Computer Science Concepts and Definitions

An Overview of the Field

Ira Woodring

PUBLISHED BY PUBLISHER

IRAWOODRING.NET

First release, March 2021

Contents

Introduction

Wisdom.... comes not from age, but from education and learning.

–Anton Checkov

Learning a new field of study is hard. Throughout my career as an educator and a student I have identified several factors that hinder learning. I summarize these factors as such:

- Students and educators outside of the field of education or psychology aren't taught how learning occurs.
- Accomplished members in a field tend to - over time - trivialize what they consider "simple" concepts of the field.
- Practicioners of the field often use highly specialized language in desribing the concepts of the field.

The reasons for each of these factors arise rather organically. Students are already expected to master reading, writing and mathematics. We add to that science classes, history, and the spectrum of civics classes and there really isn't time to study learning theory. Which is sad, since a solid understanding of learning theory makes learning easier. The larger problem though, is that educators often aren't taught theories of learning either. Most Ph.D. programs concentrate very specifically on one (or occassionally two) very narrow areas of study. The expectation is that by becoming a master of your field you can teach it to others. Unfortunately (as every student who has ever taken a class knows), being accomplished in a specific area of study does not magically endow individuals the ability to teach well. Nor should it, as the study of how learning occurs is a science of its own.

The second roadblock to learning is that people that have accomplished a high level of skill in a particular field often forget the process by which they gained proficiency in the first place. Again, this happens rather naturally. Most topics that people learn are topics that build on prior concepts. I think of it as building a building - first you need a strong foundation, then solid framing, a good roof, etc. If any of these pieces are built poorly or missing the overall structure of the building will not be strong. The same process occurs with learning. Students first need to attach new and simpler concepts to their own existing knowledge. Then they must develop good understanding of core concepts of the field they are trying to learn, and attach those core concepts to the foundation of their existing knowledge. Again, if any of the information is missing or not understood fully the overall learning will be weak. Consider the topic of looping. Long-time programmers assume (correctly) that most people already understand the concept of a repeating process. But what they may forget in teaching is that students may not have a lot of experience with breaking a large problem down into smaller, repeatable parts, or building up a larger solution to a problem by repeating smaller atomic processes. To the long-time practicioner the need for looping is obvious. But to students learning to code there may be questions about how this concept fits in to the overall picture. I have had students ask questions such as, "Is this the only way to do it?", "Why are there multiple types of loops?", and "How do I know when I need to loop?". These questions point to a larger problem that must be addressed before teaching loops (or many other computing concepts) - "How do we use these machines to solve problems?"

Professionals who have gained competence in a field are at the point that the mental structures they have built are strong. This enables them to think at a higher level about their fields, and to connect concepts to other concepts more rapidly. In doing so, they no longer have to think about the connections between the older "simpler" (to them at this point) concepts that they already mastered. This can make communicating that information to acolytes of the field tougher. This problem greatly slows the knowledge transfer from more accomplished practicioners to those who are new to the field and may even cause confusion for them. Consider for instance the idea of a "variable" in programming. It was not uncommon during my computing education to hear people say something along the lines of "it is a variable, just like in math". However, other than the word being the same there is virtually no aspect of variables in mathematics that are the same for computing languages[1]! Variables in math are definitions; once defined by an equation the values cannot be changed;

$$3x+1=10$$

defines that

$$x=3$$

and it can be nothing else. Variables in computing are truly allowed to vary - not just in value but in address, scope, lifetime, type, name and size. Competent practicioners who use the flawed variable analogy are undoubtedly trying their best to simplify the concept and to help tie new knowledge to older knowledge. Unfortunately their choice of analogy, colored by their higher-level understanding of the nuances and caveats of variables in computing is a poor one, as it overly simplifies a complex topic that is foundational to the understanding of many other computing concepts such as type systems and memory management.

