IPC144
Introduction to C Programming

Week-4

Arrays
Testing & Debugging

Managing a Lot of Data

Up to now

If you were asked to store 30 student grades of float type you would declare 30 float variables with different names:

```
float grade1,  grade2,  grade3,  grade4,  grade5,  grade6,
      grade7,  grade8,  grade9,  grade10, grade11, grade12,
      grade13, grade14, grade15, grade16, grade17, grade18,
      grade19, grade20, grade21, grade22, grade23, grade24,
      grade25, grade26, grade27, grade28, grade29, grade30
```

What if you had to manage 2,000 grades?
There must be a better way!!!

Welcome to Arrays…

Arrays

- Provides us with the ability to:
  - Store multiple values (like a list)
  - Using a single variable name
  - But must be of the same data type
- Each individual item in the array is referred to as an **ELEMENT**
- The element's **INDEX** is the position within the array
- Array indexing is **zero-based** meaning:
  - The 1st element is always at index **0**

Array Construct

Array Declaration

```
type identifier [ size ];
float  grades  [30];
```

This array declaration will attempt to reserve enough contiguous memory to hold 30 float type numbers for a variable called “grades”

float = 4 bytes * 30 = **120 bytes** (continuous block of memory)

```
grades[0]  is the 1st  ELEMENT in the array at INDEX 0

grades[29] is the 30th ELEMENT in the array at INDEX 29
```

Arrays, Types and Organization

What good is a list of grades without knowing who it belongs to?

Let's expand on the previous example of grades…

New Requirements	**Data Type**
▪ Student number	**int** (integer)
▪ Semester number	**int** (integer)
▪ Grade	**float** (floating-point)

How can we do this?

How can we organize this?

Parallel Arrays

Parallel Arrays!

New Requirements	Data Type	Array Declaration
▪ Student number	**int**	**int** studentID[30];
▪ Semester number	**int**	**int** semester[30];
▪ Grade	**float**	**float** grades[30];

Organized and easy to access!

Let's show the 5th student's related information (**INDEX 4**):

```
printf("Student number:%d\n",studentID[4]);
printf("Semester:      %d\n", semester[4]);
printf("Grade:         %.1f\n", grades[4]);
```

Visual Studio Debugging

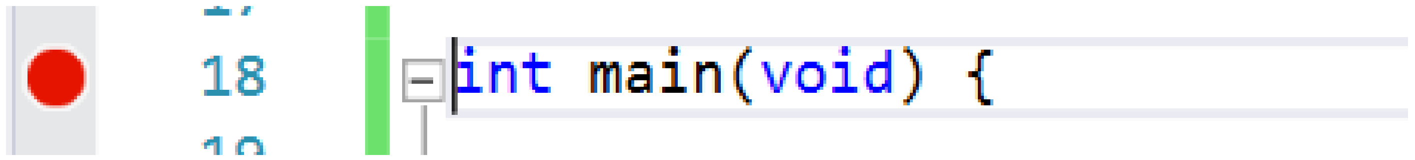
Debugging

Provides us with the ability to:

- "step" through the code
- View variable values as the program executes
- Experience the program's data flow

Breakpoint

- Show's a red bullet marker alongside your code that indicates where the execution should pause when it is executed:



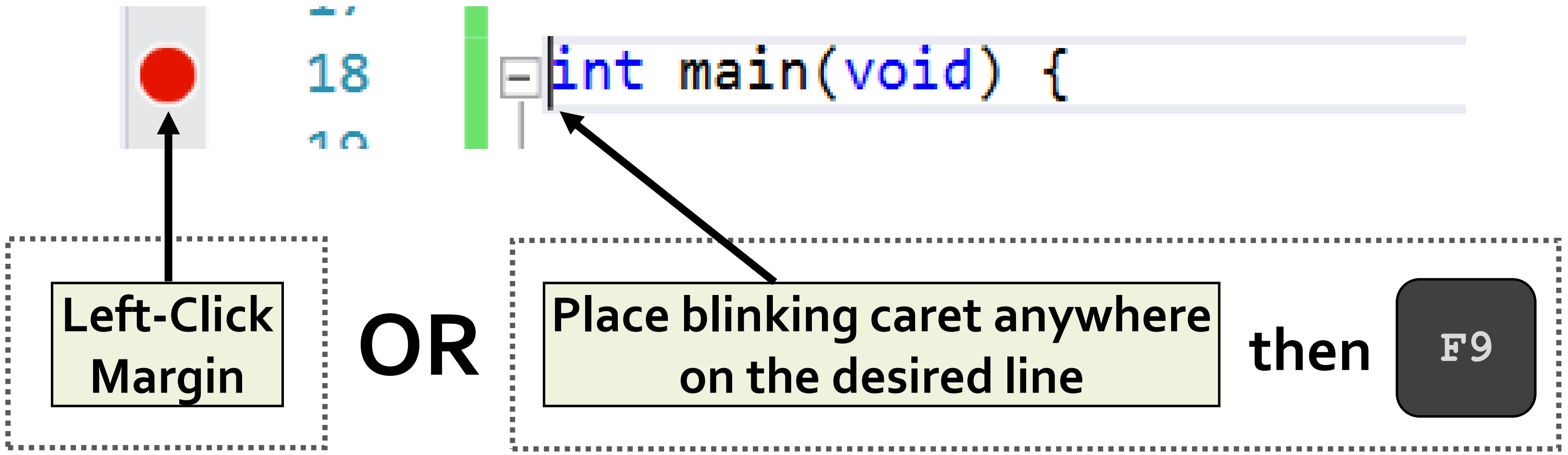
```
int main(void) {
```

Visual Studio Debugging

Breakpoint

Can be set one of two ways:

1. Left clicking on the left-most margin on the line of code you want to place a pause in execution
2. Place your the caret (blinking text input symbol) on the line of code you want to place a pause in execution and press **F9**



```
int main(void) {
```

Visual Studio Debugging

Running a Debug Session

In normal program execution, you do the following:

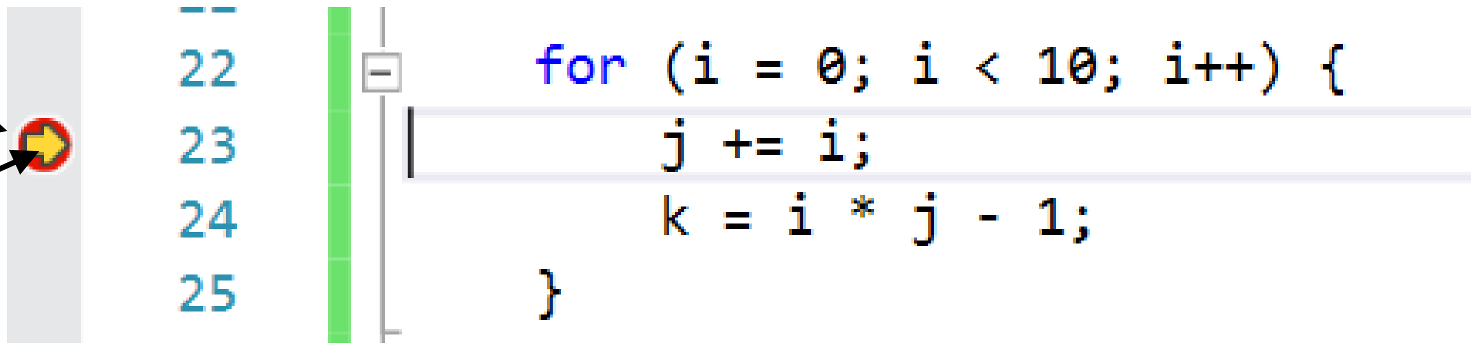
`Ctrl` + `F5` Execution **without** debugging

To run **WITH** debugging so execution will pause on the breakpoint(s) you have set, simply run your program by pressing only the **`F5`** key

~~`Ctrl`~~ + `F5` Execution **with** debugging

Visual Studio Debugging

Program execution will pause on the line the breakpoint was specified



```
for (i = 0; i < 10; i++) {
    j += i;
    k = i * j - 1;
}
```

