No Code: Null Programs

Nick Montfort

December 2013 TROPE–13–03

Abstract

To continue the productive discussion of uninscribed artworks in Craig Dworkin's *No Medium,* this report discusses, in detail, those computer programs that have no code, and are thus empty or null. Several specific examples that have been offered in different contexts (the demoscene, obfuscated coding, a programming challenge, etc.) are analyzed. The concept of a null program is discussed with reference to null strings and files. This limit case of computing shows that both technical and cultural means of analysis are important to a complete understanding of programs – even in the unusual case that they lack code.

A technical report from

The Trope Tank
Massachusetts Institute of Technology
77 Massachusetts Ave, 14N-233
Cambridge, MA 02139 USA
http://trope-tank.mit.edu

Craig Dworkin's book *No Medium* contributes to our understanding of many forms of art, and to the concept of art itself, by surveying a wide range of uninscribed works, artworks in several different categories that are blank (such as Tom Friedman's *1,000 Hours of Staring*), erased (such as Robert Rauchenberg's *Erased de Kooning Drawing*), clear (such as almost all of Nam June Paik's *Zen for Film*), and silent (such as John Cage's *4' 33"*).

An argument that Dworkin presents (one that he references in the book's title) is that such works, although they may be made of materials, do not actually have a *medium* because these materials are not in any way inscribed. A medium conveys or transmits, but these are materials without anything to convey or transmit. *1,000 Hours of Staring*, a blank square of paper, is in the drawing collection of the MoMA: Why the drawing collection? Nothing is drawn upon it. It is culturally, curatorially, and institutionally treated like a drawing, but it consists of nothing but a drawing material, paper. Dworkin explains the conundrum:

> To ask the question concretely: Was the paper you are holding already a medium before it was brought together with the ink? Any object, it seems, could conceivably be inscribed in some way, and so the mere potential for inscription expands the notion of "medium" so broadly that it is no longer precise enough to designate or distinguish in a useful way. But for some prior inscription to be the defining factor of media raises other questions. If paper, for instance, is one of the media used for "recording or reproducing" my text here, is it still a medium in Friedman's *1,000 Hours of Staring,* when nothing has been recorded or reproduced? If not, we are left with a situation in which the material specifics of the paper ... are integral to the meaning of the work, but where the specific material (the sheet of paper) is not the work's medium. And if, on the contrary, we do recognize that blank sheet of paper as a medium, or if we define media on the basis of inscriptibility, we are faced with the question of how to know when mere materials have been identifiable media. A certain sense of medium is caught between impossible chronologies. (Dworkin 2013, p. 29)

The study undertaken in *No Medium* throws into even sharper relief what was already known and recognized generally, but sometimes not as acutely as it should be: that artworks cannot be meaningfully understood based on their content or inscription alone. Art *can,* however, be considered, analyzed, and discussed even if it has no content, because of important aspects of presentation, reception, and other sorts of context. In this discussion, I am to show that the same is true when it comes to computation: The use and cultural meaning of computer programs can be assessed even when they have no code. That means that looking beyond the code is essential in this case, and it suggests that it is important in others. This does not mean, however, that a discussion of this sort can be developed without technical awareness and analysis.

This report aims to extend the study of works with no medium to the digital, and specifically to the computational. Computer programs generally have code as their medium; this report looks at computer programs that are exceptional in that they have

no code. These have been called empty programs or *null programs.* The discussion here is of strings, sets, files, and programs that are null in the sense of being empty.

This report is meant to be a extension of the more detailed and wider-ranging book *No Medium* and an appropriate next step after the analysis of a one-line BASIC program I conducted with nine other authors in our book *10 PRINT CHR$(205.5+RND(1)); : GOTO 10.* After considering a specific program as an example of those that have only one line, and yet are meaningful, why not go on to analyze those of zero lines? Null programs are also simpler and even more trivial than programs such as the utility *yes* (Montfort 2012), and might have a similar potenial to yield insights about programming.

