mlampis.dev

C++

Manuale per le ripetizioni di Informatica

Marco Lampis

9 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 di alto livello, orientato agli oggetti, che viene utilizzato per lo sviluppo di applicazioni software. È un linguaggio di programmazione multiparadigma, ovvero consente di utilizzare più stili di programmazione, come ad esempio la programmazione ad oggetti, la programmazione imperativa e la programmazione funzionale.

Il C++ è un linguaggio di programmazione ad alto livello che offre una vasta gamma di funzionalità e un elevato livello di controllo sulla programmazione dei computer. È stato creato come evoluzione del linguaggio C, aggiungendo la gestione degli oggetti e l'ereditarietà. Il C++ è un linguaggio compilato, il che significa che il codice sorgente viene convertito in un linguaggio macchina direttamente eseguibile dal computer. Questo lo rende veloce ed efficiente, ma richiede anche un maggiore impegno da parte del programmatore per gestire il codice sorgente e gli errori di compilazione.

Il C++ è utilizzato in molti ambiti, dallo sviluppo di sistemi operativi e driver di dispositivi ai giochi e alle applicazioni desktop. È anche uno dei linguaggi più utilizzati per lo sviluppo di applicazioni per il settore scientifico e quello finanziario. Se sei interessato a imparare il C++, ti consiglio di iniziare con i concetti di base del linguaggio, come le variabili, i tipi di dati, le operazioni e le strutture di controllo del flusso del programma.

2 Variabili

In C++, una variabile è un contenitore per memorizzare un valore. Ogni variabile ha un nome e un tipo, che determina il tipo di valore che può contenere. Ad esempio, una variabile di tipo int può contenere un numero intero, mentre una variabile di tipo char può contenere un singolo carattere.

Per dichiarare una variabile in C++, si utilizza la sintassi seguente:

```
tipo nomeVariabile;
```

Ad esempio, per dichiarare una variabile di nome numero di tipo int, potremmo scrivere:

```
int numero;
```

Per assegnare un valore a una variabile, si utilizza l'operatore di assegnamento =, ad esempio:

```
numero = 10;
```

Ecco alcuni tipi di variabili comunemente usati in C++:

int: contiene un numero intero, ad esempio 1, -10, 100. double: contiene un numero con la virgola, ad esempio 3.14, -0.5, 0.0. char: contiene un singolo carattere, ad esempio 'A', 'B', '1'. Ecco un esempio che mostra come dichiarare e utilizzare alcune variabili di diversi tipi:

```
int numeroIntero = 10;
double numeroVirgola = 3.14;
char carattere = 'A';

std::cout << numeroIntero << std::endl; // stamperà 10
std::cout << numeroVirgola << std::endl; // stamperà 3.14
std::cout << carattere << std::endl; // stamperà A
```

3 if

Il costrutto `if` consente di verificare se una condizione è verificata ed eseguire un blocco di codice in base al risultato. L'esito è sempre di tipo **booleano**, ovvero può essere solo `true` o `false`.

3.1 Sintassi

La sintassi di `if` è la seguente:

```
if (condizione1){
    // codice eseguito se la condizione é vera
}

else if(condizione2){
    // codice eseguito se la condizione1 non é vera
    // ma é vera condizione2
}

else{
    // codice eseguito se nessuna delle condizioni é vera
}
```

Attenzione: Il blocco di codice che segue `else if` ed `else` è opzionale. Se non viene specificato nessun blocco di codice, il programma non esegue alcuna operazione. Inoltre è importante scrivere le keyword con caratteri minuscoli, in caso contrario si avrà errore

Suggerimento: Non utilizzare `else if` ed `else` se non è necessario, in quanto possono rendere il codice meno leggibile.

