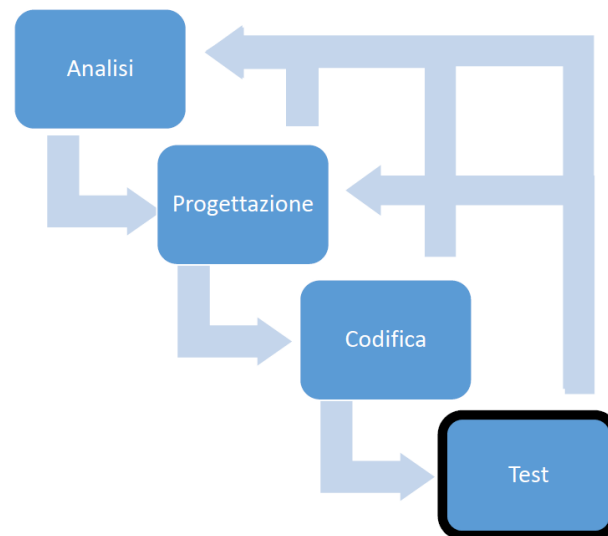


Debugging di un programma

Testing di un Programma

- Si tratta di parte integrante del processo di sviluppo del software, in cui si verifica con **approcci sistematici, oggettivi e ripetibili la correttezza** di un programma.
- Si distingue dal **debugging**, in cui si rimuovono errori logici (bugs) emersi durante il **testing**.

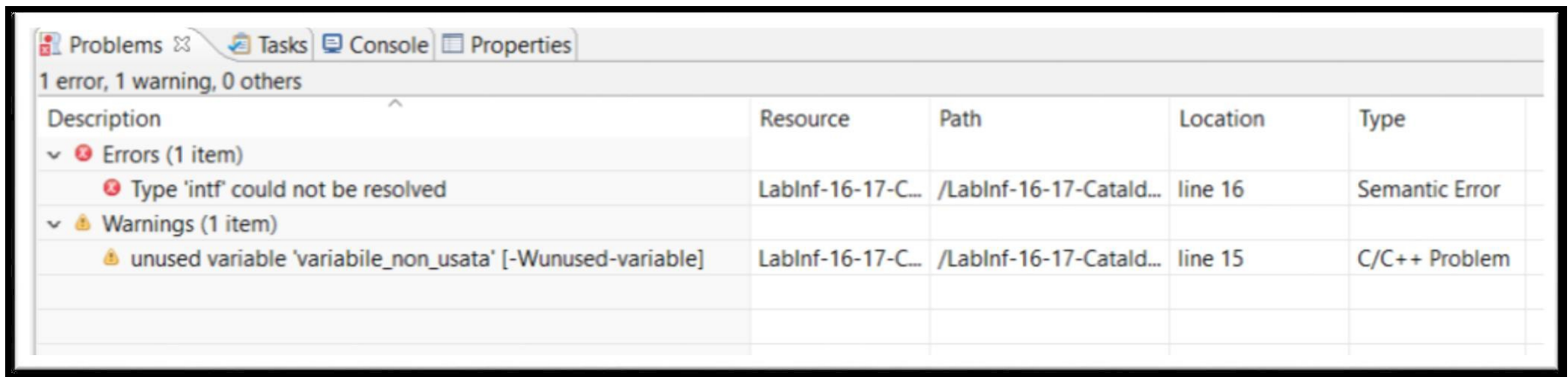


Generalità

- Processo di identificazione e rimozione di errori logici/semantici (bugs)
- Strutturato in tre fasi, una successive all'altra
 1. trovare istruzioni che causano il bug
 - fase che richiede maggiore sforzo.
 - spesso supportata da tools
 2. scoprire il motivo del bug
 3. correggere le istruzioni
- Nella fase 1, la prima attività è mettere in pratica alcune line guida.

Linee guida

- Analizzare messaggi di avvertimento (warning) emessi da compilatori o IDE tools.
- uninitialized variable
- no return



The screenshot shows the 'Problems' window in Visual Studio. It displays a summary of '1 error, 1 warning, 0 others'. Below this, there are two expandable sections: 'Errors (1 item)' and 'Warnings (1 item)'. The 'Errors' section shows a 'Semantic Error' where the type 'intf' could not be resolved. The 'Warnings' section shows a 'C/C++ Problem' for an unused variable 'variabile_non_usata'.

