Introduzione alla Programmazione - Ambiente di sviluppo - CT0441

Claudio Lucchese Alvise Spanò Alberto Veneri Antonio Cinà
Federico Marcuzzi

Indice

1 Introduzione

In questo PDF potrete trovare dei brevi tutorial per aiutarvi a completare il setup del vostro computer in modo da seguire al meglio il corso di Introduzione alla Programmazione.

Vi proponiamo una breve guida sui comandi principali da utilizzare su una Unix shell, un tutorial per l'installazione di un compilatore C su alcuni dei sistemi operativi più diffusi e i passi principali per la configurazione dell'IDE che utilizzeremo in questo corso.

Per qualsiasi dubbio vi consigliamo di aprire una discussione sul forum moodle del corso, dove potrete porre domande, collaborare e dare delle risposte con la supervisione degli insegnanti e dei tutor.

2 Comandi di base per una Unix Shell

2.1 Terminologia

Prima di iniziare con una breve guida ai comandi per la shell, è bene fare chiarezza con i termini. Con il termine "Unix Shell", o più semplicemente "shell", intendiamo l'interfaccia a linea di comando che vi permette di interagire con un Sistema Operativo (SO) Unix-like. Per fare un esempio, sono SO Unix-like tutti i sistemi operativi basati sul kernel Linux e il sistema operativo macOS prodotto da Apple. Windows fa parte di un'altra categoria di sistemi operativi, ma potrete eseguire i comandi di una shell utilizzando il Windows Subsystem for Linux. Studierete più nel dettaglio la differenza fra i vari sistemi operativi nel corso di Sistemi Operativi (CT0125).

2.2 Utilizzare una Unix Shell su Windows

Per utilizzare una Unix Shell su Windows vi consigliamo di installare il sottosistema Windows per Linux seguendo l'apposita guida ufficiale. Vi suggeriamo di installare la seconda versione del sistema (WSL 2) e di usare come distribuzione Ubuntu 20.04 LTS.

2.3 Gli emulatori del terminale

Gli emulatori del terminale sono dei programmi che permettono di simulare il funzionamento di un terminale testuale e quindi di simulare l'interfaccia a linea di comando.

Tra gli emulatori più popolari troviamo:

- GNOME Terminal per l'ambiente desktop GNOME e quindi usato anche in Ubuntu Gnome
- Xterm per i SO che usano il gestore grafico X Window System
- Terminal per macOS

Tutti i comandi che vedremo dovranno essere eseguiti all'interno di un emulatore del terminale che per semplicità chiameremo "terminale".

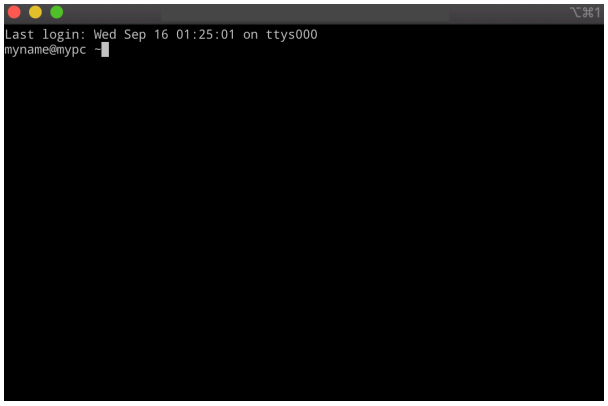


Figura 1: Screenshot di Terminal

2.4 I primi comandi

Dopo aver aperto un terminale, vi troverete di fronte ad una schermata come quella in Figura 1.

Da qui potrete cominciare ad utilizzare alcuni comandi di base della shell, basterà scrivere il comando e alla fine premere il tasto invio.

2.4.1 Print Working Directory - pwd

Dopo aver aperto il terminale vi troverete in maniera predefinita all'interno della cartella home dell'utente corrente. Per controllare esattamente la cartella in cui vi trovate potete usare il comando:

```
pwd
```

In risposta otterrete il vostro path corrente. Per esempio se state utilizzando macOS, vi trovare nella vostra home e se il vostro utente si chiama `mariorossi`, riceverete in risposta:

```
/Users/mariorossi
```

2.4.2 Change Directory - cd

Per cambiare la directory corrente potete utilizzare il comando `cd` che ha la seguenti sintassi di base:

```
cd <percorso>
```

dove `<percorso>` rappresenta il nuovo percorso dove spostare la directory corrente. Potete poi verificare la vostra posizione corrente utilizzando il comando `pwd`. Vi consigliamo, per esempio, di spostarvi all'interno della cartella `/tmp` che è adibita a contenere file temporanei nei sistemi Unix-like.

```
mariorossi@mypc ~ cd /tmp
mariorossi@mypc ~ pwd
/tmp
```

2.4.3 List - ls

Per avere una lista dei file e delle cartella presenti all'interno della cartella corrente potete usare il comando:

```
ls
```

Se per esempio vi trovate nella vostra home directory (la cartella solitamente di default all'apertura del terminale), l'output che otterrete sarà simile al seguente:

```
mariorossi@mypc ~ ls
Applications          Downloads             Public
Library               Movies                Desktop
Music                 Documents             Pictures
```

2.4.4 Make Directory - mkdir

Se volete creare una cartella dovrete usare un comando del tipo:

```
mkdir <nome cartella>
```

dove `<nome cartella>` rappresenta il nome della cartella che vogliamo creare all'interno della cartella corrente. Se volete creare per esempio una cartella con il nome `test` dovrete scrivere:

```
mkdir test
```

Potrete poi verificare l'avvenuta creazione della cartella usando il comando `ls`. Se volete farlo all'interno della cartella `/tmp` dovete scrivere:

```
mariorossi@mypc ~ cd /tmp
mariorossi@mypc ~ mkdir test
mariorossi@mypc ~ ls
a-file                        a-folder
another-file                  test
another-folder                tmp-inception
```

2.4.5 Touch - touch

Per creare un file vuoto potete usare il comando `touch` con la seguente sintassi:

```
touch <nome file>
```

In questo modo creerete un file vuoto di nome `<nome file>` all'interno della cartella corrente. Per esempio se volete creare un file di nome `main.c` il comando da eseguire è:

```
touch main.c
```

2.4.6 Remove - rm

ATTENZIONE: non usate questo comando su file o cartelle personali o di sistema, altrimenti potreste perdere definitivamente i vostri file e/o danneggiare il sistema.

Per eliminare un file potete usare il comando `rm` con la seguente sintassi:

```
rm <nome file>
```

Per esempio per eliminare il file creato in precedenza con touch, il comando sarà il seguente:

```
rm main.c
```

Se invece volete eliminare una cartella dovrete aggiungere l'opzione `-r` al comando. Per eliminare la cartella di test dovrete quindi eseguire:

```
rm -r test
```

Vi segnaliamo inoltre che potreste aver bisogno a volte dell'opzione `-f` per forzare l'eliminazione di un file o di una cartella.

2.5 Tutorial aggiuntivi

Per conoscere altri comandi, vi consigliamo di leggere le seguenti guide:

- https://wiki.ubuntu-it.org/AmministrazioneSistema/ComandiBase
- http://linuxcommand.org/index.php (in inglese)

3 Compilatore per C

In questa sezione vi condividiamo delle brevi istruzioni su come installare un compilatore per il linguaggio C (GCC o Clang). I sistemi su cui abbiamo testato questa guida sono stati i seguenti:

- Ubuntu 20.04.3 LTS
- macOS 11.6
- Windows 10 - build 19.041.450

Se notate delle differenze con il vostro sistema operativo, o avete dei dubbi sull'installazione, vi consigliamo di aprire una discussione nel forum.

3.1 Ubuntu

L'installazione del compilatore `gcc` sul sistema operativo Ubuntu richiede l'utilizzo dei privilegi di amministratore.

