Introduzione a C

Loris Giulivi – Nicolò Folloni

Ambiente di sviluppo

1

Windows

https://www.codeblocks.org/downloads

Microsoft Windows

File	Download from
codeblocks-20.03-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-32bit-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-32bit-nosetup.zip	FossHUB or Sourceforge.net

Debian Linux

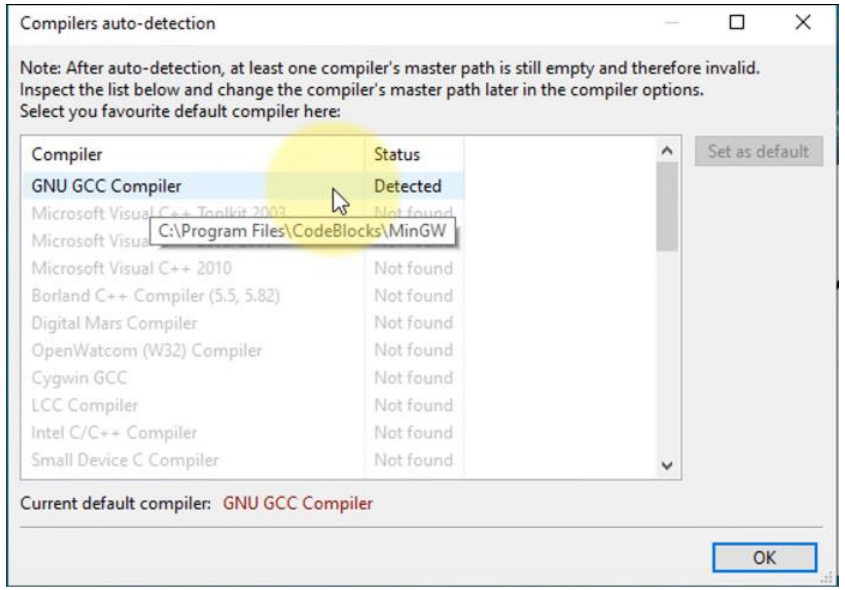
```
apt-get install codeblocks
```

Altri OS

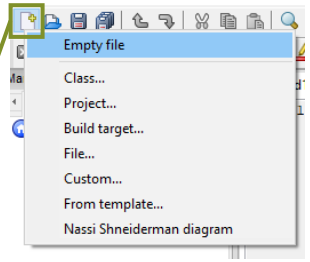
¯_(ツ)_/¯

2

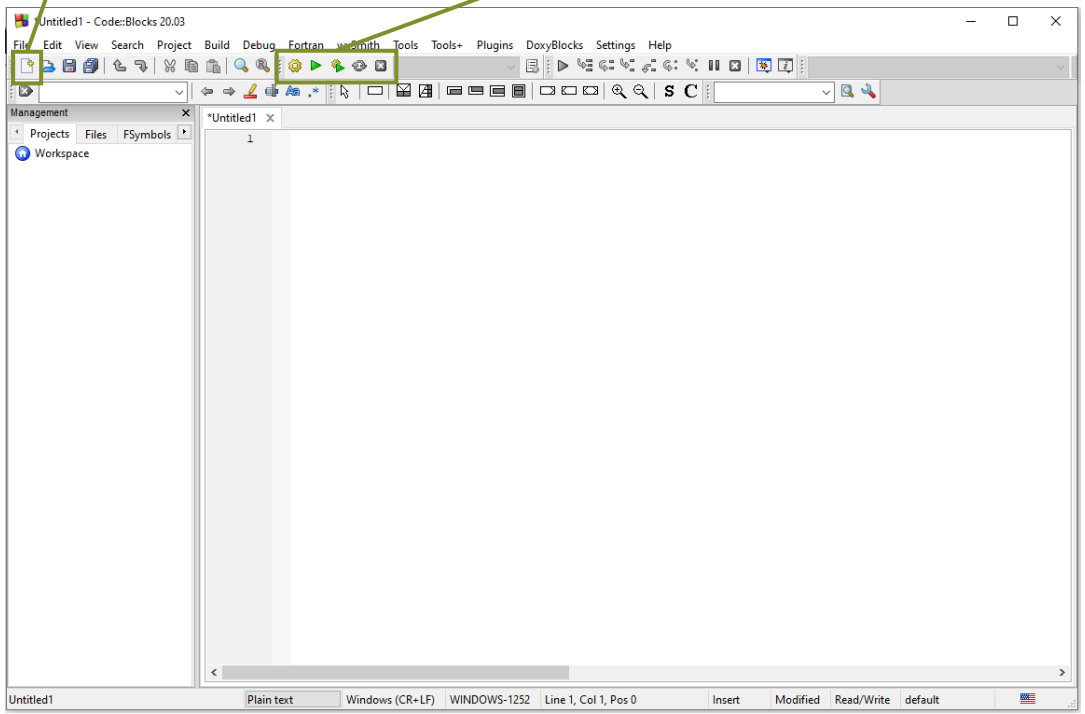
Code::Blocks
The open source, cross-platform IDE

