FCPA 2022  
  
The C Toolchain

Student Workbook 09

January 20, 2022

Author

Paul Kimball  
Interface Associates

1. Introduction to C

The C Programming Language

* Developed by Dennis Ritchie and Brian Kernighan in the 1970s
  + Used to write most of the UNIX operating system
  + Became the de facto language for system programmers
  + Standardized by ANSI in 1989, ISO in 1990
* No extra fat; use only what you need
* Compiled
  + Human-readable source code is transformed to machine-specific instructions and data
  + Fast execution, small memory footprint
* Structured
  + Code is organized in reusable functions
  + Data can be organized in structured types
  + Can build systems of any complexity
* Imperative
  + You explicitly tell the computer what to do
  + Very few predefined semantic rules
  + Very little happens "automagically" or "conveniently"
* Low-level compared to modern languages
  + Bitwise manipulation of data, addresses, hardware - makes it a good fit for embedded programming
  + Compact compiled code
  + Standard libraries are limited to the simplest necessary functions
* Portable
  + IF conventions are followed carefully
  + More portable than assembler - you don't need to learn the entire CPU instruction/register/deployment architecture
* it's super fast, super important, and it's inside everything

The Standard, or The Specification

* Original C89 standard from ANSI
* Now maintained by ISO/IEC with input from national standards bodies, e.g., ANSI
  + Joint Technical Committee 1
  + Subcommittee 22 (Portability)
  + Working Group 14 - C

We just call them "The Working Group"

* ANSI C89/C90
  + Accepted by ISO as ISO/IEC 9899:1990
  + Amendment 1 published in 1995 is sometimes called "C95"
* ANSI C99
  + ISO/IEC 9899:1999
* C11
  + ISO/IEC 9899:2011
* C17
  + ISO/IEC 9899:2018
* C23
  + ISO/IEC 9899:2023 (anticipated)

Behavior

* The Specification defines repeatable, externally observable behavior

### The "portable" behavior of code

## Using incorrect syntax or erroneous data is always wrong

### But we all know that

### The compiler finds syntax errors

### Testing finds data errors

### So, what else could go wrong?

## The specification identifies several types of portability issue

### Undefined Behavior

### Unspecified Behavior

#### Implementation-defined behavior

#### Locale-specific behavior

## A *strictly conforming* implementation must not produce output that depends on these

### Real apps are NOT strictly conforming

Undefined Behavior (UB)

## Standard imposes NO requirements; whatever happens is nothing "portable" that you can take advantage of

### Implementation might ignore with unpredictable results

### Might behave in some interesting undocumented way characteristic of the environment (with or without a message)

### Might terminate translation or execution with a message

### EXAMPLE: Integer overflow

## Generally, don't do it

* + Unless you're intentionally writing non-portable code and your compiler vendor guarantees your results

Unspecified Behavior

## Standard provides two or more correct possibilities and does not say which is chosen in any instance

### EXAMPLE The order in which the arguments to a function are evaluated.

### ***Don't depend on it***

## Implementation-defined behavior

* + Unspecified, but implementation *must* document how the choice is made

#### EXAMPLE Propagation of high-order bit when signed integer is shifted right

### Using documented but non-portable constructs may be conforming

### ***Be careful with it***

## Locale-specific behavior

* + Unspecified, but implementation documents local conventions of nationality, culture, and language

#### EXAMPLE: Whether islower() returns true for characters other than the 26 lowercase Latin letters

### ***Be aware of it***

Sources of Information

* On-line references
  + <https://cppreference.com>
  + <http://c-faq.com>

comp.lang.c newsgroup FAQ

* + stackoverflow.com
  + google
* 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>

