



**UNIMORE**  
UNIVERSITÀ DEGLI STUDI DI  
MODENA E REGGIO EMILIA

# Dispense per il Laboratorio di Strutture Dati e Algoritmi

Federico Bolelli

## **Esercitazione 02: Funzioni Ricorsive**

Ultimo aggiornamento: 09/03/2023

# Ricorsione - SommaFinoA

- Esercizio:

Creare i file `somma.h` e `somma.c` che consentano di utilizzare la seguente funzione:

```
extern int SommaFinoA(int n);
```

La funzione accetta come parametro un numero intero positivo  $n$  e ritorna la somma dei primi  $n$  numeri naturali (0 escluso) calcolata ricorsivamente. Nel caso che  $n$  sia minore o uguale a 0 la funzione deve ritornare -1. Se ad esempio  $n$  vale 2 la funzione deve ritornare 3 ( $1 + 2$ ).

Si scriva un opportuno `main()` di prova per testare la funzione.

# Ricorsione - SommaFinoA

- Prima di procedere con l'implementazione dobbiamo identificare la funzione matematica che descrive il problema in termini ricorsivi.
- L'obiettivo è quello di definire il problema originale in termini di problemi più semplici, identificando una o più condizioni (o casi) di base il cui risultato è noto.
- Le condizioni di base sono indispensabili affinché la definizione sia corretta.
- Il problema può essere visto come la somma di due addendi:
$$(1 + 2 + 3 + 4 + 5 + \dots + (n - 1)) \text{ e } n$$
- dove  $n$  è noto e la somma dei primi  $n - 1$  numeri è un sottoproblema di quello originale.

# Ricorsione - SommaFinoA

- Il caso base si ha quando  $n = 1$ , in quanto la somma dei numeri naturali da 1 a 1 vale proprio 1.

- La funzione matematica che descrive la soluzione è dunque:

$$f(n) = \begin{cases} 1, & n = 1 \\ f(n - 1) + n, & n > 1 \end{cases}$$

- Trovata la definizione matematica non ci resta che tradurla in codice...

# Ricorsione - SommaFinoA

```
/* somma.h */
#ifndef SOMMA_H_
#define SOMMA_H_

extern int SommaFinoA(int n);

#endif // SOMMA_H_
```

```
/* somma.c */
#include "somma.h"
int SommaFinoA(int n) {

    // Caso base
    if (n == 1) {
        return 1;
    }

    return SommaFinoA(n - 1) + n;
}
```

```
/* main.c */
#include "somma.h"

int main(void) {
    int s;
    s = SommaFinoA(1); // s = 1
    s = SommaFinoA(2); // s = 3
    s = SommaFinoA(3); // s = 6
    s = SommaFinoA(4); // s = 10
    s = SommaFinoA(5); // s = 15
    s = SommaFinoA(20); // s = 210
}
```

# Ricorsione - SommaFinoA

- Cosa manca?

# Ricorsione - SommaFinoA

- Esercizio:

Creare i file `somma.h` e `somma.c` che consentano di utilizzare la seguente funzione:

```
extern int SommaFinoA(int n);
```

La funzione accetta come parametro un numero intero positivo  $n$  e ritorna la somma dei primi  $n$  numeri naturali (0 escluso) calcolata ricorsivamente. Nel caso che  $n$  sia minore o uguale a 0 la funzione deve ritornare -1. Se ad esempio  $n$  vale 2 la funzione deve ritornare 3 ( $1 + 2$ ).

Si scriva un opportuno `main()` di prova per testare la funzione.

# Ricorsione - SommaFinoA

- Non abbiamo gestito i casi particolari!
- In questo esercizio  $n \leq 0$

```
/* main.c */
#include "somma.h"

int main(void) {
    int s;
    // ...
    s = SommaFinoA(0); // s = -1
    s = SommaFinoA(-1); // s = -1
    s = SommaFinoA(-2); // s = -1
}
```

```
/* somma.c */
#include "somma.h"

int SommaFinoA(int n) {

    // Caso particolare
    if (n <= 0) {
        return -1;
    }

    // Caso base
    if (n == 1) {
        return 1;
    }

    return SommaFinoA(n - 1) + n;
}
```



# Ricorsione - SommaFinoA

- È una buona idea ripetere la verifica dei casi particolari ad ogni invocazione ricorsiva?

