C++ Program Design
-- How to design your first program

Junjie Cao @ DLUT

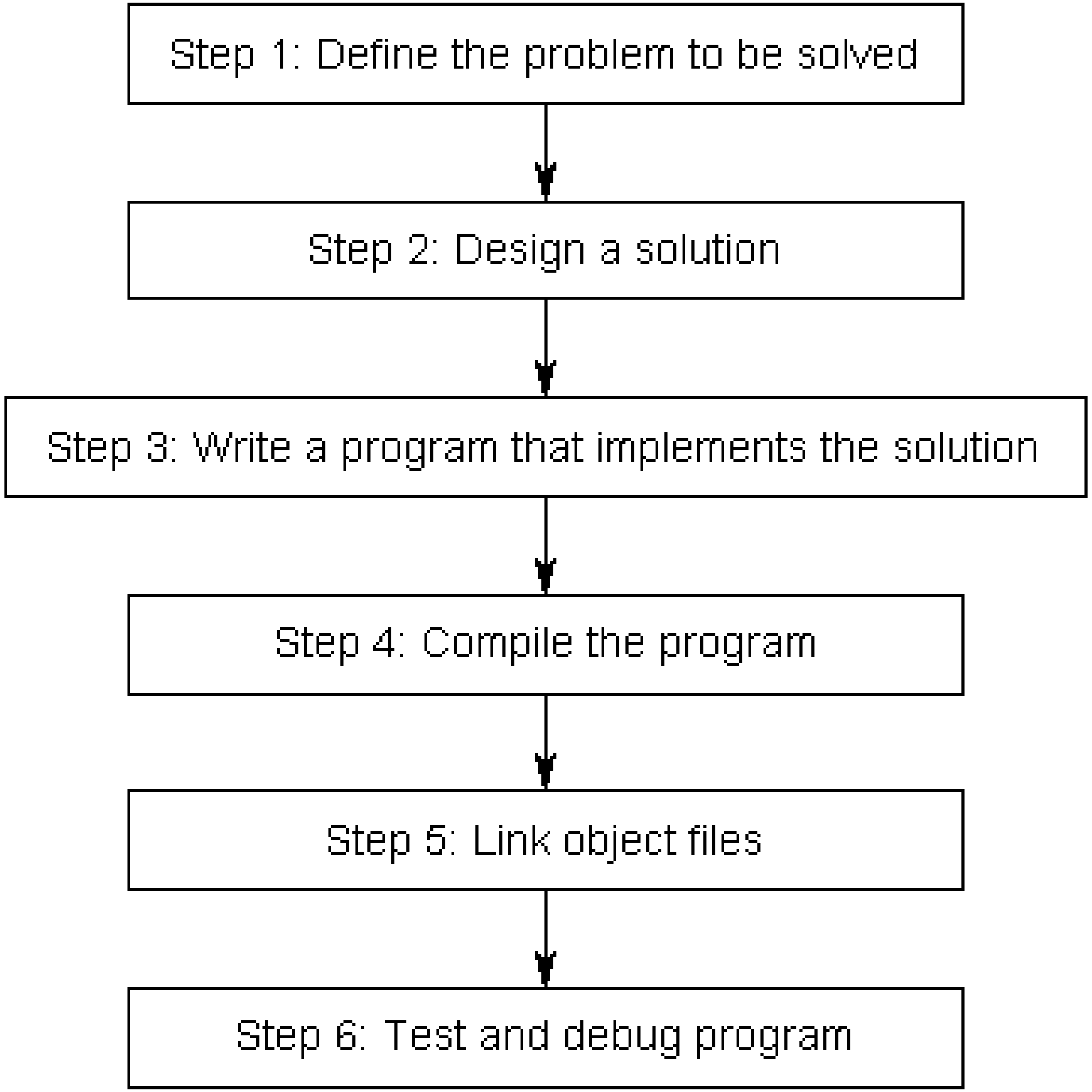
Summer 2016

http://jjcao.github.io/cPlusPlus

design your program

- design *before coding*
- programming is like architecture
  - Spend lots of time fixing problems that could been avoided with a little thinking ahead

Introduction to development



Step 1: Define the problem to be solved

Step 2: Design a solution

Step 3: Write a program that implements the solution

Step 4: Compile the program

Step 5: Link object files

Step 6: Test and debug program

Step 1: Define the problem

- **what** problem your program is attempting to solve
- state this in a sentence or two
  - I want to write a phone book application to help me keep track of my friend’s phone numbers.
  - I want to write a program that will read information about stocks from the internet and predict which ones I should buy.

- The worst thing you can do is write a program that doesn’t actually do what you (or your boss) wanted!

Step 2: Determine how you are going to solve the problem

- **Always many** ways to solve a problem
- some solutions are good & some of them are **bad**.
  - Too often, a programmer will get an idea, sit down, and immediately start coding a solution. This often generates a solution that falls into the bad category.
  - Studies show: only 20% of a programmer’s time is spent writing the initial program. The other **80%** is spent debugging (fixing errors) or maintaining (adding features to) a program.

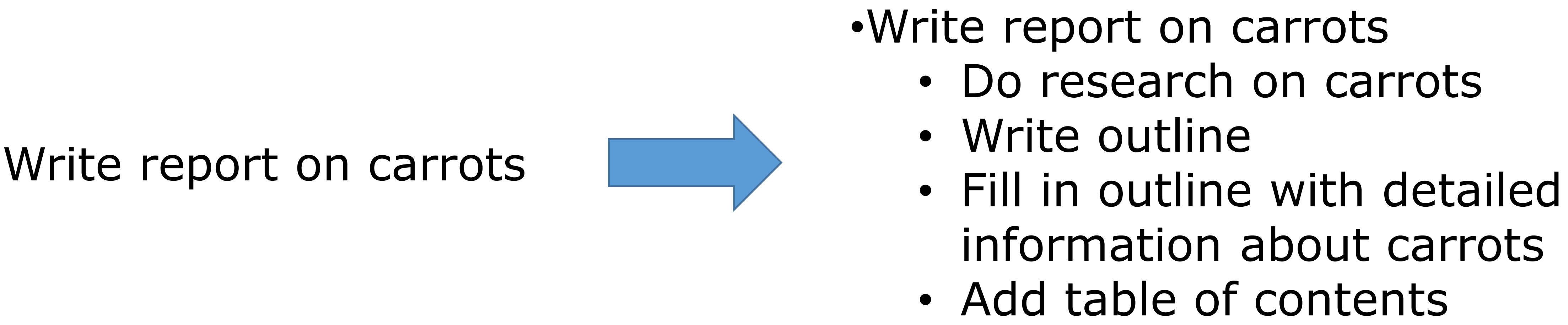
- Good solutions have the following characteristics:
  - They are **straightforward**.
  - They are well **documented** (especially any assumptions being made).
  - They are built modularly, so parts can be **reused** or changed later without impacting other parts of the program.
  - They are robust, and can recover or give useful **error messages** when something unexpected happens.

Step 3: Define your tools, targets, and backup plan

- When you are an experienced programmer:
  - Understanding who your target **users** are and what they **want**.
  - Defining what **target** architecture and/or **OS** your program will run on.
  - Determining what set of **tools** you will be using.
  - Determining whether you will write your program alone or as part of a **team**.
  - Collecting requirements (a documented list of what the program needs to do).
  - Defining your **testing/feedback/release** strategy.
  - **Determining how you will back up your code - version control (e.g. github).**

Step 4: Break hard problems down into easy problems

- **continuously** splitting complex tasks into simpler ones until each of them is manageable
- **top down**



Write report on carrots

- Write report on carrots
  - Do research on carrots
  - Write outline
  - Fill in outline with detailed information about carrots
  - Add table of contents

Write report on carrots

- Write report on carrots
    - Do research on carrots
    - Write outline
    - Fill in outline with detailed information about carrots
    - Add table of contents

- Write report on carrots
    - Do research on carrots
        - Go to library and get book on carrots
        - Look for information about carrots on internet
        - Take notes on relevant sections from reference material
    - Write outline
        - Information about growing
        - Information about processing
        - Information about nutrition
    - Fill in outline with detailed information about carrots
    - Add table of contents

bottom up

- Pick out clothes
- Get dressed
- Eat breakfast
- Drive to work
- Brush your teeth
- Get out of bed
- Prepare breakfast
- Get in your car
- Take a shower

