

 UNIVERSITÀ  
DEGLI STUDI DI BARI  
ALDO MORO

Corso di Laurea in Informatica e Tecnologie per la Produzione del Software (*Track B*) - A.A. 2017/2018

# Laboratorio di Informatica

## Programmazione Modulare

(Parte 2)

docente: Veronica Rossano

[Veronica.rossano@uniba.it](mailto:Veronica.rossano@uniba.it)

## Programmazione Modulare

- **Funzioni e Procedure sono un meccanismo di astrazione**
  - Le funzioni sono un esempio di astrazione sui dati, perchè ci permettono di estendere gli operatori disponibili nel linguaggio

```

65 // programma principale
66 int main() {
67     float BMI[5] = {0.0}; // dichiaro l'array dei BMI
68
69     acquireInput(BMI,DIMENSION); // metodo di acquisizione dell'input
70     float avg = calculateAverage(BMI,DIMENSION); // metodo per il calcolo della media
71     int max = calculateMaximum(BMI,DIMENSION); // metodo per il calcolo del massimo
72     printOutput(BMI, DIMENSION, avg, max); // Stampa dell'output
73 }
```

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      3

## Programmazione Modulare

- **Funzioni e Procedure sono un meccanismo di astrazione**
  - Le funzioni sono un esempio di astrazione sui dati, perchè ci permettono di estendere gli operatori disponibili nel linguaggio
  - Le procedure sono un esempio di astrazione sulle istruzioni, perchè ci permettono di estendere le istruzioni primitive disponibili nel linguaggio

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      2

## Programmazione Modulare

- **Funzioni e Procedure sono un meccanismo di astrazione**
  - Le funzioni sono un esempio di astrazione sui dati, perchè ci permettono di estendere gli operatori disponibili nel linguaggio

```

65 // programma principale
66 int main() {
67     float BMI[5] = {0.0}; // dichiaro l'array dei BMI
68
69     acquireInput(BMI,DIMENSION); // metodo di acquisizione
70     float avg = calculateAverage(BMI,DIMENSION); //
71     int max = calculateMaximum(BMI,DIMENSION); // met
72     printOutput(BMI, DIMENSION, avg, max); // Stampa
73 }
```

Le funzioni introducono nuovi meccanismi per elaborare i dati. Prendono in input dei dati e ne producono altri.

Così come altre funzioni che abbiamo già usato, es: `sqrt(99.0)` oppure `isdigit('a')`

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      4

## Programmazione Modulare

- **Funzioni e Procedure sono un meccanismo di astrazione**
  - Le procedure sono un esempio di astrazione sulle istruzioni, perchè ci permettono di estendere le istruzioni primitive disponibili nel linguaggio

```

65 // programma principale
66 int main() {
67     float BMI[5] = {0.0}; // dichiaro 1
68
69     acquireInput(BMI,DIMENSION); // mett
70     float avg = calculateAverage(BMI,DI
71     int max = calculateMaximum(BMI,DIME
72     printOutput(BMI, DIMENSION, avg, ma
73 }

```

Le procedure non introducono nuovi dati, ma introducono **nuove istruzioni** non originariamente disponibili nel linguaggio.

Le procedure tipicamente si concentrano sulla gestione dell'input/output, così come printf() o puts()

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

5

## Programmazione Modulare – Classi di memoria

- La **classe di memoria** di un identificatore determina:
  - La sua **permanenza in memoria (statica o automatica)**
    - Periodo durante il quale l'identificatore **esiste in memoria**

Veronica Rossano - Programmazione Modulare (Parte 2)  
Laboratorio di Informatica (INF, Track B) – Università degli Studi  
di Bari – A.A. 2018/2019

6

## Programmazione Modulare – Classi di memoria

- La **classe di memoria** di un identificatore determina:
  - La sua **permanenza in memoria (statica o automatica)**
    - Periodo durante il quale l'identificatore **esiste in memoria**
  - Il suo **campo di azione (scope in inglese)**
    - È **dove** l'identificatore **può essere menzionato** in un programma

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2)  
Laboratorio di Informatica (INF, Track B) – Università degli Studi  
di Bari – A.A. 2018/2019

7

## Programmazione Modulare – Classi di memoria

- La **classe di memoria** di un identificatore determina:
  - La sua **permanenza in memoria (statica o automatica)**
    - Periodo durante il quale l'identificatore **esiste in memoria**
  - Il suo **campo di azione (scope in inglese)**
    - È **dove** l'identificatore **può essere menzionato** in un programma
  - Il suo **collegamento**
    - Determina, quando il programma è composto da più file, **se** l'identificatore **è conosciuto solo nel file sorgente corrente** o in **qualunque** file sorgente ad esso collegato

Veronica Rossano - Programmazione Modulare (Parte 2)  
Laboratorio di Informatica (INF, Track B) – Università degli Studi  
di Bari – A.A. 2018/2019

8

## Programmazione Modulare – Permanenza in memoria

- In Linguaggio C esistono degli specificatori per «definire» la permanenza in memoria di una variabile

- Auto
- Extern
- Static
- Register

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

9

## Programmazione Modulare - Permanenza in memoria

- In Linguaggio C esistono degli specificatori per «definire» la permanenza in memoria di una variabile

- Auto
  - Valore di default per le variabili locali (non serve indicarlo). Imposta la permanenza in memoria pari al ciclo di vita del blocco.

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

10

## Programmazione Modulare - Permanenza in memoria

- In Linguaggio C esistono degli specificatori per «definire» la permanenza in memoria di una variabile

- Auto
  - Valore di default per le variabili locali (non serve indicarlo). Imposta la permanenza in memoria pari al ciclo di vita del blocco.
- Extern
  - Valore di default per le variabili globali. La permanenza in memoria è pari all'intera esecuzione del file.

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

11

## Programmazione Modulare - Permanenza in memoria

- In Linguaggio C esistono degli specificatori per «definire» la permanenza in memoria di una variabile

- Auto
  - Valore di default per le variabili locali (non serve indicarlo). Imposta la permanenza in memoria pari al ciclo di vita del blocco.
- Extern
  - Valore di default per le variabili globali. La permanenza in memoria è pari all'intera esecuzione del file.
- Static
  - La permanenza in memoria è pari all'esecuzione di una funzione, ma il suo valore non viene distrutto quando termina l'esecuzione della funzione

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

12

## Programmazione Modulare - Permanenza in memoria

- In Linguaggio C esistono degli speciatori per «definire» la permanenza in memoria di una variabile**
  - Auto**
    - Valore di default per le variabili locali (non serve indicarlo). **Imposta la permanenza in memoria pari al ciclo di vita del blocco.**
  - Extern**
    - Valore di default per le variabili globali. **La permanenza in memoria è pari all'intera esecuzione del file.**
  - Static**
    - La permanenza in memoria è pari all'esecuzione di una funzione, **ma il suo valore non viene distrutto quando termina l'esecuzione della funzione**
  - Register (NB OBSOLETO, non utilizzato)**
    - Memorizza la variabile in registri ad alta velocità, per velocizzarne l'accesso. Utile per variabili cui si accede spesso, es. i contatori

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

13

## Programmazione Modulare - Permanenza in memoria

- In Linguaggio C esistono degli speciatori per «definire» la permanenza in memoria di una variabile**

### Nota bene:

Le variabili globali può causare effetti secondari non voluti quando una funzione che non ha necessità di accedere alla variabile al modifica accidentalmente.

In generale, le variabili globali vanno evitate

- Auto**
  - Valore di default per le variabili locali (non serve indicarlo). **Imposta la permanenza in memoria pari al ciclo di vita del blocco.**
- Extern**
  - Valore di default per le variabili globali. **La permanenza in memoria è pari all'esecuzione del file.**
- Static**
  - La permanenza in memoria è pari all'esecuzione di una funzione, **ma il suo valore non viene distrutto quando termina l'esecuzione della funzione**
- Register (NB OBSOLETO, non utilizzato)**
  - Memorizza la variabile in registri ad alta velocità, per velocizzarne l'accesso. Utile per variabili cui si accede spesso, es. i contatori

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

14

## Speciatori - Esempio

```

12 int main() {
13     // i = variabile register|
14     for register int i=1; i<=10; i++) // ciclo
15     {
16         f(i); // chiamo una funzione
17     }
18 }
```

Come viene passato il parametro?