Il primo passo è l'aggiornamento dei pacchetti attualmente installati nel sistema, mediante il comando:

```
$ sudo apt update
```

Successivamente si passerà all'installazione del pacchetto `build-essential` (contenente gli strumenti necessari per lo sviluppo in C e C++) con il comando:

```
$ sudo apt install build-essential
```

A questo punto, se non ci sono stati errori e la procedura è terminata, potete lanciare dal terminale il seguente comando per verificarne la corretta installazione:

```
gcc --version
```

Dovreste ottenere un risultato simile o uguale a questo:

```
gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

3.2 macOS

Su macOS vi proponiamo di installare Clang attraverso il pacchetto contenente gli strumenti per lo sviluppo offerto dal sistema (Command Line Tools Package).
Per installarlo vi basterà aprire un terminale, ad esempio usando l'applicazione `Terminal.app`, ed usare il comando:

```
xcode-select --install
```

A quel punto vi apparirà una finestra che vi chiederà di installare gli strumenti per lo sviluppo come in Figura 2.

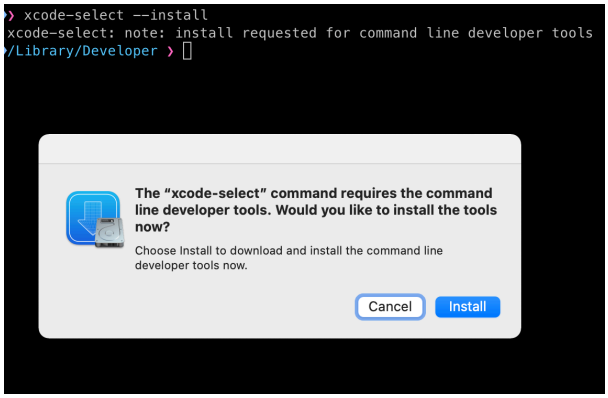


Figura 2: Screenshot finestra installazione Developer Line Tools

Proseguite quindi con il processo fino al termine dell'installazione del pacchetto.
Per verificare che l'installazione sia avvenuta con successo potete usare il comando:

```
clang -v
```

Con il quale dovreste ottenere in risposta le informazioni sulla vostra installazione, e dovreste ottenere come risultato qualcosa di simile a:

```
Apple clang version 12.0.5 (clang-1205.0.22.11)
Target: x86_64-apple-darwin20.6.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

3.3 Windows 10

Prima di tutto, vi consigliamo di installare il sottosistema Windows per Linux seguendo l'apposita guida ufficiale. Vi suggeriamo di installare la seconda versione del sistema (WSL 2) e di usare come distribuzione Ubuntu 20.04 LTS. A quel punto, aprendo l'applicazione Ubuntu 20.04 LTS vi troverete davanti ad una schermata come quella in Figura 3.

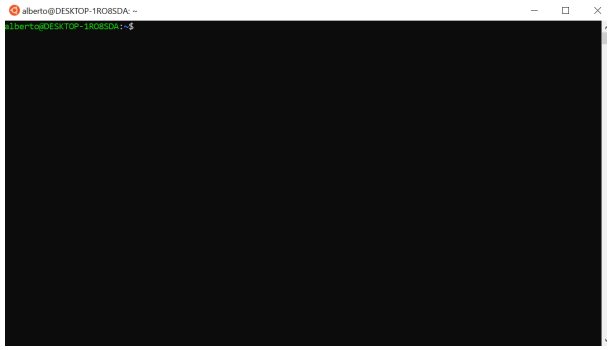


Figura 3: Screenshot WSL

Da qui potrete seguire gli stessi passi d'installazione di GCC per Ubuntu, ricordando che i vostri file saranno accessibili al path `/mnt/<lettera disco>/<resto del percorso>`. Ad esempio il file:

```
C:\Users\MarioRossi\Desktop\mytxt.txt
```

sarà disponibile a:

```
/mnt/c/Users/MarioRossi/Desktop/mytxt.txt
```

4 Compilare il primo programma - Hello World

Il primo programma più semplice che possiate scrivere è il classico "Hello world!", ovvero "Ciao mondo!". Per creare il vostro primo programma dovete aprire un editor di testo semplice, ad esempio `gedit`, `textEdit` o `Notepad`, e copiare ed incollare il seguente testo:

```
#include <stdio.h>

int main(){
    printf("Hello world!\n");
    return 0;
}
```

Dopodichè lo dovete salvare nella cartella che volete nominandolo come più vi piace. Ipotizziamo che lo abbiate salvato nel Desktop e che lo abbiate chiamato `main.c`. Per compilarlo da shell dovete: aprire il terminale, posizionarvi nella cartella rappresentante il Desktop e compilare usando gcc con la seguente sintassi:

```
gcc <flag di compilazione> <nome file sorgente> -o <nome programma eseguibile>
```

In questo caso useremo come flag di compilazione le seguenti opzioni:

- `-ansi`: per disabilitare le features di GCC che sono incompatibili con lo standard ISO C90.
- `-pedantic`: per richiedere al compilatore di essere più "pedante" nel rispettare le regole di ISO C, potete leggere più informazioni su cosa realmente accade a questo link.
- `-O2`: flag per richiedere al compilatore di applicare un ottimizzazione di livello 2 del codice eseguibile. Maggiori informazioni a questo link.

Dopo aver creato il vostro file nel Desktop, quello che dovete fare è spostarvi all'interno del vostro Desktop, compilare il vostro file `main.c` e infine eseguirlo per verificare che tutto sia andato a buon fine:

```
mariorossi@my-pc ~ cd ~/Desktop
mariorossi@my-pc ~ gcc -ansi -pedantic -O2 main.c -o my_exe
mariorossi@my-pc ~ ./my_exe
Hello world!
```

5 Un editor di testo per sviluppatori: Visual Studio Code

Gli editor di testo preinstallati nei PC, come ad esempio `gedit`, `textEdit` o `Notepad`, possono essere comodi per modificare dei file velocemente o scrivere il proprio primo semplice programma in `C`. Generalmente però si utilizzano degli editor di testo pensati appositamente per gli sviluppatori e talvolta questi prendono il nome di "Ambienti di Sviluppo Integrati", o "Development Integrated Environment" (IDE) in inglese, se forniscono un supporto avanzato allo sviluppatore in fase di scrittura del codice e debugging. Quello che vi presentiamo in questo breve tutorial viene definito un "code editor", un ibrido fra un text editor ed un IDE. Stiamo parlando di `Visual Studio Code` sviluppato da Microsoft con licenza open source MIT License. `Visual Studio Code` è in pratica un text editor con moltissime funzionalità aggiuntive dedicate agli sviluppatori, installabili grazie ad un vasta scelta di plugin proposta in un marketplace ad hoc. Naturalmente, dato che siamo interessati alla programmazione in `C`, quello che andremo a fare sarà installare l'estensione dedicata allo sviluppo con il linguagio `C`. Inoltre, daremo una breve lista di semplici passi da seguire per la configurazione dell'ambiente. Nei paragrafi seguenti vedremo come:

- Installare Visual Studio Code.
- Installare l'estensione per lo sviluppo a `C`.
- Configurare l'ambiente con il vostro compilatore con qualche suggerimento in base alla piattaforma utilizzata.

5.1 Installare Visual Studio Code

Per installare Visual Studio Code (VSCode) basterà andare sul sito di Microsoft dedicato e seguire le classiche operazioni per l'installazione di un qualsiasi applicativo per il vostro sistema operativo. È naturalmente possibile installare VSCode anche dal vostro package manager preferito, ad esempio con brew potete installarlo usando:

```
brew install visual-studio-code
```

Una volta installato, potete sia aprirlo dall'interfaccia grafica sia da linea di comando. Per esempio, digitando:

```
code
```

Si aprirà semplicemente VSCode, mentre digitando:

```
code <nome file o cartella>
```

Si aprirà il file (o la cartella) di nome `<nome file o cartella>` direttamente in VSCode.

5.2 Installare l'estensione C/C++ di Microsoft

Per installare l'estensione che vi agevolerà lo sviluppo di applicazioni in C, dovrete prima di tutto aprire il pannello per la gestioni delle estensioni cliccando sull'icona relativa nella barra delle attività che trovate alla sinistra della finestra di Visual Studio Code. L'icona sarà simile a quella dello screenshot qui sotto.

Figura 4: Icona pannello estensioni

Dopo aver fatto ciò, potete ricercare all'interno del marketplace l'estensione che si chiama `C/C++` avente come sviluppatore la stessa Microsoft. Cliccando poi il pulsante di installazione, `Install` nello screenshot sottostante, avvierete il processo di installazione.

Così facendo, installerete quindi l'estensione all'interno di VSCode e sarete pronti per sviluppare il vostro programma in C con tutte le agevolazioni fornite dalla stessa.