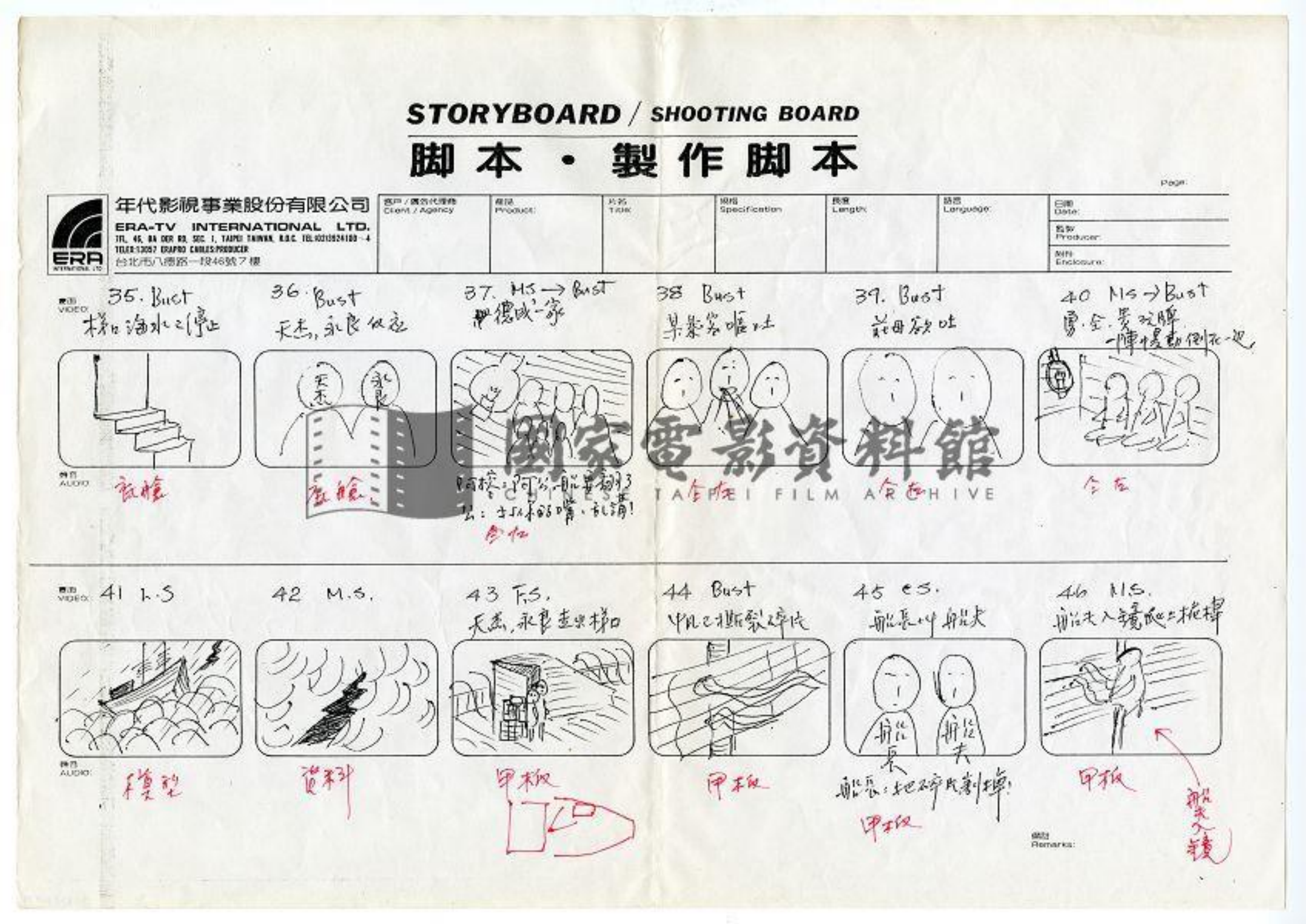
→

- Get from bed to work
  - Bedroom things
    - Get out of bed
    - Pick out clothes
  - Bathroom things
    - Take a shower
    - Brush your teeth
  - Breakfast things
    - Prepare breakfast
    - Eat breakfast
  - Transportation things
    - Get in your car
    - Drive to work

task hierarchy

- The top level task => main()
  - “Write a report on carrots”
  - “Get from bed to work”
- The subitems => functions

Step 5: Figure out the sequence of events



STORYBOARD / SHOOTING BOARD

脚本・製作脚本

Page:

年代影視事業股份有限公司
ERA-TV INTERNATIONAL LTD.
7FL, 46, BA DER RD, SEC. 1, TAIPEI TAIWAN, R.O.C. TEL:(02)2924180-4
TELEX:13057 ERAPRO CABLES:PRODUCER
台北市八德路一段46號7樓

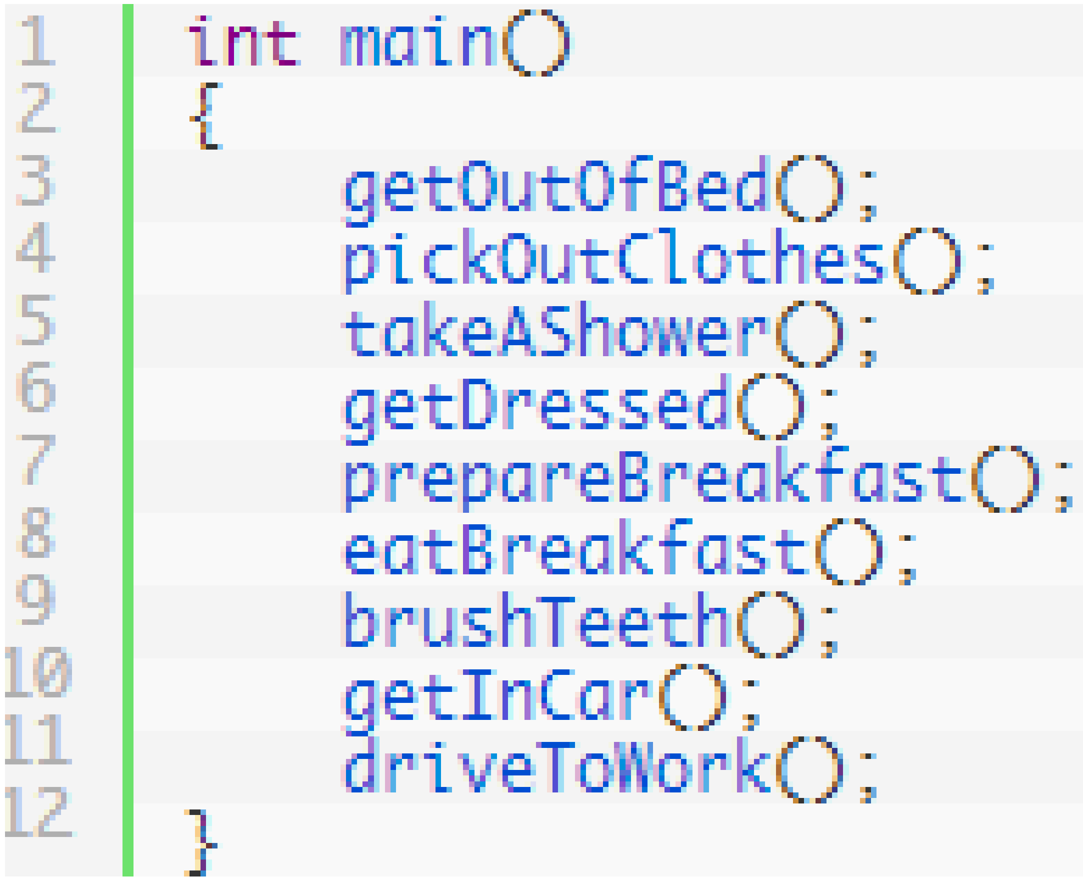
客戶／廣告代理商 Client / Agency:	產品 Product:	片名 Title:	規格 Specification:	長度 Length:	語言 Language:	日期 Date:
						監製 Producer:
						附件 Enclosure:

畫面 VIDEO:	35. Bust 梯口海水之停止	36. Bust 天杰，永良似在	37. MS → Bust 想德成一家	38 Bust 某些客嘔吐	39. Bust 前母欲吐	40 MS → Bust 勇.金.旻改瞭 一陣搖動倒在一起
聲音 AUDIO:	底艙	底艙	阿樑：阿公，船要翻了 公：去ㄙ个烏鴉嘴，亂講！ 仝左	仝左	仝左	仝左

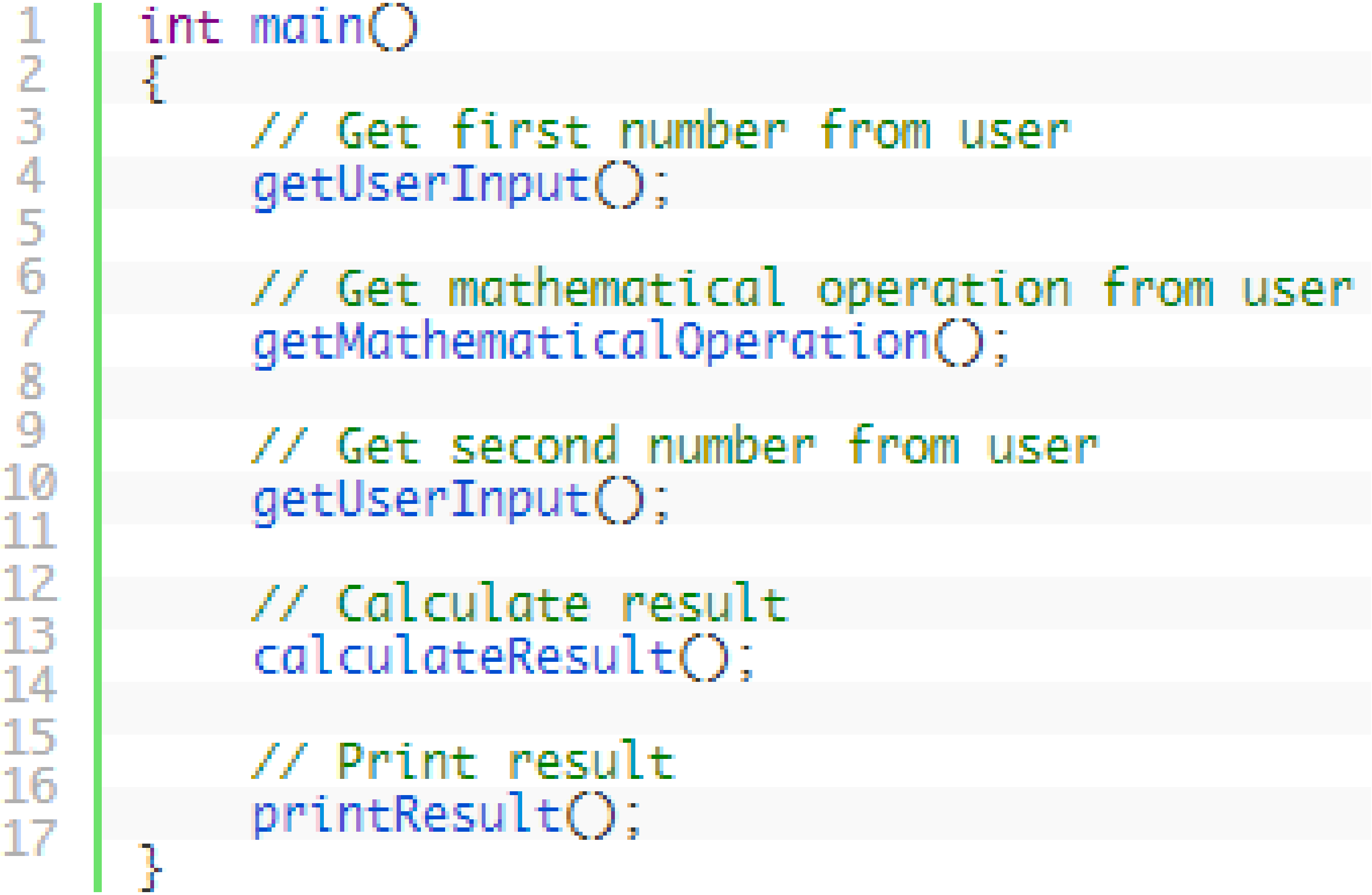
畫面 VIDEO:	41 L.S	42 M.S.	43 F.S. 天杰，永良走出梯口	44 Bust 帆之撕裂碎片	45 C.S. 船長叫船夫	46 M.S. 船夫入鏡爬上桅桿
聲音 AUDIO:	模型	資料	甲板	甲板	船長：把碎片割掉！ 甲板	甲板 船夫入鏡

備註 Remarks:

Step 6: Figure out the sequence of events



```
int main()
{
        getOutOfBed();
        pickOutClothes();
        takeAShower();
        getDressed();
        prepareBreakfast();
        eatBreakfast();
        brushTeeth();
        getInCar();
        driveToWork();
}
```



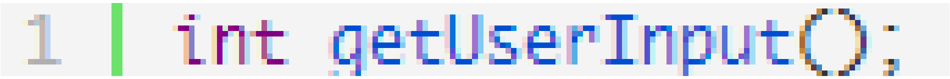
```
int main()
{
        // Get first number from user
        getUserInput();

        // Get mathematical operation from user
        getMathematicalOperation();

        // Get second number from user
        getUserInput();

        // Calculate result
        calculateResult();

        // Print result
        printResult();
}
```

- comment each of these out until you actually write them
- work on them **one at a time**, **testing** each as you go.