# Ricorsione - SommaFinoA

- No! Se ho strutturato bene l'implementazione, dopo aver validato l'input alla prima invocazione avrò la certezza che tutte quelle successive verranno effettuate con argomenti corretti.
- Quindi come posso migliorare l'implementazione?

# Ricorsione - SommaFinoA

- Utilizzando una funzione ausiliaria!

```
/* somma.c */
#include "somma.h"

static int SommaFinoARec(int n) {

    // Caso base
    if (n == 1) {
        return 1;
    }

    return SommaFinoARec(n - 1) + n;
}
```

```
int SommaFinoA(int n) {

    // Caso particolare
    if (n < 1) {
        return -1;
    }

    return SommaFinoARec(n);
}
```

# Google C++ Style Guide

- I nomi dei file contengono sempre solo lettere minuscole ed eventualmente il carattere <trattino basso> per separare parole diverse.
- L'*#include guard* o *macro guard* contiene gli stessi caratteri del nome del file, ma convertiti maiuscolo e seguiti dai caratteri «\_H\_».
- Nomi delle funzioni sono «Title Case» e non usano mai il <trattino basso>.
- I nomi delle variabili sono sempre minuscoli e possono fare uso del carattere <trattino basso> per separare parole diverse.

```
/* somma.h */  
#ifndef SOMMA_H_  
#define SOMMA_H_  
extern int SommaFinoA(int n);  
#endif // SOMMA_H_
```

---

```
/* somma.c */  
#include "somma.h"  
  
static int SommaFinoARec(int n) {  
  
    // Caso base  
    if (n == 1) {  
        return 1;  
    }  
  
    return SommaFinoA(n - 1) + n;  
}  
// ...
```

# Google C++ Style Guide

- Si cerchi di includere solo i file che vengono realmente utilizzati.
- La { di apertura di un blocco *if*, *while*, *for* e *switch* deve essere preceduta da uno <spazio>, posizionata sulla stessa riga e seguita da un <a capo>. Lo stesso vale per la definizione di una funzione.
- Le parole chiave *if*, *while*, *for* e *switch* devono sempre essere seguite da uno spazio prima della parentesi (
- La } di chiusura dei blocchi *if*, *while*, *for* e *switch* deve essere sempre su di una nuova riga e seguita da <a capo>. Lo stesso vale per le funzioni.

```
/* somma.h */  
#ifndef SOMMA_H_  
#define SOMMA_H_  
extern int SommaFinoA(int n);  
#endif // SOMMA_H_
```

```
/* somma.c */  
#include "somma.h"  
  
int SommaFinoARec(int n) {  
  
    // Caso base  
    if (n == 1) {  
        return 1;  
    }  
  
    return SommaFinoA(n - 1) + n;  
  
    // ...  
}
```

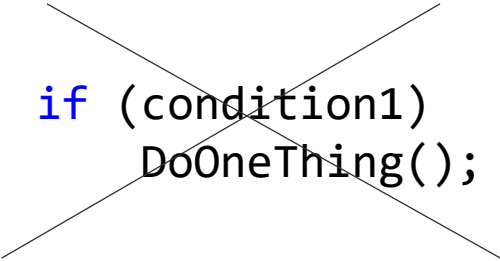
# Google C++ Style Guide

- In caso di *if* e *else* annidati la formattazione da preferire è la seguente:

```
if (condition1) {  
    DoOneThing();  
} else if (condition2) {  
    DoAnotherThing();  
} else {  
    DoNothing();  
}
```

- Il **mio** consiglio è di racchiudere sempre tra { } le istruzioni del blocco, anche quando non imposto dal linguaggio:

```
if (condition1) {  
    DoOneThing();  
}
```



```
if (condition1)  
    DoOneThing();
```