Un'altra possibilità è quella di installare l'estensione da linea di comando. La sintassi per installare un'estensione è:

```
code --install-extension <id estensione>
```

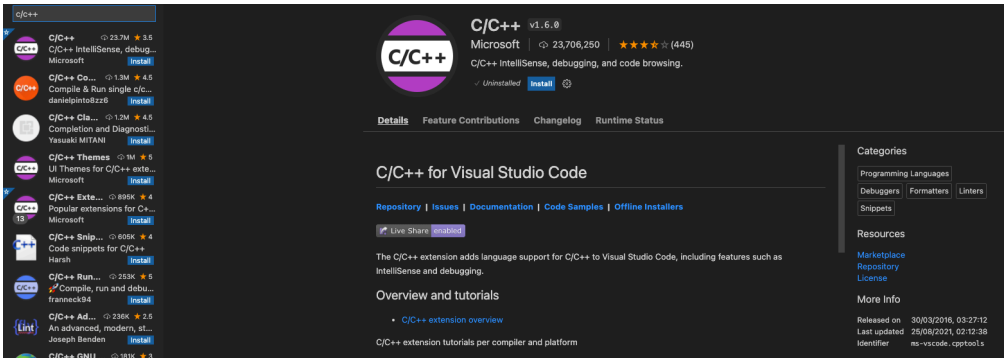


Figura 5: Icona pannello estensioni

Per installare l'estensione C/C++ via linea di comando potete quindi usare:

```
code --install-extension ms-vscode.cpptools
```

Per maggiori informazioni sulla gestione delle estensioni potete andare alla guida dedicata sul sito di VSCode.

5.3 Configurare l'estensione di C/C++

Per configurare correttamente l'estensione ed usufruire di tutti i suoi vantaggi, bisogna configurare il compilatore che dovrà utilizzare. Nello specifico, si consiglia di creare prima di tutto un file di prova `main.c` scrivendo, per esempio, un programma `Hello World` come quello presentato nella sezione precedente. Dopodiché, dovrete configurare il compilatore andando su `Terminal > Run Build Task` dal menù principale che troverete in alto nella finestra. Infine, dovrete scegliere da una schermata simile a quella qui sotto, il compilatore giusto per la vostra installazione, che solitamente sarà `gcc` oppure `clang`.

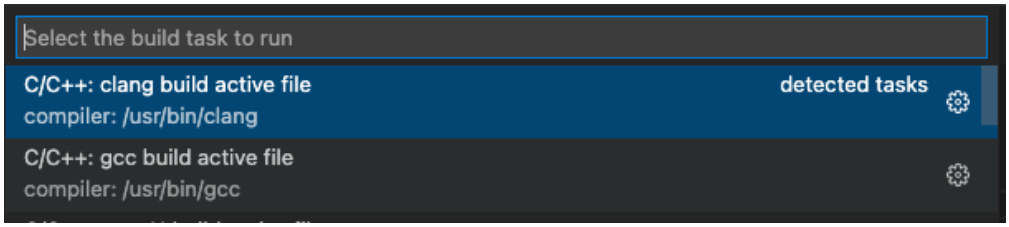


Figura 6: Run Build Task

Attenzione, per vedere gli stessi suggerimenti dei task per la compilazione dovrete aprire la cartella dov'è contenuto il file creando quello che viene chiamato in VSCode un `workspace`. Non vedrete la stesse opzioni se aprirete direttamente il file.

Cliccando sull'ingranaggio vicino a una delle opzioni del menu `Run Build Task` potrete modificare le impostazioni del task per la compilazione. Vi si aprirà quindi un file in formato JSON, dove potrete specificare le informazioni per la compilazione. Il contenuto predefinito del file sarà simile a quello proposto qui sotto:

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "cppbuild",
            "label": "C/C++: gcc build active file",
            "command": "/usr/bin/gcc",
            "args": [
                "-g",
                "${file}",
                "-o",
```

```
                "${fileDirname}/${fileBasenameNoExtension}"
            ],
            "options": {
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": "build",
            "detail": "compiler: /usr/bin/gcc"
        }
    ]
}
```

Si consiglia quindi di modificare il file per adattarlo agli standard che andremo ad utilizzare, e quindi scrivere in corrispondenza della chiave `args` le indicazioni per la compilazioni viste nelle sezioni precedenti. Altrimenti la compilazione di default sarà:

```
gcc -g <nome file> -o <nome file senza estensione>
```

Dove `-g` è un flag che fornisce delle informazioni in più per il debugging che consigliamo di lasciare in caso si volesse usare il debugger offerto da VSCode. Mentre per avere un comando di compilazione del tipo:

```
gcc -ansi -pedantic -O2 -g <nome file> -o <nome file senza estensione>
```

Dovrete cambiare il file di configurazione come segue:

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "cppbuild",
            "label": "C/C++: gcc build active file",
            "command": "/usr/bin/gcc",
            "args": [
                "-g",
                "-ansi",
                "-pedantic",
                "-O2",
                "${file}",
                "-o",
                "${fileDirname}/${fileBasenameNoExtension}"
            ],
            "options": {
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": "build",
            "detail": "compiler: /usr/bin/gcc"
        }
    ]
}
```

Vi consigliamo di leggere anche la guida dettagliata sul sito di VSCode nella quale potrete trovare tutte le indicazioni necessarie alla compilazione su diverse piattaforme. In particolare, nel caso usiate `WSL 2` ci sono dei passaggi aggiuntivi che trovate spiegati nel dettaglio sempre in una guida all'interno della documentazione di VSCode.

6 Un ambiente di sviluppo integrato: CLion

CLion è l'IDE (Integrated Development Environment) di punta prodotto da JetBrains per lo sviluppo in C e C++. È uno strumento a pagamento per i professionisti, ma per gli studenti JetBrains mette a disposizione delle licenze gratuite! Di seguito illustreremo come registrarsi al sito della JetBrains e richiedere la licenza per CLion.

Andare al seguente pagina https://www.jetbrains.com/shop/eform/students. A questo punto dovrebbe comparirvi una schermata simile a quella in Figura 7

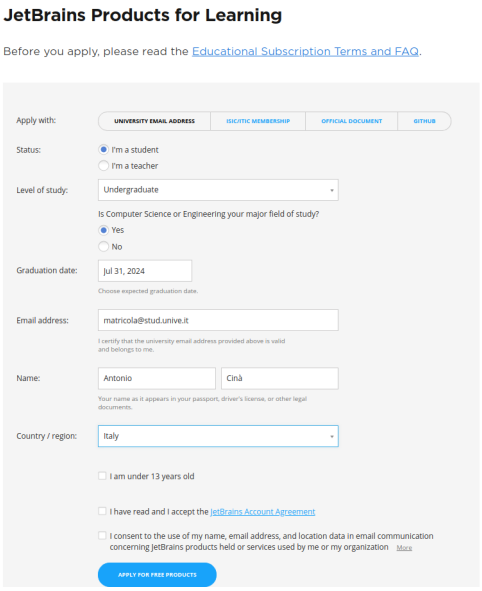


Figura 7: JetBrains SignUp

A questo punto vi viene chiesto di compilare il form con i vostri dati personali. Ricordate di utilizzare la vostra mail universitaria matricola@stud.unive.it. Noterete un `graduation date`, dove vi viene chiesta una stima del tempo di utilizzo del software, dovrete mettere una data coerente con il vostro percorso universitario.

Una volta compilato e confermato il form di registrazione riceverete una mail in cui vi viene chiesto di confermare l'attivazione dell'account. Ritornate sul sito della JetBrains e loggatevi con le vostre nuove credenziali (Figura 8)

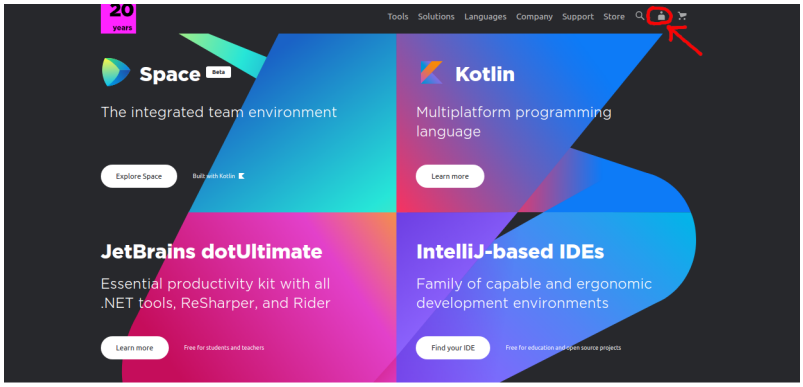


Figura 8: JetBrains login

Una volta effettuato il login vi si aprirà una pagine simile a quella in Figura 9

In rosso è evidenziato un link alla pagina di CLion dal quale potrete cliccare su download per scaricarlo, oppure potrete farlo direttamente dal menù a tendina `Download`.