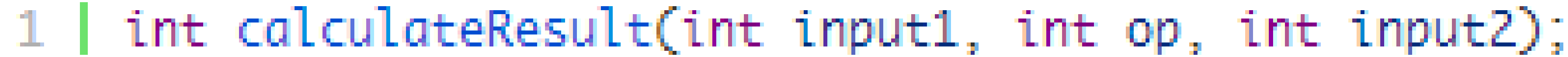
Step 7: Figure out the data inputs and outputs for each task

- a hierarchy and a sequence of events
- input data => parameters.
- output => a return value or parameters

function prototype



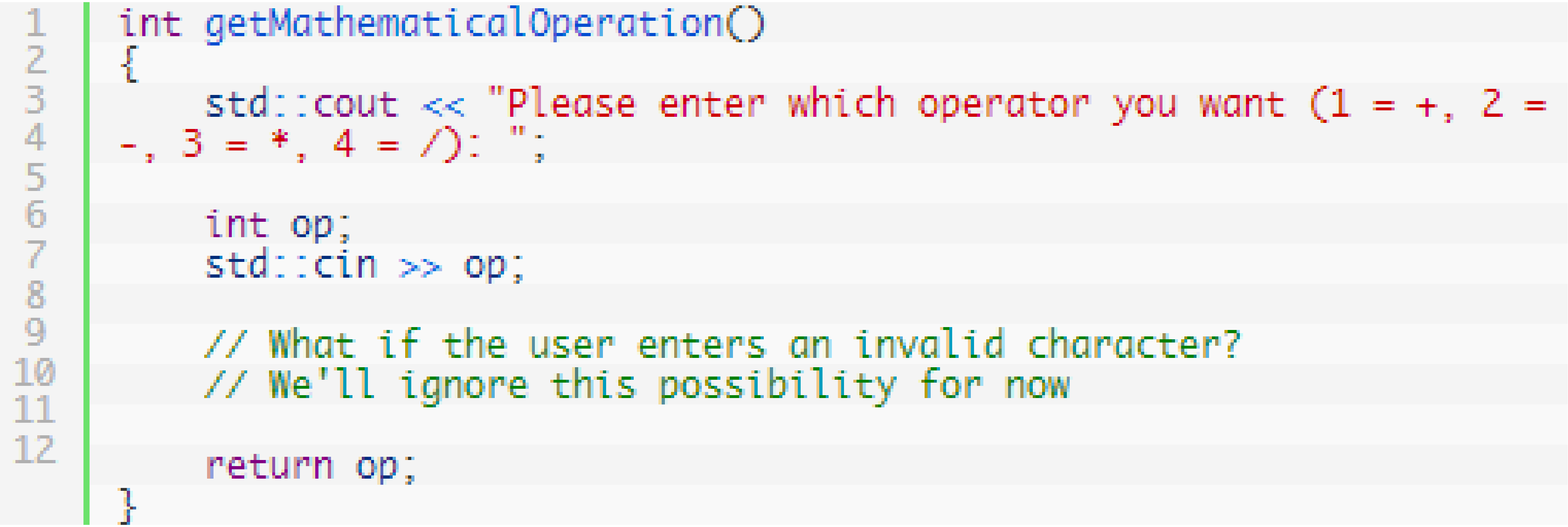
```
int getUserInput();
```



```
int calculateResult(int input1, int op, int input2);
```

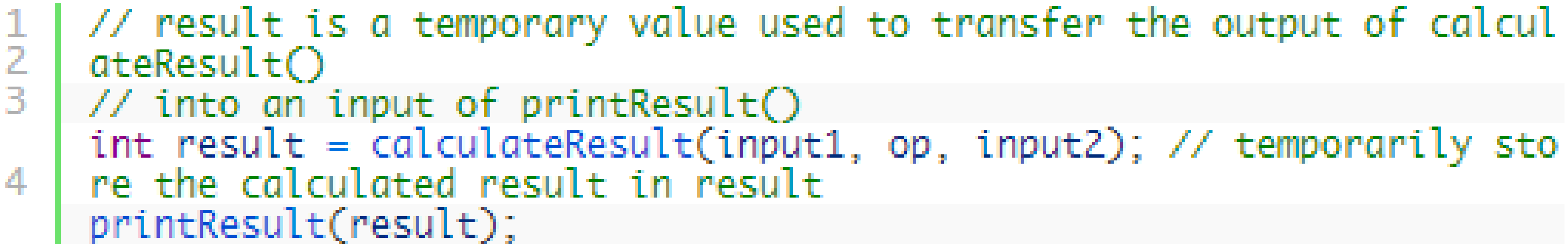
Step 8: Write the task details

- write implementation.
    - simple & straightforward or
    - be broken down into subtasks.

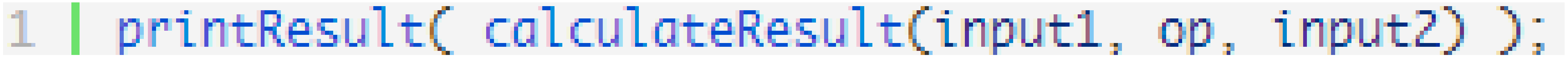


```
int getMathematicalOperation()
{
    std::cout << "Please enter which operator you want (1 = +, 2 = -, 3 = *, 4 = /): ";

    int op;
    std::cin >> op;

    // What if the user enters an invalid character?
    // We'll ignore this possibility for now

    return op;
}
```

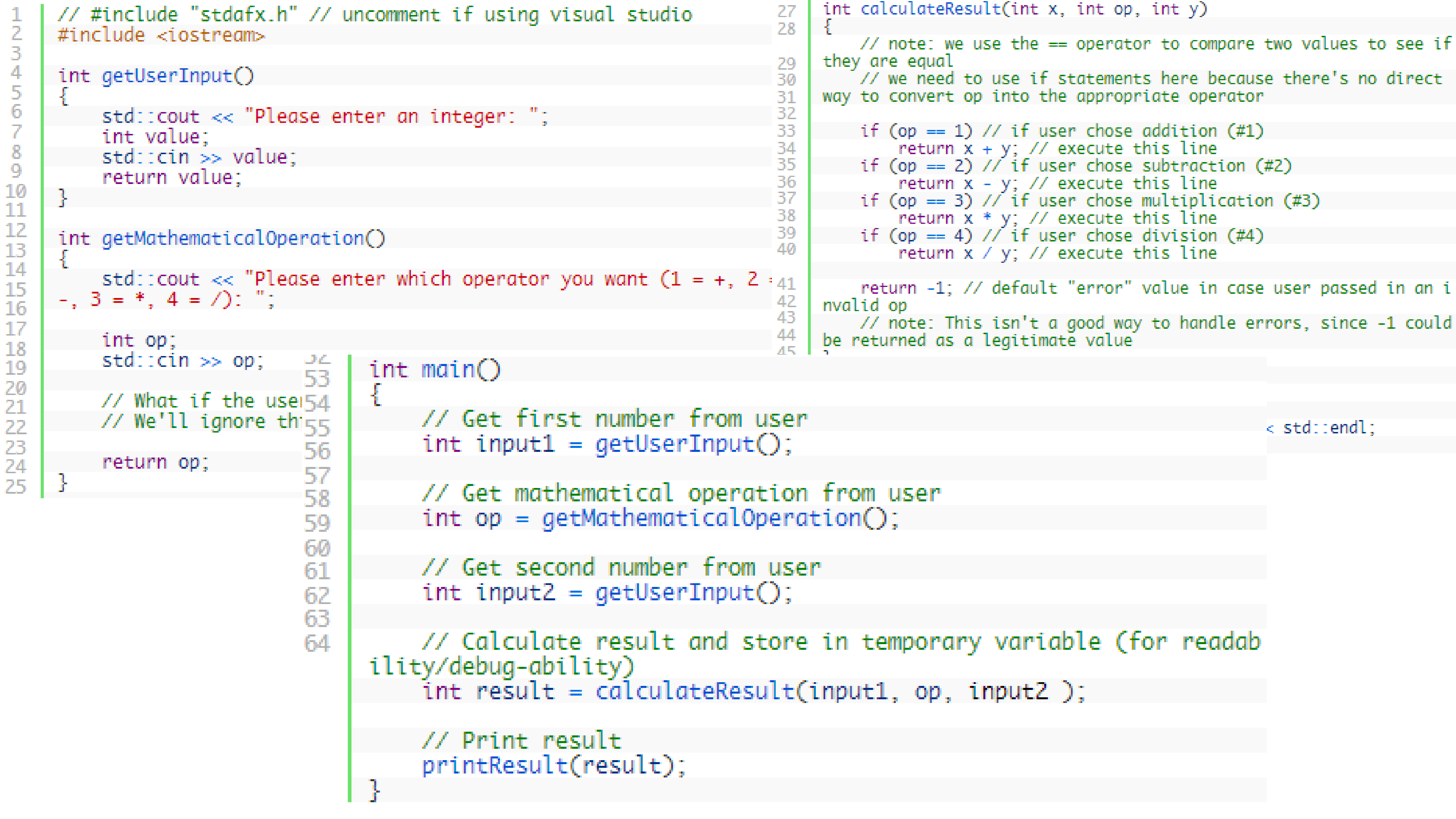
Step 9: Connect the data inputs and outputs



```
// result is a temporary value used to transfer the output of calculateResult()
// into an input of printResult()
int result = calculateResult(input1, op, input2); // temporarily store the calculated result in result
printResult(result);
```



```
printResult( calculateResult(input1, op, input2) );
```



```cpp
// #include "stdafx.h" // uncomment if using visual studio
#include <iostream>

int getUserInput()
{
	std::cout << "Please enter an integer: ";
	int value;
	std::cin >> value;
	return value;
}

int getMathematicalOperation()
{
	std::cout << "Please enter which operator you want (1 = +, 2 
-, 3 = *, 4 = /): ";

	int op;
	std::cin >> op;

	// What if the use
	// We'll ignore th

	return op;
}
```

```cpp
int calculateResult(int x, int op, int y)
{
	// note: we use the == operator to compare two values to see if
they are equal
	// we need to use if statements here because there's no direct
way to convert op into the appropriate operator

	if (op == 1) // if user chose addition (#1)
		return x + y; // execute this line
	if (op == 2) // if user chose subtraction (#2)
		return x - y; // execute this line
	if (op == 3) // if user chose multiplication (#3)
		return x * y; // execute this line
	if (op == 4) // if user chose division (#4)
		return x / y; // execute this line

	return -1; // default "error" value in case user passed in an i
nvalid op
	// note: This isn't a good way to handle errors, since -1 could
be returned as a legitimate value
```

```cpp
< std::endl;
```

```cpp
int main()
{
	// Get first number from user
	int input1 = getUserInput();

	// Get mathematical operation from user
	int op = getMathematicalOperation();

	// Get second number from user
	int input2 = getUserInput();

	// Calculate result and store in temporary variable (for readab
ility/debug-ability)
	int result = calculateResult(input1, op, input2 );

	// Print result
	printResult(result);
}
```

