

Introduzione al modulo e cenni sul linguaggio C

Matteo Ferrara

Dipartimento di Informatica - Scienza e Ingegneria

matteo.ferrara@unibo.it

Informazioni di base

Generalità

Docente:	Matteo Ferrara
Email:	matteo.ferrara@unibo.it
Ricevimento:	giovedì 10.00 - 11.00 e venerdì 15.00 – 16.00
Orario delle lezioni:	giovedì 14.00 - 17.00
Aula:	Laboratori 3.1 e 3.3
Tutor:	Gabriele Graffieti (gabriele.graffieti@unibo.it) giovedì 10.00 - 11.00 Chiara Varini (chiara.varini2@unibo.it) venerdì 15.00 – 16.00

Libri di testo consigliati:

- *Introduzione agli Algoritmi e Strutture Dati*, T. Cormen, C. Leiserson, R. Rivest, C. Stein, 3a edizione, McGraw-Hill.
- *Lezioni di algoritmi e strutture dati*, V. Maniezzo, L. Margara, 2003.
- Lucidi delle lezioni

Prerequisiti

- Competenze di programmazione **ANSI C** (non è necessario avere sostenuto l'esame ma è necessario avere una minima fluidità nella codifica);
- Conoscenza della teoria relativa agli algoritmi che si implementeranno in laboratorio.

Modalità di esame (1)

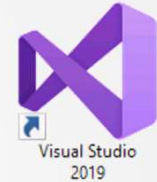
- L'esame di ASD prevede anche il superamento della **prova di laboratorio**;
- La prova di laboratorio consiste nello **sviluppo di un progetto**, da realizzare in linguaggio C;
- Il progetto non ha una valutazione numerica: se valutato positivamente, permette di accedere alla prova scritta;
- Dopo aver consegnato il progetto, non sono previste modifiche o nuove consegne fino all'avvenuta valutazione;
- Il progetto sarà corretto e valutato automaticamente da uno specifico applicativo.

Modalità di esame (2)

- Il progetto deve essere sviluppato **individualmente**;
- I progetti saranno controllati da un **sistema antiplagio** e confrontati con tutti quelli finora consegnati. Indici di correlazione troppo alti porteranno a **rifiutare il progetto**;
- Il progetto deve essere consegnato obbligatoriamente **entro una settimana prima della prova scritta**;
- Una volta consegnato e corretto, **il progetto resta valido per tutte le prove scritte seguenti** (anche per i successivi anni accademici);
- Maggiori dettagli sulla struttura del progetto saranno esposti durante l'ultima lezione di laboratorio.

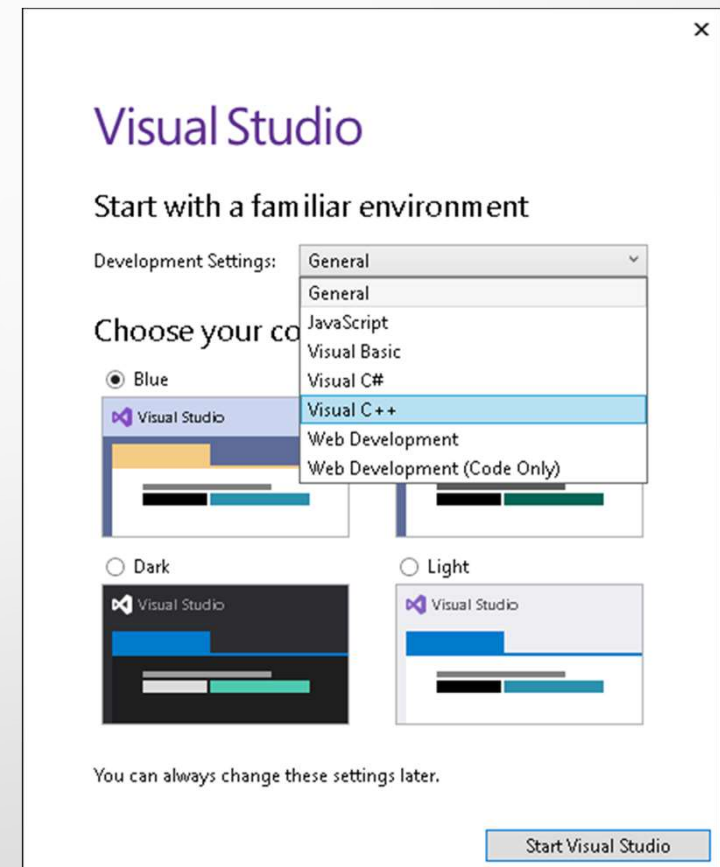
Visual Studio

Avviare l'ambiente di sviluppo **Visual Studio 2019** utilizzando il collegamento presente sul desktop.