Description	Resource	Path	Location	Type
✖ Errors (1 item)				
✖ Type 'intf' could not be resolved	LabInf-16-17-C...	/LabInf-16-17-Catald...	line 16	Semantic Error
⚠ Warnings (1 item)				
⚠ unused variable 'variabile_non_usata' [-Wunused-variable]	LabInf-16-17-C...	/LabInf-16-17-Catald...	line 15	C/C++ Problem

Linee guida

- Riconoscere alterazioni di blocchi di istruzioni che vengono frequentemente usati

```
for (int a=0;a<10;a++)
```

```
int b=0;
```

```
for (a=0;a<10;a++)
```

Linee guida

- Esaminare blocchi di istruzioni simili. Tipicamente, nelle istruzioni iterative, si tende a introdurre stessi bugs.

```
int a[10]  
for (int a=0;a<=10;a++)
```

```
Studente elenco[10]  
for (int elenco=0;elenco<=10;elenco++)
```

Linee guida

- Seguire al contrario la sequenza di istruzioni che ha generato il risultato
- Per prima cosa, va considerata la variabile che contiene il risultato, quindi, le istruzioni che hanno modificato quella variabile.

Linee guida

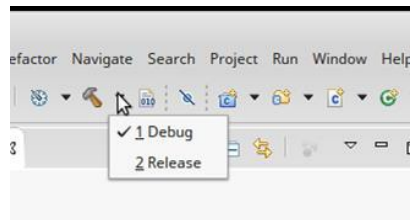
- Funzioni e procedure vanno implementate e testate di pari passo. Seguire al contrario la sequenza di istruzioni che ha generato il risultato
- Se c'è un bug ogni volta che viene invocata una specifica funzione, quella funzione nasconde una bug.

Debugger

- In aggiunta a queste line guida, si utilizza un tool chiamato debugger, spesso associato col compilatore oppure è una funzionalità disponibile nell'IDE.
- Un debugger principalmente consente di
 - seguire l'esecuzione del programma attraverso l'esecuzione di singole istruzioni (**tracing**)
 - seguire le invocazioni, da funzione chiamante all'interno della funzione chiamata (**stack trace**)
 - visualizzazione del contenuto delle **variabili**
 - valutazione delle **espressioni**

Debugger

- Esso lavora tra codice eseguibile (compilato) e codice sorgente, quindi ha bisogno di informazioni maggiori rispetto alla normale versione eseguibile. Per questo, il debug usa una versione non ottimizzata dell'eseguibile.
- In Eclipse CDT infatti esistono due modalità per generare un programma eseguibile



- In Eclipse CDT, il debugger può essere lanciato attraverso il tasto



Debugger

The screenshot shows the Eclipse IDE with the following components:

- Debug Console:** Shows the current thread as `Thread #1 0 (Suspended : Breakpoint)` at `main() at Debugging.c:15 0x40144e`.
- Variables Window:** Displays the variable `somma` of type `int` with a value of `4194432`.
- Source Editor:** Shows the source code of `Debugging.c` with the current instruction `int somma = 0;` highlighted.
- Outline View:** Shows the project structure with `main(void) : int` selected.

Annotations on the image:

- Programma in debugging:** Points to the Debug Console.
- Funzione in debugging:** Points to the current thread in the Debug Console.
- Stato delle variabili:** Points to the Variables window.
- Istruzione correntemente in esecuzione:** Points to the current instruction in the Source Editor.

```
11 #include <stdio.h>
12 #include <stdlib.h>
13
14 int main(void) {
15     int somma = 0;
16
17     for(int i=0; i<10; i++) {
18         somma = somma + i;
19     }
20
21     printf("Somma:%d"+somma);
22 }
23
```

Name	Type	Value
somma	int	4194432

Outline:

- stdio.h
- stdlib.h
- main(void) : int

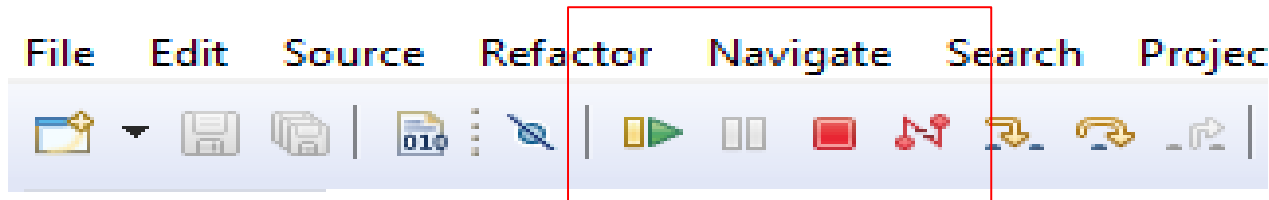
Console:



Debugging.exe [C/C++ Application] Debugging.exe

Writable | Smart Insert | 15 : 1

Debugger

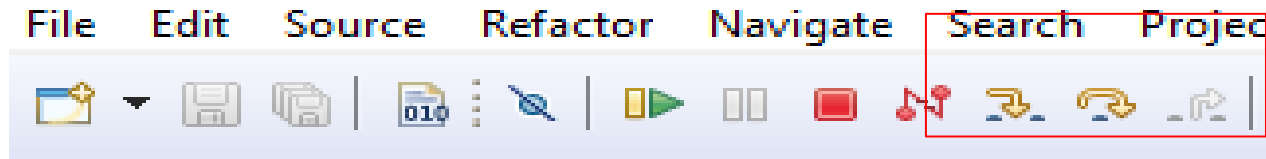
- Un debugger principalmente consente di
 - *seguire l'esecuzione del programma attraverso l'esecuzione di singole istruzioni (**tracing**)*






-  • **Resume:** esegue le istruzioni fino al prossimo breakpoint oppure fino al termine del programma
 - Breakpoint: punto di interruzione momentanea del debugging.
-  • **Terminate:** termina l'esecuzione del programma.
 - Usato per terminare il programma quando va in loop infinito.
 - A conclusion del debugging, un programma deve essere sempre terminato, perchè diversamente è in esecuzione per il sistema operativo.

Debugger

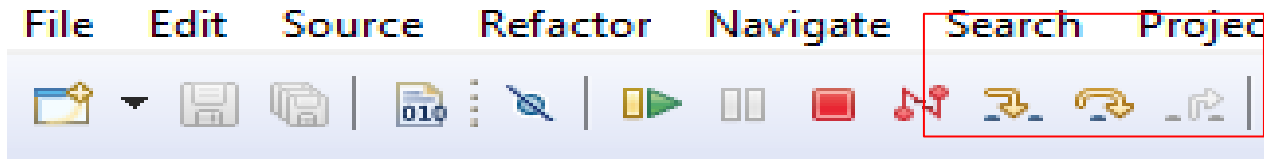
- Un debugger principalmente consente di
 - *seguire l'esecuzione del programma attraverso l'esecuzione di singole istruzioni (**tracing**).*
 - Comandi per il controllo di ogni singola istruzione:



-  **Step into:** esegue il programma invocando le funzioni ed entrando nella funzione invocata
-  **Step over:** esegue il programma invocando le funzione senza entrare nella funzione invocate
-  **Step out:** quando si è entrati in una funzione invocata (Step into), torna alla funzione chiamante
- Ogni istruzione è eseguita/considerata cliccando su uno dei comandi.

Debugger

- Un debugger principalmente consente di
 - *seguire le invocazioni, da funzione chiamante all'interno della funzione chiamata (**stack trace**)*
 - Comandi per il controllo di ogni singola istruzione:



- **Step into:** esegue il programma invocando le funzioni ed entrando nella funzione invocata

```

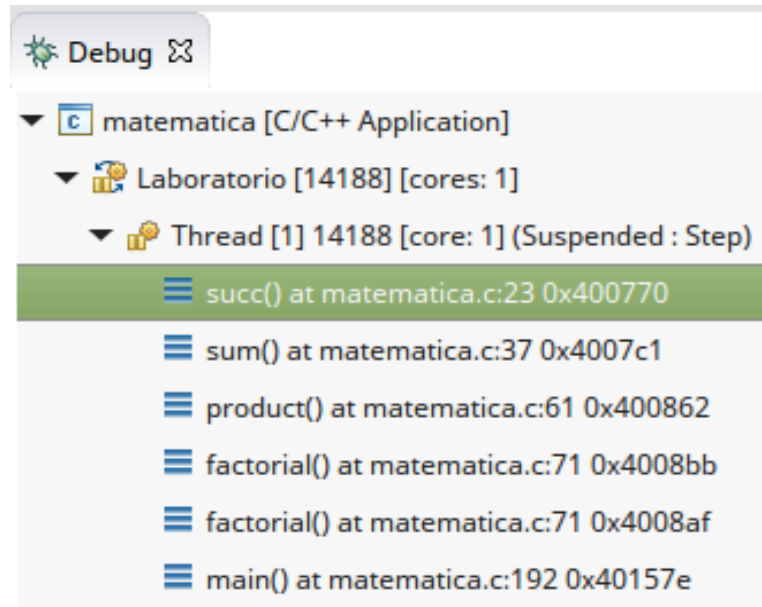
59     int i;
60     for (i=0; i<b; i++){
61         result = sum(result, a);
62     }
  
```

```

30 int sum(int a, int b) {
31     if (b == 0){
32         return a;
33     } else if (b > 0) {
  