Words of advice when writing programs

- **Keep your programs simple to start**.
  - Grand vision
    - “I want to write a role-playing game with graphics and sound and random monsters and dungeons, with a town you can visit to sell the items that you find in the dungeon”
  - overwhelmed and discouraged at your lack of progress
- **Add features over time**.
  - have your simple program working and working well
  - then you can add features to it

Words of advice when writing programs

- **Focus on one area at a time**

- **Test each piece of code as you go**
  - Not write the entire program in one pass
  - the compiler reports hundreds of errors
  - not only be intimidating, if your code doesn’t work, it may be hard to figure out why
  - write a piece of code, and then compile and test it immediately.
  - If it doesn’t work, you’ll know exactly where the problem is, and it will be easy to fix.
  - Once you are sure that the code works, move to the next piece and repeat

Too many steps & suggestions?

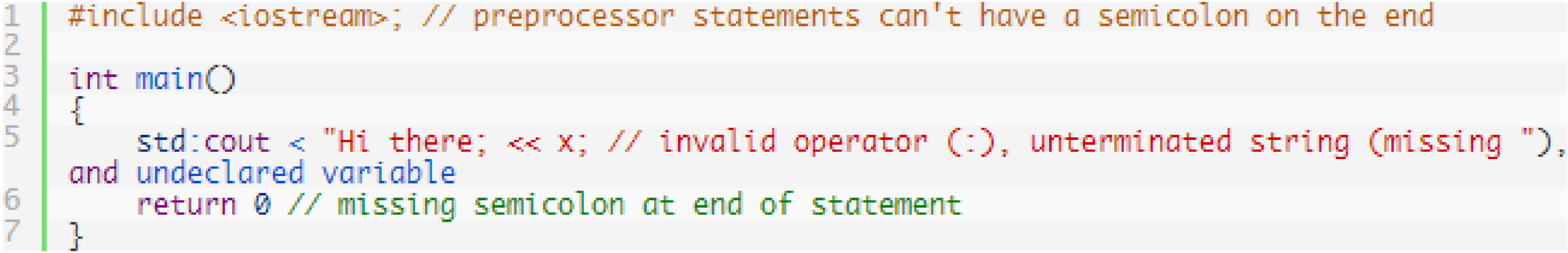
- following these steps will definitely save you a lot of time in the long run.
- A little planning up front saves a lot of debugging at the end.

- **Good news**:
  - they will start coming naturally to you without even thinking about it.
  - Eventually you will get to the point where you can write entire functions without any pre-planning at all.

Debugging your program

Syntax and semantic errors

- Errors: syntax errors, & semantic errors (logic errors).
- A **syntax error** not valid according to the grammar of the C++
    - missing semicolons
    - undeclared variables
    - mismatched parentheses or braces
    - unterminated strings

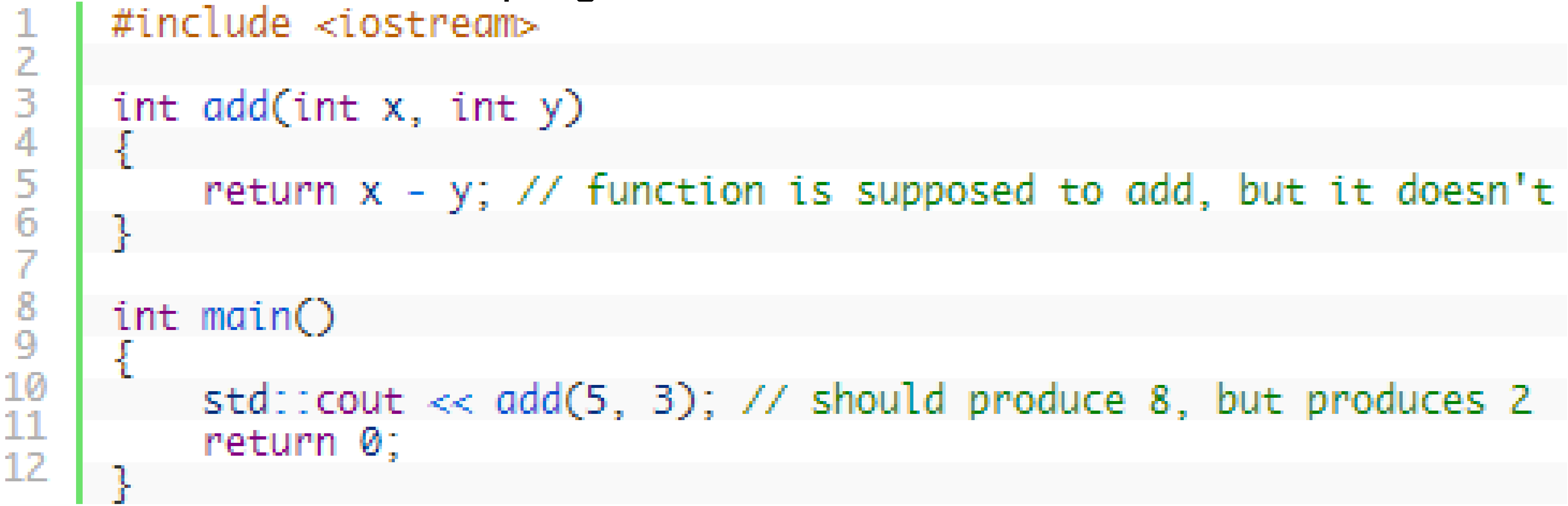


```
#include <iostream>; // preprocessor statements can't have a semicolon on the end

int main()
{
    std:cout < "Hi there; << x; // invalid operator (:), unterminated string (missing "), and undeclared variable
    return 0 // missing semicolon at end of statement
}
```

- compiler will generally catch syntax errors and generate warnings or errors

Syntax and semantic errors

- Errors: syntax errors, & semantic errors (logic errors).
- A **semantic error** occurs when a statement is syntactically valid, but does not do what the programmer intended.



```cpp
#include <iostream>

int add(int x, int y)
{
    return x - y; // function is supposed to add, but it doesn't
}

int main()
{
    std::cout << add(5, 3); // should produce 8, but produces 2
    return 0;
}
```

- compiler will not be able to catch these types of problems => **Debugger**

The debugger

- A **debugger** is a computer program that allows the programmer to control how a program executes and watch what happens as it runs.

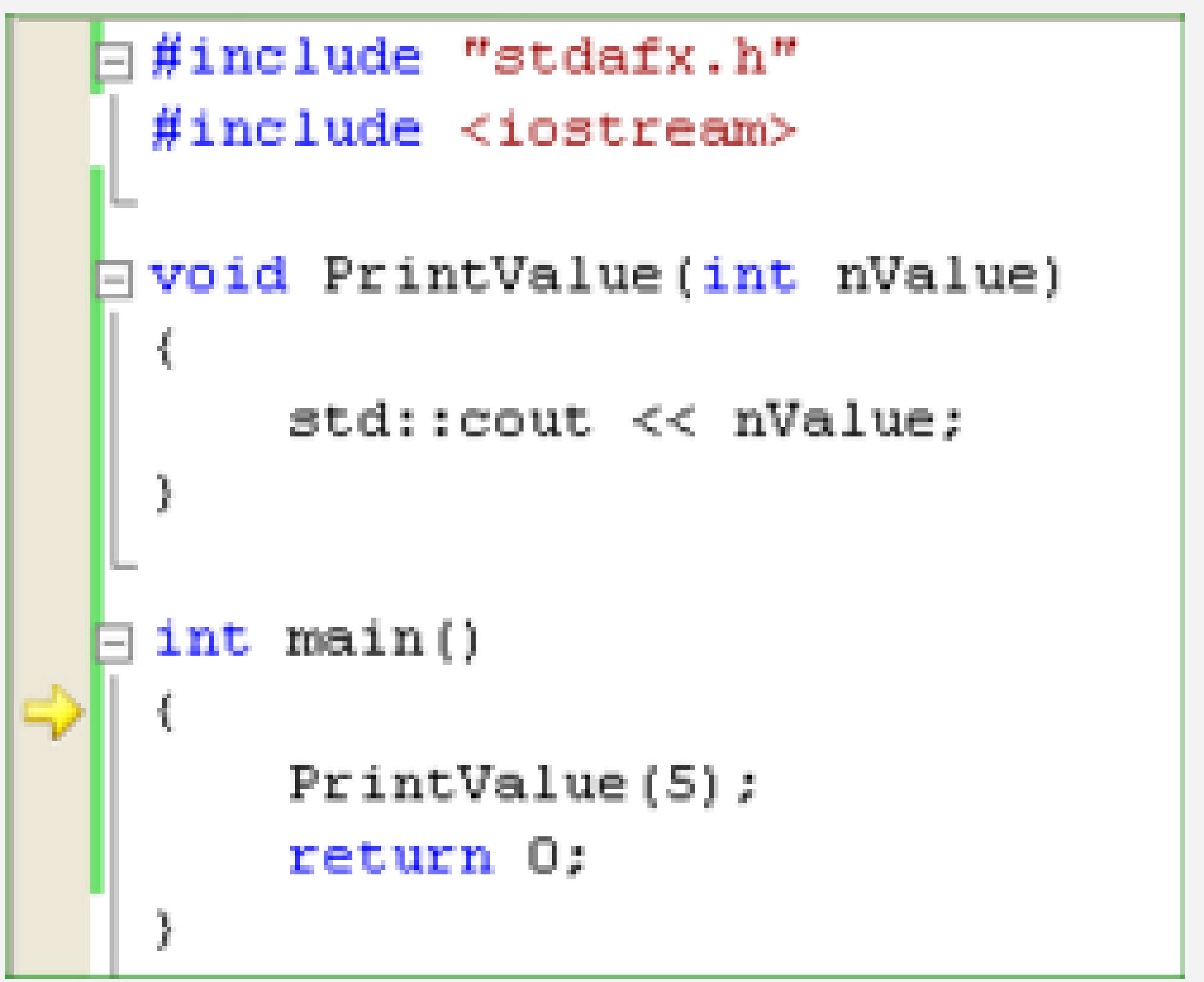
Before proceeding: Make sure your program is set to use the ***debug build configuration****.*