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

by Robert C. Seacord

ISBN-13: 978-1718501041

ISBN-10: 1718501048

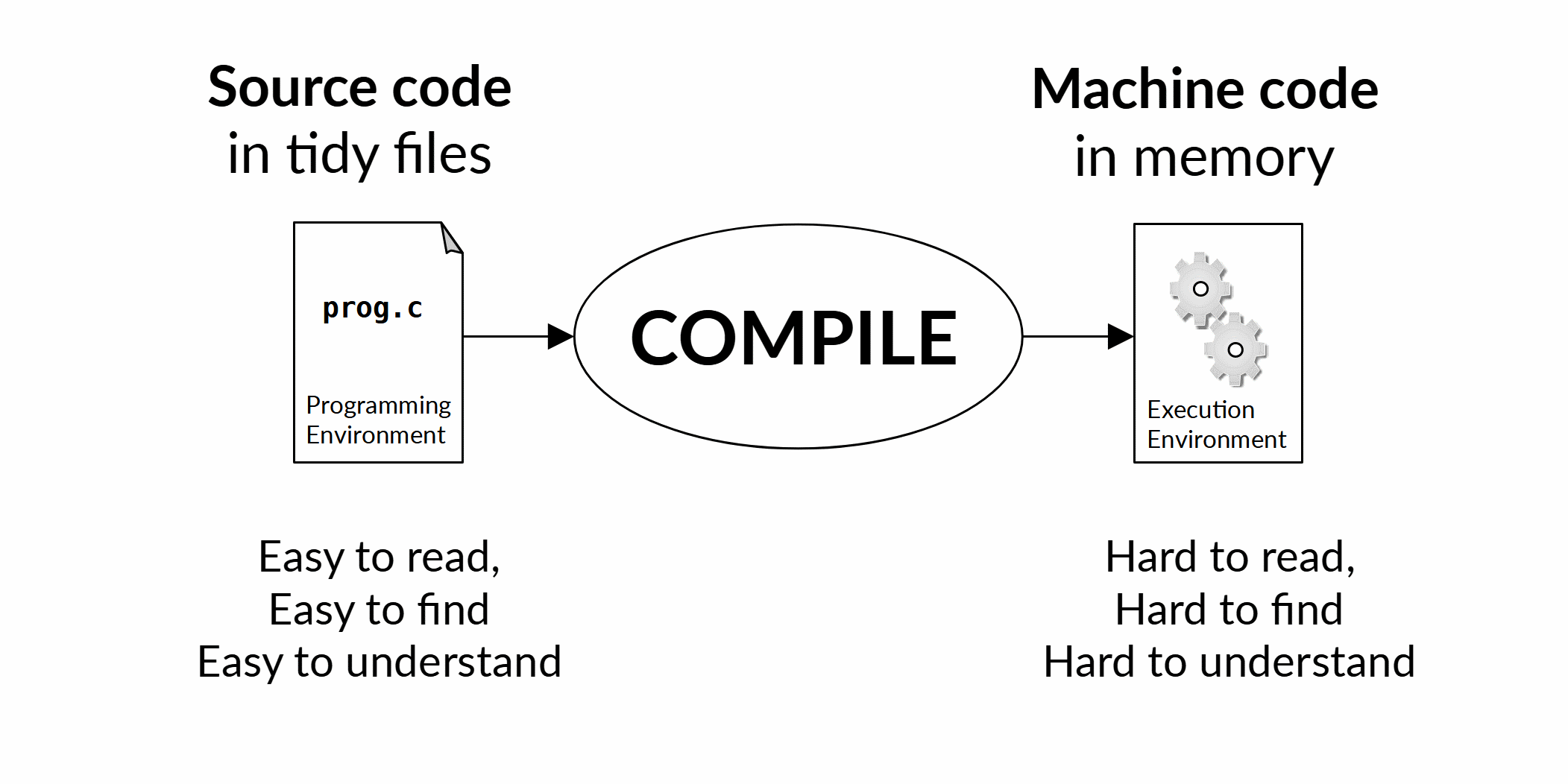
* + C Programming Language, 2nd Edition

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

ISBN-13: 978-0131103627

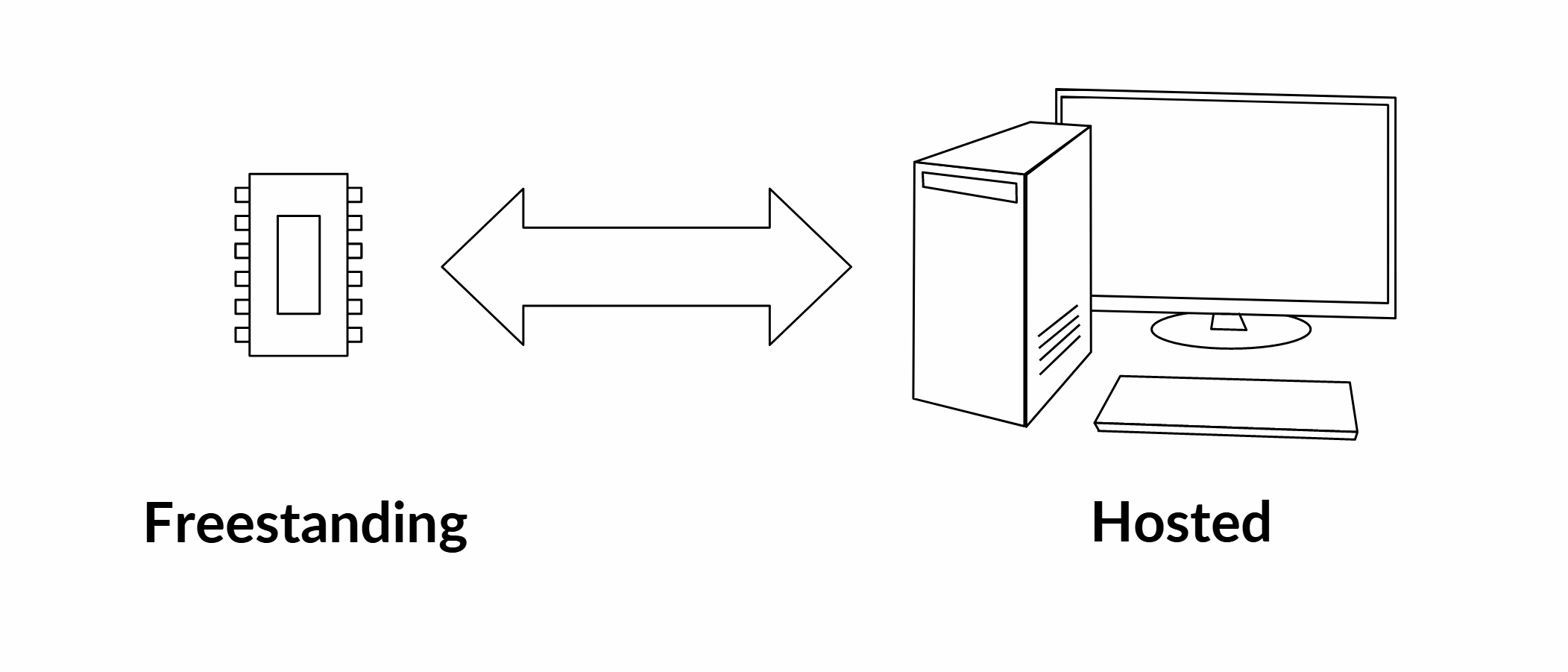
ISBN-10: 0131103628

The Simplest View



C Execution Environments

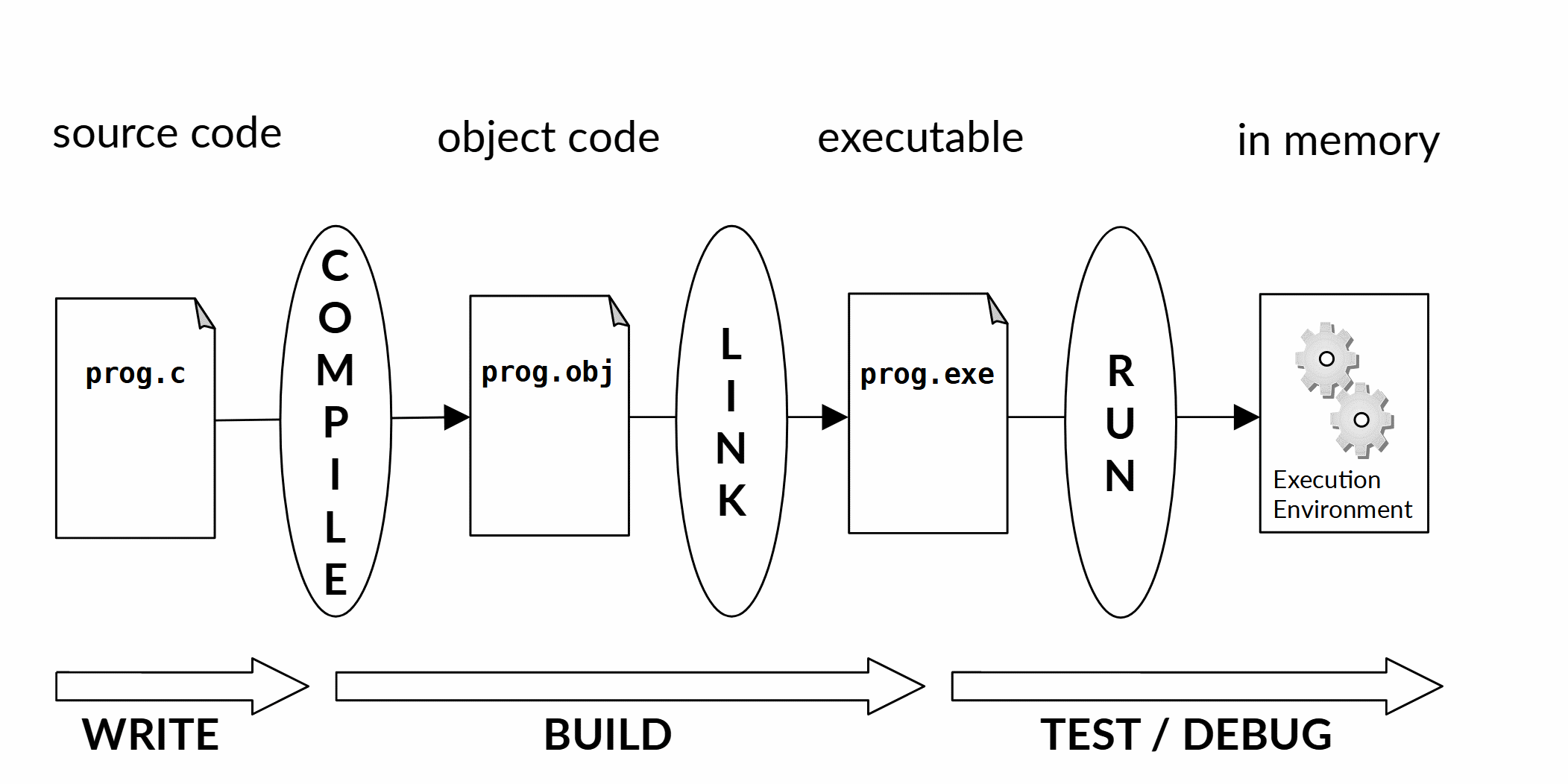
**C can be applied to either one**



|  |  |
| --- | --- |
| Hardware resources  managed by program | Virtualized resources  managed by Operating System |
| Price / Performance | Portability |
| Precise control | Convenient programming |
| Single program | Multiple processes and threads |
| Limited, fixed amount of ROM, RAM | Expandable memory |
| Real address space | Virtual address space |
| Limited libraries | Full set of standard libraries |
| Privileged processor instructions | Privileged programs |
| Limited file system, or none | OS file system(s) |
