



Use Visual Studio with Visual
Micro, hardware debug with
AtmelStudio (MicrochipStudio).

DEFENSIVE C++ ARDUINO PROGRAMMING

Self-published via Amazon by the author - Michèle Delsol

Defensive C++ Arduino Programming by Michèle Delsol

Proper names, trademarks, and designations used by the author are capitalized to distinguish them from ordinary text. They are the property of their respective owners. The author and publisher of this book have no intent at establishing any relationship whatsoever with the owners of these names, trademarks, and designations.

The author and publisher have exercised due diligence as to the exactitude of the book's content and issue no explicit or implied warranty of any kind as to the suitability of content presented for any purposes whatsoever and assume no responsibility for errors or omissions. The author and publisher assume no liability for incidental or consequential damages resulting from the use of information, code snippets, and programs presented in this book.

First published via Amazon August 2023.

Delsol, Michèle

Defensive C++ Arduino Programming / Michèle Delsol, first edition

Copyright © 2023 by Michèle Delsol

All rights reserved. Printed via Amazon "Print-on-demand" in the United States and in other countries where Amazon distributes this book. This book is protected by United States and international copyright laws. No parts of this book may be copied in any form whatsoever; permission must be obtained to reproduce parts and the entirety of this book by any means whatsoever: electronic, photocopying, mechanical, or other.

Indexing was undertaken via JavaScript scripts applied on manually created tags. The indexing system was created by the author.

The warthog illustration on the front cover is an original pencil and China ink drawing by the author.

Paperback edition : ISBN 978-2-9585628-2-3

First edition published via Amazon August 2023 in paperback, hard cover, and electronic (Kindle) formats.

Table of contents

List of tables and figures	xi
Acknowledgements	xiii
Preface	xv
Introduction	1
Chapter 1 Common sense and new tools	5
Chapter 2 Why choose Arduino	7
Chapter 3 The build toolchain	9
3.1 Intelligent C++ editor	10
3.2 Make	10
3.3 Preprocessor	11
3.4 Compiler	11
3.5 Linker	11
3.6 Uploader	11
3.7 Bootloader	11
3.8 Serial terminal	12
Chapter 4 Hardware setup	13
4.1 Breadboards	14
4.2 Prototype board	15
4.3 Final PCB	15
Chapter 5 Interoperability	17
5.1 Directory structure	17
5.2 IDE specific entry files	18
5.3 Render setup/loop independent of code changes	19
5.4 Single-level directory structure interoperability scenario	20
5.5 Two-level directory structure interoperability scenario	21
Chapter 6 Which IDE to work with?	23
6.1 Dedicated IDEs	23
6.2 Plugins	23
Chapter 7 Arduino IDE	25
7.1 Arduino distribution	26
7.2 Arduino Basic features	27
7.3 Arduino IDE V2 specifics	28
7.4 Arduino caveats (legacy version 1.8.19)	30
Chapter 8 AtmelStudio	33

8.1 Why adopt it for your Arduino projects	33
8.2 Missing features	35
8.3 Managing directories	35
8.4 Importing an Arduino project (AtmelStudio)	36
8.5 Upload code into the microcontroller (create external tool)	39
8.6 Interface with Arduino applications	43
8.7 Editor features	44
8.7.1 Workspace	46
8.7.2 Intelligent window scrollbars	47
8.7.3 Quick access to functions	48
8.7.4 Go to implementation	48
8.7.5 Inline function related tasks	48
8.7.6 Color coding	49
8.7.7 Inline syntax checking	50
8.7.8 Find/Replace	50
8.7.9 Name completion	51
8.7.10 Code collapsing and indentation	52
8.7.11 Collapse /*...*/ group	54
8.7.12 Group comment/uncomment	54
8.7.13 Refactoring	54
8.7.14 Compiler error reporting	55
8.7.15 Code navigation	55
8.7.16 Bookmarks	56
8.7.17 Spell-checker	56
8.7.18 Go to line number	56
8.8 AtmelStudio hardware-based debugging	56
8.9 AtmelStudio caveats	57
8.9.1 Workspace font size	59
8.9.2 Application code size (AtmelStudio)	60
8.9.3 Failure to upload code	60
8.9.4 Code changes	61
8.9.5 Second set of header files import	62
8.9.6 Quick access to functions pulldown selection box failure	62
8.9.7 Improper indenting	63
8.9.8 Failed code formatting	63
8.9.9 Segmentation faults	64
8.9.10 Errors not listed in the error list	65
8.9.11 Make utility errors not listed	66
8.9.12 Inexistent file in project not listed in error messages window	67
8.9.13 Search and replace	67
8.9.14 AtmelStudio code formatting and code collapsing bug	67
8.9.15 Auto format if...else problem	68

8.9.16 Refusal to uncollapse a section of code	69
8.9.17 Unable to generate code for new microcontroller	69
8.10 Keyboard shortcuts	70
8.11 Documentation	71
Chapter 9 Visual Studio	73
9.1 Visual Studio features	73
9.2 Visual Studio download	74
9.3 Visual Studio 2019 vs. Visual Studio 2022	74
9.4 Visual Studio for Arduino developers	75
9.5 Visual Studio (Arduino project template)	75
9.6 Visual Studio (Visual Micro)	76
9.7 Visual Studio (debugging)	76
Chapter 10 Visual Micro	77
10.1 Visual Micro for AtmelStudio (MicrochipStudio)	77
10.2 Visual Micro for Visual Studio	78
Chapter 11 VS Code	79
11.1 VS Code features	80
11.2 VS Code installation	80
11.3 VS Code navigation	81
11.4 VS Code configuration	82
11.5 VS Code - folder vs. project	83
11.6 VS Code extensions	84
11.6.1 Arduino CLI	85
11.6.2 VS Code Arduino extension	85
11.6.3 VS Code C++ extension	87
11.6.4 VS Code serial terminal extension	87
11.6.5 VS Code PlatformIO extension	87
11.6.6 VS Code Awk extensions	88
11.6.7 VS Code Perl extensions	88
11.6.8 VS Code regular expressions (regex) extensions	88
11.7 .json files	88
11.8 VS Code caveats	89
Chapter 12 PlatformIO	91
12.1 PlatformIO Installation	93
12.2 PlatformIO features	94
12.3 PlatformIO managing projects	95
12.4 PlatformIO creating projects	95
12.5 PlatformIO project configuration (platformio.ini)	98
12.6 PlatformIO interoperability	99
12.7 PlatformIO hardware-based debugging	100
12.8 PlatformIO help and documentation	100

12.9 PlatformIO gotchas and caveats	101
Chapter 13 Other IDEs (Code::Blocks and MPLAB)	103
13.1 Code::Blocks	103
13.2 MPLAB	104
Chapter 14 Debugging	105
14.1 Avoiding bugs	106
14.2 Types of problems	106
14.2.1 Glitches	106
14.2.2 Code thrashing	107
14.2.3 Misdoings	109
14.3 Print-based debugging	110
14.4 Hardware-based debugging	110
14.4.1 ATmega328P-Xmini	112
14.4.2 ATmega2560RFR2 Xplained Pro	113
14.4.3 Atmel ICE	113
14.5 Serial debugging	116
Chapter 15 Good programming practices	119
15.1 Be consistent	122
15.2 Naming conventions	122
15.3 Code formatting	123
15.3.1 Readability	123
15.3.2 Indentation	123
15.3.3 Collapsing	124
15.4 Think	125
15.5 Document code	125
15.6 Plan your work offline	126
15.7 Never assume anything	127
15.8 Error handling	127
15.9 Project organization	128
15.10 Monitor memory use	129
15.11 Careful with lenient compiler type checking	129
15.12 Abide by the KISS principle	130
15.13 Task wrap-up phase	130
15.14 Mental condition	132
15.15 Use C++ macros	133
15.16 Define constants only once	133
15.17 Use enum lists	134
15.18 Parameter default initialization	135
15.19 Initialize using curly braces	135
15.20 Comment closing curly braces and #endif	136
15.21 Use the auto-indent feature of the editor	136

15.22	Always start with code skeletons	136
15.23	Exploit C++ features sparingly	136
15.24	Do your homework	137
Chapter 16	Frameworks	139
16.1	Organizational Frameworks	141
16.1.1	Project Files Framework	142
16.1.2	Program Documentation Framework	144
16.1.3	Function Creation Framework	146
16.2	Data Handling Frameworks	147
16.2.1	Class Data Framework	148
16.2.2	Data Packets Framework	150
16.2.3	Format Driven float to byte Conversion Framework	152
16.2.4	DataGroup Framework	154
16.2.5	Bitfield Storage Framework	155
16.2.6	Event Storage Framework	160
16.2.7	Linked List Framework	161
16.3	Specialized Frameworks	162
16.3.1	Algorithm Test Framework	163
16.3.2	Class and Function Names Referencing Framework	164
16.3.3	Class and Function Names Referencing Framework instrumentation	166
16.3.4	Memory Management Framework	167
16.3.5	Pseudo Exception Handling Framework	172
16.3.6	Error Reporting Framework	173
16.3.7	Operator overloading	174
16.3.8	Print-based Debugging Framework	176
Chapter 17	Should know tools	179
17.1	Regular expressions (regex)	180
17.1.1	Regex terminology	182
17.1.2	Regex primer	183
17.1.3	Metacharacters and literal characters	184
17.1.4	Regex tidbits	185
17.1.5	Regex example - search for enums	186
17.1.6	Regex look ahead/behind	186
17.1.7	Regex groupings	187
17.1.8	Regex greediness	188
17.2	Awk vs. Perl	190
17.3	Awk	192
17.3.1	Awk terminology	194
17.3.2	Awk program structure	194
17.3.3	Awk conditions	197
17.3.4	Awk action-blocks	197

17.3.5	Awk rules	198
17.3.6	Awk BEGIN and END	202
17.3.7	Awk functions and variables	202
17.3.8	Awk vs. C	202
17.3.9	Awk gotchas	203
17.4	Perl	205
17.4.1	Perl terminology	205
17.4.2	Perl primer	206
17.4.3	Perl program structure	208
17.4.4	Perl quick overview	208
17.4.5	Simulate Awk using Perl	210
17.4.6	Perl vs. C	211
17.4.7	Perl caveats	212
Chapter 18	From back of the envelope to final product	213
18.1	Tinkercad	214
18.2	Eagle	214
18.3	Fusion 360	215
18.4	3D printing	216
18.5	Fused deposition modeling FDM	217
Chapter 19	Appendix	219
Chapter 20	RAM requirements	221
20.1	Code size (IDE comparisons)	221
20.2	Code size (build options)	222
Chapter 21	DOS box (appendix)	225
21.1	DOS box - What is it?	225
21.2	DOS box launch	226
21.3	DOS commands	228
21.4	DOS batch files	228
21.5	DOS box - other info	230
Chapter 22	Frameworks (appendix)	231
22.1	Organizational Frameworks	231
22.1.1	Project Files Framework	231
22.1.2	Program Documentation Framework	232
22.1.3	Function Creation Framework	236
22.1.4	Class and Function Names Referencing Framework instrumentation	239
22.2	Data Handling Frameworks	241
22.2.1	Class Data Framework	241
22.2.2	Data Packets Framework	244
22.2.3	Format Driven float to byte Conversion Framework	245
22.2.4	Bitfield Storage Framework	248

22.2.5 DataGroup Framework	252
22.2.6 Event Storage Framework	255
22.2.7 Linked List Framework (appendix)	257
22.3 Specialized Frameworks	258
22.3.1 Algorithm Test Framework	259
22.3.2 Class and Function Names Referencing Framework	261
22.3.3 Memory Management Framework	264
22.3.4 Pseudo Exception Handling Framework	267
22.3.5 Error Reporting Framework	269
22.3.6 Print-based Debugging Framework	270
Chapter 23 Misdoings (appendix)	277
23.1 = instead of == in if/while	277
23.2 == instead of = in assignment	278
23.3 switch statement empty or missing default case	279
23.4 F() macro in Serial.print missing	279
23.5 Work to do	279
Chapter 24 Memory structure (appendix)	281
24.1 Memory use macros	281
24.2 Determining memory usage	282
24.3 Highly fragmented heap	284
24.4 How do free and delete know how much memory to deallocate	284
Chapter 25 Awk (appendix)	285
25.1 Download Awk	285
25.2 Invoking Awk from a DOS box	285
25.3 Awk short example	286
25.4 Awk some features illustrated	288
25.4.1 Awk scope	288
25.4.2 Awk arrays	289
25.4.3 Awk reference	292
25.4.4 Awk command line	293
25.4.5 Awk comments	294
25.4.6 Awk rules	294
25.4.7 Awk condition only and action-block only rules	295
25.4.8 Awk interrupt processing lines	295
25.4.9 Awk BEGIN and END	295
25.4.10 Awk functions and variables	295
25.4.11 Awk built-in variables	296
25.4.12 Awk program flow control	297
25.4.13 Awk operators	298
25.4.14 Awk strings	298
25.4.15 Awk print	298

25.4.16 Awk mathematical functions	300
Chapter 26 Perl (appendix)	301
26.1 Download Perl	302
26.2 Invoking Perl from DOS box	302
26.3 Perl simulate Awk	303
26.4 Perl file handling	304
26.5 Perl subroutines (functions)	305
26.6 Perl strings	305
26.7 Perl string interpolation - single or double quotes	305
26.8 Perl lists	306
26.9 Perl program flow control	306
26.10 Perl arrays	307
26.11 Perl multidimensional arrays	310
26.12 Perl pass by reference	311
26.13 Perl built-in variables	311
26.14 Perl string functions	312
26.15 Perl importing packages	313
26.15.1 Perl variable types	313
26.15.2 Perl scope resolution operator :: and my	313
26.16 Perl text match operator	313
Chapter 27 Bibliography	315
27.1 Bibliography - C/C++ programming	315
27.2 Bibliography - PROGMEM framework	316
27.3 Bibliography - Software Engineering	317
27.4 Bibliography - Regular Expressions (regex)	317
27.5 Bibliography - Awk	317
27.6 Bibliography - Perl	317
27.7 Bibliography - Arduino	318
27.8 Bibliography - AtmelStudio (now MicrochipStudio)	318
27.9 Bibliography - Visual Micro	318
27.10 Bibliography - PlatformIO	319
27.11 Bibliography - Espruino and JavaScript	319
27.12 Bibliography - Hardware-based debugging	319
27.13 Bibliography - Programming psychology	319
A note on the book's source code	321
About the author	323
Beehive weighing system	325
27.14 Beehive weighing system architecture	325
Index table	327

List of tables and figures

Table 16.1 - Job class - Table of bitfield based variables.	156
Table 16.2 - Memory used as the application runs.	168
Table 20.1 - RAM memory and flash memory required by the three IDEs to run the <i>MemMgt</i> application with the <i>-flto</i> option.	221
Figure 22.1 - Test Arduino-based beehive weighing system execution paths with Algorithm Test Framework.	259
Table 25.1 - Weekly grocery sales	287

Acknowledgements

This book, *Defensive C++ Arduino Programming* and its companion, *Pragmatic C++ Arduino Programming*, are the result of chance encounters which led me to beekeeping and to create *Arduino-based* gadgets. Put the two together and, aha! why not create an *Arduino-based beehive weighing system*. That is how it all got started. And one thing leading to another, I got into writing two books which address the needs of C++ savvy DIY Arduino makers.

I must thank the many who contributed to my getting started on writing these books and continuing it to its ultimate conclusion.

First in line is Daniel T. I am particularly grateful to him since he made me discover Arduino. His electronics advice, despite his being a practicing pediatrics surgeon, hence electronics not being his field at all, contributed immensely towards getting me started with Arduino.

And I thank Christine C., my psychoanalyst, whom I see regularly to express my little travails. She has been an unconditional supporter of my endeavor.

And then there is Charlie H., a practicing physician and fellow airplane builder (RV8). He introduced me to beekeeping - a few visits to his bee yard and I was hooked.

There is of course Xavier M., who purchased my company years back and who has since become a friend. His continued support has contributed to my persevering in this book's endeavor.