The Null String

A common data type in computing is the string, a sequence of characters that is often used to represent texts of different sorts. The *null string* is that string with zero characters in it. Generally strings are distinguished from one another only by their contents, by the data they encode. The string `"a"` is equal to the string `"a"` regardless of where it appears in a program, what variable it is assigned to, and so on. There are as many strings of length one as there are characters, and if there are *n* characters there are *n* × *n* ways to assemble a string of length two. On the other hand, there is only one string of length zero, because there is only one way to assemble the contents (nothing) into a string of zero length. Hence, it makes sense to use the phrase *the null string,* including the definite article.

Strings and other sequences, as well as sets, can be empty, lacking any elements. This is not true of every data type. There is nothing corresponding to "the null integer," although 0 is perfectly fine as a value for an integer. Only when zero or more elements are allowed can a particular value be empty.

In certain programming languages there is also a special value called null or NULL which is used to indicate that a pointer does not refer to a valid object. This is a related use of the term "null," but not the same sense. The null programs we are considering here are, like a null string or a null set, simply empty; they lack any code.

Although the computer science definition of strings provided here allows for empty ones, in practice, bureaucracies almost always enforce that either an entire field is optional (one may or may not have a middle name) or that the strings used to fill them are not null. Try to obtain a vanity license place that is blank and you will see this in action. In Illinois, for instance, the information on such plates specified that "Vanity plates contain up to 3 numbers or 1 to 7 letters only." In Pennsylvania, regulations state that "A personalized registration plate may contain a combination of up to seven letters and numbers." Both regulations, taken at face value, allow for a person to be issued a blank license plate. So, in either of these states, or any other, apply for a plate that contains nothing but 0 numbers (and thus is blank) and see what the bureaucracy will do. One's choice of letters and numbers is supposedly free, but, on the other hand, the form that indicates this choice must be filled out to be accepted and processed.

Based on the existence of a unique null string, it is reasonable to guess that it would be sensible to refer to *the null program* as well. However, in the category of music and

sound work, there are different silent pieces; there are even different pieces by the same composer, John Cage, which can result in silence of the same duration. The famous *4′ 33″* is specified to be silent and four minutes and thirty-three seconds long, while Cage's *0′ 00″* (1962) can be of any duration, including 4:33, and can involve the performer doing anything, including remaining silent. Cage's *Tacet* (1960) is for one or more instrumentalists and can last for any amount of time, too, so a performance of four minutes and thirty-three seconds of silence could be of any one of these three pieces (Dworkin 2013, p. 145). For somewhat analogous reasons, there are also different null programs, different programs with no code.

Zero-Byte Files

Programs are most typically (although not always) stored on a computer as files; files, too, can be of zero or more bytes. Without undertaking an exhaustive examination of zero-byte files, it is important to note that there can be many different zero-byte files, and that such files are formally valid for certain file types but not for others. While zero-length strings are all the same and other sequences of length zero are all the same, this is not true for the shortest possible files.

Zero-byte files are written to disk for several reasons. In some cases, a program opens a file for writing but then writes nothing. In some cases, this is due to an error, but in other cases, nothing (the absence of any data) may be the correct file contents and may be meaningful. For instance, a program may start a log file to monitor some system activity, such as when a job is sent to the printer. If there have not yet been any jobs sent to the printer, it would be appropriate for the log file to be empty. Inspecting this zero-byte file could provide useful information: Nothing has been printed since logging began.

In other cases, a file is intentionally placed in a directory to serve as a signal of some sort. It might be a lock, indicating that another file is currently being accessed. (If all processes attempt to create this lock file when starting access, and if they only proceed if the file is not already present and the file can be created, it's possible to guarantee that only one process has access at a time.) Such a file can signal a number of other things, including that the installation of some software is complete or that a file transfer has started. Sometimes a lock file can get left over by mistake, but these files, too, are usually useful.

In Unix-like systems such as GNU/Linux and Mac OS X, one can create a zero-byte file easily using the command `touch` and the file name. This command can be used, for instance, to test whether one has write access to a particular directory.

Unlike strings, files are not defined by their contents alone. They also have, for instance, a file name and a location in the file system, a creation time, a modification time, and information about permissions. Some file systems associate other metadata with files.