3.1.1 Esempio

```
int a = 5;
int b = 6;

```

```
4  if (b > a){
5      cout<<"b é maggiore di a";
6  }
7
8  else if (a == b){
9      cout<<"a e b sono uguali";
10 }
11
12 else{
13     cout<<"a é maggiore di b";
14 }
```

Puoi trovare a questo link alcuni esercizi su **if**: https://esercizi.mlampis.dev/book.php?langs=cpp

3.1.2 Esercitazione

> Verificare se un numero inserito in input è pari.

```
1  #include <iostream>
2  using namespace std;
3
4
5  int main(){
6
7    // dichiariamo una variabile
8    int a;
9
10   // invitiamo l'utente a inserire un valore
11   cout<<"Inserisci un numero: ";
12
13   // lo facciamo inserire
14   cin >> a;
15
16   // verifichiamo se un numero é pari
17   if (a%2 == 0){
18     cout<<"Numero pari";
19   }
20
21   else {
22     cout<<"Numero dispari";
23   }
24
25     return 0;
26 }
```

4 Switch

Il costrutto switch è una struttura di controllo che viene utilizzata nei linguaggi di programmazione per eseguire uno specifico blocco di codice in base al valore di una variabile o di un'espressione.

La sintassi del costrutto switch in C++ è la seguente:

```
switch (espressione) {
  case valore1:
    codice da eseguire se espressione == valore1
    break;
  case valore2:
    codice da eseguire se espressione == valore2
    break;
  ...
  default:
    codice da eseguire se nessun altro caso viene soddisfatto
}
```

La sezione **`case`** specifica un valore per il quale il codice nel blocco viene eseguito, mentre la sezione **`default`** viene eseguita se nessun altro caso viene soddisfatto.

Ecco un esempio di utilizzo del costrutto **`switch`** in C++:

```
int x = 2;
switch (x) {
  case 1:
    cout << "x é 1" << endl;
    break;
  case 2:
    cout << "x é 2" << endl;
    break;
  default:
    cout << "x non é 1 né 2" << endl;
}
```

In questo caso, il codice nel secondo caso viene eseguito, stampando "x è 2".

5 For

Il ciclo `for` viene utilizzato quando si conosce il numero di iterazioni da compiere. Il ciclo `for` è composto da tre parti:

- **inizializzazione**: viene inizializzato un contatore
- **verifica**: viene verificata la condizione di uscita dal ciclo
- **passo**: viene incrementato il contatore

5.1 Sintassi

La sintassi di `for` è la seguente:

```
for (inizializzazione; verifica; passo){
 // codice eseguito
}
```

Nota: spesso si utilizza il contatore `i` per indicare il numero di iterazioni.

5.1.1 Esempio

Esempio di esercizio che stampa i numeri da 1 a 5 (escluso):

```
for (int i = 0; i < 5; i++) {
  cout << i << "\n";
}
```

5.2 Cicli annidati

E' possibile inserire un `for` all'interno di un'altro `for` (senza limiti di volte), ottenendo un ciclo annidato. In questo caso, il ciclo esterno viene eseguito prima del ciclo interno.

5.2.1 Esempio

```cpp
// ciclo esterno
for (int i = 1; i <= 2; ++i) {
  cout << "Outer: " << i << "\n"; // Eseguito 2 volte

  // Inner loop
  for (int j = 1; j <= 3; ++j) {
    cout << " Inner: " << j << "\n"; // Eseguito 6 volte (2 * 3)
  }
}
```

5.3 Esercitazione

> Dato una numero mostrato a schermo, mostrarne la tabellina.

esempio di output:

```
Quale tabellina vorresti mostrare? 5
5 * 1   =  5
5 * 2   =  10
5 * 3   =  15
5 * 4   =  20
5 * 5   =  25
5 * 6   =  30
5 * 7   =  35
5 * 8   =  40
5 * 9   =  45
5 * 10  =  50
```

```cpp
#include <iostream>
using namespace std;

int main(){

 // creiamo la variabile per salvare la tabellina
 int numero;

 cout<<"Quale tabellina vorrestri mostrare? ";

 // facciamo inserire all'utente il numero
 cin>>numero;

 // vogliamo fare una azione ripetuta
 // partena -> 1
 // arrivo  -> 10 (incluso)
 for(int cont=1; cont <= 10; cont = cont+1){
```

```
18
19    // stiamo mostrando qualcosa a schermo:
20    // numero " * " cont " = " prodotto
21    int prodotto= numero*cont;
22
23    // 5 * 1   =  5
24    cout<<numero<<" * "<<cont<< " = "<<prodotto<<endl;
25   }
26
27   return 0;
28  }
```