My neighbors Martine and Olivier B. continuously supported my endeavors. My special thanks to them. I must say that as the project advanced from milestone to milestone, we celebrated by opening one or two bottles of Champagne - by now, a few cases have gone down our respective esophagus.

As luck would have it, Allison (Olivier's daughter) is an InDesign professional consultant. I am thankful to her as she accepted to create the print and digital ready document. She was patient as she suffered through my unorthodox approach which consisted in creating tags in Word which would be used by an InDesign JavaScript script to produce the finished book (layout, cross references, indices, table of contents, etc.).

I must also thank daughter #1 Giselle, also an author, for her continued support.

Finally, but not least, I owe daughter #2 Pascale special thanks as she patiently proofread both manuscripts, a total of four hundred plus A4 pages of tight letter size text. Since I am an engineer, my thought process, hence sentence construction, tends to be a bit linear (somewhat tedious to read). She managed to smooth things out and put some pep into many of my phrases. And I thank all those others who manifested their support as they patiently heard me out as I described my project.

Preface

How did I come to write *Pragmatic C++ Arduino Programming* and *Defensive C++ Arduino Programming*? The journey started due to my being an amateur beekeeper. In order to check the health of my hives, I weigh them regularly, weekly, or daily, depending on the season. I use an ordinary luggage scale to lift the rear of the hive. I naturally asked myself "How about hacking an electronic *Arduino-based beehive weighing system* to replace my trips to the beehives. Monitoring the weight provides an indication on the beehive's health and how much honey is being produced. It also helps determine whether the beehive has swarmed. Swarming means that the old queen and half the bees leave the beehive; a 6 lbs. loss in 10 minutes.

The back of the envelope design of the *Arduino beehive weighing system* seemed simple. Building such a system was the type of challenge I would enjoy. Little did I know that the electronics would prove easy, while the software to control the electronics would be a major endeavor which would take two+ years to complete! As I progressed on my project, I was forced into transitioning from a hobbyist mindset to a professional programmer mindset. These two books are the concepts, procedures, and tools which enabled me to complete the *beehive weighing system software* – its size is now 35 files/15000 lines of code.

I quickly defined the general characteristics of the project: ordinary bathroom scale strain gauges amplified by opamps, star shaped network radio communications, clocks to synchronize the modules, a GSM board to send data to my smartphone. The deciding factors on choosing components were cost, availability, suitability, and ease of development. The inexpensive Arduino board, its free easy to use development environment, and C++ being the best language for producing compact, fast, executables were the deciding factors. What's more, I already had experience programming C++.

I then wrote code to try out the subsystems one at a time. Throughout this early dabbling, I was impressed by the ease provided by the Arduino IDE. After experimenting with the examples, I started my *beehive weighing system* with one *.ino* file, then, slowly but surely, the application grew. I added *.h/cpp* files, created classes, and defined macros.

I started the project by validating my choice of electronic components and experimenting with each subsystem. And this is where the fun began. Getting subsystems to work one at a time was easy. Getting them to work together got to be particularly time consuming. I confess that I had unknowingly embarked on a major project with a hobbyist mindset. I soon spent much more time debugging than coding. It got to be a painful experience.

As I progressed and the application was getting more complex, I was at the mercy of poor planning, and I no longer controlled development. I could not continue with my undisciplined hobbyist approach hacking out an *Arduino-based* gadget by coding directly, straight from mind to code. I had to modify my ways because the application was getting unwieldy. It had reached a size and complexity which required that I adopt formal design and implementation techniques. If I were to bring my project to completion, I had to step back, rethink my approach, and get organized. Furthermore, I had to update my C++ knowhow since I had become a little rusty and was making way too many mistakes.

As luck would have it, I had already developed applications at a professional level and had

been project manager of a large industrial plant project, but that was years ago. I thought back on that work and reviewed literature on the subject. My past experience incited me, slowly but surely, to adopt good programming practices and to structure development. This proved to be difficult and challenging. It demanded hard work, but the fun returned.

The foremost change in my approach was to formalize the parts and pieces of the project, define them as individual modules, and lay them down on paper. Understanding the system's structure from both the electronic and software standpoint led to partitioning the project into modules. There were subsystems at the electronic level (GSM, clock, radio, EEPROM) and there were subsystems at the software level (data storage, data validation, data conversion, data transmission, etc.). Thanks to C++'s object-oriented features, I started the application's design *top-down*. The details were to be filled-in *bottom-up* later on.

Informal solutions to manage certain tasks morphed into formal frameworks. Getting these done required drawing flow charts, creating algorithms, defining functionalities, components, and procedures. Working offline on paper proved to be a critical first step. This planning turned out to be vital to the application's robustness, as well as to my own productivity. After a while, the application began to take shape as pieces fell into place.

Development became a two-pronged process: *design and implement the big picture* (i.e., *top-down* via classes and wrapper functions) and *design and implement low level helper functions and functional frameworks - bottom up*. The process was iterative: go to the drawing board; work on a module; change things; go to the computer; implement new code; test and examine results; then go back to the drawing board. I did this until I got it right. Documenting each step helped me learn and clarify concepts. Many hours later, I had laid the groundwork and developed standards, procedures, and frameworks. In the process of doing this, I documented code as much as possible for future reference.

I took notes on the causes of the various bugs I was getting (array overruns, forgetting the end null in `char` strings, etc.). Then I created frameworks (file organization, register storage of small values, extract documentation out of source files). I jotted down items of interest as I progressed to hone my understanding of C/C++ features: adopt practical how-to procedures; create frameworks to accomplish specific tasks; learn how to use AtmelStudio; use regular expressions to facilitate searches; use a *Perl* program to produce product documentation, one of many tasks *Perl* could help me with. It then dawned on me that there were amateur Arduino developers, not professional C/C++ programmers, who could benefit from this work. I could write a *pragmatic presentation* of the C/C++ know-how I had learned. This morphed into *Pragmatic C++ Arduino Programming*. The *defensive procedures and frameworks* I had developed became *Defensive C++ Arduino Programming*.

Writing these two books turned out to be as much of a challenge as designing and creating my *Arduino-based beehive weighing system*. Little did I suspect, when I started putting together the electronic components of my system, that it would take me two years to get the software done, and that I would write two books to convey these newly acquired skills.

Knowhow, good practices, good tools, good frameworks are the keys to success. I should add hard work to this list; success does not come free. Adopting these concepts and tools will make you more efficient: avoid C/C++ gotchas, manage memory, make the program efficient (small and fast), facilitate debugging, understand what you wrote months past because you commented your code correctly, build in safeguards to protect the application as it unravels, modify your code painlessly, etc.

Introduction

I wrote this book with the intent that it should help you, a DIY C/C++ Arduino programmer, evolve from an amateur mindset condition into a professional one, so that you could undertake large complex projects. Time spent adopting good practices, learning how to use professional tools, and implementing frameworks will be amply compensated by less debugging time, much improved adaptability to change, faster more compact code, more robust applications, and a more enjoyable programming experience.

- *Productivity* - Get the project out faster with less pain. Programming should be fun.
- *Maintainability* - Modifications should be easy. Logic should be clear and simple. Complexity should be reduced via encapsulation and frameworks.
- *Robustness* - The application should be free of crashes. This means not using bad pointers, avoiding runaway memory condition leading to stack overflow or heap exhaustion, and staying clear of other such C/C++ mishaps. You will need to hone your C++ skills. The companion book, which dwells deeply into the why's and how's of C++, *Pragmatic C++ Arduino Programming*, could help you with this.
- *Compactness and speed* - Monitoring code size is important because there is a premium on RAM. You may also need to monitor application speed if this were to become critical.

Writing small code which fits in a single sketch file differs from writing large code which needs to be spread across files. The latter requires planning, workflow, tools, and a corresponding mindset. If the application is small (one *.ino* sketch file) you could hack it out directly - mind to keyboard: *incremental programming*. However, if the application is large, you should adopt good programming practices, plan and implement procedures and frameworks, and use professional tools. I refer to this as *defensive programming*. If you do not evolve from a *hobbyist mindset* to a *professional mindset*, you will spend ages trying to get your application to work, the danger being that you might never get it finished.

This book is a compilation of lessons learned and frameworks I developed through two years of experience developing my *Arduino-based beehive weighing system application* (35 files/15000 lines of code). It covers the Arduino IDE (1 and 2), AtmelStudio, Visual Studio, Visual Micro, PlatformIO, VS Code, Code::Blocks, and MPLAB (the last two superficially). It also covers debugging, program organization, program documentation, frameworks, etc. Whichever IDE you use, you can benefit from being disciplined as you work on your project and use frameworks to manage various functional domains.

The chapter *Common sense and new tools* (page 1) should be read first. It presents a concise summary of the content of this book. The chapters *Good programming practices* (page 1), *Frameworks* (page 1) and *Debugging* (page 1) should be read next, they build on each other. The remaining chapters and the *Appendix* (page 1) are self-explanatory. They can be read in random order. They will most certainly influence your development choices.

The main chapters of this book are:

- *Chapter 1 - Common sense and new tools* (page 1) summarizes the main subjects covered in this book.
- *Chapter 2 - Why choose Arduino* (page 1) - You chose Arduino as your core hardware, as

opposed to ESP32, RaspberryPi, or another microcontroller. For my *beehive weighing system*, Arduino with C++ was the best compromise.

- *Chapter 3 - The build toolchain* (page 2) - You have been using the Arduino IDE since you first discovered Arduino. It is a great tool. As this book is going to print, Arduino just released version 2 of its IDE, an improvement over the one we have been using (latest 1.8.19). Version 2 did manifest a few problems; I feel that it is not solid enough to fully replace the old version (referred to as legacy Arduino). I dedicated a section to version 2 (*Arduino IDE V2 specifics* page 2) and covered version 1 more extensively (*Arduino IDE* page 2).
- *Chapter 4 - Hardware setup* (page 2) - Programming your chip is one thing. Getting it to drive the electronics is another thing. Initial experimentation is usually done on breadboards. You ultimately reach a point where the jumble of wires proves unsatisfactory. This is when it is time to get the soldering iron out and migrate to prototype boards. When the application runs properly and the design has been validated, go for your own PCB with its own Atmel microcontroller. You may ultimately develop a final professional grade SMT (surface-mount technology) PCB.
- *Chapter 5 - Interoperability* (page 2) - The Arduino IDE is a fantastically easy to use tool. But, you will find that you will be much more productive if you adopt a professional grade tool such as *AtmelStudio* (page 2), *Visual Studio* (page 2), *Visual Micro* (page 2), or *PlatformIO* (page 2). And there will be times when you will want to switch to using one or the other on a given project. *Interoperability* means seamlessly switching from one tool to the other on the same set of files.
- *Chapter 6 - Which IDE to work with?* (page 2) - We have all started with the Arduino IDE and are probably still using it for our daily Arduino work. But you would be considerably more productive, and your application would be more robust, if you were to adopt a professional grade tool such as *AtmelStudio* (page 2), *Visual Studio* (page 2), or *PlatformIO* (page 2).
- *Chapter 7 - Arduino IDE* (page 2) - You probably know the Arduino IDE well. This chapter presents its strengths and weaknesses.
- *Chapter 8 - AtmelStudio* (page 2) - If you are serious about programming Arduino, you owe it to yourself to look into how much more you can do with AtmelStudio as compared with the Arduino IDE, particularly when doing hardware-based debugging.
- *Chapter 9 - Visual Studio* (page 2) with *Visual Micro* (page 2) is practically the same as AtmelStudio with Visual Micro. So, if you have switched from the Arduino IDE to AtmelStudio, why not do your programming with Visual Studio/Visual Micro, it is far more versatile and futureproofs your knowhow.
- *Chapter 10 - Visual Micro* (page 2) - This plugin is available for both AtmelStudio (MicrochipStudio) and Visual Studio. It adds a serial monitor, an upload tool, serial-based debugging, and more. As for Visual Studio, it renders Microsoft's flagship IDE Arduino compatible.
- *Chapter 11 - VS Code* (page 2) is a Microsoft foundation tool which is designed to be enhanced via extensions. PlatformIO, for example, is such an extension. It is well worth knowing how to use it as it could meet a diversity of needs (Python programming, *Perl*...).
- *Chapter 12 - PlatformIO* (page 2), a VS Code extension, is a formidable tool which could

prove to be your choice if you intend to migrate to ESP32 or other microcontrollers.

- *Chapter 13 - Other IDEs (Code::Blocks and MPLAB)* (page 3) - Code::Blocks is a good free IDE for C++ development but which, unfortunately, does not seem to be Arduino compatible. MPLAB is a professional grade tool for general Microchip microcontrollers embedded development.
- *Chapter 14 - Debugging* (page 3) - Three debugging techniques are presented: print-based (*Print-based Debugging Framework* page 3), *Hardware-based debugging* (page 3), and *Serial debugging* (page 3) provided by Visual Micro.
- *Chapter 15 - Good programming practices* (page 3) - Programming is a craft. A set of simple guidelines, easy to implement but which require a little discipline, should contribute to your improving your performance.
- *Chapter 16 - Frameworks* (page 3) are application independent modules designed to accomplish specific tasks (data transfer, data storage, etc.). Taken together they constitute a toolbox which enhances reuse, robustness, and productivity.
- *Chapter 17 - Three Should know tools* (page 3) are presented: *regular expressions*, *Awk*, and *Perl*. *Regular expressions (regex)* (page 3) is a tool which does text searches using wildcards. Do yourself a favor, learn the fundamentals of regular expressions. It will provide you with unsuspected possibilities when doing text analyses, such as search/replace, syntax analysis, etc. *Awk* (page 3) and *Perl* (page 3) are text processing programming languages designed to implement complete programs. *Awk* is basically used for search and replace. *Perl* is *Awk* with extended programming capabilities.
- *Chapter 18 - From back of the envelope to final product* (page 3) - Once you have gotten the electronics to work, i.e. you have created a soldered prototype of your gizmo, you may have to do a lot more work such as creating a PCB, fabricating enclosures, sensor supports, and battery holders, whatnot. These require tools for creating PCB files (Gerber files with *Eagle* page 3), a CAD tool to create 3D parts (*Fusion 360* page 3), and get them 3D printed (*Zortrax 3D printing* page 3). You will find descriptions of the tools I use and of other tools I did some research on.
- *Chapter 19 - The Appendix* (page 3) expands on certain subjects: application code size, DOS box, memory use, frameworks, misdoings, *Awk*, and *Perl*.
- *Chapter 20 - The Bibliography* (page 3) lists a few books which might be helpful to deepen your C++ know-how. It also includes Web links and YouTube videos.
- *A note on the book's source code* (page 3) - This book and its companion *Pragmatic C++ Arduino Programming* contain C++ code, C++ frameworks, and *Awk* and *Perl* programs. These are provided on an as is basis, free for noncommercial use, no guarantees whatsoever as to quality and suitability, under an MIT type Open-Source Licensing basis (go to md-dsl.fr to download the code). The initial release is incomplete, I shall improve on it progressively.
- *About the author* (page 3) - I, the author, have led a rich multi-national, multi-discipline life. My current interests are beekeeping, flying the airplane I built (RV8), creating electronic devices, 3D printing (Zortrax + Fusion 360), writing these books and possibly more to come, and much more a retiree indulges in.
- *Beehive weighing system* (page 3) - This two-book set, *Pragmatic C++ Arduino Programming* and *Defensive C++ Arduino Programming*, would not exist had I not embarked on creating

an *Arduino-based beehive weighing system*.

- The *Index table* (page 4) was designed so that you could find in it just about all of the book's content meant to be accessed by pertinent keywords.

In a nutshell, the contents of these two books cover C++ knowhow, good programming practices, IDEs and other tools, methodologies and procedures, and frameworks to help you be more productive when creating large, fast, compact, robust, and maintainable *Arduino-based* C++ applications.

Common sense and new tools

Developing the software to drive my *Arduino-based beehive weighing system* turned out to be an unanticipated adventure. I developed methodologies which I qualify as common sense and discovered new tools (AtmelStudio, Visual Studio, Visual Micro, VS Code, PlatformIO, *Perl*, *regular expressions*) which enabled me to be considerably more productive.

What follows is a set of common-sense directives and advice on tools to improve your programming and the application's quality.