```

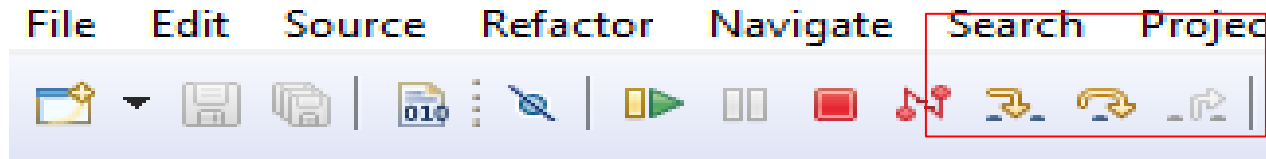
Debugger

- Un debugger principalmente consente di
 - *seguire le invocazioni, da funzione chiamante all'interno della funzione chiamata (**stack trace**)*
- Quando vengono invocate le funzioni, si aggiorna lo stack delle chiamate



Debugger

- Un debugger principalmente consente di
 - *seguire l'esecuzione del programma attraverso l'esecuzione di singole istruzioni (**tracing**).*
 - Comandi per il controllo di ogni singola istruzione:



- **Step over:** esegue il programma invocando le funzione senza entrare nella funzione invocate

```

59     int i;
60     for (i=0; i<b; i++){
61         result = sum(result, a);
62     }

```

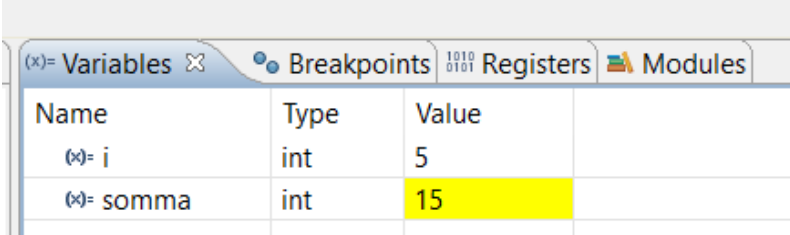
```

59     int i;
60     for (i=0; i<b; i++){
61         result = sum(result, a);
62     }

```

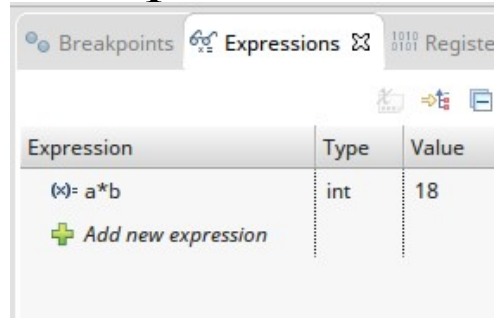

Debugger

- Un debugger principalmente consente di
 - *visualizzazione del contenuto delle **variabili***
 - *valutazione delle **espressioni***
- Parallelamente al tracing, il contenuto delle variabili vengono aggiornata man mano che le istruzioni che le manipolano vengono eseguite.



Name	Type	Value
(x)= i	int	5
(x)= somma	int	15

- Possiamo anche monitorare il valore di espressioni, che il programmatore deve formulare nella scheda Expression

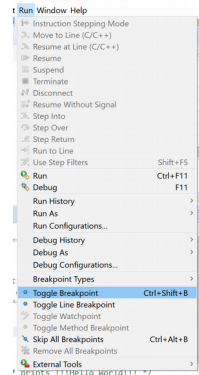


Expression	Type	Value
(x)= a*b	int	18

+ Add new expression

Debugger

- Un debugger principalmente consente di
 - ***Resume:** esegue le istruzioni fino al prossimo breakpoint oppure fino al termine del programma*
 - ***Breakpoint:** punto di interruzione momentanea del debugging.*
- Inserito dal programmatore in corrispondenza della istruzione selezionata, tramite menu Run o doppio click sulla barra di scorrimento.



```
12 #include <stdlib.h>
13
14 int main(void) {
15     puts("!!!Hello World!!!"); /* prints !
16     return EXIT_SUCCESS;
17 }
```

- Quando si lancia il debugger, il debugging si sospende al primo break point, quindi si eseguire procedendo con gli step oppure continuare direttamente fino al prossimo breakpoint.

Esercizio

Eseguire in debugging con gli step il seguente codice.

Cosa fa questo programma?

```
#include <stdio.h>
int somma(int a, int b);

int main() {
    int totale = 0;
    int doppioTotale= 0;
    int i=0;
    for(i=0; i<10; i++) {
        totale = somma(totale, i);
        doppioTotale= somma(doppioTotale, i) * 2;
    }

    printf("Totale:%d\n", totale);
    printf("Doppio Totale:%d\n", doppioTotale);
}

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