For all of these reasons, zero-byte files are not meaningless and are not worthless. It is true that they occupy space — the metadata must be stored, so they do occupy room on disk; along these lines, there is generally a minimum block size that even zero-byte files

will occupy. Yes, certain zero-byte files can be removed without harm, but it is a mistake to try to delete them wholesale from one's system. Backup programs are designed to back up these files and mirroring and synchronization programs are designed to transmit them to replicas and keep them in sync. Such programs sometimes need to be specially coded to handle the case of zero-byte files, since a program cannot rely on any contents being transmitted as these files are sent. For these reasons, it is generally important to test on zero-byte files when developing backup and synchronization programs.

The source code to programs in most languages is represented in a plain text file, usually in ASCII, although other character sets are supported by some languages. Text files do not store any additional format information as file data, that is, as part of their contents, which means it is legitimate to have a text file that is zero bytes in length. The same cannot be said for other formats. Image and sound formats are required to have headers if they are valid files, even if they contain as little data as is possible. For instance, files of the type GIF (Graphics Interchange Format), version 89a, are specified to begin with a header containing the six bytes "GIF89a" and to have other required blocks, such as the logical screen descriptor which specifies the image width and height.

Some executable computer program formats do not allow for zero-byte files. The Commodore 64's PRG format is a simple one, but it begins with two bytes that indicate where the rest of the file is to loaded into memory. A PRG file cannot be blank, as it must specify the destination location in memory. One could argue that a two-byte PRG has no code, since it consists only of the address to which the (empty) program, the lack of code, should be loaded. That may be, but the corresponding file is certainly not a zero-byte file.

These aspects of files are important to the current discussion because the null programs discussed in this report are not only null programs, but also zero-byte files. In the cases considered here, null programs have file system metadata, exist in the context of a platform, and have other information associated with them in the form of additional metadata and/or paratexts.

zerobytes: The Shortest Demo

The demoscene is a community of practice, most active in Northern Europe, that produces computer programs (demos) to computationally generate audiovisual spectacles. These demos are deterministic, in that they generate the same sort of "music video" each time. Unlike games, they are almost always non-interactive — accepting no input from users during execution. There are other demoscene activities, such as the creation of graphics and music and even "wild" entries that take other forms, but the core activity is developing demos, including some which are of very limited file size. The demoscene emerged from the practice of cracking (or removing the copy protection from) software, which allowed programmers to add text, title screens, and eventually small audiovisual programs to the beginning of disks that had been altered in this way.

One common restriction imposed on demos is the 64 KB limit, which makes producing

Windows demos a challenge and inclines programmers to generate 3D objects and 2D patterns from parameters rather than including elaborate 3D models or large bitmaps. Such demos, and ones that are smaller, are called "intros" in reference to the graphical introductions that were added to cracked software. More restrictive still is the 4 KB demo or intro, which, when created for Windows with OpenGL, can still manage to produce stunning landscape imagery, as demonstrated by the 2009 demo *elevated* by the groups rgba and TBC. This demo, in only about a page of machine language code, shows a simulated landscape, virtual camera moving over it, with a song playing and club-like lights kicking in at one point. There are smaller demos/intros for different platforms, probing the lower limits of program size.

Demos are typically shown at demoparties, where the attendees vote on them. Typically they are uploaded to the pouet.net site, where they can be voted up or down by individuals online.

Since the demoscene is concerned with how small programs can be, one might wish to know what the smallest demo is; it would be impressive to produce such a demo. A program that produces an audiovisual display of some sort is a requirement, in this context; a program that does nothing visible or audible, or that simply outputs a message such as "Hello world," would not obviously qualify as a demo.

In September 2005, optimus uploaded a demo called *zerobytes* to pouet.net. As suggested by the title, this is a zero-byte demo, declared to be for Windows. When one runs it, or attempts to run it, it produces the following audiovisual effect:

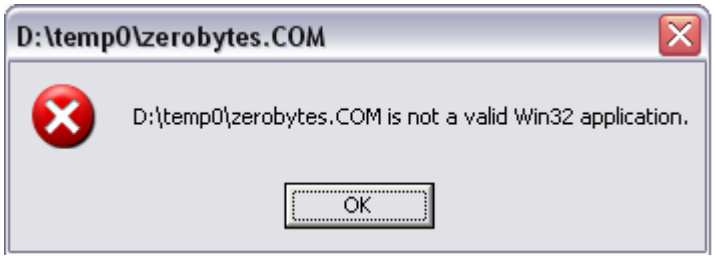

Figure 1. *zerobytes* "executing" in Windows.