Specificatore utile per le variabili cui si accede spesso. Velocizza l'accesso inserendole nei registri. Obsoleto!

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

16

## Speciatori - Esempio

```

12 int main() {
13     // i = variabile register|
14     for register int i=1; i<=10; i++) // ciclo
15     {
16         f(i); // chiamo una funzione
17     }
18 }
```

Parametro passato per valore.

Specificatore utile per le variabili cui si accede spesso. Velocizza l'accesso inserendole nei registri. Obsoleto!

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

17

## Specificatori - Esempio

```

1* int f(int x) {
2    static int a = 0; // variabile statica
3    int b = 0; // variabile locale
4
5    a++; // incremento le variabili
6    b++;
7
8    // stampo i valori
9    printf("Chiamata n.%d \ta=%d \tb=%d\n", x, a, b);
10   }

```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

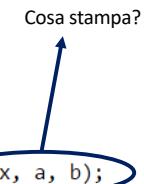
18

## Specificatori - Esempio

```

1* int f(int x) {
2    static int a = 0; // variabile statica
3    int b = 0; // variabile locale
4
5    a++; // incremento le variabili
6    b++;
7
8    // stampo i valori
9    printf("Chiamata n.%d \ta=%d \tb=%d\n", x, a, b);
10   }

```



03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

19

## Specificatori – Esempio (Output)

```
Chiamata n.1      a=1      b=1
```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

20

## Specificatori – Esempio (Output)

```
Chiamata n.1      a=1      b=1
Chiamata n.2      a=2      b=1
```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

21

## Specificatori – Esempio (Output)

```

Chiamata n.1    a=1    b=1
Chiamata n.2    a=2    b=1
Chiamata n.3    a=3    b=1
Chiamata n.4    a=4    b=1
Chiamata n.5    a=5    b=1
Chiamata n.6    a=6    b=1
Chiamata n.7    a=7    b=1
Chiamata n.8    a=8    b=1
Chiamata n.9    a=9    b=1
Chiamata n.10   a=10   b=1
  
```

**a** è una **variabile statica** che **resta in memoria**, quindi ad ogni invocazione della funzione il valore continua a essere incrementato.

**b** è un parametro **passato per valore**, quindi il suo valore **viene azzerato** (e inizializzato) ad ogni invocazione della funzione **f**.

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

22

## Programmazione Modulare - Scope

- **Lo scope (visibilità) di una variabile è il frammento di codice in cui una variabile è «visibile»**
  - **Visibile** → è nota al compilatore e può essere utilizzata nel codice senza produrre errori.

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

23

## Programmazione Modulare - Scope

- **Lo scope (visibilità) di una variabile è il frammento di codice in cui una variabile è «visibile»**
  - **Visibile** → è nota al compilatore e può essere utilizzata nel codice senza produrre errori.
- **Regola Generale di Scope**
  - Una variabile è visibile **nel blocco in cui viene definita e in tutti i blocchi innestati**, a meno di ridefinizioni
  - Sulla base di questa regola si definiscono quattro **diverse tipologie di scope**

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

24

## Programmazione Modulare - Scope

- **Tipologie di scope**
  - File Scope
  - Function Scope
  - Block Scope
  - Function Prototype Scope

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

25

## Programmazione Modulare - Scope

- **Tipologie di scope**

- **File Scope**

- Tutti gli identificatori definiti fuori dalle funzioni **sono visibili in tutto il file**.
- Si usa per i **prototipi di funzione** (che devono poter essere richiamati in ogni momento) e le **variabili globali** (che sono utilizzabili e visibili in tutto il codice sorgente)

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

26

## Programmazione Modulare - Scope

- **Tipologie di scope**

- **File Scope**

- Tutti gli identificatori definiti fuori dalle funzioni **sono visibili in tutto il file**.
- Si usa per i **prototipi di funzione** (che devono poter essere richiamati in ogni momento) e le **variabili globali** (che sono utilizzabili e visibili in tutto il codice sorgente)

- **Function Scope**

- Gli identificatori definiti nel corpo di una funzione sono visibili solo in quella funzione (a meno di ridefinizioni in un blocco)
- Si usa per le **variabili (locali)**.

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

27

## Programmazione Modulare - Scope

- **Tipologie di scope**

- **File Scope**

- Tutti gli identificatori definiti fuori dalle funzioni **sono visibili in tutto il file**.
- Si usa per i **prototipi di funzione** (che devono poter essere richiamati in ogni momento) e le **variabili globali** (che sono utilizzabili e visibili in tutto il codice sorgente)

- **Function Scope**

- Gli identificatori definiti nel corpo di una funzione sono visibili solo in quella funzione (a meno di ridefinizioni in un blocco)
- Si usa per le **variabili (locali)**.

- **Block Scope**

- Tutti gli identificatori **definiti in un blocco**, sono visibili solo in quel blocco;

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

28

## Programmazione Modulare - Scope

- **Tipologie di scope**

- **File Scope**

- Tutti gli identificatori definiti fuori dalle funzioni **sono visibili in tutto il file**.
- Si usa per i **prototipi di funzione** (che devono poter essere richiamati in ogni momento) e le **variabili globali** (che sono utilizzabili e visibili in tutto il codice sorgente)

- **Function Scope**

- Gli identificatori definiti nel corpo di una funzione sono visibili solo in quella funzione (a meno di ridefinizioni in un blocco)
- Si usa per le **variabili (locali)**.

- **Block Scope**

- Tutti gli identificatori **definiti in un blocco**, sono visibili solo in quel blocco;

- **Function Prototype Scope**

- Gli identificatori definiti nel prototipo di una funzione valgono solo in esso.

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

29

## Programmazione Modulare - Scope

```
void function(int var1);
int var2;

int main() {
    int var3=0;

    if(var3==0) {
        int var4 = 1;
    }
}
```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

30

## Programmazione Modulare - Scope

```
void function(int var1); // var1 = ???
int var2; // var2 = ???

int main() {
    int var3=0; // var3 = ???

    if(var3==0) {
        int var4 = 1; // var4 = ???
    }
}
```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

31

## Programmazione Modulare - Scope

```
void function(int var1); // var1 = visibilità nel prototipo
int var2; // var2 = visibilità globale

int main() {
    int var3=0; // var3 = visibilità locale

    if(var3==0) {
        int var4 = 1; // var4 = visibilità nel blocco
    }
}
```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

32

## Programmazione Modulare - Scope

```
void function(int var1); // var1 = visibilità nel prototipo
int var2; // var2 = visibilità globale

int main() {
    int var3=0; // var3 = visibilità locale
    var2=1; // è una istruzione valida
    if(var3==0) {
        int var4 = 1; // var4 = visibilità nel blocco
        var3 = 2;
    }
    var1 = 3;
    var4 = 4;
}
```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

33

## Programmazione Modulare - Scope

```
void function(int var1); // var1 = visibilità nel prototipo
int var2; // var2 = visibilità globale

int main() {
    int var3=0; // var3 = visibilità locale
    var2=1; // si, var2 è globale
    if(var3==0) {
        int var4 = 1; // var4 = visibilità nel blocco
        var3 = 2;
    }
    var1 = 3;
    var4 = 4;
}
```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

34

## Programmazione Modulare - Scope

```
void function(int var1); // var1 = visibilità nel prototipo
int var2; // var2 = visibilità globale

int main() {
    int var3=0; // var3 = visibilità locale
    var2=1; // si, var2 è globale
    if(var3==0) {
        int var4 = 1; // var4 = visibilità nel blocco
        var3 = 2; // è una istruzione valida?
    }
    var1 = 3;
    var4 = 4;
}
```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

35

## Programmazione Modulare - Scope

```
void function(int var1); // var1 = visibilità nel prototipo
int var2; // var2 = visibilità globale

int main() {
    int var3=0; // var3 = visibilità locale
    var2=1; // si, var2 è globale
    if(var3==0) {
        int var4 = 1; // var4 = visibilità nel blocco
        var3 = 2; // si, var3 è locale
    }
    var1 = 3;
    var4 = 4;
}
```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

