# Esercitazione Studenti – C + IA con Debug su gdb online

### Focus: Spec → Code → Test (manuale) → Debug (solo gdb online)

---

## 0) Obiettivo

Usare l'IA come **supporto controllato** per sviluppare una mini-libreria in C e verificare il comportamento tramite: - compilazione con warning; - test manuali; - debug **solo con gdb online**.

Consegna finale: sorgenti + report con prompt e risultati.

---

# 1) Esempio introduttivo: "mini-libreria" in C

In C una "libreria" (in questo contesto) è un insieme di funzioni riutilizzabili organizzate in: - un file **header** `.h` (prototipi e tipi), - un file **sorgente** `.c` (implementazioni), - un file `main.c` che usa la libreria.

## 1.1 Esempio minimo: libreria `add`

Copia questi 3 file nel tuo gdb online e verifica che compili.

**File: add.h**

```c
#ifndef ADD_H
#define ADD_H

int add(int a, int b);

#endif
```

**File: add.c**

```c
#include "add.h"

int add(int a, int b) {
    return a + b;
}
```

**File: main.c**

```c
#include <stdio.h>
#include "add.h"

int main(void) {
    int r = add(2, 3);
    printf("r=%d\n", r);
    return 0;
}
```

## 1.2 Compilazione (importante)

Compila sempre con: - standard C99 - warning attivi - simboli di debug per gdb

Comando:

```
gcc -std=c99 -Wall -Wextra -Wpedantic -O0 -g main.c add.c -o app
```

Esecuzione:

```
./app
```

## 1.3 Debug rapido con gdb

Avvio:

```
gdb ./app
```

Comandi minimi: - `break main` - `run` - `next` (passo senza entrare) - `step` (entra nella funzione) - `print r` - `quit`

---

# 2) Esercitazione principale: Mini-libreria `stats`

## 2.1 Specifica (traccia funzionale)

Realizzare una mini-libreria `stats` per array di interi `int` con funzioni:

1. `int stats_min(const int *v, int n, int *out);`

2. `int stats_max(const int *v, int n, int *out);`
3. `int stats_sum_ll(const int *v, int n, long long *out);`
4. `int stats_mean(const int *v, int n, double *out);`
5. `int stats_variance_population(const int *v, int n, double *out);`

### Regole obbligatorie

• Standard: **C99**
• Ritorno: `0` se OK, `!=0` se errore
• Errori da gestire:
• `v == NULL`
• `out == NULL`
• `n <= 0`
• `sum_ll`: accumulo in `long long`
• `variance_population`: varianza di popolazione = media dei quadrati degli scarti (divisore `n`)
• Le funzioni non devono stampare nulla
• Niente input da tastiera (`scanf` vietato)

---

## 2.2 Flusso obbligatorio con IA (da documentare nel report)

### Fase A – Generazione specifica tecnica

Produrre una specifica testuale con: - firme delle funzioni - pre/post-condizioni - codici di errore - complessità - note overflow/precisione

### Fase B – Generazione codice

Generare 3 file: - `stats.h` - `stats.c` - `main.c` (che chiama le funzioni e stampa risultati per test manuali)

### Fase C – Generazione piano di test

Un elenco di casi con: - input - output atteso - motivazione

### Fase D – Test manuale

Eseguire in `main.c` almeno 6 casi e confrontare a mano “atteso vs ottenuto”.

### Fase E – Debug (gdb online)

Se un caso fallisce: debug con breakpoint e stampa variabili.

---

# 3) Prompt modello (da copiare e incollare nel report)

## Prompt A – Specifica tecnica

```
Genera una specifica tecnica in italiano per una mini-libreria C99 chiamata
stats.
Deve includere: prototipi, pre/post-condizioni, codici errore, complessità e
note su overflow/precisione.
Funzioni: min, max, somma long long, media double, varianza popolazione double
su array di int.
Vincoli: portabile, nessuna dipendenza esterna, ogni funzione valida v/out/n.
Formato: sezioni numerate.
```

## Prompt B – Codice

```
Genera codice completo C99 per stats.h, stats.c, main.c.
Compilazione con gcc: -std=c99 -Wall -Wextra -Wpedantic -O0 -g.
Regole: codici errore coerenti, out scritto solo se OK.
Varianza popolazione: (1/n) * somma( (xi - mean)^2 ).
In main.c: esegui casi di prova e stampa risultati in modo leggibile (senza
scanf).
```

## Prompt C – Piano test

```
Genera un piano di test per stats (min, max, sum_ll, mean, variance_population).
Includi: casi normali, negativi, n==1, input invalidi (NULL, n<=0), valori
grandi per sum_ll.
Per ogni test: input, output atteso, motivazione.
```

## Prompt D – Debug assistito (solo analisi, non copia-incolla)

```
Ti fornisco un problema: un test manuale fallisce. Analizza possibili cause e
dimmi dove mettere breakpoint in gdb.
Non riscrivere tutto il codice: suggerisci controlli e variabili da osservare.
```

---

# 4) Compilazione ed esecuzione nel gdb online

## 4.1 Compilazione

Eseguire:

```
gcc -std=c99 -Wall -Wextra -Wpedantic -O0 -g main.c stats.c -o app -lm
```

Note: - `-O0` evita ottimizzazioni che rendono più difficile il debug - `-g` abilita simboli per gdb - `-lm` serve se usate funzioni matematiche come `sqrt` (opzionale)

## 4.2 Esecuzione

```
./app
```

---

# 5) Debug operativo (solo gdb online)

Avvio:

```
gdb ./app
```

Comandi essenziali (sequenza tipica): 1. `break main` 2. `run` 3. `next` finché arrivi alla chiamata che fallisce 4. `step` per entrare nella funzione `stats_*` 5. `print n` 6. `print v[0]` 7. `print i` (se sei nel ciclo) 8. `print mean` / `print acc` (se stai calcolando media/varianza) 9. `backtrace` se c'è crash 10. `quit`

### Breakpoint mirati

Se sai dove è il problema: - `break stats_variance_population` - `break stats_mean` - `break stats_sum_ll`

### Watchpoint (se disponibile)

Utile per vedere quando una variabile cambia: - `watch i` - `watch acc`

---

# 6) Test manuali minimi (obbligatori)

Scegli almeno questi casi nel tuo `main.c` e stampa i risultati:

1) `v={1,2,3,4}` - min=1, max=4, sum=10, mean=2.5, var_pop=1.25

2) `v={-5,-1,-3}` - min=-5, max=-1, sum=-9, mean=-3.0

3) `v={7}` - mean=7.0, var_pop=0.0

4) caso errore: `v=NULL` - rc != 0

5) caso errore: `out=NULL` - rc != 0

6) caso errore: `n<=0` - rc != 0

7) somma grande: `v={1000000000,1000000000,1000000000}` - sum=3000000000 (necessario long long)

---

# 7) Consegna

Consegnare: - `stats.h`, `stats.c`, `main.c` - `report.txt` con: - prompt A/B/C (incollati) - piano di test (tabella atteso/ottenuto) - eventuali bug trovati e come li hai risolti - comandi gdb usati (breakpoint, variabili osservate)

---

# 8) Estensione facoltativa

1) Aggiungere `stats_stddev_population` usando `sqrt`. 2) Aggiungere varianza campionaria (divisore `n-1`) e commentare la differenza. 3) Migliorare robustezza: evitare scrittura su `out` in caso di errore.

---

### Fine documento