- *Plan ahead* - If a module gets a little complex, step away from the computer for a spell. Sit down at a table with pencil, paper, and eraser to draw flow charts; list objectives, requirements, parameters. Brainstorm the module, how it works, what it is meant to accomplish. This phase is crucial. Assiduous attention to detail at this stage will save you tons of time later on. Furthermore, when coding directly on the computer, your hands being tied to the keyboards and your eyes glued to the screen, you are subject to tunnel vision and to getting lost in details. A large table with paper and books spread out liberates your mind. You are given free rein to enhance your creativity and imagination. You are freer to express itself. You will do a better job.
- *Top-down design, bottom-up details* - Use wrapper functions and C++ classes to define the big picture, *top-down* or *outside-in*. Use functions to handle details *bottom-up* or *inside-out*.
- Use a *smart C/C++ editor* - Imagine writing letters with *Notepad* as opposed to using *Word*. *Word* will format text, correct spelling, do grammatical error checks. The same holds with coding. A good programmer's editor will color text based on syntax, indent code, do code completion, correct and/or signal errors, and much more. It is important to use the correct tool for the job. *Notepad++* or the embedded editors in AtmelStudio, Visual Studio/Visual Micro, or PlatformIO are alternatives to the Arduino IDE.
- A good *C/C++ compiler* will generate good code and will error check or issue warnings. Fortunately, all four IDEs covered will do this (Arduino IDE, AtmelStudio, Visual Studio with Visual Micro, PlatformIO) since they incorporate the excellent GNU compiler.
- *Migrate to Visual Micro for Visual Studio* (page 5) - The Arduino IDE is great; without it, you most probably would never have adopted Arduino. Visual Studio/Visual Micro is more complete; the learning curve is short; and the Arduino sketch creation process is easy and works well. Your productivity and programming comfort will be immensely improved. They provide several other important features such as grammar checking as you type, goto implementation, direct from error reporting to source code, and more. Adopting them is a no brainer. What is more, it is possible to establish interactivity between IDEs, switch from one IDE to the another in less than 30 seconds, by activating an IDE specific `#define` macro. You could also envision using *PlatformIO* (page 6), but I find Visual Studio/Visual Micro more user friendly. Program with Visual Studio/Visual Micro; hardware-debug with AtmelStudio (see *Hardware-based debugging* page 6).
- *Adopt Good programming practices* (page 6) - There are resources (papers, videos, seminars, etc.) which advise on procedures and good programming practices (upper/lower-case, error checking, naming conventions, file organization). The sheer quantity is

overwhelming and generally addresses professional programming teams. Having looked at many of these, I compiled a subset of simple rules and procedures which should help you reduce errors and render programs more readable: Adopt naming conventions, break big functions up into several smaller functions thus easier to manage, comment your code, etc.

- *Frameworks* (page 6) - Source file organization, data storage, memory management, error reporting, etc. are tasks an application needs to implement. Frameworks respond to such needs; these ready-made tools enhance the programmer's productivity and *the application's robustness*.
- *Memory management* - Monitor RAM to avoid being out of heap or stack space. Bring things out of scope early to release memory. Optimize data storage. Whenever possible, encapsulate code in curly braces. Managing memory is one of the keys to robust, fast, compact, efficient code. See *Memory Management Framework* (page 6) and *Memory structure (appendix)* (page 6).
- *Documenting your code is often an afterthought*. My Perl based *Program Documentation Framework* (page 6) meets two objectives: document the code in a highly organized way (this cannot be overemphasized) and extract the corresponding documentation. The golden rule being *Comment! Comment! Comment!* Why not use a little discipline to structure these comments so that *Perl* could extract program documentation?
- *Use Perl* (page 6) *as a static checker of vicious bugs* - Simple mistakes like = instead of == in an if statement, or the reverse in an assignment statement, can remain unnoticed for ages. Attempting to discover a problem's origin can eat up precious time. A *Perl* program will extract all pertinent lines of code to enable you to peruse through them thereby detecting potential problems. I used it on my 15000 lines application and found three such mistakes. It saved me hours of debugging.
- *Regular expressions (regex)* (page 6) - If you are to use *Perl*, or profit from the fact that professional grade IDEs (AtmelStudio, Visual Studio, PlatformIO) search functions support regular expressions, then, by all means, learn how to use *regexes*. They can get cryptic however you only need a basic understanding to quickly undertake tasks you would have otherwise thought impossible. The learning curve is short.
- *Hardware-based debugging* (page 6) is a desirable activity since it should save on debugging time. Hardware-based debugging is based on interfacing the microcontroller with a hardware tool which enables step-by-step program execution (set breakpoints, display variable values). Atmel proposes many tools, 3 of which seem to respond to Arduino hardware-based debugging needs. These require that AtmelStudio be used. *Serial debugging* (page 6), provided by Visual Micro, is an additional debugging method. It lies halfway between print-based and hardware-based debugging. See *Debugging* (page 6).

Why choose Arduino

This book addresses the needs of Arduino developers who embark on major projects. By this I mean a project that contains subsystems which interact with each other: many modules, many conceptual entities, many files. The question remains: *Why choose Arduino for a particular project?* There are lots of other microcontrollers. *Why Arduino?*

Most of us started using Arduino because it is readily available, inexpensive, and particularly easy to work with. There are thousands of developers, possibly millions, which explains why there are libraries for just about every electronic component. A GitHub search on Arduino yields more than 100,000 results. And, it is inexpensive, but it carries a cost. Its main programming language is C++, a professional grade language which can lead the inexperienced to excessive debugging time. There is the alternative of using Python to program Arduino, but this carries important costs, greater memory requirements, slower application, and reduced programming possibilities. But programming is easier and faster.

It is a sure thing that, as with most things in life, a choice is a compromise. Certain criteria weigh more than others in a particular context thereby tipping the scales. Here are reasons for my having chosen Arduino and C++ for my *Arduino-based beehive weighing system* project:

- *Cost* - I developed my *Arduino-based beehive weighing system* for personal use, cost was not an issue as long as it remained reasonable. But, lurking in the back of my mind, if I succeeded in developing a sufficiently low-priced useful gadget, it could be turned into a money-making venture. Low component costs thereby became high priority. This implies the smallest possible program to minimize microcontroller size, thereby imposing C++ right away. All other development languages take up considerably more space and are slower. Hardware cost was another issue. My own PCB stand-alone based Atmel microcontroller would be less expensive than designs using board-based microcontrollers such as the ESP32 or RaspberryPi.
- *Development tools* - Most cross-development toolchains are costly from an Arduino developer's perspective. They are reserved for professional teams, one main provider being the Swedish company IAR Systems (check them out - iar.com); they propose a rich set of tools for just about every microcontroller. Luckily, Arduino developers may use one of several free development environments, the *Arduino IDE* (page 7) developed and maintained by the Arduino organization (arduino.cc), *AtmelStudio* (page 7), a Microsoft Visual Studio specifically designed for Arduino microcontrollers, *PlatformIO* (page 7), a plugin for Microsoft's VS Code targets embedded applications on numerous microcontrollers, including the ESP32. Finally, you may opt for *Visual Studio (Visual Micro)* (page 7), an almost identical twin of *Visual Micro for AtmelStudio (MicrochipStudio)* (page 7).

Other IDEs (Code::Blocks and MPLAB) (page 8) are either not suited for Arduino development (Code::Blocks) or entail an extensive learning curve (MPLAB).

- *Sensors* - There are thousands of Arduino compatible sensors, boards, and devices. By Arduino compatible, I mean devices for which someone has developed an Arduino interface in C/C++. This alone can tilt the scale in favor of Arduino. There is no point in choosing a particular microcontroller if the desired device interface is not available.

- *Community* - When a problem arises, it is comforting to rely on a community of savvy developers who can respond knowledgeably to an issue. The Arduino community is huge. Just about every imaginable question has been posed.

Arduino with C++ was an obvious choice. I could have chosen RaspeberryPi because it is powerful, can be programmed in Python, and sits on top of an operating system, which makes it a minicomputer - keyboard, screen, and all. It is, however, more costly since it comes as a board as opposed to Arduino whose Atmel microcontrollers can be bought as stand-alone items.

The build toolchain

There is a lot more to getting a microcontroller to do what you intend it to do than downloading the Arduino IDE, creating a program, followed by a click on the build/upload button. This is what you would do when trying out the Arduino IDE with the *Blink* program. But, if you are to create an application which does substantial work, you will necessarily end up with many functions, many classes, i.e., many lines of code in many files. Upon attaining this level, you shall need to understand what is happening behind the scenes to remain productive, i.e., what the Arduino IDE is doing. You shall also need to look at debugging options other than putting occasional prints in your source code. And finally, you should examine professional grade IDE's; they will enhance your productivity and their learning curve is usually not steep.

This chapter covers the behind-the-scenes tools driven by the IDEs (compiler, linker, etc.). The chapters which follow cover the IDEs individually, starting with the Arduino IDE.

When you create a program with the Arduino IDE and click the build/upload button, a great deal of work gets undertaken behind the scenes. Understanding what is going on is essential to your being able to figure out what is happening when things get a little awry. The only tool you see is the Arduino C++ editor along with the IDE's toolbar. The behind-the-scenes tools (compiler, linker, etc.) take your C++ code and magically do what it takes for you to witness what your electronic gizmo does.

The build toolchain was extensively covered in the companion book *Pragmatic C++ Arduino Programming*. Here is a recap:

- *Intelligent C++ editor (next section)* - Creating a program relies on a good editor with which to write code. The source code is then passed to the preprocessor to handle macros, if any.
- *Make* (page 9) - C++ being multi-module, make handles which modules the IDE is to work on depending on their timestamp compared against the last build's timestamp. It controls work so that it should only process files which have been modified since the last build.
- *Preprocessor* (page 9) - C++ offers programmers a unique tool, the preprocessor. It enables doing text substitution, conditional inclusions, and file assembly. These tasks are controlled by macros which the preprocessor recognizes. Please note that this is an extremely useful C++ feature not available in most other mainstream languages although some may have third-party preprocessors available as plugins.
- *Compiler* (page 9) - Human readable programs (source code) need to be converted into machine code for the target microcontroller.
- The *Linker* (page 9) assembles modules and library components into an executable.
- The *Uploader* (page 10) loads the executable into the microprocessor so that it be stored inside one of its permanent memory, flash memory in Arduino's case.
- The *Bootloader* (page 10) is a program which resides in the microprocessor; it converts it into being its own ISP (In System Programmer). The ISP is electronic hardware whose role is to load a program into a microcontroller. The bootloader makes it so that the

microcontroller becomes its own ISP.

- *Serial terminal* (page 10) - If you want to send information to the outside world, you can do so by displaying it on a screen connected to the microcontroller via a serial port. Your application might not require your doing so; but, during the development phase, you might want to investigate what it is doing. Your only recourse might be to print values as the application runs; this is when a serial terminal comes in handy.
- *Hardware setup* (page 10) - Developing your gizmo may start out by piecemeal breadboard proof-of-concept work gradually evolving to a final SMT (surface mount technology). There are different types of hardware you may use at each stage of the development process (breadboard, prototype board...).
- *Debugging* (page 10) - It is a fact of life that as you program, you make mistakes; your program fails to work properly - it has bugs. Finding bugs is what debugging is all about. You can do so by placing prints to get to know what your program is doing; or, you can use a hardware-based debugger to stop the application as it runs, and look at the program's state.

These are summarily described below. *Debugging* (page 10) is described in its own chapter.

1.1 Intelligent C++ editor

If you are to create a C++ application for your gizmo, you must write the code into a file and feed it to the build toolchain. The editor should do a lot more than just indent code according to its scope level. It should color code according to syntax; do name completion to help you avoid misspelling already defined variables; functions, etc.; detect programming errors from improperly written constructs, (missing parentheses, missing items,...); report undefined entities...

Dedicated programmer editors such as Notepad++ and Arduino IDE V1.8 are basic editors. Arduino IDE V2 begins to provide some editing enhancements. *AtmelStudio* (page 10), *Visual Studio* (page 10), and *PlatformIO* (page 10) incorporate intelligent editors which vastly improve programmer productivity.

1.2 Make

If your program is small, one or two small modules, you may resort to recompiling everything and create the executable. But what if your program is large, 40 modules, 100,000 lines of code. You should only recompile modules which have been modified since the last build, one or two instead of forty. The IDE should create a make file which establishes which modules to recompile and which modules to assemble to create an executable. Fortunately, all IDEs covered in this book use *make* to control the build process.

1.3 Preprocessor

C++ has a unique feature referred to as macros. These are special variables which begin with a # (pound sign) which the preprocessor recognizes. They enable accomplishing three fundamental tasks:

- *Text substitution* - If you write `#define BUFFER_SIZE 256`, the preprocessor replaces every occurrence of `BUFFER_SIZE` with `256`.

- *Conditional inclusions* - You may want to differentiate targeting an ATmega328P from an ATmega2560. You can do this by placing code inside an conditional inclusion such as `#ifdef ATMEGA328P...#else...#endif`.
- *File assembly* - Files can be pasted inside other files via a macro such as `#include <Arduino.h>`.

C++ is unique in that no other language except possibly Fortran formally offers this feature. You may however find third-party preprocessors for other programming languages (Python in particular).

1.4 Compiler

The compiler transforms human readable source code (*.h/.cpp* file) into machine code (*.o* file). Because C++ applications can be very large, they are multi-module. The compiler works on one module at a time.

1.5 Linker

The linker creates your program. Once the compiler has finished compiling all of a program's modules, the linker assembles them, along with library components, into a whole, an executable, also referred to as image (usually a *.hex* file).

1.6 Uploader

Once the executable has been created, it must be uploaded into the microcontroller. A specialized program, *avrdude*, handles this task. It sends the program to an ISP, electronic hardware whose job is to load (burn) the program into the microcontroller. One of the neat features of Arduino boards resides in their becoming their own ISP via installed software, referred to as a *bootloader* (next paragraph).

1.7 Bootloader

The ISP (In System Programmer) is a piece of electronic hardware which receives the program via its com port and transfers it into the microcontroller's memory. An ISP must consequently be placed between the development computer and the microcontroller board. The *bootloader* is a program installed in an Arduino board which enables it to become its own ISP; an external ISP is no longer needed. The price to pay is little less flash memory.

1.8 Serial terminal

It is all good and well to create a gizmo which opens a chicken coop door at sunrise and closes it at sunset. But, during the development process, you may want to display info to understand what the program is doing. You do this by writing to one of your computer's serial terminal applications. The terminal could also be a LED screen connected to your board.

A note on the book's source code

Frameworks source code: You may download the complete *as is* source code from md-dsl.fr/c-arduino-programming, modify code to your heart's content, and use it free of charge at your own risk on a non-commercial or commercial basis. It is subject to an MIT open-source type license agreement with some restrictions, as follows:

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to incorporate it inside of an executable or other machine language component without restriction for personal or commercial use. The Software may not be redistributed as source code or in any recognizable human readable form in any form whatsoever for any use whatsoever.

The Software is the property of Michèle Delsol (France), copyright 2023, USA and international. For any questions concerning use of the software contact Michèle Delsol at CPPArduino@md-dsl.fr.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Note that some frameworks have benefited from extensive development work, and they are quite solid; others are in their infancy and may be buggy.

You will find, throughout this book, code snippets, classes, functions... to illustrate how C++ works. They cover a lot of material. Some of them are short and to the point; others are longer and can be more or less complex. I have tested all of them, at least I think I have, which means that you might find omissions, mistakes, inaccuracies... If you happened to come across such failings, please send me an email at cppArduino@md-dsl.fr explaining what it is you think is wrong. I shall look into it, try to respond, and bring in corrections for the next edition of this book, currently 1st edition.

My Web site account contains the frameworks, and *Awk* and *Perl* programs presented in these two books. To download these, go to md-dsl.fr/c-arduino-programming (a little over 1.5 megabytes).

You will also find notes and acknowledgements as to events concerning these two books in my Web site md-dsl.fr.