36

## Programmazione Modulare - Scope

```
void function(int var1); // var1 = visibilità nel prototipo
int var2; // var2 = visibilità globale

int main() {
    int var3=0; // var3 = visibilità locale
    var2=1; // si, var2 è globale
    if(var3==0) {
        int var4 = 1; // var4 = visibilità nel blocco
        var3 = 2; // si, var3 è locale
    }
    var1 = 3;
    var4 = 4;
}
```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

37

## Programmazione Modulare - Scope

```
void function(int var1); // var1 = visibilità nel prototipo
int var2; // var2 = visibilità globale

int main() {
    int var3=0; // var3 = visibilità locale
    var2=1; // si, var2 è globale
    if(var3==0) {
        int var4 = 1; // var4 = visibilità nel blocco
        var3 = 2; // si, var3 è locale
    }
    var1 = 3; // errore, var1 è visibile solo nel prototipo
    var4 = 4; // errore, var4 è visibile solo nel blocco
}
```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

38

## Programmazione Modulare - Scope

```
void function(int var1); // var1 = visibilità nel prototipo
int var2; // var2 = visibilità globale

int main() {
    int var3=0; // var3 = visibilità locale
    var2=1; // si, var2 è globale
    if(var3==0) {
        int var4 = 1; // var4 = visibilità nel blocco
        var3 = 2; // si, var3 è locale
    }
    var1 = 3; // errore, var1 è visibile solo nel prototipo
    int var4 = 4; ////
}
```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

39

## Programmazione Modulare - Scope

```
void function(int var1); // var1 = visibilità nel prototipo
int var2; // var2 = visibilità globale

int main() {
    int var3=0; // var3 = visibilità locale
    var2=1; // si, var2 è globale
    if(var3==0) {
        int var4 = 1; // var4 = visibilità nel blocco
        var3 = 2; // si, var3 è locale
    }
    var1 = 3; // errore, var1 è visibile solo nel prototipo
    int var4 = 4; // OK! Perché una volta terminato il blocco variale var4 può essere ri-definita.
}
```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

Ma non lo fate!!!

40

## Programmazione modulare - suggerimenti

- È importante utilizzare il principio dell'**Information Hiding o occultamento delle informazioni**

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

41

## Programmazione modulare - suggerimenti

- È importante utilizzare il principio dell'**Information Hiding o occultamento delle informazioni**

Ogni oggetto, per motivi di sicurezza del software e per facilitare la risoluzione di errori, necessita della protezione dei dati a un livello più «interno» e meno «esposto»

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2)  
Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

42

## Programmazione modulare - suggerimenti

- È importante utilizzare il principio dell'**Information Hiding o occultamento delle informazioni**

Ogni oggetto, per motivi di sicurezza del software e per facilitare la risoluzione di errori, necessita della protezione dei dati a un livello più «interno» e meno «esposto»

**Il principio del minimo privilegio**, al codice deve essere garantita soltanto la quantità di privilegi e di accessi **necessaria** per eseguire il suo compito designato, ma non di più

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2)  
Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

43

## Progettazione Modulare

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

44

## Progettazione Modulare

- L'utilizzo delle funzioni e delle procedure **risolve parzialmente il problema della programmazione modulare**
- **Perché?**

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

45

## Progettazione Modulare

- L'utilizzo delle funzioni e delle procedure **risolve parzialmente il problema della programmazione modulare**
- **Perché?**
  - Il codice sorgente è comunque tutto aggregato in un unico file (anche se «spacchettato» in diverse funzioni).
  - Per implementare totalmente i principi della programmazione modulare è necessario anche dividere «**fisicamente**» il codice sorgente
- **Come?**

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

46

## Progettazione Modulare

- L'utilizzo delle funzioni e delle procedure **risolve parzialmente il problema della programmazione modulare**
- **Perché?**
  - Il codice sorgente è comunque tutto aggregato in un unico file (anche se «spacchettato» in diverse funzioni).
  - Per implementare totalmente i principi della programmazione modulare è necessario anche dividere «**fisicamente**» il codice sorgente
- **Come?**
  - Header Files
  - Librerie Statiche

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

47

## Progettazione Modulare

	<b>Header Files</b>	<b>Librerie Statiche</b>
Repl.it	SI	NO
Eclipse	SI	SI

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

48

## Progettazione Modulare

	<b>Header Files</b>	<b>Librerie Statiche</b>
Repl.it	SI	NO
Eclipse	SI	SI

**Eclipse** da la possibilità di implementare la programmazione modulare sia attraverso l'utilizzo delle **librerie statiche** che attraverso la definizione degli **header files**.

**Repl.it** permette solo di dividere il codice sorgente in più file più piccoli, utilizzando gli **header files**

Entrambe le modalità sono accettate.

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

49

## Progettazione Modulare – Header Files

- Gli header files sono ‘files di intestazione’ (tradotto letteralmente)
  - Hanno estensione .h (esempio: functions.h)
  - Sono accompagnati da un file .c con lo stesso nome (functions.c)
  - Contengono le intestazioni delle funzioni e delle procedure che vogliamo separare dal programma principale

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

50

## Progettazione Modulare – Header Files

- Gli header files sono ‘files di intestazione’ (tradotto letteralmente)
  - Hanno estensione .h (esempio: functions.h)
  - Sono accompagnati da un file .c con lo stesso nome (functions.c)
  - Contengono le intestazioni delle funzioni e delle procedure che vogliamo separare dal programma principale
- Si creano uno o più nuovi files e si inseriscono le funzioni in questi files
  - Le funzioni vengono aggregate in diversi header files, in base al loro scopo (es. tutte le funzioni che si occupano di input/output, tutte le funzioni per operazioni matematiche, etc.)
  - Un po’ come avviene nelle funzioni della libreria standard del C (es. <string.h> <ctype.h> etc.)
- Come cambia la struttura dei programmi?

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

51

## Progettazione Modulare – Header Files

```
void f(int x); // prototipo
int f(int x) { // funzione
    static int a = 0;
    int b = 0;
    printf("Ciclo %d:%d\t%d\n", x,a,b)
}
int main() { // main
    for(int i=1; i<10; i++) {
        f(i); // invocazione
    }
}
```

main.c

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

52

## Progettazione Modulare – Header Files

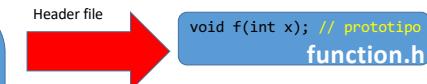
```
void f(int x); // prototipo
int f(int x) { // funzione
    static int a = 0;
    int b = 0;
    printf("Ciclo %d:%d\t%d\n", x,a,b)
}
int main() { // main
    for(int i=1; i<10; i++) {
        f(i); // invocazione
    }
}
```

main.c

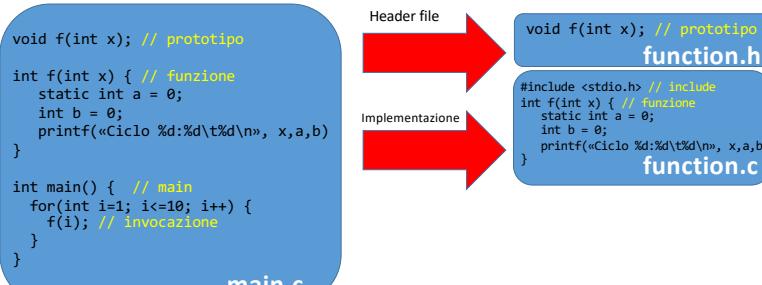
03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

