

REVEALING CODE: WHAT CAN LANGUAGE TEACH SOFTWARE?

A Thesis
Presented to
The Academic Faculty

By

Steve Hodges

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Information Design and Technology

Georgia Institute of Technology
May 2004

REVEALING CODE: WHAT CAN LANGUAGE TEACH SOFTWARE?

Approved By:

Dr. Eugene Thacker, Chair
Dr. Jay Bolter
Dr. Xin Wei Sha
Dr. Steven Mamber

Date Approved:

ACKNOWLEDGMENTS

I would like to thank Professor Eugene Thacker for his guidance in writing this thesis; week after week, I came away from our meetings with useful new ideas. During my entire time as a student in IDT, Professor Thacker provided me a great deal of intellectual support and encouragement.

I would also like to express my appreciation to Professor Steven Mamber, in whose class I first learned about the Oulipo. At first the Oulipo seemed an odd topic in a course titled 'The History of Digital Media', but the connection proved strong and fascinating.

Thank you to Professors Jay Bolter and Xin Wei Sha for their willingness to serve on my committee.

And of course, without the support of my friends and family, I'm not sure where I'd be.

TABLE OF CONTENTS

ACKNOWLEDGMENTS.....	iii
LIST OF FIGURES.....	v
LIST OF TABLES.....	vi
SUMMARY.....	vii
INTRODUCTION.....	1
A BIT ABOUT LANGUAGE.....	7
LANGUAGE ABOUT INFORMATION, INFORMATION ABOUT LANGUAGE.....	30
LANGUAGE BLOWN TO BITS: CRYPTOGRAPHY.....	46
CONCLUSION.....	66
APPENDIX A: CODEWORK.....	83

LIST OF FIGURES

Figure 1	AT&T advertisement <i>Rain</i> , showing a rainstorm of binary digits.....	6
Figure 2	AT&T advertisement showing people composed of binary digits.....	6
Figure 3	Hierarchical nature of computer languages.....	8
Figure 4	Saussure's axes of linguistic study.....	11
Figure 5	Saussure's signifier/signified relationship.....	13
Figure 6	Saussure's visualization of communication process.....	15
Figure 7	Process of compiling the word <i>tree</i> , as it relates to signifier and signified.....	17
Figure 8	Code as a form of stored or potential linguistic communication.....	18
Figure 9	Process of compiling the word <i>printf</i> , as it relates to signifier and signified.....	20
Figure 10	Saussure's illustration of linguistic value.....	22
Figure 11	Thinkmap Visual Thesaurus.....	23
Figure 12	Shannon's schematic representation of communication.....	30
Figure 13	Entropy (information) in a situation of two choices.....	36
Figure 14	Relationships within Calvino's <i>If On A Winter's Night A Traveler</i>	40
Figure 15	Shannon's schematic describing a cryptography system.....	49
Figure 16	The Knight's Tour in Perec's <i>Life A User's Manual</i>	58
Figure 17	HTML rendered within a web browser.....	75
Figure 18	The Web Stalker.....	81

LIST OF TABLES

Table 1	Evolution of spoken and written French.....	12
Table 2	Language-code dialectic.....	67

SUMMARY

In the last twenty years, computer code has emerged from obscure beginnings to occupy a rather prominent place in our culture. We can see evidence of code's cultural presence in our everyday conversation, in the way we interact with computers and networks, and in many current advertisements. Code also occupies an important place in the study of new media; some in that field have gone so far as to call code "the language of our time."

My thesis aims to comprehend the dimensions of this important relationship.

I adopt an interdisciplinary approach to this comparison between language and code, using theory and examples from structuralist linguistics, information theory, computer programming, and literature. A group of experimental French authors called the Oulipo provides many excellent examples for our comparison of language and code. The science of cryptography provides another conceptual bridge between the two areas. The comparison will lead to an examination of some current efforts to engage in a criticism of software and will suggest additional future challenges.

INTRODUCTION

There is an unmistakable association between human language and computer code in today's technology-focused culture. The appropriation of terms from natural languages for technological concepts shows one symptom of this connection; for example, many educators speak with passion about the need for *computer literacy*. At face value, this phrase suggests there exists some base of computer knowledge that all students should learn. But the word *literacy* evokes a powerful connection: Usually *literacy* means knowing how to read human language, and the word also shares a common root with the word *literature*. Thus the much-discussed phrase *computer literacy* establishes a strong association between computers and language.

Such appropriations may in one sense be considered casual turns of phrase, but taken as a whole they do show a kind of cultural framing of computers in terms of language. The very phrase *programming language* certainly implies that computer code can be understood in terms of human language. Programmers speak of "reading code" when they refer to the process of understanding a code's function, and they speak of gaining "fluency" in a given programming language when they refer to learning its syntax and vocabulary — two terms which themselves further the comparison.

There are other indications of this cultural association of code and language. The 2003 Ars Electronica Festival for Art, Technology, and Society suggested that code is "the language of our time." This provocative statement comes from a prominent organization that studies new media and thus should not be dismissed as a throwaway phrase. Ars Electronica is a long-established international festival for electronic artists and new media theorists; it takes place each year at the Ars Electronica Center, built with government funding in Linz, Austria. Several large software and electronics firms have sponsored the festival, including Microsoft, Hewlett-Packard, and SAP (Ars Electronica Center Linz 7). The festival's 2003 theme thus represents a significant statement from an organization with corporate, government, and artistic input.

Part of the Ars statement implies the ubiquity of code, and in that respect it does seem accurate; software and the digital codes that form it are everywhere in our culture. In some cases we are directly confronted with code; when we type 'http://' into a web

browser we name one of the codes underlying the web. Our new digital camera may require us to know whether we prefer to store digital images encoded in JPEG or PNG format. Other codes operate less in the foreground: the internet's Domain Name System (DNS) and TCP/IP codes are essential to the operation of web browsers, even if we needn't be aware of their specific workings. Still other codes are the domain of professional programmers; Java, C, and SQL provide programmers specialized tools used to produce software that may be widely used.

Some examples from the popular media further support the idea of code as a ubiquitous presence. In the 1984 book *Neuromancer*, where the term *cyberspace* originated, William Gibson hinted at a future of abstract codes underlying all manner of social interaction:

Cyberspace. A consensual hallucination experienced daily by billions of legitimate operators, in every nation, by children being taught mathematical concepts... A graphical representation of data abstracted from the banks of every computer in the human system. Unthinkable complexity. Lines of light ranged in the non-space of the mind, clusters and constellations of data. Like city lights, receding... (Gibson 51)

A current AT&T ad campaign also suggests the omnipresence of code. In an advertisement named *Rain*, a metaphorical rainstorm of binary codes showers down upon people frozen in time, bringing them to life (Figure 1). Another ad shows office workers that, upon close inspection, are in fact composed of binary codes (Figure 2). These mainstream ads show how mass culture has come to understand the presence and importance of codes in our lives.

But the Ars declaration of code as "the language of our time" goes beyond merely pointing out the pervasiveness of code. In specifically referring to code as language, the comparison expresses the communicative aspects of code. The examples of HTTP, DNS, and TCP/IP cited above provide obvious examples of this communicative aspect, since they make data flow properly on the internet. Even the JPEG or PNG image format is communicative; when we attach the image to an email or even simply store it for later, we are communicating with another person across space, or perhaps merely with ourselves across time. And specialized programming code such as Java and C can be considered communicative in nature; the software programmer would surely agree that she

communicates with her computer in some sense.

But the comparison of language and code suggests something even more powerful than ubiquity or communication. The Ars Electronica organizers did not describe code simply as "a method of communication of our time" but rather the *language* of our time, and the use of the word *language* elevates code to a very high cultural and philosophical status. For humans, language has come to be seen as the very stuff of thought. In writing about the history of linguistics, author John Sturrock writes that "we have become aware that what we think is conditional on the structure of the language in which we think, and that no communicable thought is possible independently of language. By giving our attention to the medium [of language] it turns out that we are giving our attention to the substance of thought" (Sturrock 25). If we agree with Sturrock's view, then to equate language with code is to equate code with thought!

Thus a comparison of code with language should not pass without careful examination. Code should be thoroughly compared with language before bestowing on it such important status. I propose to carry out this comparison by developing novel relationships between language and code, and to comprehend through these relationships the communicative and cultural aspects of code.

This comparison will require an interdisciplinary approach, relying on theory and examples from linguistics, computer programming, information theory, and literature. In particular, the structuralist linguistic theory of Ferdinand de Saussure and Roman Jakobson and the information theory of Claude Shannon provide a deep theoretical nexus between language and code. Both areas are intimately concerned with issues of language and communication, although each approaches these problems from a different starting point: linguists wanted to explore language's components and structures to understand how meaning arises from symbols, while information theory used mathematical analysis of language to enable efficient communication. In short, perhaps we can say that structuralist linguistics shows an interest in examining the code-like properties of language, and information theory has an essential need to encode language.

Our comparison will also involve selected examples from Oulipian literature. The Oulipo, whose name is a shortened form of Ouvrier de Litterature Potentielle (loosely translated as "Workshop of Potential Literature"), began in the 1960s with a group of French

authors and mathematicians who sought to explore the broad relationships between algorithm and literature. Their methods involved not only analysis of established literary forms but also the discovery of new experimental forms. For example, the group found interest in the lipogram, an older technique in which the writer consciously avoids using one or more letters; an example of a new Oulipian technique is the "S + 7" method, in which each substantive (noun) of a text is replaced with the seventh following it in the dictionary. The use of such algorithmic or rule-based methods, as we will see, demonstrates a hallmark of Oulipian literature; Oulipians also commonly employed mathematical principles within their literary experiments.

Oulipians engaged in type of linguistic and literary criticism that was different from many others in the *level* at which it approached language and literature. Oulipians showed a fascination with discrete linguistic symbols — letters, words, and sentences — and how they are combined in formal, mathematical ways. Perhaps it is this aspect of the Oulipo that provides particular relevance to a discussion of language and code, since both language and code find great interest in how their constituent parts relate to the respective systems. To use Oulipian examples in a discussion of code is not an obvious choice, but we will see how the Oulipo's approach to exploring language suggests ways of understanding language in terms of code. Thus the Oulipo can provide us with an intellectual bridge between the two areas.

The science of cryptography will also lend theory and examples to our comparison. In its quest to obscure language algorithmically, cryptography demands a thorough mathematical analysis of the language it seeks to conceal. Cryptography brings together an informatic interest in language with a linguistic interest in code and thus will supply us with a second bridge between the two areas. We will find similarities between the methods of Oulipian linguistic experimentation and cryptographic encoding of language; each uses manipulation, combination, and juxtaposition of linguistic symbols. But while their methods are similar, their goals differ: Oulipians want to discover principles of language and literature through linguistic manipulation, while cryptographers seek to obfuscate linguistic meaning.

Once established, our comparison between language and code may imply areas for future research. My aim is to suggest that we examine the methods of linguistics and literary criticism for their relevance to the newer fields of software and information technology.

Perhaps because code is a relatively new arrival in our culture, it has not yet been subjected to the same critical analysis as language. And the perception of computer code as an ideologically and creatively neutral problem-solving tool may dissuade some would-be critics. But linguists long ago established that language influences our perception of the world at the same time that it allows us to communicate our thoughts. Code may be similarly influential; as As Matthew Fuller writes in *Behind The Blip: Essays on the Culture of Software*, "each piece of software constructs ways of seeing, knowing, and doing in the world" (Fuller 19). In other words, while we humans program computers, computer code also has ways of programming *us*. "The language of our time" should be subjected to the same evaluation and questioning as the natural language it has supposedly superseded. My hope is that our language-code comparison will provide material to bolster an inquiry into code and its cultural effects.



Figure 1. AT&T advertisement *Rain*, showing a rainstorm of binary digits (AT&T Advertising).

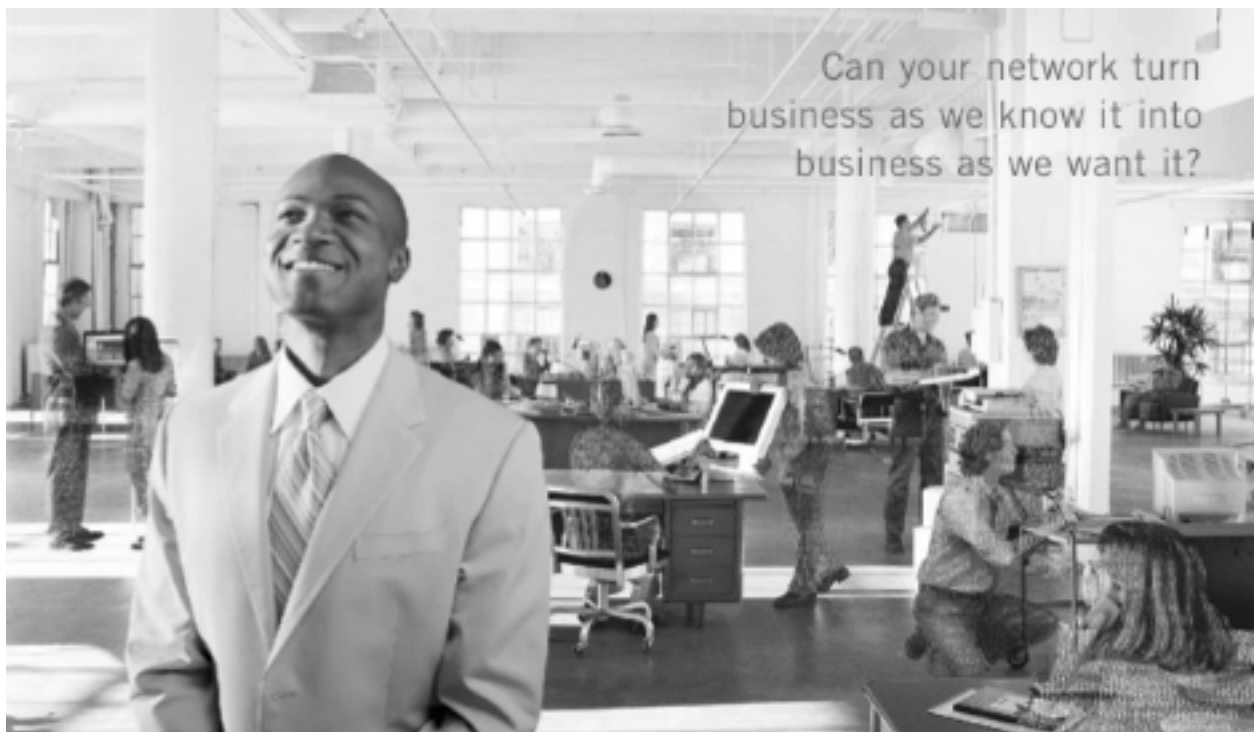


Figure 2. AT&T advertisement showing people composed of binary digits (AT&T Advertising).

A BIT ABOUT LANGUAGE

To begin our comparison of language and code, we turn to the basic definitions of those terms. The ease with which people use these terms belies the complexity of their underlying concepts; as we will see, entire books can be filled with considerations of the nature of language.

A linguistics textbook might approach a definition of language by noting that a language comprises a system of sounds used for communication; knowing a language means knowing its sounds as well as how they can be combined. Languages also consist of the meanings or semantics associated with certain sound patterns or words; this is knowledge of the language's vocabulary. Those who know a language must also know how its words can be combined to form phrases, sentences, and paragraphs; as long as these rules of grammar or syntax are observed a language allows much creativity in linguistic expression. In a nutshell, "a language consists of all the sounds, words, and possible sentences. When you know a language, you know the sounds, the words, and the rules for their combination" (Fromkin and Rodman 11).

This definition is broad enough to cover two large subcategories into which languages are divided: natural languages and computer languages. Humans speak and write natural languages such as English, Spanish, or Chinese, while computer languages are standardized communication techniques for expressing instructions to a computer. Computer languages are usually further classified into a hierarchy according to their similarity to natural languages.

Machine languages are the lowest level languages that a computer can interpret; they are easily understood by computers but almost impossible for humans to understand because they consist entirely of numbers. All instructions given to a computer must at some point be translated to machine language, since the computer will not process any other language directly.

Assembly languages are one step removed from machine languages, in that they allow some limited use of human-recognizable words instead of only numbers.

High-level languages (also known as programming languages) are more similar to human languages in that they frequently use words adopted from natural languages and are

therefore easier for humans to read and write. Examples of high-level languages include C, Java, Perl, and Pascal. Similar to natural languages, programming languages also consist of both syntax (how the various symbols of the language may be combined) and semantics (the meaning of the language constructs) (Webopedia).

We can visualize the relationship among the various computer languages in the following hierarchy; machine language occupies the lowest part of the hierarchy, closest to the computer's hardware, while high-level languages sit atop the hierarchy, closest to the human programmer.

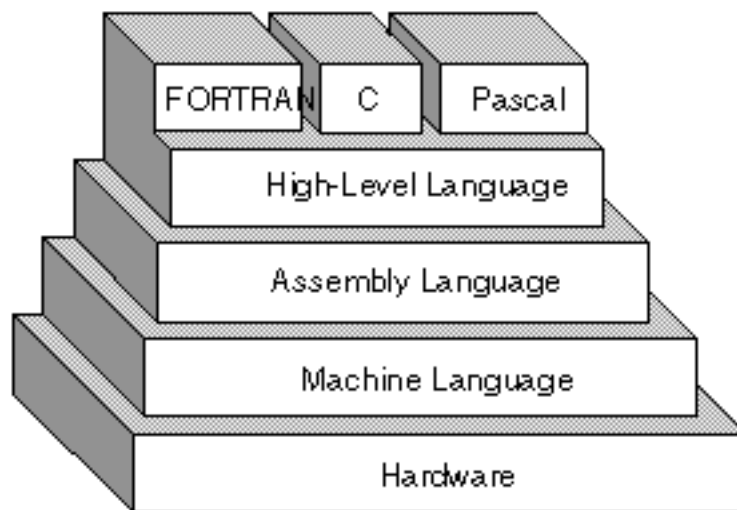


Figure 3. Hierarchical nature of computer languages (Webopedia).

This hierarchical nature of computer languages necessitates a process of *translation* from high-level language to machine language before the computer can act on the instructions; this process is known as *compiling*. Instructions written in high-level language are frequently referred to as *source code*, while instructions written in low-level language are referred to as *object code*. Thus, in other words, compiling is the process of translating source code to object code.

To what does the word *code* refer? The term may be used in different senses; among other things, we might use the term to refer to:

- a system of symbols into which language is converted to allow electronic, secure, or efficient communication of messages, e.g., ASCII code, Morse code;
- a system of letters or numbers used to categorize objects, e.g. ZIP code, area code;
- a system of symbols used for communicating with a computer, e.g., source code, object code;

Thus, beginning with a definition of language, now arrive at the word *code* to describe the nature of computer languages. In light of this, perhaps we can rephrase our original question; if computers operate on codes, such as source code and object code, and yet these codes are frequently referred to as languages, what is the exact nature of the relationship between language and code? The basic definitions we have considered show a certain crossover in terminology between natural languages and computer languages. The very use of the word *language* to describe both natural and computer languages of course implies the strongest connection, and we have seen other overlap as well. Both types of language use the terms *vocabulary*, *syntax*, *grammar*, and *semantics*. We also see a certain cross-pollination between the two areas in everyday language when computer programmers talk of being "fluent" in a given programming language; when a person having difficulty understanding a statement speaks of an inability to "parse" the sentence; when software developers speak of "reading" code; and when computer scientists borrow the terms "translation" and "interpreter" to describe concepts in their discipline. In the introduction we also saw a notable example of terminological mixing in the theme of the 2003 Ars Electronica conference: "Code — The Language of our Time."

But aside from these popular notions, this chapter seeks to dig deeper into the exact nature of the relationship between these two areas. As we will see, concepts from structuralist linguistics and from information theory will provide a useful nexus between language and code. We will also rely on specific examples from Oulipian literature to help with the connection. Oulipian linguistic experimentation with mathematics and combinatorics led them into the a province of exploration that was perhaps equally relevant to the concept of information. In this way, perhaps the Oulipo can be seen as straddling the fence between language and code.

Structuralism

Structuralist linguistics, as first described by Ferdinand de Saussure's 1916 *Course*

in General Linguistics, offers an excellent starting point for exploring the relationships between language, code. In exploring the structure and system of language in general, Saussure's work departed from previous linguistic study, which emphasized the historical development of particular languages. Rather than studying properties of specific languages, Saussure sought to study language as a "complex whole capable of analysis into its constituent elements, these elements being themselves definable not as independent entities but in terms of the mappable relations in which they stand to the other elements with which they coexist in a particular system or structure" (Sturrock 21). In other words, Saussure's approach focused not on a reductionist or atomistic approach which studies constituent parts of a system, but rather on a holistic or gestalt view of language as a whole system. Saussure saw language as a "self-contained system whose interdependent parts function and acquire value through their relationship to the whole" (Saussure xvi).