About the author

The author, Michèle Delsol, born in France, educated in the USA (MIT - B.Sc., Sc.D.), now retired and living in France, has worked in industrial firms in South America, the USA, and France. For the last 25 years of her career, she was CEO and CTO of the company she created. As CTO she gained experience working with Fortran, Fort, Lisp, Java, JavaScript, Visual Basic, PHP, HTML, and C++.

Not shying from rolling up her sleeves, she is practical and dives into hands-on work. Her passion for flying led her to build and fly her own airplane (RV8). She also claims some artistic capabilities (the warthog and marmoset on the covers of this two-book tandem are hers), did some acting (theater), and is now an enthusiastic beekeeper.

Her most recent electronics endeavor is an *Arduino-based beehive weighing system* to help monitor her bees' wellbeing (full details in the author's book *Defensive C++ Arduino Programming* - see *Bibliography - C/C++ programming* (page 39). It began as a back of the envelope idea which grew into a full-fledged system. As the project progressed, from proof-of-concept of the individual components to a comprehensive integrated system, the application grew to more than 35 files (15,000 lines of code). Her early development was fairly undisciplined and incremental, which led her to too much time debugging - what was initially an enjoyable pastime became a gruesome burden. She consequently stepped back and researched why programmers make mistakes. She found that she needed to adopt good programming practices and use better tools. It brought her to switch to AtmelStudio and later to Visual Studio/Visual Micro in lieu of the Arduino IDE, to extensively review C++, to create frameworks to handle specific tasks, to relearn adequate *Awk* and *Perl* to extract documentation from the source files, to learn regular expressions, and to undertake other useful programming chores.

This experience led her to write extensive notes on the material she covered to keep track of things as the literature and the Internet were lacking in resources for amateur programmers who already programmed in C++. Or rather, there was too much material, most of which was unprofessional, verbose, and did not address the question posed directly. Ask a simple question, get long complicated responses. These notes gradually morphed into two books: *Pragmatic C++ Arduino Programming*, a reference work to help already C++ savvy Arduino programmers avoid the many gotchas C++ can throw at them, and *Defensive C++ Arduino Programming* which presents C++ tools and frameworks to improve programmer productivity to write efficient, robust, maintainable, and compact Arduino applications.

Her *Arduino-based beehive weighing system* is not quite finished. Future work would focus on transferring her project to an *Open-source Software* team, on redesigning the PCB for SMT technology, on implementing alternate communications to handle remote areas where GSM is not available, on developing Web and smartphone based user-friendly interfaces, and more. Whether these will be undertaken remains to be seen since she has other projects she intends to work on. Amongst the many ideas that float in her mind, a device to detect Asian wasps hovering in front of the beehives. She also plans to interpret a colony's activities via the sound they make. These projects are a tall order - they imply learning and implementing digital signal processing, digital image processing, and AI. These should keep her busy for years to come. Needless to say, she is never bored. Aside from the shared benefits that beekeepers might gain from her hard-won endeavors, her ongoing satisfaction lies in the challenge and the doing.

Beehive weighing system

Life being a collection of chance encounters, a friend of mine (Charlie H.) introduced me to beekeeping, an immensely rewarding activity. One of my first tasks as a new beekeeper was to monitor beehive weights via a hand-held scale. This information is crucial to assess how bees are doing.

Another chance encounter (Daniel T.) introduced me to Arduino. It seems that life is conditioned by chance encounters. Anyway, as soon as I understood what I could do with Arduino, I asked myself: *Why not hack an electronic Arduino-based beehive weighing system?* On the surface the idea was simple enough: get strain gages cannibalized from ordinary bathroom scales, throw in opAmps, voltage regulators and a few other little gizmos. Add to that a GSM board, a radio, and a clock and I would get the beehives' weights, temperatures, humidity, and battery voltages on my cell phone. Simple! Nothing to it! Well, not quite.

To make a long story short: *beekeeping + Arduino + C++ = electronic Arduino-based beehive weighing system = challenge*. Little did I know how much I did not know and just how much work would be required to bring this project to fruition. Having defined how to get the data from the beehives, and how to get them into my cell phone, I trudged along, slowly, and painfully. I grossly underestimated the work and amount of learning required to get the system to function as I would like it to.

My electronic know-how being limited to Ohm's law, the challenge proved to be daunting. Being retired, with nothing in particular to do aside from keeping fit, maintaining social ties, and being a law-abiding citizen, this project quickly became a full-time job. I was never bored. I had to crash course transistors, opAmps and voltage regulators. Discovering this world was fun. Getting an NPN/PNP transistor-based latch to work, and understanding why it worked, brought me intense satisfaction.

As for Arduino, what a discovery! The microcontroller and the Arduino IDE are fantastic and cost next to nothing. My enthusiasm was immediate. I first got breadboard proof-of-concept prototypes working. I got strain gages which I pulled out of an electronic bathroom scale to deliver a signal via an opAmp. Radio communications were based on XBee (Zigbee protocol). I chose the DS3231 RTC clock to synchronize the units to minimize wakeup time. An A6 GSM board established two-way SMS communications with my cell phone.

1.9 Beehive weighing system architecture

As I became savvier with the electronic components I could use, system characteristics slowly percolated out of a host of possibilities:

- *Parent-child architecture* - I opted for a parent/child star network system as opposed to independent beehive modules, each reporting directly. Children would be on beehives. They generate data and send it to the parent (coordinator) which would be near-by. It would collate data from the beehives and send the info to my smartphone.
- *RF communications* - The coordinator and beehives communicate with each other via RF communications, no wires between the beehives. I chose a ZigBee based system (XBee).
- *Batteries only* - Since beehives are in remote locations, they require electrical autonomy, making batteries necessary (18650 LiIon). I rejected the use of solar panels since they

would be conspicuous, obviously valuable items. There was every likelihood that they get stolen.

- *Cost* - If the system were to become a commercially viable product, cost would be an issue.
- *Ease of use* - The system needed to be flexible: When to weigh and how often, get data to trigger notifications on specific events, etc.
- *Interface with the outside world* - I had to decide how to interface with the system (Firefox, Lora, WiFi, Bluetooth, cell phone). I chose an A6 GSM board to communicate with my cell phone as it seemed to be the easiest, most practical, and immediate solution.

Given three years of hindsight, I must say that the electronic aspect of the project was indeed easy. However, the C++ program to manage the system proved to be a lot more complex than anticipated. I was no longer inside an *.ino* sketch mindset - direct programming from mind to computer. The application grew inexorably to 35 files of code at last count. I had to improve my C++ know-how, develop new techniques, and change my mindset - professionalize my work.

As for my know-how, I had C++ experience. I poured over Kernighan and Ritchie's *The C programming Language*. I then went on to Bjarne Stroustrup's *The C++ Programming Language*. But my experience was rusty. I still have early editions of these books; I consider them collectors' items.

I gradually improved and organized my programming by various means: C/C++ know-how, good programming practices, frameworks, and better IDEs (AtmelStudio, Visual Studio + Visual Micro, PlatformIO). These helped me attain the desired objectives: productivity, application maintainability and robustness, speed and compactness.

I have finished the first version of my *Arduino-based beehive weighing system*. The next phase is to run it through its paces on several beehives and establish its resistance to weather. After that, if I were to make it into a commercially viable product, I would need to undertake fairly extensive work, incorporate less expensive components, use a cheaper radio, create alternative interfaces to the outside world, and use SMT based PCBs to lower costs.

Where to from now

The current version of the beehive is functional however, several slight problems remain:

- The XBee radio modules are far too expensive (\$30.00 apiece). Current work centers around using BRF24L01 modules. They come in at around less than \$3 apiece.
- Using a cell phone to get data is cumbersome because of the limitations of an SMS message and costly because of the requirement to pay for SIM cards.
- A friendly Internet based user interface to visualize data and schedule jobs needs to be put together.
- Envision migrating from through-hole to surface mount technology (SMT).

As soon as these books get published, I shall resume work on the *beehive weighing systems*.

Index table

Symbols

?		
Regex lazy search	188	
/*...*/		
C-style comments collapse supported by AtmelStudio and by the Arduino IDE	52	
\		
pass by reference Perl operator	311	
%		
Perl provides three variable types: \$scalars, @arrays, %hashes	313	
+=		
AddHive illustrates += operator overloading	174	
=		
= instead of == in an if or while, == instead of = in an - assignment	109	
==		
Perl pattern matching operator	207, 210	
\$ (dollar sign)		
\$0 - Awk: record just read in, Perl: program name (also \$PROGRAM_NAME)	205, 297, 311	
Awk \$1, etc. contains tokens of record just read in; Perl contains captures from regex	297, 312	
Perl \$, alternate \$INPUT_LINE_NUMBER or \$NR, line # of file being processed	312	
Perl \$, alternate \$OUTPUT_FIELD_SEPARATOR, or \$OFS separates fields from each other	312	
Perl \$\ alternate \$OUTPUT_RECORD_SEPARATOR (default newline)	312	
Perl \$; alternate \$\$SUPSEP - index separator, comma by default	312	
Perl \$ARG - same as \$_, is the complete record (line of text)	312	
Perl \$ARGV array contains current file name when reading from <> array	312	
Perl - \$ _ contains the entire line that has just been read	309, 312	
Perl \$INPUT_RECORD_SEPARATOR, or \$RS, or \$/- default \n	205, 307, 312	
Perl \$NR alternate \$INPUT_LINE_NUMBER or \$., line # of file being processed	312	
Perl \$OFS alternate \$OUTPUT_FIELD_SEPARATOR or \$, separates fields from each other	312	
Perl \$RS alternate \$INPUT_RECORD_SEPARATOR or \$/- input record separator, default \n	312	
Perl provides three variable types: \$scalars, %hashes, @arrays	209, 313	
\$File handle See Perl (file handles)		
\$ProjectDir, \$TargetDir, \$TargetName See AtmelStudio (build)		
3D Printing		
3D printing starts with CAD tool	216	
Fused deposition modeling, Stereolithography, Selective layer sintering	217	
I chose a Zortrax 200M 3D printer	216	
Several 3D printing techniques available	216	
Standardized file format .stl for 3D printing	216	
Use Fusion 360 to design part	216	
3D Printing See Fused deposition modeling FDM		
@ARGV (Perl array)		
Perl arg1 arg2... command line arguments stored in @ARGV array	302, 312	
@array See Perl (arrays)		
@ (at symbol)		
Perl @_ alternate @ARG array passes arguments to a function	312	
Perl @ARG or @_ - array passes arguments to a function	312	
Perl provides three variable types: \$scalars, %hashes, @arrays	313	
__brkval		
Address of begin heap instead of __flp when there are no holes	282	
#define See Macros		
__flp		
Address of begin heap when there are holes	282	
-flto flag		
Code size depends on -flto flag; reduce by 40%	34, 57, 60, 221, 222	
-fpermissive		
AtmelStudio (import Arduino project) - add flag to C/C++ compiler options	39	
Converts compiler errors into warnings, build completes	39	
Included by default in Arduino IDE, PlatformIO, not AtmelStudio	39	
%hash		
Perl provides three variable types: \$scalars, %hashes, @arrays	209	
.h/.cpp files		
Arduino loads all project directory .h/.cpp file into		

the editor	27	Compile/upload uses bootloader to upload program	40
.hex files See bootloader		-flto and -fpermissive flags included by default in Arduino IDE project properties	221
.h files See Header files (.h)		.ino file is Arduino's entry file (C++ source code)	27, 142
#ifdef...#endif		Installs basic and third-party libraries	26
Collapsing supported by AtmelStudio	52	Loads all project directory .h/.cpp file into the editor	27
#if, #ifdef, #ifndef, #endif See Macros and Project Files Framework		main is generated behind the scenes, calls setup and loop	28
#ifndef and #include See Macros and Header files		setup and loop - mandatory functions for Arduino programs; leave loop blank	28
.ino See Arduino (build) - application entry file (sketch)		Specify microcontroller and board, changing them transparent to programmer	69
{ } See curly braces		Suggestion: do not use loop, do inside setup	28
:: See DOS batch files		ArduinoBuilder See Code::Blocks	
# sign See Perl and Awk comment symbol, same as C++ style //		Arduino (caveats)	
-Wlint flag		After a while, the Arduino IDE gives up, undo ctrl-Z messes up code; fragmentation fault blocking	30, 31
Awk - use -Wlint command line flag to display errors	203, 296	Compiler error reporting - must manually go to file/line # to access code	55
-Wlint flag See Awk (command line)		ArduinoDev See Code::Blocks	
-Wundef See AtmelStudio (build)		Arduino (editor)	
A		Button compiles and uploads application	27
Aardvark		Collapses curly braces, not AtmelStudio	123
Aardvark() contains original setup code, key to interoperability; in Aardvark.h/.cpp	20, 21, 22, 142	Ctrl-L takes you to line # in file	56
Aardvark See Arduino, AtmelStudio, Visual Studio/ Visual Micro, PlatformIO(interoperability)		Edit menu item changes workspace font size	59
Action-blocks See Awk (rules)		Indentation, code collapsing not cosmetic	119
Algorithm Test Framework		Serial monitor interfaces with board	28
Define execution paths and controlling parameters, then fill in real code	163, 259	Arduino (general)	
Formal procedure to test program logic; logic and work done are distinct concepts	163	Arduino chips are AVR type	111
Process: decision tree several levels deep	163	Arduino distribution seamless installation; IDE, libraries, toolchain, serial monitor	26, 27
Start with flow diagrams to peg down algorithm's logic	163, 259	Arduino tools are user-friendly and free, components cheap, community huge	7, 25
Two-pronged process: methodology (plan on paper and then code), and thinking (top down how one thinks and bottom up fill-in the details)	xvi	Distribution includes: tools to create program, libraries, upload, serial terminal	25, 26
Unravel add beehive logic with flow diagram	259	For new Arduino IDE version see Arduino (version 2)	25
Application code size See Code size		IDE for Atmel 8-bit and ESP32 boards, no hardware-based debugging	25
Application entry files		IDEs: Arduino, AtmelStudio, Visual Studio, VS Code, PlatformIO, Visual Micro	25, 88
Arduino: .ino file, AtmelStudio and Visual Studio/ Visual Micro, PlatformIO: main.cpp	142	Two versions: 1.8.19 (legacy) and V2 (new features)	25, 28
Arduino (build)		Wraps tools inside a clean, practical interface	27
Arduino, AtmelStudio, PlatformIO, Visual Micro use the GNU C++ compiler	5, 26	Arduino (interoperability)	
Arduino project's name is that of .ino file	27	ARDUINO_IDE macro enables Arduino IDE	22
avrdude uploads the executable into the microcontroller	26	Interoperability between Arduino IDE, AtmelStudio, PlatformIO, Visual Micro, hassle free	23
Build command details	222	Arduino (upload)	