Finally, professionals in all fields use specialized language and acronyms to convey information. Computing is no different - we talk about type systems, programming paradigms, binding times, agile development and use acronyms like REST, RFC, GPU and more when discussing computing. Terms that are second nature to professionals - even what someone in the field a few years may consider obvious - may not be obvious to beginners. Even words such as "protocol" - a word that has meanings outside of the field - should be conveyed clearly for new practicioners. I learned this word when I overheard someone complaining that they had witnessed someone using "improper social protocol" at an awards dinner as a teenager so I looked it up. If I had not gone to that dinner [2], who knows when or if I would have encountered the word at all. And even if I had encountered it before doesn't mean I have the definition correct. Nor does it mean I understand what idea is being conveyed by using that word in the context of computing.

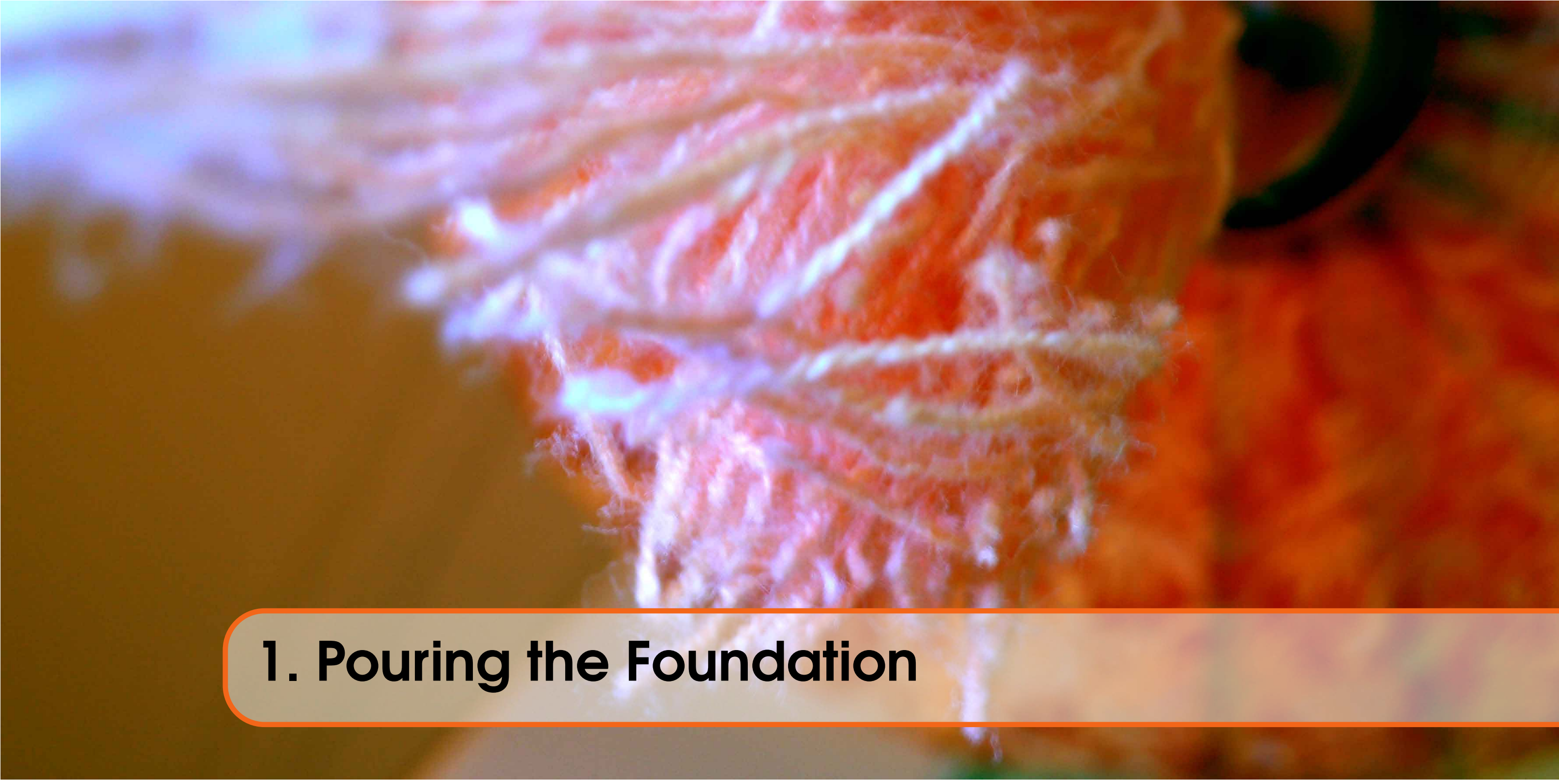
[1] I am speaking here of the (much more commonly first taught) imperative computing langauges, not functional ones. I leave the topic of whether functional paradigms in computing should be taught earlier in the computing curriculum up to stronger theorists than I!

