Strutture di Controllo in C

Informatica B AA 2017 / 2018

Giacomo Boracchi

27 Settembre 2017

giacomo.boracchi@polimi.it

NOTE PER I LABORATORI

Per il prossimo lab,

- scaricate sempre gli esercizi dal sito del laboratorio (link nella pagina del corso)
- Installate Code::Blocks sul vostro laptop (o presentatevi in anticipo per ritirare un laptop del poli)
- Se disponete di un cavo di rete, portatelo (non più indispensabile, hanno installato il wifi nelle aule del lab)

CodeBlocks Installation (Win e Mac OS)

Installazione Code::Blocks

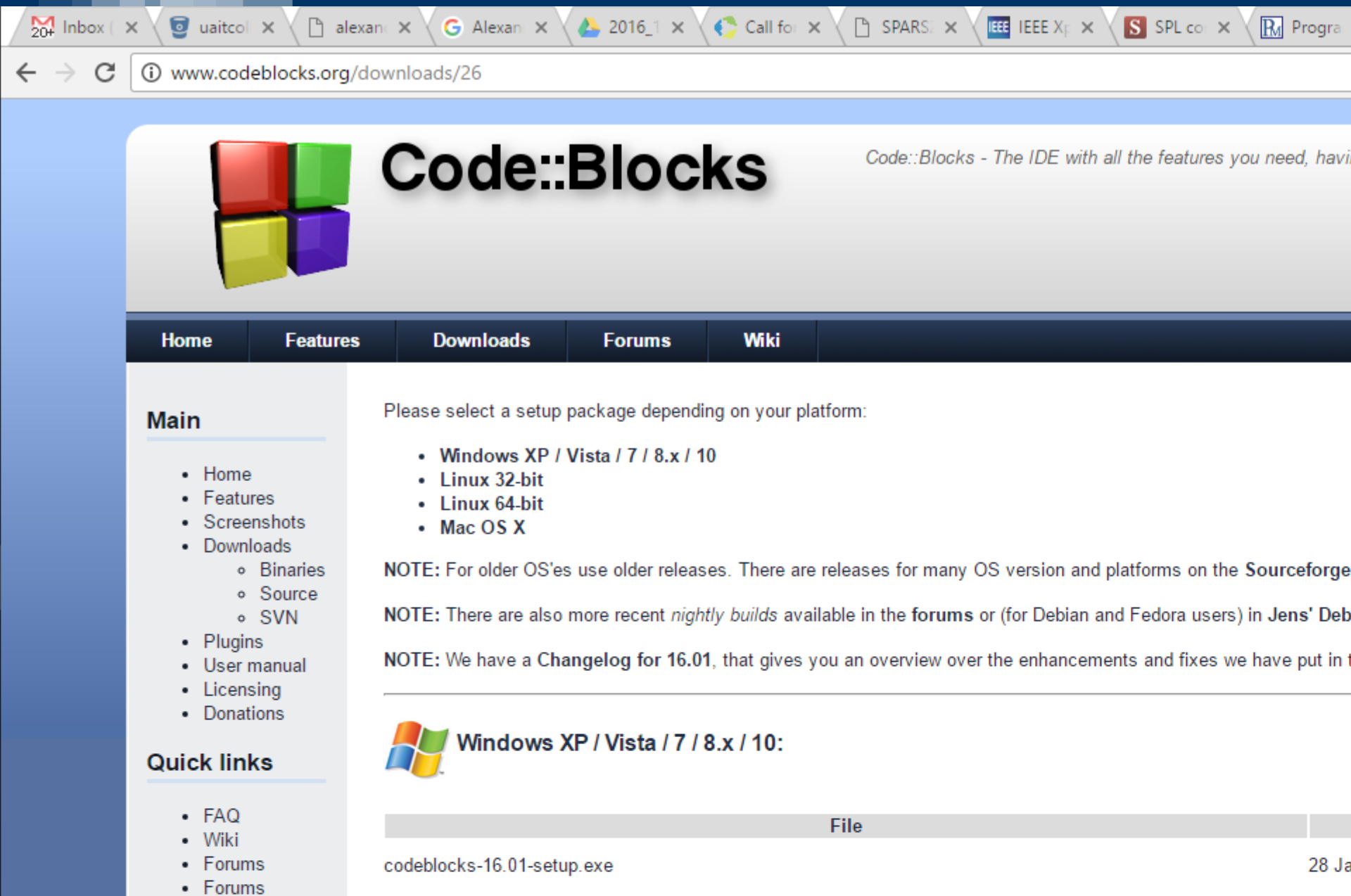
- Chi dispone di un PC provi ad installare Code::Blocks autonomamente.
- Il sito del laboratorio è http://home.deib.polimi.it/santambr/dida/infob/1718/labgb/

Code::Blocks


www.codeblocks.org
Code::Blocks
Code::Blocks - The IDE with all the features you need, havi
Home
Features
Downloads
Forums
Wiki
Main
Home
Features
Screenshots
Downloads
Plugins
User manual
Licensing
Donations
Quick links
FAQ
Wiki
Forums
Forums (mobile)
Nightlies
Ticket
The open source, cross platform, free C, C++ and Fortra
Code::Blocks is a free C, C++ and Fortran IDE built to meet the most demanding needs of its users. It is designed to be
Finally, an IDE with all the features you need, having a consistent look, feel and operation across platforms.
Built around a plugin framework, Code::Blocks can be extended with plugins. Any kind of functionality can be added by i
functionality is already provided by plugins!
Special credits go to darmar for his great work on the FortranProject plugin, bundled since release 13.12.
We hope you enjoy using Code::Blocks!
The Code::Blocks Team
Code::Blocks 16.01 is here!

Download the binary release

Scegliete il Vostro OS



Per Windows, scaricare il pacchetto Setup con Mingw

Sono circa 100MB e contiene sia l'IDE (Integrated Development Environment) che il compilatore

Cliccare sul repository da cui scaricare (e.g. Sourceforge.net)

Windows XP / Vista / 7 / 8.x / 10:

File	Date	
codeblocks-16.01-setup.exe	28 Jan 2016	**Sourceforge.net** or
codeblocks-16.01-setup-nonadmin.exe	28 Jan 2016	**Sourceforge.net** or
codeblocks-16.01-nosetup.zip	28 Jan 2016	**Sourceforge.net** or
codeblocks-16.01mingw-setup.exe	28 Jan 2016	**Sourceforge.net** or
codeblocks-16.01mingw-nosetup.zip	28 Jan 2016	**Sourceforge.net** or
codeblocks-16.01mingw_fortran-setup.exe	28 Jan 2016	**Sourceforge.net** or

NOTE: The codeblocks-16.01-setup.exe file includes Code::Blocks with all plugins. The codeblocks-16.01-setup-nonadmin.exe file is provided for convenience to users

NOTE: The codeblocks-16.01mingw-setup.exe file includes *additionally* the GCC/G++ compiler and GDB debugger from **TDM**-GCC (version 4.9.2, 32 bit, SJLJ). The co
the GFortran compiler (**TDM**-GCC).

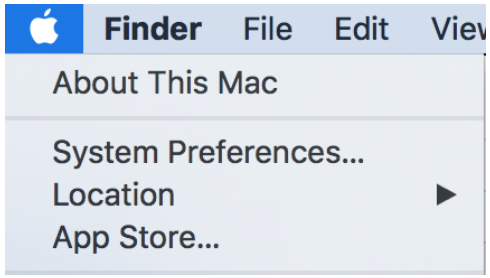
NOTE: The codeblocks-16.01(mingw)-nosetup.zip files are provided for convenience to users that are allergic against installers. However, it will not allow to select plugi
shortcuts. For the "installation" you are on your own.

If unsure, please use codeblocks-16.01mingw-setup.exe!

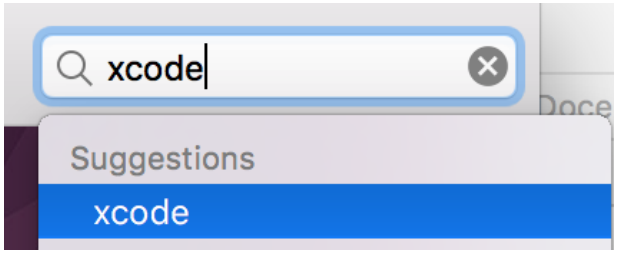
Installazione Xcode su Mac OS (Credits Ing. Conti)

Prima di installare Code::Blocks è necessario installare un prerequisito: Xcode

Prerequisiti: Installare Xcode



andare su AppStore



e scaricare Xcode

Prerequisiti: Installare Xcode (cnt)

se non si vuole scaricare Xcode, su puo' in alternativa scaricare da sito developer di Apple:

Command Line Tools (macOS 10.12) for Xcode 8.dmg

Download CodeBlocks

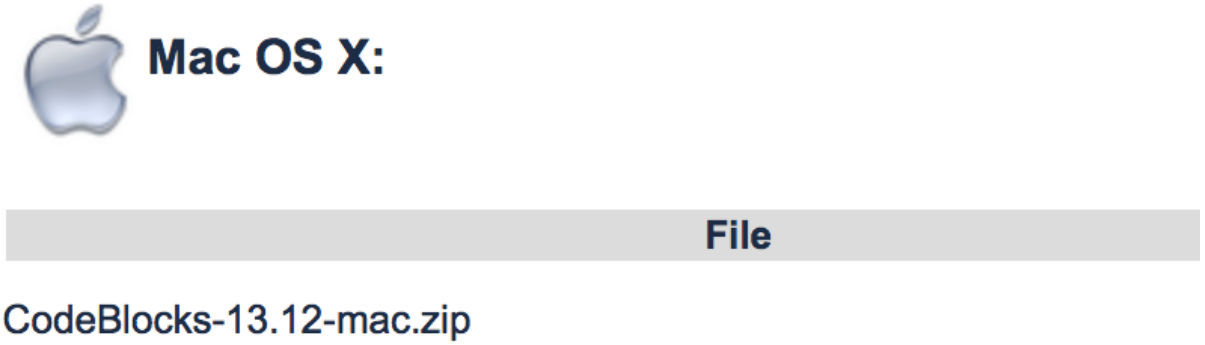
andare al sito:

http://www.codeblocks.org/downloads

cliccare su: “**Download the binary release**” (percorso illustrato nelle slides precedenti)

Scaricate la versione per Mac OS

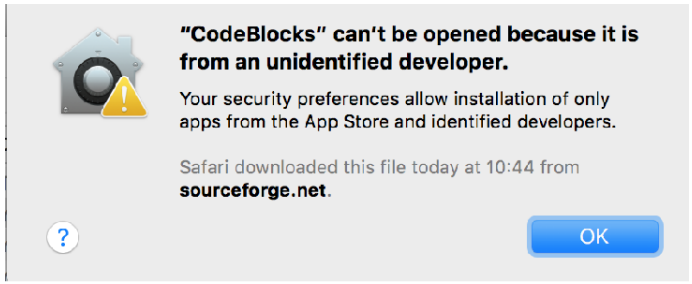
(dovrebbe essere:
http://www.codeblocks.org/downloads/26 **)**



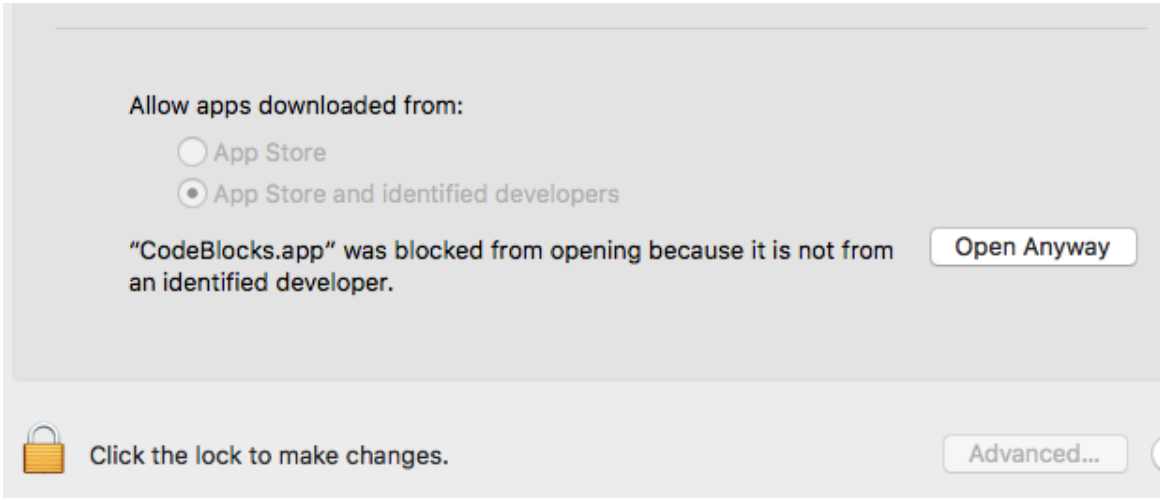
scaricare lo Zip

scompattare lo zip con doppio click.

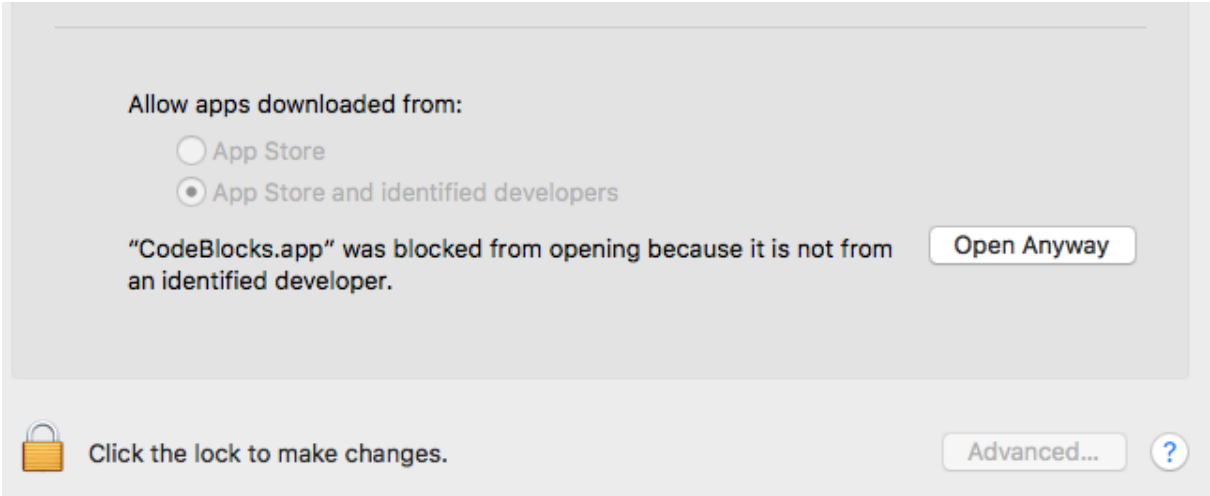
poiché l'eseguibile non e' firmato digitalmente, apparira:



andare nell prefs di MacOs sezione pannello Security ed abilitarlo.