18 | Index table

Arduino build output	40	Build fail due to inexistant file in project not in error messages	58, 59, 67
Full paths of avrdude.exe and avrdude.conf	40	Careful with find/replace Opened Documents	59
Validate verbose checkbox in options to show details of avrdude upload command	40	Code changes no effect on runtime	61
Arduino (version 2)		Code remains obstinately collapsed	59, 69
Click on compilation error in output window does not take you to file/line	30	Code size by default 40% too large, needs -fno flag	34, 57, 60, 221, 222
Copying text from serial terminal problematic	29	Does not support curly braces collapsing	46
C-Style /*...*/ comments collapsing inoperant	30	Error and warning messages not displayed	65
Editor widows not undockable	29	Extra */ causes indentation failure	58, 62, 63
Find/replace supports regular expressions	29	Failure to reckon with Debug vs. Release output causes confusion	58, 61
#ifdef...#endif pairs collapsible but renders function not collapsible	30	File has 20+ functions, only two showed up in the quick access selection box	58, 62
Look and feel similar to legacy version	29	Importing Arduino project - directory navigation bug on searching .ino file	37, 57
Major advance over version 1, long way to go	30	Indenting of if...else statement depends on where opening curly brace is placed	58, 63
New features and minor problems	28	Make error from missing .cpp file drowned in reams of messages	58, 65, 66
New output window: serial plotter	29	Mysteriously imports second set of header files into ArduinoCore directory	58, 62
Problem loading code into ATmega328P-Xmini	29	Occasional segmentation faults are benign; no clues on causes; do rebuild to fix	58, 64, 65
Provides name completion on defined items	29	Select device board does not seem relevant when creating Arduino project	38
Refactoring does not seem to be supported any more	30	Workspace font size not documented; - '>' and '<' become ',' and ';' on French keyboard - tied to physical key, not to key mapping	57, 59
Right click to visualize function declaration	29	AtmelStudio (documentation)	
Serial terminal is fixed bottom pane, not separate window	29	Atmel (Microchip) Web site two inline doc packages: Getting started and videos	35, 71
Slow to open	29	Inline help under the Help menu; VAssist help under top level pull-down VAssist menu	71
arg1 arg2, etc. See Perl (command line)		AtmelStudio (editor)	
ARGC, ARGV, ARGIND See Awk (command line)		Arduino monitor displays output	37, 44
ARM (Advanced Risk Machine) See Microcontrollers		Bookmark to navigate back and forth	46, 56
Arrays See Awk arrays and Perl arrays		Change signature changes the name and the type of a declaration and refactors	48
Atest.h/.cpp See File organization		Clicking on function opens a goto multiple-choice window for declarations	49
ATmega328P-Xmini, ATmega2560RFR2 Xplained Pro, Atmel ICE See Debugging hardware-based		Code collapsing in AtmelStudio is outlining; indenting is formatting	49, 52, 58
ATmega2560 See Hardware setup		Collapses classes, functions, enums, ifs, whiles, etc. and #ifdef...#endif pairs	46, 52, 123
AtmelStudio (build)		Color coding highly customizable	45
\$ProjectDir, \$TargetDir, \$TargetName define .hex file location and name	41, 61	Compiler one-click access to faulty line	46, 55
Add -fpermissive and -fno flags compiler flags	39	Create declaration in corresponding .h file	48
Add -Wundef to detect unused macros	39	Ctrl-G takes you to line # in file	46, 56
Arduino, AtmelStudio, Visual Studio/Visual Micro, PlatformIO use the GNU C++ compiler	5	Documentation above a function call	48
AtmelStudio upload tool parameters: identify microcontroller, define COM port, baud rate	41	Editor displays colors: functions red, variables blue, comments pale green, etc.	49
Build command details	223	Establish Arduino IDE code collapsing #ifdef	
External tool for upload not needed with Visual Micro	37, 39, 42		
Solution Explorer - add/choose existing Item	38		
AtmelStudio (caveats)			
Arduino core content seems locked into device first compiled for	69		
avrdude - cannot open input file error message hard to see	61		
avrdude fails to upload code into chip	43, 57, 60		

interoperability by adding curly braces	53	hardware debug with AtmelStudio + Xplained boards	74
Extensive user assist editor features	5, 33, 34, 44	Free, professional grade, short learning curve, easy import of Arduino projects	33
Find reference: all uses of variable/function	48	IDEs: Arduino, AtmelStudio, Visual Studio, VS Code, PlatformIO, Visual Micro	33
Format document indents code entire file	63	Improve productivity by orders of magnitude	5
Indentation, code collapsing not cosmetic	119	MicrochipStudio is new name - continue referring to AtmelStudio in book	33
Indenting in AtmelStudio called formatting	63	Supports hardware-based debugging	33
Inline tasks provide multiple-choice text box to rename, create, declare, etc.	34, 45	Visual Micro brings AtmelStudio missing features: upload tool, serial monitor, serial debugging	35
Insert snippet such as if, while, for, etc.	49	Visual Micro enables serial debugging w/o dedicated hardware	116
Intelligent window scrollbars provide quick access to various tasks	45, 47, 50	AtmelStudio (hardware-based debugging)	
Intellisense complementary to VAssist	44	Develop with Visual Studio 2022 + Visual Micro, hardware debug with AtmelStudio + Xplained boards	74
Keyboard shortcuts tied to physical keys or to symbol mapping	59	Hardware-based debugging with ATmega328P-Xmini successful, with Atmel ICE not so	49, 111
Keyboard shortcut to quickly comment/uncomment highlighted sections	46, 54	Visual Studio plus Visual Micro and AtmelStudio + the ATmega328P-Xmini	113
Multiple-choice quick access to functions	45, 48	YouTube Atmel ICE tutorials	114
Name completion contextual popup opens	45, 51	AtmelStudio (interoperability)	
Navigate to places recently worked on	46, 55	Aardvark() contains original setup code, key to interoperability	20, 22
Refactoring - rename items throughout app	46, 49, 54, 67	AtmelStudio Arduino project entry file sketch.cpp contains setup and loop	21, 142
Right-click on function for quick access	48	ATMEL_STUDIO macro to enable AtmelStudio	22
Spell checks strings and comments	46, 56	AtmelStudio's project name is Arduino project name	35
Supports one-click all code collapse	52	AtmelStudio upload tool parameters: identify microcontroller, define COM port, baud rate	41
Surround selected code with a for, if, etc.	49	Copy the Arduino sketch file (.ino file) and .h/.cpp files into the AtmelStudio project files	21
Syntax checking - color goes black when there is a programming error	34, 45, 49	External custom avrdude upload tool needed, not with Visual Micro	39
Unnamed code blocks improve readability, decrease RAM	53	External upload avrdude tool needed	37
VAssist has many help mechanisms	44	Importing Arduino project procedure	18, 21, 36
Workspace contains two splittable unpinable work windows and Solution Explorer	44, 46	Interoperability between Arduino IDE, AtmelStudio, PlatformIO, Visual Micro, hassle free	23, 34
AtmelStudio (file management)		AtmelStudio (keyboard shortcuts)	
\$ProjectDir, \$TargetDir, \$TargetName define .hex file location and name	41	AtmelStudio V7 User Guide - 14 pages which itemize keyboard shortcuts	70
AtmelStudio directories key to interoperability	21	Standard Windows shortcuts supported	71
AtmelStudio target release: Debug or Release	41	Workspace font size not documented - '>' and '<' become ',' and ';' on French keyboard - tied to physical key, not to key mapping	59
Differentiate projects (managed individually) from solutions (managed as group of projects)	35, 37	Workspace font size not documented - '>' and '<' become ',' and ';' on French keyboard - tied to physical key, not to key mapping<	57
Solution Explorer access project components	47	AtmelStudio project and solution See AtmelStudio	
AtmelStudio (find/replace)			
Careful with find/replace Opened Documents	67		
Little orange patches in vertical scrollbars show find location	50		
Supports whole word, regular expressions, past searches	34, 45, 50, 180		
To change application wise item name, consider refactoring instead of find/replace	67		
AtmelStudio (general)			
AtmelStudio Arduino compatible in two versions, without and with Visual Micro	33		
AtmelStudio is Atmel specific Visual Studio	33, 73		
Develop with Visual Studio 2022 + Visual Micro,			

20 | Index table

(file management)	
Avoiding bugs See Bugs (avoiding them)	
avrdude	
Arduino compile/upload uses avrdude to send .hex file to board, bootloader uploads it	27
Arduino verbose details upload command	40
AtmelStudio requires external custom avrdude upload tool, not with Visual Micro	37, 39
Full path of avrdude.exe and avrdude.conf	40
avrdude See Build toolchain	
AVR See Microcontrollers	
Awk (arrays)	
Array indices can be integer, float, strings, similar to key into database	203, 289
Arrays are associative, mix strings and numeric values	289, 291
Arrays defined when array items used	290, 291
delete removes array items	290
length function yields number of items in one-dimensional arrays	290
Loop construct for (idx in array) facilitates array traversal	290
Multi-dimensional arrays supported	289
Two-dimensional array example	292
Using multidimensional arrays is complex	290
Awk (Awk vs. C)	
Array indices can be integer, float, string	203
End of statement semicolon not mandatory	203
Origins trace back to Unix, hence based on the C language	202, 295
Print statement not quite literal, requires understanding its mechanics	203
Strings are 1-based	203
Supports regular expressions	203
The # sign signals a comment to end of line	203
Two variable types: floats and strings - they are not declared, they are used	203
Variables used in functions are global	203
Your C knowhow can be a false friend	202
Awk (Awk vs. Perl) See Perl (Awk vs. Perl)	
Awk (built-in variables and functions)	
\$1, \$2, etc. - contain tokens of record just read in, \$0 contains entire record just read in	297
Command line variables particularly useful: ARGV, ARGV, ARGIND	296
exit stops processing files; END called	297
FILENAME is file being processed, same as (ARGV[ARGIND])	295, 297
FNR is record number of file being processed; NR is overall input record number	297
FS - fields separator (default " "); RS input record separator (default "\n")	198, 297
next, getline interrupt rules processing	295, 298
OFS output field separator controls whether space printed when comma encountered	299
Output record separator - default is newline (ORS = "\n")	198, 299
String functions: index, length, substr, match	298
Awk (caveats)	
Careful with spaces in if, while, functions, etc. parens must be against if, etc.	204
Cycling through array in C-style for loop may create unwanted array items	289
Excessive simplicity cause of gotchas	203
Functions similar to C functions, may return value of any type	202
Function variables global, parameters local	204
Individual characters are strings	204
Program particularly sensitive to spelling errors, no warnings	296
Stray line of code silently interpreted as rule	204
Strings are 1-based	204
Variables typed when used - no declarations	204
Awk (command line)	
Command line: Awk -Wlint -f MyProgram.awk file1 file2 ... > Output.txt	285
Contains flags, program, files to process	296
-Wlint flag displays errors	193, 285
Awk (examples)	
Find macros regex: #define, #ifdef, etc.	294
Generate documentation from code comment templates, list macros and enums	191
Generate function call reference - who calls who and who gets called by whom	191
List potential misdoings e.g. = instead of ==	191
Print subset of a file	191
Regex finds macros: #define, #ifdef...	199
Short database example Awk program	286
Two-dimensional array example	292
Awk (general)	
Download Awk, go to SourceForge	285
Edit Awk, Perl programs with Notepad++	197
Extract info from text files via logical statement & regex	180, 190, 192
Learning curve is shallow - nothing to it	191
Program is sequence of condition/action-blocks (rules) and functions	193, 198
Provides trig, log, type conversion, random numbers math functions	300
Uses Awk may be good for	190
Why I started with Awk, then migrated to Perl	190, 192
Your C knowhow, a false friend when learning Awk	202
Awk (regex)	
Find macros regex: #define, #ifdef, etc.	294

Parens-based captures not supported - Perl does support it	191	Awk (variables and functions)	
Regex finds macros: #define, #ifdef...	199	Function declaration starts with 'function'	202
Awk (rules)		Functions similar to C functions, may return value of any type	196, 202, 295
Action-block similar to function - contains C-like statements enclosed in curly braces	197	Function variables global, parameters local	288
After loading new line, Awk tokenizes it	193, 195	Variables created on the fly when first used	296
BEGIN and END blocks do work at start and end of program	195, 202, 295	-Wlint flag warns nonexistent return	296
Comment is # sign; down to end of line	196, 294	B	
Concatenates files into set of continuous text processed one line at a time	195	Batch files See DOS batch files	
Condition based on pattern search: simple logical statements or regexes	190, 195, 294	Beehive weighing system	
Condition only rule prints line if condition successful	200, 295	Example of DataGroup Framework use	254
Default record separator ORS is newline (line of text); default field separator OFS is spaces or tabs (word in line of text)	198	Frameworks required to improve beehive weighing system program	139
Files broken up into records, records into fields (lines of text/words)	198	How the beehive weighing system got me into writing these two books	xv, 325
getline, next stops processing line	197, 295, 297	Monitor weights, temperature, humidity; based on RF communications and GSM board	325, 326
Invoke Awk from DOS box via command: program name, -w option, file names	293	Test beehive weighing system execution paths with Algorithm Test Framework; unravel add beehive logic with flow diagram	259
Logical statements for simple conditions, regexes for complex ones, or mix of the two	200, 201	BEGIN See Awk and Perl (build)	
Multiple file processing passes possible	286	Bitfield Storage Framework	
No condition rule is always true - action-block invoked	200, 295	BitfieldStorage class provides bitfield storage via inheritance	249
Print equivalent to C++'s Serial.print	298	BitfieldStorage class requires info on variables stored in VarCharacteristics; uses bit-masks to operate on bytes	157, 251
Process text files with sequence of condition/action-blocks (rules)	190, 194	Bitfield storage example, 3 arrays: JobData, VarsMinMax, JobLabels data arrays	156, 157, 250
Program is sequence of condition/action-blocks (rules) and functions	190, 193	Job class inherits from the Bitfield class	156, 160
Provides arithmetic, increment, assignment, unary, logical, text match operators and conditional expression	298	Optimize data storage via bitfield packed variables	155, 248
Provides useful built-in functions and built-in variables	196	Put bitfields in a structure to access them easily via an offset from the structure's address	158
Rule could be condition only or action-block only	195	Variables accesses via enum driven generalized Get/Set functions	158, 248, 249
Rules use fields \$ 1, \$ 2, etc. from line tokenization	199, 294	Bjarne Stroustrup	
Supports standard C program flow control operators if, while, for, etc.	297	Suggests not using macros	133
Variables OFS, ORS control print's behavior	299	Bookmarks See AtmelStudio (editor)	
Awk (strings)		Book's Web site (md-dsl.fr)	
Similar to C++'s String class, char array	298	Events concerning book blogged in md-dsl.fr	321
String functions: index, length, substr, match	298	For comments and info, please send email to cppArduino@md-dsl.fr	321
Strings are 1-based	203	Go to https://md-dsl.fr/c-arduino-programming for a link to download most of the code in the two books	321
Awk (terminology)		Bootloader See Build toolchain	
Rules, conditions, action-blocks, statements, fields...	194	Bottom of stack See Memory (use)	
		Bottom-up design See Object-oriented programming	
		Breadboards See Hardware setup	

22 | Index table

Bugs (avoiding them)

- Apply good programming practices and adhere to Golden rules - check data, never assume anything; do error handling 119
- ATmega328P-Xmini - watch variables to find glitches 6
- Careful with compiler function parameter type leniency and default initializations 129
- Editor's auto indent detects bad curly braces 136
- Inline comments clarify logic, fewer bugs 108
- Ounce of prevention is worth pound of cure 106
- switch default case missing or empty 279
- Two-step approach: review code and apply systematic validity checks 108

Bugs (possible causes and cures)

- Application displays gibberish, restarts, etc. 107
- Apply methodology for hard to find bug, understand the logic 107
- ATmega328P-Xmini - watch variables to find glitches 107, 108
- Brute force approach to finding glitches; pare down application and rebuild progressively 107
- First step to find glitch's cause, eliminate randomness 107
- Glitches worse programming problems one can encounter; how does one fix them? 106
- Strange behavior or crash - probably ran out of memory 167
- Type checking leniency creates bugs due to undetected misplaced parameters 108

Build options See AtmelStudio and Arduino IDE (build)

Build toolchain

- Bootloader transforms board into self ISP 11
- C++ editor, make, preprocessor, compiler, linker, upload (avrdude) 9
- Compiler --> source code to machine code 11
- Intelligent C++ editor - AtmelStudio/Visual Studio best 10
- Linker assembles machine code files + library components into executable file 11
- Make - timestamp defines what to work on 10
- Preprocessor enables text replacement, conditional inclusions, pasting files 11
- Serial terminal - visualize program output 12
- Uploader (avrdude) loads executable into micro-controller via an ISP 11

C

C++ Editor See Build toolchain

C++ exception handling See Exception handling (C++) and Pseudo exception handling

Capture See Awk and Perl (regex), Regex (captures/

groupings)

catch, try, throw See Exception handling (C++) and Pseudo Exception Handling

Character class See Regex (general)

Chunks of data See Data Packets Framework

Class and Function Names Referencing Framework

- Automate class and function IDs insertion 166, 263
- class CandFNames does all the work 263
- Framework used for event reporting 164
- Identify classes/functions via IDs 164, 239
- Instrumentation synchronizes class and function enum lists with names 166
- MemAllocEvt uses function IDs to report with function names 171

