FCPA 2022  
  
Writing C in   
Visual Studio 2017

Student Workbook 02

January 10, 2022

Author

Paul Kimball  
Interface Associates

1. A Crash Course in Visual Studio

# Visual Studio

## Integrated Development Environment (IDE) from Microsoft

### Supported on Windows and macOS

* + Enterprise, Professional, and Community (free) versions
  + Supported since 1997
* Supports numerous Microsoft languages and environments

### Visual Basic, C#, **C/C++**, JavaScript, Python, F# etc.

### Azure server, desktop and mobile platforms

* Later releases = better support for Linux and embedded platforms

### Clang, Windows Subsystem for Linux (WSL), bash

* Customizable and extendable

### Downloadable modules from Microsoft for diverse workloads

### Public extensions from the Visual Studio Marketplace

### Public .NET libraries from NuGet so you don't have to code everything

### Customizations for fonts, themes, highlighting, formatting, etc.

## Documentation:

### <https://docs.microsoft.com/en-us/visualstudio>

# Visual Studio Installation

## Download from:

### <https://visualstudio.microsoft.com/downloads/>

* Select version that you want
  + 2017, 2019, 2022
  + Enterprise, Professional, Community (free) version
  + Windows, macOS installers
  + 64-bit and 32-bit versions
  + Install for individual user (does not require admin rights)
  + Install as administrator for all users
* Download and run the installer

### Microsoft compilers and tools are installed to match your selected workloads

* Every tool has it's own way of seeing the world

### In the next few pages we'll cover Visual Studio core concepts

# Workloads

* Visual Studio supports many programming languages, execution environments, and plugins

### *You don't need them all, especially not at the same time.*

* A "workload" is a way of using VS to produce a certain kind of software product

### We will be writing "console apps" in C

#### You can execute them at the command line

#### No fancy windows or dialog boxes

## Choose which workloads you want to support when you install VS

### Each workload comes with tools, compilers, libraries, services and UI extensions

#### Azure Development

#### .NET desktop development

#### **Desktop development with C++**

#### **Linux Development with C++**

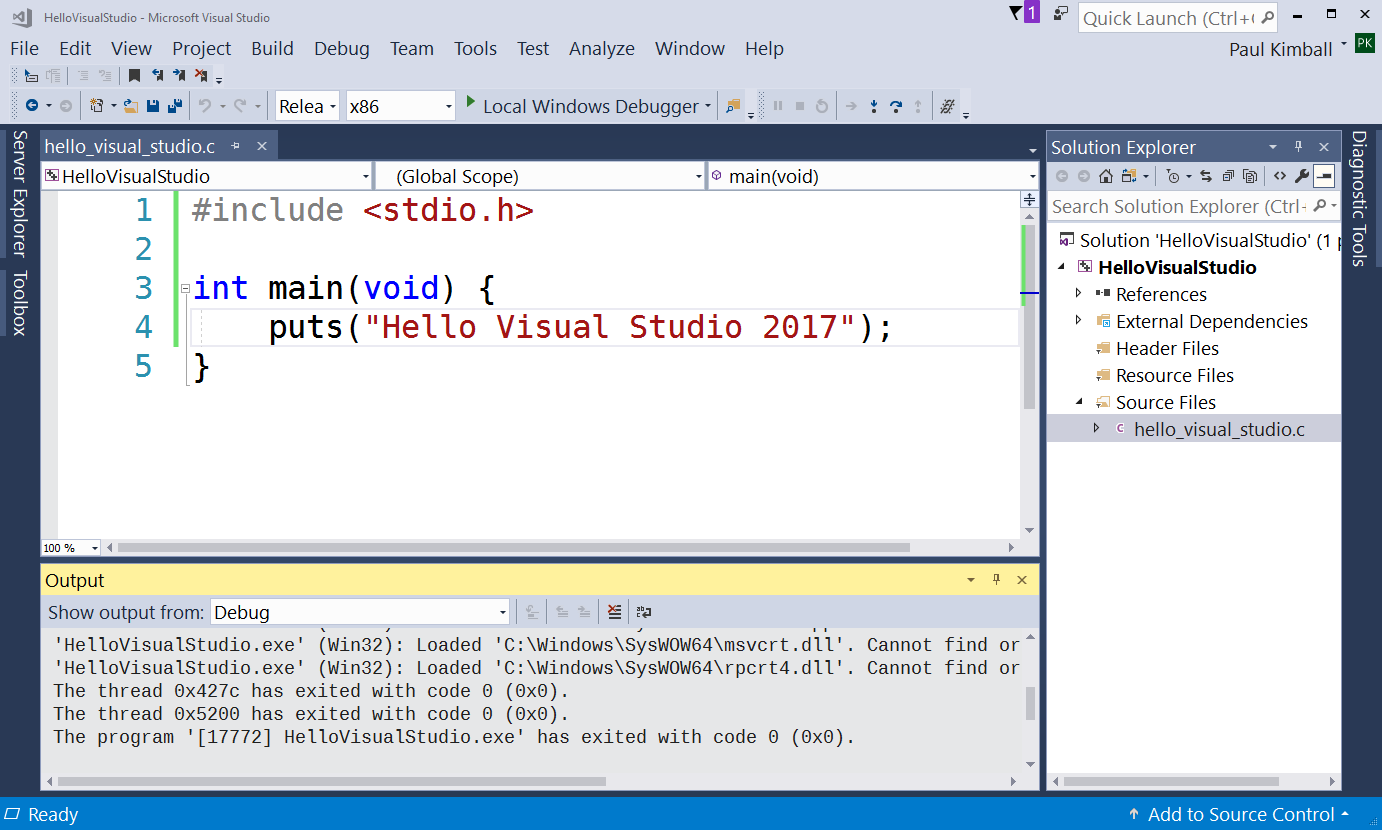
#### Mobile development with .NET/Xamarin