Six Stages of Debugging

1. That can’t happen.
2. That doesn’t happen on my machine.
3. That shouldn’t happen.
4. Why does that happen?
5. Oh, I see.
6. How did that ever work?

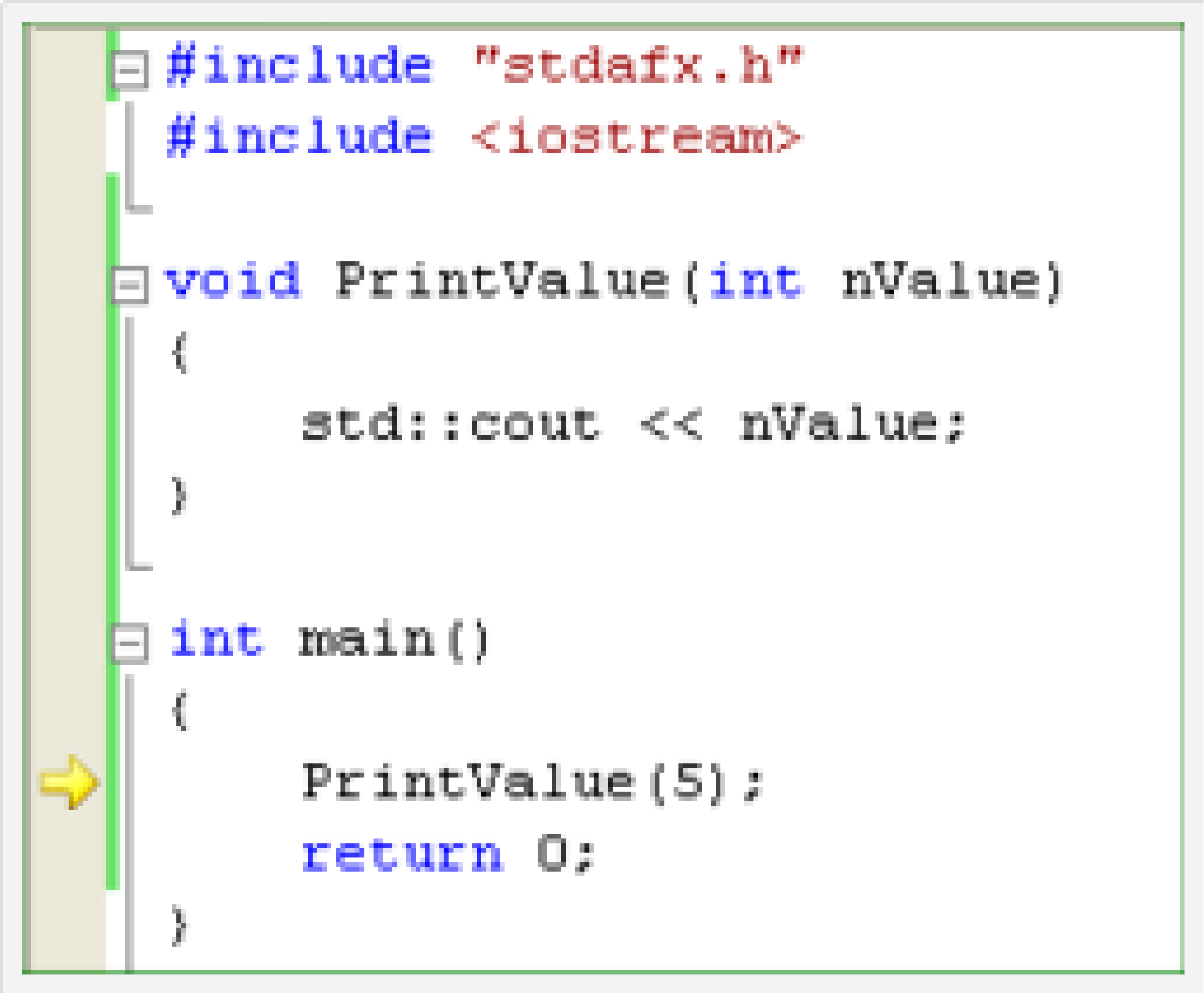
Stepping

- execute (step through) your code line by line
- 3 different stepping commands: step into, step over, and step out
- **step into**
  - executes the next line of code
  - If this line is a function call, step into enters the function and returns control at the top of the function.
  - go to the debug menu and choose “Step Into”, or press F11.
  - arrow indicates:
    - the line being pointed to will be executed next

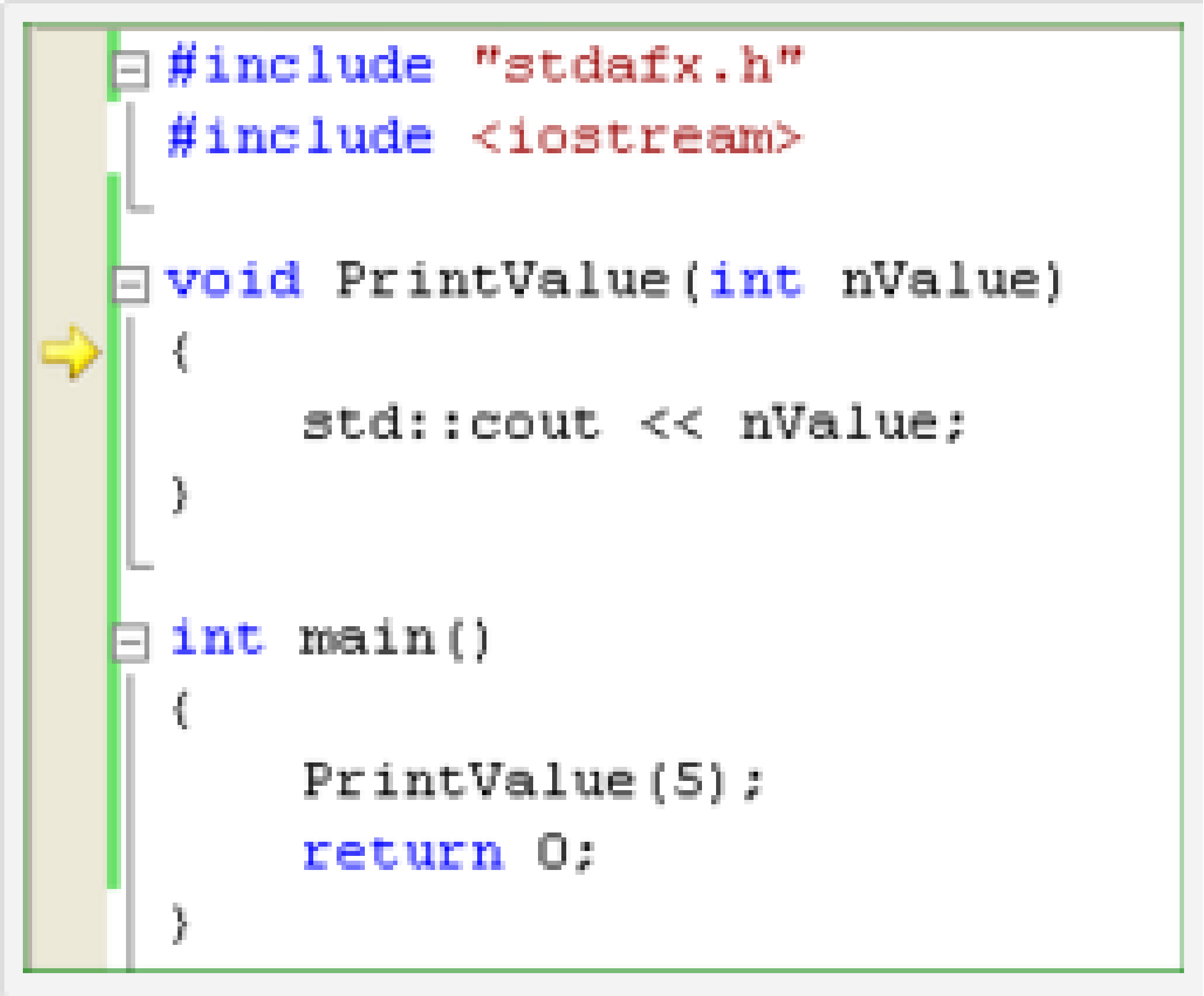


```
#include "stdafx.h"
#include <iostream>

void PrintValue(int nValue)
{
    std::cout << nValue;
}

int main()
{
    PrintValue(5);
    return 0;
}
```