Class Data Framework

- Access data via Get/Set enum driven offsets 148, 241

CLion See PlatformIO

cmd.exe See DOS box

Code::Blocks

- ArduinoBuilder - third-party plugin to Code::Blocks 103
- Free, low learning curve C++ IDE; does not seem suited for Arduino dev 103
- FreemeticsBuilder - third-party plugin to Code::Blocks 103

Code collapsing See Arduino IDE and AtmelStudio (editor)

Code size

- Are the IDEs as efficient code size wise 219
- AtmelStudio code size by default 40% too large, needs -flto flag 34, 222
- Code size depends on compiler options and on -flto flag; reduces size by 40% 221, 222
- Compile time code size corroborated by heap space at startup 222, 224
- Develop with AtmelStudio, final release may be preferable with Arduino 222
- Identical compile/link options yield different RAM/flash memory requirements 221
- RAM used depends on IDE and use of PROGMEM and F() macro 221, 224, 279

Code skeletons

- Start application with code skeletons 136
- When creating functions, templates ensure items not forgotten 132

Code thrashing See Bugs

Collapsing code See Arduino IDE/AtmelStudio (editor)

Color coding See Arduino IDE/AtmelStudio (editor)

Command box See DOS box

Comments		
Awk/Perl comment symbol is pound sign #	294	
Comment closing curly braces and #endif	136	
Comment! Comment! Comment!	126	
Countless reasons for commenting code	6, 119, 125, 126	
Inline comments clarify logic, fewer bugs	108	
Program Documentation Framework - use Awk or Perl regexes to extract comments	108, 126, 233	
Comments See Program Documentation Framework		
Common sense See Good programming practices		
Compiler See Build toolchain, Arduino, AtmelStudio, Visual Studio/Visual Micro, PlatformIO(build)		
Conditional inclusions See Macros		
Conditions See Awk (rules)		
Constants		
#define or const variables to define constants; centralize their location	121, 133, 134	
Contiguous heap See Memory (heap contiguous)		
Crash See Bugs		
Critical available memory and critical reporting macros See Memory (monitoring functions), Macros		
Curly braces		
AtmelStudio does not collapse curly braces; Arduino IDE does	123	
Comment closing curly braces and #endif	121, 136	
Extra curly braces facilitate code collapsing	123	
Initialize using curly braces	121	
Curly braces See Arduino IDE and AtmelStudio (editor), Good programming practices		
D		
DataGroup Framework		
Beehive weighing system good example	254	
DataGroup class does the work: header defines characteristics, body contains data	154, 252	
Functions enable finding, inserting, removing records and defragment storage	155	
GetRecord/WriteRecord/RemoveRecord get/set/remove data	253, 254	
Index/sequential fixed/variable length data	154	
Data Handling Frameworks		
Bitfield Storage Framework stores data inside bitfields	248	
Class Data Framework accesses class data via enums	241	
DataGroup Framework supports index/sequential fixed/variable length data	252	
Data Packets Framework breaks data into chunks for serial transmission	244	
Event Storage Framework stores evens in		
contiguous byte arrays	255	
Format Driven float to byte Conversion Framework reduces data size	245	
Frameworks to handle data storage, data conversion, data compacting	140, 147	
Linked List Framework enables any class to store values inside linked lists	257	
Data Packets Framework		
Break up data into chunks/packets	150, 244	
Packet has header (start code, packet size, ID, sequencing info), body (data), tail (end code, checksum)	150	
Packet is broken up into header (start code, packet size, ID, sequencing info); body (data); tail (end code, checksum)	151	
Send/receive packets via serial ports	150	
Validity checks: data received and checksum on data	151	
Debugging hardware-based		
Arduino does not support hardware-based debugging; Arduino V2 does	111	
ATmega328P and ATmega2560 compatible with hardware debugging with debugWire and JTAG respectively	111	
ATmega328P-Xmini Uno pin compatible supported by AtmelStudio, success using it	6, 33, 111, 112	
ATmega328P-Xmini - watch variables to find glitches	6, 107, 108	
ATmega2560RFR2 Xplained Pro - specialized hardware for RF communications, not Arduino boards pin compatible, success using it	111, 113	
Atmel ICE - Not successful debugging Uno or Mega	33, 105, 111, 113	
AtmelStudio with specialized boards enable hardware-based debugging	105, 111	
Bugs categorized as glitches, code thrashing, misdoings	105	
Complementary tool relative to print-based debugging, requires special hardware and compatible IDE	110, 176	
Debugging hardware must be compatible with microcontroller architecture (AVR, ARM)	111	
Develop with Visual Studio 2022 + Visual Micro, hardware debug with AtmelStudio + Xplained boards	113	
Dragon Programmer, STK500/600 should enable fixing bad microcontroller using HVPP	115, 116	
Olimex - hardware-based debugger for ESP32	111	
PlatformIO supports Arduino, ESP32 hardware-based debugging, other	100	
Watch corrupted variable to pause execution and find glitch's cause	107, 108, 110	
Debugging print-based See Print-based Debugging		

Framework	..\ to backtrack one directory, .\ to proceed from current directory	227
Debugging serial		
Visual Micro enables serial debugging w/o dedicated hardware		77, 78, 116
debugWire See Debugging hardware-based Atmega328P		
default case in switch statement See Bugs, Good programming practices, Error Reporting Framework, Misdoings		
Defensive Programming		
Good programming practices, plan and implement procedures and frameworks, use professional tools		xv, 1
delete See Perl (arrays)		
Dependencies See Header files (.h)		
Development tools See Arduino, AtmelStudio, Visual Studio, Visual Micro, PlatformIO, VS Code, Code::Blocks, MPLAB		
Documentation templates See Program Documentation Framework		
DOS batch files		
Batch file example		229
Batch files enable multiple DOS commands		229
Displays output or redirects to file (> and >>)		229
ECHO prints a message in the DOS box		229
Launch Awk and Perl from batch files		229, 302
Manage batch file execution: START, TIMEOUT		228
May take parameters such as %1, %2, etc.		229
PAUSE interrupts processing, EXIT and ctrl-C closes DOS box		229, 230
:: undocumented comment for batch files		228
DOS box		
Awk invoked via DOS box - output screen or file		196
cmd.exe (DOS box) launched from screen or toolbar icon or run menu item		226
Copy/paste to/from other apps		230
Cryptic not found message		226, 227
DOS box in c:\Windows\System32\cmd.exe		226
Microsoft created DOS operating system for the IBM personal computer		225
Other details you ought to be familiar with		230
Prompt looks like this: D:\Dev\Atmel>		226
Recalcitrant app? TaskManager (ctrl+alt+del)		230
Text user interface (old Unix type console)		225
Windows DOS box for text interface programs		225
DOS commands		
basic set: CD, DIR, DEL, EXIT, MKDIR, COPY, XCOPY, PAUSE, ECHO, REM		228
CD command to navigate to other directories		227
Do HELP for DOS commands		226
PATH displays or sets environment variables		227
E		
Eagle software		
Electronic schematic and Gerber files; now in Fusion 360		15, 214
Editor C++ See Arduino, AtmelStudio, Visual Studio, VS Code, PlatformIO (editor), Notepad++, VS Code		
EEPROM (Electrically Erasable Programmable Read Only Memory) See Memory (pools)		
EGO trap		
Avoid proving to yourself that you are an absolute genius		136
Exploit C++ features sparingly		121, 136
Psychological factors extensively covered in companion book Pragmatic C++ Arduino Programming		136
END See Awk and Perl (build)		
enums		
class in enum declaration is scope specifier		134
enum lists begin with start, finish with end		135
enums better than #define macros		121, 134
enums synchronize multiple arrays		134, 149
Program Documentation Framework - enums are often an alternative to macros		146
Specifier defines size of individual enum items		134
Error handling See Validate data		
Error Reporting Framework		
enums synchronize events with array strings		174
ErrorEvt and ErrorList inherit from LinkedItem and LinkedList respectively		270
ErrorEvt class stores info on event and links them as linked list (ErrorList)		173, 269
Handles error logging and reporting		174
switch default case missing or empty		173, 279
Error reporting macros See Macros and Print-based Debugging Framework		
ESP32		
ESP32 chips are ARM devices		111
PlatformIO supports Arduino, ESP32 hardware-based debugging, other		100
Supported by Arduino IDE and PlatformIO, not by AtmelStudio		25, 91
Event-based programming See Arduino (build - loop function)		
Events See Memory Management Framework and Event Storage Framework		
Event Storage Framework		
256 events (8-bits), 244 event (7-bits)		160, 255
enum driven Get/Set functions get/set events		

packed in bytes	160, 255	reduces data transfer from 24 to 9 bytes	247
Masks used to manage individual events	256	Converts float values to packed bytes and vice versa, packing controlled by parameters, FDFBC class does all the work	152, 245
Predefined true/false conditions stored in 1 bit per event byte array, saves RAM	160, 255	float values stored in SensorFloatData[]	153
Specific classes could be created to handle distinct event categories	161	Parameters include packed byte size, decimals, divider, range, offset	152
Exception handling (C++)		fpermissive See -fpermissive (compiler options) (first page of index table)	
Pseudo Exception Handling Framework - alternative to C++'s exception handling	172	Fragmented heap See Memory (heap fragmented)	
try establishes landing point, throw returns back to landing point, catch handles the problem	172	Frameworks	
exit See Awk (built-in variables and functions)		Application independent tools designed to accomplish specific tasks	139
External tools See AtmelStudio (upload)		Do not reinvent the wheel - use frameworks	6
F		Frameworks source code: md_dsl.fr	321
False friends		Three types of frameworks: organizational, data handling, specialized	139, 231
Your C knowhow, a false friend when learning Awk	202	Frameworks See Organizational, Data handling, Specialized Frameworks	
Your C knowhow, a false friend when learning Perl	205, 211	Free IDEs	
FDFBC class See Format Driven float to byte Conversion Framework		Arduino, AtmelStudio, PlatformIO, Visual Studio, VS Code, Code::Blocks	103
Fields See Awk (general)		Freematics See Code::Blocks	
File handles See Perl (file handles)		Function call nesting See Memory (use)	
FILENAME See Awk (built-in variables and functions)		Function Creation Framework	
File organization		Complete function skeleton example	238
Aardvark() contains original setup code, key to interoperability	142	Document functions exhaustively	147
Class definitions in ClassSpecific.h/.cpp files	142	Function template to create function	146, 236
Do your testing in the ATest() function	143	Function level macros See Print-based debugging	
Entry files - Arduino IDE: .ino file; AtmelStudio/Visual Studio: Sketch.cpp; PlatformIO: main.cpp	142	Functions See Awk and Perl (variables and functions)	
Functions.h/.cpp and FunctionsSKL.h/.cpp contain general and application specific functions	142	Function template See Function Creation Framework	
Globals.h/.cpp contain #includes and global variables; macros in Macros.h file	142	Fused deposition modeling FDM	
HelperFunctions.h/.cpp: service functions	142	Cheapest most practical for 3D printing	217
#includes in Globals.h, pivot around which all header files (.h) get included	144	Deposit molten plastic via nozzle	217
Macros.h contains macro definitions	143	Each printer mfg. has own slicer tool	217
See Project Files Framework	142	Fusion 360	
Start development in Aardvark, called by setup, then expand into other files	142	Can import Tinkercad schematics	214
Flash memory See Memory (pools)		High end CAD design tool	215
Flow diagrams See Algorithm Test Framework		Includes Eagle software (electronic schematic and Gerber files)	214, 215
F() macro See Macros, Code size, Misdoings		G	
foreach See Perl (build)		Gerber files See Hardware setup; Eagle (electronics design)	
Format document See AtmelStudio (editor)		getline See Awk (built-in variables and functions)	
Format Driven float to byte Conversion Framework (FDFBC)		Gibberish See Bugs (avoiding them, causes, cures)	
Beehive weighing system sensor values, FDFBC		Github See Why choose Arduino	
		Glitches See Bugs (avoiding them, causes, cures)	
		Golden rules	

26 | Index table

Comment! Comment! Comment!	126	Breadboards enables rapid prototyping but has drawbacks (unreliable serial communications, spaghetti error prone wiring)	13, 14
Da Vinci: "Perfection lies in details, but perfection is not a detail"	130	Breadboards, prototype board, PCB, SMT	13
Do not fall into the EGO trap	121, 136	Eagle (now in Fusion 360) for the schematics and Gerber files	15
Do your homework, know your tools, update your C++ skills, do not reinvent the wheel	121, 137	PCB should have serial comm pins to interface board with computer	15
Good mental condition crucial to good work	121	Soldered prototype boards atop ATmega2560 board replace breadboards	13, 15
KISS principle - simple solutions often better than complex ones	119, 120, 130	Hashes See Perl (variables) identified by % sign	
The right tool for the right job crucial	5	Header files (.h)	
Validate function parameters and return values; never assume anything	120, 127, 172	Arduino.h, Macros.h included in Globals.h file, #include <Arduino.h> in Macros.h	144, 232
Good programming practices (GPP)		AtmelStudio mysteriously imports second set of header files into ArduinoCore directory	62
Adhere to the 'Golden rules' (see above)	120, 127	#ifndef prevents multiple .h file inclusions	231
A hobbyist mindset leads to extensive debugging time	xv	#include in Globals.h, pivot around which all header files (.h) get included	144
Be consistent, particularly when naming files, functions, etc.	119, 122	Respect header file dependencies (top-down) and beware of interdependencies	144, 232
Comment closing curly braces and #endif	121, 136	Heap See Memory use	
Common sense improves productivity	5, 6	High voltage reset (HVPP) See Debugging hardware-based	
Countless reasons for commenting code	119, 125	Holes See Memory (fragmented heap)	
Don't neglect validity checks and error handling - Check, Check, Check!	120, 127	I	
Do your homework: know your tools, update your C++ skills, do not reinvent the wheel	137	IDEs See Arduino, AtmelStudio, Visual Studio, Visual Micro, VS Code, PlatformIO, Code::Blocks, MPLAB	
Efficient programming starts with good programming practices	xv	IDEs See Which IDE to work with?	
enums more flexible than #define macros	134	Image See avrdude (.hex file)	
Exploit C++ features sparingly	121, 136	Import Arduino project See Interoperability	
Indentation, code collapsing not cosmetic	119, 123	Incremental programming See Think	
Initialize variables, whether local or global, preferably with curly braces	121, 135	index See Awk (strings)	
Monitor memory - prevent stack overflow and lack of heap space	120, 129	Input record separator See Awk and Perl (built-in variables)	
Organize your code as separate files, .ino file should be small	120, 128	In-system programmer See avrdude (ISP)	
Plan work offline, think before typing; thinking hardest thing to do	119, 125, 126	Intelligent C++ Editor See Build toolchain	
Start application with code skeletons	121, 136	Intellisense See AtmelStudio (editor)	
Use #define macros, const variables, enums instead of hard coding values	121, 133	Interdependencies See Header files (.h)	
Wrap-up means stay focused 'til the task is completed	120, 130	Interoperability	
Greediness See Regex (greediness)		Aardvark() contains original setup code, key to interoperability	20, 22, 142
Groupings See Regex (captures/groupings)		Arduino project's name is that of .ino file	27
H		AtmelStudio import hassle free	34, 37
Hardware-based debugging See Debugging hardware-based		Code::Blocks not suited for Arduino development, does not create project	103
Hardware See Build toolchain		Common set of files independent of IDE	17
Hardware setup		Create ATMEL_STUDIO, ARDUINO_IDE, PLATFORM_IO macros	19
ATmega2560 for development, smaller board for final release	13	Develop with Visual Studio 2022 + Visual Micro, hardware debug with AtmelStudio + Xplained	

