mlampis.dev

C

Manuale per le ripetizioni di Informatica

Marco Lampis

7 dicembre 2022

Indice

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

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 platforme 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.

2 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:

```c
#include <stdio.h> // Importa la libreria standard di input/output

int main() { // Dichiara la funzione main, che rappresenta il punto di ingresso del programma
    printf("Ciao, mondo!\n"); // Stampa la stringa "Ciao, mondo!" a schermo
    return 0; // Restituisce 0 come codice di uscita del programma
}
```

Questo codice stamperà:

```
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:

```c
int main() {
    // Codice del programma
    return 0;
}
```

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.

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 é maggiore di 3\n");
3 } else {
4   printf("x é 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:

```
if (condizione) {
  // blocco di codice da eseguire se la condizione é vera
}
```

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:

```
int x = 5;
if (x > 3) {
  printf("x é maggiore di 3\n");
}
```

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:

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

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:

```
int x = 5;
if (x > 5) {
  printf("x é maggiore di 5\n");
```

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

```c
int x = 2;
switch (x) {
  case 1:
    // blocco di codice da eseguire se x é uguale a 1
    break;
  case 2:
    // blocco di codice da eseguire se x é uguale a 2
    break;
  default:
    // blocco di codice da eseguire se x non é uguale a nessuno dei
       valori specificati nei casi precedenti
    break;
}
```

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:

```c
char nome[50] = "Mario Rossi";
switch (nome) {
  case "Mario Rossi":
    // blocco di codice da eseguire se nome é uguale a "Mario Rossi"
    break;
  case "Luca Bianchi":
    // blocco di codice da eseguire se nome é uguale a "Luca Bianchi"
    break;
  default:
    // blocco di codice da eseguire se nome non é uguale a nessuno dei
       valori specificati nei casi precedenti
    break;
}
```

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:

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

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:

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

Questo codice stamperà:

```
0
1
2
3
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 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:

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

Ecco un esempio di codice do while:

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

Questo codice stamperà:

```
0
1
2
3
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.

8 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:

```
int interi[10]; // Dichiara un array di interi di dimensione 10
char stringa[20]; // Dichiara 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:

```
int interi[10]; // Dichiara un array di interi di dimensione 10

// Assegna alcuni valori all'array
interi[0] = 10;
interi[1] = 20;
interi[2] = 30;

// Stampa gli elementi dell'array
for (int i = 0; i < 10; i++) {
    printf("Elemento %d: %d\n", i, interi[i]);
}
```

Questo codice stamperà:

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

```
Elemento 8: 0
Elemento 9: 0
```

9 struct

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

Esempio:

```
struct persona {
  char nome[50];
  int età;
  float altezza;
};
```

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:

```
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 (.):

```
p1.nome = "Mario Rossi";
p1.età = 30;
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:

```
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:

```
struct persona crea_persona(char *nome, int età, float altezza) {
  struct persona p;
  p.nome = nome;
  p.età = età;
  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.