La prima volta che si avvia Visual Studio, si deve impostare la personalizzazione dell'ambiente di sviluppo:

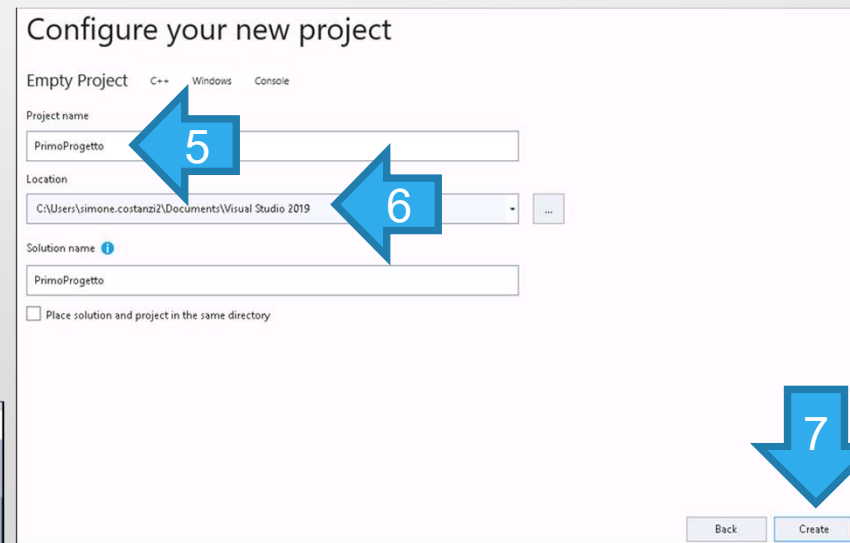
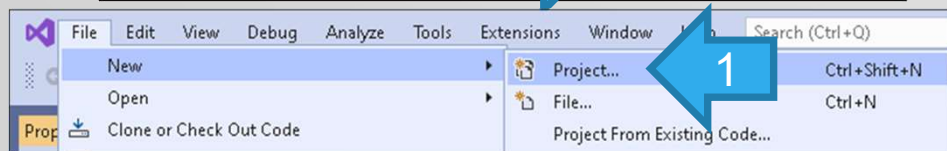
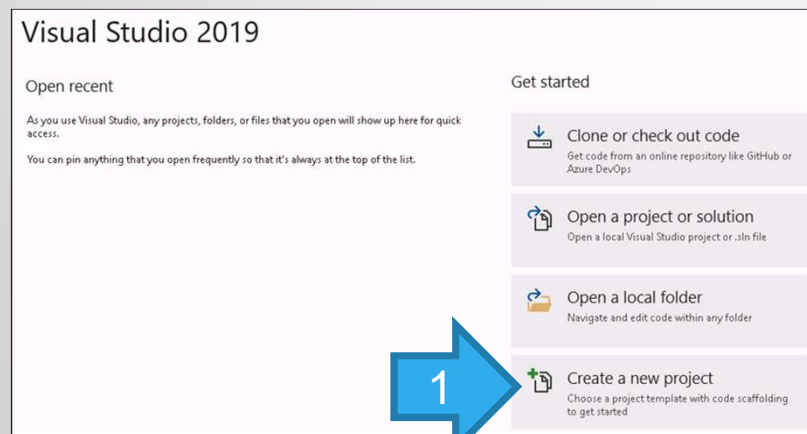
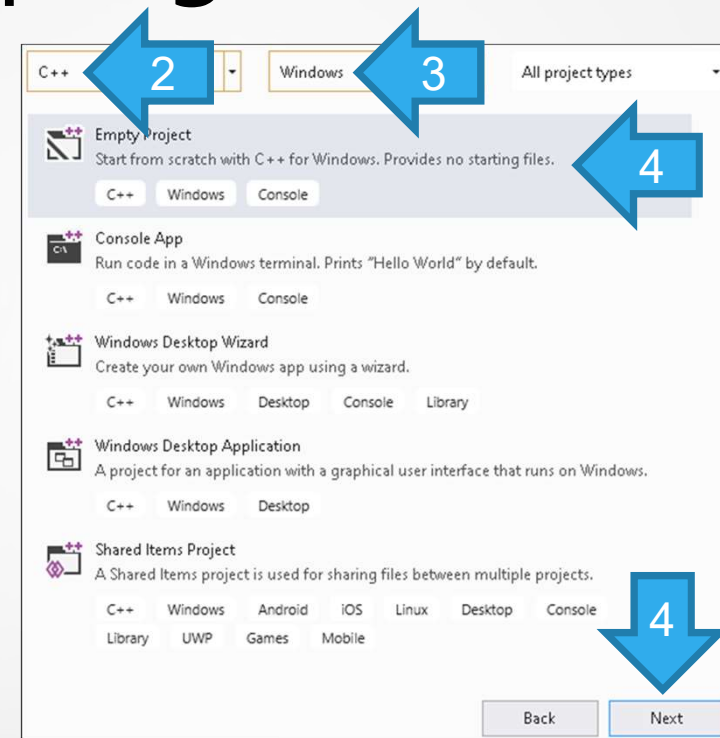
- Tecnologia utilizzata (Visual C++)
- Colore dell'ambiente di sviluppo