| Hardware interrupts | Software interrupts |
| H/W specific data types | Application-specific data types |

* Note that "embedded" systems are hosted *or* freestanding, depending on their complexity and needs

Life Cycle of a C Program



The C Development Environment

* You have lots of choices for tools
  + Here's what we use in this class:
* Operating system
  + Windows
* Editor / IDE
  + Visual Studio 2017 or greater
* Preprocessor / Compiler
  + Microsoft Visual C++ (MSVC)
* Linker
  + Microsoft Visual C++ (MSVC)
* Debugger
  + Visual Studio debugger
* Build system
  + MSBuild and CMake (later)
* Code Management System
  + Git / GitHub

# So Many Tools!

## Every one is a program you can run that does SOMETHING with your code

### Source code managers

### Language-sensitive editors

### Code generators

### **The C compilation toolchain**

#### **Preprocessor, Compiler, Assembler, Linker**

### Build and deployment frameworks

### Refactoring tools

### Static analysis tools and linters to check compliance with coding standards (e.g., MISRA, CERT) and best practices

### Testing frameworks, profilers and debuggers

### Reporting and logging tools

### Project management, scheduling and collaboration systems

### Continuous integration servers

### Container and Cloud deployment

## Tools come from a variety of sources

### Manufacturers of software and hardware

### Open source initiatives

### Standards organizations

## Many tools are designed with a command-line interface

### Promotes automated use from scripting languages, CI servers, and orchestration frameworks

## In carrying out a workflow, you combine and use your tools in a characteristic order

* + Robust and adaptable workflows are a good idea

# Integrated Development Environments

## IDEs tie together your tools and utilities

## Can offer a better overall coding experience:

* + Streamlined workflows
  + Better guidance and access to tool options
  + Consistent user interfaces
  + Graphical user interfaces