53



## Progettazione Modulare – Header Files

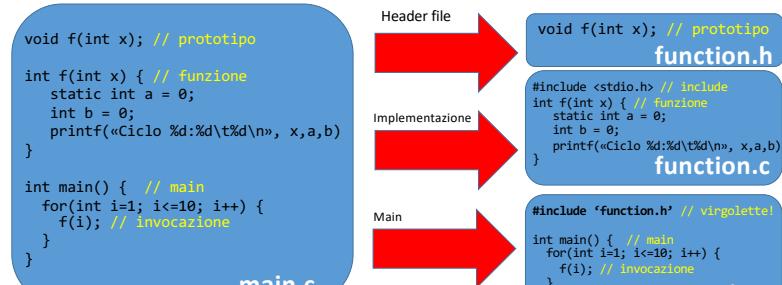


03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

54

## Progettazione Modulare – Header Files

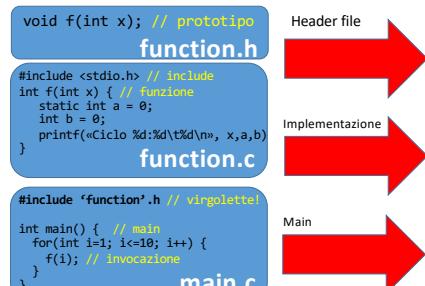


03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

55

## Progettazione Modulare – Header Files

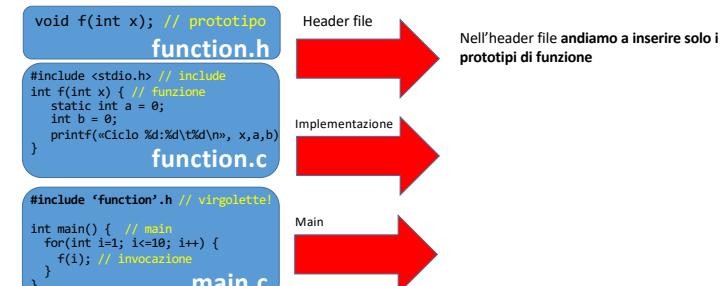


03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

56

## Progettazione Modulare – Header Files



03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

57

## Progettazione Modulare – Header Files

```
void f(int x); // prototipo
function.h

#include <stdio.h> // include
int f(int x) { // funzione
    static int a = 0;
    int b = 0;
    printf("Ciclo %d:%d\t%d\n", x,a,b)
} function.c

#include 'function'.h // virgolette!
int main() { // main
    for(int i=1; i<=10; i++) {
        f(i); // invocazione
    }
} main.c
```



Nell'header file **andiamo a inserire solo i prototipi di funzione**

Nel file di implementazione (**con lo stesso nome!**) inseriamo **l'implementazione dei prototipi**. Se necessario, i file di implementazione possono avere a loro volta delle direttive `#include`



Main

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

58

## Progettazione Modulare – Header Files

```
void f(int x); // prototipo
function.h

#include <stdio.h> // include
int f(int x) { // funzione
    static int a = 0;
    int b = 0;
    printf("Ciclo %d:%d\t%d\n", x,a,b)
} function.c

#include 'function'.h// virgolette!
int main() { // main
    for(int i=1; i<=10; i++) {
        f(i); // invocazione
    }
} main.c
```



Nell'header file **andiamo a inserire solo i prototipi di funzione**

Nel file di implementazione (**con lo stesso nome!**) inseriamo **l'implementazione dei prototipi**. Se necessario, i file di implementazione possono avere a loro volta delle direttive `#include`



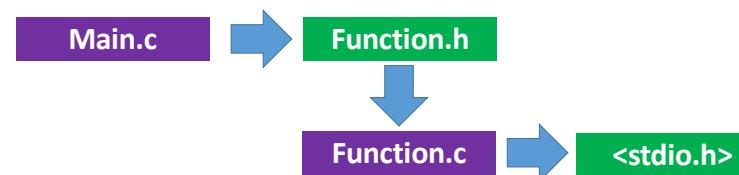
Main

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

59

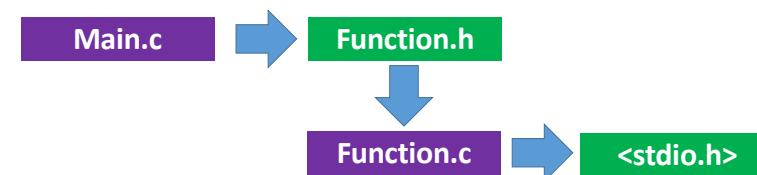
## Progettazione Modulare – Header Files



Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

60

## Progettazione Modulare – Header Files



L'utilizzo degli header files cambia la struttura dei programmi. Il **main** invoca le funzioni implementate in **function.h**, che a sua volta nella sua implementazione ha bisogno delle funzioni della **libreria standard**.

Il processo di esecuzione dei programmi **può essere rappresentato attraverso un grafo aciclico** (perché non ci possono essere dipendenze «circolari» tra librerie)

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

61

## Progettazione Modulare – Header Files

```

graph LR
    Main[Main.c] --> FunctionH[Function.h]
    FunctionH --> FunctionC[Function.c]
    FunctionC --> StdioH[<stdio.h>]
    
```

**Grafo = insieme di elementi e archi che li collegano. Gli elementi si dicono «nodi».**

L'utilizzo degli header files cambia la struttura dei programmi. Il **main** invoca le funzioni implementate in **function.h**, che a sua volta nella sua implementazione ha bisogno delle funzioni della **libreria standard**.

Il processo di esecuzione dei programmi **può essere rappresentato attraverso un grafo aclico** (perché non ci possono essere dipendenze «circolari» tra librerie)

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      62

Credits: Corrado Mencar

## Progettazione Modulare – Un Caso Reale

Source: <http://scottmcpeak.com/elkhound/>

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      63

Credits: Corrado Mencar

## Progettazione Modulare – Header Files

- Ogni header file è diviso in sette parti**
  - Prologo
  - Guardia
  - Direttive di inclusione
  - Costanti
  - Tipi di Dato
  - Variabili Globali
  - Prototipi di Funzione

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      64

Credits: Corrado Mencar

## Progettazione Modulare – Header Files

- Ogni header file è diviso in sette parti**
  - Prologo
    - Si tratta di un «commento», descrive a cosa serve il file e che tipo di informazioni contiene
    - Autori, Data, Numero di Versione, Riferimenti e Link esterni, Eventuali informazioni su License e Copyright**

```

/*
 * CUnit - A Unit testing
 * framework library for C.
 * Copyright (C) 2004-2006 Jerry
 * St.Clair
 *
 * This library is free software; ...
 * License as published by the
 * Free Software Foundation;
 ...
 * 11-Aug-2004 Initial
 * implementation of basic test
 * runner interface. (JDS)
 */

```

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      65

Credits: Corrado Mencar

## Progettazione Modulare – Header Files

- Ogni header file è diviso in sette parti**

- Guardia
  - La struttura a «grafo» delle dipendenze può portare ad avere dipendenze multiple

```

graph TD
    header1[header1.h] <--> header2[header2.h]
    header1 --> sorgente[sorgente.c]
  
```

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      66

Credits: Corrado Mencar

## Progettazione Modulare – Header Files

- Ogni header file è diviso in sette parti**

- Guardia
  - La struttura a «grafo» delle dipendenze può portare ad avere dipendenze multiple
  - La guardia serve a inserire delle condizioni nella definizione delle informazioni che sono contenute nell'header file, per evitare di fornire definizioni multiple.**

```

graph TD
    header1[header1.h] <--> header2[header2.h]
    header1 --> sorgente[sorgente.c]
  
```

```

#ifndef CUNIT_BASIC_H_SEEN
#define CUNIT_BASIC_H_SEEN
... codice ...
#endif
  
```

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      67

Thanks to Corrado Mencar

Credits: Corrado Mencar

## Progettazione Modulare – Header Files

- Ogni header file è diviso in sette parti**

- Guardia
  - La struttura a «grafo» delle dipendenze può portare ad avere dipendenze multiple
  - La guardia serve a inserire delle condizioni nella definizione delle informazioni che sono contenute nell'header file, per evitare di fornire definizioni multiple.**

```

graph TD
    header1[header1.h] <--> header2[header2.h]
    header1 --> sorgente[sorgente.c]
  
```

```

#ifndef CUNIT_BASIC_H_SEEN
#define CUNIT_BASIC_H_SEEN
... codice ...
#endif
  
```

*(Se il file non è già stato incluso, continua a leggere)*

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      68

Credits: Corrado Mencar

## Progettazione Modulare – Header Files

- Ogni header file è diviso in sette parti**

- Inclusioni
  - Un modulo può implementare delle funzioni che a loro volta abbiano bisogno di altre librerie
  - Nel nostro esempio includevamo `<stdio.h>` per poter utilizzare l'istruzione `printf()`.

```

/* System includes */
#include <stdio.h>
#include <stdlib.h>

/* Local includes */
// Unit test framework
#include "CUnit/Basic.h"
// libreria di manipolazione
matrici
#include "math/matrix.h"
  