[2] Or if I were not a native English speaker, or I had not read the word in a book, or any host of other reasons...

I am fortunate enough (and have studied the field long enough) to have "filled-in-the-blanks" in my computing education. Over time I began to see how concepts built upon one another and related to one another. I am frustrated though, that I still hear new students ask the same questions I and my peers asked when we were beginners. The aim of this text is to provide an additional resource to those learning the core concepts of the field with the hope of giving new students a better foundation for learning the field. I will explain what the concepts mean, but more importantly I will explain *why* they are the way they are. I hope you find it useful. If there are parts you feel are missing, confusing, or even wrong please let me know. After all, the learning process never really ends.

- Ira Woodring

Part One



1. Pouring the Foundation

1.1 What is a computer anyway?

I have read many definitions of computers over the years. Most definitions of a computer describe what a computer does (usually something about input, processing and output). All definitions I have read are vague or abstract, and building a strong understanding of a vague concept is like placing the foundation of a building on swampland. Here we define a computer more simply: **A computer is a machine**.

The connotation of machine is one that has mechanical parts, and indeed, early computers did have mechanical parts. Over time though, most of the mechanical pieces have been replaced by electrical ones. This in no way makes a computer less of a machine.

The futurist and famous science-fiction writer Sir Arthur C. Clarke wrote that, "Any sufficiently advanced technology is indistinguishable from magic." One might expect technology to become less "magical" after it has been around for several decades, but with computers the opposite seems to have happened. Computing technology continues to grow by adding layers to existing technology. Programming languages for instance, start with an instruction set at the processor level - but no one wants to (or should) program at such a low-level. So we develop higher-level languages that compile down to lower-level instructions. At this point we often find ourselves writing software in a language that runs on a virtual (software-based) machine which itself has been written in a high-level language that is compiled to low-level code to run on specific pieces of hardware (consider the Java VM for instance). The layers are so many and so thick at this point that the what happens at the lowest layers seems like magic again.

It is no wonder that so many students in Computer Science and related fields drop out. Students can't begin to build a foundation an ethereal concept.

An amazing mathematician named Alan Turing is largely responsible for what we consider a computer today. Before Turing's work there had been computing devices created for specific purposes, but Turing wanted to create a general purpose computing device. To do so, he needed to figure out some basic operations that machines could perform, and to figure out how to use those basic operations to solve mathematical problems. What he ultimately described was a machine with an incredibly long tape or paper that could be written to and read from. The tape would be divided into equally spaced sections, and in each section a '0' or a '1' could be written. A small head could read or write to the tape at whichever position was immediately below the head. On either side of the head were reels and the tape could be wound one way or the other. Each space on the tape could be numbered, and the head always knows where it is on the tape.