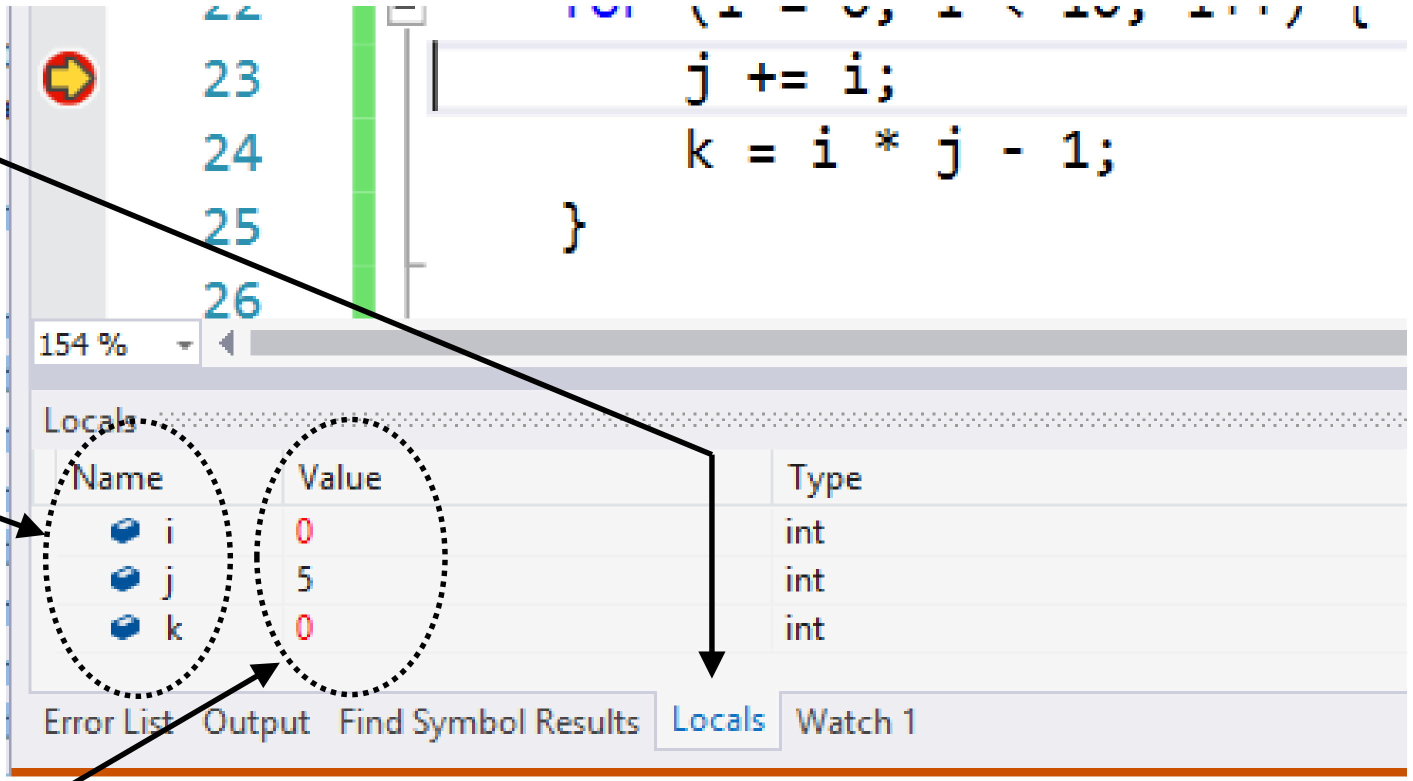
A yellow right-facing arrow indicates the active executing line

Visual Studio Debugging

At the bottom of your debug window you will see tabs (select the "Locals" tab)

All variables in the current scope with be visible in the details panel

Variable values will reflect their <u>current value at that point in time</u> of execution



```
            j += i;
            k = i * j - 1;
        }
```