Una volta scaricato avviate l'eseguibile e fate partire l'installazione. A questo punto vi verrà chiesto di inserire

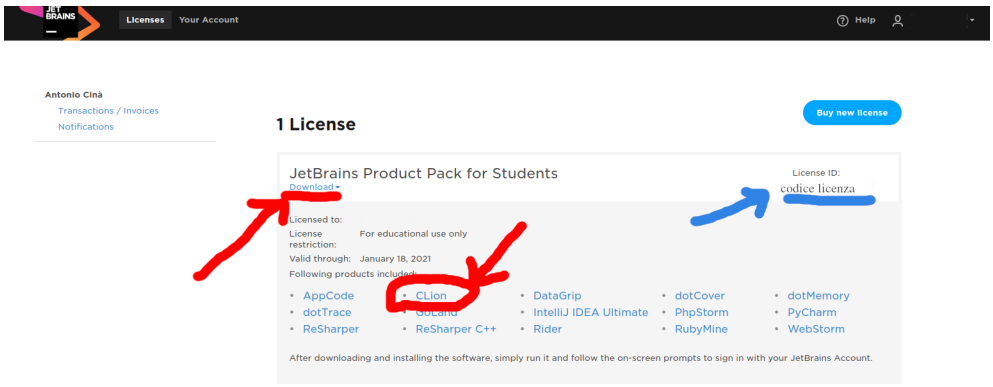


Figura 9: JetBrains download

la licenza, per cui scegliete "Attivazione con JB (JetBrains) account" e inserite le credenziali del profilo appena registrato.

6.1 CLion per utenti macOS e Linux

Al termine della fase iniziale di installazione l'ultima cosa che vi viene chiesta di configurare è la "toolchain" che utilizzerete, ovvero l'insieme di software che verranno usati per la compilazione e il debugging. Per macOS e Linux non dovreste avere molto altro da fare, se avete installato gli strumenti per sviluppare come suggerito nella sezione "Compilatore per C" del nostro tutorial, non avrete altro di cui preoccuparvi. In caso di dubbi o eventuali problemi riscontrati, potete leggere la guida Configure CLion on macOS, sebbene sia specificato per macOS, sarà facile per gli utenti Linux trovare le opportune corrispondenze.

6.2 CLion per gli utenti Windows

Per gli utenti Windows gli ultimi passi di configurazione sono leggermente diversi rispetto a quelli per macOS e Linux. Gli ambienti suggeriti da JetBrains per configurare la toolchain su CLion su Windows sono i seguenti: Cygwin, MinGW, WSL, oppure Microsoft Visual C++. Quello che vi suggeriamo noi se siete alle prime armi nel mondo dell'IT è di installare MinGW e seguire questo tutorial per collegare l'installazione di MinGW con CLion. Se invece volete collegare il vostro WSL a CLion, potete usare il seguente tutorial; questa soluzione la consigliamo solamente ad utenti più esperti.

6.3 CLion - Hello World

In questa sezione vi mostreremo come creare un "Hello, World!" in CLion ad installazione avvenuta.

Aprendo CLion per la prima volta vi troverete di fronte ad una immagine come in Figura 10.

Dopo aver selezionato "Nuovo progetto" ("New Project"), dovrete selezionare il tipo di progetto, ovvero "C executable", il tipo di standard che volete utilizzare, ovvero "C 90", e indicare la posizione per il salvataggio del progetto desiderata, nell'esempio qui sotto è: `/Users/mariorossi/Desktop/hello-world`

Al termine di questi primi due step, vi troverete davanti al vostro primo "Hello, world!" scritto utilizzando CLion!