The machine would have a built-in set of operations. For instance, it might have an operation that tells the head to read the data in location 5 and to move to location 3 if that data were a '0' or move to location 12 if that data were a '1'. Turing was able to show that a machine with the correct set of basic operations would be able to solve a very large number of mathematical problems. Today's machines operate on the same principles. Instead of a long tape we use electrical memory called Random Access Memory (RAM). The read/write head is replaced by the computer's Central Processing Unit (CPU). Every processor has set of built in operations it knows how to perform called the Instruction Set. Just like cars, refrigerators, and all other machines computers may vary in the features they provide. While each processor will have the same basic problem solving capabilities, certain processors may be faster than others or have an extended set of instructions, just as some cars may have features others don't even though they all provide transportation from one location to another.

1.2 An example

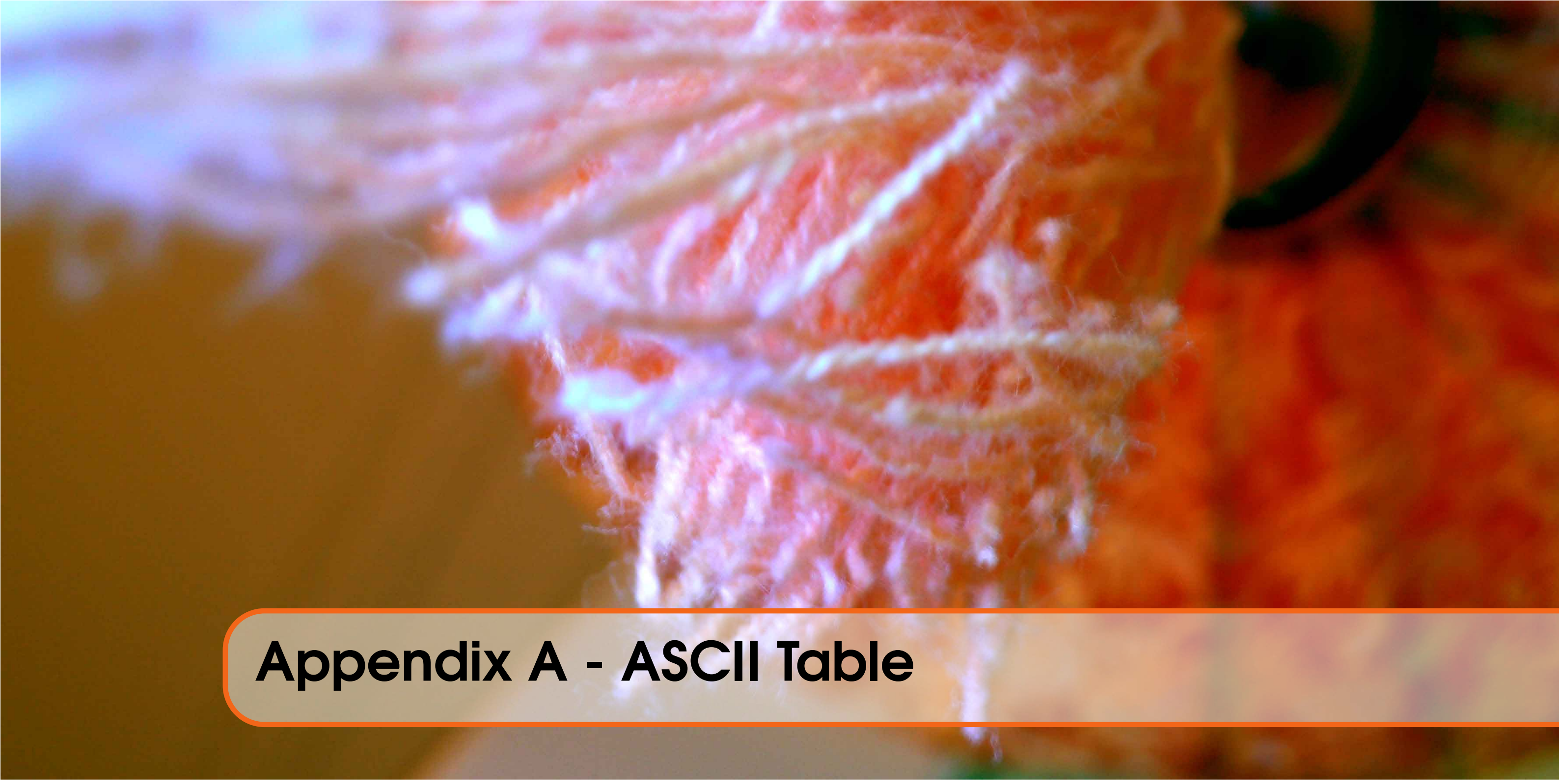
2. Communicating with Computers

The single biggest problem in communication is the illusion that it has taken place.

– George Bernard Shaw

II

End Matter



Appendix A - ASCII Table

The American Standard Code for Information Interchange (ASCII) table shows the standard values used to encode information in computing. Many newer students in the field mistakenly view the entries in the table as 'characters', but here we don't wish to use that term. The word 'character' implies printability [1], and a character is really a visual symbol. While many of the entries in the table may be represented by a character, there are many that are not. For instance, entry 0x07 represents the system bell and may cause the equipment being communicated with to emit a sound. Value 0x0A may cause a printer receiving that value to move the print head to the next line.