Structuralism is distinctive for studying its objects explicitly as wholes composed of parts— that is, in terms of the contribution the parts make to the whole. While the first example of structuralist thinking is Saussure's linguistics, in later years its ideas were applied to fields of anthropology, historiography, literature, and psychoanalysis, among others. In any of these fields, the task of the structuralist is "first to determine the elements out of which the structure in question is formed; and then to try to understand the often very complex methods of their interaction (Sturrock 22).

One innovation of Saussure's approach to linguistics was to distinguish between two aspects of language, *langue* (language) and *parole* (speaking). The word *langue* denotes language as a collective entity and a social institution, never possessed in its entirety by one individual. *Langue* is outside of the control of any individual, as the individual can never create or modify it by himself. It exists only by virtue of a sort of signed contract between the members of a community. *Parole*, or speaking, on the other hand, is an individual act whereby a speaker uses language to express his own thought. Thus, language can be thought of as a general system whose rules are owned or controlled only by the collective language community, while instances of speech can be thought of as utterances whose contents are drawn from the general system. In other words, language is a "storehouse filled by the members of a given community through their active use of speaking, a grammatical system that has a potential existence in each brain, or, more specifically, in the brains of a

group of individuals" (Saussure 13).

The distinction between langue and parole leads immediately to a departure in thinking from Saussure's predecessors in linguistics. Previous linguists had generally focused on instances of language, or particular languages and their historical development. Saussure's division of language into langue and parole enabled him to differentiate two branches of linguistics and to call on linguists to approach them separately. The diachronic or dynamic branch of linguistics would be similar to the historical work of previous linguists and would examine changes in a single part of language through time. The synchronic or static branch would seek to examine relations between coexisting parts of the language while excluding the intervention of time. Saussure represents the distinction graphically; the diachronic aspect of study is represented by the axis of successions (CD), which shows the progression of time, while the synchronic is represented by the axis of simultaneities (AB), which represents the relations of all coexisting things:

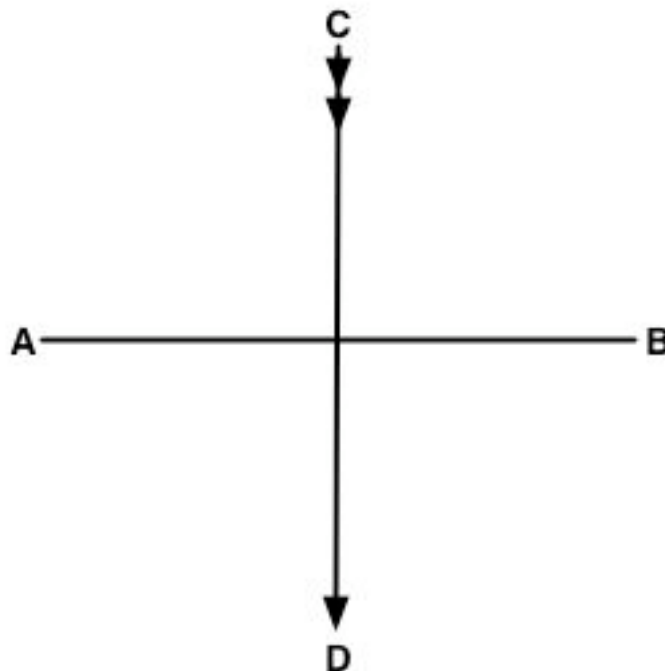


Figure 4. Saussure's axes of linguistic study (Saussure 80).

The diachronic branch of linguistics provides an interesting first comparison between natural language and computer language. Saussure notes that languages constantly evolve

over time, and this evolution provides diachronic linguistics with its focus of study. The collective nature of language ensures that linguistic evolution occurs in a way that is not controlled by any individual but rather by the language community as a whole. Saussure points out that spoken language tends to exhibit less stability over time than written language, and that pronunciation can evolve faster than spelling. He uses the evolution of the French *oi* to illustrate this situation in diachronic linguistics:

	Pronunciation	Written Form
Eleventh century	rei, lei	rei, lei
Thirteenth century	roi, loi	roi, loi
Fourteenth century	roe, loe	roi, loi
Nineteenth century	rwa, lwa	roi, loi

Table 1. Evolution of spoken and written French (Saussure 27).

It seems that that nature of computer languages prevents entirely this type of evolution. Since computer languages are only written and not spoken, computer languages are not susceptible to the kind of pronunciation-induced evolution shown in the example above. But the more important reason for this difference involves the concepts of ownership, collectivity, and standards. Saussure notes that natural languages and their evolution and rules are controlled by linguistic communities and not by individuals; they are collective entities. Computer languages may be thought of as collective entities, but in a different respect; they are special-purpose languages conceived by relatively small groups of people or standards bodies — much smaller than the language communities of which Saussure speaks. For example, the C programming language was created in 1970 by programmer Dennis Ritchie; later the American National Standards Institute (ANSI) developed and documented standards for the language’s grammar, syntax, and vocabulary. As with natural languages, the individual remains powerless to change these standards on his own, but for different reasons; the standards body holds the power to change language, rather than the collective language community.

Programming language standards restrict computer language to an extent unknown to natural languages. Both the constructor of a message (a software developer) and the recipient of the message (a compiler) must adhere to these standards precisely for a message to be passed successfully. Experimentation of the kind seen in natural language, in which small variations in the language are introduced over time, and perhaps accepted by a linguistic community, cannot happen, for anything not conforming to the standard would be rejected.

This is not to say that evolution of computer languages cannot happen at all; it just happens differently, and perhaps in a more deliberate fashion. For computer languages, perhaps a more appropriate term than *evolution* would be *development* or *modification*. Certainly, modifications to the standards of computer languages are introduced from time to time, as shown by the example of Perl 5.8, which replaced the previous language standard, Perl 5.6. But the way in which such changes happen illustrates an important difference between natural and computer languages.

Turning to synchronic linguistics and the structure of language, Saussure defines concepts central to structuralist linguistics: *sign*, *signifier*, and *signified*. The linguistic sign is, essentially, a word. Saussure subdivides the sign into two components: The signifier is a graphical or acoustic representation of the word — a sensory or physical materialization of a word; the signified, on the other hand, is the nonmaterial concept or abstract psychological understanding evoked by the signifier within a linguistic community. (Saussure is careful to distinguish the term *signified* from any actual material object that might be associated with a signifier; philosophers might refer to this as a *referent*. Rather, the signified represents only a mental concept or idea.) Saussure provides a diagrammatic representation of this duality:

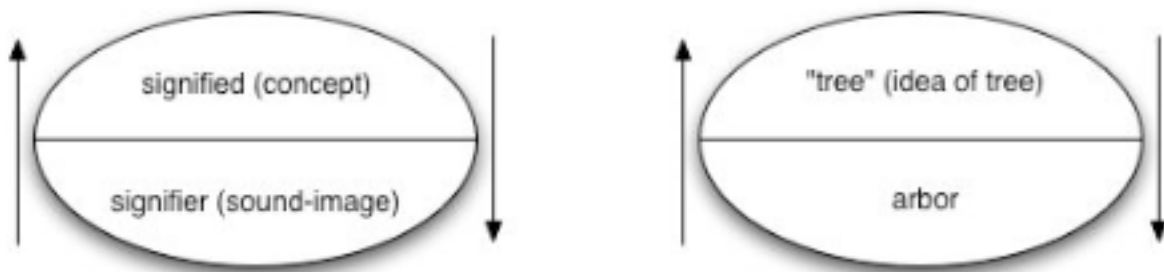


Figure 5. Saussure's signifier/signified relationship (Saussure 66-67).

Saussure proposes that the association between signifier and signified is arbitrary and that this arbitrariness is one of the fundamental principles of natural language. In Saussure's words, the "idea of *sister* is not linked by any inner relationship to the succession of sounds s-o-r which serves as its signifier in French; that it could be represented equally by just any other sequence is proved by differences among languages and by the very existence of different languages"(Saussure 67). (Saussure is careful to that his use of the term *arbitrary* does not indicate that an individual has the power to change a sign in any way once it has become established in the linguistic community; the association between signifier and signified is quite firmly established within the linguistic community. The term simply indicates that the signifier has no natural connection to the signified.) This arbitrariness provides one of the reasons why relationships among words assume such an importance in natural languages; later we will continue to explore this aspect of natural language.

However, perhaps it is time to pause again to assess how Saussure's ideas might be applied to computer languages. To attempt to apply the concepts of sign, signifier, and signified to computer language seems challenging and awkward, but it may bear fruit in our comparison of language and code.

Saussure's concepts of natural language may not transfer directly to computer language because the process and participants of communication seems to differ radically in the two realms. Saussure's concept of linguistic communication involves a minimum of two people, A and B. An act of linguistic communication begins in A's brain, "where mental facts (concepts) are associated with representations of the linguistic sounds (sound-images) that are used for their expression. A given concept unlocks a corresponding sound-image in the brain" (Saussure 11). Then brain then transmits an impulse corresponding to the image to the organs used to produce the sound. Sound waves travel from the A's mouth to B's ear, where the process is reversed; the sound-image is physiologically transmitted to the brain, where it is psychologically associated with a concept. The two participants must of course be members of a common language community. Written communication would follow a similar process.

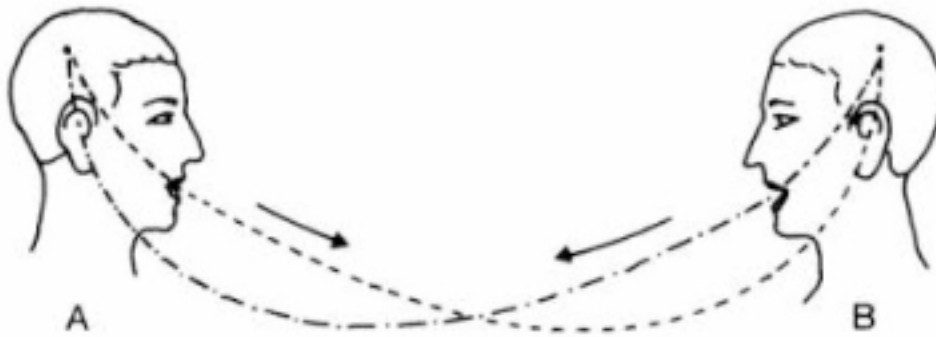


Figure 6. Saussure's visualization of communication process (Saussure 11).

An act of communication involving a computer language must necessarily happen in a different way. There are obvious differences: instead of two human participants, such a communication involves a human and a machine. The machine's part in the communication is handled by a compiler, which will ultimately receive the human's communication. In other words, while Saussure's example involves two brains and two sets of speech organs, computer programming involves one brain, one set of speech-producing organs (fingers and muscles for typing), and one compiler. Of course, not every aspect of the two scenarios is radically different; in either case the participants must be members of the same language community. In natural language communication, the two humans must both be able to use a common natural language, and in programming language communication, human and compiler must both be able to use a common programming language, such as C, Perl, or Java.

But aside from the most obvious differences, communication of computer languages departs most from natural language communication in the step of interpretation or conceptualization. Continuing with Saussure's example of the word *tree*, let us consider how we might communicate this word to a computer. Our example might at first seem nonsensical, since everyone knows that we cannot communicate with computers using normal language. But our example is intended to explore *why* we cannot. Consider first this facile bit of code from the C programming language:

```
void main() {
    printf("tree\n");
}
```

We have very limited means of communicating this word to a computer using the C language, since the word *tree* is not part of its vocabulary. We can of course simply represent the word as a string; this code does nothing more than print the string "tree", along with a newline character. Let us assume that a source code file called *tree.c* contains the text shown above. Note that if the source above is stored in the computer's memory or on the computer's disk, the very characters that comprise the file must first be translated to a binary (base 2) format before we can even proceed with the compilation step. A unix tool called *xdd* shows the file's binary representation as it is stored on the computer's disk:

```
$ xxd -b tree.c
0000000: 01110110 01101111 01101001 01100100 00100000 01101101 void m
0000006: 01100001 01101001 01101110 00101000 00101001 00100000 ain()
000000c: 01111011 00001010 00001001 01110000 01110010 01101001 {...pri
0000012: 01101110 01110100 01100110 00101000 00100010 01110100 ntf("t
0000018: 01110010 01100101 01100101 01011100 01101110 00100010 ree\n"
000001e: 00101001 00111011 00001010 01111101 00001010 00001010 );}..
```

(The output from *xxd* requires some explanation. The first column represents a line number in hexadecimal notation, while each of the next six columns represents a line of binary characters. The last column shows a translation of each of the preceding six columns into an alphabetic notation; if no such translation is possible, the output will record a period in its place.)

In some sense, maybe this shows some form of communication of the word *tree*. In this step, the word's graphical representation has been translated to the ASCII code. (The ASCII code, which stands for American Standard Code for Information Interchange, is a standard for representing English characters as numbers.)

Continuing with the example, suppose that we compile the *tree.c* file into an executable binary. That is, we provide the source code as input to the compiler, which will then translate it into machine language directly readable by the computer's processor. We might perhaps think of this step as the most analogous to the point in natural language communication in which the recipient associates the received signifier with a signified concept. To compile *tree.c*, we use the *gcc* compiler as follows:

```
$ gcc -o tree.o tree.c
```

That is, compile *tree.c* and produce the object file *tree.o* as output. If we use the *xxd* program to examine the output file *tree.o*, we again see the word *tree* among many lines of

output:

```
0000fde: 01101110 01110100 00100000 01110100 01101111 00100000 nt to
0000fe4: 01110010 01110101 01101110 00100000 01110100 01101000 run th
0000fea: 01101001 01110011 00100000 01110000 01110010 01101111 is pro
0000ff0: 01100111 01110010 01100001 01101101 00101110 00001010 gram..
0000ff6: 00000000 00000000 01110100 01110010 01100101 01100101 ..tree
0000ffc: 00001010 00000000 00000000 00000000 00000000 00000000 .....
0001002: 00000000 00000000 00000000 00000000 00000000 00000000 .....
0001008: 00000000 00000000 00000000 00000000 00000000 00000000 .....
000100e: 00011111 01011100 00000000 00000000 00100000 10000100 .\... .
```

Note that the signifier *tree* appears directly in the code, in the same ASCII form shown in the source file.

It seems that our attempt to communicate the concept *tree* to the computer results in nothing more than translation from one language system (English) to another (ASCII representation of an English word). Whereas the process of natural language communication, as described by Saussure, results in the transformation from a signified to a signifier (and vice versa) within the brains of the two participants, in our example nothing more than a translation of signifier occurs. While a human recipient of the signifier might associate the signifier with a signified concept of green, leafy, bark, shade, and so on, in this case it seems that the signified does not play a part in the communication.

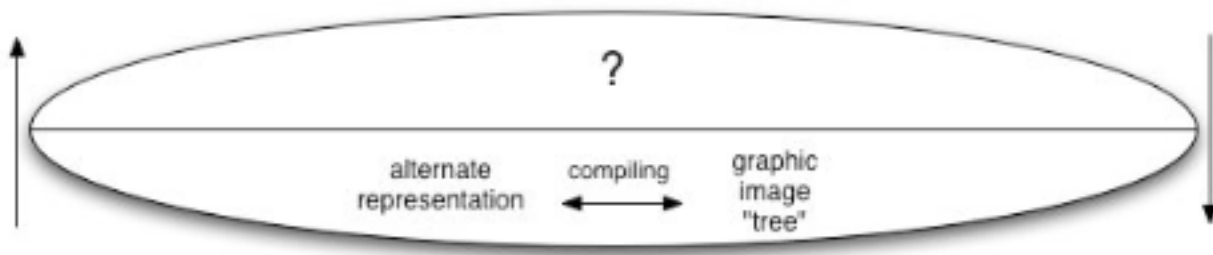


Figure 7. Process of compiling the word *tree*, as it relates to signifier and signified.

Perhaps one way to view programming languages vis-à-vis Saussure's ideas of signifier and signified is as a form of *stored* or *potential* linguistic communication. Rather than provide direct and immediate transmission of a signified, as happens in Saussure's diagram of communication, programming languages seem only to involve translating signifiers to a different form, to be used in a later act of true communication. For example, when a user runs the program *tree.o*, she sees the output:

```
$ ./tree.o  
tree
```

Only at this point does true communication take place, since the user then associates the signifier *tree* with the signified idea of tree. Considering this, perhaps the term *code* describes better the instructions given to a computer than the term programming language. As we saw previously, one definition of the word *code* describes it as a system of symbols into which language is converted. This does seem to be most appropriate description of the process we have seen; the example falls far short of true linguistic communication.

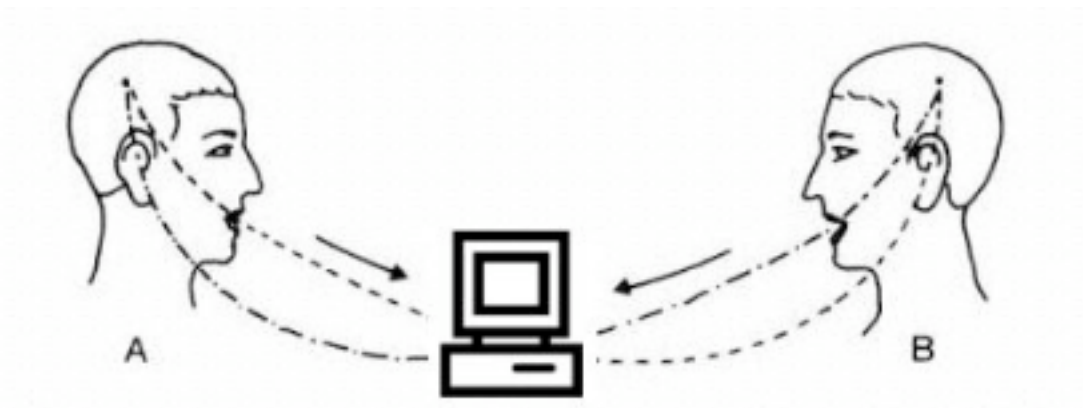


Figure 8. Code as a form of stored or potential linguistic communication.

But perhaps this example does not delve deeply enough into the specific workings of programming languages. The word *tree* is not a reserved word in the C programming language; the compiler attaches no special meaning to it and can thus treat it only as a collection of characters. Reserved words, such as *printf*, *struct*, *int*, *main*, or *char* impart specific instructions to the compiler. Each reserved word of the programming language is defined, according to language specification, such that it must result in certain outputs; *printf* must print a line of text to standard output, *int* allocates a certain amount of memory for an integer, and *main* starts the core program loop. Consider the following source file *tree2.c*:

```

void main() {
    struct tree {
        char name[20];
        char type[20];
        int age;
        char color[20];
    } oak = {
        "oak",
        "deciduous",
        4,
        "green"
    };
    printf ("The %.d year-old %s tree is %s and %s.\n", oak.age, oak.name,
oak.color, oak.type);
}

```

This code uses C's *struct* declaration, which allows creation of complex data types consisting of a collection of primitive data types, such as *int* and *char*. We create the struct named *tree*, which consists of attributes for name, type, age, and color of tree. Further, an instance of tree called *oak* is defined with specific characteristics: name "oak," type "deciduous," age 4, and color "green." Finally, the *printf* statement prints a line containing this data.

In this case, if we compile the source code and inspect the object code in the same manner as we did with the previous example, we do not find an ASCII representation of the word *tree*. Instead, among hundreds of lines of binary code in the object file, the compiler has in a sense translated *tree* to machine language, but in a different sense than it previously transformed it into an ASCII representation. In this case the use of the word *tree* to represent the defined *struct* is unnecessary to the computer's compiling process and serves only for the programmer's convenience. But considering the absence of the literal term *tree* from the object code, has the compiler in any sense translated it to a signified? It might be tempting to wonder, since if we run the program tree2.o, we see the output:

```

$ ./tree2.o
The 4 year-old oak is green and deciduous.

```

But it would be naive to impute the presence of a signified within the code, since no characteristics of *tree* have been output that have not been provided by the programmer. We are left with a more complicated example of the idea previously seen: a translation of signifiers into a different form to be used for subsequent communication.

Instead of simple strings or variable names, perhaps a reserved word will show a

case where true communication of a linguistic sort occurs. From the examples above, *printf* is a reserved word, telling the computer to print a line of text. If we examine the compiled machine code, we may not find an ASCII representation of the word *printf*, but rather, among the thousands of lines of binary code, it seems that the compiler has translated the reserved word to machine language that will perform the associated algorithm, in strict fashion, according to the definition of the C programming language. When the program is run, the computer's processor will follow the instructions of the machine language to produce the desired output.

This example may at first appear more comparable to Saussure's definition of communication. It seems that, in some sense, the compiler has "understood" the word *printf*. Instead of performing a simple direct translation of a signifier to ASCII representation, can it be said that the compiler has associated a signified with a signifier? Is it possible to construe this as an act of communication, in the Saussurean sense, between human and computer? The act of communication began in the human brain, where a concept of "print a line of text" was associated with a graphic image, *printf*, used for its expression. The human brain then transmitted an impulse to muscles and fingers, which produced a representation of the graphic image by pressing the keys p-r-i-n-t-f. In the compilation process, the compiler associates the word *printf* with a concept of *printf* — in this case the "concept" is an algorithm consisting of the steps necessary for the processor to print a line of text.