#### many more ...

## You can add and remove features from your installation at any time

### Tools menu -> Get Tools and Features...

# Visual Studio 2017 User Interface



## Old-school, one tool does everything

### Many menus with many choices

### Tons of toolbars with tiny icons

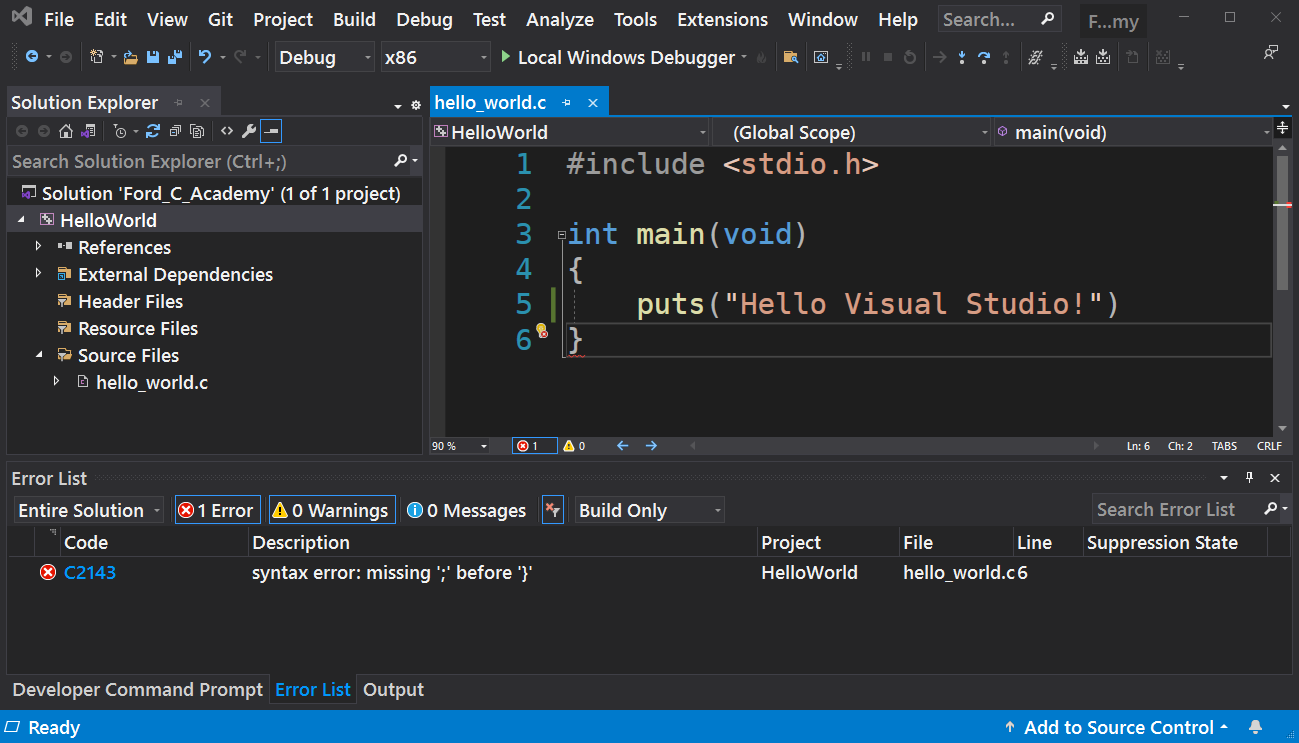
### Many tabbed panes to display different tools, views, files