The zipfile that optimus uploaded contains an assembly-language source code file zerobytes.ASM, as if to show that it is an assembled machine-language program. That file simply contains "org 100h," indicating that any further instructions — of which there are none — will be loaded at the address 100h. It also contains zerobytes.COM, the executable, which has no data, and zerobytes.txt, a file with no data.

Any invalid executable that one attempts to execute in Windows will produce a dialog box of this sort. The shortest such invalid file is an empty one. So, while the source code in this case appears to be valid and to have been assembled into the final demo, the demo that resulted is not (according to Windows) a valid application.

As of this writing, *zerobytes* is has received as of late 2013 slightly more positive ratings (thumbs up) than negative ratings (thumbs down) — 58 vs. 56. Code quality and impressive programming feats are central concerns for this community. But, those in the demoscene also appreciate humor and an awareness of the specifics of different computational platforms, some of things they may have detected in this code-free

demo. Perhaps in the demoscene, where overstepping the stated boundaries is valorized, people are more willing to admit that if it acts like a program, a message claiming that it isn't a program should be ignored.

smr.c/smr: The Shortest Quine

The term *quine* as it has come to be used in computing was coined by Douglas Hofstadter; it refers to programs that, when executed, produce their own code as output. The term is a reference to Willard Quine, whose discussion of self-reference was significant in 20th century philosophy. The following is a short quine in Python that was written by Greg Stein:

```
s = 's = %r\nprint s %% s'
print s % s
```

Quines are seldom straightforward. In this case, the `print` statement is what causes output to be displayed at all. Having the statement `print s % s`, rather than just `print s`, formats the output string, wrapping it in `s = '` on the left and `'` on the right, which is part of what allowed the output to match the original code exactly. There is much more to say about how this works and why it was composed as it was, but a full investigation of this quine could begin in this way.

Since programmers are interested in the cleverness of quines, the question has naturally arisen as to what the shortest quine is. It would be interesting to know what the shortest one is in a particular programming language and which of the general-purpose languages had the shortest quine. It is possible, of course, to define a particular programming language that is capable of general computation but in which the program 'Q' has the effect of printing the letter 'Q'. In this case the existence of this quine would not be so clever, because it was simply programmed into the language. It is worth nothing that in the esoteric language Homespring, designed to amuse fellow programmers, a null program produces the following output: "In Homespring, the null program is not a quine." (Neeman and Binder 2005, p. 4) It's generally of more interest to programmers to ask about the shortest quine in an existing language that wasn't made for quining, one in which quines are built out of general-purpose code.

A classic programming language for which this question would certainly be interesting is C, the language of the Unix operating system. C is a compiled language, so the shortest program would be a text file that compiles to an executable that, when run, produces the text of the program.

Using a liberal definition of "compile," Szymon Rusinkiewicz submitted a null source code file and instructions for the program `make` (used to compile C programs) to the 1994 IOCCC (International Obfuscated C Code Contest). The idea encoded in the makefile was for `make` to simply copy the null program so that it became the "executable," setting this file's permissions so that it could be run. With this done, the blank program could be made into a blank executable that, when run, would produce ... nothing.

While *zerobytes* fetaures valid source code and an executable that is questionable at best, this quine is the other way around: The product seems to be a valid executable, and runs without an error message being produced, but the source code, *smr.c,* will not

compile on all systems and a work-around is required in the makefile.