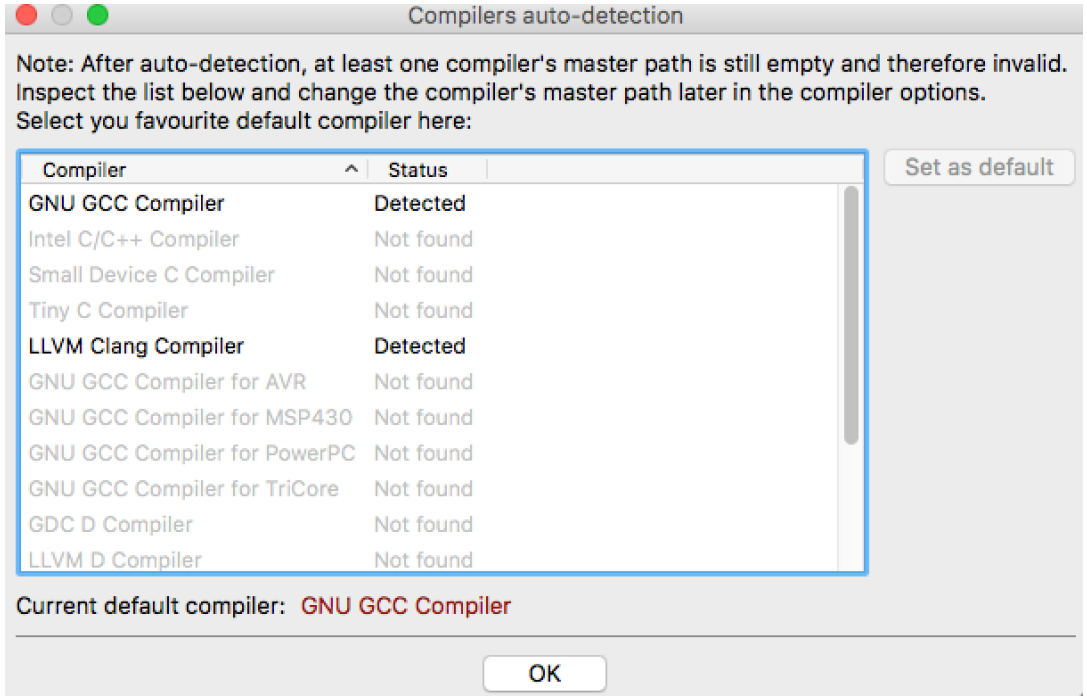
ed abilitare codeBlocks:



"Open anyway"

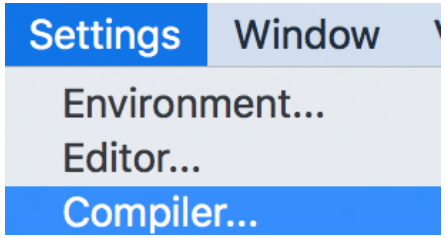
Da ora sara' abilitato anche se venisse cancellato e riscaricato.

aprire CodeBlock, dovrebbe apparire:

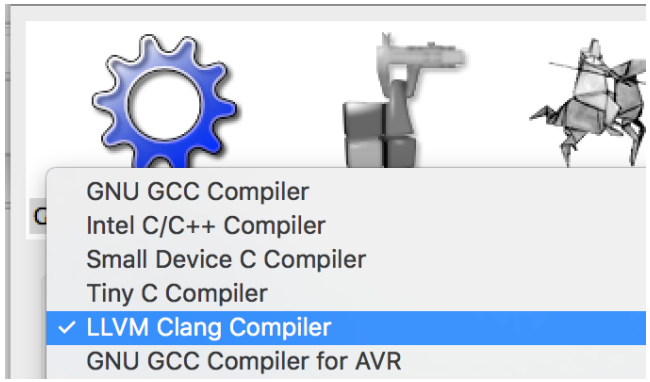


selezionate LLVM CLang Compiler.

Se non appare, andare nel menu:

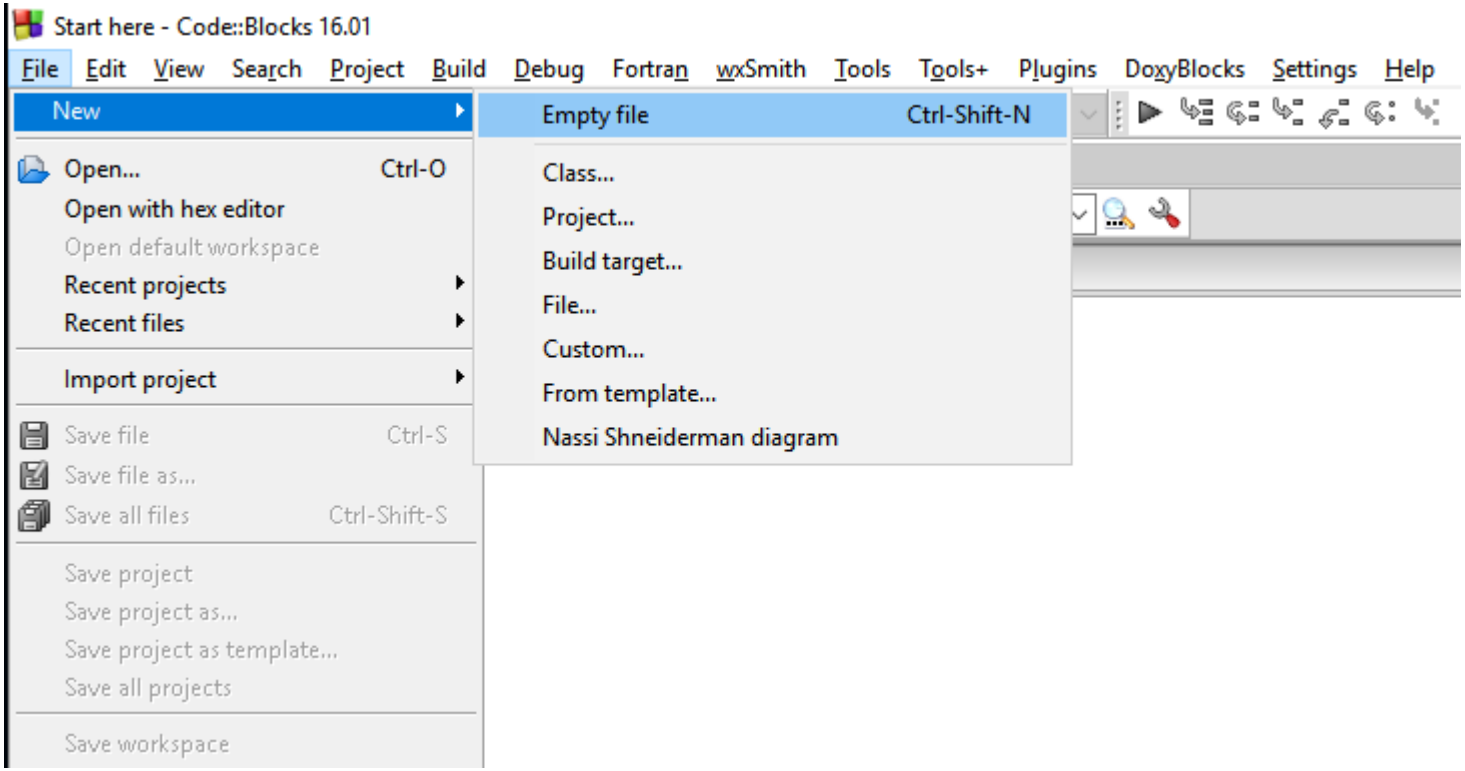


ed impostare:



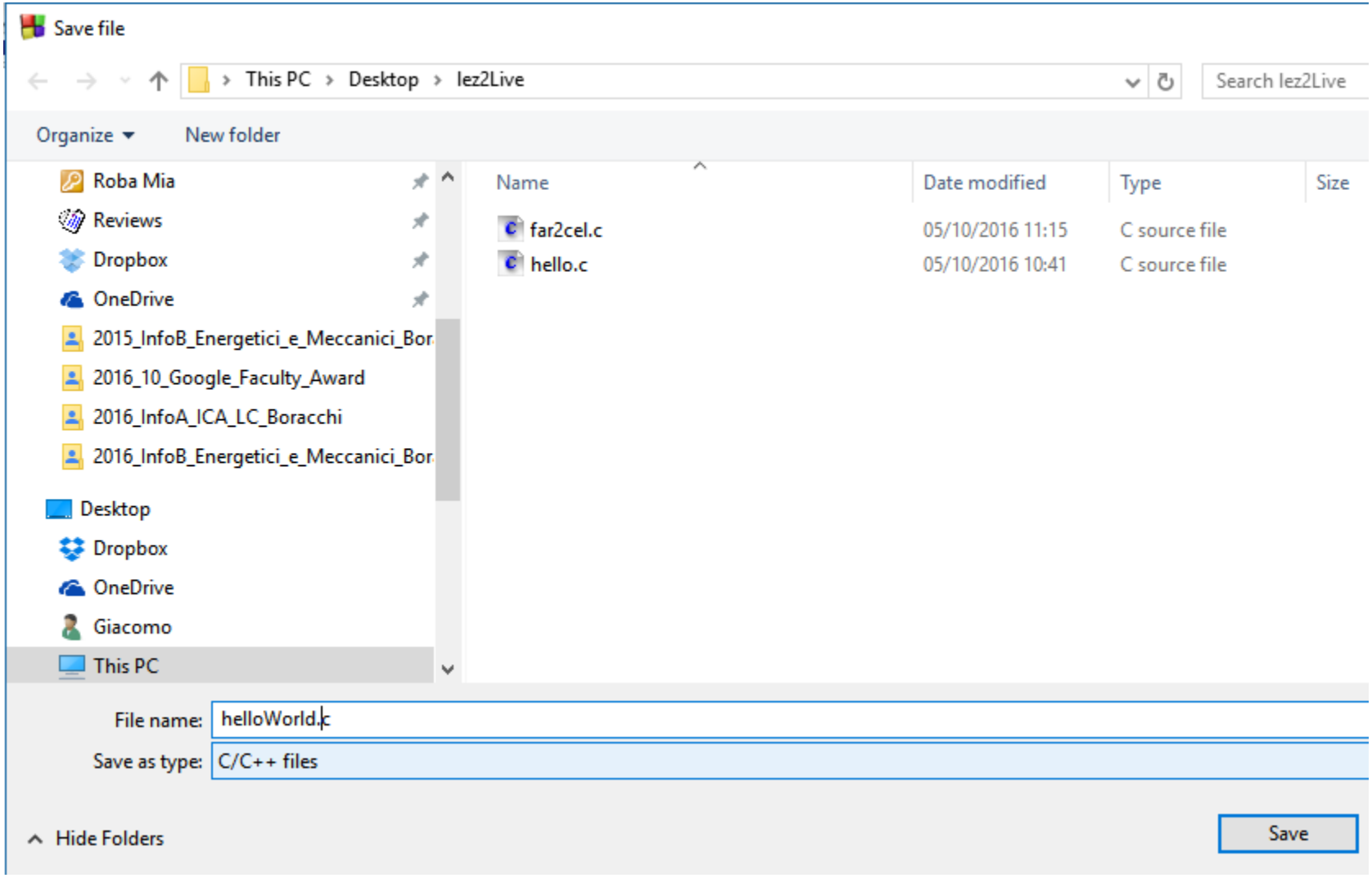
Per Iniziare (sia su Windows che su Mac OS)

Selezionare «empty file» dal menu File->New



Per Iniziare (sia su Windows che su Mac OS)

Salvare il file con estensione .c e tipo C/C++ files



Operatori ed Espressioni Logiche

- Algebra di Boole

Non solo operazioni aritmetiche

- **Espressione booleana**: espressione con valore **vero (1) o falso (0)**, determinata dagli **operatori e dal valore delle costanti o variabili** in essa contenute.
- In C abbiamo:
  - **operatori relazionali**: si applicano a **variabili, costanti** o espressioni e sono `==`, `!=`, `>`, `<`, `>=`, `<=`

    Es: `(a > 7)` , `(b % 2 == 0)`, `(x <= w)`

    danno luogo ad **espressioni Booleane**.
  - **operatori logici**: applicati a **espressioni Booleane**, permettono di costruire **espressioni composte** e sono `!`, `&&`, `||`

    Es: `(a > 7) && (b % 2 == 0)`

    `!(x >= 7) || (a == 0)`

Operazioni built-in per dati di tipo `int`

- `=` Assegnamento di un valore `int` a una variabile `int`
- `+` Somma (tra `int` ha come risultato un `int`)
- `-` Sottrazione (tra `int` ha come risultato un `int`)
- `*` Moltiplicazione (tra `int` ha come risultato un `int`)
- `/` Divisione con troncamento della parte non intera (risultato `int`)
- `%` Resto della divisione intera
- `==` Relazione di uguaglianza
- `!=` Relazione di diversità
- `<` Relazione “minore di”
- `>` Relazione “maggiore di”
- `<=` Relazione “minore o uguale a”
- `>=` Relazione “maggiore o uguale a”

Operatoori Aritmetici

Operatoori Relazionali

Operazioni built-in per dati di tipo `int`

- `=` Assegnamento di un valore `int` a una variabile `int`
- `+` Somma (tra `int` ha come risultato un `int`)
- `-` Sottrazione (tra `int` ha come risultato un `int`)
- `*` Moltiplicazione (tra `int` ha come risultato un `int`)
- `/` Divisione con troncamento della parte non intera (risultato `int`)
- `%` Resto della divisione intera
- `==` Relazione di uguaglianza
- `!=` Relazione di diversità
- `<` Relazione “minore di”
- `>` Relazione “maggiore di”
- `<=` Relazione “minore o uguale a”
- `>=` Relazione “maggiore o uguale a”

Operatoori Aritmetici

Operatoori Relazionali

Non solo operazioni aritmetiche

- **Espressione booleana**: espressione con valore **vero (1) o falso (0)**, determinata dagli **operatori e dal valore delle costanti o variabili** in essa contenute.
- In C abbiamo:
  - **operatori relazionali**: si applicano a **variabili, costanti** o espressioni e sono `==`, `!=`, `>`, `<`, `>=`, `<=`

    Es: `(a > 7)` , `(b % 2 == 0)`, `(x <= w)`

    danno luogo ad **espressioni Booleane**.
  - **operatori logici**: applicati a **espressioni Booleane**, permettono di costruire **espressioni composte** e sono `!`, `&&`, `||`

    Es: `(a > 7) && (b % 2 == 0)`

    `!(x >= 7) || (a == 0)`

Aritmetica Operatori Logici

Ordine Operatori Logici in assenza di parentesi (elementi a priorità maggiore in alto):

1. negazione (NOT) `!`
2. operatori di relazione `<`, `>`, `<=`, `>=`
3. uguaglianza `==`, disuguaglianza `!=`,
4. congiunzione (AND) `&&`
5. disgiunzione (OR) `||`

Esempi

- `x > 0 || y == 3 && !z`
- `(x > 0) || ((y == 3) && (!z))`

Aritmetica degli Operatori Logici

- Gli operatori `&&` e `||` sono commutativi
  - `(a && b) == (b && a)`
  - `(a || b) == (b || a)`
- Le doppie negazioni si elidono: `!!a == a`

Come Funzionano gli Operatori Logici?

- Ogni espressione booleana può assumere solo due valori
- Posso quindi considerare tutti i possibili valori degli ingress ad un'espressione booleana e calcolare i valori di output corispondenti
- Questo corrisponde alla tabella di verità
- Incominciamo a farla per definire gli operatori `!`, `&&` e `||`

Tavole di Verità degli Operatori Logici

- Le tabelle di verità stabiliscono i valori di predicati composti

- Il NOT è un operatore **unario**, che prende in ingresso **una** sola espressione.

- `!A` è l'opposto di `A`

Tavole di Verità degli Operatori Logici

- Le tabelle di verità stabiliscono i valori di predicati composti
- Il NOT è un operatore **unario**, che prende in ingresso **una** sola espressione.
- **`!A`** è l'opposto di **`A`**

negazione (NOT)	
A	***!A***
0	1
1	0

Tavole di Verità degli Operatori Logici

- Le tabelle di verità stabiliscono i valori di predicati composti

- L'operatore AND è **binario**, prende in ingresso **due** espressioni.

- `A && B` è vero se e solo se sia `A` che `B` sono vere.