## Help enforce best practices defined by an organization or project

### Coding style and formatting

### Project organization

### Code management

## Integrate with external tools and systems

### Command line tools

### Git

### Testing tools

## Reality check:

* + Can easily grow large, unfocused, outdated

# IDEs for C/C++

## The C Programming Language has produced many related offshoots with similar syntax and lifecycles

### C++, C#, Objective C, Java, Kotlin, etc

## Programmers must still use command line shells and scripting languages

### For tool orchestration

### For work in minimalist coding environments

## In embedded systems development, integration with custom tools is essential

### Cross compilers

### Flash programmers

### Single-wire debugging interfaces

### On-board diagnostics

### Custom header files, libraries, support hardware

## The most popular IDEs

* + Eclipse
  + Visual Studio
  + VS Code (with extensions)

## Reference

* + https://en.wikipedia.org/wiki/Comparison\_of\_integrated\_development\_environments#C/C++

# Visual Studio

## Closed source, licensed

## Free "Community" version

* + Paid "Enterprise" and "Professional" versions available

## Extensible through:

### Visual Studio Installer

### Extensions marketplace

### NuGet

### External tools

* You've seen it in class ...

# VS Code

## Free from Microsoft

## Visual Studio's young, hip cousin

* + "Just" an editor, but a myriad of extensions

## Supported on Windows, macOS, Linux

* *Depends on extensions and external tools* to be a complete solution
  + You must add your own compilers, linkers, tools

## Fast, lightweight, highly configurable

### Configuration via JSON files

# Eclipse

## Open Source, Free

## Written in Java; very portable

### Long history with regular updates

## Customizable workbench

* + It is the sum of its plugins
  + Many plugins from many sources

## Eclipse Marketplace to distribute plug-ins

## Several bundled distributions for special purposes

* + e.g., C Development Tools (CDT)

## Sometimes a bit clumsy and old-fashioned

* + Can be spruced up by customizing fonts, themes, formatters

## A very popular base platform for semiconductor vendors

* + Vendor-specific builds may be based on older distros

# Some Tools Based on Eclipse IDE

## Texas Instruments

### Code Composer Studio (CCSTUDIO)

#### https://www.ti.com/tool/CCSTUDIO

## ST Micro

### STM32CubeIDE

#### https://www.st.com/en/development-tools/stm32cubeide.html

## Renesas

### e-squared Studio

#### [**https://www.renesas.com/us/en/software-tool/e-studio**](https://www.renesas.com/us/en/software-tool/e-studio)

## NXP

### MCUXpresso, LPCXpresso, CodeWarrior, etc.

## Arm Ltd.

### Arm DS-5 Development Studio 5

## Infineon

### AURIX Development Studio

## Arduino

### Arduino IDE

# Others

## NetBeans

### Open source

### Originally from Sun Microsystems ( now Oracle )

### Runs on Windows, macOS, Linux, Solaris

### Some vendor tools based on NetBeans

#### Microchip

#### MPLAB X

## C-Lion

* + JetBrains

## Code::Blocks

* + Open source

# Compiler Choices

## C is a standard language

* + There are many compilers that implement the standard

### You will use many different compilers in your work

## Your choice of compiler(s) depends on

### Development system

### Target system

## Development System

* + Runs tools to edit, compile, link, package, and manage your source code
  + Usually a graphics workstation with desktop OS

#### Windows

#### UNIX/GNU/Linux

#### macOS

### Has its own environment (network, servers, commands, scripts, locale, file system, etc.)

## Target System

### Executes the code that you wrote

### Often the same as the development system while writing code

### Often VERY different during test and production

#### Embedded MCU - bare-metal

#### Embedded MCU with Real-Time OS (FreeRTOS, QNX, etc.)

#### Embedded MCU/MPU with Linux/\*NIX OS

#### Mobile Platform (e.g. Android, iOS, Chrome OS, many others)

#### Mainframe with custom OS

#### Container-based deployment (Azure, AWS, Docker, etc.)

#### Software Emulator (e.g., qemu)

### Has its own (possibly restricted) environment

## If development system != target system, code is "cross-compiled"

# The Big Ones

## GNU Compiler Collection (GCC)

* + The oldest and most established

## MinGW

* + A port of GCC to run on the Windows OS and produce Windows executables

## Clang / LLVM

* + The default compiler on Apple
  + Supports many of the same command-line options as GCC

## MSVC - Microsoft

* + Packaged with Visual Studio
  + Runs on Windows, can target windows or Linux (WSL)