```

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      69

## Progettazione Modulare – Header Files

- Ogni header file è diviso in sette parti**

4. Definizioni

- Un header file deve includere anche le definizioni di costanti e macro.
- E' possibile anche qui usare `#ifndef` per evitare ridefinizioni di costanti già definite nel programma

```

#define CU_VERSION "2.1-2"

#define CU_MAX_TEST_NAME_LENGTH 256

#ifndef CU_TRUE
#define CU_TRUE 1
#endif

#define CU_MAX(a,b)
((a) >= (b)) ? (a) : (b))

```

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

03/04/19 70

## Progettazione Modulare – Header Files

- Ogni header file è diviso in sette parti**

5. Tipi di dato

- Un header file deve includere anche le definizioni di tipi di dato
- Come abbiamo visto, e' utile definire nuovi tipi di dato con il comando `typedef`. Tutti i nuovi tipi di dato sono da includere nell'header file.
- Utilità
  - Astrazione
  - Leggibilità

```

typedef enum {
    CU_BRM_NORMAL = 0,
    CU_BRM_SILENT,
    CU_BRM_VERBOSE
} CU_BasicRunMode;

```

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

03/04/19 71

## Progettazione Modulare – Header Files

- Ogni header file è diviso in sette parti**

6. Variabili

- Le variabili definite negli header file hanno visibilità globale, sono visibili da tutte le funzioni o da tutti i moduli.
- Per alcuni problemi può essere utile avere delle variabili globali accessibili da tutti e visibili a tutti.

```

int global_variable;
// visibile a livello
// globale. EVITARE

static int mod_var;
// visibile solo
// a livello di
// modulo

```

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

03/04/19 72

## Progettazione Modulare – Header Files

- Ogni header file è diviso in sette parti**

7. Prototipi di funzione

- A livello minimale, un header file deve contenere almeno un prototipo di funzione

```

CU_ErrorCode CU_basic_run_suite(CU_pSuite pSuite);
/**<
 * Runs all tests for a specific suite in the
 * basic interface...

```

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

03/04/19 73

18



Progettazione Modulare – Header Files (in Repl)

```

Files saved share run
main.c
1 #include "stdio.h"
2 #include "function.h" // includo la mia funzione
3
4 int main(void) {
5     for (int i=0; i<10; i++) {
6         f(i); // invoco la funzione f(i) ad ogni ciclo
7     }
8
9     return 0;
10 }
```

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      76

Progettazione Modulare – Header Files (in Repl)

Crea nuovi file sorgente nel progetto

```

Files saved share run
main.c
1 #include "stdio.h"
2 #include "function.h" // includo la mia funzione
3
4 int main(void) {
5     for (int i=0; i<10; i++) {
6         f(i); // invoco la funzione f(i) ad ogni ciclo
7     }
8
9     return 0;
10 }
```

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      77

### Progettazione Modulare – Header Files (in Repl)



Crea nuovi file sorgente nel progetto

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      78

### Progettazione Modulare – Header Files (in Repl)



03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      79

### Progettazione Modulare – Header Files (in Repl)



03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      80

### Progettazione Modulare – Header Files (in Repl)



Cosa stampa in output?

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      81

### Progettazione Modulare – Header Files (in Repl)



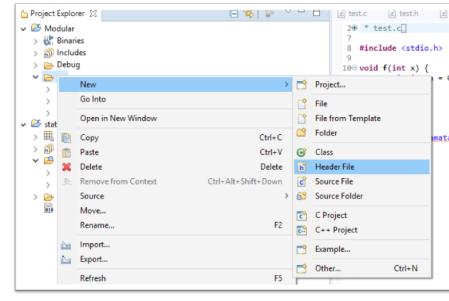
```
function.c
1 // implementazione della funzione
2 #include <stdio.h>
3
4 void f(int i) {
5     static int a = 10; // variabile statica
6
7     printf("%d\t%d\n", a++, i); // stampa il valore di a e di i
8 }
9
```

