FCPA 2022  
  
Filling in Your Collection

Student Workbook 13

January 26, 2022

Author

Paul Kimball  
Interface Associates

1. Sorting Things Out

What's coming up

* Thursday - MISRA and Clean Code
* Friday - Embedded systems
* Monday - office hours?

Stages of Collecting

* Accumulate
  + Take any opportunity to build your collection
  + Get as much as possible
* Appreciate
  + Learn about the details
  + Recognize the differences
  + Do a little sorting
  + Enjoy!
* Curate
  + Throw out the mundane
  + Catalog the useful
  + Preserve the important
  + Exalt the extraordinary

Little Things (but not Simple)

* The C language
* Memory
* The toolchain - preprocessor, compiler, linker
* Visual Studio
* The Command Line
* Version Control - Git/GitHub
* Hardware

Views of a Language

* The Unfocused view is Smooth
  + A mess of words and symbols
* The microscopic view is Gritty more than Chunky
  + Each word and symbol has a specific meaning and a purpose
  + But there are so many words and operators!
* The close-up view is Chunky
  + Statements in functions
  + Data in variables
  + Structured data types - structs and arrays
* The far-away view is even more Chunky
  + Header file and C source files
  + Libraries
  + Source code in files, projects, repositories

Four Things about C

* Variables are Data
  + A named chunk of memory
  + Holds a value (an "object")
* Functions are Instructions
  + A named block of statements
  + Performs a repetitive chore
* Data Types are the Rules of the Game
  + Describes what data represents and what you can do with it legally
* We write source code
  + It's what we understand
  + It's what we care about

Views of Memory

## The Unfocused view is deceptively Smooth

* + A sea of bytes
  + Pointers could point anywhere?
  + This is the "Memory" view in the debugger

## The microscopic view is Gritty

* + Just bytes with small values and alignment issues

## The Close-up view is Chunky

### Chunks of memory for interesting VALUES

### char, int, float, double, pointer, array, struct + members, union

### Objects have boundaries and sizes

* + Pointers point to specific things

## The Far-away view is Chunky

* + H/W and O/S define regions of memory for special purposes
  + Text/code, data, stack, heap

Memory Maps

* Maps emphasize the important features of a space
  + Size
  + Boundaries
  + Relationships
  + Routes for travel and trade
* Memory maps are drawn at different scales, depending on what you want to look at
  + Pick the one that suits your problem

Views of your Tools

* The Unfocused view is Smooth
  + Products and projects clamor for attention
  + Sales reps try to show you differences that may not be that important
  + Everything looks cool
* The close-up view is Chunky/Smooth
  + Thousands of tools with small responsibilities
  + "The" editor, "The" compiler, command line, linker, git, file system, etc.
* The far-off view is Chunky
  + There are recognizable categories of tools - Compilers, editors,
  + Different problem domains use specific toolsets

Filling in Your Collection

* Storage classes and copying values
* Bigger, better structured types
  + Combining the array, struct, enum and union,
* Memory allocation
  + malloc and free
  + Pointers to structs
  + Reusing memory
* Some special operators
  + Bitwise
  + Assignment
  + Increment/decrement

Storage class

* Memory is allocated for a variable when it is declared
  + Storage class modifiers affects the lifetime, placement, and initialization of its value
* auto
  + **The default for all local variables**
  + Lifetime: from point of declaration until end of scope
  + Placement: the stack
  + Initialization: Must be explicitly initialized in program, else garbage
* \_Thread\_local
  + Like auto, but associated with a thread of control
* static
  + **The default for variables declared outside of a function (file scope)**
  + **May be used on a local variable to retain its value while out of scope**
  + Lifetime: entire life of running program
  + Placement: initialized data segment (?)
  + Initialization: May be initialized, but only with a constant expression
* extern
  + refers to a static identifier that is initialized in a different compilation unit
  + Will be resolved by the linker, not the compiler
* register
  + A hint to the compiler that value could be in a CPU register
  + Implies that variable will be accessed frequently and must be fast
  + Can't have & operator applied, since not in RAM
  + Size must allow it to fit in a register

Copying Memory

* These workhorses copy chunks of memory from one location to another.
  + The destination storage must be allocated and big enough !
* The assignment operator ( = )
  + Works for scalar types
  + Works for enums
  + Works for pointers (copies the address only)
  + Works for structs and unions

As long as they DO NOT have pointer members

* + **Does NOT work for arrays or NULL-terminated strings**
* memcpy, memmove
  + General-purpose memory copy functions
  + Copies as though value are arrays of unsigned char
  + **Works on anything, as long as it's referenced through an address**
* strcpy, strncpy, strcat, strncat
  + The NULL-terminated string library functions
  + Works on NULL-terminated strings
  + Multibyte and wide-character variants too

Allocating Memory

* These functions allocate memory dynamically
* malloc
  + allocates memory
* calloc
  + allocates and zeroes memory
* realloc
  + expands (and maybe moves) a previously allocated memory block
* free
  + deallocates previously allocated memory
* aligned\_alloc (C11)
  + allocates aligned memory

Data Types

* How to declare a variable
  + Think: Why do I need a new variable?
  + To make code easier to read
  + Give a meaningful name to an important value that will appear multiple times in calculations
  + To hold the return value from a function for later use
* How to write a function
  + Think: Why do I need a new function?
  + Is there a library function that I could use?
  + To make code easier to read
  + Give a meaningful name to a repetitive task that will be done over and over
* How to define a data type
  + Think: Why do I need a new type?
  + Is there a built-in type that does what I want?
  + Is there a type defined in a library that does what I want?
  + To make code easier to read
  + Give a more meaningful name to an existing type
  + Define a type that combines a group of values