Ho acquistato il corso:  
[**The C++20 Masterclass : From Fundamentals to Advanced**](https://www.udemy.com/course-dashboard-redirect/?course_id=2987082)

# 20-11-2022

* Il codice di ogni lezione è disponibile su [GitHub](https://github.com/rutura/The-C-20-Masterclass-Source-Code)
* Le slide di ogni lezione (a partire dalla terza) sono allegate alla lezione finale di ogni sezione
  + Ci sono anche le slide complessive, le ho scaricate [qui](Slides-Single_zip)
* Useremo Visual Studio Code
  + Provando vari compilatori.  
    Su Windows ci sono
    - MSVC,
    - GCC,
    - Clang
  + Nota ”GCC compiler is wrapped in Mingw compiler in Windows”
* Avevo già Visual Studio Code installato
* Ho installato Visual Studio 2022 (l’IDE) per avere Msvs (il compilatore c++ Microsoft)
  + Controllo che il compilatore sia presente come suggerito
    - Apro Text, logo

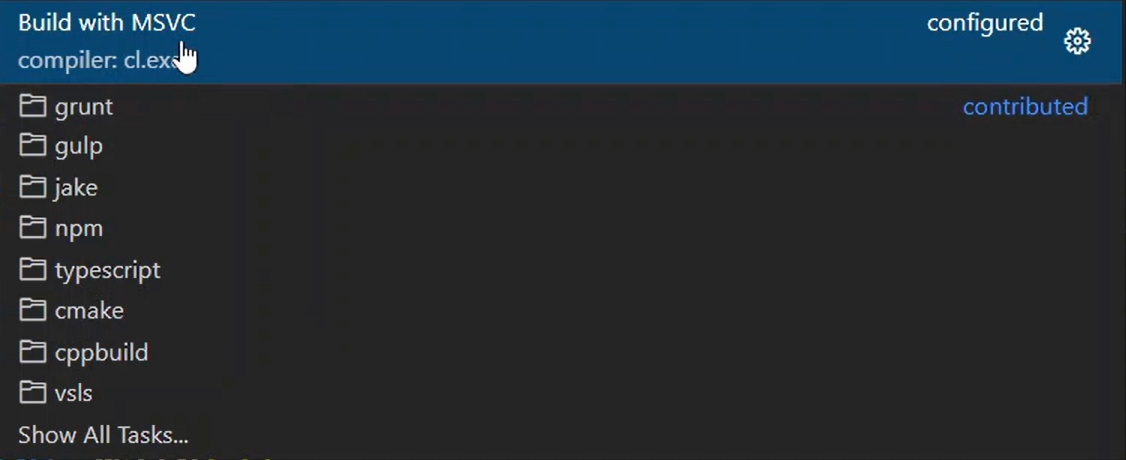
      Description automatically generated
    - Dando *cl.exe*si ha un output che conferma la presenza del compilatore  
      Text

      Description automatically generated
* Lez. 8 conclusa, mi fermo

# 21-11-2022

* Lez. 9:
  + Per avviare Visual Studio Code in modo che sappia dove si trova il compilatore devo **avviarlo da terminale** Text, logo

    Description automatically generated con “*code .*”  
    Text

    Description automatically generated
  + Ho clonato la repository:  
    > C:\Users\Luca\Documents\GitHub\The-C-20-Masterclass-Source-Code
* Cancello I file e seguo la lezione
  + Andando in Terminal-> config tasks   
    puoi selezionare il compilatore e trovi suggerito cl.exe
  + In automatico si crea un file .Json
    - Però non è configurato per il c++20, mentre nel main.cpp c’è l’espressione “consteval” che è del cpp20
  + Aggiungere cpp20: copia gli “args” del tasks.json dalla repo
    -  questo dice di usare il c++ puù recente, quindi c++20
    -  questo dice che ogni file .cpp presente nella cartella di lavoro va compilato
    - questo dice di creare in uscita il file eseguibile chiamato “rooster.exe”
    - Modifica  con 
  + BUILD: andando in terminal->run task
    - Viene proposto quanto abbiamo impostato nel file tasks  
      
  + Abbiamo il file roostes.exe  
    A picture containing text

    Description automatically generated
  + Da terminale possiamo lanciarlo con .\rooster.exe   
    Text