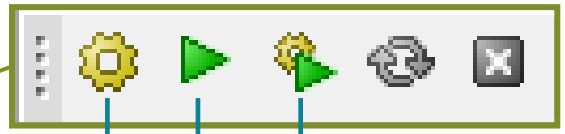


Ambiente di sviluppo



New empty file per creare un file sorgente





Compila ed esegue il file

Esegue senza compilare

⚠ Se avete effettuato modifiche al codice queste non avranno effetto

Compila senza eseguire

Per compilare da CLI:

```
gcc <source.c> -o <destination>
```

Introduzione al linguaggio C

Struttura di un programma C

- Librerie: Includono funzionalità comuni a molti programmi e si possono dunque importare in maniera standard.

- main(): è la funzione di ingresso per ogni programma C. Deve esserci obbligatoriamente. Il codice al suo interno è il punto di partenza per il programma.

- Commenti: le linee precedute da "//" non sono eseguite. È molto importante usare commenti per non perdere traccia di quello che si sta facendo.

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

void main()
{
    //DO SOMETHING
}
```

Commenti

Altri modi per commentare codice

```
printf("hi"); //This is an in-line comment


/*
This is a multi-line comment.
All this text won't be executed.
This goes on until you write:
*/
```

printf

Stampare a schermo - **printf**

La funzione **printf** ci permette di stampare a schermo e comunicare con l'utente.

```
printf("hello world!");
```

Chiamare una funzione

Per chiamare una funzione, la sintassi è:

```
nome_funzione(parametro_1, parametro_2, parametro_n);
```


In C, le istruzioni terminano con un punto e virgola. Dimenticarselo è un errore molto comune, quindi se qualcosa non va, è tra le prime cose da guardare.

Esercizio 0

ES0: Scrivere un programma che stampa a schermo le seguenti informazioni:

`data di oggi`
`nome, cognome, matricola`

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

void main()
{
    ?

}
```

Per andare a capo, si scrive: **`\n`**

Il backslash **`\`** è un carattere di *escape*, ed è usato per indicare i caratteri speciali come **`\n`** (new line).

Per poter scrivere **`\`**, si può scrivere **`\\`**.

Un altro esempio di carattere speciale è il tab, che si scrive con **`\t`**.

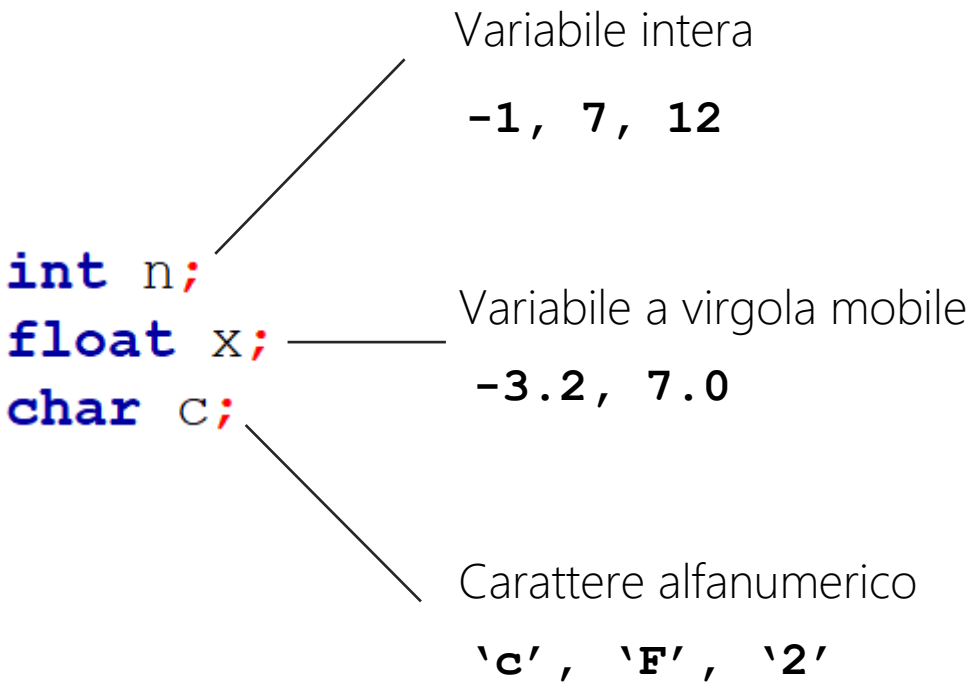
Variabili

Salvare un valore

Le variabili sono contenitori per i nostri dati. Al loro interno possiamo salvare informazione di vario tipo.