Locals

Name	Value	Type
i	0	int
j	5	int
k	0	int

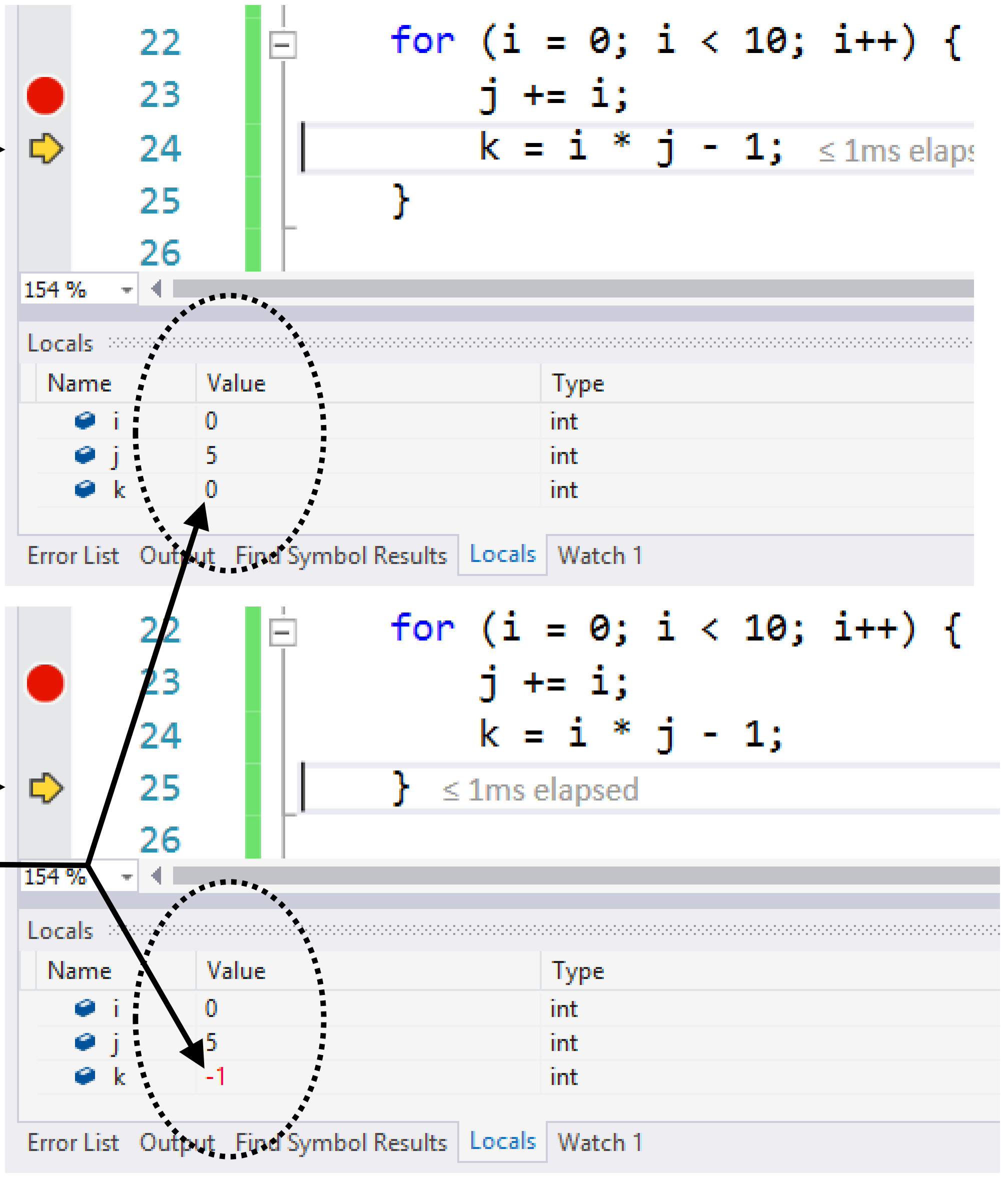
Visual Studio Debugging

Advance to the next line by pressing the **F10** key

F10

The yellow arrow will advance to indicate the executing line

Watch the values change as you progress!



```
for (i = 0; i < 10; i++) {
    j += i;
    k = i * j - 1;   ≤ 1ms elaps
}
```

154 %

Locals

Name	Value	Type
i	0	int
j	5	int
k	0	int

```
for (i = 0; i < 10; i++) {
    j += i;
    k = i * j - 1;
}  ≤ 1ms elapsed
```

154 %

Locals

Name	Value	Type
i	0	int
j	5	int
k	-1	int

Visual Studio Debugging

<u>Continuing Execution</u>

When you are done debugging the section of code you are interested in you can continue regular execution by pressing the **`F5`** key

F5

<u>Note</u>:

- Execution will stop however if another breakpoint is encountered
- If a breakpoint is on or inside an iteration construct (while, do/while, for) execution will stop again on the next iteration
- ***If there are no more breakpoints, your program will complete and the <u>console window will close</u>***

Visual Studio Debugging

Continuing Execution

Tip

- It is suggested when you are running in a debug session you place a breakpoint on the closing curly brace of main().
- Doing this will pause execution just prior to ending and provides you with an opportunity to review the output window before the window closes
- When finished reviewing the output window, simply press **F5** a final time and the debugging session will end and close the command window.

F5

Visual Studio Debugging

Breakpoint States

Breakpoints have two states:

- On/Enabled ●
- Off/Disabled ○

It is handy to disable a breakpoint when you are inside an iterator and want to execute past the loop without stopping with each iteration (but keeps the breakpoint's location)

Alternatively, you could simply remove the breakpoint entirely, but if you intend on using it again you will have to create it again.

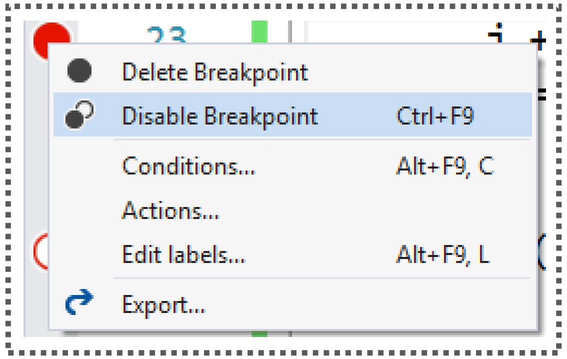
F5

Visual Studio Debugging

Breakpoint Management

- Toggle create/delete Breakpoint **Left-Click Margin/Symbol** **OR** **Place blinking caret anywhere on the desired line** **then** F9

- On/Enabled ●
- Off/Disabled ○



```
Delete Breakpoint
Disable Breakpoint     Ctrl+F9
Conditions...          Alt+F9, C
Actions...
Edit labels...         Alt+F9, L
Export...
```

OR Ctrl + F9