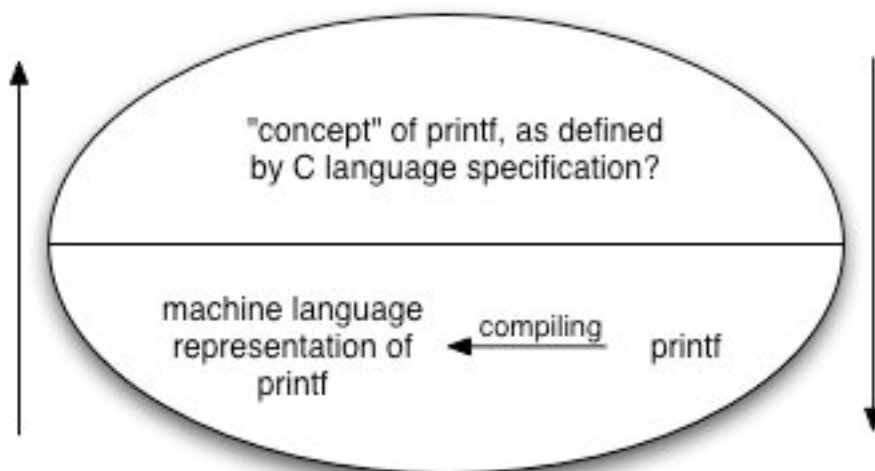


Figure 9. Process of compiling the word *printf*, as it relates to signifier and signified.

Before we conclude that this example shows some genuine correspondence between communication with natural languages and with programming languages, we must consider that it ignores entire dimensions of linguistic theory. One problem with this example concerns our search for evidence of a "signified" within compiled computer code. The search for evidence of a "signified" associated with the word *printf* somewhere within the computer may be sheer folly. Perhaps it would be impossible to find evidence of any "signified" — a concept, a thought, an idea in a nonmaterialized form — somewhere within a computer, as it would be impossible to find material evidence of a concept within the brain of a human being. Any attempt to locate physical evidence of a signified — a purely conceptual entity — introduces a problem: How can one find physical evidence of something which consists only of an idea?

Thus it may ultimately not be possible definitively to answer the question of whether a developer who interacts with a computer via a programming language really communicates with the computer in the Saussurean or linguistic sense. To determine that would seem to require reaching outside of language altogether, to answer to the question of "Can computers think?", or to require that the computer gain sentience.

Our failed attempt to communicate the idea of tree also hints at other stark differences between natural languages and programming languages. Recall that Saussure proposed that a fundamental characteristic of natural language is the arbitrariness of the association between signifier and signified; the idea of *sister* is not linked by any innate relationship to its acoustic or graphic representation. Considering this arbitrariness, Saussure developed the concept of "value" to account for how linguistic signs gain meaning in a language system. Signs gain meaning only when viewed as part of the larger linguistic system; for example, if we could conceive of a language consisting only of the sign *sister*, that word would have no meaning or value without other contrasting terms, such as *mother*, *father*, and *brother*. In Saussure's words, "language is a system of interdependent terms in which the value of each term results solely from the simultaneous presence of the others" (Saussure 114). This concept is quintessentially structuralist, since it proposes that one must examine the entire structure (language) and the relationships between its constituent parts (signs) to determine the proper value of any one element within the system. Saussure represents his concept of value in the following diagram, in which value is represented by arrows between

signs:

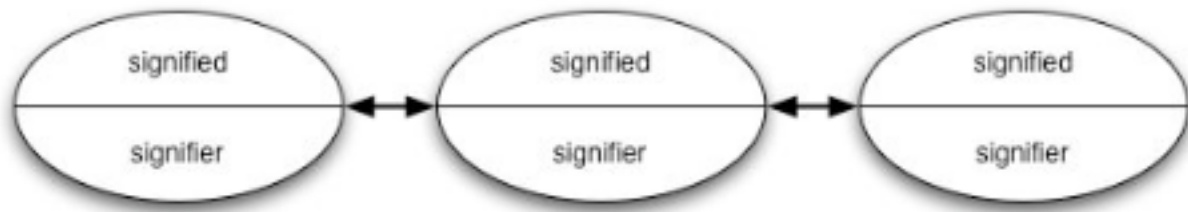


Figure 10. Saussure's illustration of linguistic value (Saussure 115).

The relationships among signs which create value give rise to what Saussure calls 'associative' relations which can link signs in various ways. These relations can arise from similarities in the signifier or the appearance of a word; for example, we may associate the words *painful*, *delightful*, *frightful* because of the common suffix, and we might associate *painful*, *painless*, *pain-free* because of the common root. Associations can also spring from similarities in the signified or concept; we may associate the words *teaching*, *instruction*, *learning*, *education*. These types of associations might be called a 'vertical' dimension of language, since "a word can always evoke everything that can be associated with it in one way or another." As Saussure sees it, "a particular word is like the center of a constellation; it is the point of convergence of an indefinite number of coordinated terms" (Saussure 126).

(One interesting representation of this concept can be found in the Visual Thesaurus, an online program which creates a visual mapping of various words that looks like the constellation Saussure proposed, though it only maps associations of meaning, not of similarity in word appearance.)

has neither ideas nor sounds that existed before the linguistic system, but only conceptual and phonic differences that have issued from the system” (Saussure 120). But with programming languages, meaning of terms is specified when the language is created, and thus terms do not acquire value through associative relationships.

A Failed Comparison?

At this point, one might conclude that there is little substantive similarity between natural and programming languages beyond the fact that both are forms of communication. But one aspect of language, and of structuralist linguistics in particular, remains to be explored: the pattern of language, and how differences in meaning emerge from differences in pattern. Perhaps this aspect will provide a more fruitful comparison between language and code.

In his 1952 essay "Pattern in Linguistics," the structuralist linguist Roman Jakobson wrote that the basic problem of linguistics during the previous hundred years was "finding out the pattern of a given language" (Jakobson 223). To discover such a pattern, Jakobson and other structuralists, including Saussure, saw the necessity of breaking language down into the smallest, most basic, unchanging elements. For a time the smallest elements were considered to be phonemes; these are the sounds that distinguish between two words. "Thus, English p and b are separate phonemes because *pill* and *bill* are separate words; similarly, i and u are separate phonemes because *pill* and *pull* are separate words, and l and t because *pill* and *pit* are separate words" (Sturrock 44). The separate English words *rash* and *lash* show that r and l are distinct phonemes in English; in Korean they are not distinct, so a speaker of Korean will frequently substitute one letter for another. Jakobson further divided phonemes into even smaller opposing elements called "distinctive features," such as 'voiced' and 'not voiced', 'nasalized' and 'not nasalized'.

This linguistic work with phonemes and distinctive features bears some similarity to work in the field of information theory undertaken by Claude Shannon in the 1940s. Shannon developed a specialized concept of information as it applies to communications engineering by examining the situation where one has a simple message to communicate. If one is confronted with a very elementary situation where he has to choose one of two alternative messages, then the message can be communicated using symbolic notation of

0 or 1 to represent the two choices. Shannon used the term *bit*, or binary digit, to describe this notation, since the binary (base 2) system consists only of the digits 0 and 1.

Building on this most basic two-choice situation, Shannon observed that series of bits could represent situations of more than two alternative messages. Thus, he defined information as the minimum number of bits which enable communication of a set of messages. For example, 3 bits of information can represent 8 possible messages by using the following groupings of bits:

000
001
010
011
100
101
110
111

While he was interested in the problems and techniques of communication, Shannon saw no place for meaning within his theory of information. "The word *information*, in this theory, is used in a special sense that must not be confused with its ordinary usage. In particular, *information* must not be confused with meaning," his colleague Warren Weaver wrote (Shannon and Weaver 8). Shannon insisted that the "semantic aspects of communication are irrelevant to the engineering problem" (Shannon and Weaver 31). Thus, a string of bits could represent a character in ASCII code, a decimal number, a pixel within an image, a portion of an MP3 file, or any other digital data.

Despite Shannon's attempt to divorce information theory from semantics, Roman Jakobson saw great similarity between the basic units of linguistic theory and of information theory and this saw a great possibility for "cooperation between modern linguistics and the exact sciences" (Kay 297). Jakobson's work with distinctive features caused him to think of language as a series of yes-or-no decisions in sequential form, much the same as a string of bits of information can be thought of as a series of yes-or-no decisions. "The dichotomous principle underlying the whole system of distinctive features in language has gradually been disclosed by linguists and has found corroboration in the binary digits (or to use the popular port-manteau, bits) employed as a unit of measurement by communication engineers," he wrote (Jakobson 571). The series of yes-or-no decisions shown in a series of bits is also "the real attitude of a receiver of linguistic messages, who for each unit has to make a binary

choice. In the Manhattan Telephone Directory, we find names such as *Bitter*, *Fitter*, *Pitter*, *Titter*. When you first hear one of these names, let us say, *Pitter*, you must first make a set of binary decisions: is it *Pitter* or *Titter*, *Pitter* or *Bitter*, *Pitter* or *Fitter*" (Jakobson 225). Thus Jakobson saw structuralist linguistics and information theory united in their emphasis on context and pattern, "where a message is signified only in relation to other messages" (Kay 300).

In the early 1960s criticism mounted of Jakobson's attempt to employ information theory to describe language; opponents recalled Shannon's assertion that the semantic aspects of communication fall outside the realm of information theory. Information theory requires absolute quantification and precise definition of messages and therefore cannot account for the imprecise and relative nature of language. Recall our frustrating attempt to communicate the signified *tree* to the computer.

But despite information theory's inability to account for all aspects of linguistic communication, the comparison does allow us to see some qualities which language and code share; each system emphasizes the importance of context and pattern in creating different messages.

As we begin to see some basic connection between language and code, some works of Oulipian literature may also help us to understand the connection. Oulipian works seem to borrow concepts from both structuralist linguistics and information theory and may therefore serve as a bridge between the two. On the one hand, Oulipians can be seen as structuralist experimentalists; they seek to explore the relationships between constituent elements of literature by using mathematical and algorithmic techniques that systematically juxtapose these elements in various ways. Some works manipulate the smallest available unit of writing, the letter, while others manipulate syllables, words, sentences, paragraphs, and chapters. But it seems that almost all Oulipian works show an interest in methodically creating and examining patterns and asking how variations in pattern and context affect meaning. As we have seen, this type of thinking is at the very core of information theory, in which variations in the patterns of bits are at the heart of communication.

One Oulipian work which experiments with manipulation is Raymond Queneau's *Exercises in Style*, which tells several different versions of a simple two-paragraph story involving a bus rider, his hat, and an argument with another passenger. The first paragraph

of the exercise entitled "Permutations by groups of 1, 2, 3, and 4 words" reads:

Day one midday towards the on platform rear an of bus S saw
I young a whose man was neck long too who and wearing was
hat a a with cord plaited it round. Started to suddenly he
neighbor claiming harangue his purposely trod that he toes
every on his got in time anyone or out (Queneau, *Exercices*
133).

The first sentence transposes each word with the next, while the second sentence transposes each pair of words with the next. Recomposing the first sentence, for example, we get

One day towards midday on the rear platform of an S bus I
saw a young man whose neck was too long and who was
wearing a hat with a plaited cord round it.

This example shows a common Oulipian interest in upsetting the normal order of delivery as a means of testing the impact that order has on our reading of literature. (Significantly, this upsetting is carried out in an algorithmic, predictable way.) In a somewhat similar vein, the Oulipian author Harry Mathews developed an unusual method for juggling fragments of language in the construction of literature; his method is another of many Oulipian creations that shows the interest in how meaning arises from pattern. In Mathews's algorithm, sets of literary elements (letters, words, sentences, etc.) are manipulated in a certain way. For example, suppose a, b, c, and d represent literary elements of a similar function; that is, if they represent words, they fulfill the same grammatical function. Several sets of such elements are superimposed above one another to form a table:

1	a1	b1	c1	d1
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a4	b4	c4	d4

These sets are juggled in such a way that each member of set n is shifted $n-1$ places to the left or to the right; the resulting table is read downward starting with the first element (a). Mathews gives the following example, using words:

Truth	left	him	cold.
Wealth	made	her	glad.
Work	turned	you	sour.
Love	kept	me	free.

Each word in set n is shifted $n-1$ places left:

Truth	left	him	cold.
made	her	glad.	Wealth
you	sour.	Work	turned
free.	Love	kept	me

Finally, new sentences are formed from the resulting table by reading each vertical set downward:

Truth	made	you	free.
Love	left	her	sour.
Work	kept	him	glad.
Wealth	turned	me	cold.

A different result can be obtained from shifting each set $n+1$ places right and reading the resulting table upward starting with the initial element (Motte 131).

Georges Perec's *Life A User's Manual* provides a subtler example of the Oulipian interest in creating patterns within literature. The novel's very structure and its references operate on two levels; a surface level provides richly descriptive passages, while a deeper level, not necessarily readily observed by the reader, is based on a formalized method of combinatorics. According to Perec's description of this construction, the work was built around a structure called a 10 x 10 Greco-Latin bi-square; this structure guided the placement of furniture, decorations, characters, and other objects within the chapters of the novel.

To understand the Greco-Latin bi-square, Perec suggests that we consider a smaller 3 x 3 version of the one involved in the novel:

Imagine a story 3 chapters long involving 3 characters named Jones, Smith, and Wolkowski. Supply the 3 individuals with 2 sets of attributes: first, headgear -- a cap (C), a bowler hat (H), and a beret (B); second something handheld -- a dog (D), a suitcase (S), and a bouquet of roses (R). Assume the problem to be that of telling a story in which these 6 items will be ascribed to the 3 characters in turn without their ever having the same 2. The following formula:

	Jones	Smith	Wolkowski
chapter 1	CS	BR	HD
chapter 2	BD	HS	CR
chapter 3	HR	CD	BS

— which is nothing more than a very simple 3 x 3 Greco-Latin bi-square — provides the solution. In the first chapter, Jones has a cap and a suitcase, Smith a beret and a bouquet of roses, Wolkowski a bowler hat and a dog.... In *Life A User's Manual*, instead of 2 series of 3 items, 21 times 2 series of 10 items are permuted in this fashion to determine the material of each chapter... At the end of these laborious permutations I found myself with a kind of "schedule of obligations": for every chapter, it listed 42 themes that must appear in it (Mathews, Brotchie, and Queneau 172).

This type of mathematical construct is striking in several ways, not least of which is are its sheer ingenuity and its seeming unorthodoxy. But perhaps the greatest significance lies in the way in which the author juxtaposes objects in pursuit of telling a story. In a very broad way, Perec uses combinations of small units into larger wholes to create elaborate structures and to create an interesting story. In some ways the method seems far from information theory's simple combinations of bits, but at the same time one can see some similarity in the general ideas: small units being combined in different patterns to create varied meaning and message.

In this chapter we have attempted to establish a connection between language and code, beginning from the standpoint of language. Is it possible to do the reverse? That is, starting from a standpoint of information and code, can we find our way back to language? In the next chapter we consider more thoroughly information theory and its relationship to language.

LANGUAGE ABOUT INFORMATION, INFORMATION ABOUT LANGUAGE

In the last chapter we saw that language and code can be compared using principles from structuralist linguistics and information theory. We examined the basic concepts of structuralist linguistic theory — langue and parole, signifier and signified, system of differences, value — and compared them with the theory of information. In this chapter we consider the comparison from the opposite starting point: How can information theory inform our understanding of language? We will continue to examine examples from Oulipian literature to help in the comparison.

The seminal work of information theory, Claude Shannon's 1948 paper *The Mathematical Theory of Communication*, sets out to explore the following problem: One wishes to reproduce at one point a message selected at another point, obviously at some distance. Shannon used the following schematic to illustrate this situation:

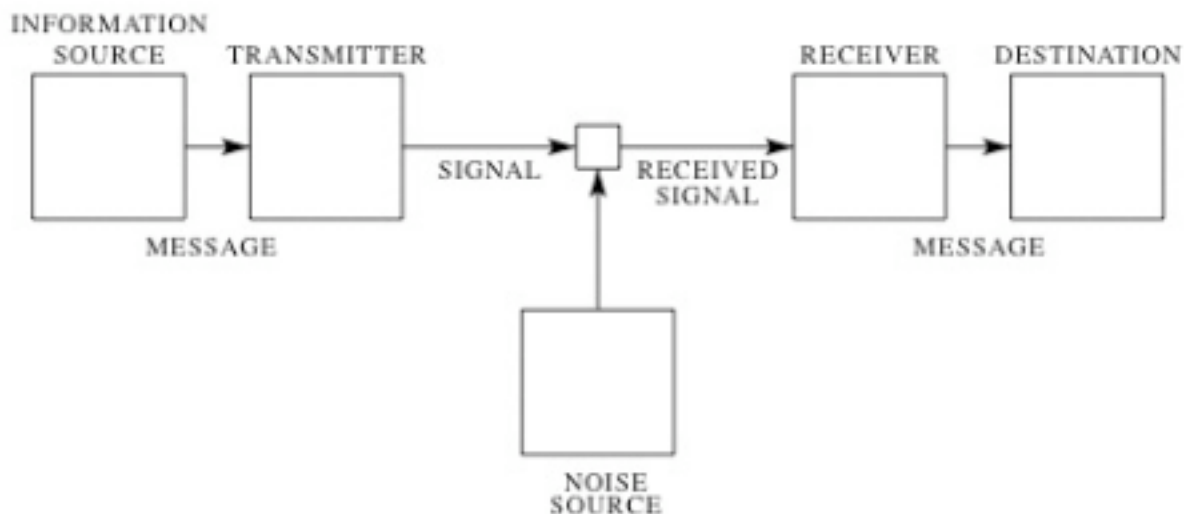


Figure 12. Shannon's schematic representation of communication (Shannon and Weaver 7).

The transmitter transforms the message into a signal suitable for sending over a communication channel to the receiver. For example, in the case of a telephone, the channel is a wire, the signal a varying electrical current on this wire; the transmitter is the telephone, which changes the sound pressure of the voice into the varying electrical current (Shannon and Weaver 7).

A significant point of communication theory is that the message to be communicated

is selected from a *set of possible messages*. As we will see, it is the probabilities introduced by this fact that allow Shannon to develop a mathematical method of quantifying the basic elements of communication.

A simple example shows the mathematical foundation of Shannon's thinking. Suppose the message to be sent is one of two possible messages. The binary digits (also known as *bits*) 0 and 1 of binary (base 2) arithmetic can encode these two messages efficiently; perhaps 0 means "do not send the troops" and 1 means "send the troops." Thus a single bit characterizes a single choice between two possible arbitrary messages. In Shannon's theory, the bit serves as the basic unit of information, and information is defined as the minimum number of bits required to communicate a message. The simple example above is characterized by one bit of information.

Situations presenting more than two possible messages can be described using additional bits. For example, a situation of 8 possible messages can be characterized by 3 bits of information, as follows:

```
message 1: 000
message 2: 001
message 3: 010
message 4: 011
message 5: 100
message 6: 101
message 7: 110
message 8: 111
```

In general, adding one additional bit of information doubles the number of messages that can be represented, as given by the logarithmic function

$$\log_2 y = x$$

where x is the number of bits required, and y is the number of possible messages in a given situation. Thus, if one desires to communicate one of 256 possible messages, he must send 8 bits of information, since

$$\log_2 256 = 8$$

As we noted previously, Shannon considered the meaning of messages communicated to be outside the scope of his theory. He wrote, "frequently, the messages have meaning; that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem" (Shannon and Weaver 31). The technical aspects of transmitting

the series of bits '10011101' is the same whether the pattern of the bits represents an ASCII character or a pixel value within an image.

This agnostic nature of communication theory towards semantics might seem to short-circuit our attempt to compare aspects of Shannon's information theory and language; the purely technical aspects of communication upon which Shannon focuses would seem to have little to do with language or literature. But one need not dig into Shannon's work too deeply to find many ideas relevant to our comparison. Even though his theory avoids questions of semantics, Shannon's work takes into account the link between communication and language. For example, one area of Shannon's interest involved the efficient encoding of messages; since many messages are constructed from language, Shannon saw the statistics of language as a key to efficient communication and to "reducing the required capacity of the communication channel." (Shannon and Weaver 11) (This point is perhaps less true today than in Shannon's time, since many messages today are nonlinguistic, as in the case of digital images, sounds, video, and other types of media. Still, the fact remains that many of Shannon's examples do use examples from language to illustrate points of communication theory.)

To understand Shannon's point about how statistical properties of language can be used to improve coding efficiency, consider the example of Morse code. In English the letter *e* appears more frequently than the letter *q*; because of this, Morse code encodes *e* into the shortest symbol, a dot, while it encodes *q* into a longer sequence of dots and dashes. Morse code thus uses the statistics of language to achieve a savings of time and channel capacity.