In a file, each of these entries could be represented by a single byte. Viewed in hexadecimal notation "HELLO" would then become

```
48 45 4C 4C 4F
```

[1] While some may consider moving a printhead to be "printing", by printable we mean to imply that ink has been applied to paper.

Decimal Value	Hex Value	Character	Note
0	0x00	NUL	Null character. Not printable.
1	0x01	SOH	Start of header. Not printable.
2	0x02	STX	Start of text. Not printable.
3	0x03	ETX	End of text. Not printable.
4	0x04	EOT	End of transmission. Not printable.
5	0x05	ENQ	Enquiry. Not printable.
6	0x06	ACK	Acknowledgement. Not printable.
7	0x07	BEL	Bell. Not printable.
8	0x08	BS	Backspace. Not printable.
9	0x09	HT	Horizontal Tab. Not printable.
10	0x0A	LF	Line Feed. Not printable.
11	0x0B	VT	Vertical Tab. Not printable.
12	0x0C	FF	Form Feed. Not printable.
13	0x0D	CR	Carriage Return. Not printable.
14	0x0E	SO	Shift Out. Not printable.
15	0x0F	SI	Shift In. Not printable.
16	0x10	DLE	Data Link Escape. Not printable.
17	0x11	DC1	Device Control 1. Not printable.
18	0x12	DC2	Device Control 2. Not printable.
19	0x13	DC3	Device Control 3. Not printable.
20	0x14	DC4	Device Control 4. Not printable.
21	0x15	NAK	Negative Acknowledgement. Not printable.
22	0x16	SYNC	Synchronous Idle. Not printable.
23	0x17	ETB	End of Transmission Block. Not printable.
24	0x18	CAN	Cancel. Not printable.
25	0x19	EM	End of Medium. Not printable.
26	0x1A	SUB	Substitute. Not printable.
27	0x1B	ESC	Escape. Not printable.
28	0x1C	FS	File separator. Not printable.
29	0x1D	GS	Group separator. Not printable.
30	0x1E	RS	Record Separator. Not printable.
31	0x1F	US	Unit Separator. Not printable.
32	0x20	Space	Space.
33	0x21	!	
34	0x22	"	
35	0x23	#	Octothorpe.
36	0x24	$	
37	0x25	%	
38	0x26	&	Ampersand.
39	0x27	'	
40	0x28	(	
41	0x29	)	
42	0x2A	*	Asterisk.
43	0x2B	+	
44	0x2C	,	
45	0x2D	-	
46	0x2E	.	
47	0x2F	/	