Tavole di Verità degli Operatori Logici

- Le tabelle di verità stabiliscono i valori di predicati composti
- L'operatore AND è **binario**, prende in ingresso **due** espressioni.
- **`A && B`** è vero se e solo se sia **`A`** che **`B`** sono vere.

congiunzione (AND)		
A	**B**	**A && B**
0	0	0
0	1	0
1	0	0
1	1	1

Tavole di Verità degli Operatori Logici

- Le tabelle di verità stabiliscono i valori di predicati composti
- L'operatore AND è **binario**, prende in ingresso **due** espressioni.
- **`A && B`** è vero se e solo se sia **`A`** che **`B`** sono vere.

congiunzione (AND)		
A	**B**	**A && B**
0	0	0
0	1	0
1	0	0
1	1	1

Tavole di Verità degli Operatori Logici

- Le tabelle di verità stabiliscono i valori di predicati composti
- L'operatore OR è **binario**, prende in ingresso **due** espressioni.
- **`A || B`** è vero se almeno una delle due è vera.

disgiunzione (OR)		
`A`	**`B`**	**`A || B`**
`0`	`0`	`0`
`0`	`1`	`1`
`1`	`0`	`1`
`1`	`1`	`1`

Tavole di Verità degli Operatori Logici

- Le tabelle di verità stabiliscono i valori di predicati composti
- L'operatore OR è **binario**, prende in ingresso **due** espressioni.
- **`A || B`** è vero se almeno una delle due è vera.
- **NB:** non è un OR esclusivo, come spesso accade nella lingua parlata

disgiunzione (OR)		
`A`	**`B`**	**`A || B`**
`0`	`0`	`0`
`0`	`1`	`1`
`1`	`0`	`1`
`1`	`1`	`1`

Tabelle di Verità

- Rappresenta tutti i possibili modi di valutare un' espressione booleana composta
- Una riga per ogni possibile assegnamento di valori logici alle variabili:
  - n variabili logiche (espressioni booleane) $\rightarrow 2^n$ possibili assegnamenti, quindi 2^n righe.
- Una colonna per ogni espressione che compone l'espressione data (inclusa la formula stessa)

Esempio Tabella di Verità

- `A && !B || C`

Esempio Tabella di Verità

- `A && !B || C`

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Esempio Tabella di Verità

- `A && !B || C`

A	B	C	!B	A && !B	A && !B || C
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Esempio Tabella di Verità

- `A && !B || C`

A	B	C	!B	A && !B	A && !B || C
0	0	0	1		
0	0	1	1		
0	1	0	0		
0	1	1	0		
1	0	0	1		
1	0	1	1		
1	1	0	0		
1	1	1	0		

Esempio Tabella di Verità

- `A && !B || C`

A	B	C	!B	A && !B	A && !B || C
0	0	0	1	0	
0	0	1	1	0	
0	1	0	0	0	
0	1	1	0	0	
1	0	0	1	1	
1	0	1	1	1	
1	1	0	0	0	
1	1	1	0	0	

Esempio Tabella di Verità

- `A && !B || C`

A	B	C	!B	A && !B	A && !B || C
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	1	0	0	1

Altro Esempio di Tabella di Verità

- `A && (!B || C)`

Altro Esempio di Tabella di Verità

- `A && (!B || C)`

A	B	C	!B	!B || C	A && (!B||C)
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Operatori Logici: Leggi di de Morgan

- Leggi di De Morgan: illustrano come distribuire la negazione rispetto a `||` e `&&`
  1. `!(a && b) == !a || !b`
  2. `!(a || b) == !a && !b`

- Es: `!((a >= 5) && (a <= 10))` -> [De Morgan]

  `!(a >= 5)||!(a <= 10)` -> [proprietà `>=` e `<=`]

  `!!(a < 5)|| !!(a > 10)` -> [doppia negazione]

  `((a < 5) || (a > 10))`

Esempio

- Dimostrare che le seguenti espressioni sono equivalenti
  - `A || C && !B`
  - `!((B || !C) && !A)`

- Due possibili soluzioni:
  - Applicando le leggi di De Morgan cerco di passare da una all'altra
  - Calcolo entrambe le tabella di verità e mostro che coincidono

Esempio

- Dimostrare che le seguenti espressioni sono equivalenti
  - `A || C && !B`
  - `!((B || !C) && !A)`

- `!((B || !C) && !A)`

Esempio

- Dimostrare che le seguenti espressioni sono equivalenti
  - `A || C && !B`
  - `!((B || !C) && !A)`

- `!((B || !C) && !A)`
- `(!(B || !C) || !!A)`

Esempio

- Dimostrare che le seguenti espressioni sono equivalenti
  - `A || C && !B`
  - `!((B || !C) && !A)`

- `!((B || !C) && !A)`
- `(!(B || !C) || !!A)`
- `!(B || !C) || A`

Esempio

- Dimostrare che le seguenti espressioni sono equivalenti
  - `A || C && !B`
  - `!((B || !C) && !A)`

- `!((B || !C) && !A)`
- `(!(B || !C) || !!A)`
- `!(B || !C) || A`
- `(!B && C) || A`

Esempio

- Dimostrare che le seguenti espressioni sono equivalenti
  - `A || C && !B`
  - `!((B || !C) && !A)`

- `!((B || !C) && !A)`
- `(!(B || !C) || !!A)`
- `!(B || !C) || A`
- `(!B && C) || A`
- `A || (!B && C)`

Esempio

- Dimostrare che le seguenti espressioni sono equivalenti
  - `A || C && !B`
  - `!((B || !C) && !A)`

- `!((B || !C) && !A)`
- `(!(B || !C) || !!A)`
- `!(B || !C) || A`
- `(!B && C) || A`
- `A || (!B && C)`
- `A || (C && !B)`

Esempio

- Dimostrare che le seguenti espressioni sono equivalenti
  - `A || C && !B`
  - `!((B || !C) && !A)`

- `!((B || !C) && !A)`
- `(!(B || !C) || !!A)`
- `!(B || !C) || A`
- `(!B && C) || A`
- `A || (!B && C)`
- `A || (C && !B)`
- `A || C && !B`

Espressioni Booleane in C

- Servono per definire condizioni che vengono impiegate in istruzioni composte:
  - Costrutti condizionali: **`if`**, **`switch`**
  - Costrutti iterativi: **`while`**, **`do while`**, **`for`**

Espressioni Intere come Booleane in C

- **Espressioni intere e booleane** sono **intercambiabili**: esiste una regola di conversione automatica
  - **0 ⇔ falso**
  - **qualsiasi valore `!=0` ⇔ vero**
- Ciò utilizzato in pratica (anche se non bello dal punto di vista concettuale) per
  - **memorizzare in variabili intere risultati di condizioni** (valori di espressioni logiche, non esistono del resto variabili di un apposito tipo in C)
  - **utilizzare espressioni aritmetiche al posto di condizioni** nelle istruzioni **`if`** e **`while`**

Operatori Logici

Esempio: Mondiali 2014 – Gruppo D

	Squadra	Punti	Differenza reti	Reti segnate
	Costa Rica	6	+3	4
	Italia	3	0	2
	Uruguay	3	-1	3
	Inghilterra	0	-2	2

Criteri di passaggio del turno: passano le prime due squadre con più:

- punti
- maggiore differenza reti
- maggiori reti segnate

Credits: Francesco Trovò

Passa l'italia se:

L'italia passa solo se pareggia o vince con un qualunque risultato

Dobbiamo controllare i predicati:

VI: «vince l'Italia»
PI: «pareggia l'Italia»

Passa l'Italia		
VI	PI	VI || PI
0	0	0
0	1	1
1	0	1
1	1	1

Passa l'italia se:

L'italia passa solo se pareggia o vince con un qualunque risultato

Dobbiamo controllare i predicati:

VI: «vince l'Italia»
PI: «pareggia l'Italia»

Passa l'Italia		
VI	PI	VI || PI
0	0	0
0	1	1
1	0	1
~~1~~	~~1~~	~~1~~

ATTENZIONE!!!
I due eventi sono mutualmente esclusivi. Non si possono verificare contemporaneamente!!!

L'Italia passa come prima se:

L'italia passa come prima se vince, la Costa rica perde ed è verificata una delle seguenti:

- la sua differenza reti è maggiore della Costa Rica
- la sua differenza reti è uguale a quella della Costa Rica e ha segnato più reti della Costa rica

Dobbiamo controllare i predicati:

- VI: «vince l'Italia»
- CRP: «perde la Costa Rica»
- DRM: «l'Italia ha una differenza reti maggiore della Costa Rica»
- DRU: «l'Italia ha una differenza reti uguale alla Costa Rica»
- GSM: «l'Italia ha una segnato più reti della Costa Rica»

L'Italia passa come prima se:

L'italia passa come prima se vince, la Costa rica perde ed è verificata una delle seguenti:

- la sua differenza reti è maggiore della Costa Rica
- la sua differenza reti è uguale a quella della Costa Rica e ha segnato più reti della Costa rica

Dobbiamo controllare i predicati:

- VI: «vince l'Italia»
- CRP: «perde la Costa Rica»
- DRM: «l'Italia ha una differenza reti maggiore della Costa Rica»
- DRU: «l'Italia ha una differenza reti uguale alla Costa Rica»
- GSM: «l'Italia ha una segnato più reti della Costa Rica»

```
VI && CRP && ( DRM || (DRU && GSM))
```

La Tavola di Verità

VI	CRP	DRM	DRU	GSM	DRU && GSM	DRM ||(DRU && GSM)	VI && CRP && (DRM||(DRU && GSM))
1	1	1	1	1			
1	1	1	1	0			
1	1	1	0	1			
1	1	1	0	0			
1	1	0	1	1			
1	1	0	1	0			
1	1	0	0	1			
1	1	0	0	0			
1	0	1	1	1			
1	0	1	1	0			
1	0	1	0	1			
1	0	1	0	0			
1	0	0	1	1			
1	0	0	1	0			
1	0	0	0	1			
1	0	0	0	0			

La Tavola di Verità

VI	CRP	DRM	DRU	GSM	DRU && GSM	DRM ||(DRU && GSM)	VI && CRP && (DRM||(DRU && GSM))
1	1	1	1	1	1		
1	1	1	1	0	0		
1	1	1	0	1	0		
1	1	1	0	0	0		
1	1	0	1	1	1		
1	1	0	1	0	0		
1	1	0	0	1	0		
1	1	0	0	0	0		
1	0	1	1	1	1		
1	0	1	1	0	0		
1	0	1	0	1	0		
1	0	1	0	0	0		
1	0	0	1	1	1		
1	0	0	1	0	0		
1	0	0	0	1	0		
1	0	0	0	0	0		

La Tavola di Verità

VI	CRP	DRM	DRU	GSM	DRU && GSM	DRM ||(DRU && GSM)	VI && CRP && (DRM||(DRU && GSM))
1	1	1	1	1	1	1	
1	1	1	1	0	0	1	
1	1	1	0	1	0	1	
1	1	1	0	0	0	1	
1	1	0	1	1	1	1	
1	1	0	1	0	0	0	
1	1	0	0	1	0	0	
1	1	0	0	0	0	0	
1	0	1	1	1	1	1	
1	0	1	1	0	0	1	
1	0	1	0	1	0	1	
1	0	1	0	0	0	0	
1	0	0	1	1	1	1	
1	0	0	1	0	0	0	
1	0	0	0	1	0	0	
1	0	0	0	0	0	0	

La Tavola di Verità

VI	CRP	DRM	DRU	GSM	DRU && GSM	DRM ||(DRU && GSM)	VI && CRP && (DRM||(DRU && GSM))
1	1	1	1	1	1	1	1
1	1	1	1	0	0	1	1
1	1	1	0	1	0	1	1
1	1	1	0	0	0	1	1
1	1	0	1	1	1	1	1
1	1	0	1	0	0	0	0
1	1	0	0	1	0	0	0
1	1	0	0	0	0	0	0
1	0	1	1	1	1	1	0
1	0	1	1	0	0	1	0
1	0	1	0	1	0	1	0
1	0	1	0	0	0	0	0
1	0	0	1	1	1	1	0
1	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0

Eventi impossibili

VI	CRP	DRM	DRU	GSM	DRU && GSM	DRM ||(DRU && GSM)	VI && CRP && (DRM||(DRU && GSM))
1	1	1	1	1	1	1	1
1	1	1	1	0	0	1	1
1	1	1	0	1	0	1	1
1	1	1	0	0	0	1	1
1	1	0	1	1	1	1	1
1	1	0	1	0	0	0	0
1	1	0	0	1	0	0	0
1	1	0	0	0	0	0	0
1	0	1	1	1	1	1	0
1	0	1	1	0	0	1	0
1	0	1	0	1	0	1	0
1	0	1	0	0	0	0	0
1	0	0	1	1	1	1	0
1	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0

DRU e DRM non possono essere entrambe vere

Tre possibili scenari

- Italia-Uruguay 2-0, Costa Rica-Inghilterra 0-2
- Italia-Uruguay 2-0, Costa Rica-Inghilterra 1-2
- Italia-Uruguay 4-2, Costa Rica-Inghilterra 0-1

	Squadra	Punti	Differenza reti	Reti segnate
	Costa Rica	6	+3	4
	Italia	3	0	2
	Uruguay	3	-1	3
	Inghilterra	0	-2	2

Italia-Uruguay 2-0, Costa Rica-Inghilterra 0-2

VI	CRP	DRM	DRU	GSM	DRU && GSM	DRM ||(DRU && GSM)	VI && CRP && (DRM||(DRU && GSM))
1	1	1	1	1	1	1	1
1	1	1	1	0	0	1	1
1	1	1	0	1	0	1	1
1	1	1	0	0	0	1	1
1	1	0	1	1	1	1	1
1	1	0	1	0	0	0	0
1	1	0	0	1	0	0	0
1	1	0	0	0	0	0	0
1	0	1	1	1	1	1	0
1	0	1	1	0	0	1	0
1	0	1	0	1	0	1	0
1	0	1	0	0	0	0	0
1	0	0	1	1	1	1	0
1	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0

Tre possibili scenari

- Italia-Uruguay 2-0, Costa Rica-Inghilterra 0-2
  - Italia passa come prima
- Italia-Uruguay 2-0, Costa Rica-Inghilterra 1-2
- Italia-Uruguay 4-2, Costa Rica-Inghilterra 0-1

Italia-Uruguay 2-0, Costa Rica-Inghilterra 1-2

VI	CRP	DRM	DRU	GSM	DRU && GSM	DRM ||(DRU && GSM)	VI && CRP && (DRM||(DRU && GSM))
1	1	1	1	1	1	1	1
1	1	1	1	0	0	1	1
1	1	1	0	1	0	1	1
1	1	1	0	0	0	1	1
1	1	0	1	1	1	1	1
1	1	0	1	0	0	0	0
1	1	0	0	1	0	0	0
1	1	0	0	0	0	0	0
1	0	1	1	1	1	1	0
1	0	1	1	0	0	1	0
1	0	1	0	1	0	1	0
1	0	1	0	0	0	0	0
1	0	0	1	1	1	1	0
1	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0

Tre possibili scenari

- Italia-Uruguay 2-0, Costa Rica-Inghilterra 0-2
  - Italia passa come prima
- Italia-Uruguay 2-0, Costa Rica-Inghilterra 1-2
  - Italia passa come seconda
- Italia-Uruguay 4-2, Costa Rica-Inghilterra 0-1

Italia-Uruguay 4-2, Costa Rica-Inghilterra 0-1

VI	CRP	DRM	DRU	GSM	DRU && GSM	DRM ||(DRU && GSM)	VI && CRP && (DRM||(DRU && GSM))
1	1	1	1	1	1	1	1
1	1	1	1	0	0	1	1
1	1	1	0	1	0	1	1
1	1	1	0	0	0	1	1
1	1	0	1	1	1	1	1
1	1	0	1	0	0	0	0
1	1	0	0	1	0	0	0
1	1	0	0	0	0	0	0
1	0	1	1	1	1	1	0
1	0	1	1	0	0	1	0
1	0	1	0	1	0	1	0
1	0	1	0	0	0	0	0
1	0	0	1	1	1	1	0
1	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0

Tre possibili scenari

- Italia-Uruguay 2-0, Costa Rica-Inghilterra 0-2
  - Italia passa come prima
- Italia-Uruguay 2-0, Costa Rica-Inghilterra 1-2
  - Italia passa come seconda
- Italia-Uruguay 4-2, Costa Rica-Inghilterra 0-1
  - Italia passa come prima