More elaborate statistical measurements of language can be made, and Shannon notes that the selection of letters and words depends upon much more than simple letter frequency. Language can be thought of as an information system that generates a message, symbol by symbol, where successive symbols are chosen according to certain probabilities. A system which produces a sequence of symbols governed by a set of probabilities is known in mathematics as a *stochastic process*; a special kind of stochastic process in which the probability of any symbol choice depends on the preceding choices is called a *Markoff process* or *Markoff chain*. Shannon applied these existing mathematical notions to language. A simple example of these ideas can be seen in the fact that, in

English, the probability that the letter *u* follows the letter *q* is quite high, while the probability that the letter *z* follows the letter *q* is very low or zero. Probabilistic measurements can also be applied to whole words: After the three words "in the event," the probability for 'that' as the next word is high, and for 'elephant' as the next word is low (Shannon and Weaver 11).

Because of his use of language to illustrate his points, large parts of Shannon's work bear a striking similarity to a structuralist linguistics text, or even to an Oulipian literary experiment. But it is certainly true that each group focused on different aspects of language in their studies. As we saw in the previous chapter, structuralist linguistics sought to determine the basic components of language and to understand how their opposition creates differences in meaning — the difference between the word *pot* and the word *put*, for example. Shannon, on the other hand, focused mostly on the statistical properties of language — the high probability that the letters *t* or *n* follows *tha*, for example. The Oulipo could perhaps be situated on the fence between structuralists and Shannon. As creators of literature, they were naturally interested in questions of meaning, theme, and narrative; as mathematicians, Oulipians were certainly interested in the quantitative properties of language as well.

Shannon's 'Series of Approximations to English' provides a clear and fascinating example of the probabilistic nature of language. The series is reproduced here in its entirety:

To give a visual idea of how this series of processes approaches a language, typical sequences in the approximations of English have been constructed and are given below. In all cases we have assumed a 27-symbol "alphabet," the 26 letters and a space.

1. Zero-order approximation (symbols independent and equiprobable).

XFOML RXKHRJFFJUJ ZLPWCFWKCYJ
FFJEYVKCQSGHYD QPAAMKBZAACIBZLHJQD.

2. First-order approximation (symbols independent but with frequencies of English text).

OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH
EEI ALHENHTTPA OOBTTVA NAH BRL.

3. Second-order approximation (digram structure as in English).

ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY
ACHIN D ILONASIVE TUCOOWE AT TEASONARE
FUSO TIZIN ANDY TOBE SEACE CTISBE.

4. Third-order approximation (trigram structure as in English).

IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID
PONDENOME OF DEMONSTURES OF THE REPTAGIN
IS REGOACTIONA OF CRE.

5. First-order word approximation. Rather than continue with tetragram, ... , n-gram structure it is easier and better to jump at this point to word units. Here words are chosen independently but with their appropriate frequencies.

REPRESENTING AND SPEEDILY IS AN GOOD APT OR
COME CAN DIFFERENT NATURAL HERE HE THE A IN
CAME THE TO OF TO EXPERT GRAY COME TO
FURNISHES THE LINE MESSAGE HAD BE THESE.

6. Second-order word approximation. The word transition probabilities are correct but no further structure is included.

THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH
WRITER THAT THE CHARACTER OF THIS POINT IS
THEREFORE ANOTHER METHOD FOR THE LETTERS
THAT THE TIME OF WHO EVER TOLD THE PROBLEM
FOR AN UNEXPECTED

The resemblance to ordinary English text increases quite noticeably at each of the above steps (Shannon and Weaver 43).

Shannon's technique for creating these approximations would not seem out of place in a book of Oulipian experiments. To create an approximation of digram structure (example 3), one "opens a book at random and selects a letter at random on the page. This letter is recorded. The book is then opened to another page and one reads until this letter is encountered. The succeeding letter is then recorded. Turning to another page this second letter is searched for and the succeeding letter recorded, etc." (Shannon and Weaver 43). Shannon used a similar technique for examples 4, 5, and 6.

One is tempted to see in these examples not only connections to Oulipian literature but also to questions of artificial intelligence and the Turing test; if one were able to give a computer adequate statistical information about language, could the computer create a fully intelligible and plausible narrative? For our purposes, however, the more important aspect

is that these examples show the underlying mathematical principles behind language. They also offer a point at which to begin to consider the related concepts of probability, choice, uncertainty, and freedom in language. After reading the previous examples, one may be left with the feeling that writing is a more predictable and less freeform enterprise than previously thought. This notion of predictability in writing is essential to Shannon's effort to quantify information, and as we will see, it also has relevance to language and to Oulipian writing.

To understand the relationships between probability, choice, uncertainty, and freedom, we must return to the definition of information and expand upon it. Consider again the simple system in which we want to communicate one of two possible messages. Suppose that the probability of message A is p ; then the probability of message B is $(1-p)$. (This is because the sum of the two probabilities must be 1.) Now consider the extreme case where A is a certain to be chosen; that is, the probability of choosing message A is 1; therefore, B will certainly not be chosen and has probability zero. Such a case, Shannon points out, is characterized by no information at all, since the result is not in doubt; the outcome is a foregone conclusion. Thus, no choice of message results in complete certainty, and no information.

Now consider another case in which the two messages are equally likely to be chosen; that is, they have equal probabilities of 0.5. In this case, choice is maximized, since neither message is more likely than the other. This situation is characterized by one bit of information — the maximum possible information given a single choice. From these examples we can begin to understand the relationship between probability, choice, uncertainty, freedom, and information. The relationship could be stated as follows. When the probabilities of potential messages are nearly equal, this means that:

- choice and uncertainty are maximized, since each message has an equal chance of being chosen;
- freedom is maximized, since the choice is in no part prescribed by the probabilities of the situation itself;
- the situation requires maximum information to communicate the message chosen.

Shannon adopted the term *entropy* from the physical sciences because he found that its mathematical definition exactly matched his quantification of information. The following graph shows the relationship between probabilities of two messages and the resulting

information, or entropy (H):

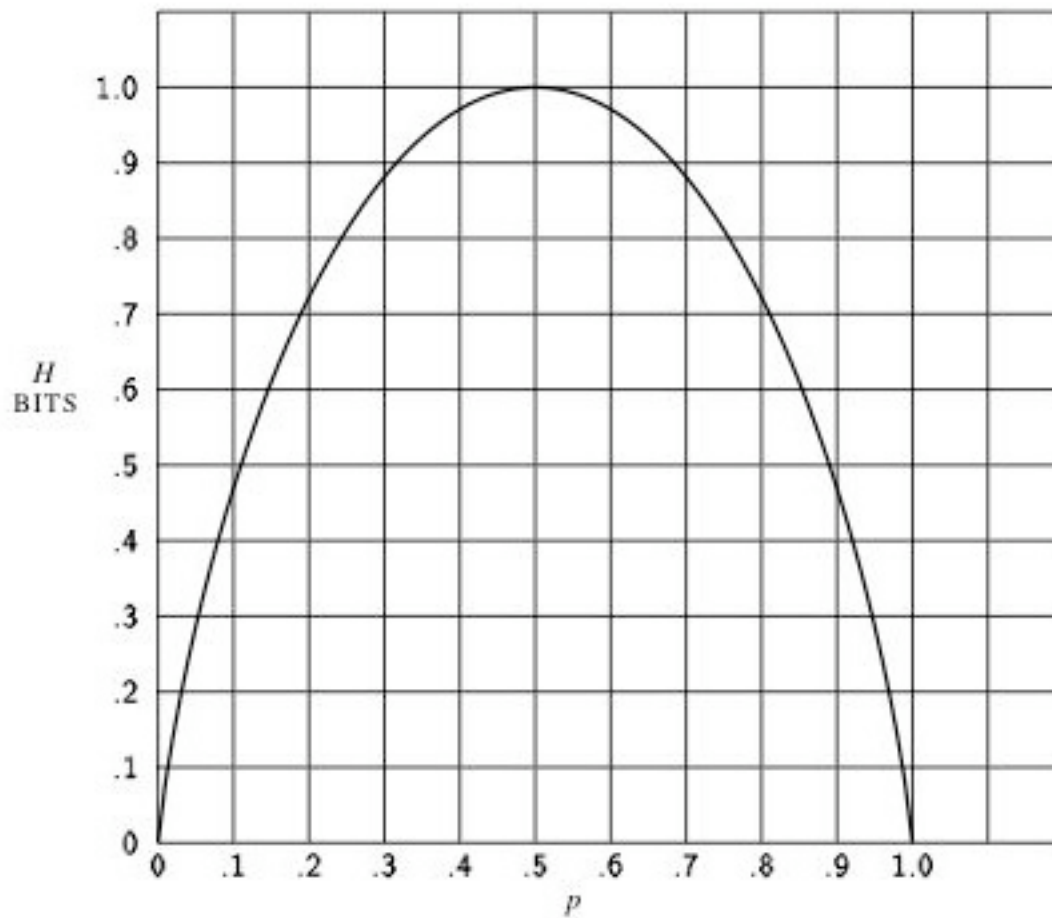


Figure 13. Entropy (information) in a situation of two choices with probabilities p and $1-p$.
($H = - (p \log p + q \log q)$) (Shannon and Weaver 50).

Shannon's thinking shows us that information systems and languages consist of some symbols that are freely chosen and some that are not. In other words, each choice of letter or word made by a writer or speaker consists of some component of free choice and some component which is not free. "That there are probabilities which exert a certain degree of control over the English language also becomes obvious if one thinks, for example, of the fact that in our language the dictionary contains no words whatsoever in which the initial letter j is followed by $b, c, d, f, g, j, k, l, q, r, t, v, w, x,$ or z , so that the probability is actually zero that an initial j be followed by any of these letters" (Shannon and Weaver 11).

Shannon's theory accounted for this fact with the concept of *redundancy*, which can

be thought of as a quantification of that part of an information system or language in which choice does not play a part in the selection of symbols. That is, a symbol is redundant to the extent that the "rules governing the use of the symbols in question" demand its inclusion (Shannon and Weaver 13). For example, if the English *q* is always followed by *u*, it could be said that *u* following *q* is entirely redundant, since its presence could be assumed. An information source might be described as having a *relative entropy* of 80 per cent, and this roughly means that this source is, in its choice of symbols to form a message, about 80 per cent as free as it could possibly be with these same symbols. In this case, the source's redundancy would be 20 per cent (Shannon and Weaver 13). At one time, Shannon estimated the redundancy of English at 50 per cent; "that means that when we write English half of what we write is determined by the structure of the language and half is chosen freely."

The idea of redundancy — that all parts of what we write are not freely chosen — corresponds to currents of thought within Oulipian philosophy. Oulipian thinking focuses heavily on the tension between freedom and rules in language and literature. While some people may conceive of writing as a fully creative endeavor guided solely by inspiration, the idea of linguistic rules is apparent to anyone who has studied spelling, grammar, and writing: verbs must agree with subjects; traffic takes on a *k* when adding *ing*; paragraphs consist of several related sentences.

Such rules bear a distinct resemblance to Shannon's examples of redundancy in that they reduce the writer's freedom of choice and increase the predictability of a work; any writing must observe basic rules simply in order to be understood. In his essay "Rule and Constraint," the Oulipian author Marcel Benabou notes this fact of language, writing that "even the most rabid critics of formalism are forced to admit that there are formal demands which a work cannot elude" (Motte 41).

But it is their *reaction* to these rules that distinguished Oulipians from other literary experimentalists. Rather than challenge the very idea of linguistic or literary rules, Oulipians embraced them and called further for the voluntary adoption of *additional* constraints. Such constraints could, seemingly paradoxically, both expose the natural rules of language as well as provoke creation of new literary forms. Benabou notes that some authors do not see value in the willing adoption of restrictive techniques, suggesting that "it seems like an

unnecessary rule, a superfluous redoubling of the exigencies of technique ... and is exaggerative and excessive" (Motte 41). But he argues that the boundary between innate rule and arbitrary constraint must be challenged "in the name of a better knowledge of the functional modes of language and writing ... to the extent that constraint goes beyond the rules which seem natural only to those people who have barely questioned language, it forces the system out of its routine functioning, thereby compelling it to reveal its hidden resources" (Motte 41).

This presents an interesting contrast between Oulipian thinking and information theory. In Shannon's and Weaver's examples, linguistic rules and conventions are the source of redundancy in language. One might not be blamed for coming away from reading Shannon's work with a somewhat negative impression of those portions of language which are redundant. The very word suggests these portions are superfluous and that these components impinge on the free use of language. Oulipians, on the other hand, seem to revel in the idea of rule-based linguistic construction and seek to embrace and extend the idea of rule within literature. To Oulipians, well-chosen rules can serve as a source of liberation and creativity.

Georges Perec's novel *La Disparition* (translated into English under the title *A Void*) illustrates the Oulipian use of voluntary constraint to spur creativity. The novel is an example of a lipogram, or a text which consciously avoids using one or more letters. According to Perec biographer David Bellos, Perec became interested in the lipogram during an extended period of writer's block during 1967-1968. Though he did not invent the lipogram, Perec and other Oulipians found it fascinating, particularly depending on the difficulty of construction. Oulipian Claude Berge noted that "the Oulipian interest in the lipogram was in proportion to the natural frequency of the letter(s) missed out. In French or in English a lipogram on x is so easy as to lack any literary potential at all" (Bellos 395). But a lipogram on e would be the hardest and the most potentially creative of all. Perec took up that challenge. His obsession with a lipogram on e grew into an effort to write an entire novel.

The effort seems to have helped Perec overcome his writer's block, showing how how constraint could indeed spur creativity. Not only did the challenge of avoiding the letter influence word choice, but the constraint also guided the novel on a larger scale. Beyond its

novelty, the book's "chief experimental aim was to see if e-less French could invent its own story ... and thus vindicate the principle of constraint" (Bellos 399). Indeed the constraint does create the essence of the book; its title, the double entendre *La Disaparition*, refers both to the disappearance of the letter and the disappearance of a man. When his friends begin to look for him, they disappear too. "It is a multiple whodunnit in which everything leads to disappearance, the only possible outcome when a letter has been purloined" (Bellos 401).

Returning to Shannon's examples of redundancy, we can see an interplay between the ideas of information theory and the lipogrammatic constraint of *La Disaparition*. Shannon showed that one characteristic of redundant portions of language is that they can be removed from a message with little or no loss of meaning. In his words, "the redundancy of English is also exhibited by the fact that a great many letters can be deleted from a sentence without making it impossible for a reader to fill the gaps and determine the original meaning. For example, in the following sentence the vowels have been deleted:

MST PPL HV LTTL DFFCLTY N RDNG THS SNTNC (Shannon 215).

It is interesting to compare this example with *La Disaparition*. Both show that the absence of one or more vowels from English can change language but not render it useless. Shannon's example is intended to demonstrate that the rules of language enable us to make compromises to achieve efficiencies, while *La Disaparition* illustrates that the rules of languages can be mobilized and, in a sense, subverted to achieve new literary forms with unexpected results. The rules considered by Shannon and the constraints adopted by the Oulipo stem from common observations of language but are used for very different purposes.

Some Oulipian texts use constraint in such a way that the resulting text may not appear particularly remarkable at all; the constraining principle is largely hidden from the reader. At least one professional reviewer of *La Disaparition* is said not to have noticed the absence of the letter *e*. Another example of such a well-hidden constraint comes from Italo Calvino's novel *If On A Winter's Night A Traveler*. The narrative adopts a game-like repetitive and self-referential quality as its main character, the Reader, himself begins reading a novel that is abruptly cut short; he goes on a quest to find the rest of the story but manages only to find other interesting stories that too are abruptly cut short.

Beneath the surface of the novel lies a complex mechanism guiding the appearances of each character in the book. Since Calvino never makes mention of this mechanism, a reader is hard pressed to notice the constraining principle. In his separate article "How I Wrote One Of My Books," Calvino details the method of construction, explaining an elaborate series of relationships among characters in each of twelve chapters. Calvino offers the following diagrammatic representation of the relationships among the entities within the novel:

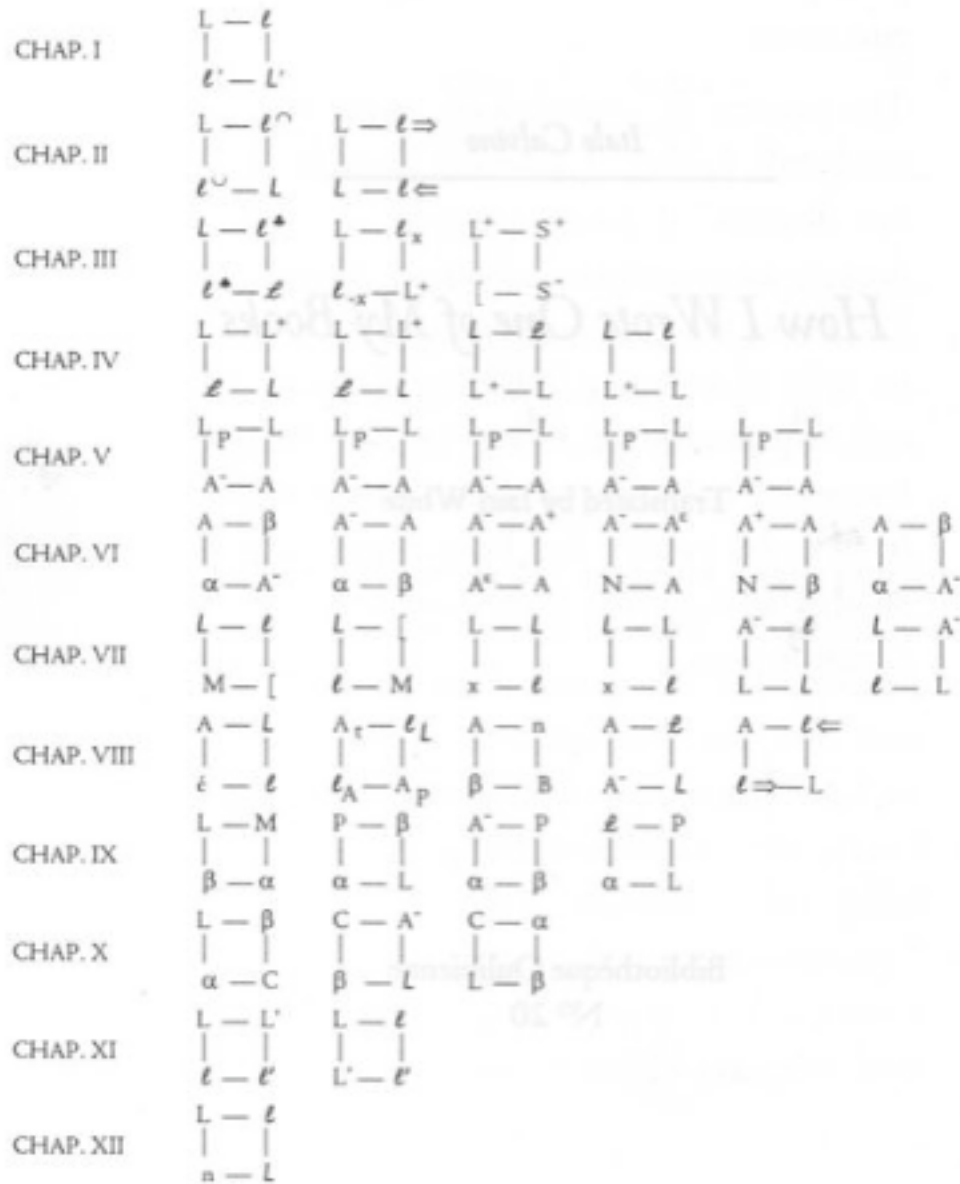


Figure 14. Relationships within Calvino's *If On A Winter's Night A Traveler* (Oulipo Laboratory).

For example, in chapter one, Calvino elaborates:

The reader who is there (L) is reading the book that is there (i)
The book that is there relates the story of a reader who is in the
book (L')
The reader who is in the book does not succeed in reading the
book in the book (i') (*Oulipo Laboratory*)

That some Oulipian texts may appear as random creations when in fact they are produced by formal and predictable methods might be considered an aspect of the text's beauty, especially by Oulipians. Such works offer a point of comparison with information theory and with other literary experimentalists. Shannon's theory showed that randomness of an information source is, in a sense, the opposite of redundancy; that is, a text high in redundancy possesses less information than one randomly generated, since the randomly generated text is created with maximum choice and freedom. In a seemingly related vein, some literary movements sought to break out of the confines of linguistic rules by incorporating techniques of random text generation, producing texts with no underlying structure. But the Oulipo took a consciously opposite approach, frowning upon methods of random text generation and preferring rule-based methods of literary creation. In Raymond Queneau's words, "the classical playwright who writes his tragedy observing a certain number of familiar rules is freer than the poet who writes that which comes into his head and who is the slave of other rules of which he is ignorant (Motte 41).