step into

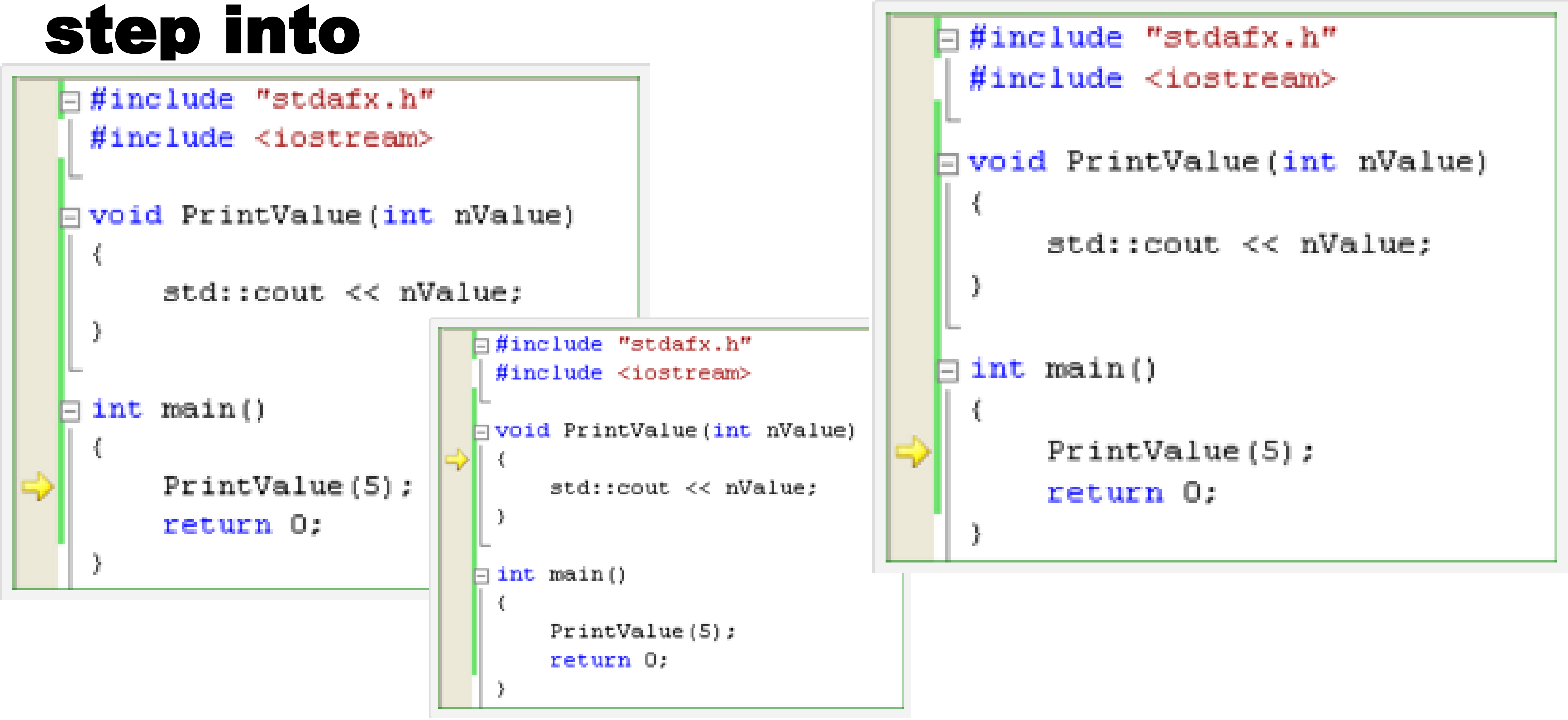


```
#include "stdafx.h"
#include <iostream>

void PrintValue(int nValue)
{
    std::cout << nValue;
}

int main()
{
    PrintValue(5);
    return 0;
}
```



```
#include "stdafx.h"
#include <iostream>

void PrintValue(int nValue)
{
    std::cout << nValue;
}

int main()
{
    PrintValue(5);
    return 0;
}
```

step into



```
#include "stdafx.h"
#include <iostream>

void PrintValue(int nValue)
{
    std::cout << nValue;
}

int main()
{
    PrintValue(5);
    return 0;
}
```

```
#include "stdafx.h"
#include <iostream>

void PrintValue(int nValue)
{
    std::cout << nValue;
}

int main()
{
    PrintValue(5);
    return 0;
}
```

```
#include "stdafx.h"
#include <iostream>

void PrintValue(int nValue)
{
    std::cout << nValue;
}

int main()
{
    PrintValue(5);
    return 0;
}
```

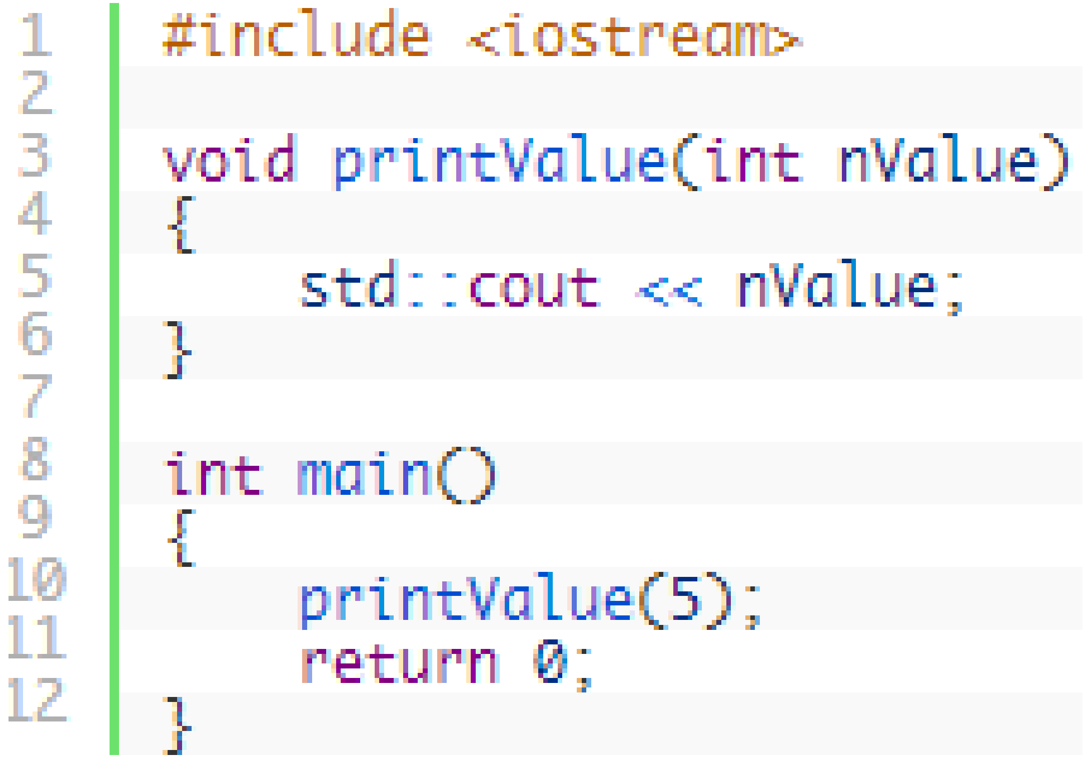
- choose “Stop Debugging” from the debug menu. This will terminate your debugging session.

step into, step over, and step out

- **Step over**
  - command executes the next line of code.
  - If this line is a function call, “Step over” executes all the code in the function
  - and returns control to you after the function has been executed.
- Step out
  - executes all remaining code in the function you are currently in,
  - and returns control to you when the function has finished executing.

Run to cursor, Run

- Run to cursor: executes the program like normal until it gets to the line of code selected by your cursor.

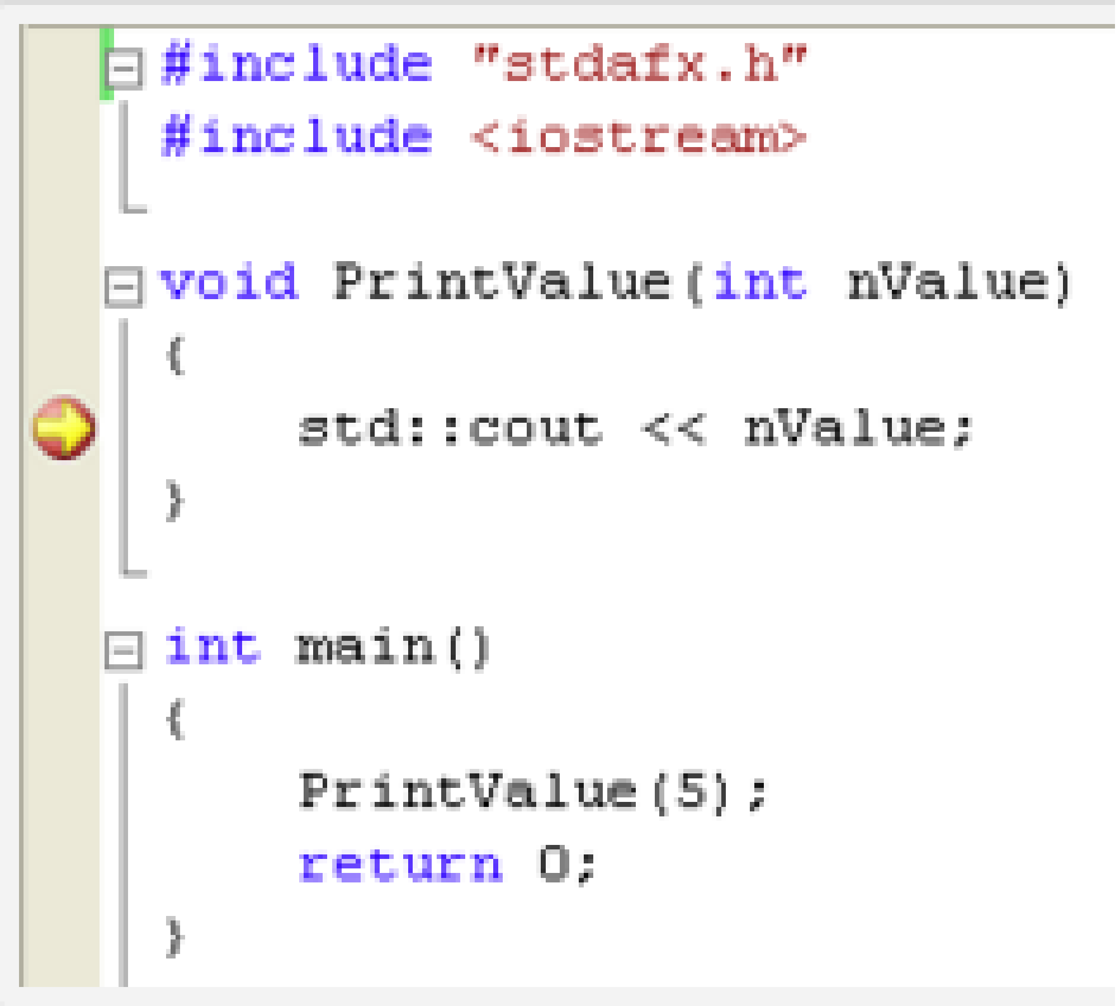


```cpp
#include <iostream>

void printValue(int nValue)
{
    std::cout << nValue;
}

int main()
{
    printValue(5);
    return 0;
}
```

- Simply put your **cursor** on the std::cout << nValue; line inside of printValue(), then right click and choose “Run to cursor”.
- Run: it may be called “Go” or “Continue”

Breakpoints

- tells the debugger to stop execution of the program at the breakpoint when running in debug mode.
- “Toggle Breakpoint” (right click, choose Breakpoint -> Insert Breakpoint).