ATTENZIONE: alla prima esecuzione Visual Studio potrebbe richiedere un po' di tempo prima di avviarsi

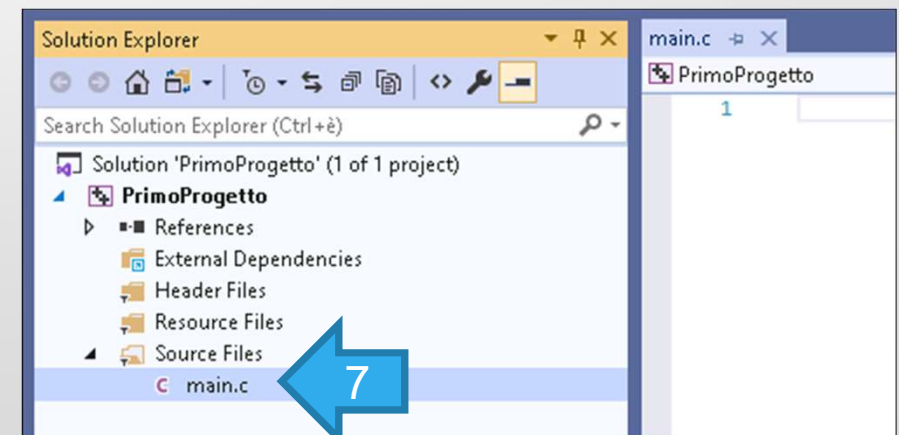
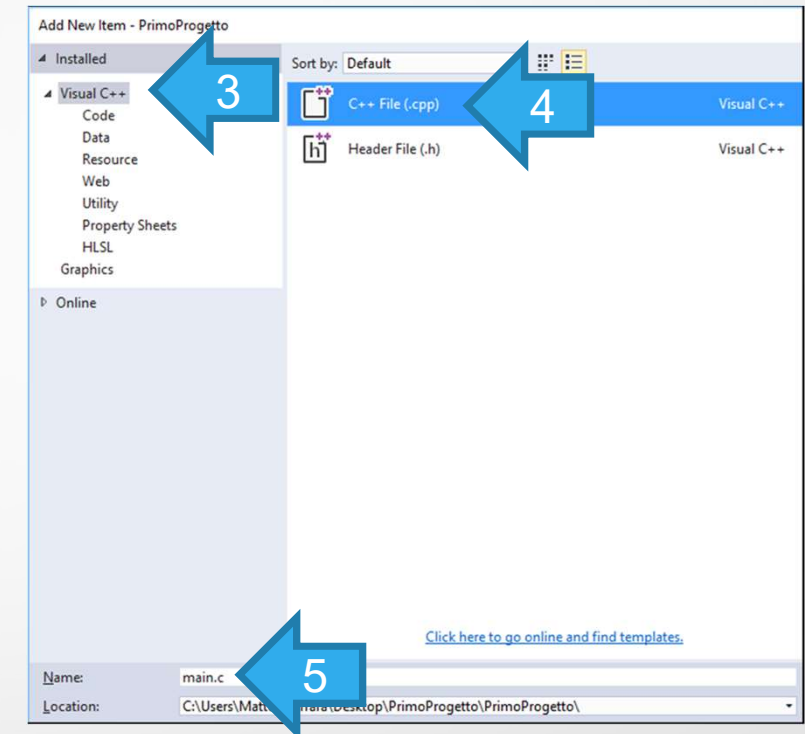
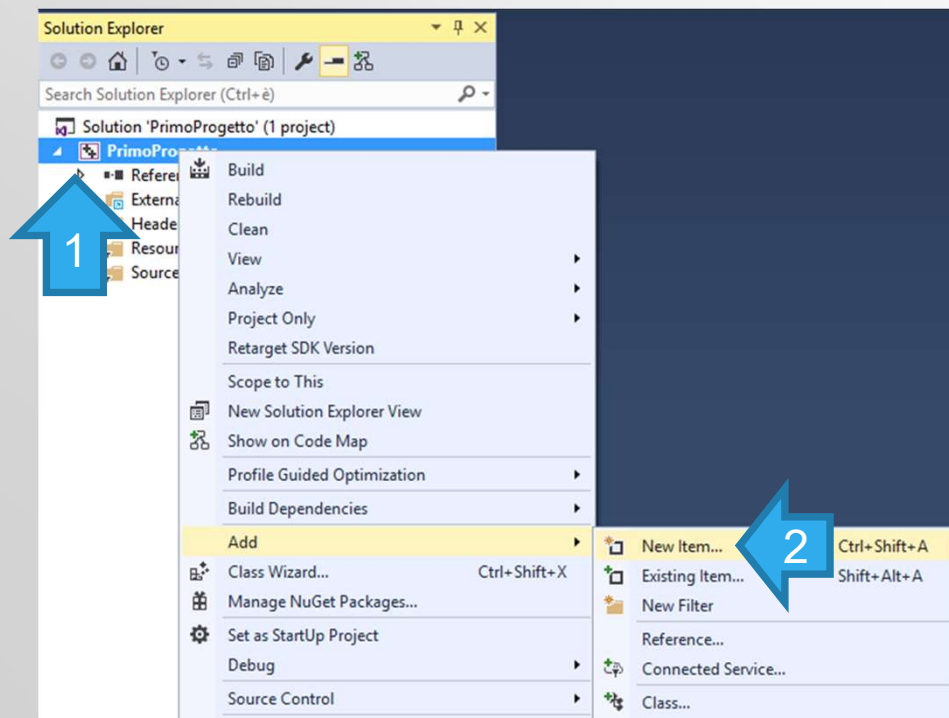
Creazione di un nuovo progetto