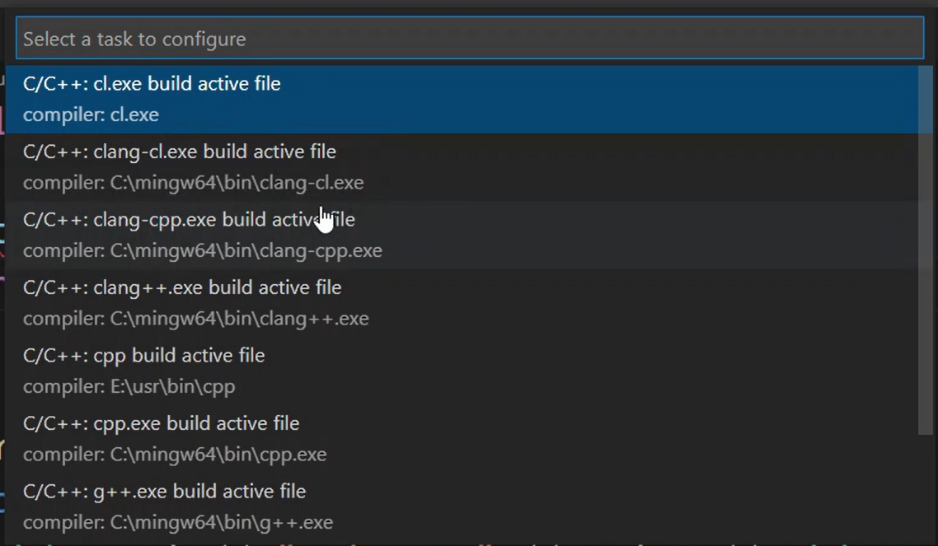
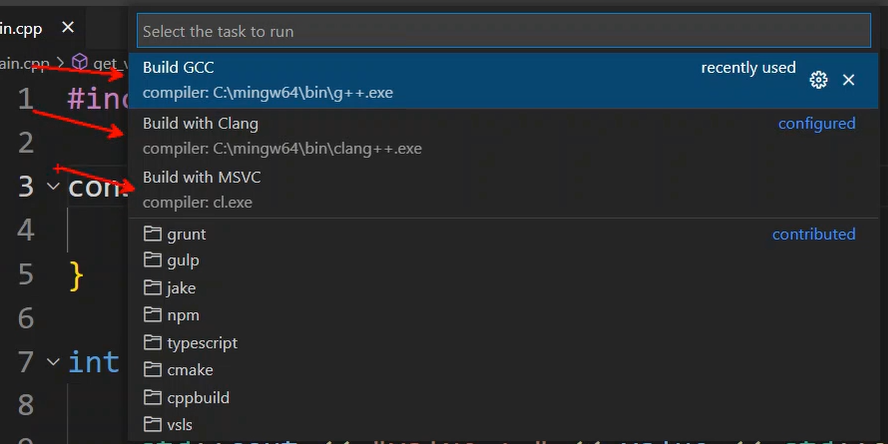
    Description automatically generated
* Risolvere le sottolineature rosse  
  A screenshot of a computer

  Description automatically generated with medium confidence
  + Questo accade perché VScode non sa di dover usare i tipi standard di c++20 per interpretare il codice che scriviamo (ma il compilatore sì, per quello ha compilato)
  + Dobbiamo configurare l’intellisense
  + Vai in *view-> command palette -> C/C++ edit configuration (UI)*Graphical user interface, text, website

    Description automatically generated
    - Qui puoi vedere le config dell’*intellisense*
    - Vedi che viene usato come compilatore quello di VisualStudio 2022
    - Modifica questo  
      Text

      Description automatically generated
  + Ora le sottolineature sono scomparse  
    A screenshot of a computer

    Description automatically generated with medium confidence
* Lez. 10
  + Installazione di altri compilatori: GCC-CLANG
    - Ho messo la cartella estratta in   
      C:\mingw64
    - Ho aggiunto al path il percorso di gcc e clang  
      test OK  
      Text

      Description automatically generated
* Lez. 11
  + Configurare VScode per Gcc
  + Quando vado in terminal->run task  
    ora ci sono tanti compilatori  
    
* Lez. 12
  + Configurare per Clang
  + Stessa procedura
    - Vai in terminal->run task->configure
    - Scegli clang++
* Lez 13
  + Configurare un Progetto per usare alternativamente tutti i compilatori che abbiamo visto!
  + Molto utile se hai un progetto cross-platform, così puoi usare il compilatore di ogni piattaforma e vedere se va tutto a buon fine
  + In pratica **basta aggiungere configurazioni al file tasks.json**
  + Adesso, quando vai in *terminal->run\_task->* permette di scegliere fra i vari compilatori
  + 
  + Per creare nuovi progetti sulla base di questo posso semplicemente copiare la cartella e usarla per il nuovo progetto!