gcc version 4.6.3

10 0  
11 1  
12 2  
13 3  
14 4  
15 5  
16 6  
17 7  
18 8  
19 9

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      82

### Progettazione Modulare – Header Files (Eclipse)



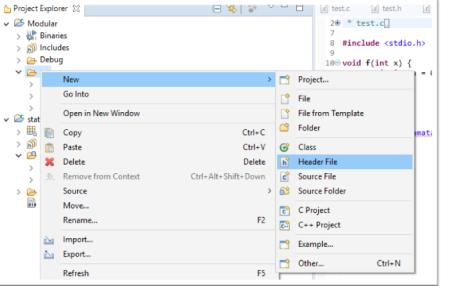
**Creazione dell'header file**

**Creazione Progetto**  
 → Tasto Destro sulla cartella dei sorgenti  
 → New  
 → Header File

**(importante, non dimenticare l'estensione del file!)**

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      83

### Progettazione Modulare – Header Files (Eclipse)



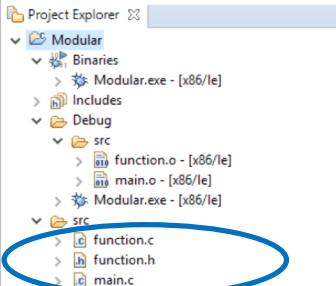
**Creazione del file implementazione**

**Creazione Progetto**  
 → Tasto Destro sulla cartella dei sorgenti  
 → New  
 → Source File

**(importante, non dimenticare l'estensione del file!)**

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      84

### Progettazione Modulare – Header Files (Eclipse)

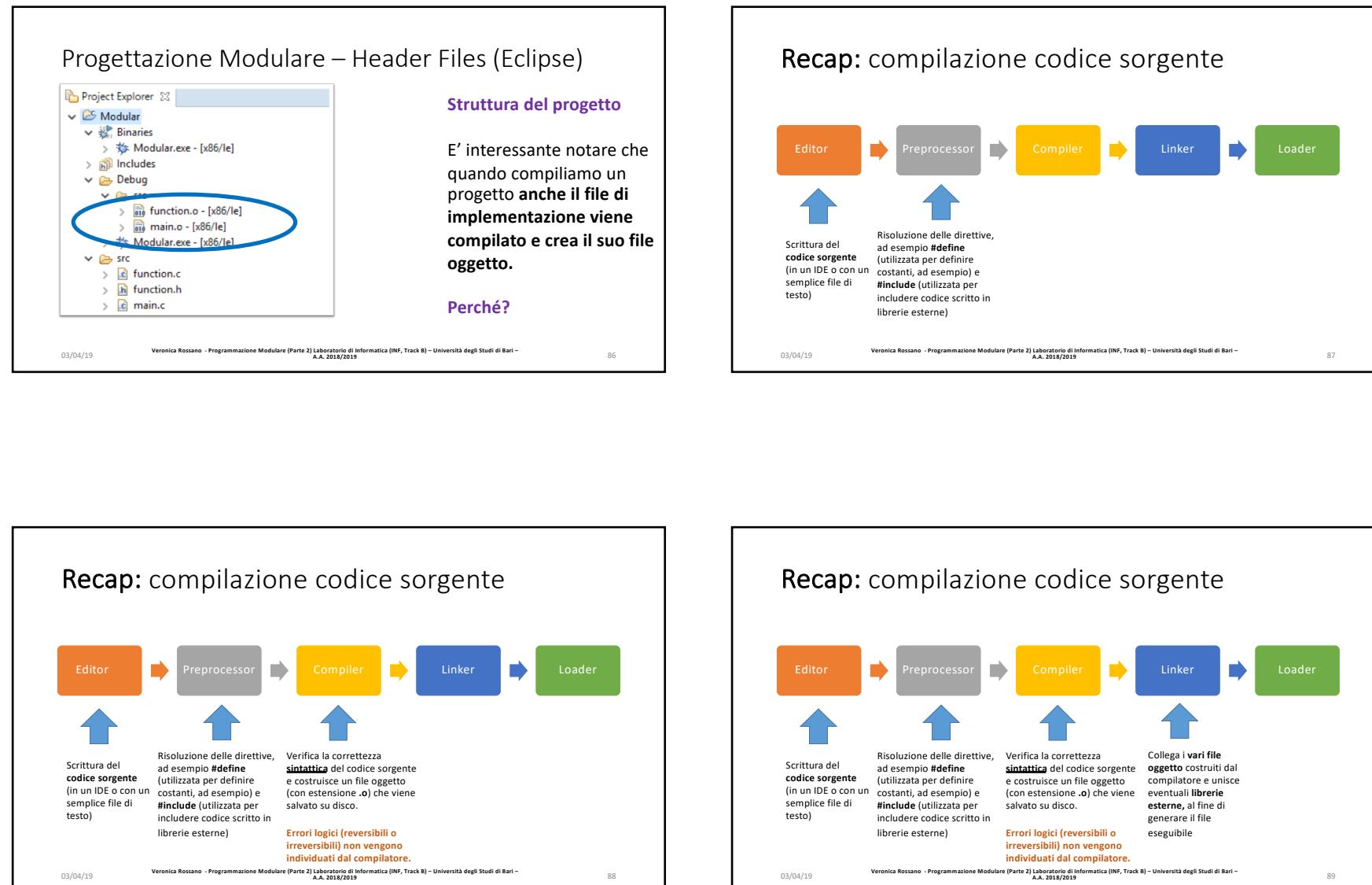


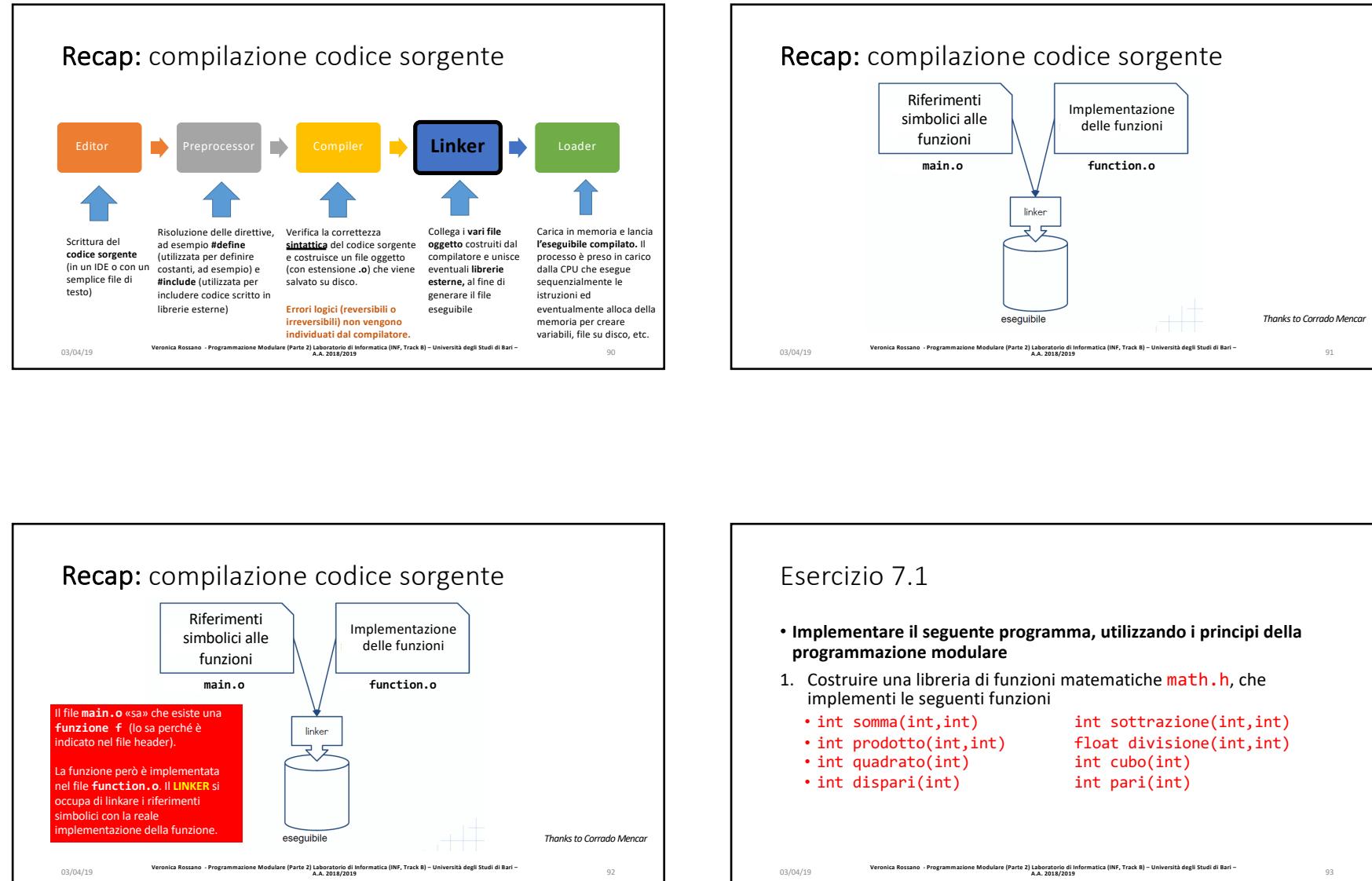
**Struttura del progetto**

Modular  
 Binaries  
 Includes  
 Debug  
 src  
 SRC

function.c  
 function.h  
 main.c

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      85





## Esercizio 7.1

- Implementare il seguente programma, utilizzando i principi della programmazione modulare
2. Invocare la libreria in un file **main.c**, che generi **random** due numeri **a** e **b** e calcoli il valore della seguente espressione (attenti all'utilizzo delle parentesi e alla priorità tra gli operatori)
 
$$(a^2 * b) - (a + b^3)$$
  3. Stampare un messaggio diverso a seconda che il risultato sia pari o dispari, utilizzando la funzione implementata
  4. Implementare la soluzione sia su Repl o su Eclipse (attraverso la divisione in file **.h** e **.c**). **Implementare controlli sull'input! Attenzione a commenti e stile di programmazione.**

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

94

## Esercizio 7.1 (Soluzione)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "math.h" // // importo la libreria appena creata
#define SIZE 64 // fisso la dimensione massima dell'input

int main( ) {
    char input_A[SIZE] = {0}; // memorizza l'input
    char input_B[SIZE] = {0}; // memorizza l'input
    char* remainingInput_A = NULL; // serve a memorizzare quello che resta dell'input
    char* remainingInput_B = NULL; // serve a memorizzare quello che resta dell'input
    int inputCorrect = 0; // serve a uscire dal ciclo appena l'input è corretto
```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

95

## Esercizio 7.1 (Soluzione)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "math.h" // // importo la libreria appena creata
#define SIZE 64 // fisso la dimensione massima dell'input

int main( ) {
    char input_A[SIZE] = {0}; // memorizza l'input
    char input_B[SIZE] = {0}; // memorizza l'input
    char* remainingInput_A = NULL; // serve a memorizzare quello che resta dell'input
    char* remainingInput_B = NULL; // serve a memorizzare quello che resta dell'input
    int inputCorrect = 0; // serve a uscire dal ciclo appena l'input è corretto
```

Leggo delle stringhe,  
non degli interi!

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

96

## Esercizio 7.1 (Soluzione – cont.)

```
do {
    printf("\nInserire due valori numerici in input, separati da spazio: ");
    scanf("%20s %20s",input_A, input_B);

    int a = (int) strtol(input_A, &remainingInput_A, 10);
    int b = (int) strtol(input_B, &remainingInput_B, 10);

    // l'input è correto quando la parte rimanente della stringa è vuota
    if( strcmp(remainingInput_A, "") == 0 && strcmp(remainingInput_B, "") == 0 )
        inputCorrect = 1;

    // calcolo l'espressione (a^2*b)-(a+b^3) e stampo il risultato
    if ( inputCorrect ) {
        int result = sottrazione( prodotto( quadrato(a), b), somma( a, cubo(b)));
        printf("Risultato: %d", result);
    }
} while( inputCorrect == 0 ); // cicla finchè l'input non è corretto
```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

97

## Esercizio 7.1 (Soluzione – cont.)

```

do {
    printf("\nInserire due valori numerici in input, separati da spazio: ");
    scanf("%20s %20s",input_A, input_B);

    int a = (int) strtol(input_A, &remainingInput_A, 0);
    int b = (int) strtol(input_B, &remainingInput_B, 0); → Utilizzo strtol() per convertire
    // l'input è corretto quando la parte rimanente della stringa è vuota
    if( strcmp(remainingInput_A, "") == 0 && strcmp(remainingInput_B, "") == 0 )
        inputCorrect = 1;

    // calcolo l'espressione (a^2*b)-(a+b^3) e stampo il risultato
    if ( inputCorrect ) {
        int result = sottrazione( prodotto( quadrato(a), b), somma( a, cubo(b)));
        printf("Risultato: %d", result);
    }
} while( inputCorrect == 0 ); // cicla finchè l'input non è corretto
}