Decimal Value	Hex Value	Character	Note
48	0x30	0	
49	0x31	1	
50	0x32	2	
51	0x33	3	
52	0x34	4	
53	0x35	5	
54	0x36	6	
55	0x37	7	
56	0x38	8	
57	0x39	9	
58	0x3A	:	
59	0x3B	;	
60	0x3C	<	
61	0x3D	=	
62	0x3D	>	
63	0x3F	?	
64	0x40	@	
65	0x41	A	
66	0x42	B	
67	0x43	C	
68	0x44	D	
69	0x45	E	
70	0x46	F	
71	0x47	G	
72	0x48	H	
73	0x49	I	
74	0x4A	J	
75	0x4B	K	
76	0x4C	L	
77	0x4D	M	
78	0x4E	N	
79	0x4F	O	
80	0x50	P	
81	0x51	Q	
82	0x52	R	
83	0x53	S	
84	0x54	T	
85	0x55	U	
86	0x56	V	
87	0x57	W	
88	0x58	X	
89	0x59	Y	
90	0x5A	Z	
91	0x5B	[	
92	0x5C	\	
93	0x5D	]	
94	0x5E	^	
95	0x5F	_	

<table>
<tr><th>Decimal Value</th><th>Hex Value</th><th>Character</th><th>Note</th></tr>
<tr><td>96</td><td>0x60</td><td>‘</td><td></td></tr>
<tr><td>97</td><td>0x61</td><td>a</td><td></td></tr>
<tr><td>98</td><td>0x62</td><td>b</td><td></td></tr>
<tr><td>99</td><td>0x63</td><td>c</td><td></td></tr>
<tr><td>100</td><td>0x64</td><td>d</td><td></td></tr>
<tr><td>101</td><td>0x65</td><td>e</td><td></td></tr>
<tr><td>102</td><td>0x66</td><td>f</td><td></td></tr>
<tr><td>103</td><td>0x67</td><td>g</td><td></td></tr>
<tr><td>104</td><td>0x68</td><td>h</td><td></td></tr>
<tr><td>105</td><td>0x69</td><td>i</td><td></td></tr>
<tr><td>106</td><td>0x6A</td><td>j</td><td></td></tr>
<tr><td>107</td><td>0x6B</td><td>k</td><td></td></tr>
<tr><td>108</td><td>0x6C</td><td>l</td><td></td></tr>
<tr><td>109</td><td>0x6D</td><td>m</td><td></td></tr>
<tr><td>110</td><td>0x6E</td><td>n</td><td></td></tr>
<tr><td>111</td><td>0x6F</td><td>o</td><td></td></tr>
<tr><td>112</td><td>0x70</td><td>p</td><td></td></tr>
<tr><td>113</td><td>0x71</td><td>q</td><td></td></tr>
<tr><td>114</td><td>0x72</td><td>r</td><td></td></tr>
<tr><td>115</td><td>0x73</td><td>s</td><td></td></tr>
<tr><td>116</td><td>0x74</td><td>t</td><td></td></tr>
<tr><td>117</td><td>0x75</td><td>u</td><td></td></tr>
<tr><td>118</td><td>0x76</td><td>v</td><td></td></tr>
<tr><td>119</td><td>0x77</td><td>w</td><td></td></tr>
<tr><td>120</td><td>0x78</td><td>x</td><td></td></tr>
<tr><td>121</td><td>0x79</td><td>y</td><td></td></tr>
<tr><td>122</td><td>0x7A</td><td>z</td><td></td></tr>
<tr><td>123</td><td>0x7B</td><td>{</td><td></td></tr>
<tr><td>124</td><td>0x7C</td><td>|</td><td>Pipe.</td></tr>
<tr><td>125</td><td>0x7D</td><td>}</td><td></td></tr>
<tr><td>126</td><td>0x7E</td><td></td><td>Tilde.</td></tr>
<tr><td>127</td><td>0x7F</td><td>DEL</td><td>Delete.</td></tr>
</table>

Bibliography

virtual machine A program that emulates a computer. 23

C