This program won the "Worst Abuse of the Rules" award, which seems to acknowledge that it is a program as the rules defined one, although this status is problematic. The judges' comments were as follows:

> Nearly every year, one or more people would submit what they claimed was the world's smallest self reproducing program. While the sizes of these submissions varied, a quick glance would reveal that they were too big, until this entry came along.
>
> While strictly speaking, smr.c is not a valid C program, it is not an invalid C program either! Some C compilers will compile an empty file into a program that does nothing. But even if your compiler can't, the build instructions supplied with this entry will produce an executable file. On most systems, the stdout from the executable will exactly match original source.
>
> In the future, the contest rules will specify a minimum size that is one character larger than this entry, forever eliminating this sort of program from contest. After all, how many variations can one make on this entry? :-)

The final smiley, of course, refers to the fact that (neglecting metadata such as the file name) there is only one C program of length zero. Variations, from this perspective, are not really possible.

What is a Computer Program?

In their famous book on programming, Abelson and Sussman characterize programs in terms of computational processes, noting that "The programs we use to conjure processes are like a sorcerer's spells." (Abelson, Sussman, and Sussman 1996, p. 1) Given this wizardly definition, could the spell be — nothing?

The strange limit case of no code — an empty file — does raise the question of what exactly a computer program is as we look to the lower limit of code. It also asks whether an empty file can just "accidentally" be a computer program, or whether such a code-free program falls into theoretical definitions of computer programs that declare it valid or invalid.

According to the U.S. Copyright Act as amended in 1980, and as declared on Circular 61 from the United States Copyright Office, "A 'computer program' is a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result."

This establishes the legal definition of a computer program, although it is a problematic definition from a science or computer science standpoint. For one thing, a "set" is, mathematically, an unordered collection of entities, each of which can occur at most once. {*Wallet*, *Keys*} is the same set as {*Keys*, *Wallet*}; it does not matter in what order the elements are written down. In a machine-language computer program (to restrict our consideration to these programs at this time), the order of the instructions is of course highly significant. Also, an instruction may occur more than once and this may be significant; while *multisets* allow for multiple instances of an element, ordinary sets do not. It would be better to say that programs, at least ones of this sort, are "sequences" rather than "sets."

In this view of programs as "sequences," however, it is still reasonable to think that a null program can exist. The same would actually be true even if we did view programs as sets; A set can be empty and a sequence can be of length zero. So, an empty file is, intuitively at least, a computer program according to U. S. law, as long as it exists "in order to bring about a certain result." This is a view connected to an *imperative* concept of programming, in which programs consist of instructions. This idea corresponds well to programming in assembly language and in C. It is not the only view, however. Without attempting a comprehensive survey of programming paradigms, it is useful to discuss some of the most well-known and influential ones.

Object-oriented programming developed from the imperative model and defined particular ways in which data and code were to be encapsulated together. This paradigm still involves defining a flow of control and particular instructions, as does imperative programming. In "pure" object-oriented languages, everything is implemented in every way as an object, even pieces of data such as integers or characters. Other languages that support this type of programming, such as Python and Java, have certain primitive data types that are not implemented in every way as objects.

In the abstract, it is not clear that OOP systems allow null programs. In Featherweight Java, proposed as a minimal core calculus that models the type system of Java, "a program consists of a collection of class definitions plus a term to be evaluated, corresponding to the body of the main method in full Java." (Pierce 2002, p. 249) A null program would be allowed, then, by this theoretical definition of an abstract version of Java, if the collection of class definitions is allowed to be empty *and* the term can be empty as well.

There is also the functional view of programs in which they transform inputs to outputs, a view usually taken by programmers who use Lisp, Haskell, and other languages that support a functional idiom. In this case particular instructions and the order in which they are executed are not important to defining programs. A Lisp program, for instance, consists of a sequence of expressions, specifically symbolic expressions or s-expressions, each of which produces a value when evaluated.

Another programming idiom is that of logic programming, in which the code defines facts and rules. Prolog was developed to support this type of programming. The execution of a Prolog program corresponds to evaluating a particular query to determine its value. Because logic programming involves declaring facts and rules rather than issuing commands or instructions, it is one of several types of *declarative* programming.

In languages where a (nonempty) query is needed, or where a "term to be evaluated" that is nonempty is needed, or where there must be at least one expression to be evaluated, a program cannot be empty. But if we consider that the query (and facts and rules) can be nothing, that the term (and the collection of class definitions) can be empty, and that there can be any number of expressions, including zero, then null programs can be admitted without any trouble. In theoretical definitions of computer programs, whether the definition makes it clear that null programs are allowed or not is, at least, one test of how complete and formal the definition of a program is.

