     // Codice da eseguire ripetutamente  
3     printf("%d\n", i);  
4 }
```

Questo codice stamperà:

```
1 0  
2 1  
3 2  
4 3  
5 4
```

Come puoi vedere, il ciclo for include tre espressioni all'interno delle sue parentesi:

1. L'espressione iniziale (`int i = 0`) viene eseguita una volta all'inizio del ciclo, prima della prima iterazione. In questo caso, viene dichiarata una variabile intera `i` e viene assegnato il valore 0.

2. L'espressione condizione ($i < 5$) viene verificata prima di ogni iterazione del ciclo. Se la condizione è verificata, il ciclo viene eseguito; altrimenti, il ciclo viene interrotto e il programma prosegue con l'esecuzione del codice successivo. In questo caso, la condizione è verificata finché i è minore di 5.
3. L'espressione iterazione ($i++$) viene eseguita alla fine di ogni iterazione del ciclo. In questo caso, viene incrementato il valore di i di 1 ad ogni iterazione del ciclo.

7 while

Il costrutto **while** è una struttura di controllo che viene utilizzata nei linguaggi di programmazione per eseguire un blocco di codice ripetutamente fino a quando una condizione specificata è vera.

La sintassi del costrutto **while** in C è la seguente:

```
1 while (condizione) {  
2     codice da eseguire  
3 }
```

La sezione di condizione viene controllata ad ogni iterazione del ciclo. Se la condizione è vera, il codice nel blocco viene eseguito. Se la condizione è falsa, il ciclo viene interrotto e il controllo passa alla linea di codice successiva.

Ecco un esempio di utilizzo del costrutto while in C++:

```
1 int i = 0;  
2 while (i < 10) {  
3     cout << i << endl;  
4     i++;  
5 }
```

In questo caso, il ciclo viene eseguito 10 volte, stampando i numeri da 0 a 9.

8 do-while

La costruzione do-while in C è un tipo di ciclo che viene utilizzato per eseguire un blocco di codice ripetutamente finché una determinata condizione è verificata. A differenza della costruzione while, che esegue il blocco di codice solo se la condizione è verificata, il ciclo do-while esegue sempre almeno una volta il blocco di codice prima di verificare la condizione.

La sintassi di un ciclo do-while in C è la seguente:

```
1 do {  
2     // Codice da eseguire ripetutamente  
3 } while (condizione);
```

Ecco un esempio di codice do while:

```
1 int i = 0;  
2 do {  
3     // Codice da eseguire ripetutamente  
4     printf("%d\n", i);  
5     i++;  
6 } while (i < 5);
```

Questo codice stamperà:

```
1 0  
2 1  
3 2  
4 3  
5 4
```

Come puoi vedere, il ciclo do-while esegue il blocco di codice all'interno delle parentesi graffe {} almeno una volta, quindi verifica la condizione ($i < 5$) alla fine di ogni iterazione del ciclo. Se la condizione è verificata, il ciclo viene eseguito di nuovo; altrimenti, il ciclo viene interrotto e il programma prosegue con l'esecuzione del codice successivo.

9 array

In C, un array è una struttura di dati che ti permette di memorizzare una serie di elementi del medesimo tipo in un unico oggetto. Ad esempio, potresti creare un array di interi per memorizzare una serie di numeri interi, oppure un array di caratteri per memorizzare una stringa di testo.

Per dichiarare un array in C, devi specificare il tipo di dati degli elementi che desideri memorizzare, seguito dal nome dell'array e dalle dimensioni dell'array tra parentesi quadre:

```
1 int interi[10]; // Dichiarare un array di interi di dimensione 10
2 char stringa[20]; // Dichiarare un array di caratteri di dimensione 20
```

Una volta che hai dichiarato un array, puoi accedere ai singoli elementi dell'array usando l'operatore di indice. L'indice di un elemento dell'array inizia sempre da 0, quindi per accedere al primo elemento dell'array interi dichiarato sopra, ad esempio, useresti `interi[0]`.

Ecco un esempio di come potresti utilizzare un array di interi per memorizzare e stampare una serie di numeri:

```
1 int interi[10]; // Dichiarare un array di interi di dimensione 10
2
3 // Assegna alcuni valori all'array
4 interi[0] = 10;
5 interi[1] = 20;
6 interi[2] = 30;
7
8 // Stampa gli elementi dell'array
9 for (int i = 0; i < 10; i++) {
10     printf("Elemento %d: %d\n", i, interi[i]);
11 }
```

Questo codice stamperà:

```
1 Elemento 0: 10
2 Elemento 1: 20
3 Elemento 2: 30
4 Elemento 3: 0
5 Elemento 4: 0
6 Elemento 5: 0
7 Elemento 6: 0
8 Elemento 7: 0
```

9	Elemento	8:	0
10	Elemento	9:	0

10 stringhe

In C, una stringa è una sequenza di caratteri che può essere manipolata come un'unica entità.

Esempio:

```
1 char nome[50] = "Mario Rossi";
```

In questo esempio, abbiamo dichiarato una variabile di tipo `char` chiamata `nome` che può contenere una stringa di al massimo 50 caratteri. Abbiamo quindi assegnato alla variabile `nome` la stringa "Mario Rossi".

Possiamo utilizzare le funzioni della libreria `string.h` per manipolare le stringhe in C. Ad esempio, la funzione `strlen` può essere utilizzata per determinare la lunghezza di una stringa:

```
1 int lunghezza = strlen(nome); // lunghezza sarà uguale a 11
```

In questo caso, abbiamo utilizzato la funzione `strlen` per determinare la lunghezza della stringa contenuta nella variabile `nome`. La variabile `lunghezza` conterrà quindi il valore 11, poiché la stringa "Mario Rossi" è composta da 11 caratteri.

È anche possibile utilizzare la funzione `strcpy` per copiare una stringa in un'altra variabile:

```
1 char nome_copia[50];  
2 strcpy(nome_copia, nome); // nome_copia conterrà "Mario Rossi"
```

In questo caso, abbiamo utilizzato la funzione `strcpy` per copiare la stringa contenuta nella variabile `nome` nella variabile `nome_copia`. La variabile `nome_copia` conterrà quindi la stringa "Mario Rossi".

È inoltre possibile utilizzare la funzione `strcat` per concatenare due stringhe:

```
1 char cognome[50] = "Bianchi";  
2 strcat(nome, cognome); // nome conterrà "Mario Rossi Bianchi"
```

In questo caso, abbiamo utilizzato la funzione `strcat` per concatenare la stringa contenuta nella variabile `cognome` alla fine della stringa contenuta nella variabile `nome`. La variabile `nome` conterrà quindi la stringa "Mario Rossi Bianchi".

Le stringhe sono un tipo di dati molto utile in C poiché permettono di manipolare facilmente sequenze di caratteri. Tuttavia, è importante ricordare di allocare sempre abbastanza spazio nella memoria per

le stringhe utilizzate nel nostro programma per evitare errori o crash.

11 struct

In C, una struct è una struttura dati utilizzata per organizzare dati di tipi diversi in un unico blocco.

Esempio:

```
1 struct persona {  
2     char nome[50];  
3     int età;  
4     float altezza;  
5 };
```

In questo esempio, abbiamo definito una struct chiamata persona che contiene tre campi: nome, età e altezza. Ogni campo può contenere un valore di un tipo di dati diverso.

Per utilizzare una struct in un programma, è necessario dichiararne una variabile:

```
1 struct persona p1;
```

In questo caso, abbiamo dichiarato una variabile chiamata p1 che è una struct persona. Possiamo quindi accedere ai campi della struct utilizzando l'operatore di accesso ai membri (.):

```
1 p1.nome = "Mario Rossi";  
2 p1.età = 30;  
3 p1.altezza = 1.75;
```

In questo caso, abbiamo assegnato valori ai campi nome, età e altezza della struct p1. Possiamo quindi utilizzare questi valori nella nostra logica di programma:

```
1 printf("%s ha %d anni e %.2f di altezza\n", p1.nome, p1.età, p1.altezza  
    );
```

In questo caso, stiamo utilizzando i valori delle variabili nome, età e altezza all'interno del nostro comando printf per stampare i valori della struct p1.

È inoltre possibile utilizzare una struct come tipo di dati per una funzione:

```
1 struct persona crea_persona(char *nome, int età, float altezza) {  
2     struct persona p;  
3     p.nome = nome;  
4     p.età = età;  
5     p.altezza = altezza;
```

```
6     return p;  
7 }
```

In questo caso, abbiamo creato una funzione `crea_persona` che accetta tre argomenti: `nome`, `età` e `altezza`. La funzione crea una nuova struct `persona` e assegna ai suoi campi i valori degli argomenti. La funzione restituisce poi la struct `persona`. Possiamo quindi utilizzare i valori delle variabili `nome`, `età` e `altezza` all'interno del nostro comando `printf` per stampare i valori della struct `p1`.

È inoltre possibile utilizzare una struct come tipo di dati per una funzione:

```
1 struct persona crea_persona(char *nome, int età, float altezza) {  
2     struct persona p;  
3     p.nome = nome;  
4     p.età = età;  
5     p.altezza = altezza;  
6     return p;  
7 }
```

In questo caso, abbiamo creato una funzione `crea_persona` che accetta tre argomenti: `nome`, `età` e `altezza`. La funzione crea una nuova struct `persona` e assegna ai suoi campi i valori degli argomenti. La funzione restituisce poi la struct `persona`. Possiamo quindi utilizzare questa funzione per creare nuove struct `persona`:

```
1 struct persona p1 = crea_persona("Mario Rossi", 30, 1.75);  
2 struct persona p2 = crea_persona("Luca Bianchi", 28, 1.80);
```

In questo caso, abbiamo creato due nuove struct `persona` utilizzando la funzione `crea_persona` e assegnando i valori restituiti alle variabili `p1` e `p2`. Possiamo quindi utilizzare queste struct `persona` all'interno del nostro programma come desiderato.

12 funzioni

In C, una funzione è una porzione di codice che viene eseguita in risposta a una determinata richiesta. Le funzioni sono utilizzate per suddividere il codice in blocchi riutilizzabili che possono essere richiamati da altre parti del programma.

Per dichiarare una funzione in C, devi specificare il tipo di dati del valore restituito dalla funzione, seguito dal nome della funzione e dai suoi argomenti tra parentesi:

```
1 int somma(int x, int y) {  
2     // Codice della funzione  
3     return x + y;  
4 }
```

In questo esempio, la funzione `somma` riceve due argomenti di tipo `int` (`x` e `y`) e restituisce un valore di tipo `int` che rappresenta la somma di `x` e `y`.

Per richiamare una funzione in C, devi semplicemente usare il suo nome seguito dai suoi argomenti tra parentesi:

```
1 int main() {  
2     int a = 5, b = 7;  
3     int risultato = somma(a, b); // Richiama la funzione somma  
4     printf("La somma di %d e %d è %d\n", a, b, risultato);  
5     return 0;  
6 }
```

Questo codice stamperà:

```
1 La somma di 5 e 7 è 12
```

Come puoi vedere, la funzione `somma` viene dichiarata all'inizio del codice, specificando il tipo di dati del valore restituito (`int`) e i suoi argomenti (due interi chiamati `x` e `y`). Il codice all'interno delle parentesi graffe `{}` rappresenta il corpo della funzione, che viene eseguito quando la funzione viene richiamata.

Nel `main()`, la funzione `somma` viene richiamata passando come argomenti i valori di `a` e `b`. La funzione esegue il suo codice, che in questo caso consiste nella somma di `x` e `y` e nella restituzione del risultato, che viene assegnato alla variabile `risultato`. Quindi, il valore di `risultato` viene stampato a schermo.

13 Ricorsione

Le funzioni ricorsive in C sono funzioni che si richiamano da sole, ovvero una funzione richiama se stessa all'interno del proprio corpo.

In pratica, il funzionamento è il seguente: la funzione esegue una determinata operazione su un input, dopodiché si richiama passando come input una versione modificata dell'input originale. Questa operazione si ripete finché non si raggiunge una condizione di terminazione, ovvero un caso base che impedisce alla funzione di richiamarsi ulteriormente.

Ogni funzione ricorsiva è composta da due componenti principali:

- **Caso base:** condizione che impedisce alla funzione di richiamarsi ulteriormente. Se non viene specificata, la funzione si richiamerà all'infinito.
- **Caso ricorsivo:** chiamata ricorsiva della funzione, che passa come input una versione modificata dell'input originale.

Per esempio, consideriamo la funzione fattoriale:

```
1 int fattoriale(int n) {  
2     if (n == 0) { // caso base  
3         return 1;  
4     } else {  
5         return n * fattoriale(n-1); // chiamata ricorsiva  
6     }  
7 }
```

In questo caso, se n è uguale a 0, la funzione ritorna 1, altrimenti ritorna n moltiplicato per il fattoriale di $n-1$, ovvero la stessa funzione chiamata con un input diverso. Questo processo si ripete finché non si raggiunge il caso base, ovvero $n = 0$.

Un altro esempio potrebbe essere la funzione Fibonacci:

```
1 int fibonacci(int n) {  
2     if (n == 0 || n == 1) { // casi base  
3         return n;  
4     }  
5  
6     else {  
7         return fibonacci(n-1) + fibonacci(n-2); // chiamate ricorsive
```