The philosopher Gregory Bateson suggested an alternative definition of information that can perhaps add something to Shannon's mathematical definition; Bateson suggested that information is "the difference that makes a difference" (Hillis x). In other words, information "is in the distinctions we choose to make significant" and comes about through observation of interactions and relationships. A chameleon can appear invisible when camouflaged on a rock, but when it moves the observer becomes aware of its presence; Bateson suggests that information is produced by this kind of movement or change.

A section of Georges Perec's *Life A User's Manual* evokes Bateson's concept of information. The section consists of numbered lines, each of which contains a sentence fragment describing a scene elsewhere in the book (Perec *Life* 230):

61 Selim's arrow hitting the end wall of a room 888 metres long
62 The staff sergeant deceasing because of his rubber-gum binge
63 The mate of the Fox alighting on Fitz-James's final messages
64 The student staying in a room for six months without budging
65 The producer's wife off yet again on a trip around the globe
66 The central-heating engineer making sure the fueljet ignites
67 The executive who entertained all his workmates very grandly
68 The boy sorting medical blotters he'd been collecting avidly
69 The actor-cook hired by an American lady who was hugely rich
70 The former croupier who turned into a shy, retiring old lady
71 The technician trying a new experiment, and losing 3 fingers
72 The young lady living in the Ardennes with a Belgian builder
73 The Dr's ancestor nearly solving the synthetic gem conundrum
74 The ravishing American magician and Mephisto agreeing a deal
75 The curio dealer's son in red leather on his Guzzi motorbike

In an interview, Perec tantalized his readers by explaining only part of the pattern within these lines:

The "Compendium" that constitutes the centre of *Life A User's Manual* and lists 179 of its characters is a poem subject to two rules, one of which can easily be checked: each line contains sixty typographical signs, with spaces between words counting as signs (Bellos 594).

Beyond the fact that each line consists of 60 characters, there at first does not appear to be any particular relationship among the lines. However, when the eye moves over the lines in various directions it may perceive a pattern: the letter *g* appears in the 60th space in the first line, the 59th space in the second line, and so on until it appears in the first space in the 60th line. One is hard pressed to notice such a constraining principle, though its application could certainly not have been a simple matter. With the constrained letter highlighted one can see the pattern more clearly:

61 Selim's arrow hitting the end wall of a room 888 metres long
62 The staff sergeant deceasing because of his rubber-gum binge
63 The mate of the Fox alighting on Fitz-James's final messages
64 The student staying in a room for six months without budging
65 The producer's wife off yet again on a trip around the globe
66 The central-heating engineer making sure the fueljet ignites
67 The executive who entertained all his workmates very grandly
68 The boy sorting medical blotters he'd been collecting avidly
69 The actor-cook hired by an American lady who was hugely rich
70 The former croupier who turned into a shy, retiring old lady
71 The technician trying a new experiment, and losing 3 fingers
72 The young lady living in the Ardennes with a Belgian builder
73 The Dr's ancestor nearly solving the synthetic gem conundrum
74 The ravishing American magician and Mephisto agreeing a deal
75 The curio dealer's son in red leather on his Guzzi motorbike

Other Oulipian techniques can be related to information theory in a different way. The use of algorithms to generate text can be considered as a somewhat different form of constraint, and a great many Oulipian works involve the use of generative algorithms. Some selection from Queneau's *Exercises in Style* exhibit this quality; the "Epenthesis" exercise begins, "Once dazy abogut mildday own thye repar platforom oaf ann S bugs," while "Permutations by groups of 2, 3, 4, and 5 letters" begins, "Ed on to ay rd wa id sm yo da he nt ar re at pl rm fo an of us sb..." (Queneau Exercises 148, 129). These texts may appear more or less random, and the reader is left to discern the rule used to produce them. ("Epenthesis" inserts an unrelated letter into the midst of each word; the original sentence is "One day about midday on the rear platform of an S bus." The permutation by groups of 2 divides the message into groups of two letters and swaps the order of each pair of groups; the original sentence is "One day towards midday on the rear platform of an S bus.")

The Oulipian techniques which we have seen certainly can provide a different experience from reading a traditional narrative. Perhaps enough has been written about literary structures that allow a reader to choose a path through the narrative. Many have pondered how affording the reader such prerogative affects the idea of authorship of a work. Queneau's *100,000,000,000,000 Poems (100 Trillion Poems)* is one example of an Oulipian work that affords the reader a great deal of flexibility in choosing how to reconstruct the work. It consists of 10 fourteen-line sonnets, each line of which can be interchanged with the corresponding line in any other sonnet. Thus in this extreme case, a reader can produce vast numbers of different sonnets, in effect choosing a different path through the work each time.

But perhaps our comparison of information theory and Oulipian literature can offer something more to the consideration of the author-reader divide besides the well-established ability of a reader to choose his own path. Through their methods of combinatorics and constraint, Oulipians produced works rich in pattern, and it seems to be this fascination with pattern that makes the Oulipo distinct from other experimental literary groups. Oulipian texts, infused as we have seen with ideas corresponding to structuralist linguistics and to information theory, do call on the reader to take on a role not usually needed with traditional texts. The Oulipo frequently offers the reader a structure within which she can juxtapose literary elements and ask herself how context may affect the meaning of

a work.

A reader who endeavors to discover patterns within an Oulipian work inevitably wonders whether all the patterns he discovers were in fact intentionally created by the author. This question seems to be at the heart of many reflections of the nature of authorship, such as Michel Foucault's essay "What Is An Author?". Some consider that the term *author* refers simply to a flesh-and-blood human being; Foucault writes, "critics try to give this intelligible being a realistic status, by discerning, in the individual, a 'deep' motive, a 'creative' power, or a 'design,' the milieu in which writing originates" (Foucault 111). But he suggests that the concept of authorship is also in part a projection of a reader's own interpretation of a text. He continues, "nevertheless, these aspects of an individual which we designate as making him an author are only a projection, in more or less psychologizing terms, of the operations that we force texts to undergo, the connections that we make, the traits that we establish as pertinent, the continuities that we recognize, and the exclusions that we practice" (Foucault 111). In other words, when a reader notices patterns in an Oulipian text or discerns a possible algorithm used to create it, the reader is in a sense projecting authorship, in Foucault's terms, and this experience is part of the thrill of reading an Oulipian work.

Perhaps reading any work of literature, or for that matter any written communication at all, involves a process of recognizing patterns amongst a sea of symbols and deriving meaning from them. Oulipian literature can be thought of as taking an explicit interest in generating these patterns, where traditional literature simply accepts them as a practical necessity of communication. Oulipians carry the art of pattern generation to new levels of mathematical sophistication and implicitly challenge readers to discover their designs. In this sense Oulipian works ask readers to engage in a practice that resembles the science of cryptography, which we will consider in the next chapter.

In chapter two we explored how linguistics can be viewed in terms of information theory; in particular, we saw how some linguists found a comparison between the basic structural units of language and those of information theory. Both in language and information, different messages are created when these units appear in different contexts and different patterns. In this chapter, we added to our comparison many of the mathematical concepts from information theory that also have relevance to language: probability, entropy, and

redundancy. In the next chapter we will further our comparison by looking at a field which can serve as a second bridge between the areas of language and code.

LANGUAGE BLOWN TO BITS: CRYPTOGRAPHY

Before continuing our discussion, it may be useful to review and summarize the basic points we have seen thus far in our comparison of language and code.

Structuralist linguistics suggests that one way of discovering language's properties is to consider relationships among the its most basic components. Linguistic components have no inherent value, but language acquires meaning through context, when the various components are juxtaposed and differentiated from each other.

Information theory too proposed a basic component, the bit, and explored how bits can be combined to produce distinct messages. Linguistic communications can be encoded into a series of yes-or-no choices, or bits, in the same way that some linguists thought language could be represented. Information theory's exposition of the mathematical properties of information was concerned mainly with efficiency and not with semantic meaning, but nonetheless Shannon's many language-oriented examples help our comparison between code and language.

Along the way we have used selected Oulipian works to exemplify concepts from linguistics and information theory because Oulipian literature reflects ideas central to each of these fields. Like linguists, Oulipians found interest in how relationships among components affect the whole; where linguists focused on parts like the phoneme, Oulipians chose to work at the literary level of the letter, sentence, paragraph, chapter, and other units. We have seen many examples of Oulipian literature that implicitly ask how different combinations and contexts effect the meaning and aesthetics of the work. Like information theorists, Oulipians also relied heavily on mathematics in their work with language. Oulipians frequently employed mathematical or algorithmic mechanisms to generate multiple combinatorial possibilities; they also considered the structural properties of language when devising challenging and interesting constraints.

Now it is time to introduce another field that will help us to relate language and code. We have noted that Oulipian authors were not always explicit about their methods and techniques for producing literature and pattern. Calvino gave no indication of the structure behind *If On A Winter's Night A Traveler*, at least within the work itself; Perec's *Life A User's Manual* does not announce the influence of the Greco-Latin bi-square; and some of

Queneau's *Exercises in Style* may take the reader several minutes or more to decipher. It is a question of some interest whether Oulipians actually intended readers to discover their more elaborate constructive schemes, but it is perhaps safe to say that Oulipians wanted to challenge readers by hiding some aspects of their literary creations. The hidden or encoded aspects of Oulipian writing suggests we should also consider the science of secret writing, or cryptography, which happens neatly to combine both considerations of language and information theory.

Technical And Literary Aspects of Cryptography: An Overview

The term *cryptography* is derived from ancient Greek roots that mean "hidden writing," and it has been suggested that the desire to create secret writing is as old as writing itself. Edgar Allen Poe, who was deeply interested in secret writing and whose work frequently references it, suggested in his 1841 essay "A Few Words on Secret Writing" that he could "scarcely imagine a time when there did not exist a necessity, or at least a desire, of transmitting information from one individual to another in such a manner to elude general comprehension" (Rosenheim 19). Poe's observation suggests that the concept of secret writing is "in some sense internal to writing" itself (Rosenheim 19).

But despite its long history, cryptography has only really entered into mainstream consciousness only within the last fifty years. Its importance in military applications and electronic commerce have changed cryptography from a mysterious and unfamiliar issue into a familiar one, even though most people do not comprehend its abstruse workings. Increased awareness also stems from the recent development of cryptography into a specialized professional field. The field owes much of its foundations to Claude Shannon's work in information theory and the specific mathematical definition of information he developed. Perhaps somewhat ironically the concept of information, which in normal usage is associated with *revealing* meaning, turns out to have direct application to cryptography, which is concerned with *concealing* meaning. Shannon's 1946 paper entitled "Communication Theory of Secrecy Systems" shows how the mathematical concepts of information theory apply directly to cryptography.

In his book *The Cryptographic Imagination*, Shawn Rosenheim helps us connect cryptography as a purely technical issue with its direct and indirect applications to literature.

He notes cases in which authors seem to mimic directly the methods of cryptography, swapping letters or hiding messages within texts. In other cases authors do not employ textual manipulation but rather hide aspects of a work within its very structure. Rosenheim finds both of these aspects within Poe's works, and we will see examples of both approaches within Oulipian works as well. The association of cryptography with literature highlights a tension between wanting to convey meaning and seeking to conceal it.

Technical development of cryptography in the last fifty years has focused on the computer as the primary means of achieving high levels of secrecy in communication. Before the advent of computerized cryptography, humans might reasonably expect to decipher manually-created cryptographic writing, but current cryptography makes it virtually impossible to decipher encrypted text without the aid of a computer. Today, cryptographic software like Pretty Good Privacy, or PGP, uses mathematical techniques to provide unprecedented levels of privacy in communications. Computerized cryptography changes the way humans relate to encrypted texts, and it may perhaps change the way humans relate to some types of literature; instead of a game in which humans can participate directly, today's cryptography is necessarily mediated by the computer.

Cryptography Basics

While the methods behind cryptography can be hard to follow, cryptography generally involves the following basic components (Garfinkel 33):

- the process of scrambling a message to render it readable only to certain parties, called *encryption*;
- the message you wish to encrypt, called the *plaintext* or simply *message*;
- the message after it is encrypted, called the *ciphertext* or sometimes *cryptogram*;
- the technique used to transform the plaintext into the ciphertext, called the *encryption algorithm* or *cipher*;
- a secret component used with the cipher, without which it should not be possible to reconstruct the plaintext; this is called the *key*;
- the implicit individual who attempts to intercept and decipher the ciphertext, called the *cryptanalyst* or perhaps simply the *enemy*.

Claude Shannon suggests the following schematic diagram that shows the placement of each of these elements in a secure communication scheme:

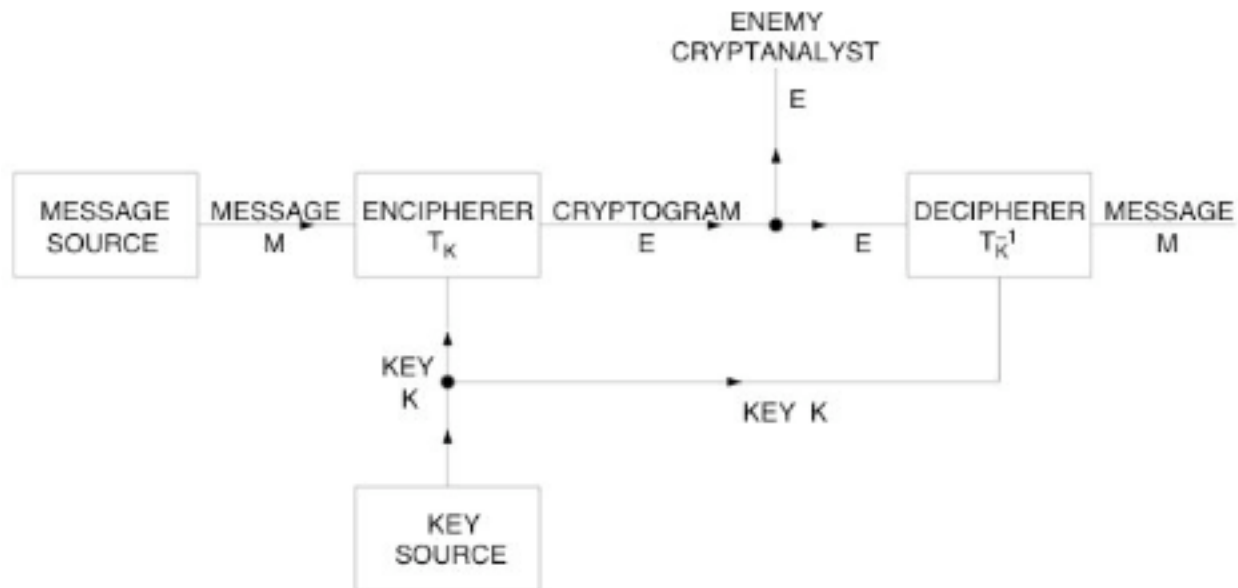


Figure 15. Shannon's schematic describing a cryptography system (Shannon 89).

A complete discussion of encryption methods is beyond the scope of this paper, but a general description of the common methods is necessary for our purposes. One of the least complicated and oldest methods of sending secret messages involves the use of simple codes. For example, suppose a president wants to communicate with a general whether to launch an attack. They could create a simple codebook that contains a different word for each possibility, and each would get a copy of the codebook:

Code Word	Meaning
Mobius	launch the attack
Zebra	don't launch the attack

While this approach can be very secure, it suffers several drawbacks. For most needs it does not offer a sufficiently general method of encryption, since it enables only a small set of prearranged messages. Also, the codebook loses security with each use; through experience, the enemy can easily learn which codes correspond to each action. Of course, the codebook could be used only once and then discarded for a very high level of security, but the number of codebooks needed might quickly become overwhelming. These weaknesses of codes suggest the need for more general algorithmic methods for encrypting arbitrary messages. Most traditional algorithmic techniques worked at the level of

the letter rather than at the level of the word or phrase (as in our codebook example). (The computer, as we will see, changed this traditional alphabetic orientation of ciphers.)

Some alphabetic ciphers in use through the mid-twentieth century are described in Claude Shannon's "Communication Theory of Secrecy Systems" (Shannon 665-670). One simple method is known as transposition, in which the message is divided into groups of equal numbers of letters and some permutation is applied to each group. For example, the message

NEED YOUR HELP RIGHT NOW

when subjected to a permutation of order 41253, is first divided into the 5-letter groups

NEEDY OURHE LPRIG HTNOW

The letters are then transposed according the permutation: the fourth letter appears first, the first appears second, and so on to yield

DNEYE HOUER ILPGR OHTWN

Another traditional method of cryptography is known as simple substitution; rather than changing the ordering of letters, each letter of the plaintext is substituted with a different letter. The key is a permutation of the alphabet, such as

Plaintext		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Substitution		X	G	U	A	C	D	T	B	F	H	R	S	L	M	Q	V	Y	Z	W	I	E	J	O	K	N	P

That is, the letter *A* is replaced with *X*, *B* with *G*, etc. Using this key, simple substitution of the plaintext GEORGES PEREC would yield the ciphertext TCQZTCW VCZCU.

A method similar to simple substitution is digram substitution, in which each two-letter group (digram) in the message is substituted using a key consisting of a permutation consisting of the 676 (26 * 26) possible digrams. The key can be expressed as a 26 x 26 matrix in which the row corresponds to the first letter of the digram and the column corresponds to the second. For example, the key could be expressed as

	A	B	C	D	E	F	G	...
A	MJ	UI	ES	WD	SA	WO	QP	
B	IL	GA	MN	WS	PL	FR	CA	
...								

The digram BE would be replaced by PL in the ciphertext, and the digram AC would be replaced by ES.

The historical development of cryptographic techniques can be seen as a game of

cat-and-mouse between cryptographers and cryptanalysts; each advance in cryptanalysis may render useless previous cryptographic methods and necessitate further innovations. Undoubtedly the security provided by simple substitution was at one time considerable, but a technique called *frequency analysis* rendered it ineffective. In frequency analysis, one begins with knowledge of the frequency of symbols in a language; in English, *E* is the most common letter, followed by *T*, *A*, *O*, and so on. Then the relative frequencies of letters in the ciphertext are calculated. If the text is long enough, the probability is high that the cryptanalyst will be able to see a correspondence between frequencies; if the letter *Z* is the most frequent letter in the ciphertext one might guess that it represents *E* in the plaintext. Ciphertexts encrypted with digram substitution pose little more challenge for frequency analysis, since digram frequencies can also be obtained easily.

Here we pause for a moment to note connections between cryptanalysis and information theory. Recall that Shannon described *redundancy* in an information source as that relative portion in which choice does not play a role in the selection of the next symbol. Writers of English are not perfectly free to choose any letter they want at any time; thanks to the statistical nature of the language they are disproportionately likely to choose the letter *E*. Whereas Shannon's communication theory found this fact useful to improve communication efficiency, cryptanalysts find redundancy useful because it offers them clues about the original message. A text of perfectly random characters, representing maximum information and minimum redundancy, would offer the cryptanalyst no means of reconstructing the plaintext. Cryptography can be thought of as an effort to wring redundancy out of a plaintext, maximizing randomness and destroying its regularity and pattern, while still allowing reconstruction of the original.

After frequency analysis rendered substitution ciphers mostly useless, cryptographers needed new methods to foil cryptanalysts. The Vigniere cipher, published by French diplomat Blaise de Vigniere in 1586, offered such a method. The key consists of a series of letters written below the plaintext; the two letters are added modulo 26 (considering an alphabet where $A=0$ and $Z=25$). For example, with the key GEORGES the encoding process is as follows:

message	C O M E T O M Y A I D
repeated key	G E O R G E S G E O R
cryptogram	I S A V Z O E E E W U

The following Vigniere table can assist with the encryption. To translate the first *C* of the plaintext, locate the column with *C* atop and the row that begins with the relevant letter of the key, in this case *G*. The ciphertext is the letter *I*, located at the intersection of this column and row.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

The Vigniere cipher thwarts frequency analysis because the same plaintext letter may be substituted by different letters in the ciphertext; in the example above, note that the first *M* is replaced by *A* in the ciphertext, while the second *M* is replaced by *E*.

The solution to the Vigniere was more than 200 years in coming and was first discovered by Charles Babbage, who also conceptualized the first computing machine. Babbage noticed that, even with the Vigniere cipher's resistance to frequency analysis, certain patterns in the ciphertext repeated with regularity, thanks to the fact that common words are likely enciphered in only a few different ways. For example, using a key of *KING*, the word *THE* may be enciphered one of 4 ways, depending on where it falls within the

plaintext: it may be enciphered as *DPR*, *BUK*, *GNO*, or *ZRM*. If there are only four ways to encipher such a common words as *the*, it is quite likely that these encipherments will be repeated in the ciphertext. Babbage used this repetition to figure out the length of the key, and in combination with standard frequency analysis he was able to defeat the Vigniere cipher (Singh 69). Once again, the pattern, repetition, and redundancy of language led to the downfall of a cipher.