1. Selezionare “Create a new project” nella finestra iniziale dopo l’apertura dell’ambiente o “New→Project” dal menù File
2. Selezionare la voce “C++” dal menù di sinistra
3. Selezionare “Windows” dal menù centrale
4. Selezionare il progetto “Empty Project” e premere “Next”
5. Digitare il nome del progetto
6. Verificare che la *location* del progetto sia sotto la cartella “Documents” del vostro utente
7. Premere “Create”



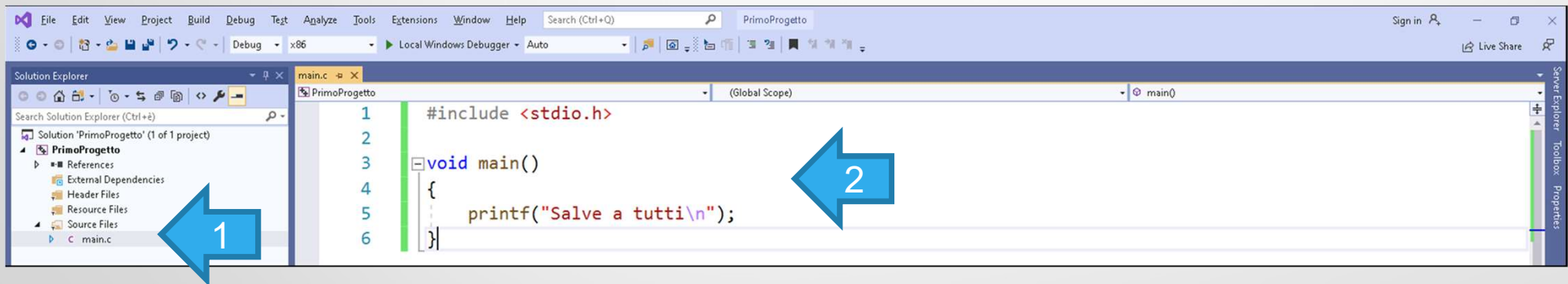
Inserire un file nel progetto

1. Fare click con il tasto destro sul progetto
2. Selezionare "Add → New Item" dal menù contestuale
3. Selezionare la voce "Visual C++" dal menù di sinistra
4. Selezionare "C++ File (.cpp)" dal menù centrale
5. Specificare il nome e l'estensione del file (Es. main.c)
6. Premere "Add"
7. Il file appena creato comparirà nella cartella "Source Files" del progetto