```
8     }  
9 }
```

In questo caso, se n è uguale a 0 o 1, la funzione ritorna direttamente n , altrimenti ritorna la somma dei due numeri di Fibonacci precedenti, ovvero la stessa funzione chiamata con $n-1$ e $n-2$. Anche qui, il processo si ripete finché non si raggiunge un caso base.

Tuttavia, è importante notare che le funzioni ricorsive possono portare a problemi di memoria se non gestite correttamente. Ogni volta che una funzione si richiama, viene creata una nuova istanza della funzione, che viene mantenuta in memoria fino alla fine della chiamata ricorsiva. Se la ricorsione è troppo profonda o se la funzione richiede molta memoria, si può arrivare a un esaurimento della memoria disponibile (stack overflow).

Quindi, è importante assicurarsi che la ricorsione termini in modo appropriato e che non ci siano chiamate ricorsive infinite o troppo profonde. Inoltre, in alcuni casi, le funzioni iterative possono essere più efficienti e meno rischiose delle funzioni ricorsive.

14 file

In **C**, come in molti altri linguaggi di programmazione, la gestione dei file viene effettuata tramite delle funzioni specifiche che permettono di aprire, leggere, scrivere e chiudere un file.

Suggerimento: puoi pensare ai file come dei “pacchi”. Prima di utilizzare un pacco, è necessario aprirlo, poi leggere il suo contenuto, scrivere qualcosa all’interno e infine chiuderlo (va sempre chiuso prima di essere spedito!).

14.1 Aprire un file

Per aprire un file, si può utilizzare la funzione `fopen()`, che prende come argomenti il nome del file e la modalità di apertura (lettura, scrittura, lettura e scrittura, etc.). Ad esempio, per aprire un file chiamato “test.txt” in modalità di lettura, si può utilizzare il seguente codice:

```
1 FILE *file_ptr;  
2 file_ptr = fopen("test.txt", "r");
```

I file possono essere aperti in varie modalità:

- **lettura:** `r`, consente la lettura del file.
- **scrittura:** `w`, consente la scrittura sul file. Se il file esiste già, il suo contenuto viene sovrascritto.
- **append:** `a`, consente la scrittura sul file. Se il file esiste già, il nuovo contenuto viene aggiunto in coda al file.

Analogia: Hai aperto il pacco per poi utilizzarlo in seguito.

14.2 Leggere un file

Il **C** mette a disposizione molte funzioni per leggere i dati da un file, di seguito ne sono illustrate alcune.

Analogia: stai guardando il contenuto del pacco.

14.2.1 fread()

Per leggere il contenuto di un file aperto, si può utilizzare la funzione `fread()`, che prende come argomenti il puntatore al file, la dimensione di ogni elemento da leggere, il numero di elementi da leggere e un puntatore all'area di memoria in cui verranno scritti i dati letti dal file. Ad esempio, per leggere un array di interi dal file "test.txt" aperto in precedenza, si può utilizzare il seguente codice:

```
1 int array[100];
2 fread(array, sizeof(int), 100, file_ptr);
```

14.2.2 getline()

Per leggere il contenuto di un file aperto riga per riga è possibile utilizzare la funzione `getline()`, che prende come argomenti il puntatore all'area di memoria in cui verrà scritta la riga letta, il numero massimo di caratteri da leggere e il puntatore al file da cui leggere. Ad esempio, per leggere il contenuto del file "test.txt" aperto in precedenza riga per riga, si può utilizzare il seguente codice:

```
1 char * line = NULL;
2 size_t len = 0;
3 ssize_t read;
4
5 // fino a quando il file non finisce
6 while ((read = getline(&line, &len, fp)) != -1) {
7     printf("%s", line); // stampa la riga letta
8 }
```

14.3 Scrivere su un file

Il **C** mette a disposizione molte funzioni per leggere i dati da un file, di seguito ne sono illustrate alcune.

Analogia: Stai inserendo delle informazioni all'interno del pacco.

14.3.1 fwrite()

Per scrivere su un file aperto, si può utilizzare la funzione `fwrite()`, che prende come argomenti il puntatore all'area di memoria contenente i dati da scrivere, la dimensione di ogni elemento da scrivere, il numero di elementi da scrivere e un puntatore al file su cui scrivere. Ad esempio, per scrivere l'array di interi precedentemente letto sul file "test.txt", si può utilizzare il seguente codice:

```
1 fwrite(array, sizeof(int), 100, file_ptr);
```

14.3.2 fprintf()

Per scrivere su un file aperto, si può utilizzare la funzione `fprintf()`, che prende come argomenti il puntatore al file su cui scrivere e una stringa contenente il testo da scrivere. Ad esempio, per scrivere una stringa sul file "test.txt" aperto in precedenza, si può utilizzare il seguente codice:

```
1 fprintf(file_ptr, "Hello World!");
```

14.4 Chiudere un file

Per chiudere un file aperto, si può utilizzare la funzione `fclose()`, che prende come argomento il puntatore al file da chiudere. Ad esempio, per chiudere il file "test.txt" aperto in precedenza, si può utilizzare il seguente codice:

```
1 fclose(file_ptr);
```

Analogia: Stai chiudendo il pacco per poterlo spedire.

Importante: Ricorda sempre di chiudere un file dopo aver finito di utilizzarlo. Per poterlo riutilizzare sarà poi necessario riapirlo.