### Many optional plugins

### Hundreds of settings to control everything from fonts to compilers to code management systems

## Looks impressive...

# Simplify



## Tip #1 - Get rid of unused features and tools

### Install only the features and extensions you need

### You can always get them back

## Tip #2 - Customize UI to minimize clutter and suit your style, e.g.,

### Choose a Color Theme

### Set your fonts

### Close unnecessary views and toolbars

### Move some things around

# How to Add/Remove Features

## Tools menu -> Get Tools and Features ...

### Runs the Visual Studio installer

### Add or remove official Microsoft workloads and options

#### Languages, Build systems, Runtime Environments (WSL, Azure, etc.)

* There are a few other ways to extend Visual Studio, which we'll see later on:
  + Extensions
  + NuGet
  + External Tools

# How to Set Options

## Tools menu -> Options... [Alt-T O]

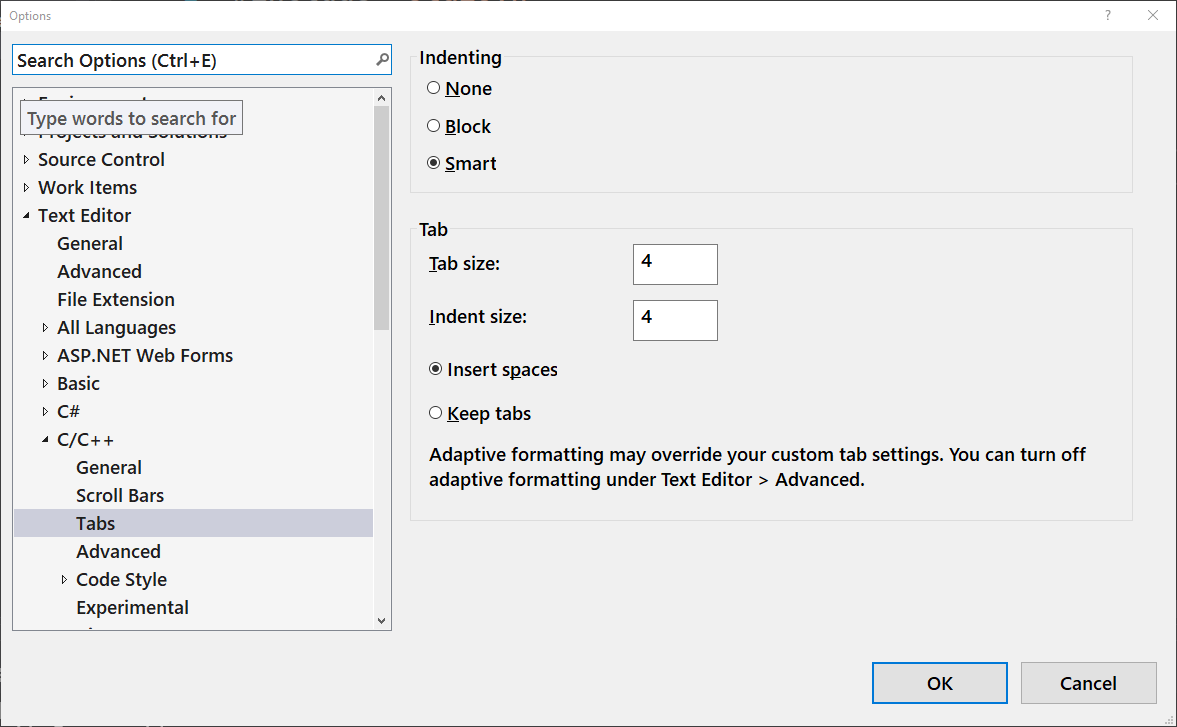
### Grouped in categories/subcategories

#### At least it has a "search" feature

### The "infinite options" dialog is the bane of all big IDEs

#### Eclipse and NetBeans are like this, too

## Don't worry, most of these options you will never change



## Make a cheat sheet for options you care about; google everything else

Personalize the UI

## Move the views to suit yourself

### Views and Toolbars are draggable, dockable, pinnable

## Select which views you want to see

### Show a view:

#### **View** menu -> select view

### Close a view:

#### Select X icon in title or tab

#### Never be afraid to close a view; you can always get it back

## Set your UI theme

### **Tools menu -> Options... [Alt-T O]**

#### Category: **Environment / General**

#### Select **Color Theme** (it also saves your font changes)

## Choose fonts for editors, terminals, menus and other views

### **Tools menu -> Options... [Alt-T O]**

#### Category: **Environment / Fonts and Colors**

### Then, select "Show settings for:"

#### **Environment** - set font for UI menus, tabs, lists - (including Solution Explorer) May be called "Environment Font" on earlier releases

#### **Text Editor** - sets font for code editors

#### **Terminal** (VS2019+) - set font for command prompts and shells

#### etc.