Tre possibili scenari

- Italia-Uruguay 2-0, Costa Rica-Inghilterra 0-2
  - Italia passa come prima
- Italia-Uruguay 2-0, Costa Rica-Inghilterra 1-2
  - Italia passa come seconda
- Italia-Uruguay 4-2, Costa Rica-Inghilterra 0-1
  - Italia passa come prima

L'amara verità

- Italia-Uruguay 0-1, Costa Rica-Inghilterra 0-0
  - Italia eliminata

Linguaggio C: Costrutto Condizionale

Istruzioni composta: `if`

Costrutto Condizionale: `if`, la sintassi

- Il costrutto **condizionale** permette di eseguire alcune istruzioni a seconda del valore di un'espressione booleana a runtime
- `if,else` keywords
- **`expression`** espressione booleana (vale 0 o 1)
- **`statement`** è la sequenza di istruzioni da eseguire (corpo) quando **`expression`** è vera (nel caso di **`statement0`** quando è falsa)
- se **`statement`** contiene più istruzioni, va delimitato tra { }
- **NB** indentatura irrilevante

```
if(expression)
    statement
```

```
if(expression)
    statement1
else
    statement0
```

Costrutto Condizionale: `if`, l'esecuzione

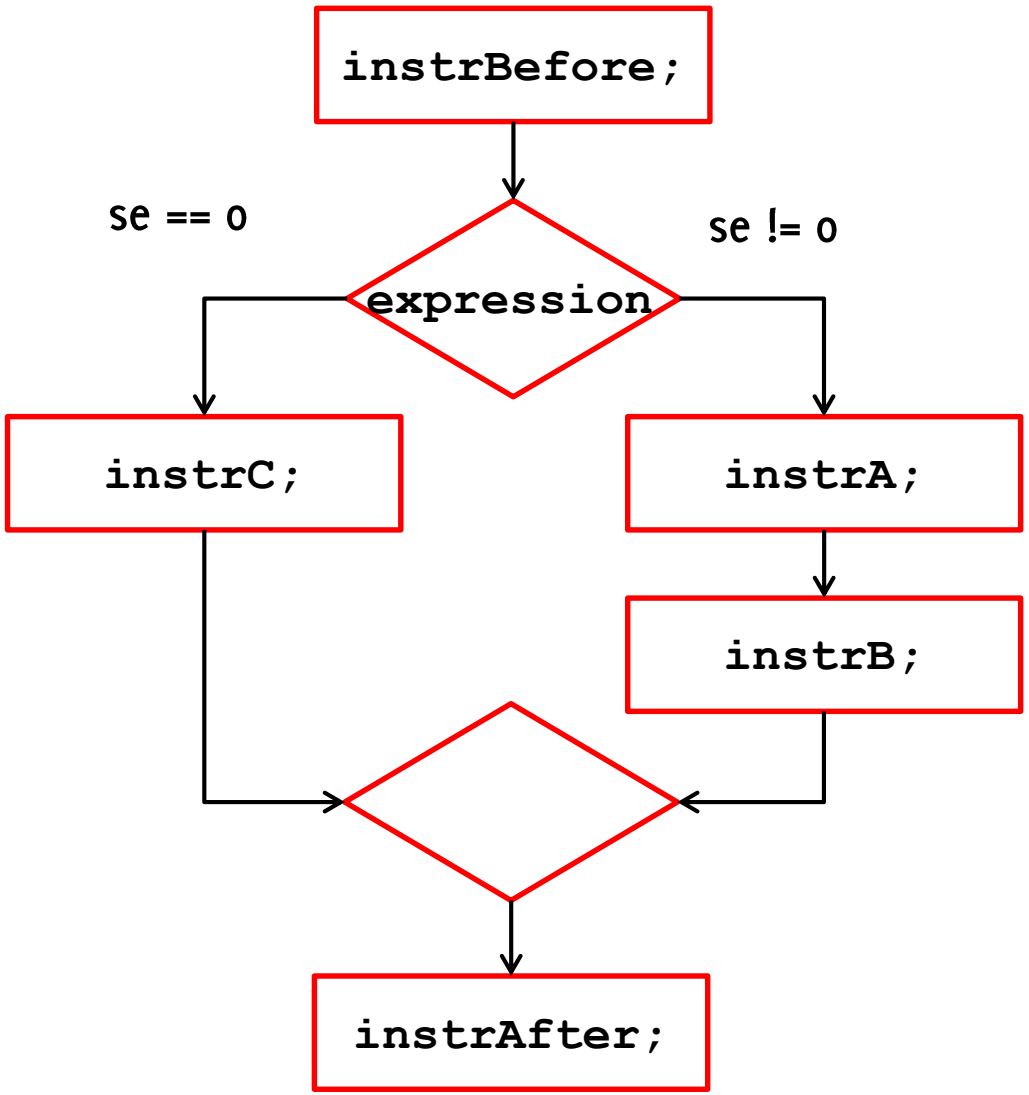
1. Terminata **`instrBefore`**, valuto **`expression`**,
2. Se **`expression`** è vera (**`!=0`**), allora eseguo **`statement1`**, altrimenti eseguo **`statement0`**. (se è presente **`else`**)
3. Terminato lo statement dell'**`if`** , procedi con **`instrAfter`**, la prima istruzione fuori dall'**`if`**

N.B. **`else`** è opzionale

N.B **`if(expression)`** **non** richiede il **`;`** perché l'istruzione non termina dopo **`)`**

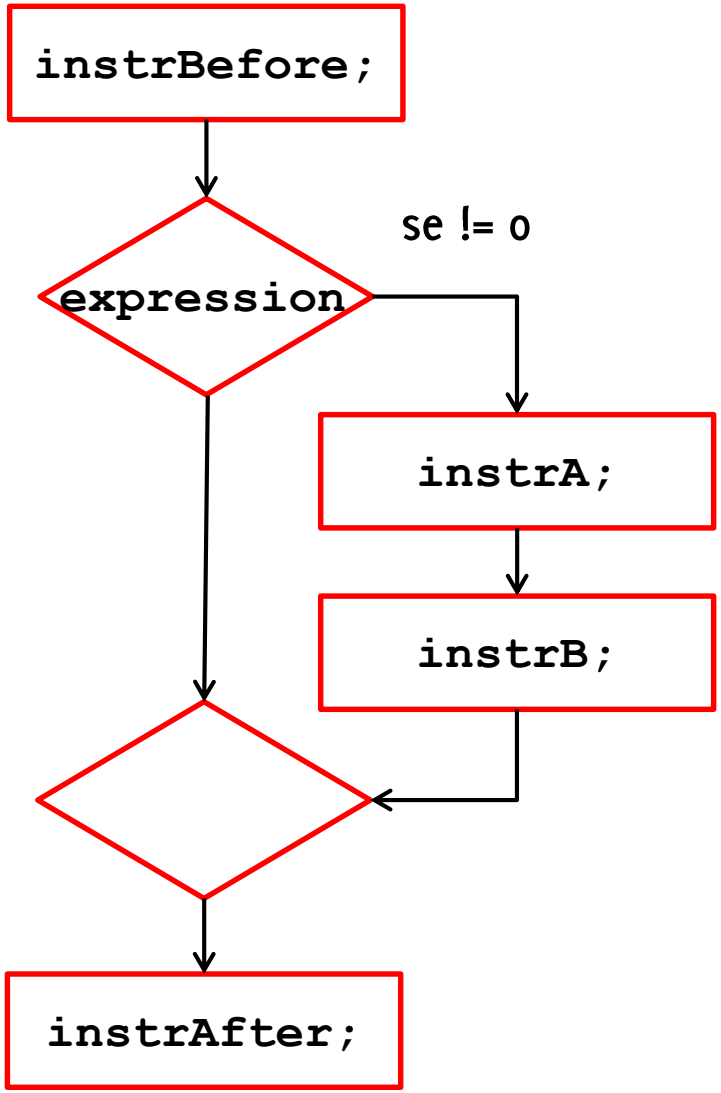
```
instrBefore;
if(expression)
        statement1;
else
        statement0;
instrAfter;
```

Costrutto Condizionale: `if`, l'esecuzione




```
instrBefore;
if(expression)
        {
        instrA;
        instrB;
        }
else
        instrC;
instrAfter;
```

Costrutto Condizionale: `if`, l'esecuzione




```
instrBefore;
if(expression)
        {
        instrA;
        instrB;
        }
instrAfter;
```

Esempio

```
//N.B: incolonnamento codice irrilevante!
if (x % 7 == 0)
     printf("%d multiplo di 7" , x);
else
     printf("%d non multiplo di 7" , x);

//si può fare senza else?
```

Esempio

```
//N.B: incolonnamento codice irrilevante!
if (x % 7 == 0)
     printf("%d multiplo di 7" , x);
else
     printf("%d non multiplo di 7" , x);

//senza else.
printf("%d " , x);
if (x % 7 != 0)
     printf("non "); // { printf("non "); }
printf(" multiplo di 7");
```

`if` Annidati

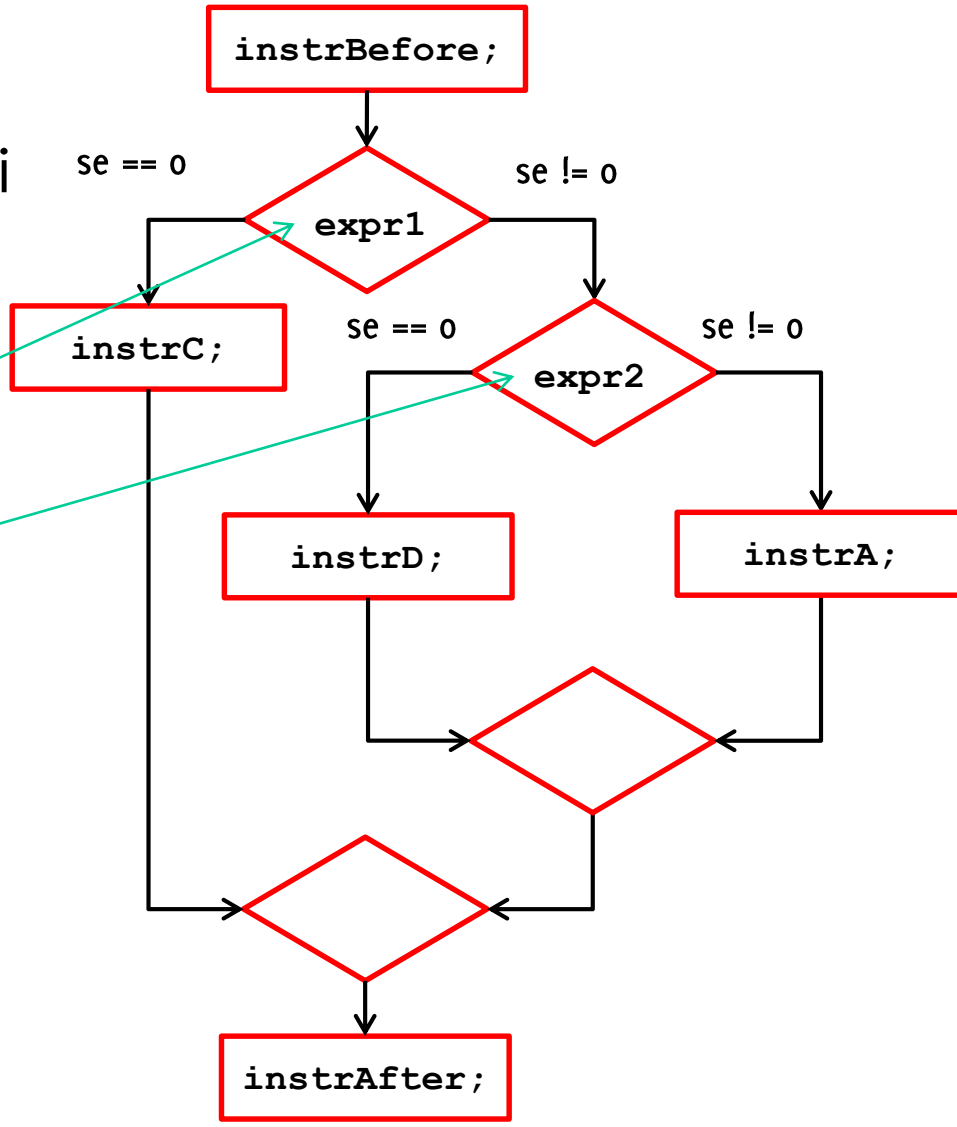
Il corpo di un **`if`** (cioè gli **`statement`**) può a loro volta contenere costrutti altri **`if:`** si realizzano quindi istruzioni condizionali **annidate**

```
instrBefore;
if(expr1)
 if(expr2)
      instrA;
  else
      instrD;
else
 instrC;
instrAfter;
```

`if` Annidati

Il corpo di un **`if`** (cioè gli **`statement`**) può a loro volta contenere costrutti altri **`if:`** si realizzano quindi istruzioni condizionali **annidate**

```
instrBefore;
if(expr1)
 if(expr2)
      instrA;
  else
      instrD;
else
 instrC;
instrAfter;
```



`if` Annidati

Le istruzioni condizionali possono essere annidate, inserendo un ulteriore **`if`** all'interno di **`statement1`** o **`statement0`**

Esempio

```
if ( x % 5 == 0 )
    if (x % 7 == 0 )
        printf("x multiplo di 5 e anche di 7");
    else
        printf("x multiplo di 5 ma non di 7");
else
    printf("x non multiplo di 5");
```

Regole per `if` annidati

Regola: In caso di costrutti annidati, ed in assenza di parentesi che indichino diversamente, ogni **`else`** viene associato all' **`if`** più vicino.

```
if ( x % 5 == 0 )
      if (x % 7 == 0 )
            printf("x multiplo di 5 e anche di 7");
      else
            printf("x multiplo di 5 ma non di 7");
```

Regole per `if` annidati

Regola: In caso di costrutti annidati, ed in assenza di parentesi che indichino diversamente, ogni **`else`** viene associato all' **`if`** più vicino.

```
if ( x % 5 == 0 )
        {
        if (x % 7 == 0 )
            printf("x multiplo di 5 e anche di 7");
        }
else
        printf("x non multiplo di 5");
```