Figura 10: Screenshot 1 - Hello World CLion

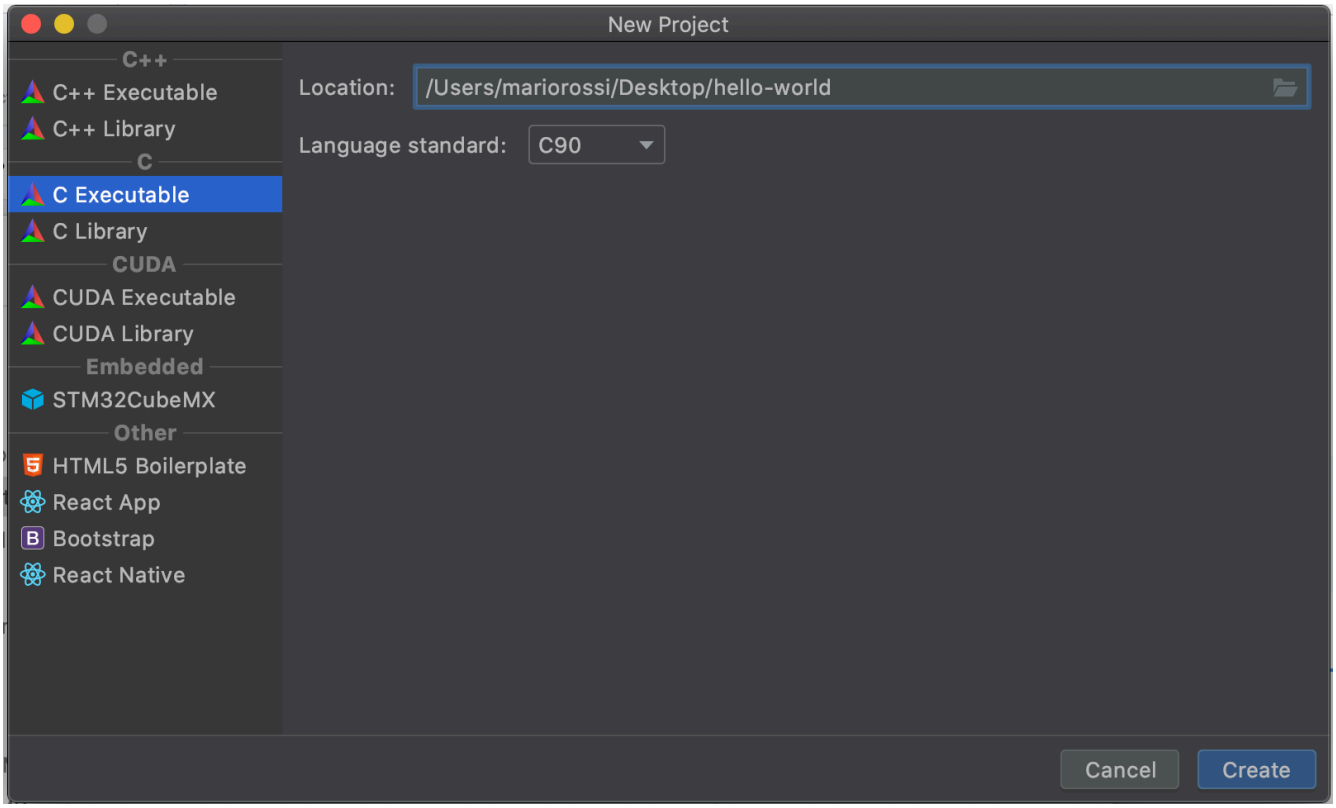


Figura 11: Screenshot 2 - Hello World CLion

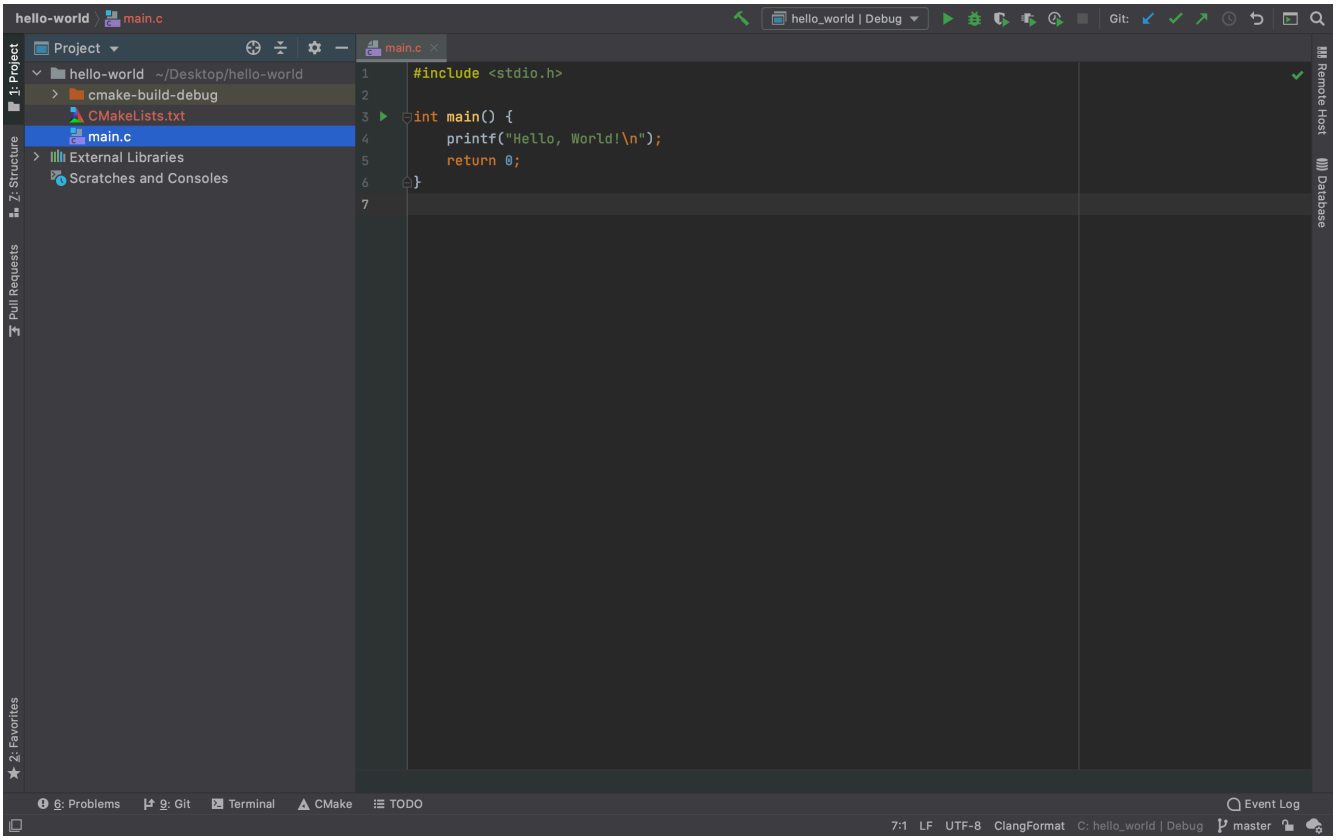


Figura 12: Screenshot 3 - Hello World CLion

Prima di poter eseguire il codice, dovrete modificare il file denominato `CMakeLists.txt` aggiungendo le seguenti righe:

- `set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -pedantic")` per aggiunger il flag `pedantic` al vostro compilatore
- `target_link_libraries(hello_world m)` per collegare la libreria contenente le funzioni matematiche (`-lm`)

Dovrete inserirle all'interno del file nello stesso ordine in cui compaiono in questo esempio:

```
cmake_minimum_required(VERSION 3.17)
project(hello_world C)

set(CMAKE_C_STANDARD 90)

set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -pedantic")

add_executable(hello_world main.c)

target_link_libraries(hello_world m)
```

Badate bene di sostituire il nome del vostro progetto al nome che abbiamo usato noi nell'esempio, ovvero `hello_world`.

Facciamo notare che abbiamo fatto queste modifiche perché vogliamo usare il flag di compilazione `-pedantic` e collegare la libreria contenente le funzioni matematiche come avremmo fatto da terminale con il comando:

```
gcc -ansi -pedantic main.c -lm
```

Il flag `-ansi` nel file `CMakeLists.txt` è invece rappresentato dal comando `set(CMAKE_C_STANDARD 90)`.

Dopo queste due modifiche vi verrà fuori il messaggio: `CMake project needs to be reloaded`, cliccate su `Reload`

`changes` per ricaricare ed aggiornare le impostazioni di compilazione del progetto. Adesso siete pronti per compilare ed eseguire il codice; per far ciò, non dovrete far altro che cliccare sul tasto play di colore verde nella parte alta della finestra e otterrete il risultato dell'esecuzione nella parte bassa della finestra.

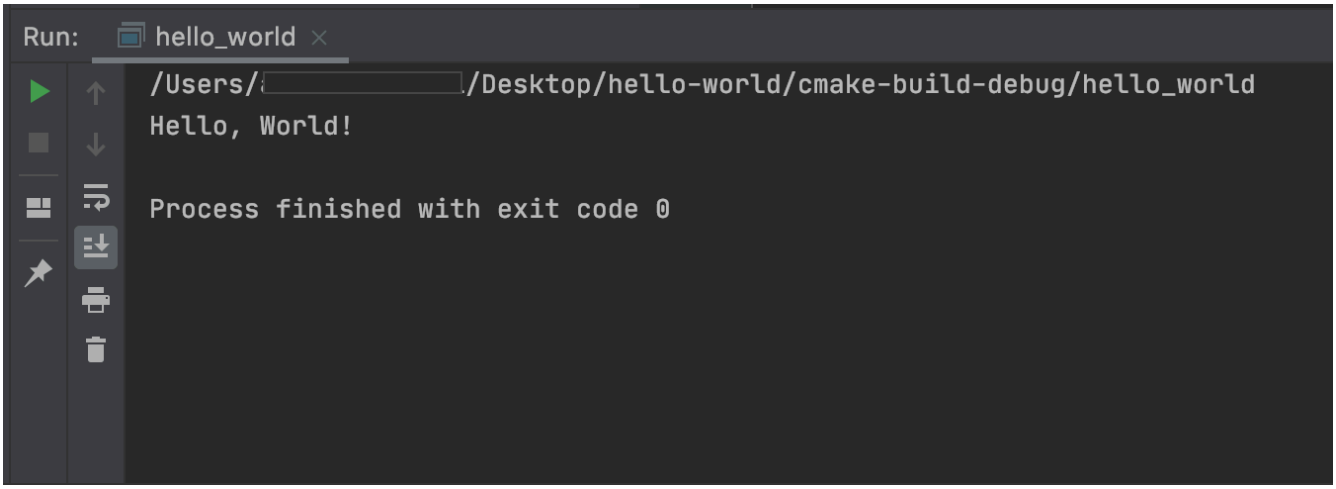


Figura 13: Screenshot 4 - Hello World CLion

Ultima nota, il flag di compilazione `pedantic` fa si che il compilatore lanci tutte le eccezioni richieste dagli standard ISO C; inoltre, rifiuta di compilare tutti i programmi che utilizzano estensioni non permesse e alcuni programmi che non seguono lo standard ISO C. Lo standard ISO C di riferimento è quello specificato con il comando `set(CMAKE_C_STANDARD 90)`, ovvero C90. Se volete che non vengano solamente lanciati dei warning, ma volete ricevere degli errori ogni volta che non rispettate una regola definita dallo standard C90, dovrete usare il flag `-pedantic-errors` al posto di `pedantic`.

7 Doxygen

7.1 Introduzione

Doxygen è uno strumento che permette di generare la documentazione per un codice sorgente in maniera automatica, semplicemente rispettando alcuni standard. Può essere usato sotto i diversi sistemi operativi (Windows, macOS, Linux, etc.) e supporta diversi linguaggi di programmazione, tra questi C. Il sistema estrae la documentazione che può essere esportata in diversi formati, come HTML e PDF.

7.2 Installazione

Per gli ambiente Windows, macOS e Linux vengono messi a disposizione gli eseguibili di installazione al seguente link: https://www.doxygen.nl/download.html. Basterà scaricarli ed eseguire le istruzioni di installazione per installare correttamente Doxygen sul vostro sistema.

Per installare invece il programma da linea di comando, su Ubuntu potete usare:

```
$ sudo apt-get install doxygen
```

Mentre su macOS, se avete installato homebrew, potete usare:

```
$ brew install doxygen
```

7.3 Regole di formato

Il funzionamento di Doxygen richiede una particolare formattazione dei commenti inseriti nel codice sorgente. Le regole di formattazione, oltre ad essere analoghe ad altri software con lo stesso fine, sono chiaramente documentate nel manuale. Per strutturare e analizzare la documentazione generata, Doxygen fornisce un numero elevato (> 170) di comandi speciali. Tutti i comandi nella documentazione iniziano con un backslash (\) o il carattere "chiocciola" (@), link lista completa.