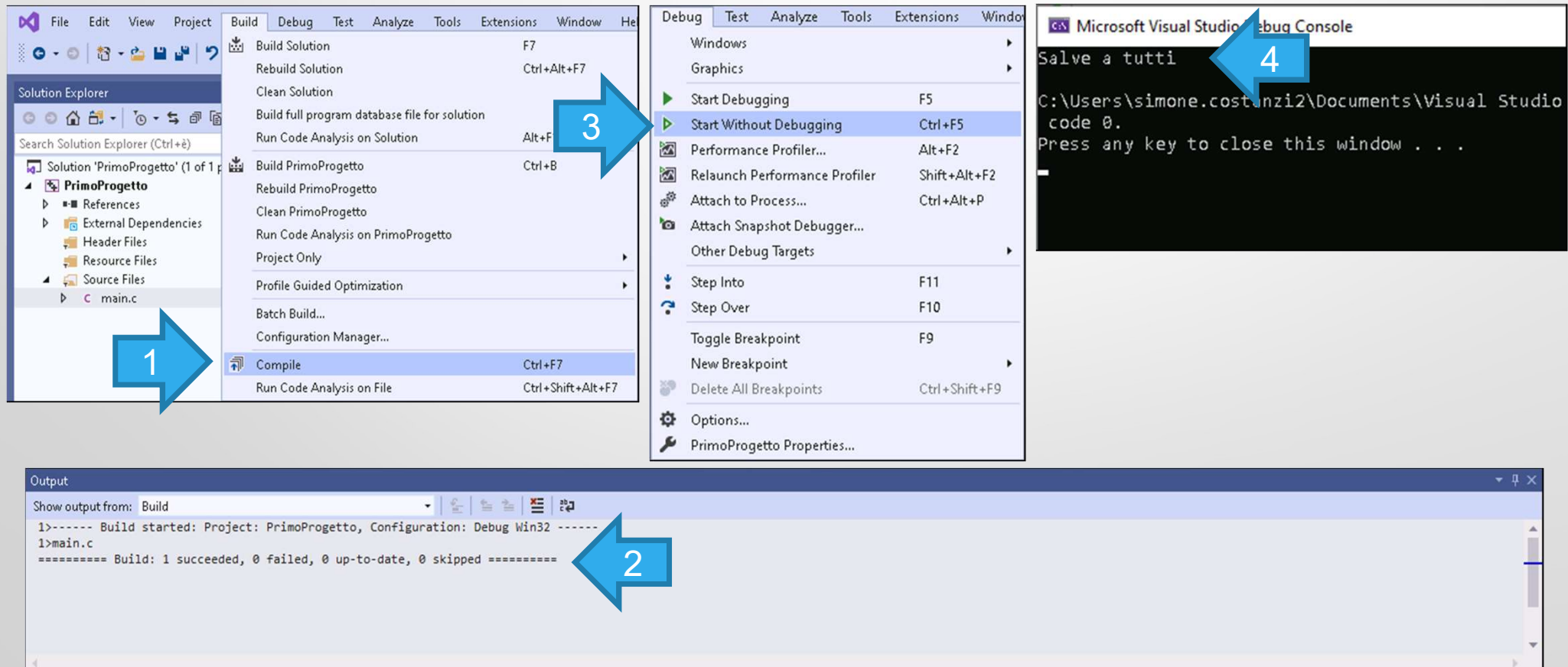
Un primo programma in C

1. Doppio click sul file .c nell'albero del progetto
2. Scrivere il programma riportato in figura



Compilare ed eseguire il programma

1. Selezionare dal menù Build la voce "Compile" (Ctrl+F7) per compilare e linkare il programma
2. Controllare il risultato nella finestra di output (scheda "Build")
3. Selezionare "Start Without Debugging" (Ctrl+F5) dal menù Debug per eseguire il programma
4. Notare la stampa nella console di Visual Studio come risultato dell'esecuzione



Debug del programma (1)

1. Scrivere il programma come mostrato in figura
2. Cliccare sulla barra grigia a sinistra dell'istruzione in riga 7
3. Selezionare "Start Debugging" (F5) dal menù Debug per eseguire il programma: l'esecuzione si interromperà in corrispondenza del breakpoint
4. Selezionare "Step Over" (F10) dallo stesso menù per avanzare di una riga alla volta
5. Selezionare "Continue" (F5) per lasciare proseguire il programma fino alla fine

The following images illustrate the steps to debug the program:

- Step 1:** The code editor shows the program:

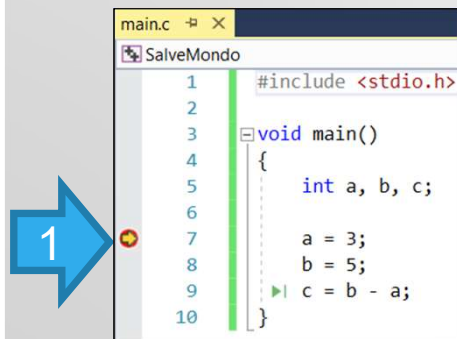

```

1  #include <stdio.h>
2
3  void main()
4  {
5      int a, b, c;
6
7      a = 3;
8      b = 5;
9      c = b - a;
10 }
      
```
- Step 2:** A breakpoint (red dot) is set on the line `a = 3;` in the `main` function.
- Step 3:** The **Debug** menu is open, and **Start Debugging** (F5) is selected. The program execution starts and stops at the breakpoint.
- Step 4:** The **Debug** menu is open, and **Step Over** (F10) is selected. The program advances to the next line.
- Step 5:** The **Debug** menu is open, and **Continue** (F5) is selected. The program resumes execution until it reaches the end.

Debug del programma (2)

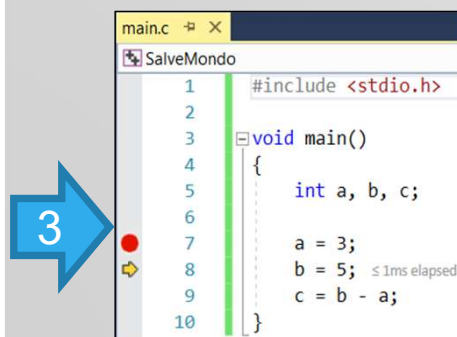
Durante la fase di debugging potrà essere molto utile visualizzare il valore contenuto (in un determinato momento) da una o più variabili:

1. Compilare ed eseguire il programma in "Debug Mode" (F5)
2. Aprire la finestra "Locals" selezionando "Windows→Locals" dal menù Debug
3. Avanzare di una istruzione alla volta con "Step Over (F10)"
4. Osservare l'effetto delle istruzioni sulle variabili
5. Scegliere "Continue" (F5) per lasciar eseguire il programma fino alla fine