`if` Annidati

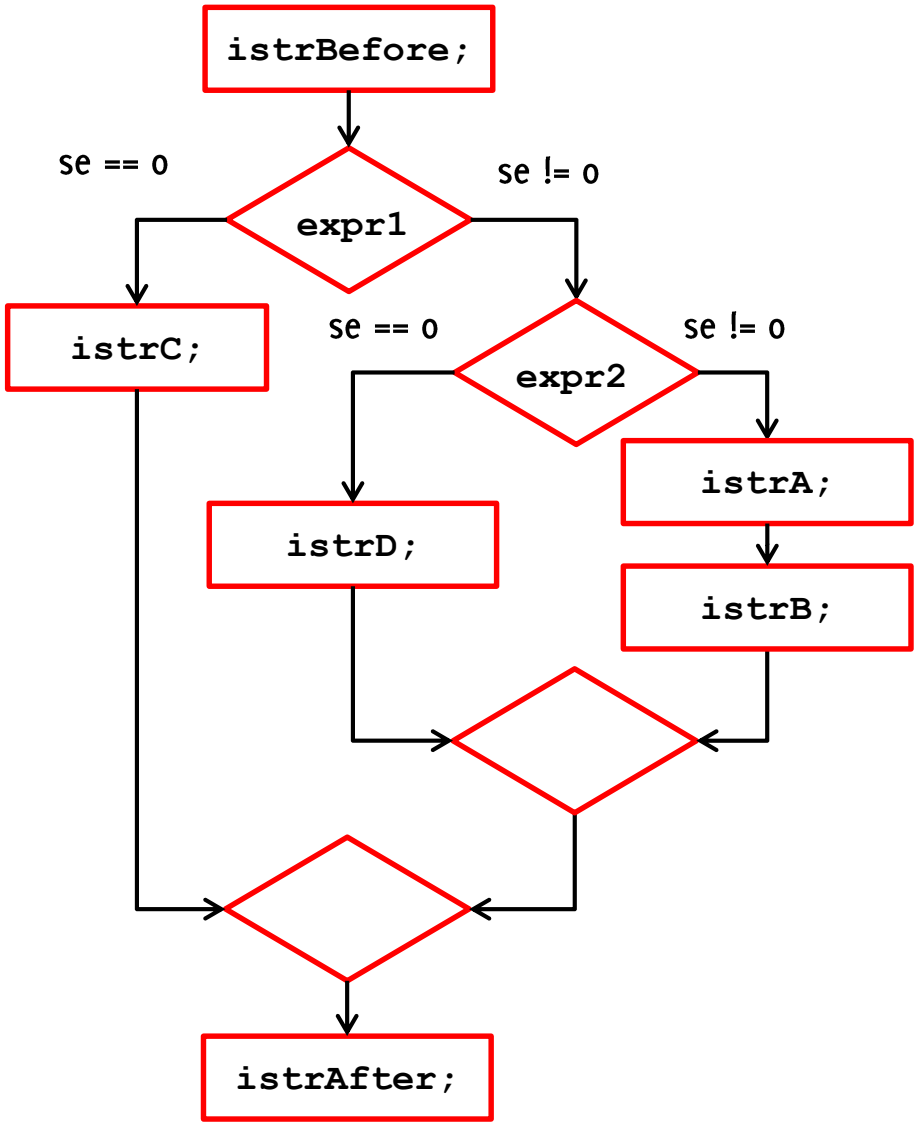
Quando il corpo di un if contiene più di un'istruzione è necessario usare parentesi.

```
instrBefore;
if(expr1)
 if(expr2)
      {instrA;
       instrB;}
  else
      instrD;
else
 instrC;
instrAfter;
```

if Annidati

Quando il corpo di un if contiene più di un'istruzione è necessario usare parentesi.

```
instrBefore;
if(expr1)
 if(expr2)
      {instrA;
       instrB;}
  else
      instrD;
else
 instrC;
instrAfter;
```



`if` Annidati

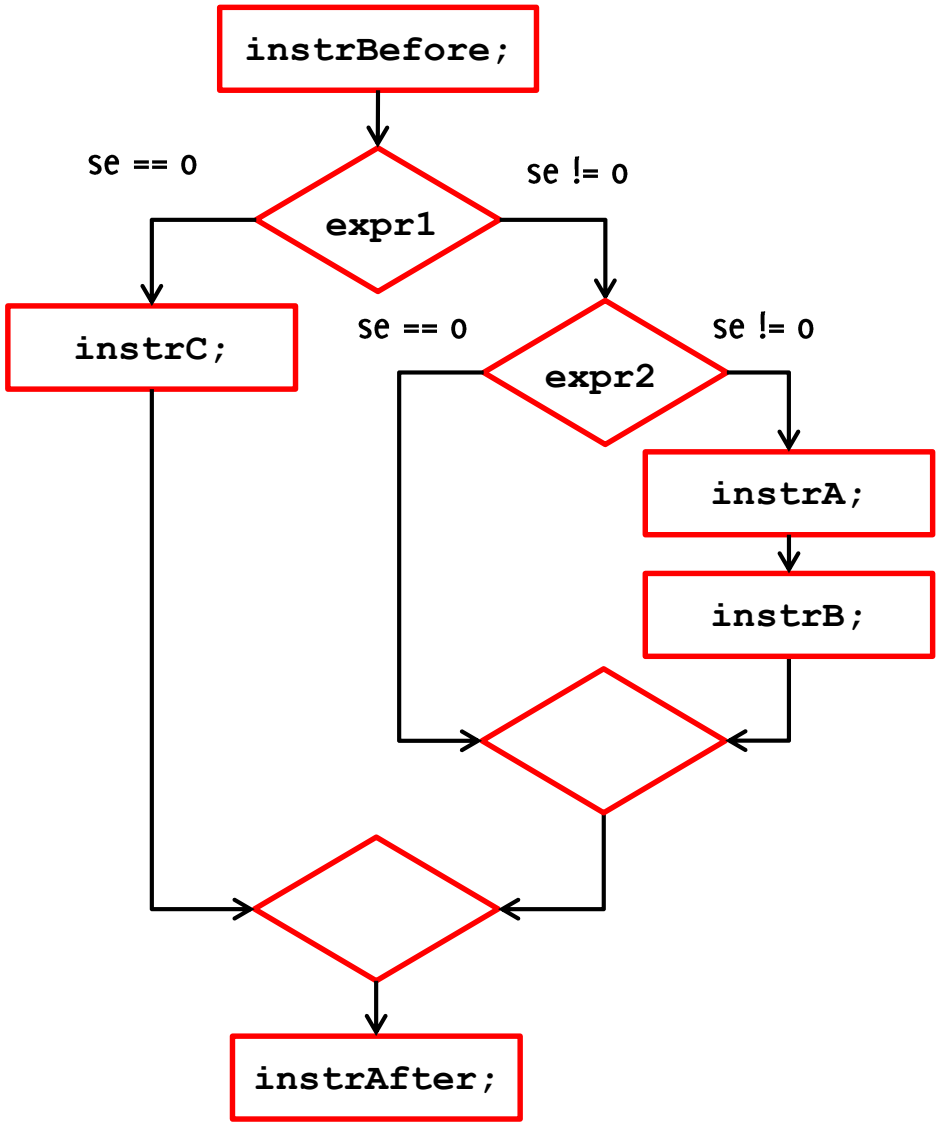
L'uso delle parentesi serve per determinare quale **`else`** associare a quale **`if`**.

```
instrBefore;
if(expr1)
{  if(expr2)
       {instrA;
        instrB;}
}
else
 instrC;
instrAfter;
```

`if` Annidati

L'uso delle parentesi serve per determinare quale **`else`** associare a quale **`if`**.

```
instrBefore;
if(expr1)
{  if(expr2)
       {instrA;
        instrB;}
}
else
 instrC;
instrAfter;
```



Esempio

Scrivere un programma che, inserito un intero positivo, determina se corrisponde ad un anno bisestile

- Un anno è bisestile se
  - è multiplo di 4 ma non di 100
  - oppure se è multiplo di 400

Esempio

Scrivere un programma che determina il massimo tra tre numeri inseriti da tastiera

Soluzione 1: If Annidati

```
#include<stdio.h>
void main()
{
int a,b,c;
printf("\ninserire a: ");
scanf("%d", &a);
printf("\ninserire b: ");
scanf("%d", &b);
printf("\ninserire c: ");
scanf("%d", &c);
if(a > b)
        if(a > c) // b non può essere il max
                printf("\nmax = %d", a);
        else
                printf("\nmax = %d", c);
else
        if(b > c)
                printf("\nmax = %d", b);
        else
                printf("\nmax = %d", c);
}
```

Soluzione 1: if Annidati

```c
#include<stdio.h>
void main()
{
int a,b,c;
printf("\ninserire a: ");
scanf("%d", &a);
printf("\ninserire b: ");
scanf("%d", &b);
printf("\ninserire c: ");
scanf("%d", &c);
if(a > b)
        if(a > c) // b non può essere il max
                printf("\nmax = %d", a);
        else
                printf("\nmax = %d", c);
else
        if(b > c)
                printf("\nmax = %d", b);
        else
                printf("\nmax = %d", c);
}
```

Osservazioni:

1. Il numero di indentazioni è n, pari a quanti numeri occorre controllare
2. Le parentesi negli if non sono necessarie qua

Soluzione 2: Condizioni Composte

```
#include<stdio.h>
void main()
{
int a,b,c;
printf("\ninserire a: ");
scanf("%d", &a);
printf("\ninserire b: ");
scanf("%d", &b);
printf("\ninserire c: ");
scanf("%d", &c);
if(a >= b && a >= c)
        printf("\nmax = %d", a);
if(b >= c && b >= a)
        printf("\nmax = %d", b);
if(c >= a && c >= b)
        printf("\nmax = %d", c)

}
```

Osservazioni:

1. Condizioni composte si allungano quando si aggiungono numeri da controllare
2. Il numero di condizioni da valutare per n numeri è n
3. If usati in sequenza
4. E' necessario mettere $\geq$ altrimenti non gestisce correttamente il caso in cui almeno due numeri sono uguali

Soluzione 3: if in sequenza

```
#include<stdio.h>
void main()
{
int a,b,c;
printf("\ninserire a: ");
scanf("%d", &a);
printf("\ninserire b: ");
scanf("%d", &b);
printf("\ninserire c: ");
scanf("%d", &c);

max = a;
if(max < b)
        max = b;
if(max < c)
        max = c;

printf("\nmax(%d,%d,%d) = %d", a, b, c, max);
}
```

Osservazioni:

1. L'uso della variabile ausiliaria facilita le cose
2. Non ricorda niente questa soluzione?

Vi ricordate?
Algoritmo per ricercare il prodotto migliore

1. Prendi in mano il primo prodotto: assumi che sia il migliore
2. Procedi fino al prossimo prodotto
3. Confrontalo con quello che hai in mano
4. **Se** il prodotto davanti a te è migliore: abbandona il prodotto che hai in mano e prendi quello sullo scaffale
5. **Ripeti** i passi **2 - 4** **fino a** raggiungere la fine della corsia
6. Hai in mano il prodotto migliore.

Algoritmo per trovare il massimo di una sequenza numerica

Linguaggio C: Costrutti Iterativi

Istruzioni composte: **`while, do while, for`**

Costrutto Iterativo: `while`, la sintassi

- Il costrutto iterativo permette di ripetere l'esecuzione di istruzioni finché una condizione è valida
- **`while`** è una keyword
- **`expression`** espressione booleana, condizione che determina la **permanenza** nel ciclo
- **`statement`** sequenza di istruzioni da eseguire (corpo del ciclo)
- **NB**: come per **`if`**, se **`statement`** contiene più istruzioni, va delimitato tra **{ }**

```
while (expression)
    statement
```

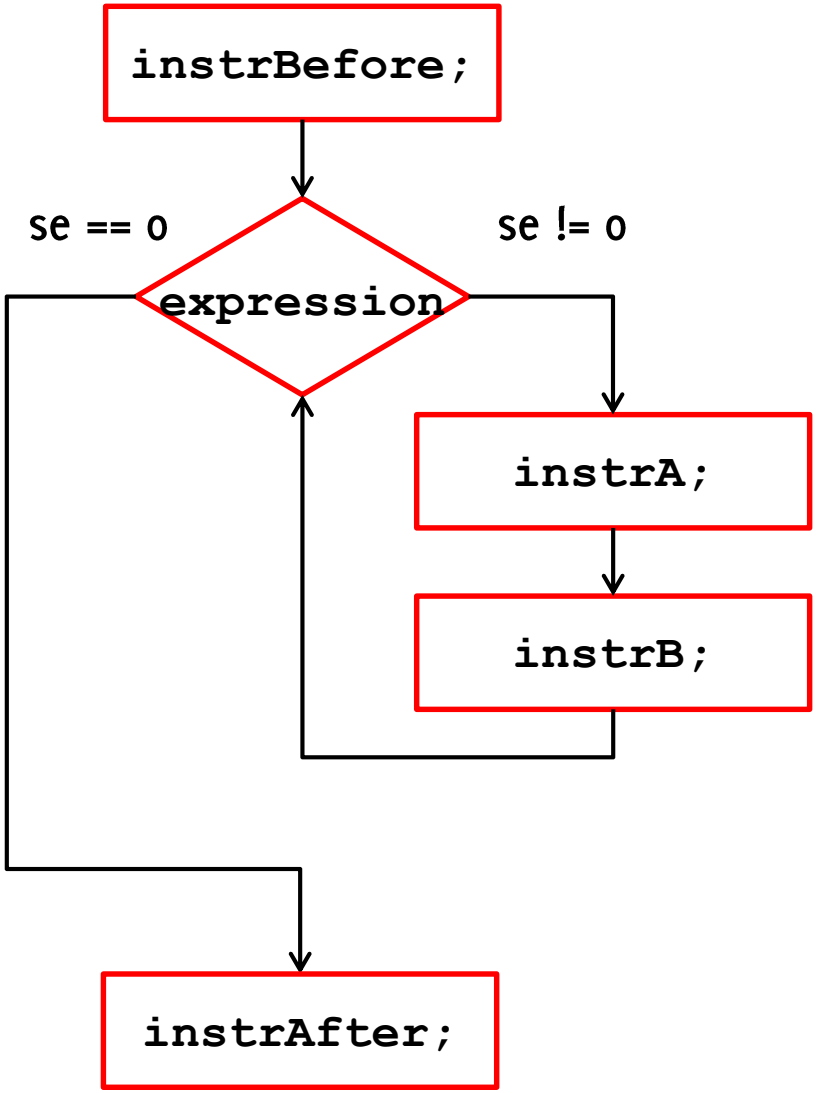
Costrutto Iterativo: `while`, l'esecuzione

1. Terminata **`instrBefore`** viene valutata **`expression`**
2. Se **`expression`** è vera (0 != 0) viene eseguito **`statement`**
3. Al termine, viene valutata nuovamente **`expression`** e la procedura continua finché **`expression`** è falsa (== 0)
4. Uscito dal ciclo, eseguo **`instrAfter`**

```
instrBefore;
while(expression)
        statement;
instrAfter;
```

N.B: **`while(expression)`** **non** richiede il **`;`** perché l'istruzione non termina dopo **`)`** ma con lo **`statement:`**
`while(expression);` è un ciclo senza corpo

Costrutto Iterativo: `while`, l'esecuzione




```
instrBefore;
while(expression)
      {
      instrA;
      instrB;
      }
instrAfter;
```

Esempio

```
/* stampa i primi 100 numeri*/
```

Esempio

```
/* stampa i primi 100 numeri*/
# include<stdio.h>
void main()
{
      int a = 1;
      while(a < 100)
      {
           printf("\n%d" , a);
           a++;
      }
}
```

Esempio

```
/* stampa i primi 100 numeri pari */
```

Esempio

```
/* stampa i primi 100 numeri pari */
# include<stdio.h>
void main()
{
      int a = 1;
      while(a < 100)
      {
           printf("\n%d" , 2*a);
           a++;
      }
}
```

Costrutto Iterativo: `while`, Avvertenze

- Il corpo del **`while`** non viene mai eseguito quando **`expression`** risulta falsa al primo controllo
- Se **`expression`** è vera ed il corpo non ne modifica mai il valore, allora abbiamo un loop infinito (l'esecuzione del programma **non** termina)

```
/* Esempio: stampa i primi 100 numeri pari */
# include<stdio.h>
void main()
{
        int a = 1;
        while(a < 100)
        {
            printf("\n%d" , 2*a);
            a++;
        }
}
```

Costrutto Iterativo: `while`, Avvertenze

- Il corpo del **`while`** non viene mai eseguito quando **`expression`** risulta falsa al primo controllo
- Se **`expression`** è vera ed il corpo non ne modifica mai il valore, allora abbiamo un loop infinito (l'esecuzione del programma **non** termina)

```
/* Esempio: stampa i primi 100 numeri pari */
# include<stdio.h>
void main()
{
        int a = 1;
        while(a > 0)
        {
            printf("\n%d" , 2*a);
            a++;
        }
}
```

Costrutto Iterativo: `while`

```
/* eseguire la somma di una sequenza di numeri
inseriti dall'utente (continuare fino a quando
l'utente inserisce 0)*/

# include<stdio.h>
void main()
{









}
```

Costrutto Iterativo: while

```
/* eseguire la somma di una sequenza di numeri
inseriti dall'utente (continuare fino a quando
l'utente inserisce 0)*/

# include<stdio.h>
void main()
{
        int a , somma;
        somma = 0;
        printf("\nInserire a:");
        scanf("%d" , &a);
        while (a > 0)
        {
                somma += a; //somma = somma + a;
                printf("\nInserire a:");
                scanf("%d" , &a);
        }
        printf("\nSomma = %d" ,somma);
}
```

Costrutto Iterativo: `while`

```
/* eseguire la somma e la media di una sequenza di numeri
inseriti dall'utente (continuare fino a quando l'utente
inserisce 0)*/

# include<stdio.h>
void main()
{









}
```

Costrutto Iterativo: while

```
/* eseguire la somma e la media di una sequenza di numeri
inseriti dall'utente (continuare fino a quando l'utente
inserisce 0)*/

# include<stdio.h>
void main()
{
        int a , somma , n; float media;
        somma = 0; n = 0;
        printf("\nInserire a:");
        scanf("%d" , &a);
        while (a > 0)
        {
                somma += a;
                n++; //n = n + 1;
                printf("\nInserire a:");
                scanf("%d" , &a);
        }
        media = (1.0 * somma) / n;
        printf("\nSomma = %d , media  = %f", somma , media);
}
```

Esercizio di warm up

Preparare un programma C per giocare a
Carta / Sasso / Forbice, richiedendo all'utente di inserire I caratteri 'c', 's', 'f', controllando anche che il carattere inserito sia ammissibile.