Traditional Cryptography in Literature

Before considering the successors to Vigniere and the advent of mechanized cryptography, we should consider the relationship between cryptography and literature. The traditional alphabetic ciphers we have seen thus far have a particularly strong relationship with literature; perhaps this is because their mechanisms are simple enough to be applied on a literary level. Edgar Allen Poe provides one interesting case study of cryptography in literature; *The Cryptographic Imagination* explores Poe's literary applications of secret writing. Some works directly incorporate ciphers into the plot, such as *The Gold Bug* (1843), in which a treasure map's encrypted message plays a central role. When confronted with the ciphertext, the main character haughtily explains that "it may well be doubted whether human ingenuity can construct an enigma of the kind which human ingenuity may not, by proper application, resolve" (Poe, Project Gutenberg). He proceeds to describe the cryptanalytic methods used to decipher the message, including considerations of character frequency, character doubling, frequently used words, and a measure of educated guessing. In short, Poe was describing how the quality of predictability in writing can be used as tools of cryptanalysis.

In addition to direct incorporation of ciphers into literature, Rosenheim argues that other works adopt the mystique of cryptography in a broader sense; he uses the term *cryptographic imagination* to describe this appropriation of the spirit of cryptography, if not its exact methods, into literary texts. Specifically, Rosenheim defines the cryptographic imagination as "a constellation of literary techniques concerning secrecy in writing. These include private ciphers, acrostics, allusions, hidden signatures, chiasmal framing, etymological reference, and plagiarism; purloined writing and disappearing inks; and the thematic consequences — anonymity, doubling, identification, and the like — that follow from

cryptographic texts" (Rosenheim 2). Thus Rosenheim makes a comparison between true ciphers, which use technical means to hide meaning from all but the intended recipient, and literary mechanisms meant to conceal meaning to all except the most clever or most observant readers.

Rosenheim identifies Poe's works of detective fiction, such as *The Purloined Letter* and *The Murders in the Rue Morgue*, as examples of the cryptographic imagination; while the words themselves of these works are not enciphered, they offer a mystery that inhibits full understanding without the reader engaging in some process of gathering clues and solving the puzzle. *The Murders in the Rue Morgue*, considered the seminal work of the detective fiction genre, explains the rigorous analysis the character Dupin employs to identify the perpetrator of a double murder. The entire story is an extended description of Dupin's problem-solving skills and the general process of analysis. Poe's implication seems to be that the mysteries of the world can be solved via analysis in much the same way as an encoded message can be deciphered. And perhaps he also implies that literature should be approached in the same way.

Why would authors want to incorporate cryptography or the "cryptographic imagination" into their works? Is the use of such techniques not a violation of an author's responsibility to deal directly with the reader? Poe suggests that one reason authors would adopt such techniques is to challenge the mental faculties of analysis — his own and the reader's. Poe clearly thinks that the analysis necessary to decipher can serve as a source of pleasure. In ruminating on the nature of analysis at the beginning of *The Murders in the Rue Morgue*, Poe suggests that codes play on the natural human urge to learn and to exercise mental powers: "As the strong man exults in his physical ability, delighting in such exercises as call his muscles into action, so glories the analyst in that moral activity which disentangles" (Poe, Project Gutenberg).

Poe also implies an association between analysis and creativity; "the truly imaginative [are] never otherwise than analytic," he wrote (Poe, Project Gutenberg). Poe viewed the construction and solution of ciphers as a source for imaginative and creative mental exercise. Such thinking bears a resemblance to Oulipian observations about inspiration and constraint, which we discussed in the last chapter. In reflecting on the idea that literary creativity is achieved only by giving free reign to the imagination, Raymond

Queneau wrote that "... inspiration which consists in blind obedience to every impulse is in reality a sort of slavery" (Motte 41). Rather, the voluntary adoption of constraint was the key to creativity, since such constraints could help to "force the system out of its routine functioning (Motte 41). Poe's reasons for using cryptography in his works thus resembles the Oulipian use of constraint to spur creativity.

Studies of computer hacker culture can also perhaps provide insight into the use of cryptography in literature. The practice of hacking focuses generally on examining software, communications, and networks in an attempt to understand the details of their function; sometimes hacking involves the attempt to gain access to information considered private. In *Hacker Culture*, Douglas Thomas notes that the driving forces behind hacking include the notions of mastery, control, competition, and friendly play. The emotional thrill of solving a technological puzzle or devising an indecipherable one perhaps provides motivation for those who seek to demonstrate power over technology or superiority of knowledge. Perhaps these same psychological lures help to explain the adoption of cryptography into literature as a game of wits between author and reader.

Rosenheim argues that an author's attempt to conceal meaning from the reader may have the odd of forming a deeper bond between reader and author. He notes, cryptography offers a "utopian moment in which reader and writer are fully present to one another within their two-dimensional cipher [the text]" (Rosenheim 70). Understanding the cryptographic elements in a work fosters "the reader's awareness of the text as a code full of secrets." Poe seems to call on the reader to make an investment of time and energy in puzzling through the text; the reader may appreciate the author's construction of an elaborate and thought-provoking puzzle.

Poe's work seems to anticipate more modern literary techniques. For example, it has been said of Raymond Queneau's work that it "purposely resists the reader," "demands collaboration," and makes the reader "play the game" (Motte 20). His *Exercises in Style* shows an interest in mathematical patterns and combinatorics; he rearranges, removes, or adds letters or experiments with different permutations of words. For example, the first sentence of the exercise called "Permutations by Groups of 5, 6, 7 and 8 Letters" reads

Ytowa oneda ddayo rdsmi earpl nther mofan atfor saway sbusi
anwho oungm kwast senec gandw oolon weari howas twith
ngaha tedco aplai ndit rdrou (Queneau, *Exercises* 131).

Swapping each pair of five-letter groups and inserting spaces between words, this translates to

One day towards midday on the rear platform of an S bus I
saw a young man whose neck was too long and who was
wearing a hat with a plaited cord round it.

This type of experimentation bears a strong resemblance to a traditional transposition cipher. Queneau's use of this and other cryptographic techniques draws the reader in, forces a slower consideration of the text, and perhaps even makes the reader pause to consider the very mechanics of the reading process.

A somewhat different relationship can be seen between traditional cryptography and Georges Perec's *La Disparition*. As we saw previously, the novel is an example of a lipogram on the letter *E* — it consciously avoids using that letter throughout the entire novel. While Perec seems to have adopted the lipogram mainly as a means of challenging himself to overcome an episode of writer's block, the connection with cryptography seems more than passing. By omitting the *E*, Perec was tinkering with the principles of probability and redundancy that lie at the heart of information theory. The lack of a single *E* in the entire 300-page novel destroys the normal letter frequency one would expect; if the entire book were encrypted via a simple substitution cipher, an attempt to use frequency analysis might be stymied by the complete lack of the most frequently occurring letter in the alphabet (Singh 20). Similarly, Claude Shannon noted that omission of vowels can greatly improve secrecy; "the fact that the vowels in a passage can be omitted without essential loss suggests a simple way of greatly improving almost any ciphering system. First delete all vowels, or as much of the message as possible without running the risk of multiple reconstructions, and then encipher the residue (Shannon 701).

Perec's *Life A User's Manual* perhaps provides one of the best examples of Rosenheim's "cryptographic imagination" in Oulipian literature. While not directly concerned with traditional ciphers, the novel's very structure is based on a complicated and formalized pattern of construction. According to the author's description of this hidden framework, the work is constructed around three principle structures: the Greco-Latin bi-square, the Knight's

tour, and a permutating schedule of obligations. We examined the Greco-Latin bi-square in a previous chapter; as we saw, Perec used this mathematical concept to determine which elements — furniture, artwork, characters, etc. — must be mentioned in each chapter. Perec used another construct, the Knight's tour, to order the book's chapters. The novel describes the lives and interrelationships of several characters who all live in a Parisian apartment house. According to Perec,

it would have been tedious to describe the building floor by floor and apartment by apartment; but that was no reason to leave the chapter sequence to chance. So I decided to use a principle derived from an old problem well known to chess enthusiasts as the Knight's tour; it requires moving a knight around the 64 squares of a chess-board without its ever landing more than once on the same square... For the special case of *Life A User's Manual*, a solution for a 10 x 10 chess-board had to be found... The division of the book into six parts was derived from the same principle: each time the knight has finished touching all four sides of the square, a new section begins (Mathews, Brotchie, and Queneau 172).

The following diagram shows the Knight's tour through the apartment building:

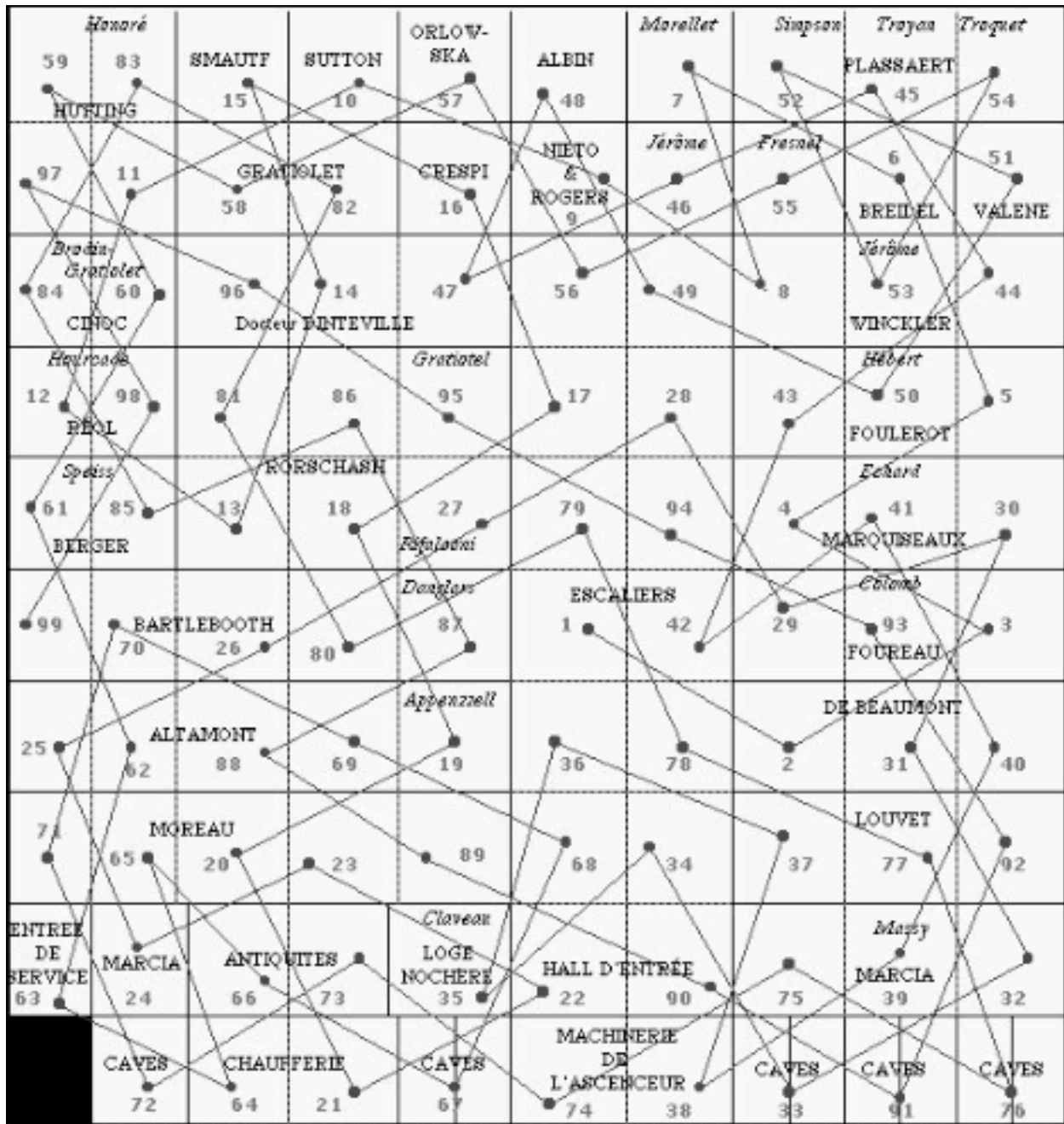


Figure 16. The Knight's Tour in Perec's *Life A User's Manual* (Plan of 11, Rue Simon-Crubellier).

This type of mathematical construct is striking in several ways, not least of which are its sheer ingenuity and its seeming unorthodoxy. The novel's hidden construction is elaborate, algorithmic, mathematical, and not obvious upon first inspection. One is perhaps even left wondering whether Perec really thought any readers would discern fully such a complex pattern. Certainly only a highly clever or observant reader, functioning in some

respects like a cryptanalyst analyzing patterns in a seemingly random cryptogram, would be able to recognize such a pattern without first being told of its existence. Might Perec have adopted this structure not for the reader's amusement but rather only for his own?

Perec seems to hint at an answer to this very question within the novel itself. In his treatment of the character Winckler, Perec describes the thoughts of the puzzle-maker in terms that could easily be understood as thoughts of a cryptographer or perhaps an author.

The puzzle maker

undertakes to ask himself all the questions the player will have to solve, and, instead of allowing chance to cover his tracks, aims to replace it with cunning, trickery, and subterfuge. The organized, coherent, structured signifying space of the picture is cut up not only into the inert, formless elements containing little information or signifying power, but also into false elements, carrying false information (Perec *Life* 191).

These musings on the art of puzzle-making might be interpreted as an analogy for the process of authoring a literary work. Perec the puzzlemaker and constructor of hidden patterns seems to want to tap into the reader's instinct to examine and decipher those patterns.

Explorations of the patterns formed by algorithmic associations of letters, words, sentences, chapters, characters, and other literary elements is perhaps the most significant characteristic of Oulipian literature, distinguishing it from other experimental movements. The interest in pattern and algorithm is perhaps the strongest nexus between the Oulipo and cryptography.

Mechanization of Cryptography

Over the past fifty years cryptography has departed from the traditional alphabetic ciphers Poe found so interesting, as machines have been adopted for use in encryption and decryption. The rapid technical progress in computerized cryptography has made Poe's analysis of the *Gold Bug* cipher seem quaint and simple. Today an approach like Poe's is of little use, since computerized cryptography requires computerized cryptanalysis.

World War II provided a great impetus for the study and advancement of cryptography; competing military needs for secrecy pushed development of machines to produce more advanced cryptography than had ever been seen before. The work of Alan Turing and others who cracked the Enigma code used by the German military has been

extensively documented. The military need for secure communication also fueled exploration of issues of coding, language, and efficiency within the field of information theory, where Claude Shannon played a key role.

In the previous chapters we saw that Shannon defined the bit as the basic unit of information and described how combinations of bits could represent different messages. A computer uses this principle to represent alphanumeric characters in a code known as the American Standard Code for Information Interchange, or ASCII. ASCII assigns a string of bits for each character; for example, a lower-case a is represented by the bits 1100001. This representation of characters has deep ramifications for cryptography in that it allows encryption to take place at a level beneath the letter. For example, suppose we wish to use a simple computer cipher to encrypt the message HELLO using the key DAVID. To understand the process within the computer, first we must translate the message into its ASCII representation:

```
HELLO = 1001000 1000101 1001100 1001100 1001111
DAVID = 1000100 1000001 1010110 1001001 1000100
```

The computer can then add corresponding bits of the message and the key to yield a ciphertext:

```
HELLO = 1001000 1000101 1001100 1001100 1001111
DAVID = 1000100 1000001 1010110 1001001 1000100
-----
0001100 0000100 0011010 0000101 0001011
```

This process shows that binary representation allows computerized cryptography to operate *at a level below the letters*, though the basic operation is similar to a traditional alphabetic substitution cipher in that elements of a message are substituted for other elements. All cryptographic operations, no matter how complex, can be broken down into combinations of such simple operations (Singh 246).

The efficiency of a computer in carrying out the simple operations of cryptography allows it to achieve unprecedented levels of secrecy. Shannon described a perfect secrecy system as one in which a cryptographer gains no advantage whatsoever no matter how much enciphered text he intercepts (Shannon 107); the enciphered text contains no clues and no meaningful patterns and seems purely random. The efficiency and speed with which computers can perform the basic operations of substitution and transposition allow

cryptographers to get much closer to a state of perfect secrecy, though of course these same computers offer cryptanalysts similar advantages in reversing the process. The cat-and-mouse game that started long ago with simple alphabetic substitution goes on, but it has simply moved beyond the human brain's ability to carry out the actual manipulations.

Thus, the foundation laid by Shannon's information theory and by digital computers' binary representation of data led to advances in traditional cryptography. Before computers became available to businesses and the general public, professional cryptography in the United States had been almost the exclusive province of the National Security Agency (NSA). But as computers began to permeate businesses and later individual use, the increasing use of electronic communication in civilian life led to the desire for strong cryptography to be made publicly available. The development of competing private standards of cryptography led the U.S. government to call for a single, interoperable standard of cryptography for computer systems dealing with unclassified data. Eventually, a version of an IBM-developed algorithm was submitted as a candidate for this standard, and a modified version became the US Data Encryption Standard (DES). DES uses 56 bits of information, the equivalent of seven ASCII characters, as its encryption key (Garfinkel 64-65).

Many people have speculated that the involvement of the NSA led to a weaker standard than was technically possible when DES was adopted. Did the NSA purposely shorten the key from 128 bits to 56 so that it still could decrypt ciphertexts encrypted with it? Or did they include a special "trap door" within DES that would allow only the NSA to decipher any DES-encrypted message? These questions led to a debate about the proper relationship of government to encryption technologies: Should the government be allowed to reserve for itself some advantage in cryptographic technologies for purposes of gathering intelligence or combating crime, or should it promote only the strongest possible encryption?

Mistrust of DES played a key role in provoking research on alternative encryption systems. Some researchers continued investigating alternative methods of traditional symmetric key cryptography, in which each party to the communication must possess the same key; each example we have seen thus far that includes a key has been a symmetric key cipher. Other researchers began to question whether there might be a way to avoid the

main problem of symmetric key cryptography: how to communicate the key itself between sender and recipient in a secure way; perhaps there was a way for two parties to communicate in a secure manner without having a previous agreement to communicate. Several such systems were proposed over the course of the early 1970s; the most promising was created by MIT scientists Ronald Rivest, Adi Shamir, and Len Adleman, and it was named for their initials — RSA.

The mathematical foundation of their method came from the fact that "it is easy to multiply two large prime numbers to create an even larger composite number, but it is very difficult to take that large composite number and find its prime factors (Garfinkel 74). That is, factoring is such a computationally intensive procedure that it can foil even computer-aided cryptanalysis. To use RSA cryptography, a person must create a key pair consisting of a public key and a private (or secret) key; the keys correspond in such a way that a message encrypted with one can be decrypted by the other. The public key is distributed freely while the secret key is kept private. If Bob wants to send Alice a private message, Bob obtains a copy of Alice's public key, which is freely available, and encrypts the message with that public key. To read the message, Alice then decrypts the message with her private key. Only Alice's private key can perform this task; not even Bob would be able to decrypt the message after it is encrypted, since he does not have the corresponding private key. This method of encryption, known broadly as public key cryptography, contrasts with traditional symmetric key cryptography.