## Intel C Compiler

## Others

* + Specialized for certain MCU/MPUs

# Compiler Messages

## The Specification says that the compiler must output a message on:

## Incorrect syntax

### A syntax problem may be severe enough that the compiler cannot continue parsing

### Always look at the first message and deal with it first; everything else is supect

## Violation of a constraint

### These produce warnings but don't stop compilation - ignore them at your peril

## Other messages are optional

* + Compilers are prized for the quality of their messages

# Compiler Switches

## Source code language and version

## Execution environment and CPU architecture

## Optimization settings

## Source file locations

## Header file locations

## Output directories and filenames

# A Deeper Dive

# The C/C++ Preprocessor

## The preprocessor prepares source code for the compiler

### Merges multiple source files into a single stream

### Performs lexical substitutions, called "macros"

### Conditionally defines types, variables and functions

### Produces a single (possibly very large) "compilation unit"

## Is controlled by directives that start with the hash character

### #include

### #define

### #if / #else / #elif / #endif

### #ifdef, #ifndef

### #pragma

### #error

# C Typical Preprocessor Uses

## Including header files

* + Header file guards

## Defining compile-time constants

## Conditional inclusion of libraries

## Conditional compilation

## Macros

* + Object-like macros
  + Function-like macros

# Preprocessor Directives

## Directive starts with '#' followed by directive name

### Space between # and name is OK

### Directive names are fixed; cannot be defined by programmer

## Each directive appears on its own line

### May be indented for clarity

## Directive and its arguments are terminated with a newline

### A backslash '\' at the end of a line escapes the newline and continues the directive on the next line

### Be very careful ! No spaces after the backslash (!)

# #include

## Incorporates the contents of another file as though you had typed it right there

* + By convention, included files are called "header files"

## A header file declares data types, functions, macros and static content provided by a library

### Use ".h" as a file extension

### Technically, you can include non C/C++ content

Please don't; the merged content must be something the compiler can understand

## There are two common way to include a header in C

### Both are implementation-defined, but here's what *usually* happens...

### Filename in quotes

File is relative to the include path, starting with the current directory

You can add additional directories on the compiler command line

Use this syntax for including your own project headers

* + Filename in angle brackets

File is in a system-administered directory, which is known to the compiler - e.g., /usr/lib on Linux

Use this syntax for including standard library headers

#include "MovieTicket.h"

#include <theater.h>

## Header files can include other headers, recursively

C Standard" Libraries

|  |  |  |  |
| --- | --- | --- | --- |
| **Header** | **Purpose of library** | **Since** | **Required in Freestanding** |
| **<assert.h>** | assert and static\_assert (C11) macros | C89 |  |
| **<complex.h>** | Complex and imaginary numbers |  |  |
| **<ctype.h>** | Character handling | C89 |  |
| **<errno.h>** | Errors | C89 |  |
| **<fenv.h>** | Floating-point environment |  |  |
| **<float.h>** | Characteristics of floating types | C89 | yes |
| **<inttypes.h>** | Format conversion of integer types |  |  |
| **<iso646.h>** | Alternative spellings of logical and bitwise ops |  | yes |
| **<limits.h>** | Sizes and bounds of integer types | C89 | yes |
| **<locale.h>** | Localization | C89 |  |
| **<math.h>** | Mathematics | C89 |  |
| **<setjmp.h>** | Nonlocal jumps | C89 |  |
| **<signal.h>** | Signal handling | C89 |  |
| **<stdalign.h>** | Alignment |  | yes |
| **<stdarg.h>** | Variable-length argument lists | C89 | yes |
| **<stdatomic.h>** | Atomics |  |  |
| **<stdbool.h>** | Boolean type and values |  | yes |
| **<stddef.h>** | Common definitions | C89 | yes |
| **<stdint.h>** | Integer types |  | yes |
| **<stdio.h>** | File and console input/output | C89 |  |
| **<stdlib.h>** | General utilities | C89 |  |
| **<stdnoreturn.h>** | noreturn |  | yes |
| **<string.h>** | NULL-terminated strings | C89 |  |
| **<tgmath.h>** | Type-generic mathematics |  |  |
| **<threads.h>** | Threads |  |  |
| **<time.h>** | Date and Time | C89 |  |
| **<uchar.h>** | Unicode utilities |  |  |
| **<wchar.h>** | Multibyte/Wide character utilities |  |  |
| **<wctype.h>** | Wide character classification and mapping |  |  |

* May be included in any order
* May be included more than once
* Described in Chapter 7 and Annex B of Specification