Utilizziamo un esempio per presentare i più comuni tipi di commenti che possono venire interpretati da doxygen. Consideriamo il seguente esempio in un file `main.h`:

```c
/**
 * @file main.h
 * @author Antonio
 * @brief Una piccola libreria per il calcolo dell'area
 * e del perimetro di un cerchio
 * @date 07/12/2020
 */
#include <stdio.h>

/** Approssimazione di Pi greco **/
const float PI = 3.1415f;
/** Una possibile costante che rappresenta un raggio predefinito in metri **/
const float RADIUS_M = 7.82f;

/**
 * Calcola l'area di un cerchio dato il suo raggio (radius).
 * Formula: area = PI*radius*radius.
 * @param radius il raggio del cerchio
 * @return area l'area del cerchio
 */
float circle_area(float radius);

/**
 * Calcola il perimetro di un cerchio dato il suo raggio (radius).
 * Formula: perimetro = 2*PI*radius
 * @param radius
```

```
 * @return perimeter
 */
float circle_perimeter(float radius);
```

Nel `main.c` mettiamo le implementazioni delle funzioni in main.c come segue:

```
/**
 * @file main.c
 * @author antonio
 * @mainpage main PI program
 * @section intro_sec Introduzione
 *
 * Questa è una piccola introduzione, tipo la consegna e le specifiche del
 * programma.
 *
 * @section install_sec Installazione
 * Qui abbiamo creato una nuova sezione dove possiamo scrivere
 * le istruzioni per installare il programma
 * @subsection step1 Step 1: Installa gcc
 *
 * etc...
 */
#include "main.h"

float circle_area(float radius) {
  float area;
  area = PI * radius * radius;
  return area;
}

float circle_perimeter(float radius) {
  float perimeter;
  perimeter = 2 * PI * radius;
  return perimeter;
}

int main() {
  float radius, area, perimeter;
  radius = RADIUS_M;
  area = circle_area(radius);
  perimeter = circle_perimeter(radius);
  printf("Area = %.2f sq.m \n", area);
  printf("Perimetro = %.2f m\n", perimeter);
  return 0;
}
```

Bene! Abbiamo il nostro programma pronto e documentato… Siamo contenti cosi? No! Doxygen può fare di più! A partire da questo pezzo di codice possiamo generare per esempio dei file HTML che conterranno la nostra documentazione in modo organizzato e pulito. Dunque per compilare e generare il file HTML dobbiamo prima scrivere un file di configurazione. Con il comando:

```
$ doxygen -g Doxyfile
```

Il comando lo potete usare sia su Windows, macOS e Linux, all'interno del terminale di CLion.

Verrà dunque generato un file di configurazione per la nostra documentazione. Per maggiori dettagli sui tag di configurazione vi invitiamo a leggere la documentazione ufficiale al seguente indirizzo: https://www.doxygen.nl/manual/config.html. Una volta fatto, passiamo alla sua modifica di alcuni suoi parametri interni come segue:

```
# Il tag PROJECT_NAME identifica il progetto
```

```
PROJECT_NAME           = "PIdoxygen"

# Il tag OUTPUT_DIRECTORY determina dove andremo a
# salvare la nostra documentazione.
# in questo caso gli stiamo dicendo "dentro la
# cartella doc" (che abbiamo già creato)
OUTPUT_DIRECTORY       = doc

# Il tag OUTPUT_LANGUAGE determina la lingua con cui
# la documentazione è scritta

OUTPUT_LANGUAGE        = Italian

# Il tag INPUT tag specifica di quali file vogliamo
# la documentazione. Esempio main.categoria.
# oppure possiamo includere una directory intera es: "/usr/src/myproject".
# Possiamo specificare più input diversi, basta separarli con lo spazio

INPUT                  = main.h main.c

# Se impostiamo RECURSIVE tag a YES allora andremo
# ricorsivamente a considerare tutti i file
# nelle sottocartelle in INPUT.

RECURSIVE              = YES

# GENERATE_HTML se YES allora genere la documentazione in formato HMTL
GENERATE_HTML          = YES

# HTML_OUTPUT tag specifica il percorso della directory
# dove mettere la documentazione HTML.
# Viene accodato al path di OUTPUT, dunque in questo caso
# ./doc/html/

HTML_OUTPUT            = html

# GENERATE_LATEX se YES allora genere la documentazione in formato LATEX
# Ottima se poi vogliamo il formato PDF

GENERATE_LATEX         = YES

# LATEX_OUTPUT tag specifica il percorso della directory dove
# mettere la documentazione latex.
# Viene accodato al path di OUTPUT, dunque in questo caso ./doc/latex/

LATEX_OUTPUT           = latex

# Di default questa impostazione è a NO, ma è utile impostarla a YES
# per ottimizzare l'ouput di certe parti della documentazione, tipo
# quelle riguardanti eventuali strutture create

OPTIMIZE_OUTPUT_FOR_C            = YES
```

Dunque ora abbiamo un progetto con la seguente struttura:

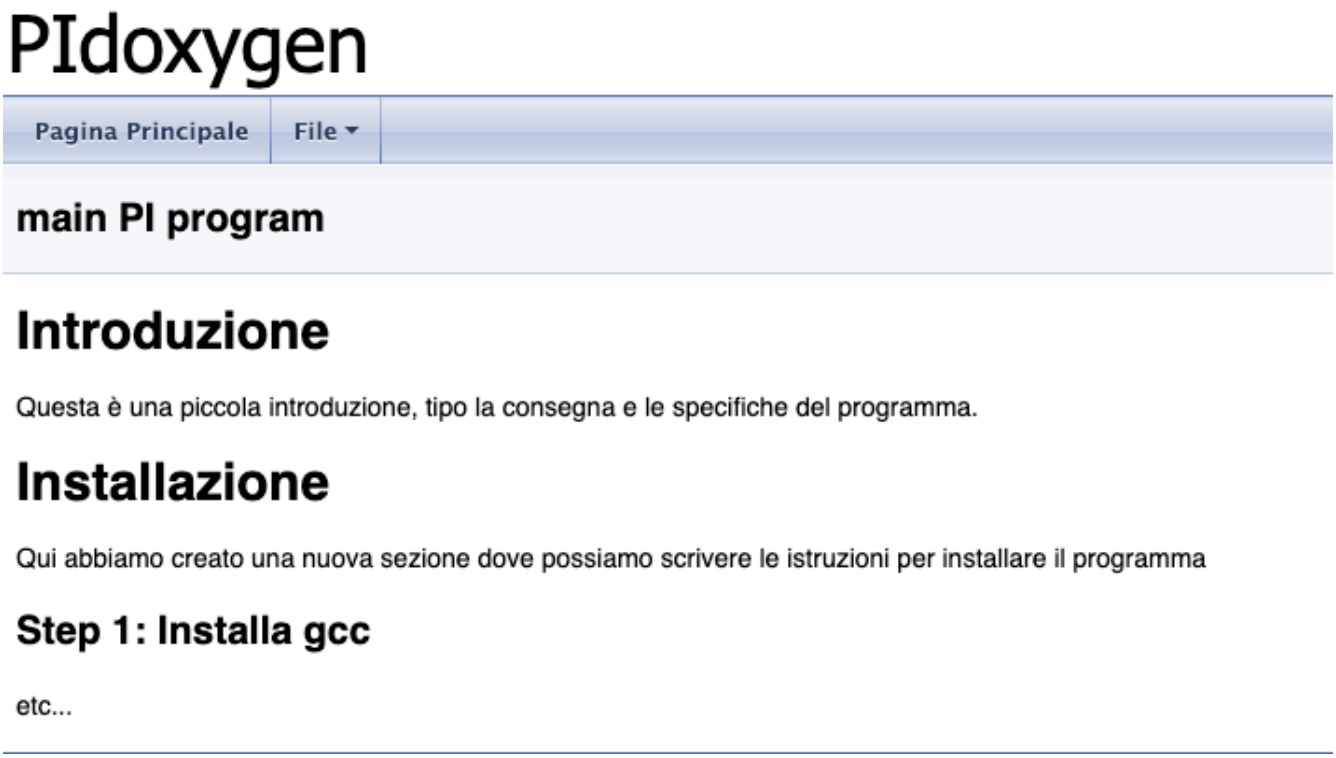
- doc/
- main.c, che contiene le implementazioni delle nostre funzioni e il main

- main.h, contiene la dichiarazione di funzioni e costanti
- Doxyfile

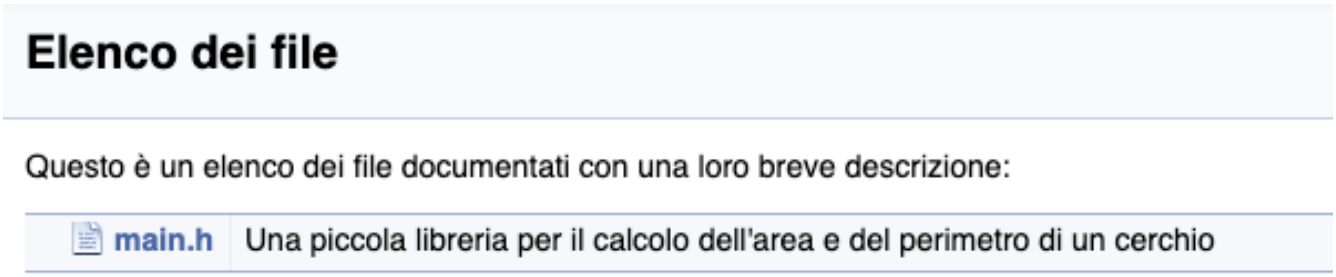
Lanciamo ora il comando:

```
$ doxygen Doxyfile
```