Tipi di variabile

In C, le variabili sono *tipizzate*. Ciò significa che il tipo di informazione contenuto nella variabile deve essere specificato durante la definizione della variabile.



Visibilità delle variabili

Scope

Il nome delle variabili le identifica all'interno del *blocco di codice* in cui sono state definite.

Ciò significa che possiamo avere più variabili diverse con lo stesso nome, se ridefinite in diversi blocchi.

Lo *scope* più ampio è quello delle variabili globali.

```
int i=3;

void main()
{
    int i=2;

    //IL CODICE QUI VEDE LA i=2
}
```

```
int VARIABILE_GLOBALE_INTERA;

void main()
{
    char VARIABILE_LOCALE_CARATTERE;

}
```

Stampare variabili con printf (1)

La stringa di formato

La funzione **`printf`** ci permette anche di stampare i valori presenti nelle variabili.

Per farlo, si scrive una *format string*, che contiene delle sequenze speciali che indicano quali valori prendere e dove mostrarli:

`%d` numero intero
`%c` carattere alfanumerico
`%f` numero decimale

```
int i=3;

void main()
{
    printf("The value is: %d", i);
}
```

La sequenza speciale `%d` indica che qui dovrà essere stampato un intero

Il nome della variabile che vogliamo stampare è poi inserito come parametro nella chiamata a `printf`.

Stampare variabili con printf (2)

La stringa di formato

È anche possibile stampare più di un valore nella stessa chiamata a `printf`. In questo caso, aggiungiamo un parametro per ogni valore che vogliamo stampare.

⚠ Le sequenze speciali indicano alla printf come interpretare i dati in memoria, è quindi importante che siano coerenti con il tipo effettivo delle variabili da stampare.

Cosa succede se chiedo alla printf di stampare un char utilizzando %d?

PROVATE VOI!!

```
int i = 3;
float f = 2.4;

printf("The integer is: %d, the float is: %f", i, f);
```

Esercizio 1

ES1: Scrivere un programma che, date due variabili *char*, stampa una parola composta dalle due lettere, ripetute due volte.

Ex. `char_1 = 'a'`
`char_2 = 'b'`
→ `stampa: 'abab'`

```
void main()
{
    char char_1 = 'a'
    char char_2 = 'b'

        ?


}
```

Operatori aritmetici

Espressioni aritmetiche nel codice

Le normali operazioni aritmetiche usano gli operatori classici:

```
+ - * /
```

```
int a = 1;
int b = 2;
int c = a + b + 3;
```

Modulo

L'operatore `%` ritorna il resto della divisione intera tra gli operandi.

```
12%5 = 2
```

Operatori di assegnamento

Operatori speciali che eseguono una operazione aritmetica e assegnano il risultato al primo dei due operandi.

```
+= -=
*= /=
```

```
a = 2;
b = 5;

a += b;

// a = 7
```

Syntactic sugar

```
a += 1;
a -= 1;
```

↔

```
a++;
a--;
```

Operazioni su interi

Implicit casting

Se il tipo degli operandi e del risultato non sono compatibili, questi vengono modificati, se possibile, per permettere l'operazione.

⚠ La conversione da **`float`** a **`int`** viene sempre effettuata troncando la parte decimale, e non per arrotondamento all'intero più vicino.

```
float a = 2.7;
int b = 1;

int c = a + b;

// c = 3
```

Ricevere input dall'utente

Ricevere input dall'utente - **scanf**

La funzione **scanf** ci permette di ricevere dati dall'utente e salvarli nelle variabili.

La stringa di formato

Come per la **printf**, una stringa di formato viene usata per determinare come ricevere i dati.

Il puntatore alla variabile

L'operatore unario **&** ritorna l'indirizzo di memoria della variabile operando.

Per indicare alla funzione dove salvare l'input dell'utente, durante la chiamata alla **scanf**, precediamo il nome della variabile da questo operatore.

&a indica che vogliamo salvare il valore nella cella di memoria dove è situata la variabile **a**.

```
int a;

scanf("%d", &a);
```

La scanf può salvare anche più di un valore alla volta, ma tipicamente questo non viene fatto, perché richiede che l'utente rispetti il formato specificato nella stringa di formato.

```
int a;
float b;

scanf("%d-%f", &a, &b);
```

Nell'esempio, l'utente dovrebbe inserire una stringa del tipo:

```
5-2.3
```

Esercizio 2

ES2: Scrivere un programma che chiede all'utente due numeri interi e che ne stampi la somma.

Esercizio 3

ES3: Scrivere un programma che chiede in input all'utente la prima lettera del proprio nome e la prima lettera del proprio cognome, e che le restituisce nell'ordine inverso (prima cognome poi nome).