```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

98

Utilizzo `strtol()` per convertire la stringa in intero (serve il cast perché restituisce un long)

## Esercizio 7.1 (Soluzione – cont.)

```

do {
    printf("\nInserire due valori numerici in input, separati da spazio: ");
    scanf("%20s %20s",input_A, input_B);

    int a = (int) strtol(input_A, &remainingInput_A, 0);
    int b = (int) strtol(input_B, &remainingInput_B, 0); → La parte numerica viene
    // l'input è corretto quando la parte rimanente della stringa è vuota
    if( strcmp(remainingInput_A, "") == 0 && strcmp(remainingInput_B, "") == 0 )
        inputCorrect = 1;

    // calcolo l'espressione (a^2*b)-(a+b^3) e stampo il risultato
    if ( inputCorrect ) {
        int result = sottrazione( prodotto( quadrato(a), b), somma( a, cubo(b)));
        printf("Risultato: %d", result);
    }
} while( inputCorrect == 0 ); // cicla finchè l'input non è corretto
}

```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

99

La parte numerica viene memorizzata in `a` oppure `b`, la parte rimanente viene memorizzata nella stringa `remainingInput`

## Esercizio 7.1 (Soluzione – cont.)

```

do {
    printf("\nInserire due valori numerici in input, separati da spazio: ");
    scanf("%20s %20s",input_A, input_B);

    int a = (int) strtol(input_A, &remainingInput_A, 0); → Esempio:
    int b = (int) strtol(input_B, &remainingInput_B, 0); → Input_A = «456aa»
    // l'input è corretto quando la parte rimanente della stringa è vuota
    if( strcmp(remainingInput_A, "") == 0 && strcmp(remainingInput_B, "") == 0 )
        inputCorrect = 1;

    // calcolo l'espressione (a^2*b)-(a+b^3) e stampo il risultato
    if ( inputCorrect ) {
        int result = sottrazione( prodotto( quadrato(a), b), somma( a, cubo(b)));
        printf("Risultato: %d", result);
    }
} while( inputCorrect == 0 ); // cicla finchè l'input non è corretto
}

```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

100

Esempio:  
Input\_A = «456aa»  
A = 456  
remainingInput\_A = «aa»

## Esercizio 7.1 (Soluzione – cont.)

```

do {
    printf("\nInserire due valori numerici in input, separati da spazio: ");
    scanf("%20s %20s",input_A, input_B);

    int a = (int) strtol(input_A, &remainingInput_A, 0); → Esempio:
    int b = (int) strtol(input_B, &remainingInput_B, 0); → Input_A = «118»
    // l'input è corretto quando la parte rimanente della stringa è vuota
    if( strcmp(remainingInput_A, "") == 0 && strcmp(remainingInput_B, "") == 0 )
        inputCorrect = 1;

    // calcolo l'espressione (a^2*b)-(a+b^3) e stampo il risultato
    if ( inputCorrect ) {
        int result = sottrazione( prodotto( quadrato(a), b), somma( a, cubo(b)));
        printf("Risultato: %d", result);
    }
} while( inputCorrect == 0 ); // cicla finchè l'input non è corretto
}

```

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

101

Esempio:  
Input\_A = «118»  
A = 118  
remainingInput\_A = «»

## Esercizio 7.1 (Soluzione – cont.)

```

do {
    printf("\nInserire due valori numerici in input, separati da spazio: ");
    scanf("%20s %20s",input_A, input_B);

    int a = (int) strtol(input_A, &remainingInput_A, 0);
    int b = (int) strtol(input_B, &remainingInput_B, 0);

    // l'input è corretto quando la parte rimanente della stringa è vuota
    if( strcmp(remainingInput_A, "") == 0 && strcmp(remainingInput_B, "") == 0 )
        inputCorrect = 1;

    // calcolo l'espressione (a^2*b)-(a+b^3) e stampo il risultato
    if ( inputCorrect ) {
        int result = sottrazione( prodotto( quadrato(a), b), somma( a, cubo(b)));
        printf("Risultato: %d", result);
    }
} while( inputCorrect == 0 ); // cicla finchè l'input non è corretto
}

```

Se remainingInput è «vuota», allora vuole dire che quello che ho letto è corretto!

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

102

## Esercizio 7.1 (Soluzione – cont.)

```

do {
    printf("\nInserire due valori numerici in input, separati da spazio: ");
    scanf("%20s %20s",input_A, input_B);

    int a = (int) strtol(input_A, &remainingInput_A, 0);
    int b = (int) strtol(input_B, &remainingInput_B, 0);

    // l'input è corretto quando la parte rimanente della stringa è vuota
    if( strcmp(remainingInput_A, "") == 0 && strcmp(remainingInput_B, "") == 0 )
        inputCorrect = 1;

    // calcolo l'espressione (a^2*b)-(a+b^3) e stampo il risultato
    if ( inputCorrect ) {
        int result = sottrazione( prodotto( quadrato(a), b), somma( a, cubo(b)));
        printf("Risultato: %d", result);
    }
} while( inputCorrect == 0 ); // cicla finchè l'input non è corretto
}

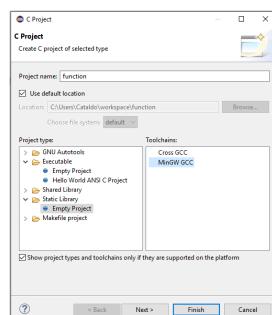
```

Se l'input è corretto, posso calcolare il valore dell'espressione e stamparlo

03/04/19 Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

103

## Progettazione Modulare – Librerie Statiche (in Eclipse)



### Modalità alternativa

per lo sviluppo di librerie esterne

**New → C Project → Static Library  
→ Empty Project**

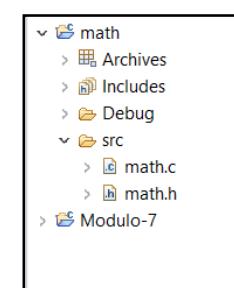
Dare un nome e selezionare ‘**MinGW GCC**’ come Toolchains.

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

104

## Progettazione Modulare – Librerie Statiche (in Eclipse)



### Creare una cartella in cui inserire i file .h e .c

Tasto dx sul progetto → New → Source Folder → dare un nome (es. src)

03/04/19 Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

105

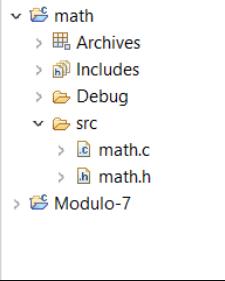
### Progettazione Modulare – Librerie Statiche (in Eclipse)

**Creare una cartella in cui inserire i file .h e .c**

Tasto dx sul progetto → New → Source Folder → dare un nome (es. src)

Si implementano i file .h e .c esattamente come nel caso precedente.

Terminata l'implementazione, si fa il «build» della libreria statica (simbolo del martello)

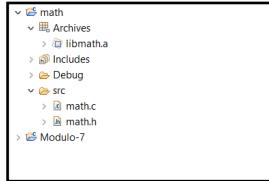


03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      106

### Progettazione Modulare – Librerie Statiche (in Eclipse)

Il build genera un file .a (archive) che rappresenta la libreria statica.

La libreria statica si può integrare in altri progetti



03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      107

### Progettazione Modulare – Librerie Statiche (in Eclipse)

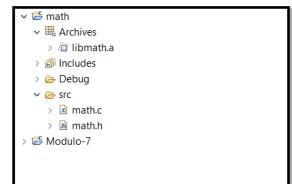
Il build genera un file .a (archive) che rappresenta la libreria statica.

La libreria statica si può integrare in altri progetti

Come?

- 1) Si copia la libreria statica nel progetto
- 2) Si aggiungono i riferimenti al file .h

In questo caso non la libreria viene sviluppata in modo totalmente separato dagli altri file, e si integra semplicemente «copiando» il file archivio all'interno del progetto.