Costrutto Iterativo: `do-while`, l'esecuzione

1. Viene eseguito **`statement`**
2. Viene valutata **`expression`** se è vera viene eseguito **`statement`** e la procedura continua finché **`expression`** diventa falsa (== 0)
3. Viene eseguita l'istruzione successiva al ciclo

```
do
   statement
while
(expression);
```

do-while

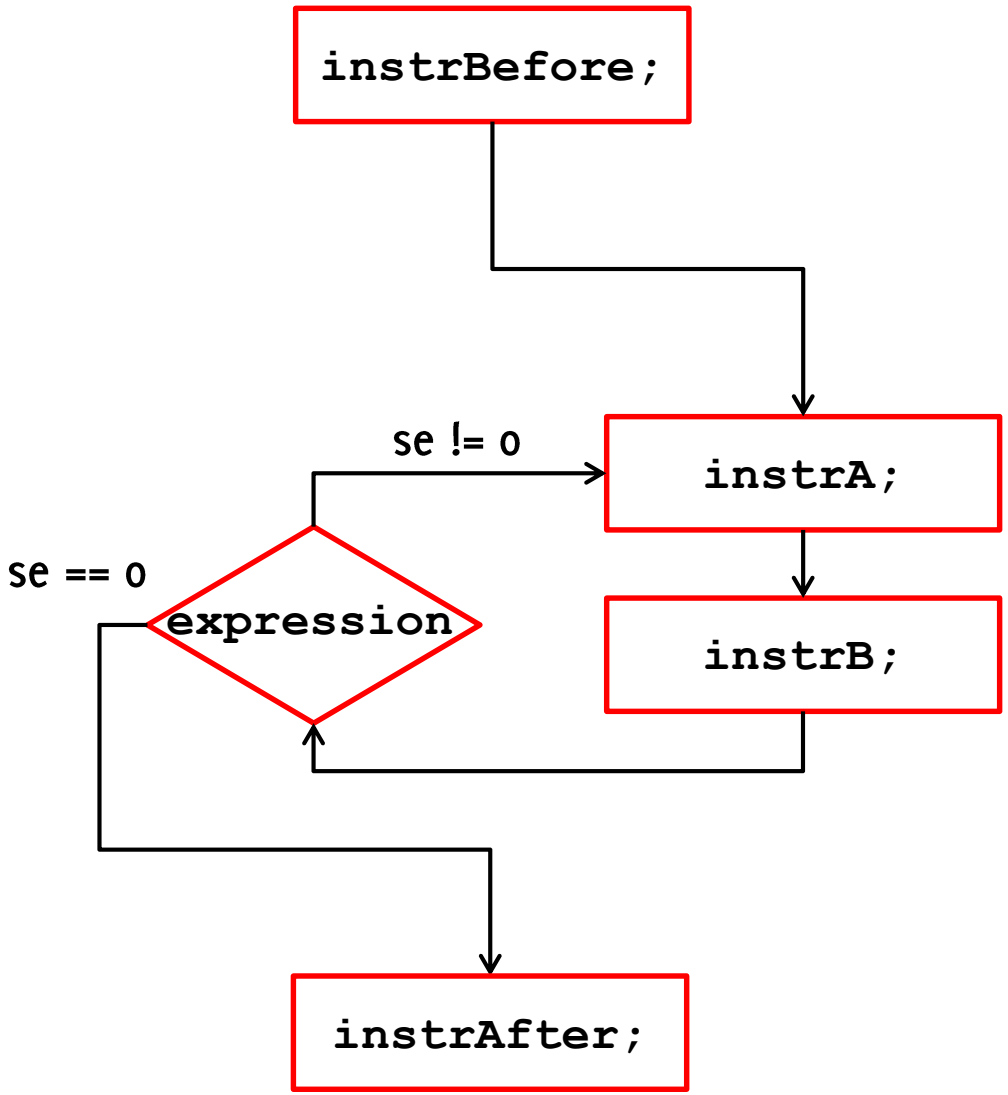
- Il costrutto **do-while** garantisce l'esecuzione del corpo del **while** almeno una volta.

```
do
   statement
while (expression);
```

```
statement
while (expression)
   statement
```

- Utile per garantire che valori acquisiti con **scanf** soddisfino certi prerequisiti
- **NB**: **do-while** richiede il **;** in **while(expression);** il **while** no
- **NB**: come per **if**, se **statement** contiene più istruzioni, va delimitato tra **{}**

do-while




```
instrBefore;
do
   {
   instrA;
   instrB;
   }
while (expression);
instrAfter;
```

Esercizi TODO:

- Inserire un controllo nel programma dell'anno bisestile per assicurarsi che il numero inserito da tastiera sia un intero positivo

- Preparare un programma C per giocare a Carta / Sasso / Forbice, richiedendo all'utente di inserire I caratteri 'c', 's', 'f', controllando anche che il carattere inserito sia ammissibile.

Teorema di Boehm-Jacopini

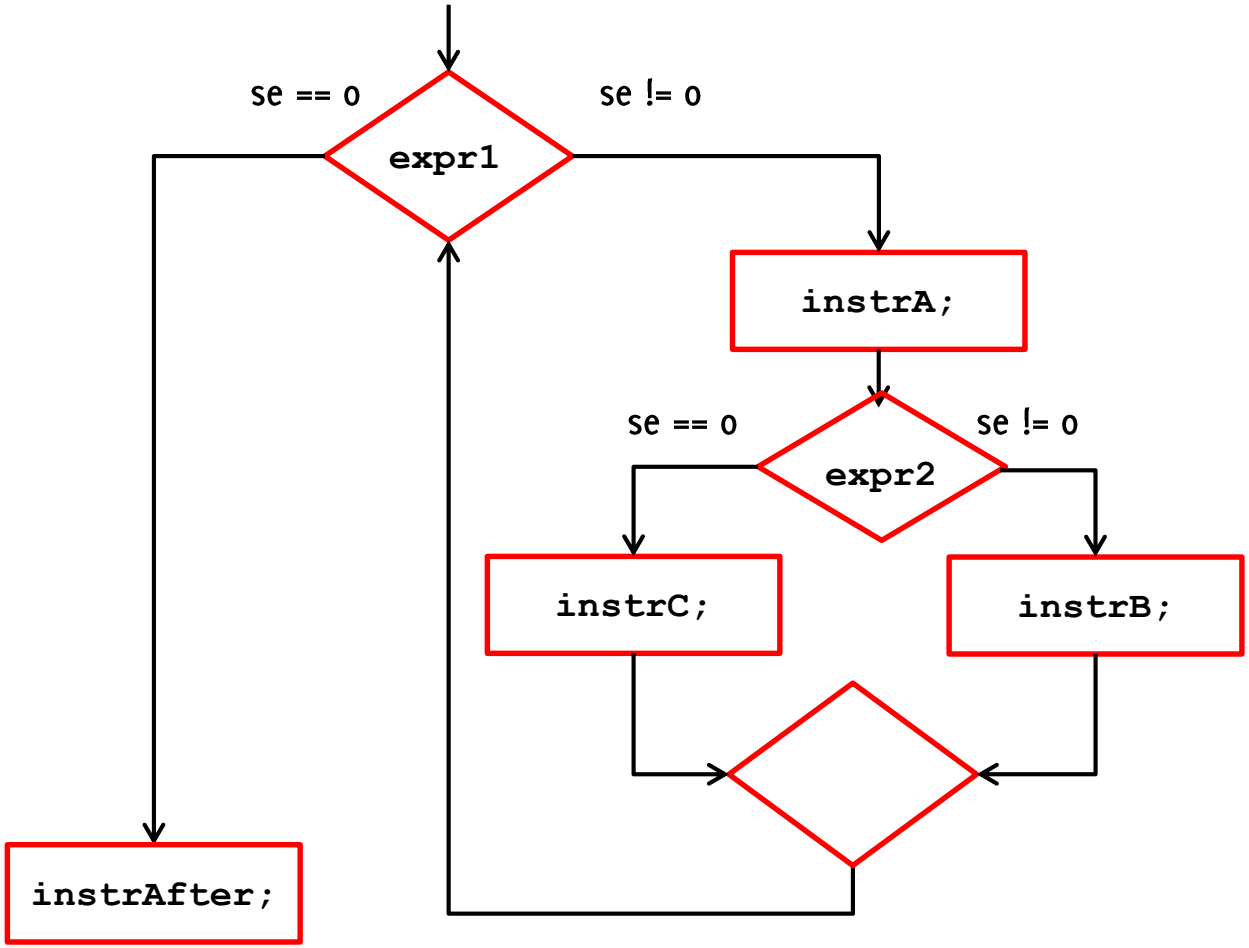
- istruzioni **`if`** e **`while`** (e la possibilità di eseguire istruzioni in sequenza) sono equivalenti a istruzioni che la macchina di Von Neumann che può manipolare registro Contatore di Programma

  ➡ istruzioni **`if`** e **`while`** sono complete:
  bastano per codificare qualsiasi algoritmo

- Per praticità e convenienza si usano però molte altre strutture di controllo

Cicli Annidati…

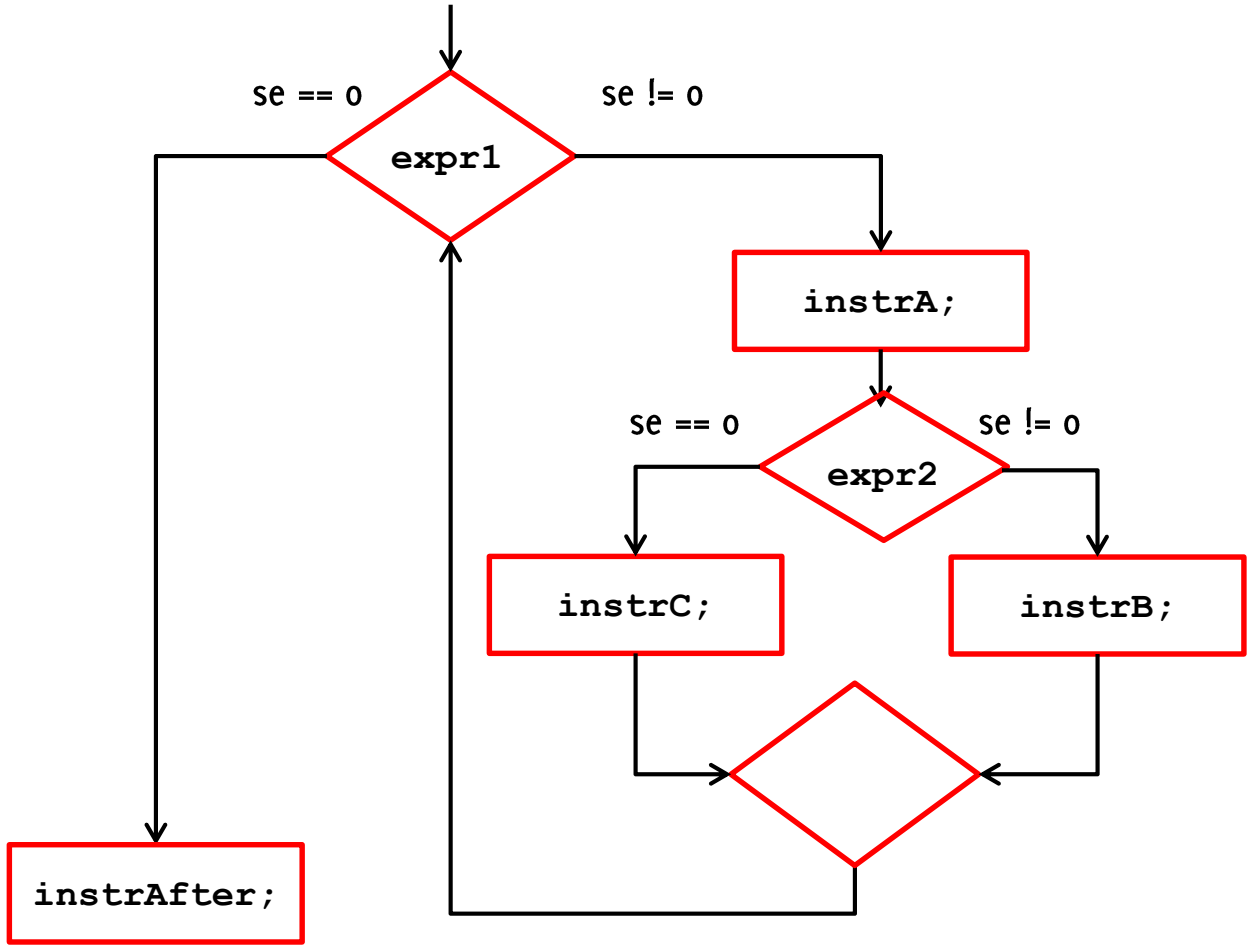
Ovviamente anche il corpo del **`while`** può contenere altri costrutti (**`while`** / **`if`** o altri che vedremo poi)



Cicli Annidati…

Ovviamente anche il corpo del **while** può contenere altri costrutti (**while** / **if** o altri che vedremo poi)

```
while(expr1)
{
   instrA;
   if (expr2)
        instrB;
   else
        instrC;
}
instrAfter;
```



Esercizi (TODO)

- Scrivere un programma che richiede all'utente una sequenza di caratteri minuscoli e ne stampa il corrispettivo maiuscolo (fino a quando l'utente non inserisce #)
- Scrivere un programma che richieda all'utente di inserire due interi e ne calcola il massimo comune divisore.
  Modificarlo per provare meno divisori del minimo tra gli input, utilizzando variabili di flag.
- Scrivere un programma che stampa la tabella pitagorica
  - **Modificarlo per stampare solo la parte triangolare alta/ solo la parte triangolare bassa / solo la diagonale**

TABELLINE

x	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10
2	0	2	4	6	8	10	12	14	16	18	20
3	0	3	6	9	12	15	18	21	24	27	30
4	0	4	8	12	16	20	24	28	32	36	40
5	0	5	10	15	20	25	30	35	40	45	50
6	0	6	12	18	24	30	36	42	48	54	60
7	0	7	14	21	28	35	42	49	56	63	70
8	0	8	16	24	32	40	48	56	64	72	80
9	0	9	18	27	36	45	54	63	72	81	90
10	0	10	20	30	40	50	60	70	80	90	100