There is also a practical consequence: If an empty file does meet the formal definition of a computer program, the compiler or interpreter should process it successfully. If it does not, the compiler or interpreter should produce an appropriate error message.

file.wc: A "Program" (or not) in wc

A somewhat quine-like programming challenge was presented on StackExchange in 2011: "write a program to print the sum of the ASCII codes of the characters of the program itself ... Program to print the lowest number wins." One user responded with a file that worked with the GNU word count utility, `wc`. With the `-c` argument this program prints the number of characters in a file. If run on an empty file, `wc -c file.wc` would print 0. Since there are no characters in an empty file, and thus no ASCII codes, it seems evident that `wc` supplies the correct answer in this case.

This "program" is an interesting idea. The main problem is not the questionable validity of the source code, as with the quine, or the questionable status of the executable, as with the demo. It is that `wc` is not a programming language, but a utility that counts characters, words, and lines. `wc` is neither designed for nor capable of general-purpose computation. There is no view of programming in which the empty file, having its characters counted, is the limit case of a program.

One could further argue that the specification included the following prohibition: "You are not allowed to open any file ..." and yet `wc` clearly opens the file specified. If it were running it as a program, that would be allowed, but the file here is opened not for the purpose of executing it but for reading it and reporting on it. The post about this "program" earned a check-mark on the thread to which it was submitted, but what this contribution illustrates most clearly is that not every zero-byte file can qualify as a program, even when people would like such a file to be considered as one.

Null Programs across Languages

So far, there has been no clear case of a null program that is unequivocally accepted as a program. The first item we considered is not a valid application; the second was judged to be an abuse of the rules of the programming contest, and the last instance doesn't involve executing any program except the utility `wc`, which then just counts the characters in a file rather than treating that file like a program. It seems that the first two of these three were accepted, at least in part and in good humor, as computer programs by the communities to which they were offered. However, it's not necessary to establish this firmly to show that null programs exist. There are dozens of others online.

On the *Rosetta Code* wiki, contributors offer programs or code snippets in many languages, in response to certain tasks. All of the code on a particular page is supposed to do the same thing, allowing visitors to the site to see how the same tasks are accomplished in different programming languages. On one of these pages, "the goal is to create the simplest possible program that is still considered 'correct.'"

More than 150 programming languages are represented on this page, many of them by the null program. In some cases, a class must be defined or a keyword such as `end` (in SNOBOL4, for example) is required. The exercise of writing the shortest possible program in a language may not seem to require a profound engagement with

computing, but it can expose what is required of all programs in that language.

Practically speaking, if the language of interest is contemporary and freely available, it is easy to try to run an empty file in that environment and see what will run and what won't. In Python, Ruby, Common Lisp, and shell scripting environments, a null file was seen to work perfectly — doing nothing, causing no error. Compiling an empty file is possible in Java, but trying to run it produces a message of the form "Error: Could not find or load main class ..." because the filename must correspond to a class.

The Abundance of Null Programs

The IOCCC judges asked of the zero-byte entry that they received, "After all, how many variations can one make on this entry? :-)" The general answer about programs, beyond the IOCCC, is, actually, very very many. Although most of the discussion was devoted to two "named" zero-byte programs — a demo presented with the demoscene and a C program offered for the IOCCC — we also found an amusing message printed by a null Homespring program and dozens of other null programs documented online. These certainly suffice to show that when all contexts are considered, there is indeed more than one null program.

Consider how this conclusion was arrived at: In the most abstract mathematical realm, we saw that there was only one null string. The judges of the IOCCC chuckled at the possibility that there could be more than one null program submitted to their contest. Because of how this coding contest is specified, they may have been right to do so. However, we find that there are many zero-byte programs for different platforms, with different names, intended to be executed in different ways, and for the consideration of different communities. Some may be offered as demos and quines; why not others as answers to homework assignments or as digital media artworks? There are technical reasons that allow more than one zero-byte program to exist, but those are only part of the story.