6 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:

```
while (condizione) {
  codice da eseguire
}
```

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++:

```
int i = 0;
while (i < 10) {
  cout << i << endl;
  i++;
}
```

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

7 Do while

In C++, il costrutto do-while è una variante del costrutto while che esegue il codice contenuto nel suo blocco almeno una volta, indipendentemente dal valore della condizione di uscita. In altre parole, il codice all'interno del blocco do viene eseguito almeno una volta, quindi la condizione viene verificata e, se risulta vera, il codice viene eseguito nuovamente. Questo processo continua finché la condizione resta vera.

Ecco un esempio di come si può utilizzare il costrutto do-while in C++ per stampare una sequenza di numeri fino a quando l'utente non inserisce il numero 0:

```cpp
#include <iostream> // Include l'header per utilizzare std::cin e std::cout

int main()
{
    int num;

    do
    {
        // Chiede all'utente di inserire un numero
        std::cout << "Inserisci un numero (0 per uscire): ";
        std::cin >> num;

        // Stampa il numero inserito
        std::cout << "Hai inserito: " << num << std::endl;
    }
    while (num != 0); // Continua a eseguire il blocco finché num non é 0

    return 0;
}
```

8 Array

In C++, un array è una struttura dati che consente di memorizzare più valori di un determinato tipo in un unico contenitore. Gli elementi di un array sono accessibili tramite un indice, ovvero un numero che indica la posizione dell'elemento all'interno dell'array.

Per dichiarare un array in C++, si utilizza la sintassi seguente:

```
tipo nomeArray[dimensione];
```

Ad esempio, per dichiarare un array di 10 elementi di tipo int, potremmo scrivere:

```
int mieiNumeri[10];
```

Per assegnare un valore a un elemento dell'array, si utilizza la sintassi seguente:

```
nomeArray[indice] = valore;
```

Ad esempio, per assegnare il valore 10 all'elemento con indice 0 dell'array mieiNumeri, potremmo scrivere:

```
mieiNumeri[0] = 10;
```

Per accedere al valore di un elemento dell'array, si utilizza la sintassi seguente:

```
valore = nomeArray[indice];
```

Ad esempio, per stampare il valore dell'elemento con indice 0 dell'array mieiNumeri, potremmo scrivere:

```
std::cout << mieiNumeri[0] << std::endl;
```

Questo codice stamperà il seguente output a video:

```
10
```

9 Vector

In C++, un vettore è una struttura di dati che rappresenta una sequenza di elementi di uno stesso tipo. I vettori sono molto utili quando si ha bisogno di lavorare con una serie di dati di uno stesso tipo, ad esempio per memorizzare le informazioni di un gruppo di studenti o per gestire un elenco di numeri.

Per utilizzare i vettori in C++, è necessario includere l'header <vector> e creare un oggetto vettore utilizzando il tipo std::vector. Una volta creato l'oggetto vettore, è possibile utilizzare i vari metodi e funzioni forniti dalla libreria per gestire la sequenza di elementi.

Ecco un esempio di come si può utilizzare un vettore in C++ per memorizzare le informazioni di un gruppo di studenti:

```cpp
#include <vector> // Include l'header per utilizzare i vettori
#include <string> // Include l'header per utilizzare le stringhe

int main()
{
    // Crea un vettore di stringhe per memorizzare i nomi degli
       studenti
    std::vector<std::string> studenti;

    // Aggiunge alcuni studenti al vettore
    studenti.push_back("Mario Rossi");
    studenti.push_back("Luigi Bianchi");
    studenti.push_back("Chiara Verdi");

    // Stampa il numero di studenti nel vettore
    std::cout << "Numero di studenti: " << studenti.size() << std::endl
       ;

    // Stampa il nome di ogni studente
    for (int i = 0; i < studenti.size(); i++)
    {
        std::cout << "Nome studente: " << studenti[i] << std::endl;
    }

    return 0;
}
```

10 Stringhe

In C++, una stringa è una sequenza di caratteri che rappresenta un testo. Una stringa può essere dichiarata utilizzando la classe string, che fa parte del namespace std.