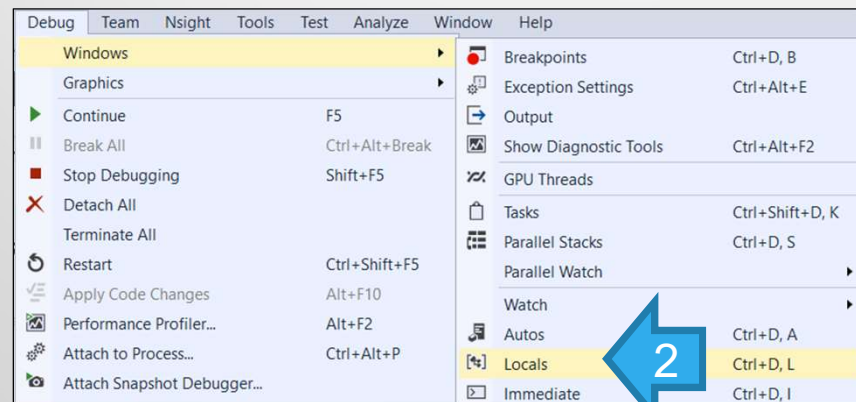
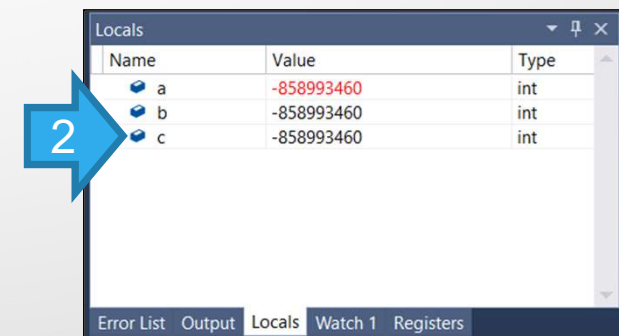
```

main.c
SalveMondo
1  #include <stdio.h>
2
3  void main()
4  {
5      int a, b, c;
6
7      a = 3;
8      b = 5;
9      c = b - a;
10 }
  
```



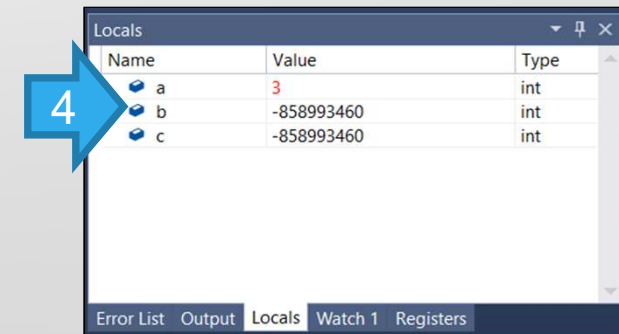
```

main.c
SalveMondo
1  #include <stdio.h>
2
3  void main()
4  {
5      int a, b, c;
6
7      a = 3;
8      b = 5;
9      c = b - a;
10 }
  
```

Name	Value	Type
a	-858993460	int
b	-858993460	int
c	-858993460	int

Error List Output Locals Watch 1 Registers



Name	Value	Type
a	3	int
b	-858993460	int
c	-858993460	int

Error List Output Locals Watch 1 Registers

Alcune funzioni C (1)

- **atoi**: converte una *stringa* in un *intero*. Per utilizzarlo è necessario includere `<stdlib.h>`.

```
int atoi(const char *str)
```

- **printf**: scrive un testo formattato su *stdout* (stream predefinito di C associato al dispositivo di visualizzazione). All'interno della stringa da mandare in output è possibile introdurre dei marcatori che saranno istanziati dinamicamente in base al valore assunto dalle rispettive variabili indicate dopo la virgola. Ad esempio, `%d` corrisponde ad un intero, `%f` ad un float, `%s` ad una stringa. Per utilizzarlo è necessario includere `<stdio.h>`.

```
int printf(const char *format, ...)
```

- **getchar**: legge un *unsigned char* da *stdin* (stream predefinito di C associato alla tastiera). Restituisce il codice ASCII corrispondente. La chiamata è bloccante, ovvero il flusso di esecuzione procede solo quando viene fornito in input un *char* su *stdin*. Per utilizzarlo è necessario includere `<stdio.h>`.

```
int getchar(void)
```

Alcune funzioni C (2)

- **clock**: rileva il tempo di CPU al momento della chiamata. Può essere molto utile per calcolare il lasso di tempo trascorso durante l'esecuzione di un blocco di codice. Per utilizzarlo è necessario includere `<time.h>`.

```
clock_t start, end;  
start = clock();  
    // Blocco di codice da monitorare  
end = clock();  
printf("Time: %f sec \n", (double)(end - start)/CLOCKS_PER_SEC);
```


Parametri da linea di comando (1)

Possiamo strutturare il nostro applicativo in modo che, una volta eseguito da linea di comando, recepisca una serie di parametri.