Costrutto Iterativo: `for`, la sintassi

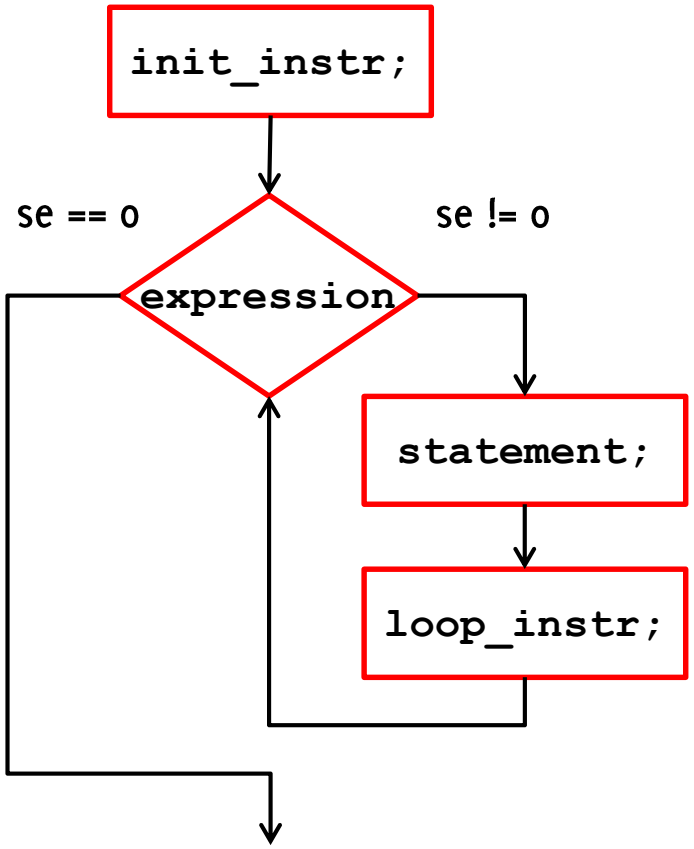
```
for(init_instr; expression; loop_instr)
      statement
```

- **`for`** è un costrutto iterativo, equivalente al **`while`**
  - **`for`** keyword
  - **`init_instr`** istruzione (di inizializzazione)
  - **`expression`** espressione booleana
  - **`loop_instr`** istruzione (di loop)
  - **`statement`** corpo del ciclo
- **NB**: se **`statement`** contiene più istruzioni, richiede **`{ }`**

Costrutto Iterativo: `for`, l'esecuzione

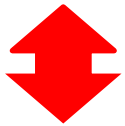
```
for(init_instr; expression; loop_instr)
    statement
```

1. Esegue **`init_instr`**
2. Valuta **`expression`**
3. Se vera, esegue **`statement`**, se falsa, termina il loop.
4. Al termine di **`statement`** esegue **`loop_instr`**
5. Valuta **`expression`**



for vs while

```
for(init_instr; expression; loop_instr)
      statement
```



```
init_instr;
while(expression)
      {
      statement;
      loop_instr;
      }
```

Utile per cicli regolati da una «variabile di loop»

- inizializzata con **`init_instr`**
- Incremento regolato da **`loop_instr`**

for vs while

```
…
i = 0;
while (i < 100)
      {
      //statement
      i++;
      }
…
```

for vs while

```
…
i = 0;
while (i < 100)
      {
      //statement
      i++;
      }
…
```

```
…
for(i=0; i<100; i++)
      //statement

…
```

Il **for**, nei cicli regolati da variabile di loop

- ha una stesura più compatta
- mette in evidenza la variabile di loop e come questa evolve

for vs while

```
…
scanf("%d", &a);
while (a < 0)
      scanf("%d", &a);
…
```

for vs while

```
…
scanf("%d", &a);
while (a < 0)
      scanf("%d", &a);
…
```

```
…
scanf("%d", &a);
for( ; a < 0; )
      scanf("%d", &a);
…
```

Nei cicli senza variabile di loop è comunque possibile usare il **for**, lasciando vuote **loop_instr** e **init_instr**

for vs while

```
…
scanf("%d", &a);
while (a < 0)
      scanf("%d", &a);
…
```

```
…
scanf("%d", &a);
for( ; a < 0; )
      scanf("%d", &a);
…
```

Nei cicli senza variabile di loop è comunque possibile usare il **for**, lasciando vuote **loop_instr** e **init_instr**

Altra soluzione (tecnicamente possibile ma inusuale)

```
…
for(scanf("%d", &a); a < 0; scanf("%d", &a))
…
```

for vs while

```
…
scanf("%d", &a);
while (a < 0)
      scanf("%d", &a);
…
```

```
…
do
      scanf("%d", &a);
while (a < 0);
…
```

In questo caso **do while**, risulta più compatto e chiaro

Esempio `for`

- Stampare i primi 100 numeri

Esempio `for`

- Stampare i primi 100 numeri

```
for ( j = 0;  j < 100;  j++)
    printf( "%d", j );
```

- Stampare i quadrati perfetti minori di L

Esempio `for`

- Stampare i primi 100 numeri

```
for ( j = 0;  j < 100;  j++)
    printf( "%d", j );
```

- Stampare i quadrati perfetti minori di L

```
for( n = 1; n*n < L; n++ )
        printf("%d", n*n);
```

- Scrivere una soluzione basata su **`for`** per gli esercizi
  - Le tabelline,
  - Le tabelline il triangolo inferiore

`break` e `continue`

- L'istruzione **`break`** termina l'esecuzione dei seguenti costrutti
  - **`while`**, **`do while`** e **`for`** (costsrutti iterativi)
  - **`switch`** (evita l'esecuzione di tutti i casi in cascata)
- L'istruzione **`continue`** all'interno di un costrutto iterativo passa direttamente all'iterazione seguente, interrompendo quella corrente.
  - **`continue`** può essere utilizzato solo nei cicli iterativi, i.e.: **`while`**, **`do while`**, **`for`**.

Cosa fa?

```
for(i=0; i<10; i++) {
      scanf("%d",&x);
      if(x < 0)
            break;
      printf("%d", x);
}
```

Cosa fa?

```
for(i=0; i<10; i++) {
      scanf("%d",&x);
      if(x < 0)
            break;
      printf("%d", x);
}
```

Richiede fino a 10 numeri e ne stampa il valore inserito. Le acquisizioni terminano anticipatamente se viene inserito un valore negativo. Il valore negativo non viene stampato a schermo.

N.B il **`break`** interrompe comunque il costrutto iterativo (**`for`**) anche se si trova all'interno dell' **`if`**

Cosa fa?

```
i = 0;
while(i<10) {
      scanf("%d",&x);
      if(x < 0)
            continue;
      printf("%d", x);
      i++;
}
```

Cosa fa?

```
i = 0;
while(i<10) {
      scanf("%d",&x);
      if(x < 0)
            continue;
      printf("%d", x);
      i++;
}
```

Richiede numeri (anche infiniti) fino a quando non ne vengono inseriti 10 positivi. Stampa a schermo il valore inserito di ogni positivo. Per i valori negativi non viene stampato il valore inserito e nemmeno incrementata **i** (il **continue** fa saltare tutte le successive istruzioni)

Cosa fa?

```
for(i=0; i<10; i++) {
      scanf("%d",&x);
      if(x < 0)
            continue;
      printf("%d", x);
}
```

Cosa fa?

```
for(i=0; i<10; i++) {
      scanf("%d",&x);
      if(x < 0)
            continue;
      printf("%d", x);
}
```

Richiede esattamente 10 numeri e ne stampa il valore inserito. Le acquisizioni **non** terminano se viene inserito un valore negativo, però non viene stampato il valore inserito (il **`continue`** fa saltare alla successiva esecuzione)

La **`loop_expr`** non viene saltata dal **`continue`**.
È una particolarità del **`for`**

Alternative a `break` e `continue`

- Utilizzo di cicli con variabili **flag (o sentinella)** per terminare anticipatamente l'esecuzione del ciclo
- Una variabile che assume un valore 0 / 1 a seconda che si verifichino o meno alcune condizioni durante l'esecuzione

Esempio: Alternativa e break e continue

Es: Scrivere un ciclo con che richiede una serie di valori interi e li associa alla variabile intera **`n`** e stampa a schermo

- non più di **`N`** richieste
- saltando i valori negativi inseriti
- interrompendo l'elaborazione al primo valore nullo incontrato

Esempi con continue e break

```
for (i = 0 ; i < N ; i++ ) {
 printf("immetti un intero>0 ");
 scanf("%d",&n);
 if (n < 0)
   continue;
 if (n == 0)
   break;
 printf("%d",n);
   .. /*elabora i positivi */
}
```

MOLTO IMPORTANTE: come farne a meno

```
int n, i, flag;
flag = 0;//diventa 1 quando inserisco zero.
for(i =0; i <= N && flag == 0; i++)
{
    printf("\ninserire n: ");
    scanf("%d", &n);
    if(n==0)
        flag = 1;
    else
        if(n > 0)
            printf(" %d", n);
/*eventuali altre istruzioni per i positivi*/
}
```

Importanza delle variabili di flag

Es: Scrivere un ciclo con che richiede una serie di valori interi e li associa alla variabile intera `a` e stampa a schermo

- non più di 10 richieste
- saltando i valori negativi inseriti
- interrompendo l'elaborazione al primo valore nullo incontrato
- **Al termine, stampare un messaggio qualora fossero stati inseriti 10 numeri positivi**

MOLTO IMPORTANTE: come farne a meno

```
int n, i, flag;
flag = 0;//diventa 1 quando inserisco zero.
for(i =0; i <= 10 && flag == 0; i++)
{
    printf("\ninserire n: ");
    scanf("%d", &n);
    if(n==0)
        flag = 1;
    else
        if(n > 0)
            printf(" %d", n);
/*eventuali altre istruzioni per i positivi*/
}
if(flag == 0)
    printf("\n tutti non nulli");
```

MOLTO IMPORTANTE: come farne a meno

```
int n, i, flag;
flag = 0;//diventa 1 quando inserisco zero.
for(i =0; i <= 10 && flag == 0; i++)
{
    printf("\ninserire n: ");
    scanf("%d", &n);
    if(n==0)
        flag = 1;
    else
        if(n > 0)
            printf(" %d", n);
/*eventuali altre istruzioni per i positivi*/
}
if(flag == 0)
    printf("\n tutti non nulli");
```

se flag è rimasto zero vuol dire che nel ciclo sopra non è mai stato inserito un valore nullo, altrimenti sarebbe diventato 1

MOLTO IMPORTANTE: come farne a meno

```
int n, i, flag;
flag = 0;//diventa 1 quando inserisco zero.
for(i =0; i <= 10 && flag == 0; i++)
{
    printf("\ninserire n: ");
    scanf("%d", &n);
    if(n==0)
        flag = 1;
    else
        if(n > 0)
            printf(" %d", n);
/*eventuali altre istruzioni per i positivi*/
}
if(flag == 0)
    printf("\n tutti non nulli");
```

Se avessi usato il break al posto della variabile di flag non avrei potuto determinare così facilmente se il ciclo sopra si fosse interrotto per via del break o se fosse terminato normalmente

Alcune precisazioni…

Riprendiamo dei dettagli e vediamo gli errori più frequenti

Nota sugli Identificatori

Gli identificatori sono unici: non è possibile associare due identificatori diversi alla stessa variabile o lo stesso identificatore a due variabili diverse.

In un programma, ogni riferimento alla variabile **`a`** rimanda alla stessa cella di memoria. Non esistono altri identificatori per quella cella.

Non si possono usare alcune espressioni come identificatori perché fanno riferimento a parole **riservate, le keywords**.

- Es: **`if`**, **`for`**, **`switch`**, **`while`**, **`main`**, **`printf`**, **`scanf`**, **`int`**, **`float`**, etc…

Note: Dichiarazione di Variabili

- Solo le variabili dichiarate possono essere utilizzate!
- Il fatto che sia richiesta la dichiarazione delle variabili permette gli editor di riconoscere eventuali typos
- Ogni sequenza di caratteri in un codice di un programma C può essere:
  - Un nome di variabile
  - Un nome di funzione
  - Una Keyword
- Provare ad usare una variabile **`a`** senza averla dichiarata. Il compilatore risponde:

```
'a' undeclared (first use in this function)
```

Note: Dichiarazione di Variabili

- Sintassi per la dichiarazione

  `nomeTipo nomeVariabile;`

  Es **`int N;`**

- Le celle di memoria **non sono «vuote»**, ma tipicamente contengono valori non sensati. La dichiarazione **non modifica** tali valori iniziali, sarà il primo assegnamento a farlo.

- Provare per credere ..

```
int N;
printf("%d",N);
scanf("%d",&N);
printf("%d",N);
```

Note: Dichiarazione di Variabili

È necessario inizializzare le variabili di tipo `float` con un valore decimale (altrimenti il valore all'interno viene considerato `int`). Quindi

```
float a = 0.0;
```

Le Costanti

- Dichiarando una costante viene associato **stabilmente** un valore ad un identificatore
- *Esempi*:

```
const float PiGreco = 3.14;
const float PiGreco = 3.1415, e = 2.718;
const int N = 100,   M = 1000;
const char CAR1 = 'A', CAR2 = 'B';
```

- Note:
  - come per le variabili si possono raggruppare dichiarazioni di più costanti dello stesso tipo
  - Il compilatore segnala come errore ogni assegnamento a una costante nella parte eseguibile

Le Costanti (cnt)

- Usare le costanti è utile perché:
  - L'identificatore suggerisce il significato di un valore
  - Permette di parametrizzare i programmi
    - riutilizzabili al cambiare di circostanze esterne

- *Es:* in un programma dichiaro:

```
const float PiGreco = 3.14;
```

poi uso **`PiGreco`** più volte nella parte esecutiva;

Se viene richiesto una precisione diversa devo solo modificare la dichiarazione

```
const float PiGreco = 3.1415;
```

Note: Abbreviazioni nell'Assegnamento

- Istruzioni della forma

  variabile = *variabile* ***operatore*** *espressione*

  si possono scrivere come

  variabile ***operatore*** = *espressione*

- `b = b + 7;` $\Longrightarrow$ `b +=7;` (Idem con altri operatori)
- Incrementare o decrementare una variabile di 1 è molto frequente, quindi c'è una notazione apposita
- `a = a + 1;` $\Longrightarrow$ `a++;`
- `b = b - 1;` $\Longrightarrow$ `b--;`

Note: caratteri di conversione

- Nella **`stringaControllo`** di **`printf`** è possibile specificare la formattazione quando viene stampato un valore di una variabile.
- **`"%5d"`** dedica 5 caratteri alla stampa del numero intero
- **`"%.2f"`** dedica due cifre dopo la virgola per un float

`switch-case`, la sintassi

- **`switch`**,**`case`**,**`default`** keywords
- **`int_expr`** espressione a valori integral (char o int)
- **`constant-expr1`** numero o carattere
- **`default`** opzionale

```
switch (int_expr)
   {
   case constant-expr1:
      statement1
   case constant-expr2:
      statement2
      …
   case constant-exprN:
      statementN
   [default : statement]
      }
```

`switch-case`, la sintassi

- **NB**: **`constant-expr1`** non può contenere una variabile,
- **NB**: **`int_expr`** può contenere variabili
- **NB**: a differenza di **`if`**, **`while`** e **`for`**,
  - **`int_expr`** non è un espressione booleana
  - Non occorre delimitare gli **`statement`** tra **`{}`**, anche nel caso contengano più istruzioni. Questi sono delimitati dal case seguente

```
switch (int_expr)
   {
   case constant-expr1:
      statement1
   case constant-expr2:
      statement2
      …
   case constant-exprN:
      statementN
   [default : statement]
      }
```

`switch-case`, l'esecuzione

1. Viene valutata **`expression`** (eventualmente convertita)
2. Si controlla se **`expression`** è **uguale** a **`constant-expr1`**
3. Se sono **uguali** eseguo **`statement1`**, ed in **cascata, tutti gli statement** dei **`case`** seguenti (senza verifiche, incluso lo statement di **`default`**)
4. Altrimenti controllo se **`expression`** è **uguale** a **`constant-expr2`** …
5. Eseguo lo statement di **`default`** [se presente]

```
switch (int_expr)
   {
   case constant-expr1:
      statement1
   case constant-expr2:
      statement2
      …
   case constant-exprN:
      statementN
   [default : statement]
      }
```

Note switch-case

Esempio di utilizzo di **switch**

```
scanf("%c" , &a);
switch (a)
    {case 'A': nA++;
    case 'E': nE++;
    case 'O': nO++;
    default : nCons++;}
```

- Se **a=='A'**, verranno incrementate **nA,nE,nO,nCons;**
- Se **a=='E'**, verranno incrementate **nE,nO,nCons;**
- Se **a=='O'**, verranno incrementate **nO,nCons;**
- Se **a=='K'**, verranno incrementa n**Cons;**

Note switch-case

Per evitare **l'esecuzione in cascata** alla prima corrispondenza trovata, occorre inserire negli statements opportuni la keyword **break**

```
scanf("%c" , &a);
switch (a)
    {case 'A': nA++; break;
    case 'E': nE++; break;
    case 'O': nO++; break;
    default : nCons++;}
```

- Se **a=='A'**, verrà incrementata **nA;**
- Se **a=='E'**, verrà incrementata **nE;**
- Se **a=='O'**, verrà incrementata **nO;**
- Se **a=='K'**, verrà incrementa n**Cons;**

TODO: switch-case

- Scrivere un programma che opera come una calcolatrice: richiede due operandi ed un operatore + - * / % e restituisce il risultato a schermo
- Scrivere un programma che richiede all'utente di inserire una dei numeri (controllando che questi siano positivi) fino a quando non viene inserito uno zero.
  Il programma conta quanti sono i
  - multipli di 2,
  - multipli di 4,
  - multipli di 6,
  - multipli di 8,

  Tra i numeri inseriti e stampa a schermo un istogramma (tanti asterischi quanti i valori inseriti).