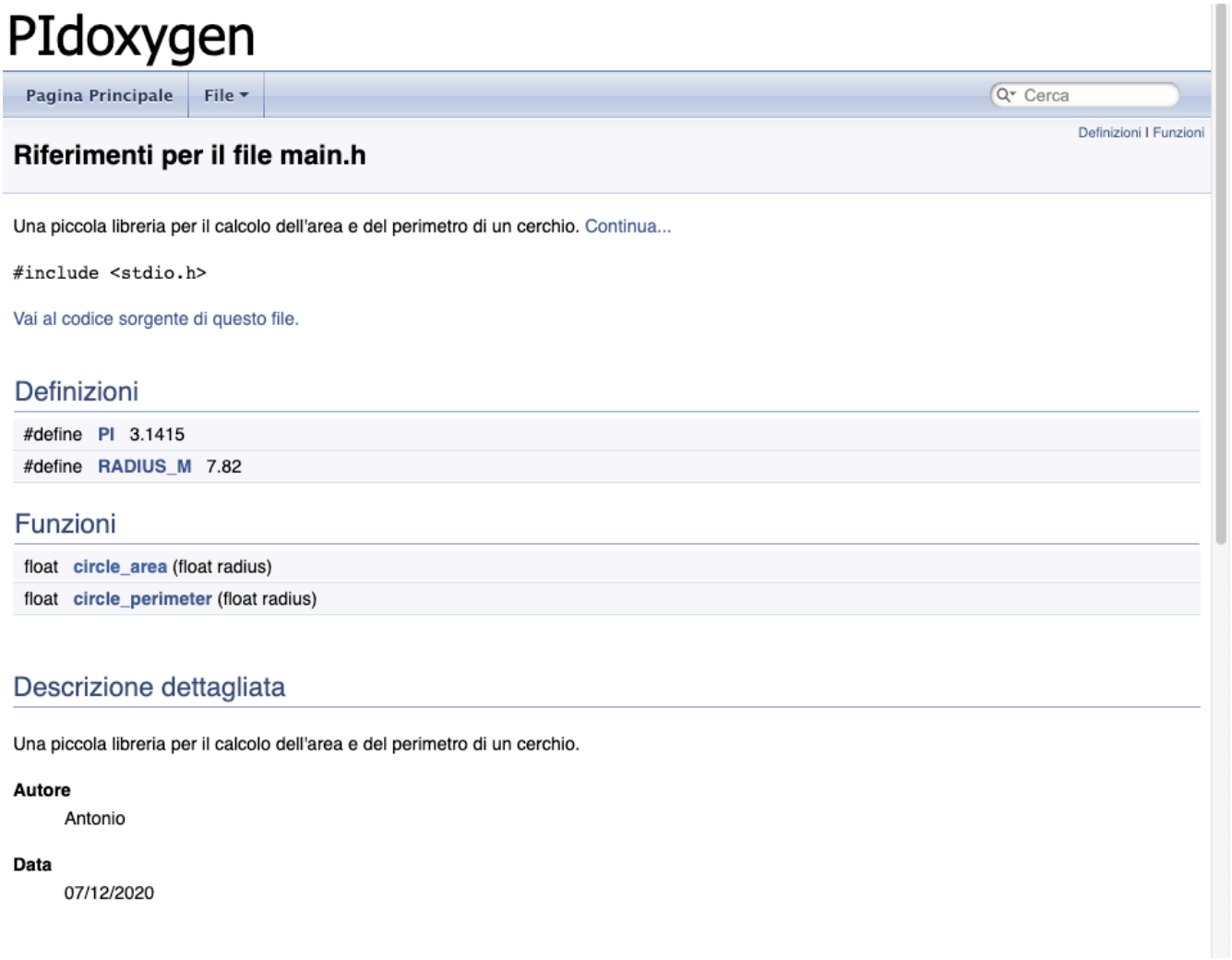
Questo andrà a generare dentro la cartella `doc` due sottocartelle `html` e `latex`. A questo punto la documentazione html è già disponibile aprendo con il browser il file `doc/html/index.html`. Una volta aperto con un browser dovreste vedere questo risultato:



In questa pagina vediamo le informazioni descritte in `main.c`, con l'introduzione e requisiti per l'installazione. Andiamo ora su `File > Elenco dei file`:



Questo contiene la lista dei file rilevati e documentati. In questo caso è presente solamente il `file.h` perché è l'unico file ad avere parti documentate al suo interno. `main.c` non compare perché l'unica parte di documentazione presente è a quella relativa alla descrizione del file, usata nella pagina principale della documentazione grazie al tag `@mainpage`.



Come potete vedere qui abbiamo una documentazione ben formattata, con costanti e funzioni definite separatamente e contenente la descrizione da noi impostata.

8 CLion Release e Debug Mode

All'interno del `CMakeLists.txt` è possibile specificare più di una configurazione per la compilazione del codice sorgente. Solitamente si utilizzano (almeno) due tipi di configurazioni: la configurazione di *Debug* e la configurazione di *Release*. In questa sezione del tutorial vedremo quindi come impostare queste configurazioni e in particolare come impostare il parametro di compilazione `O3` che permette di abilitare le ottimizzazioni di compilazione di livello 3 del nostro compilatore.

8.1 Attivare la modalità di Release in CLion

Solitamente la modalità di Debug è quella attiva di default in Clion, infatti all'interno delle impostazioni, in `Preferences | Build, Execution, Deployment | CMake` troverete una schermata simile a quella in Figura 14.

Per aggiungere la configurazione di release basta semplicemente cliccare sul tasto `+` presente nella pagina. Dopo aver fatto ciò, vedrete che si è aggiunta anche la configurazione release come in Figura 15.