## Customize how the editor formats your code

### **Tools menu -> Options... [Alt-T O]**

#### Category: **Text Editor / C/C++**

### Then select options in subcategories

#### **General** / Line numbers [ON]

#### **General** / Word wrap [ON]

#### **Code Style / Linter** -> ... select your preferences

#### **Code Style / Formatting** -> ... select your preferences

# Toolbars and Shortcuts

## Tools menu -> Customize... [Alt-T C]

### Customizations for more adventurous coders

#### Customize which toolbars are displayed by default

#### Add/remove/rearrange/rename toolbar and menu items

#### Remap keyboard shortcuts

# Projects and Solutions

## Projects organize your source code

### A project goes in its own directory

#### Directory name == Project name

### Contains the source code and resources to build a particular software artifact, e.g.,

#### An executable program that you can run at the command prompt

#### A linkable library (.dll or .lib)

### A C/C++ project is represented by a .vcxproj file

## Solutions organize your projects

### A solution goes in its own directory

#### Directory name == Solution name

### May have projects as subdirectories

### A C/C++ solution is represented by a .sln file

## Simple solutions (like "hello world") have only one project

### You can put .sln and .vcxproj in the same directory

## Reference

* + https://docs.microsoft.com/en-us/visualstudio/ide/solutions-and-projects-in-visual-studio

# Important Menu Items for Today

## File menu

* + New -> Project ...
  + Save Selected Items
  + Save All
  + Close Solution
  + Exit

## View menu

* + Code
  + Solution Explorer

## Project

* + Add new item ...

## Build

* + Build Solution

## Debug

* + Start Debugging
  + Start Without Debugging

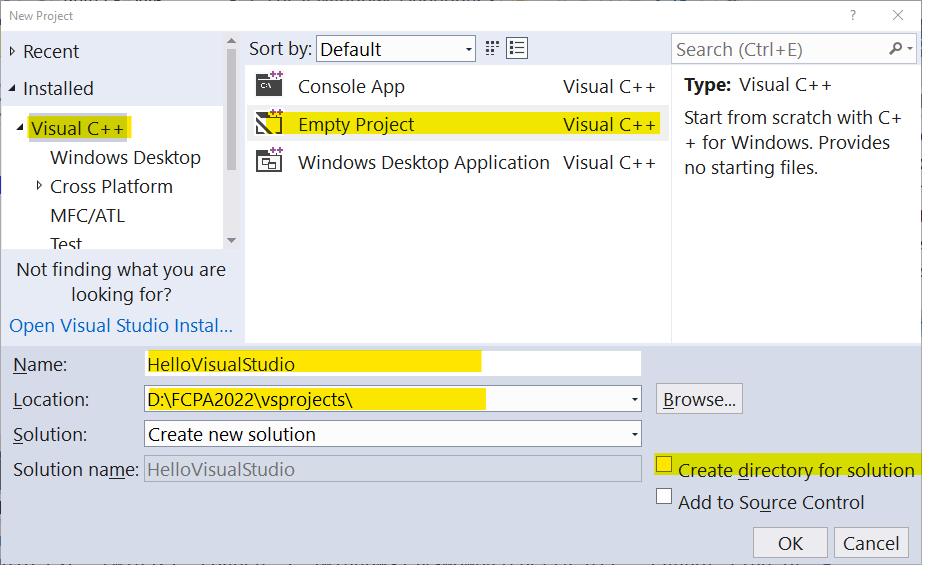
<<Live Demo/Follow Along>>

A crash course in Visual Studio

# Create a Project

## File menu -> New -> Project [Ctrl+Shift+N]

* + Language: Visual C++
  + **Empty Project**
  + Type: Console



## Set the project name

* + This will become the name of the project directory

## Set the location

* + This is the project's parent directory
* Uncheck "Create directory for solution"