If the same zero-byte files is used as a program in many different contexts, it is true that is might be considered a single null program that has been put to various use. Those who seek to write a single text that is valid program in many programming languages (a practice called polyglot programming, and introduced well in Wikipedia) seem to think of their text, which works in different programming langauge contexts, as "a computer program." So, if one takes a single null file and tries to make it work as a program in various ways, one might be testing a single null program. But null programs are not all composed in this way; they are not all considered as possible programs that might be compiled, interpreted, or executed by running the same file. And pratically speaking, in terms of how programs are received and discussed, a null program devised as a quine is not the same null program devised as a demo, and so on.

To show that there can be many null programs, it was necessary to join technical analysis (of strings, sequences, sets, imperative programs that are sequences of instructions, different platforms, different programming paradigms, the definition of and means of storing metadata, and so on) with cultural inquiry (based directly on that done by Craig Dworkin in *No Medium,* and including how three pieces can result in identical silent performances, how paratexts and cultural contexts operate, how communities of practice factor into the creation and reception of artworks, and so on).

Even when a program has no content, literally no code at all, there are aspects of it that remain to be understood. Scholars who eschew the technical aspects of programs, ignoring metadata and the equivalence of null strings, will find it impossible to explain the meaning of such null programs (and other programs). Computer scientists who seek to explain real, observed null programs (such as *zerobytes* and *smr*) in purely abstract terms, without referring to culture and communities of practice with their diverse values, will also never succeed at a complete explanation of null programs; the same can be said for non-null programs that also have these important aspects.

The programs considered in this report are, by any code-based definition, the absolutely simplest of computer programs. Some of them were devised as jokes, but other null programs and null files have clear uses. Their lack of code shows them to be degenerate programs, absolutely trivial ones. Even this simplest example of computation, then, demands both technical and cultural engagement. If both types are analysis are needed when there is no code to consider, it seems that those studying computational media more generally should definitely continue to explore and learn in both realms.

Works Cited

Abelson, Harold and Gerald Jay Sussman with Julie Sussman. *Structure and Interpretation of Computer Programs* 2nd Edition. MIT Press: Cambridge, Mass. and London. 1996.

Alexandru. "Write a program to print the sum of the ascii codes of the program." On "Programming Puzzles & Code Golf," *StackExchange.* June 21, 2011, 17:33. Thread last updated May 25, 2012 at 23:08. <http://codegolf.stackexchange.com/questions/2926/write-a-program-to-print-the-sum-of-the-ascii-codes-of-the-program/2946>

Dworkin, Craig. *No Medium.* MIT Press: Cambridge, Mass. and London. 2013.

Montfort, Nick. "The Trivial Program 'yes.'" Technical Report, The Trope Tank, MIT. TROPE-12-01. January 2012.

Montfort, Nick, Patsy Baudoin, John Bell, Ian Bogost, Jeremy Douglass, Mark C. Marino, Michael Mateas, Casey Reas, Mark Sample, and Noah Vawter. *10 PRINT CHR$(205.5+RND(1)); : GOTO 10.* MIT Press: Cambridge, Mass. and London. 2013.

Neeman, Joe and Jeff Binder. "Homespring Proposed Language Standard." November 24, 2005. <http://bunny.xeny.net/linked/Homespring-Proposed-Language-Standard.pdf>

optimus. *zerobytes.* On *Pouet.net.* September 2005. <http://www.pouet.net/prod.php?which=18860>

Pierce, Benjamin C. *Types and Programming Languages.* MIT Press: Cambridge, Mass. and London. 2002.

Rosetta Code. [Wiki.] "Empty Program." Revision as of 18:29, 13 November 2013. <http://rosettacode.org/wiki/Empty_program>

Rusinkiewicz, Szymon. *smr.c.* 1994 IOCCC entry. <http://www0.us.ioccc.org/1994/smr.hint>

United States Copyright Office. *Circular 61: Copyright Registration for Computer Programs.* Reviewed August 2011. <http://www.copyright.gov/circs/circ61.pdf>

Wikipedia. "Polyglot (computing)." Revision as of 07:37, 31 October 2013. <https://en.wikipedia.org/wiki/Polyglot_%28computing%29>