Per dichiarare una stringa in C++, si utilizza la sintassi seguente:

```
1 std::string nomeStringa;
```

Ad esempio, per dichiarare una stringa di nome saluto, potremmo scrivere:

```
1 std::string saluto;
```

Per assegnare un valore a una stringa, si utilizza l'operatore di assegnazione =, ad esempio:

```
1 saluto = "Ciao Mondo";
```

Per accedere ai singoli caratteri di una stringa, si utilizza la sintassi seguente:

```
1 carattere = nomeStringa[indice];
```

Ad esempio, per accedere al primo carattere della stringa saluto, potremmo scrivere:

```
1 char primoCarattere = saluto[0];
```

In questo caso, la variabile primoCarattere conterrà il valore C.

11 Strutture (struct)

Le strutture, solitamente chiamate `struct`, consentono il raggruppamento di più informazioni all'interno di un solo contenitore.

> Puoi pensare a una struct come a una scatola con scompartimenti, in cui ogni scompartimento può contenere a sua volta altri oggetti.

11.1 Sintassi

Per realizzare una struttura è sufficiente utilizzare la keyword `struct` seguita dalle `{}`; al loro interno andremo a mettere le variabili che faranno parte della nostra struttura.

```
1  struct {             // dichiarazione della struttura
2      int  contenuto1; // membro (variabile int)
3      char contenuto2; // membro (variabile char)
4      // ...
5  } nomeStruttura;     // nome con cui verrà richiamata
```

11.2 Accedere ai membri

Per accedere a una variabile membro della struttura si utilizza il nome della variabile seguita dal carattere `.`.

```
1  // dichiarazione della struttura
2  struct {
3      int  contenuto1; // membro (variabile int)
4      char contenuto2; // membro (variabile char)
5      // ...
6  } nomeStruttura;     // nome con cui verrà richiamata
7
8  // istanzio una struttura
9  nomeStruttura st1;
10
11 // ne assegno i valori
```

```
st1.contenuto1 = 1;
st1.contenuto2 = 'A';
```

11.3 Esercitazione

Costruire un record **Prodotto** con:

- nome
- marca
- prezzo unitario
- data scadenza
- quantità in magazzino

Fare l'input e l'output di un prodotto, stampare poi il valore totale del prodotto accantonato in magazzino.

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

// definiamo il valore massimo di caratteri in una stringa
#define size 20

// creazione della struttura per l'inserimento della data
struct data{
    int giorno;
    int mese;
    int anno;
};

// creazione della struttura per il prodotto
struct prodotto{
  char nome[size];
  char marca[size];
  float prezzo_unitario;
  struct data scadenza;
  int quantita;
};


int main(){

  // creaimo una variabile per salvare
  // le informazioni del prodotto
  struct prodotto pippo;

  // chiediamo di inserire il nome
```

```
31    printf("Inserire il nome del prodotto: ");
32    scanf("%s", pippo.nome);
33
34    // chiediamo di inserire la marca
35    printf("Inserire la marca del prodotto: ");
36    scanf("%s", pippo.marca);
37
38    // chiediamo di inserire il prezzo unitario
39    printf("Inserire il prezzo unitario: ");
40    scanf("%f", &pippo.prezzo_unitario);
41
42    // chiediamo la scadenza del prodotto
43    printf("Inserire il giorno di scadenza: ");
44    scanf("%d", &pippo.scadenza.giorno);
45
46    printf("Inserire il mese di scadenza: ");
47    scanf("%d", &pippo.scadenza.mese);
48
49    printf("Inserire l'anno di scadenza: ");
50    scanf("%d", &pippo.scadenza.anno);
51
52    // inseriamo la quantità dei prodotti
53    printf("Inserire la quantità di prodotto: ");
54    scanf("%d", &pippo.quantita);
55
56
57    printf("Il prodotto %s ha le seguenti caratteristiche: \n", pippo.
          nome);
58    printf("- nome: %s\n", pippo.nome);
59    printf("- marca: %s\n", pippo.marca);
60    printf("- prezzo unitario: %f\n",pippo.prezzo_unitario);
61    printf("- scadenza: %d-%d-%d\n", pippo.scadenza.anno, pippo.scadenza.
          mese, pippo.scadenza.giorno);
62    printf("- quantita': %d\n", pippo.quantita);
63
64    float valore_totale;
65    valore_totale = pippo.prezzo_unitario* pippo.quantita;
66
67    printf("Il valore totale e' di %.2f euro", valore_totale);
68
69    return 0;
70 }
```

12 Puntatori

I puntatori sono una caratteristica fondamentale del linguaggio di programmazione C++. Sono variabili che contengono indirizzi di memoria di altre variabili. In altre parole, un puntatore "punta" a un'altra variabile e ne permette l'accesso.

Ad esempio, supponiamo di avere una variabile intera chiamata `x` che contiene il valore 10. L'indirizzo di memoria di questa variabile è, ad esempio, `0x7ffc34aa2b10`. Se si vuole creare un puntatore che punti a `x`, si può fare in questo modo:

```
1 int x = 10;
2 int *ptr = &x;
```

In questo caso, ptr è un puntatore alla variabile `x`. L'operatore & viene utilizzato per ottenere l'indirizzo di memoria della variabile x e assegnarlo al puntatore ptr. A questo punto, ptr contiene l'indirizzo di memoria di `x`, cioè `0x7ffc34aa2b10`.

Per accedere al valore contenuto nella variabile a cui punta un puntatore, si utilizza l'operatore di **dereferenziazione** `*`. Ad esempio, per stampare il valore di x tramite il puntatore ptr, si può usare il seguente codice:

```
1 std::cout << *ptr;  // Stampa il valore di x, cioé 10
```

I puntatori sono uno strumento molto potente in C++, ma devono essere utilizzati con attenzione per evitare errori e problemi di memoria.

Ricorda: è importante ricordarsi di gestire correttamente l'allocazione e la liberazione della memoria quando si lavora con puntatori.

12.1 Riferimenti

I riferimenti in C++ sono una caratteristica della linguaggio che consente di creare un alias per una variabile esistente. In pratica, quando si crea un riferimento per una variabile, si sta creando un nuovo nome per quella variabile. Questo può essere utile in diverse situazioni, ad esempio per evitare di

dover passare grandi strutture di dati per valore a una funzione, o per creare un alias per una variabile globale all'interno di una funzione.

Per creare un riferimento in C++, si utilizza l'operatore di riferimento '&' quando si dichiara la variabile. Ad esempio:

```
int x = 5;
int &y = x;
```

In questo esempio, y è un riferimento per x. Questo significa che qualsiasi cambiamento apportato a y sarà automaticamente applicato anche a x, e viceversa. Ad esempio, se si assegna un nuovo valore a y, come nell'esempio seguente:

```
y = 10;
```

Il valore di x verrà automaticamente aggiornato a 10, poiché y è un alias per x.

I riferimenti possono essere utilizzati in diverse situazioni, ad esempio come parametri di funzione o come valori di ritorno. Ad esempio, si può definire una funzione che accetta un riferimento a una variabile come parametro, come nell'esempio seguente:

```
void increment(int &value) {
  value++;
}
```

In questo caso, la funzione `increment` accetta un riferimento a una variabile di tipo int e incrementa il suo valore di 1. Se si passa una variabile per valore a questa funzione, il suo valore verrà modificato all'interno della funzione, ma non verrà modificato una volta che la funzione terminerà. Invece, se si passa una variabile per riferimento (come nell'esempio), il suo valore verrà modificato anche una volta che la funzione terminerà.

12.1.1 Caratteristiche

I riferimenti in C++ hanno diverse caratteristiche interessanti:

- Un riferimento deve essere inizializzato con una variabile esistente al momento della sua creazione, e non può essere modificato successivamente. In altre parole, una volta creato un riferimento, non è possibile associarlo ad un'altra variabile.
- Un riferimento può essere utilizzato come una qualsiasi altra variabile, ad esempio per essere passato come parametro ad una funzione o per essere utilizzato in un'espressione.
- I riferimenti hanno lo stesso tipo della variabile a cui sono associati. Ad esempio, se si crea un riferimento per una variabile di tipo int, il riferimento avrà lo stesso tipo di quella variabile.