# 21-11-2022

* Sezione “Diving In”
* Lez 31
  + la libreria *iostream*
    - std::cout << “hello World” << std::endl;
  + return 0 dal main indica nessun errore
* Lez 32
  + Commenti
* Lez 33  
  errori e warning
  + Compile time errors vs runtime errors:  
    gli errori a compile time sono quelli che becca il compilatore, quindi errori di sintassi  
    gli errori a runtime sono errori che non sono noti a tempo di compilazione, ad es una divisione per 0.
* Lez 34:  
  statement e funzioni
  + Niente di interessante
  + Cout con numeri  
    std::cout << “var value is:” << var
* Lez 35:  
  Input & Output

|  |  |
| --- | --- |
| Stream | purpose |
| Std::cout | Print data to the console (terminal) |
| Std::cin | Read data from the terminal |
| Std::cerr | Print errors message to the console |
| Std::clog | Print log messages to the console |

* + In std ci sono anche le stringhe!  
    std::string
  + NOTA: se provi a usare *cin* e dai una stringa che contiene uno spazio, non la prenderà tutta  
    per risolvere usa  
    *std::getline(std::cin, <your string with spaces>);  
      
    std::getline* prende in ingresso una stringa e gli diamo quella che ritorna *std::cin*
* Lez. 36
* **Lez. 37  
  differenza tra le *Core Features/Standard Library/STL***
  + *Core Features*tutte le cose base del c++, come la sintassi ecc
  + *Standard Library*raggruppa un insieme di componenti specializzati da usare per scrivere il programma più in fretta. Non aumenta le features del c++, solo ne facilita l’uso, ad es iostream
  + *STL*è parte della standard library, ma è specializzata in *container* e metodi per gestirli, così come *iterators*.
* Lez. 39  
  Tipi
  + Esiste il tipo “Auto” che indica di dedurre il tipo da altre variabili…lo vedremo poi
* Lez. 40  
  basi numeriche (binario, esadecimale ecc)
  + 0x per esadecimale
  + 0 per ottale
  + 0b per binario
* Lez 41
  + Puoi inizializzare le variabili con   
    *int var {1}  
    int var (1)*
  + Quando non inizializzi  
    *int var; // può contenere cose a caso, generalmente 0 ma non è detto  
    int var{}; // sicuramente inizializzata a zero*
  + Puoi inizializzare con un’espressione  
    *int a = 0;  
    int b = 1;  
    int c = a + b;*
  + **Functional initialization VS brace initialization VS assignment initialization** 
    - Se provi a inizializzate un intero con un numero con la virgola dà errore! (FIGO, io ero abituato che troncava), **questo se usi le graffe!**  
      *int a {2.9}; // errore*
    - Se **non usi le graffe troncherà il valore, quindi le graffe sono un metodo di inizializzazione più sicuro!**  
      *int a (1.9); //compila (detto functional initialization)*
    - Inizializzazione con assegnamento  
      *int a = 1;   
      int b = 1.7; // scriverà troncando !*

# 28-11-2022

* Lez 42
  + Unsigned and signed ints
  + Short and long ints
    - Funzionano solo su variabili che immagazzinano numeri interi

# 02-12-2022

* Lez 43
  + Fractional numbers (FLOATING POINT)
  + 3 tipi:
    - **Float** 4 byte
    - **Double** 8byte
    - **Long double** 12byte
  + Puoi scriverlo manualmente o usare la **notazione scientifica**
  + Con i float puoi dividere per zero un numero non nullo, o avere 0/0
    - Num/0 = +-**infinity**
    - 0/0 = **Nan**
  + Il compilatore usa **double come default**, ricordati di mettere *f* o *L* alla fine del numero se vuoi assicurarti che quel valore venga trattato come *float* o *long double*
  + Usa *std::setprecision(<val>)* per indicare a cout di dare output testuali con precisione maggiore
    - Per usare questa funzione devi includere iomanip

# 12-12-2022

* Lez