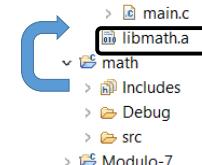


03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      108

### Progettazione Modulare – Librerie Statiche (in Eclipse)

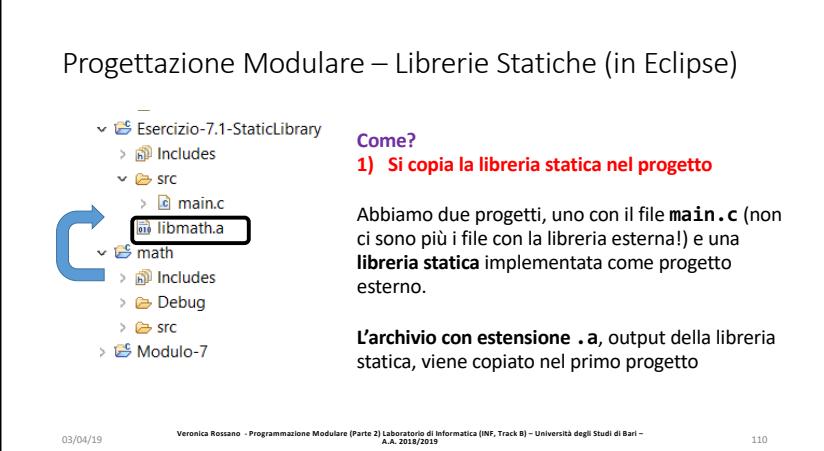
Come?

- 1) Si copia la libreria statica nel progetto



03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      109

### Progettazione Modulare – Librerie Statiche (in Eclipse)



**Come?**

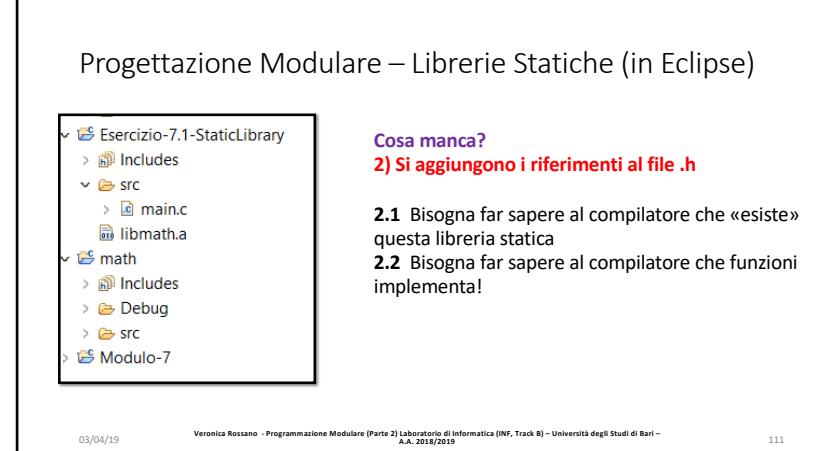
**1) Si copia la libreria statica nel progetto**

Abbiamo due progetti, uno con il file **main.c** (non ci sono più i file con la libreria esterna!) e una **libreria statica** implementata come progetto esterno.

L'archivio con estensione **.a**, output della libreria statica, viene copiato nel primo progetto

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      110

### Progettazione Modulare – Librerie Statiche (in Eclipse)



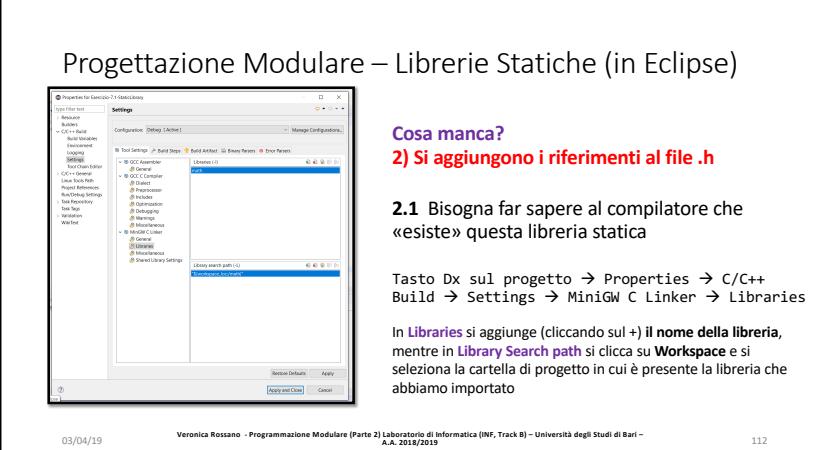
**Cosa manca?**

**2) Si aggiungono i riferimenti al file .h**

**2.1** Bisogna far sapere al compilatore che «esiste» questa libreria statica  
**2.2** Bisogna far sapere al compilatore che funzioni implementa!

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      111

### Progettazione Modulare – Librerie Statiche (in Eclipse)



**Cosa manca?**

**2) Si aggiungono i riferimenti al file .h**

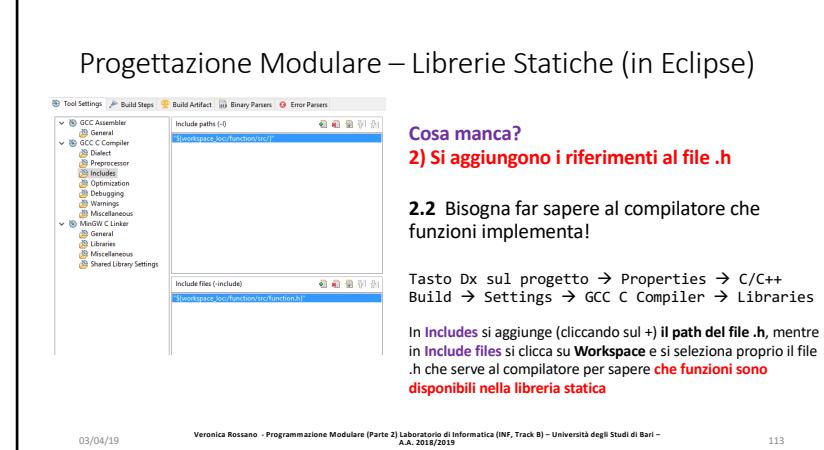
**2.1** Bisogna far sapere al compilatore che «esiste» questa libreria statica

Tasto Dx sul progetto → Properties → C/C++ Build → Settings → MiniGW C Linker → Libraries

In **Libraries** si aggiunge (cliccando sul +) **Il nome della libreria**, mentre in **Library Search path** si clicca su **Workspace** e si seleziona la cartella di progetto in cui è presente la libreria che abbiamo importato

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      112

### Progettazione Modulare – Librerie Statiche (in Eclipse)



**Cosa manca?**

**2) Si aggiungono i riferimenti al file .h**

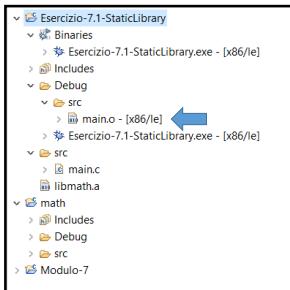
**2.2** Bisogna far sapere al compilatore che funzioni implementa!

Tasto Dx sul progetto → Properties → C/C++ Build → Settings → GCC C Compiler → Libraries

In **Includes** si aggiunge (cliccando sul +) **Il path del file .h**, mentre in **Include files (-include)** si clicca su **Workspace** e si seleziona proprio il file .h che serve al compilatore per sapere **che funzioni sono disponibili nella libreria statica**

03/04/19      Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019      113

## Progettazione Modulare – Librerie Statiche (in Eclipse)



**Terminato il processo di linking manuale della libreria, è possibile eseguire normalmente il progetto.**

**Importante:** in questo caso non abbiamo due file .o , ma un unico file (il main). Questo avviene perché gli altri file sono inclusi nella libreria che abbiamo precedentemente linkato.

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

114

**DOMANDE?**



03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2) Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

115

## Esercizio 7.2

- Partendo dalla rivisitazione dell'Esercitazione 1 creare un le librerie per le funzioni definite.
- Sottomettere il risultato sulla piattaforma di e-learning

03/04/19

Veronica Rossano - Programmazione Modulare (Parte 2)  
Laboratorio di Informatica (INF, Track B) - Università degli Studi  
di Bari - A.A. 2018/2019

116