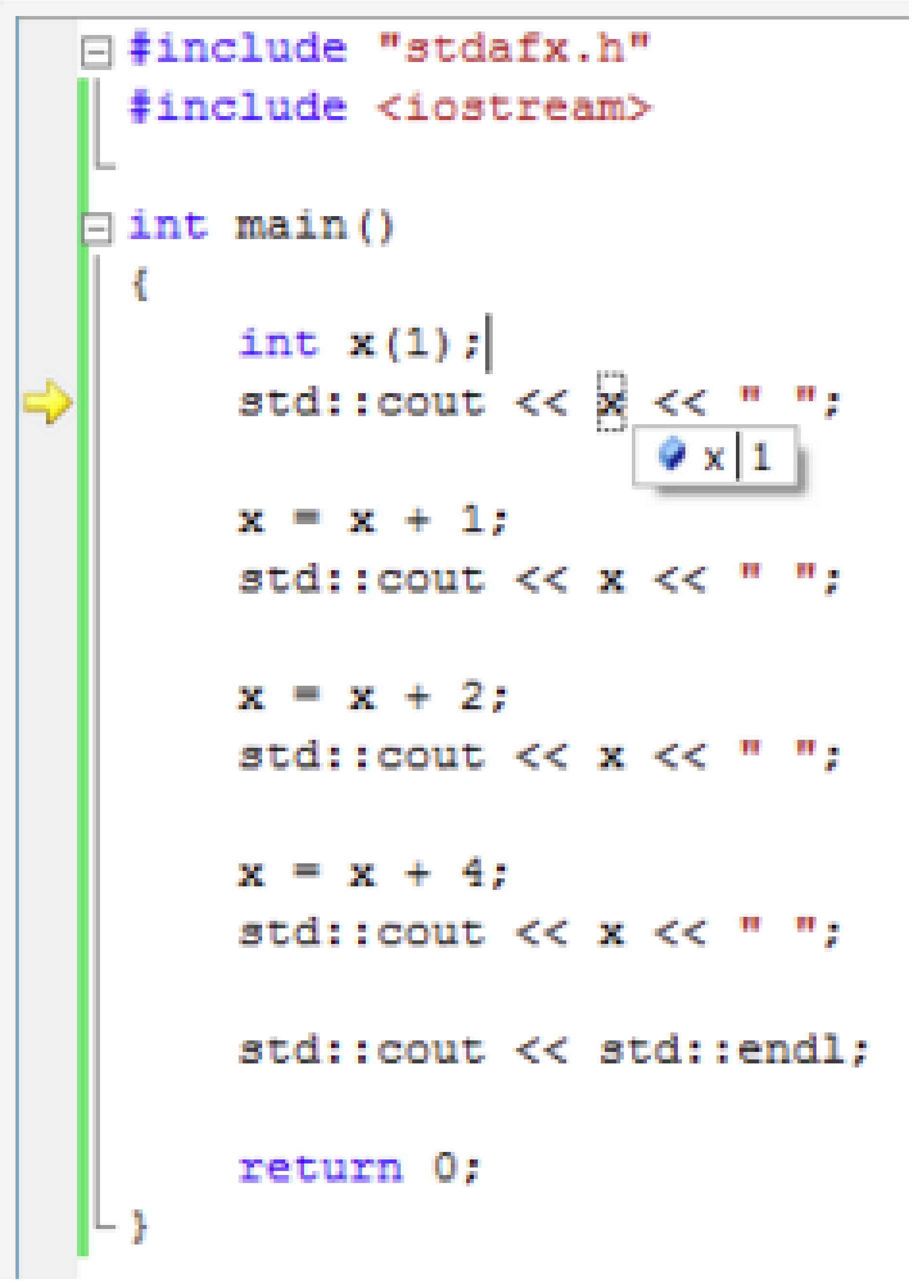


```
#include "stdafx.h"
#include <iostream>

void PrintValue(int nValue)
{
    std::cout << nValue;
}

int main()
{
    PrintValue(5);
    return 0;
}
```

Debugging your program (watching variables and the call stack)

watching variables and the call stack

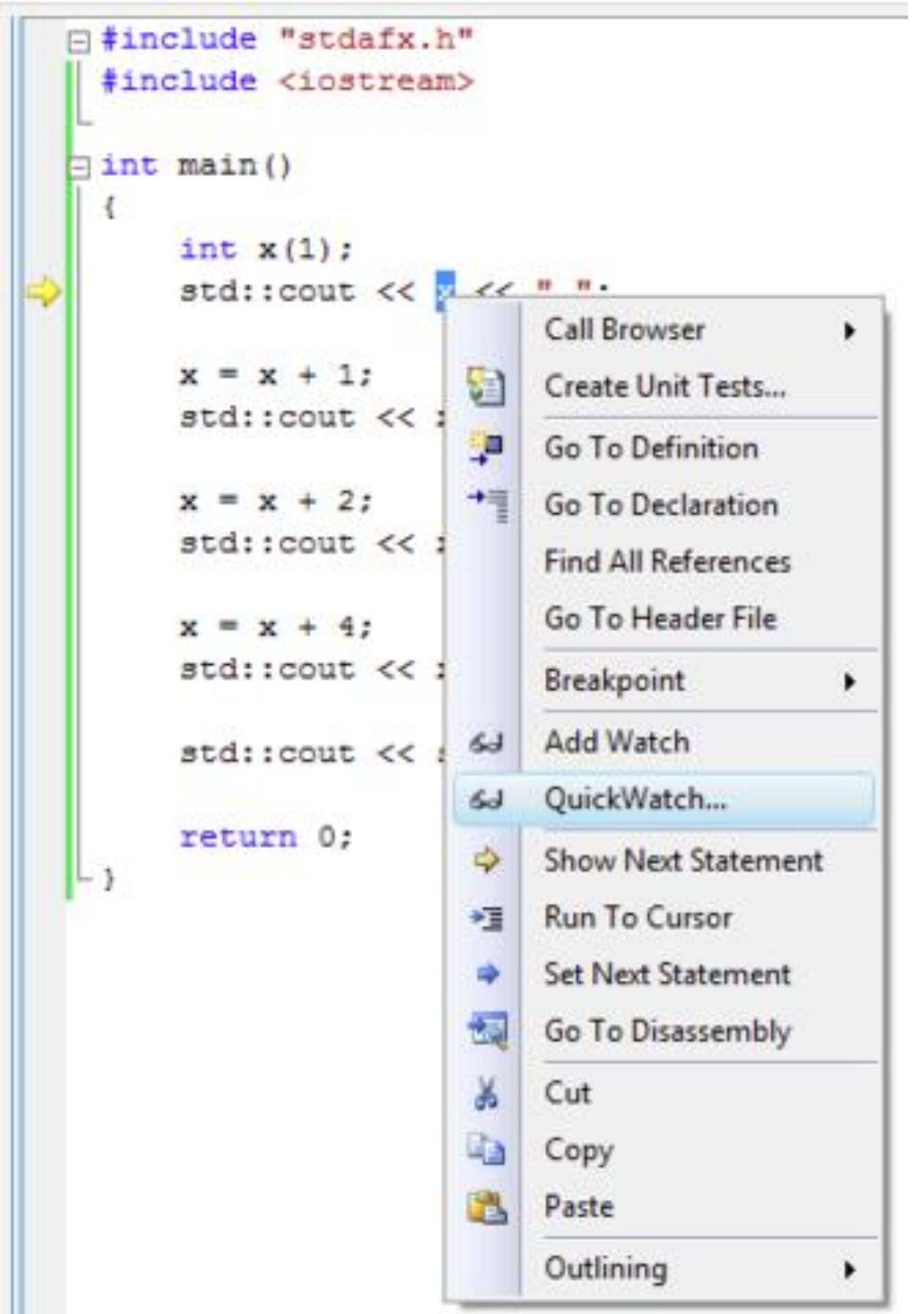
- stepping through a program
- examine the value of variables
- **Watching variables**
    - inspecting the value of a variable while the program is executing in debug mode



```
#include "stdafx.h"
#include <iostream>

int main()
{
	int x(1);
	std::cout << x << " ";

	x = x + 1;
	std::cout << x << " ";

	x = x + 2;
	std::cout << x << " ";

	x = x + 4;
	std::cout << x << " ";

	std::cout << std::endl;

	return 0;
}
```