## Click OK

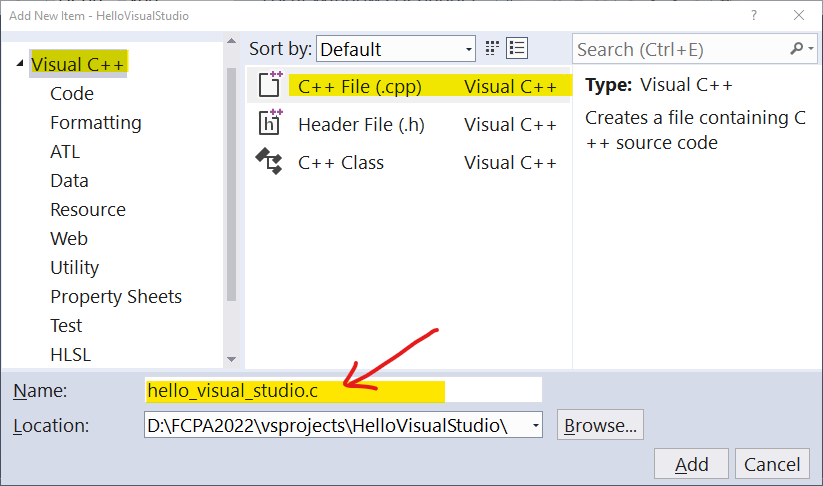
* + You new project appears in the solution explorer

# Create C source file(s)

* Project menu -> Add New Item...

### OR, Right click Source Files -> Add -> New Item...

## Watch out! Don't let VS create a C++ file



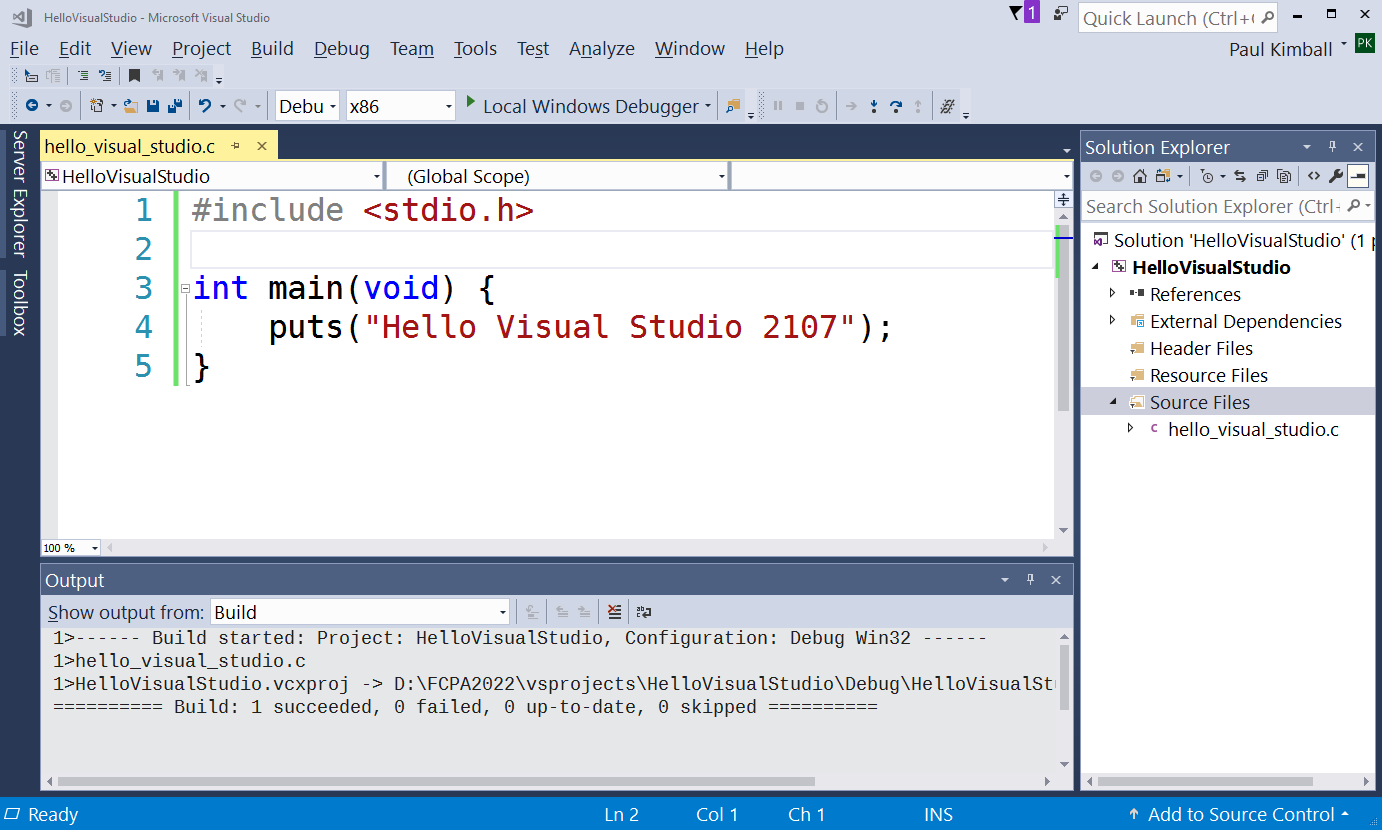
### Select Visual C++ / C++ File (.cpp) BUT...