Errori più comuni

Gli Errori

Gli errori possono essere di due tipi:

- Errori di sintassi, rilevabili a **compile-time**.
- Errori logici rilevabili a **run-time**.

Gli errori rilevabili a **compile-time** contengono **istruzioni che il compilatore non è in grado di risolvere**, per questo manda dei segnali di errore.

- Controllate sempre il log del compilatore!

Gli errori **a run-time** si **manifestano durante l'esecuzione** e possono causare:

- L'interruzione del programma
- Comportamenti inaspettati

Sono più difficili da rilevare e a volte è necessario entrare in modalità debug per trovarli

Errori a Compile-Time

- Dimenticare un ; manda l'errore alla riga seguente (trova due istruzioni in una sola riga)

error: expected ';' before 'printf'

- Variabile non inizializzata?

error: 'iniz_nome' undeclared (first use in this function)

- typo?

error: 'prinf' undeclared (first use in this function)

Errori a Run-Time

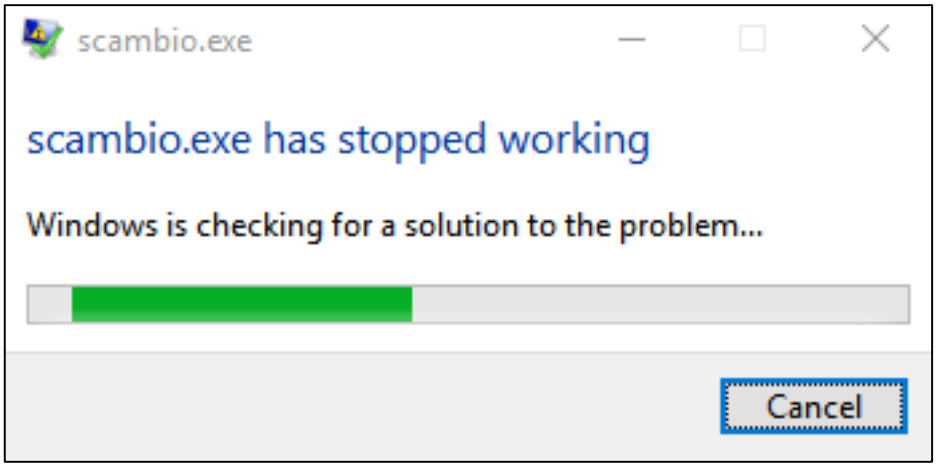
Dimenticare **`&`** nelle variabili della **`scanf`** :

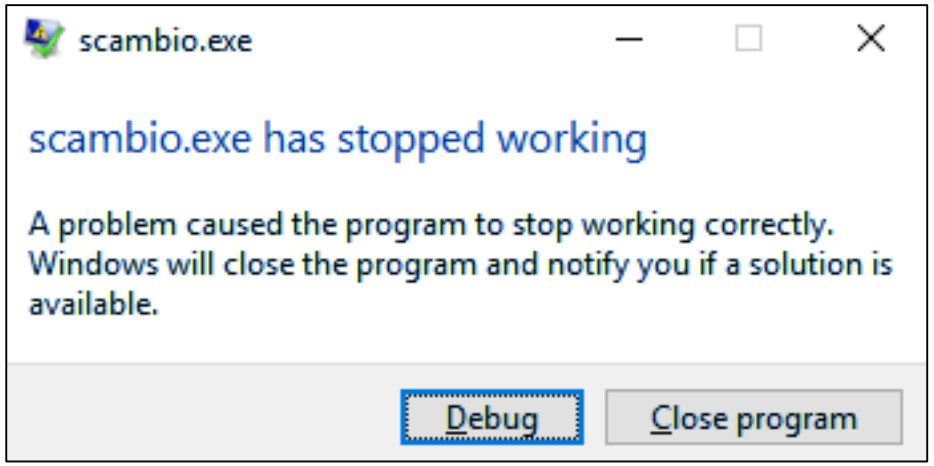
il compilatore non lo rileva! Errore rilevabile a **run-time**, quando il valore dallo stdInput viene scritto in una cella di indirizzo «sbagliato»

```
int a = 7;
```

`scanf("%d",a);` scrive nella cella all'indirizzo **`7`**.

Tipicamente se ne accorge Microsoft appena ci provate.





Errori Frequenti Rilevabili a Run-Time

Confondere l'assegnamento con il confronto

```
int a = 10;
    if(a = 7)
        printf("Vero");
    else
        printf("Falso");
```

Stampa sempre **Vero** perché **(a = 7)** è un assegnamento e l'operazione diventa **1** cioè, assegnamento eseguito con successo!

Errori Frequenti Rilevabili a Run-Time

Sbagliare lo specificatore di formato in una scanf o printf

```
void main(){
int x;
  printf("inserire x: ");
  scanf("%f", &x);
  printf("\nx = %d",x);
}
```

```
inserire x: 4

x = 1082130432
Process returned 15 (0xF)   execution time : 5.226 s
Press any key to continue.
```

Errori Frequenti Rilevabili a Run-Time

Sbagliare lo specificatore di formato in una scanf o printf

```
void main(){
float x;
  printf("inserire x: ");
  scanf("%f", &x);
  printf("\nx = %d",x);
}
```

```
inserire x: 4.98

x = 536870912
Process returned 14 (0xE)   execution time : 9.780 s
Press any key to continue.
```

E' invece possibile…

E' invece possibile stampare i char come interi

```
void main(){
char x;
  printf("inserire x: ");
  scanf("%c", &x);
  printf("\nx = '%c' = %d", x, x);
}
```

```
inserire x: a

x = 'a' = 97
Process returned 13 (0xD)   execution time : 3.579 s
Press any key to continue.
```

Error a Run-Time `i = i++;`

- L'istruzione **`i++`**

  corrisponde all'assegnamento **`i = i + 1;`**

- Non ha quindi senso **`i = i++;`** che corrisponde a

```
i = (i = i + 1);
```

Error a Run-Time: il ; nelle strutture di controllo.

Il ; termina un'istruzione e quindi non va messo dopo **if, while, for, switch,**

- Se presente, il ; specifica che il costrutto non ha corpo e l'istruzione successiva viene eseguita
- *Es*

```
int a = 10;
        if(a == 7);
                printf("Vero");
```

  - Stampa vero perché `printf("Vero");` è fuori dall' **if**
  - Se ci fosse stato un **else** il compilatore avrebbe dato errore: esiste un **else** non associato ad un **if**

- Il ; viene usato nel **do while** perché il costrutto termina con il `while(expression);`

Note sull'acquisizione di caratteri da tastiera

- Le acquisizioni di caratteri consecutivi danno problemi. In particolare, dopo uno **`scanf`**, l'invio di conferma rimane nel buffer di ingresso (stdin) e viene acquisto dal primo **`scanf("%c", &variabile);`** che segue.

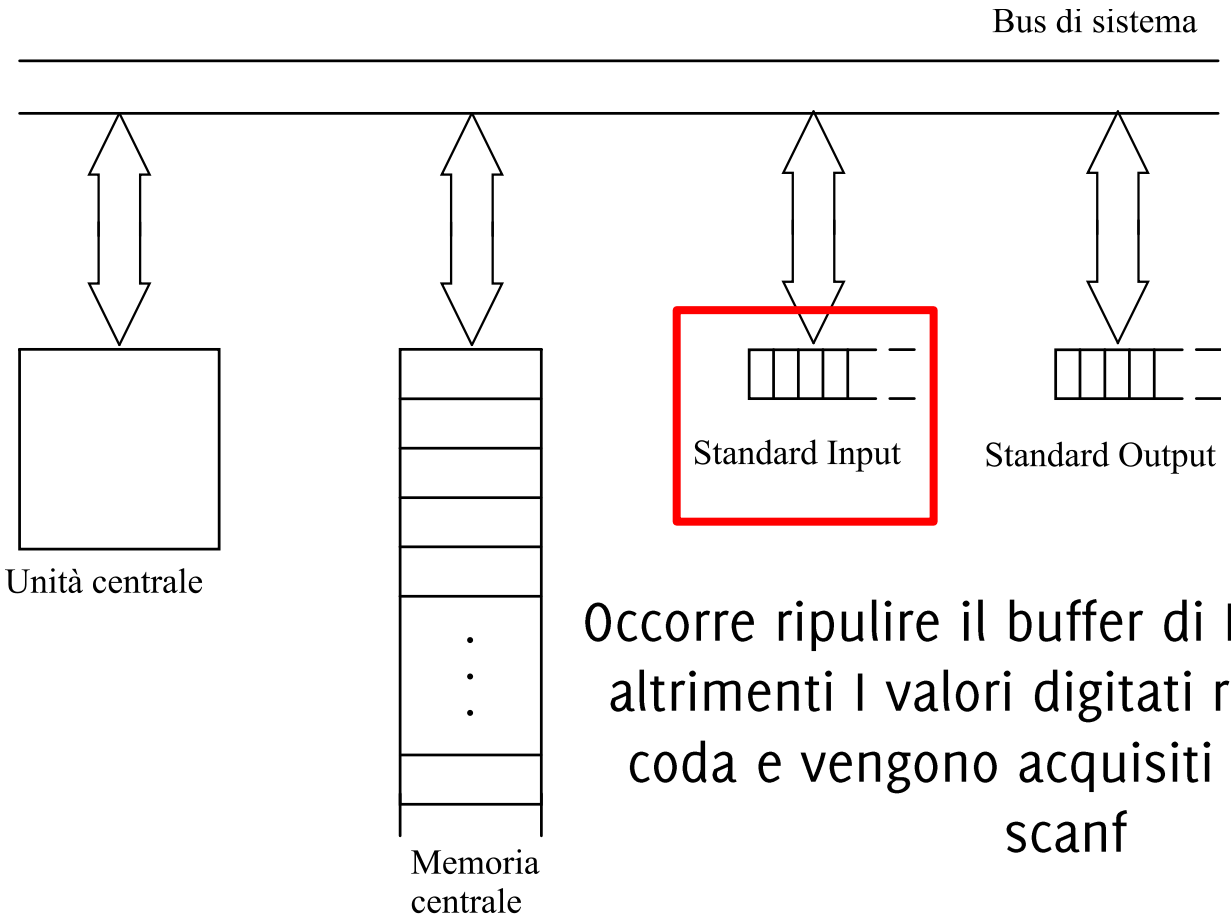
Note sull'acquisizione di caratteri da tastiera

- Le acquisizioni di caratteri consecutivi danno problemi. In particolare, dopo uno **`scanf`**, l'invio di conferma rimane nel buffer di ingresso (stdin) e viene acquisto dal primo **`scanf("%c", &variabile);`** che segue.

- Soluzioni, da mettere dopo il primo **`scanf`**
  - **`fflush(stdin);`** pulisce il buffer stdin (per windows)
  - **`scanf("%*c");`** acquisisce un secondo carattere che viene buttato via (non viene precisata la variabile di destinazione)

La Macchina C

Descrizione del linguaggio C mediante la macchina C che esegue i programmi codificati.



Occorre ripulire il buffer di Ingresso altrimenti I valori digitati rimangono in coda e vengono acquisiti al prossimo scanf

Lettura da Standard Input: `scanf`

Acquisizione di caratteri: richiede un ulteriore comando

```
char x;

scanf("%c", &x); fflush(stdin);
```

Altrimenti l'invio inserito per terminare l'acquisizione di **x** rimane in un buffer (**stdin**) e viene acquisito nella successiva acquisizione di caratteri, i.e., in eventuali **scanf("%c", &altraVariabile)** che seguono!

Lettura da Standard Input: `scanf`

Acquisizione di caratteri: richiede un ulteriore comando

```
char x;
scanf("%c", &x); fflush(stdin);
```

Altrimenti l'invio inserito per terminare l'acquisizione di **x** rimane in un buffer (**stdin**) e viene acquisito nella successiva acquisizione di caratteri, i.e., in eventuali **scanf("%c", &altraVariabile)** che seguono!

```
int main(void)
{ char a,b;
scanf("%c", &a);
fflush(stdin);// elimina tutto il buffer
scanf("%c", &b);// acquisisce da zero
printf("%c %c", a, b); }
```

fflush(stdin) e scanf("%*c");

Un'alternativa a **`fflush(stdin)`** è aggiungere un'acquisizione di un carattere "a vuoto"

```
scanf("%*c")
```

il `%*c` indica che verrà acquisito un carattere e buttato via

fflush(stdin) e scanf("%*c");

Un'alternativa a **fflush(stdin)** è aggiungere un'acquisizione di un carattere "a vuoto"

```
scanf("%*c")
```

il `%*c` indica che verrà acquisito un carattere e buttato via

```
int main(void)
{ char a,b;
scanf("%c", &a);
scanf("%*c");// acquisisce ed elimina l'invio
scanf("%c", &b);// acquisisce da zero
printf("%c %c", a, b); }
```

Confronto e Assegnamento

- L'opratore di confronto **==** non va confuso con l'operatore di assegnamento **=**
- Le loro sintassi sono simili

```
nomeVariabile == Espressione;

nomeVariabile = Espressione;
```

in entrambi i casi **Espressione** è una variabile/una costante/un valore fissato o un'espressione che coinvolge gli elementi sopra.

- Il risultato del confronto **nomeVariabile == Espressione** è **1** se **nomeVariabile** ed **Espressione** coincidono.

Esempio

```
#include<stdio.h>
void main()
{
        char a, b;
        b = '2';
        a = b == '0';
        printf("%d" , a);
}
```

- Cosa fa?

Esempio

```
#include<stdio.h>

void main()

{

        char a, b;

        b = '2';

        a = b == '0';

        printf("%d" , a);

}
```

- Cosa fa?
- Associa ad **a** il valore **1** se **b** è **0**, **1** altrimenti.
- Viene letto

```
a = (b =='0');
```

- Se **b = '2';** Stampa 0
- Se **b = '0';** Stampa 1
- Se **b = 0;** Stampa 0

Cose da non fare

- Modificare la variabile di loop nel for a mano!
- *Es*

```
for (i = 0; i < 10; i++)
    { if(i % 2 == 0)
            i++;
     printf("%d", i);}
```

- La variabile del ciclo for deve essere modificata unicamente dalla **`init_instr`** e dalla **`loop_instr`**
- Altrimenti il codice diventa di difficile interpretazione, ed è facile commettere errori.
- Piuttosto usare **`while`**