Test parametri in linea di comando

```
void main(int argc, char* argv[])
{
    int a, b, sum=0;
    clock_t start, end;

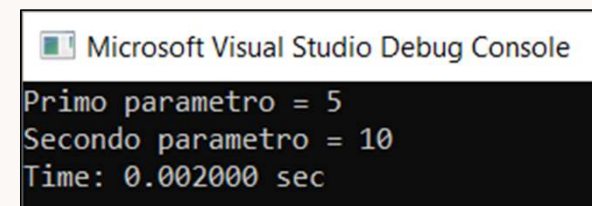
    if (argc != 3)
    {
        printf("Numero di parametri errato");
        return;
    }

    a = atoi(argv[1]);
    b = atoi(argv[2]);
    printf("Primo parametro = %d\nSecondo parametro = %d\n", a, b);

    start = clock();
    for (int i = 0; i < 1000000; i++)
        sum += a * b;

    end = clock();

    printf("Time: %f sec \n", (double)(end - start) / CLOCKS_PER_SEC);
}
```



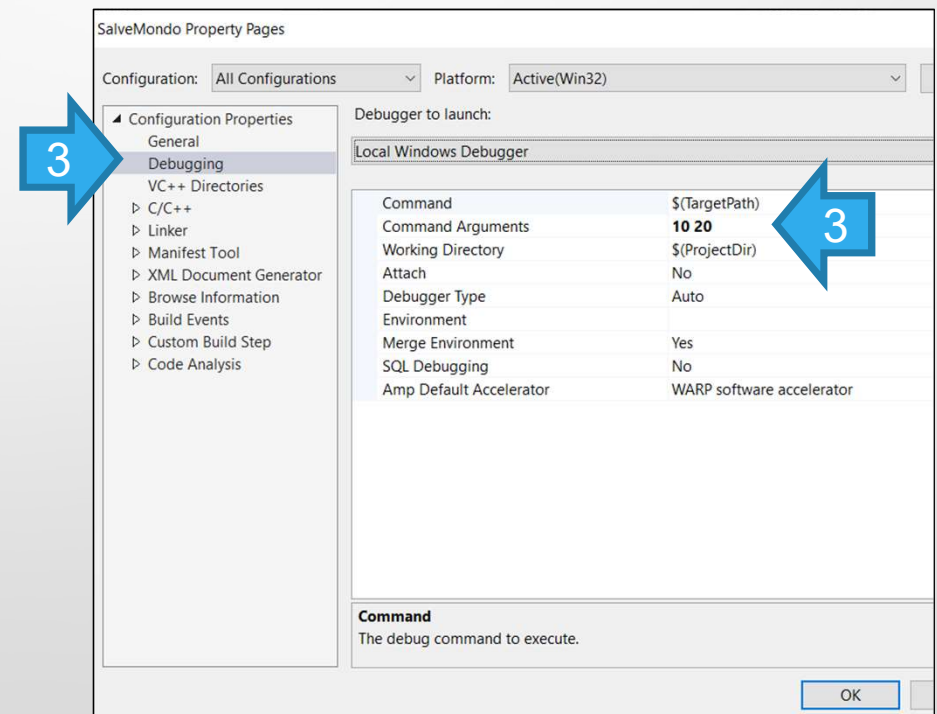
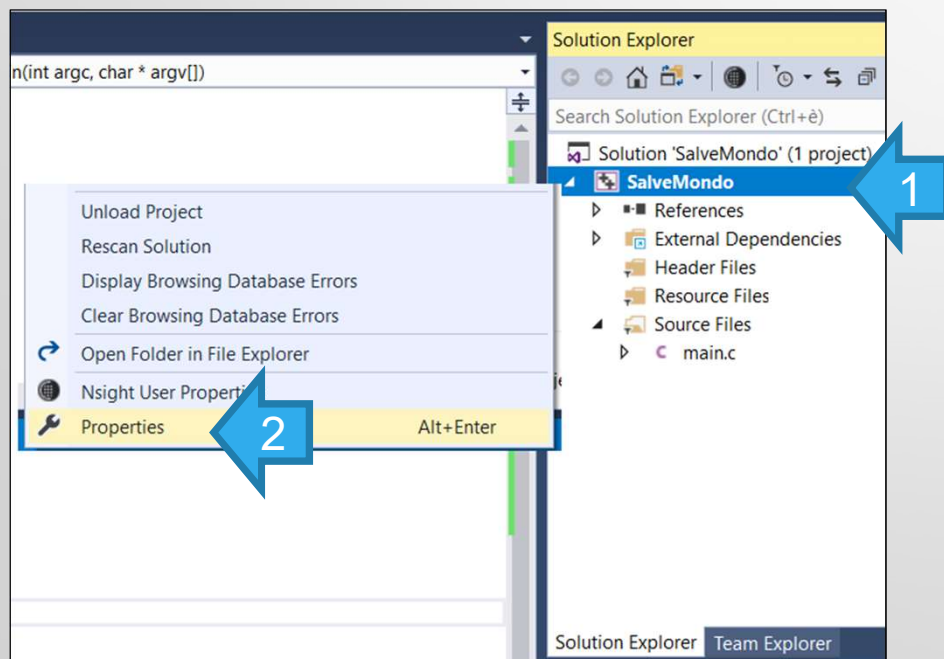
Microsoft Visual Studio Debug Console

```
Primo parametro = 5
Secondo parametro = 10
Time: 0.002000 sec
```