### **Name your file** and **Change extension to .c**

## Click Add

* + Your new file appears in the Code window

# Write code



* File menu -> Save hello\_visual\_studio.c
  + OR Ctrl-S to Save

# Build and Run

## Build your solution

### **Build menu -> Build solution [Ctrl+Shift+B]**

### Check the output window for errors

## Run your solution

### **Debug menu -> Start without debugging [Ctrl+F5]**

### Solution runs in a separate window

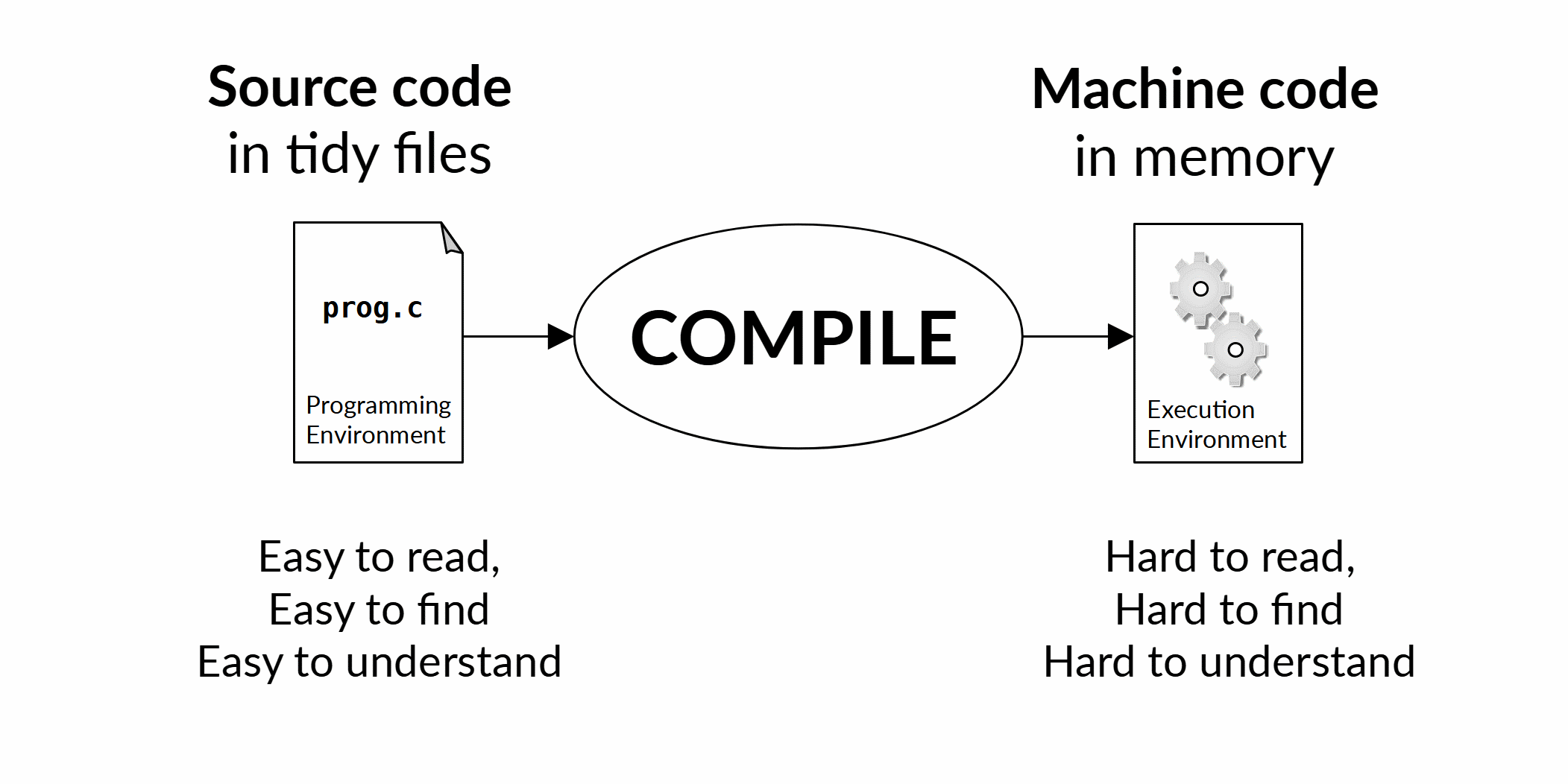
## Microsoft Visual Studio tutorial:

### <https://docs.microsoft.com/en-us/cpp/get-started/tutorial-console-cpp>

### But, it's in C++; we'll just write ordinary C code

1. Crash Course in C Programming

We Write Source Code



* Let's see how easy this is!

Structure of a C Program

* A *program* is written as one or more functions
* A *function* is a list of *statements*, surrounded by *braces*
  + Program execution starts with the function named **main**
  + Here is my program's definition of main:

int **main**(void) {

/\* Add some numbers \*/

int result = 23 + 19;

return result;

}

* Notice:
  + Any text in between /\* and \*/ is a *comment*
  + Every statement ends with a *semicolon*
  + **main** *usually* returns 0 to indicate "no problems", but what's really important is that it return a value of the proper *type* (int in this case)
  + Indentation makes it pretty, but has no other meaning

The Standard Library

* C provides a number of standard functions
  + Input/output
  + Math functions
  + Text manipulation
  + etc.
* Use the #include directive to access a library
  + This tells the compiler which functions are available

**#include <stdio.h>**

int main(void) {

/\* Add some numbers and print the sum \*/

int result = 23 + 19;

**printf("The answer is %d", result);**

return 0;

}

* Notice:
  + The **printf** function is very useful for printing formatted output!
  + *Defining* a function is different than *calling* a function
  + A *function definition* holds the code that you write

int main(void) {

/\* my code goes here, inside the braces \*/

**int result = 23 + 19;**

/\* ... \*/

**return 0;**

}

* + A *function call* runs the code in a different function

printf("Hello world"); /\* call other code \*/

Basic Syntax

* Every function, variable or type has a name, called an *identifier*
  + Some are built in, like int
  + Others, we define in our own code

int **result**; /\* result - a variable of type int \*/

int **main**(void){ /\* ... \*/ } /\* main - a function of type int \*/

* *A data type* tells the computer what kind of values we are manipulating
  + A *declaration* associates an identifier with a data type

**int** result; /\* result is an integer number \*/

**float** temperature; /\* temperature is a floating point number \*/

**int** main(void){ ... } /\* main is function returning an integer \*/

* *Delimiters* are always in pairs, and must match

{ braces around a code block }

( parentheses around a parameter list )

"Quotes around a string literal"