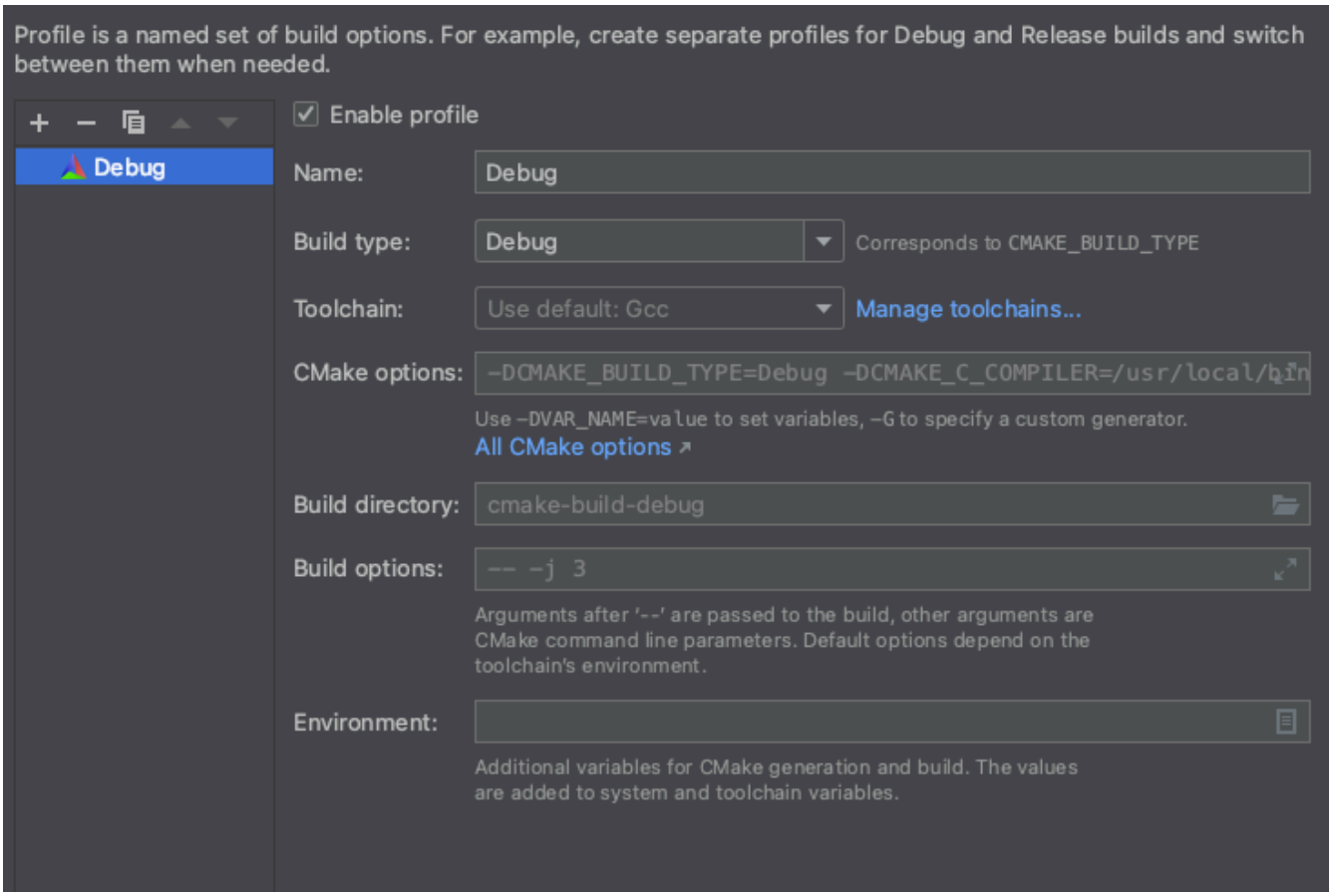


Figura 14: Preferenze debug e release

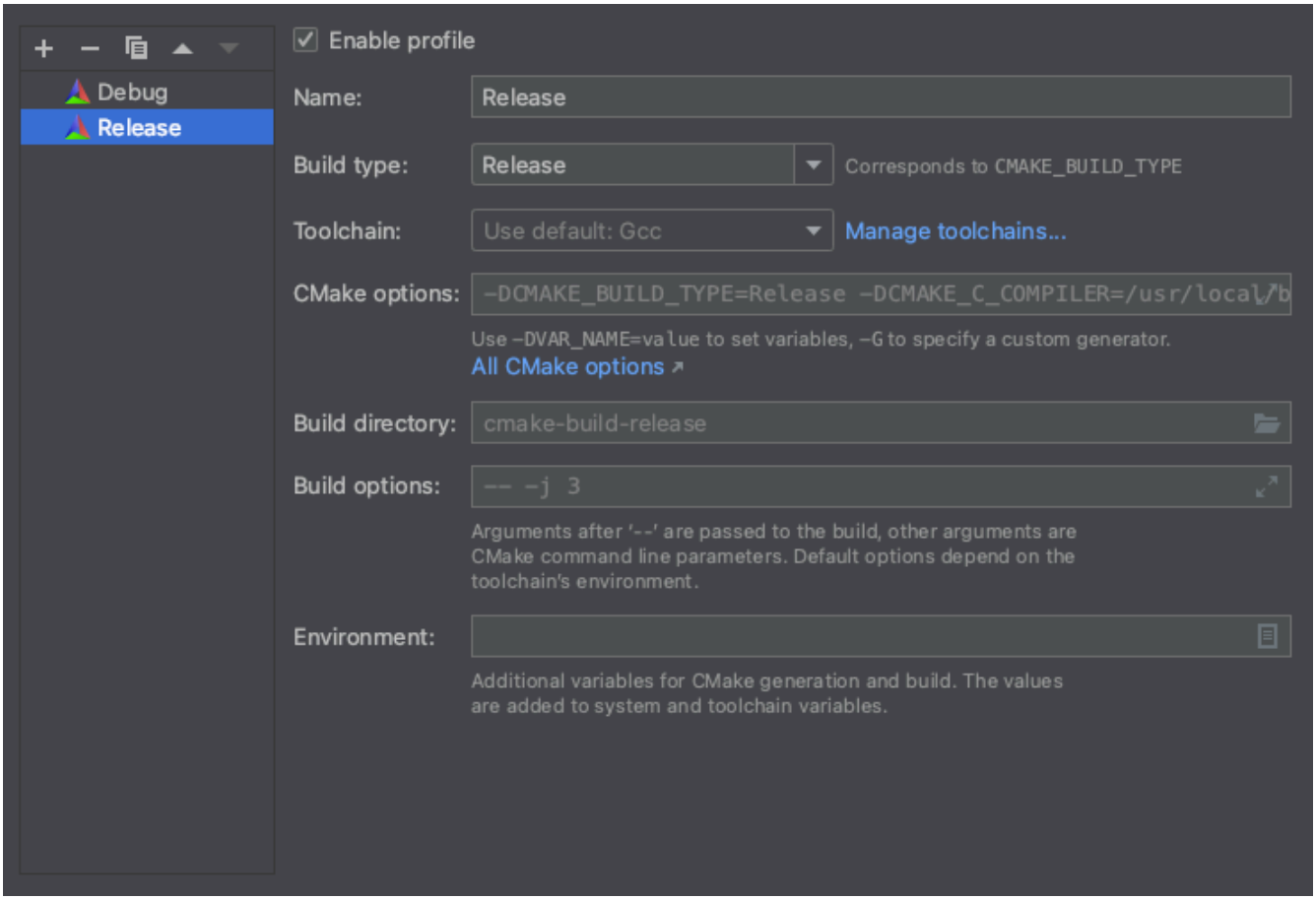


Figura 15: Preferenze debug e release -2

8.2 Attivare flag aggiuntivi in Release mode

Aggiungere un'altra configurazione per la compilazione è utile, ad esempio, per definire flag aggiuntivi durante la compilazione. In questo tutorial vi presentiamo il flag `O3` che ottimizza sia il tempo di compilazione del codice sia le sue performance. Non entriamo nei dettagli delle modifiche che vengono fatte, ma vi rimandiamo a una panoramica dei flag di ottimizzazione in fase di compilazione al seguente link.

Per aggiungere il flag `O3` solo nella modalità `release`, basta aggiungere la seguente istruzione all'interno del file `CMakeLists.txt`

```
set(CMAKE_C_FLAGS_RELEASE "${CMAKE_C_FLAGS} -O3")
```

8.3 Esempio di utilizzo

Dopo aver attivato la modalità release possiamo creare un main di esempio per vedere se tutto funziona correttamente. Vi proponiamo quindi il seguente file `main.c`, all'interno del quale è stata implementata una funzione per il calcolo della somma dei primi `n` numeri naturali. In particolare, nel programma andiamo a misurare i tempi di esecuzione della funzione creando un array con la somma dei primi `n` numeri naturali per `0 <= n <= 100000`.

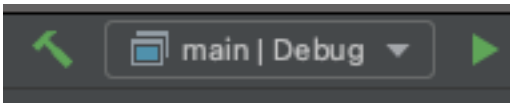
```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/**
 * Somma dei primi n numeri primi
 * @param n
 * @return la somma dei primi n numeri primi
 */
long unsigned sum_n(long unsigned n) {
  long i, res = 0;
  for (i = 1; i <= n; ++i) {
    res += i;
  }
  return res;
}

int main() {

  clock_t t;
  double elpased_ms;
  long unsigned i, end = 100000;
  long unsigned *res = (long unsigned *)malloc(end * sizeof(long unsigned));
  t = clock();
  for (i = 0; i <= end; ++i) {
    res[i] = sum_n(i);
  }
  t = clock() - t;
  elpased_ms = ((double)t) / CLOCKS_PER_SEC;
  printf("The loop took %f seconds to be executed\n", elpased_ms);
  return 0;
}
```

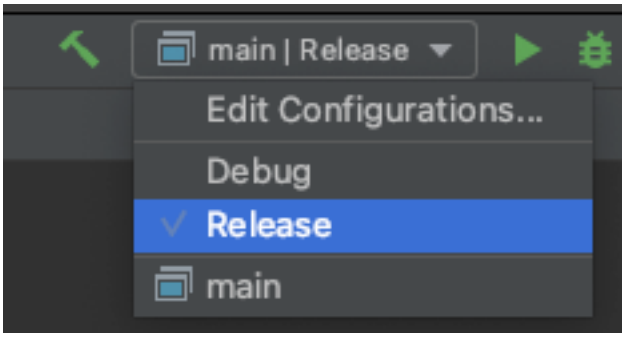
Eseguendo il programma in modalità di debug



avremo il seguente output:

```
The loop took 13.174821 seconds to be executed
```

In modalità release



l'output sarà il seguente:

```
The loop took 1.679597 seconds to be executed
```

Il programma in modalità release è circa 10 volte più veloce!

Le ottimizzazioni fatte dal compilatore possono quindi impattare in maniera significativa sui tempi di esecuzione del programma. Ci domandiamo a questo punto: perché non lasciare sempre attivi i vari flag di ottimizzazione? La risposta sta nel fatto che abilitando i flag di ottimizzazione il più delle volte il debug dell'applicazione diventa difficile, perché il debugger non riesce più a trovare corrispondenza fra il codice sorgente e il codice compilato. Maggiori informazioni sull'argomento le potete trovare anche nella documentazione ufficiale di CLion.