boards	113	and Event Storage Framework	
How to establish interoperability	18	Logical statements See Awk (rules)	
PlatformIO/AtmelStudio/Visual Micro interoperability requires AtmelStudio one solution per project scenario	36	Logic testing See Algorithm Test Framework	
PlatformIO creates main.cpp with empty setup and loop	17, 99	longjmp See Pseudo Exception Handling Framework	
Seamlessly switch between AtmelStudio, Visual Studio, PlatformIO, Arduino IDE	17, 23, 34, 99	Look ahead and look behind See Regex (look ahead/look behind)	
setup and loop located in application entry file	142	loop See Arduino IDE (build)	
Interoperability See AtmelStudio (interoperability) and PlatformIO (interoperability)		M	
ISP (in-system programmer) See avrdude		Macros	
J		AtmelStudio collapses #ifdef; extra curly braces facilitate cross platform code collapsing	123
jmp_buf See Pseudo Exception Handling Framework		Bjarne Stroustrup suggests no macros	133
join See Perl (string)		Comment closing curly braces and #endif	121, 136
json files See VS Code .json files		Critical reporting macros triggered when resource runs low (voltage, memory...)	176
JTAG See Debugging (hardware-based ATmega2560)		Error reporting macros print highly visible text to describe errors	176, 177
K		Function entry and exit macros print details upon entering and exiting functions	176, 270
KISS principle See Good programming practices		Is const instead of #define better choice?	133
Know your tools See Good programming practices		Memory use macros inform on current available memory and heap fragmentation	176, 271
L		Milestone macros highlight program's location	176, 271
Largest possible allocation See Memory (fragmented heap)		Misdoings: Serial.print missing F() macro	109, 279
Laziness See Regex (greediness)		Parameterized name/value macro simplifies printing a parameter's value	275
length See Awk (strings), Awk (arrays)		Preprocessor does text/replace via #define and conditional inclusions via #ifdef	133
Libraries See Arduino IDE (build)		Print-based Debugging Framework defines macros hierarchy	176
Linear thinking See Think		Print-based debugging macros display total, contiguous, fragmented heap	177
Line of text See Built-in variables (Perl is \$ _ ; Awk is \$0)		main.cpp See PlatformIO application entry file	
Linked List Framework		main function See Arduino, AtmelStudio, Visual Studio, PlatformIO(build)	
Based on LinkedItem and LinkedList	162	Make See Build toolchain	
Individual items or entire list may be printed	257	Manage data See Data Handling Frameworks	
LinkedList::GetNext traverses linked list forward 'till desired item is attained	162	Manage files See Project Files Framework	
LinkedList class: GetFirst, GetLast...	162, 257	Map mode See AtmelStudio (editor)	
Linked list not constrained by item size and number of items; can grow and shrink	161	Masks See Bitfield Storage Framework and Event Storage Framework	
Memory management and error reporting	162	match See Awk (strings)	
Memory Management Framework logs allocations and releases in linked list	170	MaximumPossibleAlloc See Memory (heap fragmented)	
Linker See Build toolchain, Arduino, AtmelStudio, Visual Studio, PlatformIO(build)		Memory (heap contiguous)	
Link time optimization See Code size		Based on functions: TotalHeap, FragmentedHeap, ContiguousHeap, MaximumPossibleAlloc	168
Lint like inspect utility See PlatformIO (general), Misdoings (Perl program finds program flaws)		Contiguous heap between top of allocated memory and bottom of stack	168, 282
Literal characters See Regex (metacharacters)		Essential component of Memory Management	
Log events See Memory Management Framework			

28 | Index table

Framework	264	Bottom of stack found via programming trick	282
Function calls require contiguous memory space for stack frames	169, 171	Function nesting and recursion consume too much contiguous heap space	168, 171, 282
Function nesting and recursion consume contiguous heap space	171	Hard code function memory requirements to monitor stack needs	171
Memory (heap fragmented)		Memory allocation failure avoidable with memory monitoring functions	167
Based on functions: TotalHeap, FragmentedHeap, ContiguousHeap, MaximumPossibleAlloc	168	Stack frame structure and size	283
Essential component of Memory Management Framework	264	Stack overflow thrashes memory allocations, may go undetected a long time	129, 167
__flp (holes) and __brkval (no holes) address of start of heap	282	Memory (use)	
Fragmentation ratio: fragmented/total heap	284	Allocations add 2 bytes for allocation size	284
FragmentedHeap function monitors extent of memory holes	170, 284	Bottom of RAM (system RAM) used for globals, static, system variables	168
Has holes from random memory releases	168, 281	__flp (holes) and __brkval (no holes) address of start of heap	282
Largest possible allocation: either largest hole or contiguous memory	169, 283	malloc and new need how much memory to allocate	284
Small holes may collectively yield a deceptively large heap	284	Memory allocation failure avoidable with memory monitoring functions	120, 129, 167, 171, 264, 282
Memory (heap total)		Memory used as the application runs	168
Total heap = fragmented plus contiguous heap; reveals total available memory	169, 170, 283	Print-based debugging macros display total, contiguous, fragmented heap	177
Memory (leaks)		RAM memory and flash memory requirements	221
Could be a cause of lack of memory, depletes available memory slowly but surely, do not manifest themselves during testing phase	129	Total heap is fragmented plus contiguous	129, 282
Memory Management Framework discloses memory gluttons and memory leaks	167, 264	Understand memory structure to optimize memory use	129, 281
Memory Management Framework		Memory use macros See Macros	
Based on functions: TotalHeap, FragmentedHeap, ContiguousHeap, MaximumPossibleAlloc	169, 170	Mental condition See Think	
Enough contiguous heap? Hole large enough?	169	Metacharacters See Regex (metacharacters)	
Heap has two parts: fragmented and contiguous heap	168	MicrochipStudio See AtmelStudio (general)	
MemAllocEvt and MemAllocList - base classes for managing allocations	169, 264	Microcontrollers	
MemMgt - main class for handling memory management events	264, 265	Choosing microcontroller is compromise; why Arduino?	xv
Memory allocations verify heap availability and reveal memory leaks	6, 167, 171, 264	ESP32 chips are ARM devices	111
Memory Management Framework discloses memory gluttons and memory leaks	171	Microcontroller architectures: AVR, ARM, PIC	111
Memory use macros provide a snapshot of available memory	281	PIC (Programmable Intelligent Computer): both AVR and ARM chips	111
Overloaded new: memory management, verify allocation success, log events	171, 174, 264	Microsoft Disk Operating System See DOS box	
Memory (pools)		Milestone macros See Macros	
EEPROM writes limited to 100,000	154	Mindset See Good programming practices	
PROGMEM and F() macro reduce code size	221	Mind to keyboard See Think - incremental programming	
Memory (stack)		Misdoings	
		= instead of == in an if or while or == instead of = in an assignment	109, 277
		Dumb mistakes not detected by compiler; found by Perl FindMisdoings.pl program	6, 109, 277
		Regex missing F() macro in Serial.prints	109, 279
		switch default case missing or empty	109, 277, 279
		MPLAB	

May be overkill for Arduino dev	104	foreach	209, 307
N		Create array as subset of existing array, from a Perl list, or by tokenizing string	307, 309
Naming conventions See Good programming practices		Create individual array items on the fly	307
new (overloaded) See Operator overloading		Create multidimensional arrays via nested square brackets initializations	307, 310
next See Awk (built-in variables and functions) and Linked List Framework		Cycle through the entire array, via a for loop, foreach loop, while loop	308, 309
Notepad++		Getting dimension of subarray requires programming trick	311
Edit Awk and Perl programs - configure it for Perl: also useful for C++ editing	197, 207	Multidimensional arrays is cumbersome	310
O		Perl arrays are associative, can be mix of numeric (float) and string values	191, 301, 307
Object-oriented programming (OOP)		Perl arrays can be sorted via the sort function	309
Bottom-up design is fill in the details	xvi, 5	Perl provides three variable types: \$scalars, @arrays, %hashes	209, 307, 313
Design your system the way you think - outside-in, aka top-down	xvi, 5	Process file via a shift function on array	209
Olimex hardware debugger See Debugging hardware-based		push/pop add/remove last item in Perl array; shift/unshift does same on first item	309
Operator overloading		scalar gets number of items in one-dimensional array	308
AddHive += illustrates operator overloading	174	Perl (Awk vs. Perl)	
First create function to do the work	175	Cycle through file names array with foreach, process files with file handles	303
Overloaded new: memory management, verify allocation success, log events	171, 174, 264	Does not support parens-based captures - Perl does	191
Organizational Frameworks		Perl is better than Awk	190, 191, 210
Function Creation Framework - Use function creation template to create new functions	236	Process same set of files several times - Awk limited to processing set once only	192, 211
Program Documentation Framework - Document your code	232	Simulate Awk with Perl or how to transform Awk program to Perl	210, 301, 303
Project Files Framework - Organize your project's files	231	Perl (build)	
Three frameworks help structure your files and document source code	139, 141, 231	Convert \$fileHandle to array via <...>, cycle with foreach	209
Other IDEs See Code::Blocks and MPLAB		C's switch/case/default are named given/when/default in Perl	307
Outlining See AtmelStudio (editor)		Importing packages (use directive): strict, diagnostic, features 'switch'	205, 208, 302, 313
P		Lists enable doing tasks on multiple items in one line of code	192, 301, 306
Packages See Perl (build)		Pass by value and by reference	206, 311
Parameter default initializations See Avoiding bugs		Perl handles records and fields	205
Parameter validation See Golden rules		Perl provides three variable types: \$scalars, @arrays, %hashes	206
Pattern matching See Awk, Perl, and Regular expressions		Perl's program structure resembles C's	208
PCB (Printed Circuit Board) See Hardware setup; Eagle		Process files: foreach, while on \$FileHandle	209, 306
Peripheral thinking See Think		Program flow control mechanisms are supported, namely for, if, switch (given), etc.	306
Perl (arrays)		Scope resolution operator :: implements namespace concept	206, 313
+ sign overloaded to enable merging arrays	309	The text match operator =~ enables using a regex in a statement	206
Array indices must be integers - will convert non integer indices to integer	307		
Array @_ is mechanism to pass arguments to a function	312		
Convert \$fileHandle to array via <...>, cycle with			

Perl (built-in variables and functions)

- \$ARG or \$ _ is the line of text (record) 309, 312
- \$ARGV array contains current file name when reading from <> array 312
- \$INPUT_LINE_NUMBER, \$NR, \$. line number of file being processed 312
- \$INPUT_RECORD_SEPARATOR, or \$RS, or \$/ - default \n 307, 312
- \$OUTPUT_FIELD_SEPARATOR, or \$OFS, or \$, separates fields from each other 312
- \$OUTPUT_RECORD_SEPARATOR, or \$ \ - default newline 312
- \$PROGRAM_NAME, or \$0, contains the Perl program launched 205, 311
- \$SUPSEP or \$; - index separator 312
- @ARG or @_ : arguments to function 312
- @ARGV array: command line arguments 312
- English pragma: AND instead of &&, OR instead of || 311
- Input field separator - Perl does not seem to have one 312
- Perl supports the Awk like BEGIN and END 208
- qq{} function enables embedding double quotes inside of string without escapes 312
- split and qw// functions automatically tokenizes a line of text 312

Perl (command line - DOS box)

- Invoke Perl directly or use batch file 302
- MyPerlProgram.pl arg1 arg2... > Output.txt 302
- Perl command line file/directory in arg1, arg2, etc.; in @ARGV array 207, 302

Perl (file handles)

- \$filehandle created from file name string via the open statement 205, 209, 304
- Convert \$fileHandle to array via <...>, cycle with foreach 209, 305
- Files in read, write, or update mode 205, 304
- Process files: foreach, while on \$FileHandle 209

Perl (general)

- Edit Awk, Perl programs with Notepad++ or VS Code 84, 205, 207
- Handles text files and binary files 207
- Included in Unix, Linux, and MacOS but not Windows (download Strawberry Perl) 302
- Perl details: download, use it, simulate Awk, constructs strange from C/C++ perspective 301
- Perl is better than Awk 191
- Perl program creates program documentation 108
- Perl program looks for dumb mistakes: FindMisdoings.pl 6, 109
- Perl supports regex parens-based captures - Awk does not 191, 206, 207

- Powerful C-like programming language with regex support 180, 190, 205
- Why I started with Awk, then migrated to Perl 190
- Your C knowhow, a false friend when learning Perl 205, 211

Perl (strings)

- Perl arrays to strings with list function 309
- Perl string tokenization and concatenation functions: split, qw//, join 309, 312
- Perl supports string concatenations 305
- qq{} function enables embedding double quotes inside of string without using escapes 312
- String interpolation differentiates a variable's name from its value in print statements 306
- Supports literal strings (single quotes) and interpolated strings (double quotes) 211, 305, 306
- Supports regex text match operator =~ 313

Perl (variables and functions)

- BEGIN and END blocks executed at program start and finish 207
- Functions can pass by value or by reference 207
- Functions have a name and a body but do not have a parameter list 207, 211, 305
- Functions may be called with parameters - unique way of handling (via array @_) 207, 211, 305
- Perl types differentiated by special characters: \$scalars, @arrays, %hashes 209, 313

Perl vs. C

- Arrays are associative; index is integer based 211
- Perl code resembles C but there are notable differences 211
- Perl is a command line utility/interpreter 211
- Pointers not supported 211
- print statement items not necessarily inside parentheses 211
- Regular expressions supported 211
- # sign same as C++ comment - ignore up to end of line 211
- Special character identifies variable's type: \$ scalar, @ arrays, % hashes 211

Perl Web site

- Documentation on variables, functions, operators 317

permissive See -fpermissive (first page of index table)**PIC See Microcontrollers****Plain text matches See Regex (general)****Plan your work offline See Think****PlatformIO (build)**

- Build command details 223
- flto and -fpermissive flags included by default in PlatformIO project properties 221

Uses Arduino distribution's GNU C++ compiler	5	(.ino replacement)	21
PlatformIO (general)		Operate in single- or two-level directory structure; code in src_dir platformio.ini variable	21
Compatible with the Arduino project structure	94	PlatformIO Arduino project located in AtmelS-	
Creates main.cpp with empty setup and loop	95	tudio project base directory	18
Find/replace supports regular expressions	180	Seamlessly switch between AtmelStudio, PlatformIO, Arduino IDE, Visual Studio	17, 99
Free, professional grade, easy import of Arduino projects	23, 91	PlatformIO (platformio.ini file - project config.)	
IDEs: Arduino, AtmelStudio, Visual Studio, VS Code, PlatformIO, Visual Micro	77, 91	Com port defined in upload_port in platformio.ini file	98
IDE supports cross-platform, cross-architecture, multiple frameworks	23, 91	Define board with board = mega2560 or AtMega328 in platformio.ini	97
Installation - do it progressively, VS Code first	93	Operate in single- or two-level directory structure, code in subdirectory pointed to by src_dir	95
maker.pro presents interesting tutorial on PlatformIO	100	platformio.ini file contains sections/variables	98
Offers features beyond those offered by other IDEs	94	PlatformIO project is folder which contains platformio.ini file	95
Plugin for Microsoft VS Code and CLion	23, 91, 94	Project configuration in platformio.ini file	98
Project configuration in platformio.ini file	98	Source code location defined in src_dir in platformio.ini file	99
Provides facilities for lint like inspect utility	94	PlatformIO (project management)	
Supports hardware-based debugging wide range microcontrollers	94	Creates main.cpp with empty setup and loop	95
Supports multiple libraries packs	94	Open/Close project is Open/Close VS Code folder	95
Supports version control, configuration management, continuous integration	95	PlatformIO projects are VS Code folders	95
Web site (platformio.org) rich with info	100	Project configuration in platformio.ini file	98
PlatformIO (gotchas and caveats)		pop See Perl (arrays)	
Build uses files in the project's source code directory and subdirectories	102	Pragmatic C++ Arduino Programming book	
Contents of directory not updated after directory content changes	102	Explains why C++ can lead to excessive debugging time and how to reduce it	xv
File editing window can be unpinned, editor reverts to being simple notepad like	102	Preprocessor See Build toolchain, Macros	
Has its share of gotchas and caveats	101	Print-based Debugging Framework	
Import Arduino project location hard-coded	101	Avoiding bugs - ounce of prevention is worth pound of cure	106
New com port not recognized	102	Complementary tool relative to hardware-based debugging	110, 176
Platformio.ini file variable in wrong section not reported	101	Debug control macros turn debugging on/off global, file, and function level	176, 275
Removing file from project deletes it	102	Debugging macro level: global, file, function	110
The build/upload process works fine but no output from program	102	Display function specific data and errors	176
Uploading a program fails: access denied	101	Entry/exit macros provide parameter and return values, and inform on selected items	270, 274
PlatformIO (hardware-based debugging)		Macro categories: navigation, memory use, critical situation, data values	110, 176, 270
Requires special Arduino Nano 33 or MKR boards	100	Printing strategic data values, where the program is at	110
Serial debugging not supported	100	Printed Circuit Board (PCB) See Hardware setup	
Supports Arduino hardware-based debugging, ESP32, other	100	PROGMEM (read-only data in flash memory)	
PlatformIO (interoperability)		Web sites which describes PROGMEM	316
Creates main.cpp with empty setup and loop	17, 22, 99	Program Documentation Framework	
Interoperability between Arduino IDE, AtmelS-		Class and file documentation template store class and file info	233, 235
tudio, PlatformIO, Visual Micro	23		
main.cpp is PlatformIO's Arduino app entry file			