One implementation of public key cryptography is Pretty Good Privacy, or PGP. In the following example, Steve Hodges sends a PGP encrypted message to Jane Jones using gnupg, an open source implementation of PGP. In the example, Steve has obtained a copy of Jane's public key and is preparing to encrypt the message to her. An ASCII representation of Jane's public key is:

```

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.2.3 (Darwin)
mQGiBD+1LPoRBADGnnk3177Cd65T0ggf7PqwOl3QeRuP3wVidLzI//+uEFiwQbLm
uj+PWR0X4oOkN/0NIx6+luRWVJ/ZEqZSL49YENb4n3NLnWDOAm+zGvrZ55rFQ0s/
wJfd5q5caBs18fPXk7X3sIjmVECqUnNn81eeLFeEoRkz4qYs2yoWgVACzwCgkjTi
aR03sG9P0IdBxzvVGBsLlDcD/3mXwBohQrQTACda8XezgKDbZwsjljN3oj2l2Mzl
+k59XhUW6/UCwlAzYNx0uHrQVVU5NcwU12LlAE4VdriqJqmO7gg3UNmouVP9r68W
BERa8kHTX6KE7WZUk7hQHrG1SlMk+1IHGOmroG2/a78JxnUcvGvYV7QOrc5WbQ
+2KcA/90nyxOtWJFLjAzMI7R5qJl1cbZxmerFpVfv7xkHr07y1p/5PbuMDIHFUzw
9ugjVq7ZvPEtkCepvu9JCq3gcxhVDXpaoRZfLPMCIqH8AS5grdpmL61UaIJJGd9D
hDCag674np0h7rcK2cyU9UzqsEp9imGRz+bvC/m5T7sCPqU2/bQmSmFuZSBKb25l
cyA8amFuZS5q251c0BvaXQuZ2F0ZWNOLmVkdT6IXgQTEQIAHgUCP7Us+gIbAwYL
CQGHAWIDFQIDAxYCAQIEAQIXgAAKCRBlI4PFbnkGRcPeAJ9KdhqEpjsxELmJq5bX
BCm8NeBEAgCdFcSGOHCuN4/VY9RUv37MQPyruVy5AQ0EP7Us/RAEAJW3TgTo5ROX
xP+EIJYoOthg8dTkgGyLYmLqDKPwJtqsD5qE+KpG7hHldm70p3JaU6fKczdmSDNR
mmI1lxceLOGLBdihjNCrbhbmKIPEk4tRbBM9jfmFoHfLsLigObZ4w4UZpXKEwnVE
sitorFTLPLAWHY0G/EQsVaKAoECAQhITAAMFA/9zvQBO4KZQgX9PaoA6FtbjBGE6
M/MAe+T2TR0rT+LBgvYEPHEMqhhn7waXMO5oWuqMgkPQ91aluxG2QFIGVRcJaNTW
Kv3i1Ihp9fvYXcAqsWnC/dQe1RN8CAUadZEG+I/TIaOqT0hnI9LWFKj4holVLMov
FujLlkjdnRhOZfnT2IhJBBgRAGAJBQI/tSz9AhsMAAoJEGUjg8VueQZF+gUAN3sD
0tznWOO+mzBVgqL7GDFGko32AJ9ha3uH1Ubq/ufAPCvdXp/yfpxJXQ==
=e9JL
-----END PGP PUBLIC KEY BLOCK-----

```

Steve uses gnuPG to encrypt the contents of a file called message.txt and make it readable only by Jane Jones. The command takes a plaintext file called message.txt, encrypts it so that is readable only by Jane, and produces the encrypted message in ASCII format:

```

pgg -a -e -r 'Jane Jones' < message.txt
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.2.3 (Darwin)
hQE0A/29Agalv5sJEAP9EZLDNDsW9NmQ7jwnv7HV/5CXf+5uRc1NBoK4PbGBxJji
EzoT5/qB60sqhFJrF68ArSDMJYzcyW5BnSMxEpErWhMea501grAZUyO6fkCDekh7
dIhbDElDUaDCh7DMOpI18qmiEic1RMZoZkU44NLMV55GlsEcGAYxQR4NX1VdIQQE
AIY/Z3nuyh00L2FKvL969HQaM/HKXO4DVO16/VCBBAEwAsd3i0HN3P3uxr7ZbqHN
pCcBTaFCIN5ztmR0o+8Px7DQUPZBNd95qO19jXSYtTsDHMr4BP5aUSnBTnprK8y2
eOXDCc3qzzCQeDR7HJcQgmkn5KlpHnM7SRWPAm9vQsHm01IBjle9wWJQpgBfJRe3
PYnF7WdU0cuBKgkyMrJelLbf3Nvqho/Ck/NTFpC7diCggXAkfrJ/vpO95g9h9n/D
V64rarw2wsIecajC/VQxRq7CGOKX
=bVXy
-----END PGP MESSAGE-----

```

(The availability of ASCII representations of keys and encrypted texts belies the fact that those entities are nothing more than series of bits; encryption programs make them available as text simply as a convenient way for humans to deal with them or to send them via email.)

Jane receives the message and uses gpg to decrypt the contents of the file

message.enc, which contains the ciphertext. She must supply the passphrase for her private key; this is an additional measure of security in PGP. The output is the original plaintext produced by Steve — "Hello my name is Steve":

```
gpg -d < message.enc
You need a passphrase to unlock the secret key for
user: "Jane Jones "
1024-bit ELG-E key, ID A5BF9B23, created 2003-11-14 (main key ID 6E790645)
gpg: encrypted with 1024-bit ELG-E key, ID A5BF9B23, created 2003-11-14
"Jane Jones "
Hello my name is Steve
```

The example of PGP shows how much the technology of cryptography has advanced since Poe's time. The history of cryptographic techniques has progressed from relatively simple codes and substitution ciphers to public key cryptography only decipherable by advanced computational methods. In the works of Poe and the Oulipo, authors explored the tantalizing line between surface and depth, purposely employing cryptographic and cryptography-like techniques to draw the reader into the reconstruction of a work. These works seem to offer the reader an achievable goal in that their methods allow for the possibility of human decipherment. One wonders what use Poe or Oulipian authors would make of today's strong cryptography, which moves beyond human ability to participate in the game of cryptography, using computation to achieve an unprecedented separation of signifier from signified. Does technical cryptography have any role in literature any more? If Raymond Queneau were to write *Exercises in Style* today, one wonders if there would exist a PGP-based iteration such as this:

```
gpg -a -e -r 'Jane Jones' < notation.txt
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.2.3 (Darwin)
hQE0A/29Agalv5sjEAP/fmpO4SPAff324b5LnXsC+vsdbD2sWhLJwXdme3b8s90/
hdoI4YKsT2+rLIeC56HNig97T0l7XUgEYIBYwODdGeSh0KPwNfaej+y2fROLcPry
upJLFDDq2d9f8xGyy+jPxY/A2SR4OXq2R2IQN2au1QOnQ5M4f1zaNwEiEcLHiUcD
/jyMs7OgU7jTBfySq0/FmdPgPQqrQYFMRNQcrFoKoYsPC66j+6IeAscqbFC/eljj
XYPvCPWcLUvJ08n+0feeIU23+0Tan4VY0ZFRD0L8pxUrEYuhURkpGikZuwVehk0B
bTLFuiei1h2X9H9lxYpESSGUClymzeJAtRclAJQV6aqf0sEMARNA/FdPVjtj69eX
VJa6zTD5QK9x7xwplLTyD4IpmPU9pYio+1Y47kEqouQrv24v2M9MgCB/HG/mCcMr
eiQQQDMBEY9rE2WCWcLmsABxsJcJf5m+StDQq7R1ojmCYvup/cF0GUfymRXxpAQn
KeEmNwqplvS68IDb9i2ivaed2elly9KoUsmKQIGAdijJFzEat5X2scgt+Uwyc5WA
ZpcRlBWWB/hJ7SN83MxVpY28EQBfv6T1bfYlshaBnn9m18upqSpRu/UtyVLpyHKj
AGdv6YKTeFCYphhmQL2dmFgBwhEgmLHqF9WUBhzAfx6sq1BA2ibQ6DtnbVfvWYl3
sgIRp/HJNFO/DFqDS3BKu4Hl1v4GJomq47WgD5vSBTSWP1JWhElQaPOqyPmVxIVV
kyA5DvnpV2kf3hM0lxLLAamm63dvOwMgDGF+y5TCQNDLW5CNzXYPGzFW+ps58sjU
NPgy6Fi2f2deSRRFp+r8yBmb4b4XMXChVUCAVPmwUDEHBdnLzE3QN6Q30R/2ybWM
npfSVQ4YrchKChxVxAyDkKPYer1rxJaYxY1TN98p3oiy0EQhx/mDLDiIXzFSb8u3
Im0rfv8mheM8uGGeF44odw==
=4Hsu
-----END PGP MESSAGE-----
```

CONCLUSION

The time has come to explore the significance of our comparison between human language and machine code. That is, do the various lines of comparison seen in the last chapters lead to new ways of thinking about software, or at least to questions for future research?

First let us briefly review some points of our language-code dialectic. In the beginning of this study, we noted a growing cultural realization of some relationship between human language and software code. The 2003 Ars Electronica conference declares that code is "the language of our time;" computer programmers describe themselves as "fluent in C and java;" a course title in a typical curriculum might be "Languages and Translation." We have grappled with a comparison of language and code because it forms the foundation of our contention that others' thinking about language and literature can inform our thinking about software. Certainly, language and code cannot be equated, but we have shown some areas where this comparison holds and where it breaks down. The following table provides a short summary of some of the areas we have considered:

Language	Code
natural language	programming language
tool of thought and of communication	tool of communication
human-human communication	human-computer communication computer-computer communication
understanding	compiling
structuralist linguistics	information theory
Saussure, Jakobson	Shannon, Weaver
language communities	standards bodies
signified	message
phoneme	bit
language as a series of binary choices	information as a series of binary choices
semantic meaning	communication efficiency
letter	ASCII code
system of differences	a difference that makes a difference
importance of context	importance of context
Oulipian constraint	redundancy
Oulipian language manipulation	Shannon's language examples
cryptographic imagination	technical cryptography
generative literature	software algorithms
language as code	code as language

Table 2. Language-code dialectic.

We can see that language and code do share some commonalities. A common thread between structuralist linguistics and information theory seems to be structuralist thinking: To understand a whole entity, first determine the elements out of which the whole is formed, and then to try to understand the often very complex methods of their interaction. Each field seeks to understand the properties of different arrangements of their respective components, the phoneme and bit. We also saw how each area reached into the other for theoretical enrichment. Recall Roman Jakobson's interest in an informatic understanding of

language, where he offered a view of language as a series of binary choices. And consider Claude Shannon's efforts to explain concepts like information and redundancy in linguistic terms.

But while there are similarities, code and language are not the same. Saussure's work with sign- signifier-signified and value attempts to account for semantic meaning, together with the slippery associations, incomplete definitions, and uncertainty of human thought. Information theory, on the other hand, makes no attempt to account for meaning; the important consideration in communicating a message is how many alternate messages are possible. While both linguists and information theorists are interested in patterns in communication, linguists wanted to understand how these patterns create meaning, while information theorists wanted to use pattern to achieve compression and efficiency.

In our comparison we used Oulipian literary works and the science of cryptography as case studies to show how the theoretical aspects of linguistics and information theory combine and overlap. Oulipians combined a structuralist interest in the part-to-whole relationship with an informatic interest in the mathematical properties of language. Mathews's algorithm and Queneau's *100,000,000,000,000 Sonnets* demonstrate the Oulipian fascination with manipulating language into different juxtapositions and contexts. Perec's *La Disparition* and Queneau's *Exercises in Style* illustrate the Oulipian interest in language's mathematical properties. In some cases Oulipians borrow the spirit of cryptography, seeking to employ mathematics to obscure the usual patterns of language while installing other patterns only visible to a chosen few. Traditional cryptography is, in a sense, the practical intersection of language and information theory; in it we find an informatic interest in language joined with a linguistic interest in information. We saw from Shannon's work in secrecy systems that traditional cryptography begins with an understanding of linguistic principles and seeks to subvert or hide normal linguistic pattern through application of information theory.

In our discussion of several Oulipian examples we have seen how information theory can inform our thinking about literature; the ideas of uncertainty, randomness, and redundancy are clearly at play in the *Exercises in Style* and *La Disparition*, for example. But now we should ask: does this comparison affect our thinking in the other direction? Or, in other words, does our effort to contextualize code in terms of language make a difference in

how we interpret information and the products of information, such as software?

I propose that we focus on two concepts to answer this question. Our comparison between language and code may prompt us to think of software in terms of *structures* and *levels*. Oulipian works have shows us that we can examine linguistic structures on a micro level of letters and words; or we can examine structures on a macro level. Consider the small-scale letter manipulations in *Exercises in Style* and the grand-scale complex patterns that pervade Calvino's *If On A Winter's Night A Traveler* and Perec's *Life A User's Manual*. When we are attuned to noticing such small and large structures in language, we may be more inclined to see such relationships that we might have previously overlooked. We saw in a previous chapter Perec's unusual section from *Life A User's Manual* which consists of 179 lines of 60 characters, each describing a character or event from elsewhere in the novel. With an understanding of the Oulipian interest in structures and with an eye attuned to structural relationships, we noticed a pattern of the letter g proceeding diagonally through the second section. This is interesting, but we suspect there may be more to the pattern. We continue to inspect the text for pattern and find additional diagonal patterns; highlighting the diagonals in each section, we see (Perec *Life* 228-233):

```
1 The Coronation at Covadonga of Alkhamah's victor, Don Pelage
2 The Russian singer and Schonberg living in Holland as exiles
3 The deaf cat on the top floor with one blue & one yellow eye
4 Barrels of sand being filled by order of the fumbling cretin
5 The miserly old woman marking all her expenses in a notebook
6 The puzzlemaker's backgammon game giving him his bad tempers
7 The concierge watering potted plants for residents when away
8 The parents naming their son Gilbert after Becaud their idol
9 A bigamous count's wife accepting his Turkish female rescuer
10 The businesswoman, regretting that she had to leave the land
11 The boy taking down the bins dreaming how to write his novel
12 The Australian round-the-worlder and her well-dressed nephew
13 The anthropologist, failing to locate the ever-evasive tribe
14 The cook's refusal of an oven with the self-cleansing device
15 1% sacrificed to art by the MD of a world-wide hotel company
16 The nurse casually leafing through a shiny new photomagazine
17 The poet who went on a pilgrimage shipwrecked at Arkhangelsk
18 The impatient Italian violinplayer who riled his miniaturist
19 The fat, sausage-eating couple keeping their wireless set on
20 The one-armed officer after the bombardment of General H.-Q.
21 The daughter's sad reveries, at the side of her father's bed
22 Austrian customers getting just the steamiest "Turkish Bath"
23 The Paraguayan odd-job man, getting ready to ignite a letter
24 The billionaire sporting knickerbockers to practice painting
25 The Woods & Water Dept. official opens a sanctuary for birds
26 The widow with her souvenirs wrapped in old weekly magazines
```

27 An international thief taken to be a high-ranking magistrate
28 Robinson Crusoe leading a very decent life style on his isle
29 The domino-playing rodent who feasted on dried-out Edam rind
30 The suffering "word-snuffer" messing around in old bookshops
31 The black-clad investigator selling the latest key to dreams
32 The man in vegetable oils opening a fish restaurant in Paris
33 The famous old soldier killed by a loose Venetian chandelier
34 The injured cyclist who then married his pace-maker's sister
35 The cook whose master ingested only eggs and poached haddock
36 The newly-weds taking credit over 2 yrs to have a luxury bed
37 The art dealer's deserted wife, left for an Italian Angelina
38 The childhood friend reading the biographies of her 5 nieces
39 The gentleman who inserted into bottles figures made of cork
40 An archaeologist researching the Arab kings' Spanish capital
41 The Pole living quietly in the Oise now his clowning is over
42 The hag who cut the hot water to stop her son-in-law shaving
43 A Dutchman who knew any No. could be but the sum of K primes
44 Robert Scipion devising his supremely clever cross-word clue
45 The scientist learning to lip-read the deaf-mute's equations
46 The Albanian terrorist serenading his love, an American star
47 The Stuttgarter businessman wanting to roast his leg of boar
48 Dodeca's owner's son preferring the porn trade to priesthood
49 A barman speaking pidgin in order to swap his mother-goddess
50 The boy seeing in his dream the cake he had not been allowed
51 7 actors each refusing the role after they'd seen the script
52 A deserter from US forces in Korea allowing his squad to die
53 The superstar who started out as a sex-changed guitar-player
54 A redheaded white man enjoying a rich maharajah's tiger hunt
55 A liberal grandfather moved to creation by a detective story
56 The expert penman copying suras from the Koran in the casbah
57 Angelica's aria from Arconati's Orlando requested by Orfanik
58 The actor plotting suicide with the help of a foster brother
59 Her arm held high a Japanese athlete bears the Olympic torch
60 Embattled Aetius stopping the Huns on the Catalaunian Fields

61 Selim's arrow hitting the end wall of a room 888 metres long
62 The staff sergeant deceasing because of his rubber-gum binge
63 The mate of the Fox alighting on Fitz-James's final messages
64 The student staying in a room for six months without budging
65 The producer's wife off yet again on a trip around the globe
66 The central-heating engineer making sure the fueljet ignites
67 The executive who entertained all his workmates very grandly
68 The boy sorting medical blotters he'd been collecting avidly
69 The actor-cook hired by an American lady who was hugely rich
70 The former croupier who turned into a shy, retiring old lady
71 The technician trying a new experiment, and losing 3 fingers
72 The young lady living in the Ardennes with a Belgian builder
73 The Dr's ancestor nearly solving the synthetic gem conundrum
74 The ravishing American magician and Mephisto agreeing a deal
75 The curio dealer's son in red leather on his Guzzi motorbike
76 The principal destroying the secrets of the German scientist
77 The historian, turned down 46 times, burning his 1200-pp. MS
78 A Jap who turned a quartz watch Co into a gigantic syndicate
79 The Swedish diplomat trying madly to avenge his son and wife

80 The delayed voyager begging to have her green beans returned
81 The star seeking admission by meditating a recipe for afters
82 The lady who was interested in hoarding clockwork mechanisms
83 The magician guessing answers with digits selected at random
84 The Russian prince presenting a mahogany sofa shaped in an S
85 The superfluous driver playing cardgames to use up his hours
86 A medic, hoping to make a mark on gastronomy with crab salad
87 An optimistic engineer liquidating his exotic hides business
88 The Japanese sage initiating in great anguish Three Free Men
89 A selftaught old man again going over his sanatorium stories
90 A relative twice removed, obliged to auction his inheritance
91 Customs & Excise men unpacking the raging princess's samovar
92 The trader in Indian cotton goods doing up a flat on the 8th
93 French-style overtures brought to the Hamburg opera by a Hun
94 Marguerite, restoring things seen through a magnifying glass
95 The puzzlemaker with his ginger cat taking the name of Cheri
96 The nightclub waiter, legging up on stage to start a cabaret
97 The rich amateur leaving his musical collection to a library
98 A housing and estate agency woman looking at that empty flat
99 The lady doing the Englishman's black cardboard puzzle boxes
100 The critic committing 4 crimes for 1 of Percival's seascapes
101 The Praetor ordering 30000 Lusitanians to be killed in a day
102 A student in a long coat staring at a map of the Paris metro
103 The building manager, trying to solve his cash-flow problems
104 The girl studying the craftsman's rings to sell in her store
105 Nationalists fighting the Damascene publisher who was French
106 A little girl gnawing at the edges of her shortbread cookies
107 The maid, imagining she'd seen the evil eye in an undertaker
108 A painstaking scientist examining rats' reactions to poisons
109 The pranking student who but beef stock in vegetarians' soup
110 A workman gazing at his letter, as he leaves with two others
111 The aged gentleman's gentleman recomputing his nth factorial
112 The staggered priest offering help to a Frenchman lost in NY
113 The druggist spending his fortune on the Holy Vase of Joseph
114 The jigsaw glue being perfected by a head of a chemistry lab
115 That gent in a black cloak donning new, tight-fitting gloves
116 Old Guyomard cutting Bellmer's sheet in 2 through the middle
117 Original fine champagne proffered to Colbert by Dom Perignon
118 A gay waltz being written by an old friend of Liszt & Chopin
119 Agreeably drowsy after lunch, M. Riri sitting at his counter
120 Gallant Amerigo learning a continent was to be named America

121 Mark Twain reading his obituary long before he'd intended to
122 The woman polishing a dagger that was Kleber's murder weapon
123 The college endowed by its ex-rector, an expert in philology
124 The single mother reading Pirandello's story of Daddi, Romeo
125 The historian who used pseudonyms to publish rubbishy novels
126 The librarian collecting proof that Hitler continues to live
127 A blind man tuning a Russian prima donna's grand piano-forte
128 A decorator making the most of the young pig's crimson bones
129 The agent trading cowries believing he'd make millions at it
130 The disappointed customer who in dyeing her hair lost it all
131 The assistant librarian using red pencil to ring opera crits
132 The lovelorn coachman who thought he's heard a rodent mewing

133 The kitchen-lads bringing up hot tasty snacks for a grand do
134 The nurse's milk jug spilt on the carpet by two naughty cats
135 A Tommy and his bride-to-be stuck between floors in the lift
136 The bookdealer who found three of Victor Hugo's original MSS
137 The English "au pair" reading an epistle from her boy-friend
138 The ordnance general who was shot in the lounge of his hotel
139 The doctor whom loaded fire-arms forced to carry out surgery
140 Safari-buffs with their native guide - posing for the camera
141 The French prof, getting pupils' vacation assignments marked
142 A beautiful Polish woman and her wee son dreaming of Tunisia
143 The judge's spouse whose pearls had cooked black in the fire
144 The cyclist struggling for recognition for his 1-hour record
145 A conscript startled on seeing his old physics schoolteacher
146 The ex-landlord dreaming of a "hero" of the traditional kind
147 A conductor rehearsing his band for 9 weeks, again and again
148 A gifted numerate, aspiring to construct a massive radiomast
149 Antipodean fans giving their idol a present of 71 white mice
150 The Spanish ex-concierge not too keen to unjam the lift door
151 Listening to an enormous phonogram, a smoker of an 89c cigar
152 A choreographer, returning to torment the loveless ballerina
153 The man who delivered wine on a trike doing the hall mirrors
154 An obviously pornographic old man waiting at the school gate
155 The botanist hoping an ivory Epiphyllum would carry his name
156 The so-called Russian who solved every brainteaser published
157 The infant Mozart, performing for Louis and Marie-Antoinette
158 A sword-swallower who on medication threw up a load of nails
159 A man who made religious articles dying of cold in the woods
160 Blind horses, deep down in the mine, hauling railway waggons
161 A urologist musing on the arguments of Galen and Asclepiades
162 A handsome pilot looking for the castle at Corbenic on a map
163 The carpenter's workman warming his hands at a woodchip fire
164 Visitors to the Orient trying to solve the magic ring puzzle
165 A ballet maestro beaten to death in the U.S.A. by 3 hoodlums
166 A princess, who said prayers at her regal granddad's bedside
167 The tenant (for 6 wks) insisting on full checks on all pipes
168 A manager who managed to be away for four months in the year
169 A lady who owned a curio shop fishing for a malosol cucumber
170 The man who saw his own death warrant in a newspaper cutting
171 The emperor thinking of the "Eagle" to attack the Royal Navy
172 Famous works improved by a celebrated artist's layer of haze
173 Eugene of Savoy having a list made of the relics of Golgotha
174 In a polka-dot dress, a woman who knitted beside the seaside
175 The Tommies enjoying girls' gym practices on a Pacific beach
176 Gedeon Spilett locating the last match in his trouser pocket
177 A young trapeze artist refusing to climb down from his perch
178 Woodworms' hollow honeycombs solidified by an Italian artist
179 Lonely Valene putting every bit of the block onto his canvas

We can interpret these lines on a thematic level of content, but also on a level where the structural arrangement of letters provides insight into the work. Thematically, each brief vignette consists of a character and an action; many scenes, though not all, seem to describe a failure or folly, lending the section as a whole an air of frustration and melancholy.