Parametri da linea di comando (2)

Per passare i parametri in modalità Debug:

1. Cliccare con il tasto destro sul nome del progetto
2. Selezionare la voce di menù "Properties"
3. Una volta selezionato "Debugging" dal menù di sinistra, inserire i parametri sotto "Command Arguments" separati da spazio
4. Compilare ed eseguire il programma in "Debug Mode" (F5)



Pass by reference

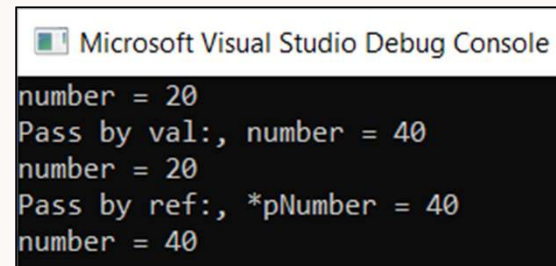
Esempio di passaggio per riferimento

```
#include <stdio.h>

void passByVal(int number)
{
    number *= 2;
    printf("Pass by val:, number = %d\n", number);
}

void passByRef(int* pNumber)
{
    *pNumber *= 2;
    printf("Pass by ref:, *pNumber = %d\n", *pNumber);
}

void main()
{
    int number = 20;
    printf("number = %d\n", number);
    passByVal(number);
    printf("number = %d\n", number);
    passByRef(&number);
    printf("number = %d\n", number);
}
```



Microsoft Visual Studio Debug Console

```
number = 20
Pass by val:, number = 40
number = 20
Pass by ref:, *pNumber = 40
number = 40
```

Leggere un file di testo (1)

Lettura di un file riga per riga

Lettura da file chiave-valore

```
#include <stdio.h>
#include <string.h>
#define LINE_LENGTH 120
void main(int argc, char* argv[])
{
    char *buf, *separator = ",", line[LINE_LENGTH]; int n, *arr;
    printf("Lettura del file %s\n", argv[1]);

    //apertura del file il cui nome è passato in linea di comando
    FILE* fin = fopen(argv[1], "r");
    if (fin == NULL)
        return;

    //conteggio delle righe presenti nel file
    n = 0;
    while (fgets(line, LINE_LENGTH, fin) != NULL)
        n++;

    //allocazione della memoria (con valori azzerati)
    arr = (int*) calloc(n, sizeof(int));
    ...
}
```

Leggere un file di testo (2)

Lettura di un file riga per riga

Lettura da file chiave-valore

```
...
//riposiziona il cursore a inizio file e legge i valori
rewind(fin);
n = 0;
while (fgets(line, LINE_LENGTH, fin) != NULL)
{
    //legge la chiave (parte che precede il separatore)
    buf = strtok(line, separator);
    //legge il valore (parte che segue il separatore) e lo salva
    arr[n]=atoi( strtok(NULL, separator) );
    printf("%s->%d\n", buf, arr[n]);  n++;
}
fclose(fin);
free(arr);
}
```

Microsoft Visual Studio Debug Console

```
Lettura del file I:\CorsiUniversitari\AlgoritmiStruttureDati\19_20\es1\data.txt
chiave1->5
chiave2->10
chiave3->1
chiave4->556
chiave5->12345
chiave6->773
```

Funzioni obsolete (1)

Alcune funzioni standard quali *fopen*, *strtok*, ecc. sono ormai considerate obsolete da Visual Studio che ne consiglia una versione più aggiornata e sicura.

Error List	
Entire Solution	
3 Errors	
0 of 3 Warnings	
0 Messages	
Build + IntelliSense	
Code	Description
C4996	'fopen': This function or variable may be unsafe. Consider using fopen_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.
C4996	'strtok': This function or variable may be unsafe. Consider using strtok_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.
C4996	'strtok': This function or variable may be unsafe. Consider using strtok_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.

Funzioni obsolete (2)

Per disabilitare questi errori:

1. Cliccare con il tasto destro sul nome del progetto e selezionare la voce di menù "Properties"
2. Selezionare "Configuration Properties → C/C++ → Preprocessor" dal menù di sinistra
3. Aprire l'editor di "Preprocessor Definitions"
4. Aggiungere **_CRT_SECURE_NO_WARNINGS** nella casella di testo