Comment! Comment! Comment!	126	R	
Create program documentation via Awk/Perl regex based comment extraction	126, 144, 233, 236	Radio transmission constraints See Format Driven float to byte Conversion Framework	
Example of comments extraction Aardvark	233	Random Access Memory (RAM) See Memory (use)	
Finding functions using regexes complicated	239	Redirection See DOS box	
Function template contains description, parameters, error checking, return values	235	Refactoring See AtmelStudio (editor)	
PROGRAM KEYWORD START and END basis of generalized info extraction mechanism	146	Regex (captures/groupings)	
Starts with inserting comments, organize them as templates (fill in the blanks)	232	Grouping (capture) extracts of parens pair content	185, 187
Who calls who? Who gets called by whom?	236	Grouping with ' ' (OR) lists alternatives	185
Program logic See Algorithm Test Framework		Regex (examples)	
Programming objectives		Extract a variable's name; define floating-point numbers	185
Enhance productivity, maintainability, robustness, compactness, and speed	1	Find text enclosed in square brackets gotcha, greediness issue	189
Monitoring code size important since premium on RAM although speed might be a requirement	1	Generate subset of a file	181
Project directory See Interoperability		Greedy vs. lazy search anything	185
ProjectDir See AtmelStudio (build)		Look ahead to find all Serial.??? which are not prints	186
Project Files Framework		Perl and Awk program use regexes to extract program documentation from source code	180
Aardvark() contains original setup code, key to interoperability	142	Regex missing F() macro in Serial.prints	279
Atest.h/.cpp contain ATest() to do testing	143	Regex to find = instead of == in an 'if', or == instead of = in an assignment	277
Avoid interdependencies between .h files	232	Regex used to list enums and #defines, find unused ones	181
ClassSpecific.h/.cpp files contain classes	143	Step-by-step example regex to find enums	186
Common sense dictates how to organize an application's source code	120, 128, 142, 231	Uncovers repeat words like the red red fox	181
Entry files: Arduino: .ino file; AtmelStudio and Visual Studio/Visual Micro: sketch.cpp; PlatformIO: main.cpp	142	ZIP code example gotcha or how laziness solved the problem	185, 188
Functions.h/.cpp and FunctionsSKL.h/.cpp contain general and application specific functions	143	Regex (general)	
Globals.h/.cpp contain #includes and global variables; macros in Macros.h file	143, 232	Anchor means look for something at the beginning ^ or at the end \$ of the line of text	183
HelperFunctions.h/.cpp: service functions	143	Character class [...] defines what characters to search for	182, 184
Macros.h contains macro definitions	143	Many tools contain a regex engine - AtmelStudio and PlatformIO find/replace, Perl, Awk, Word (wildcards)	34, 45, 50, 180
setup and loop located in application entry file	142	Match - return true if regex is successful, false otherwise	182
Prototype board See Hardware setup		Regex engine is a generalized intelligent wildcard protocol to find/replace text	6, 180
Pseudo Exception Handling Framework		Regexes use two types of characters: literal characters (what to search for) and metacharacters (regex operators)	182, 184
Alternative to C++'s exception handling based on setjmp and longjmp	172, 267	Regex segments combine individual patterns to form elaborate ones	183
Deep down the function call chain trigger longjmp back to setjmp landing point	267	Regex (greediness)	
EErrorID identifies longjmp	172	Find text enclosed in square brackets gotcha, greediness issue	189
longjmp (throw) - return landing point	172, 267	Greediness means grab all it can before relinquishing control (speed); laziness means grab	
setjmp (try) sets landing point	172, 267		
push See Perl (arrays)			
Q			
qw// and qq{} See Perl (string)			

one character only and let the regex continue (functionality)	183, 188	shift See Perl (arrays)	
Period/asterisk '.*?', period/plus '.*?' followed by question mark means lazy search anything	185	Simulate Awk See Perl (simulate Awk)	
ZIP code example gotcha or how laziness solved the problem	185, 188	Size specifier See enum	
Regex groupings See Regex (captures/groupings)		Sketch	
Regex (look ahead/behind)		Name used to refer to Arduino's .ino file (program entry file)	27
Condition search based on existing item	186	Sketch.cpp See AtmelStudio (interoperability)	
Look ahead to find all Serial.??? which are not prints	186	SMT (surface mount technique) See Hardware setup	
Regex (metacharacters)		Solution See AtmelStudio (file management)	
\$ sign anchors search to end of text	185	Specialized Frameworks	
Caret ^ used both as an anchor (beginning of text) and negation in character class	185	Algorithm Test Framework plan and test multiple execution paths	259
Context defines whether pattern matching character is metacharacter or literal	184	Class and Function Names Referencing Framework - IDs identify classes/functions	261
Escape (backslash \) transforms metacharacter into ordinary character	182, 184	Error Reporting Framework - log errors and warnings inside a linked list	269
Literal characters: characters searched for	184	Memory Management Framework informs on heap space, memory use, memory gluttons	264
? means preceding segment is optional, also means greedy or lazy when doing a search	184	Print-based Debugging Framework - selectively choose debugging print #defines	270
Metacharacters are letters or symbols which become pattern matching operators	184	Pseudo Exception Handling Framework - alternative to C++'s exception handling	267
OR symbol ' ' means one of several alternatives (choices) inside a parens group	185	Seven framework toolkits to accomplish sundry tasks	140, 162, 258
Quantifier {1-x} means repeat 1 to x times	184	Specialized macros See Macros	
Regular expressions (regex) See Regex		Spell-checker See AtmelStudio (editor)	
Return values validation See Golden rules		split See Perl (string)	
Rules See Awk (rules)		SRAM (RAM) See Memory (use)	
S		Stack frame	
say feature See Perl (build)		Determining stack frame size	283
scalar See Perl (variables) identified by \$ sign		Functions require contiguous heap space for stack frame	171
Scope See Awk (variables and functions)		Stack overflow See Memory use	
Scrollbars See AtmelStudio (editor)		Startup See avrdude (bootloader)	
Search and replace See AtmelStudio (find/replace)		strict See Perl (build)	
Segmentation faults See Arduino IDE (caveats) and AtmelStudio (caveats)		String interpolation See Perl strings	
Sensor data conversion See Format Driven float to byte Conversion Framework		substr See Awk (strings)	
Sensors		Surface Mount Technology SMT See Hardware setup	
Thousands of Arduino compatible sensors, boards, devices available	8	switch feature See Perl (build)	
Serial communications See Format Driven float to byte Conversion Framework		switch statement default case See Bugs, Good programming practices, Error Reporting Framework, Misdoings	
Serial terminal See Build toolchain, Arduino IDE (editor) and AtmelStudio (editor)		Syntax checking See AtmelStudio (editor)	
setjmp See Pseudo Exception Handling Framework		T	
setup See Aardvark, Interoperability, and Arduino IDE, AtmelStudio and PlatformIO (build)		Tables	
		Awk - short database example Awk program	286
		Bitfield Storage Framework - Table of bitfield based variables of Job class	156
		RAM memory and flash memory requirements	221
		Tab mode See AtmelStudio (editor)	

34 | Index table

TargetDir and TargetName See AtmelStudio (build)		VAssist See AtmelStudio editor and documentation	
Task creation See Code skeletons		Verbose See avrdude and AtmelStudio (import Arduino project)	
Task wrap-up phase See Wrap-up phase		Version control See PlatformIO (general)	
Text find/replace See Macros		Visibility See Awk (variables and functions)	
Text matches See Regex (general)		Visual Micro	
Text match operator See Perl (text match operator =~)		AtmelStudio Arduino compatible in two versions, without and with Visual Micro	33
Think		Develop applications for Arduino, ESP32, RaspberryPi, and others	77
Easy to sit at the computer and code	126	Develop with Visual Studio 2022 + Visual Micro, hardware debug with AtmelStudio + Xplained boards	74, 113
Good mental condition crucial to good work; do not rush, take breaks	132	Improve productivity by orders of magnitude	5
Good programming practices	125, 126	Interoperability between Arduino IDE, AtmelStudio, PlatformIO, Visual Micro, hassle free	23
incremental programming, i.e. mind to keyboard can be a costly time wise trap	1	Provides a serial monitor, compile/link/upload, serial debugging	77
Linear thinking means focus and follow a path; peripheral thinking means let your mind loose, let it dwell around a subject	131	Three identical Arduino plugins: for AtmelStudio and for Visual Studio (2019 and 2022)	35, 77
Planning offline before coding can save loads of time	5, 120	Visual Micro enables serial debugging w/o dedicated hardware	116
Thinking hardest thing to do - requires effort	119, 125	vMicro for AtmelStudio in top-level toolbar	77
Through-hole technology See Hardware setup		vMicro for Visual Studio project location defined from Arduino IDE preferences	78
throw, try, catch See Exception handling (C++) and Pseudo Exception Handling		vMicro for Visual Studio tucked away in Extensions of top-level toolbar	78
Tinkercad		Visual Studio	
Create visual breadboard wiring/schematics	214	AtmelStudio is Atmel specific Visual Studio	33
Schematics exported to Eagle (Fusion 360)	214	Choose between two versions: 2019 and 2022	74
Tokens See Awk (terminology)		Create Arduino apps via Arduino project template or via Visual Micro	73, 76
Top-down design See Object-oriented programming		Debugging - only Serial debugging via Visual Micro seems feasible	76
try, throw, catch See Exception handling (C++) and Pseudo Exception Handling		Develop with Visual Studio 2022 + Visual Micro, hardware debug with AtmelStudio + Xplained boards	74, 113
Type checking leniency See Bugs		Expand your horizons with Visual Studio (Python, Raspberry Pi...	74
U		Extensions for embedded development: Arduino, ESP32, RaspberryPi, ...	73
Undefined references See AtmelStudio (caveats)		IDEs: Arduino, AtmelStudio, Visual Studio, VS Code, PlatformIO, Visual Micro	73
unshift See Perl (arrays)		Improve productivity by orders of magnitude	5
Update your C++ skills See Good programming practices		Microsoft's flagship development tool - two versions, 2019 and 2022	73
Uploader See Build toolchain, avrdude, AtmelStudio build		Multilanguage: C++, C#, Python... and multi-platform: Windows, MacOS, Linux	73
use See Perl (build)		Out-of-the-box Arduino Project Template solution creates both .ino file and command line .exe program	75
V		Straightforward installation, two Arduino dev solutions	74
Validate data			
Apply good programming practices and adhere to Golden rules - check data, never assume anything; do error handling	120, 127		
Upon detecting error, undertake reporting and decide what to do next	172		
Variables See Awk and Perl (variables and functions)			
Variables See Initializations			

Visual Micro for Visual Studio identical with Visual Micro for AtmelStudio	76	Interoperability possible between Arduino IDE, AtmelStudio, Visual Micro, PlatformIO	23
Visual Studio Code		MPLAB may be overkill for Arduino development	104
See VS Code	79	PlatformIO - free, professional grade, easy import of Arduino projects	23, 91
Visual Studio Code See VS Code		Short list: Arduino IDE, AtmelStudio, Visual Studio, PlatformIO, VS Code	23
vMicro See Visual Micro and Debugging serial		Visual Micro - plugins for AtmelStudio and Visual Studio Arduino work	77
VS Code		Visual Studio - highly professional multi-language development tool	73
Awk extensions	88	VS Code - a foundation upon which many tools are created, including PlatformIO	79
C++ extension	87	while See Perl (arrays)	
Edit Awk, Perl programs with Notepad++ or VS Code	84	Workspace font size See Arduino IDE and AtmelStudio (editor)	
Extensions can be installed/uninstalled and enabled/disabled	80, 84	Wrap-up phase	
File, folder, workspace used for project management	80, 83	Adhere to good programming practices; stay focused 'til the task is completed	120, 130
Find/replace supports regular expressions	80	Code skeletons help ensure items not forgotten	132
IDEs: Arduino, AtmelStudio, Visual Studio, VS Code, PlatformIO, Visual Micro	79	Wundef See AtmelStudio (import Arduino project)	
Microsoft's programming foundation platform	79		
Multilanguage: there are extensions for C++, Python, Perl...	80		
Perl extensions	88		
Provides source control via Git repository	80		
Regular expressions extensions	88		
User interface appearance	82		
User space vs. workspace	80		
VS Code features	80		
VS Code caveats			
Arduino-cli failed to get installed	89		
Unable to remove folder from active folders list	89		
VS Code does not support the concept project content/disk content	89		
VS Code .json files			
Data files organized as text key-value pairs	88		
Value is string, number, Boolean, array, other json object	88		
W			
Warnings See Awk (command line) and Perl (build)			
Web site See Book's Web site			
Which IDE to work with?			
Arduino readily available, inexpensive, particularly easy to work with, required by other IDEs	7, 23		
AtmelStudio best: solid, professional tool, supports hardware-based debugging	23, 33		
Code::Blocks does not seem suited for Arduino development	103		
Develop with Visual Studio 2022 + Visual Micro, hardware debug with AtmelStudio + Xplained boards	74		
GitHub search on Arduino yields more than 100,000 results	7		

DEFENSIVE C++ ARDUINO PROGRAMMING

The right tools for Arduino
C++ programming

If you are constantly frustrated due to fighting against the tools you are working with, or feel that you are reinventing the wheel, this book is for you.

Why Arduino and C++ are an excellent choice for embedded software development.

Good programming practices help getting it right the first time around.

Which IDE should you work with? Arduino, AtmelStudio (MicrochipStudio), Visual Studio plus Visual Micro, Microsoft VS Code, PlatformIO, Code::Blocks, MPLab?

Visual Studio plus Visual Micro is, IMHO, the best tool to develop Arduino applications with. This book will guide you on using this tandem and details the process of importing Arduino sketches. The learning curve is short.

Print and hardware-based debugging are covered extensively. These are good reasons for you to develop with Visual Studio plus Visual Micro and use AtmelStudio to hardware debug with ATmega XMini boards.

The new Arduino IDE version 2 is presented. Is it ready to replace version 1.8.19?

Understand what lies under the IDEs' hoods. Behind the scenes, the IDEs drive the GNU toolchain (*preprocessor, compiler, linker, make*) - *avrdude* to upload code into your microcontroller.

Seventeen C++ frameworks save you from inadvertently reinventing the wheel. Full source code available in <https://md-dsl.fr>.

C++'s exception handling not being supported by the Arduino environment, use the Pseudo Exception Handling Framework, `setjmp/longjmp` based.

Awk, Perl, and regular expressions enable you to undertake tasks otherwise not feasible. A *Perl* program detects virtually undetectable silly mistakes (download link in <https://md-dsl.fr>).

Manage memory to determine its status (*contiguous and fragmented memory*). Anticipate *memory requirements* and discover *memory hogs* and *memory leaks*.

The companion book, *Pragmatic C++ Arduino Programming*, explains C/C++ fundamental mechanisms, the preprocessor, the PROGEM framework, the many gotchas C/C++ can throw at you, how to manage memory, and a lot more.

Download free open source licensed frameworks source code and *Awk, Perl, an regular expression* tidbits - download link in <https://md-dsl.fr>

Being defensive as you write code means being aware that your path is paved with danger. Discover how nasty C/C++ gotchas can get and how using professional grade tools can save you time and take the hassle out of developing with C/C++.



9 782958 562823

Kindle \$3.85

Paperback \$29.85

Hardcover \$34.85

Warthog drawing by the author