#include <angle\_brackets.h>

* *Punctuators* separate values, end statements, start directives

result = 12**;** /\* semicolon ends a statement \*/

printf("answer is %d"**,** c); /\* comma separates values in list \*/

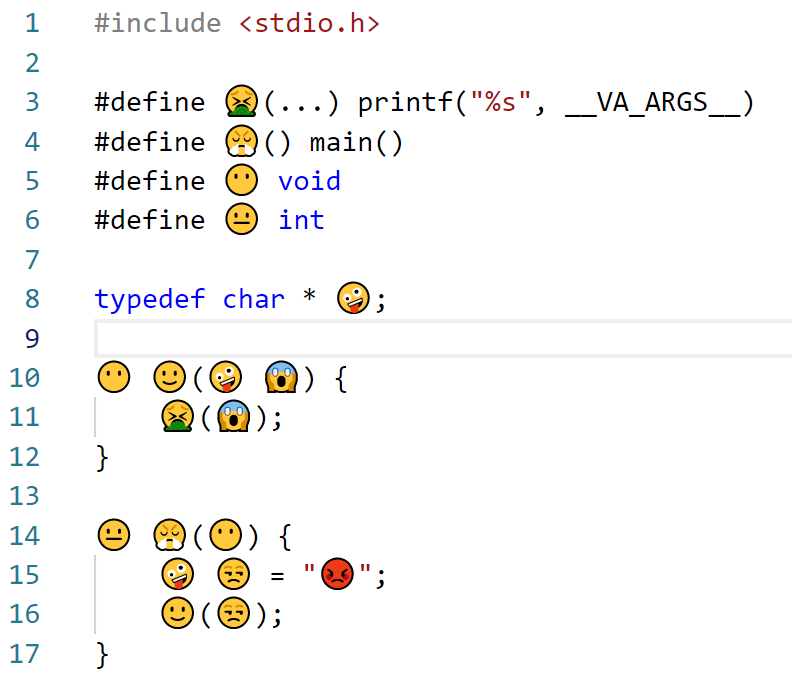
**#**include <stdio.h> /\* hashtag introduces a directive \*/

* We'll see these in more detail later

Very Very Silly C Code

## This is what the compiler thinks when it sees your identifiers

## It compiles and runs because the *syntax* is correct!



* + Good punctuation always gets high marks :-)

Numeric Values

* Integers

### Simple, fast

### Good for counting, enumerating, ordering

int answer = 42;

* Floating point

### Have a fractional part

### Wide range, decent precision, but slower than integers

### Good for engineering problems

float temperature = 42.375;

* You can do math with either type
  + Choose which type to use depending on your problem
  + You have to be careful if you mix them in an equation!

Operators and expressions

* *Operators* are symbols that combine or manipulate values

### Here are some important arithmetic operators:

result = 42; /\* assignment \*/

result = 23 **+** 19; /\* addition \*/

result = 99 **-** 57; /\* subtraction \*/

result = 84 **/** 2; /\* division \*/

result = 21 **\*** 2; /\* multiplication \*/

result = 92 **%** 50; /\* modulo or remainder /\*

* *Expressions* are formulas that may combine several operators

float degrees\_F, degrees\_C;

degrees\_C = 37.0;

**degrees\_F = (degrees\_C \* 9 / 5) + 32.0 ;**

printf("%f degrees C is %f degrees F \n", degrees\_C, degrees\_F);

Standard Console I/O

* The "stdio" library has some handy functions for printing

puts("Let me tell you the news!");

printf("The temperature is %f and the answer is still %d \n",

degrees\_F, result;

* There are functions for reading input, too:

float degrees\_F, degrees\_C;

puts("Enter a temperature in degrees C");

**scanf("%f", &degrees\_C);**

degrees\_F = (degrees\_C \* 9 / 5) + 32.0;

printf("%f degrees C is %f degrees F \n", degrees\_C, degrees\_F);

* Reading and writing is controlled by a *format string*

### The format string says what type of data we want to read or write

#### "%f" for floating point

#### "%d" for decimal integer

### When writing, it also provides extra text around the values

#### "\n" inserts a newline

### The list of variables provides the values

#### When *reading* a value, put an ampersand in front of the variable name

<<Live Demo/Follow Along>>

A crash course in C Programming

Sources of Information

* On-line reference
  + <https://cppreference.com>
* The Specification (C17)
  + ISO/IEC 9899:2018  
    Information technology — Programming languages — C

<https://www.iso.org/standard/74528.html>

* + Post-ballot working spec for download (with redlines)

<http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2310.pdf>

* + Pre-ballot draft without redlines

<https://files.lhmouse.com/standards/ISO%20C%20N2176.pdf>

* Books
  + Effective C: An Introduction to Professional C Programming

by Robert C. Seacord

ISBN-13: 978-1718501041

ISBN-10: 1718501048

* + C Programming Language, 2nd Edition

by Brian W. Kernighan (Author), Dennis M. Ritchie (Author)

ISBN-13: 978-0131103627

ISBN-10: 0131103628