AddWatch

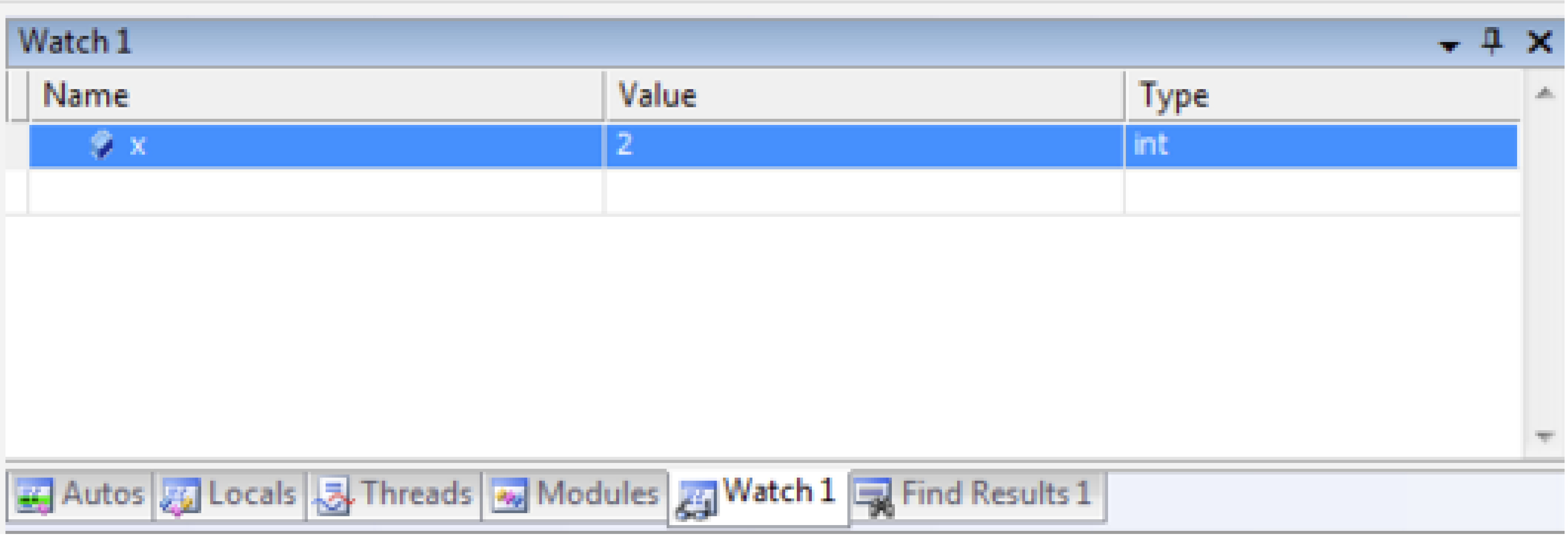
- Highlight the variable name x with your mouse,
- choose “AddWatch” from the right-click menu.



```
#include "stdafx.h"
#include <iostream>

int main()
{
    int x(1);
    std::cout << x << " ";

    x = x + 1;
    std::cout <<

    x = x + 2;
    std::cout <<

    x = x + 4;
    std::cout <<

    std::cout <<

    return 0;
}
```

The watch window

- where you can add variables you would like to continually inspect
- and these variables will be updated as you step through your program.
- Debug Menu->Windows->Watch->Watch 1
    - note: you have to be in debug mode, so step into your program first

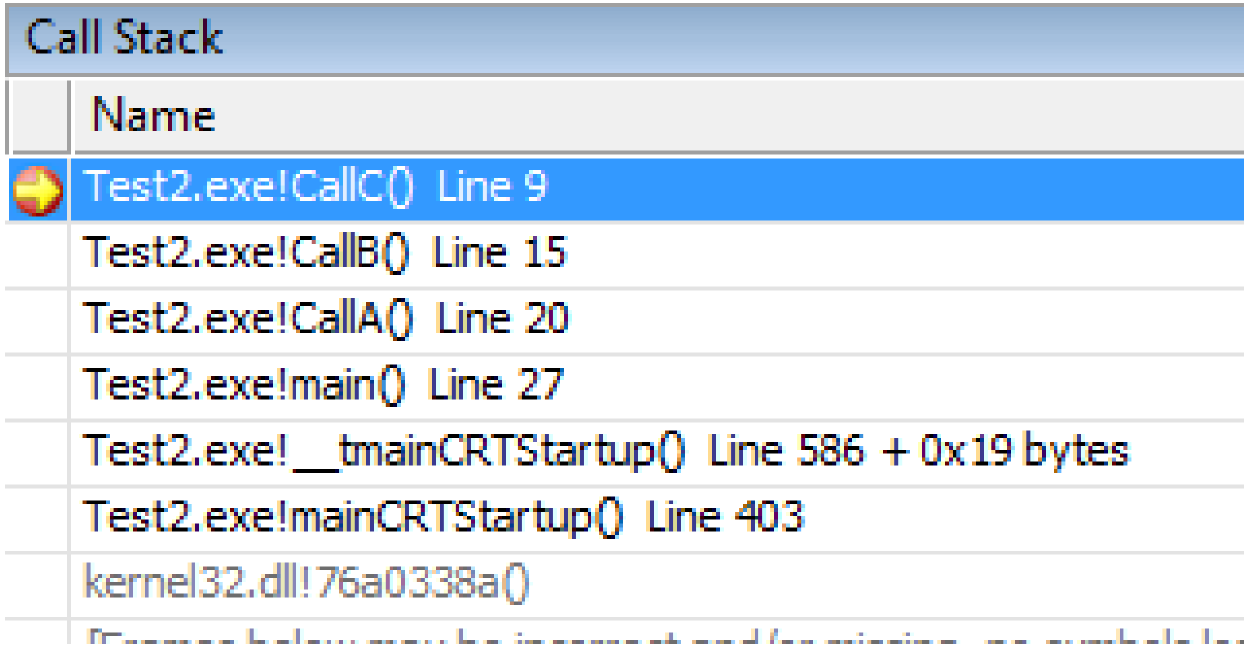


Watch 1

Name	Value	Type
x	2	int

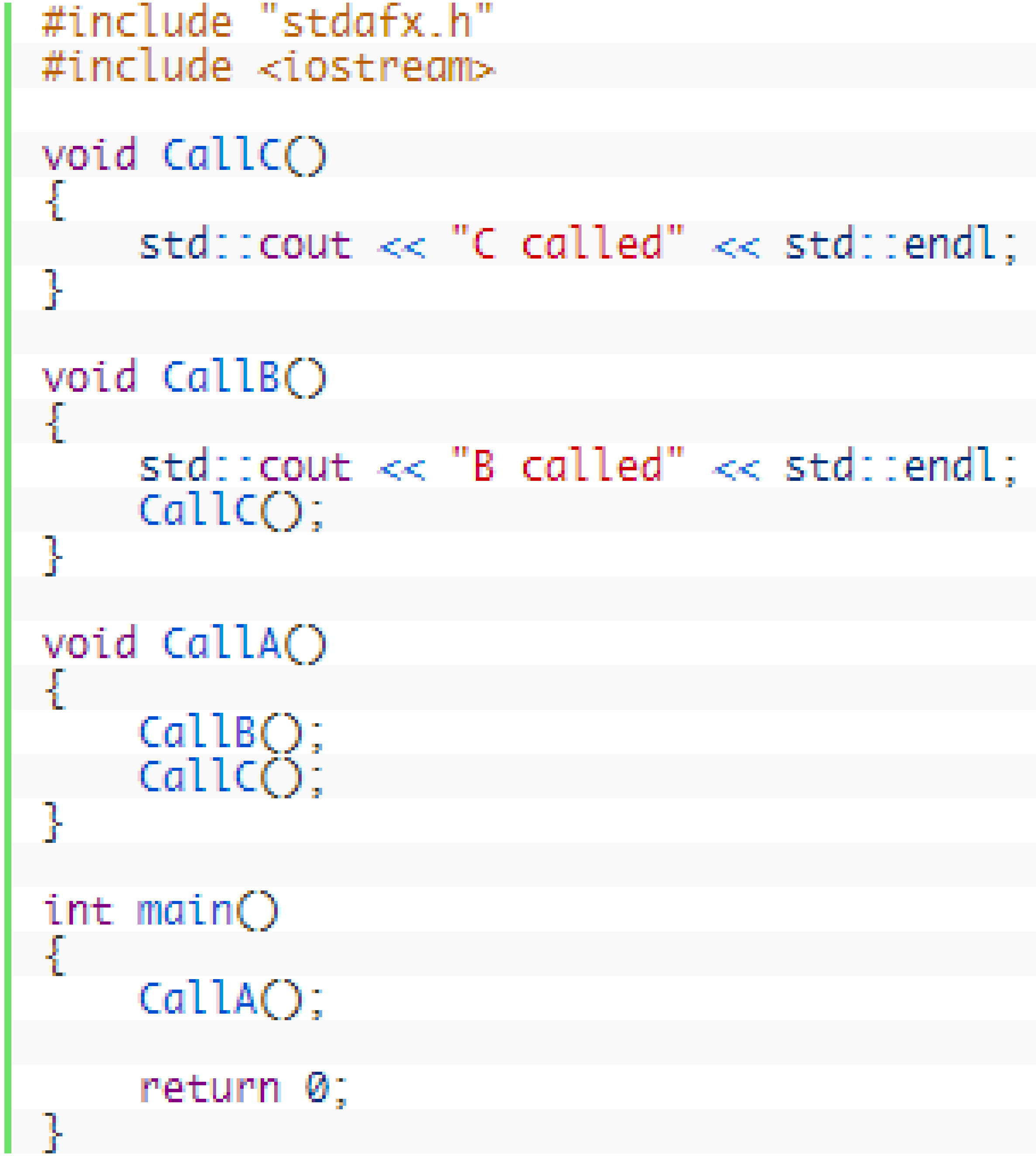
The call stack window

- a list of all the active functions that have been called to get to the current point of execution.
- Debug Menu->Windows->Call Stack



Name
Test2.exe!CallC() Line 9
Test2.exe!CallB() Line 15
Test2.exe!CallA() Line 20
Test2.exe!main() Line 27
Test2.exe!__tmainCRTStartup() Line 586 + 0x19 bytes
Test2.exe!mainCRTStartup() Line 403
kernel32.dll!76a0338a()
[Frames below may be incorrect and/or missing, no symbols lo

- **double-click** on the various lines



```
#include "stdafx.h"
#include <iostream>

void CallC()
{
    std::cout << "C called" << std::endl;
}

void CallB()
{
    std::cout << "B called" << std::endl;
    CallC();
}

void CallA()
{
    CallB();
    CallC();
}

int main()
{
    CallA();

    return 0;
}
```

Conclusion

- Design, test, coding
  - Purpose
  - Divide and conquer => Functions
  - Sequence/hierarchy of events
  - Function prototype
- Debug
  - Stepping
  - Breakpoints
  - Watches
  - Call stack window

- Takes practice, trial & error
- Definitely worth your time investment!