On another level, we find the Freudian word *ego* within the letter structure; we are left with questions about the possible meanings. Is the author (or translator) referring to the characters' actions as manifestations of psychology? Is the author embedding a hidden reference to his self within the work? Or is he playfully suggesting that such the challenge of creating this arrangement of letters amounts to an exercise of authorial egotism — a proud declaration of skill?

Our discovery of this structure leads us to wonder whether the original French can shed light on the meaning. We inspect Perec's original letter pattern and find in the diagonal the word *ame*, which translates most nearly as “soul” (Perec *La Vie* 292-298):

1 Pelage vainqueur d'Alkhamah se faisant couronner a Covdonaga
2 La cantatrice exilee de Russie suivant Schonberg a Amsterdam
3 Le petit chat sourd aux yeux vairons vivant au dernier etage
4 Le cretin chef d'ilot faisant preparer des tonneaux de sable
5 La femme avare ecrivant ses moindres depenses dans un cahier
6 Le faiseur de puzzles sacharnant dans ses parties de jacquet
7 La concierge prenant soin des plantes des locataires absents
8 Les parents prenommant leur fils Gilbert en hommage a Becaud
9 L'epouse du Comte libere par l'Ottomane acceptant la bigamie
10 La femme d'affaires regrettant de ne plus etre a la campagne
...
61 Le sultan Selim III atteignant huit cent quatre-vingt-huit m
62 Le sergent-chef trepassant d'une absorption massive de gomme
63 Le second du Fox decouvrant le dernier message de Fitz-James
64 Le jeune etudiant qui resta pendant six mois dans sa chambre
65 La femme du producteur partant pour un nouveau tour de monde
66 Le monteur en chauffage central réglant l'allumage au mazout
67 Le riche amateur leguant a la bibliotheque son argus musical
68 Le petit garcon classant ses collections de buvards medicaux
69 Le cuisinier comedien embauche par une tres riche Americaine
70 L'ancienne joueuse de tripot devenue une petite femme timide
...
71 Mark Twain decouvrant dans un journal sa notice necrologique
72 La secretaire polissant le poignard sous lequel perit Kleber
73 Le philologue faisant un legs au college dont il fut recteur
74 La jeune fille-mere prenant son bain en lisant du Pirandello
75 L'historien ecrivant sous des noms divers des romans ole-ole
76 Le vieux bibliothecaire accumulant les preuves qu'Hitler vit
77 L'aveugle en train d'accorder le piano da la chanteuse russe
78 Le decorateur tirant parti du squelette rouge d'un bebe-porc
79 L'impresario croyant faire fortune avec le trafic des cauris
80 La cliente abusee perdant ses cheveux en voulant les tiendre
...

Where the translation incorporated a psychological term for the self, the original included a theological one. At this level perhaps Perec is including a commentary about the transcendence of pattern within his texts; hidden messages and patterns may be one of the

author's methods of imbuing the work with his very presence.

Does this kind of thinking apply in any way to software? I do not suggest that the philosophy or methods of linguistics and information theory, or of Oulipian works, will have direct application to software. A careful reading of Perec will not inspire professional programmers to develop practical new applications or change their coding practices. Rather, I think the ideas of structure and level we have seen may help us, perhaps especially those of us who are not professional software programmers, to add another dimension to our thinking about software.

It has been suggested that, for a user, software consists only of the interface. That is, when a user thinks of a web browser, she thinks about the interface with which she interacts and not about the underlying code, structure, or network complexity. But if we examine the code more closely, we can not only learn more specifically the software's workings, but also we can understand better how it both limits and liberates us. We observed this same strand of thought both in our discussion of language and information. The Oulipo tapped into an unusual creativity by examining the constraints and rules language imposes and in so doing found a kind of liberation. Shannon noted a similar tension between freedom and rules when he observed that "when we write English half of what we write is determined by the structure of the language and half is chosen freely" (Shannon and Weaver 56). It is this very tension between freedom and rule that should be examined in software as the Oulipo and Shannon examined it in language.

The philosopher Gilles Deleuze has contrasted modern disciplinary societies of the industrial age with the current post-modern control societies of the information age. In comparing the two systems, he suggests that "it's not a question of asking whether the old or new system is harsher or more bearable, because there's a conflict in each in the ways they free and enslave us" (Deleuze 177). Perhaps this implies that it is worthwhile to examine the specifics of software, the fundamental tool of immaterial labor and of control society, to understand better the aspects of freedom and slavery that it introduces. When we seek to understand the rules of software, we seek to understand the limits it imposes as well as the creativity it allows, and we do this in the same spirit with which Oulipians examined the rules of language.

Following Deleuze's thinking, Alex Galloway's essay "Protocol, or, How Control

Exists After Decentralization" implies that we can take a literary approach to examining the rules of software:

we must descend instead into the distributed networks, the programming languages, the computer protocols, and other digital technologies that have transformed twenty-first-century production into a vital mass of immaterial flows and instantaneous transactions. Indeed, we must read the never ending stream of computer code as we read any text (the former having yet to achieve recognition as a "natural language"), decoding its structure of control as we would a film or novel. At the core of networked computing technologies is the concept of protocol (Galloway).

Heeding Galloway's suggestion, we can use the concepts of structure and level, coupled inspiration from linguistics, information theory, and Oulipian literature to learn to "read" code and protocol. We begin by finding an appropriate level at which to examine software's operation. At the highest level of interface, a web browser displays interpreted HTML:

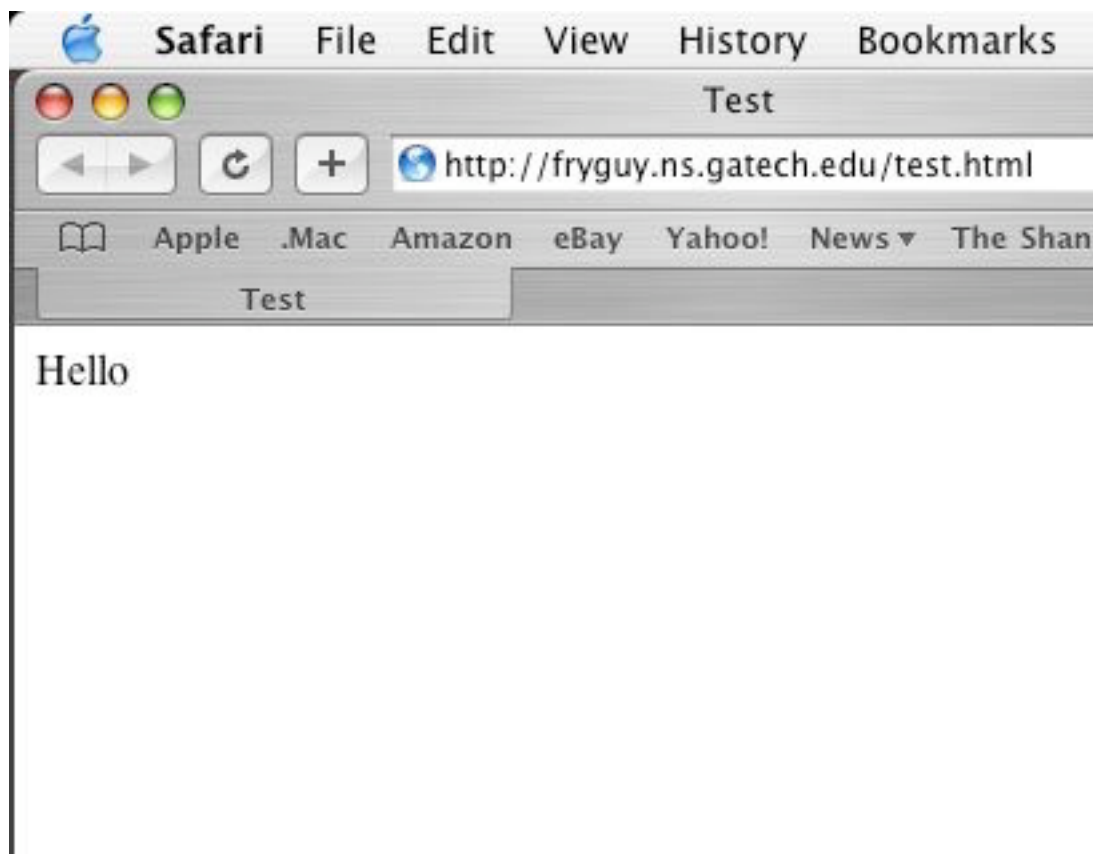


Figure 17. HTML rendered within a web browser.

Much as a reader of *Life A User's Manual* might ask, "Is there a structure that underlies this?", we can ask the same in this case. The ability to "View Source" in most web browsers allows inspection of the underlying HTML structure. Using this feature, we see

```
<html>
<head>
<title>Test</title>
</head>
<body>
Hello
</body>
</html>
```

A hallmark of software is the multiplicity of levels at which one can view its workings. Even below the level of HTML, we find other structures beneath. Using the network utility tcpflow, we can examine the HTTP protocol interaction between the browser and web server. At this level we see how the transaction is separated into a request from the client and a response from the server, and we see how the HTML document from above is in fact contained within the surrounding HTTP protocol:

```
010.000.001.033.63790-128.061.002.070.00080: GET /test.html HTTP/1.1
Host: fryguy.ns.gatech.edu
Connection: keep-alive
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/124
(KHTML, like Gecko) Safari/125
Accept: */*
Accept-Encoding: gzip, deflate;q=1.0, identity;q=0.5, *;q=0
Accept-Language: en, ja;q=0.92, ja-jp;q=0.96, fr;q=0.88, de-de;q=0.85,
de;q=0.81, es;q=0.77, it-it;q=0.73, it;q=0.69, nl-nl;q=0.65, nl;q=0.62, sv-
se;q=0.58, sv;q=0.54, no-no;q=0.50, no;q=0.46, da-dk;q=0.42

128.061.002.070.00080-010.000.001.033.63790: HTTP/1.1 200 OK
Date: Sat, 06 Mar 2004 20:58:53 GMT
Server: Apache/1.3.29 (Unix) mod_perl/1.27 mod_ssl/2.8.16 OpenSSL/0.9.6i
Last-Modified: Sat, 06 Mar 2004 20:48:05 GMT
ETag: "764ad-49-404a3905"
Accept-Ranges: bytes
Content-Length: 73
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
<html>
<head>
<title>Test</title>
</head>
<body>
Hello
</body>
</html>
```

We could continue to uncover deeper levels, until we reach a level of bit-by-bit communication. The utility tcpdump shows us a portion of this level of the communication:

```
16:14:55.271460 IP (tos 0x0, ttl 50, id 40617, offset 0, flags [DF], length:
452) fryguy.ns.gatech.edu.http > 10.0.1.33.63819: P [tcp sum ok] 1:401(400)
ack 502 win 49232 <nop,nop,timestamp 36241172 1847887278>
0x0000 4500 01c4 9ea9 4000 3206 1ae7 803d 0246 E.....@.2.....=.F
0x0010 0a00 0121 0050 f94b e7f6 f849 a16f 12b5 ...!.P.K...I.o..
0x0020 8018 c050 efc0 0000 0101 080a 0228 ff14 ...P.....(..
0x0030 6e24 85ae 4854 5450 2f31 2e31 2032 3030 n$.HTTP/1.1.200
0x0040 204f 4b0d 0a44 6174 653a 2053 6174 2c20 .OK..Date:.Sat,.
0x0050 3036 204d 6172 2032 3030 3420 3231 3a31 06.Mar.2004.21:1
0x0060 343a 3438 2047 4d54 0d0a 5365 7276 6572 4:48.GMT..Server
0x0070 3a20 4170 6163 6865 2f31 2e33 2e32 3920 :Apaches/1.3.29.
0x0080 2855 6e69 7829 206d 6f64 5f70 6572 6c2f (Unix).mod_perl/
0x0090 312e 3237 206d 6f64 5f73 736c 2f32 2e38 1.27.mod_ssl/2.8
0x00a0 2e31 3620 4f70 656e 5353 4c2f 302e 392e .16.OpenSSL/0.9.
0x00b0 3669 0d0a 4c61 7374 2d4d 6f64 6966 6965 6i..Last-Modifie
0x00c0 643a 2053 6174 2c20 3036 204d 6172 2032 d:.Sat,.06.Mar.2
0x00d0 3030 3420 3230 3a34 383a 3035 2047 4d54 004.20:48:05.GMT
0x00e0 0d0a 4554 6167 3a20 2237 3634 6164 2d34 ..ETag:..764ad-4
0x00f0 392d 3430 3461 3339 3035 220d 0a41 6363 9-404a3905"..Acc
0x0100 6570 742d 5261 6e67 6573 3a20 6279 7465 ept-Ranges:.byte
0x0110 730d 0a43 6f6e 7465 6e74 2d4c 656e 6774 s..Content-Lengt
0x0120 683a 2037 330d 0a4b 6565 702d 416c 6976 h:.73..Keep-Aliv
0x0130 653a 2074 696d 656f 7574 3d31 352c 206d e:.timeout=15,.m
0x0140 6178 3d31 3030 0d0a 436f 6e6e 6563 7469 ax=100..Connecti
0x0150 6f6e 3a20 4b65 6570 2d41 6c69 7665 0d0a on:.Keep-Alive..
0x0160 436f 6e74 656e 742d 5479 7065 3a20 7465 Content-Type:.te
0x0170 7874 2f68 746d 6c0d 0a0d 0a3c 6874 6d6c xt/html....<html
0x0180 3e0a 3c68 6561 643e 0a3c 7469 746c 653e >.<head>.<title>
0x0190 5465 7374 3c2f 7469 746c 653e 0a3c 2f68 Test</title>.</h
```

Galloway notes several interesting observations that arise from this progressive descent into the code. The structures we observe are nested, such that as we proceed down into the code, we first find HTML, then we find HTTP which contains HTML, and then we find TCP which contains HTTP. Within these structures-contained-within-structures, Galloway finds "a distributed management system that allows control to exist within immanent material relations." But Galloway's exact conclusions about the nature of protocol are not the main point here. Instead, we may see some similarity in the process of structural analysis carried out upon language and that carried out upon software. Where we find *soul* within Perek's diagonals, Galloway finds *control* within internet protocols.

Another timely example shows us a clear example of control within code. Galloway uses the example of the Domain Name System (DNS) to show how an examination of

protocol can reveal obscure power relationships. The DNS system, essential to internet operation, provides for mapping easily remembered names such as *www.gatech.edu* to the IP addresses needed for internet communication, such as *130.207.165.120*. Galloway notes that while many people consider the Internet as a vast, anarchic network, the DNS system actually imposes a rigid hierarchy. While DNS is a decentralized network, the technology underlying it concentrates ultimate authoritative control in a small number of root nodes. Galloway notes the possibility that "if some controlling authority wished to ban all Web pages ending in .org from the Internet," they could do so simply by modifying information stored in the root DNS servers. "Such a reality should shatter our image of the Internet as a vast, uncontrollable rhizome."

In September 2003, a circumstance related to the very possibility Galloway outlined came to pass. Verisign, a company which manages some of the root nodes mentioned above, implemented a new service called Site Finder. Prior to Site Finder, a DNS request for a nonexistent name would result in an error, and the end user might be presented with a message that requested site could not be found. Site Finder changed this behavior so that such a request would be answered with the IP address of a "helpful" web search page operated by Verisign itself. According to Verisign's description of Site Finder,

before this service was implemented, when a user entered a nonexistent (e.g., unregistered) domain name ending in .com or .net, his or her web browser returned an error message that contained no useful information. With the rollout of Site Finder, in the same situation users now receive a helpful web page offering links to possible intended destinations and allowing an Internet search (Verisign).

Because the Site Finder service violated long standing technical principles of the internet, and because many argued that it violated Verisign's contract to administer root nodes, the company was forced to withdraw the Site Finder service. But the incident shows the importance of "reading" code to determine its potential for liberation and limitation.

Lev Manovich wrote that "because new media is created on computers, distributed via computers, and stored and archived on computers, the logic of a computer can be expected to significantly influence the traditional cultural logic of media; that is, we may expect that the computer layer will affect the cultural layer" (Manovich 46). In a sense, Oulipian authors understood something similar about language and text; the logic of letters

and words influenced the logic of texts. Manovich seems to be pointing out a challenge that remains for new media: to understand how what underlies it affects its products.

But the multilevel nature of code poses a challenge for those who seek to read it "as we read any text." In our work with modern cryptography we saw that the logic and speed of the computer can overwhelm a human attempt to discern its workings; that is, of course, one of its strengths. Where traditional cryptography operated at the level of the letter, computerized cryptography operates at a level underneath the letter, and this level can be hard for the human mind to negotiate. For example, if we attempt to "read" and visualize the workings of the Blowfish cipher as it encrypts the word greeting, we see nothing more than a shuffling of bits. The cipher performs round after round of bit substitutions:

```
begin:      01100111011100100110010101100101 01110100011010010110111001100111
round 1:    11010101110011110001010111101000 00010111110011100110010000101001
round 2:    00010111110011100110010000101001 11001111000010011111110110100101
round 3:    11001111000010011111110110100101 11001101001101001011000101101111
round 4:    11001101001101001011000101101111 11011111010101110110111010101001
round 5:    11011111010101110110111010101001 11010011011010110100011011110011
round 6:    11010011011010110100011011110011 10100101011010100001110000011110
round 7:    10100101011010100001110000011110 00100110001011101100011000001111
round 8:    00100110001011101100011000001111 00110110111001011010010001101011
round 9:    00110110111001011010010001101011 01001111110000111100100111110011
round 10:   01001111110000111100100111110011 01110000110101100100111010111010
round 11:   01110000110101100100111010111010 11010001011000000001111110001101
round 12:   11010001011000000001111110001101 11110011011101001110011010000101
round 13:   11110011011101001110011010000101 00101010000011011000100110011111
round 14:   00101010000011011000100110011111 00111101010011000000001110101001
round 15:   00111101010011000000001110101001 00000101110101001100001000001000
round 16:   00000101110101001100001000001000 00011100010011100101010110011010
end:       01101110111001101101000001100111 00011100010011100101010110011010
key: cookiesaretasty
ciphertext: n??gNU?
```

While perhaps interesting to observe and understand modern encryption, this level does not seem to offer us, the human, the proper scale to interpret code as we would a film or novel. A very significant challenge in the interpretation of code and software seems to involve finding the proper level on which to perform analysis.

Another challenge in working with code in a critical way arises from the differences we have observed between programming languages and natural languages. We have seen Saussure's ideas about how the relationships among signs gives rise to an constellation of related signs; recall the example of the Visual Thesaurus. Some Oulipian works seem to exploit the human ability to work successfully within such a framework of associations. When

Mathews's Algorithm juxtaposes words in orderly but unusual combinations, the human mind can play along. But compilers have no such capacity for play. So any attempt to mimic Oulipian techniques, or other techniques of language manipulation, would likely fail because of programming language's inflexibility.

Perhaps one answer to the human difficulty in scrutinizing computer code is to enlist software in the task of scrutinizing itself. In his book *Behind The Blip: Essays on the Culture of Software* Matthew Fuller suggests the concept of "speculative software," which he describes as "software whose job is to reflexively investigate itself as software," providing a window into its usually obscure inner workings. This type of software, he says, "has been suggested as being software that explores the potentiality of all programming" (Fuller 30). (Here it is interesting to note in passing a comparison with the Oulipo, which within its very name expressed its desire to explore the potentiality of literature.)

A brief review of an existing work of speculative software may help illustrate the idea. I/O/D's Web Stalker is an implementation of an HTTP client, though its function differs from mainstream browsers; instead of rendering web pages in the conventional way, Web Stalker shows activity that most browsers seek to hide. Web Stalker shows raw HTML content, HTTP protocol headers, maps of links to other surrounding documents, and schematics showing objects on a page:

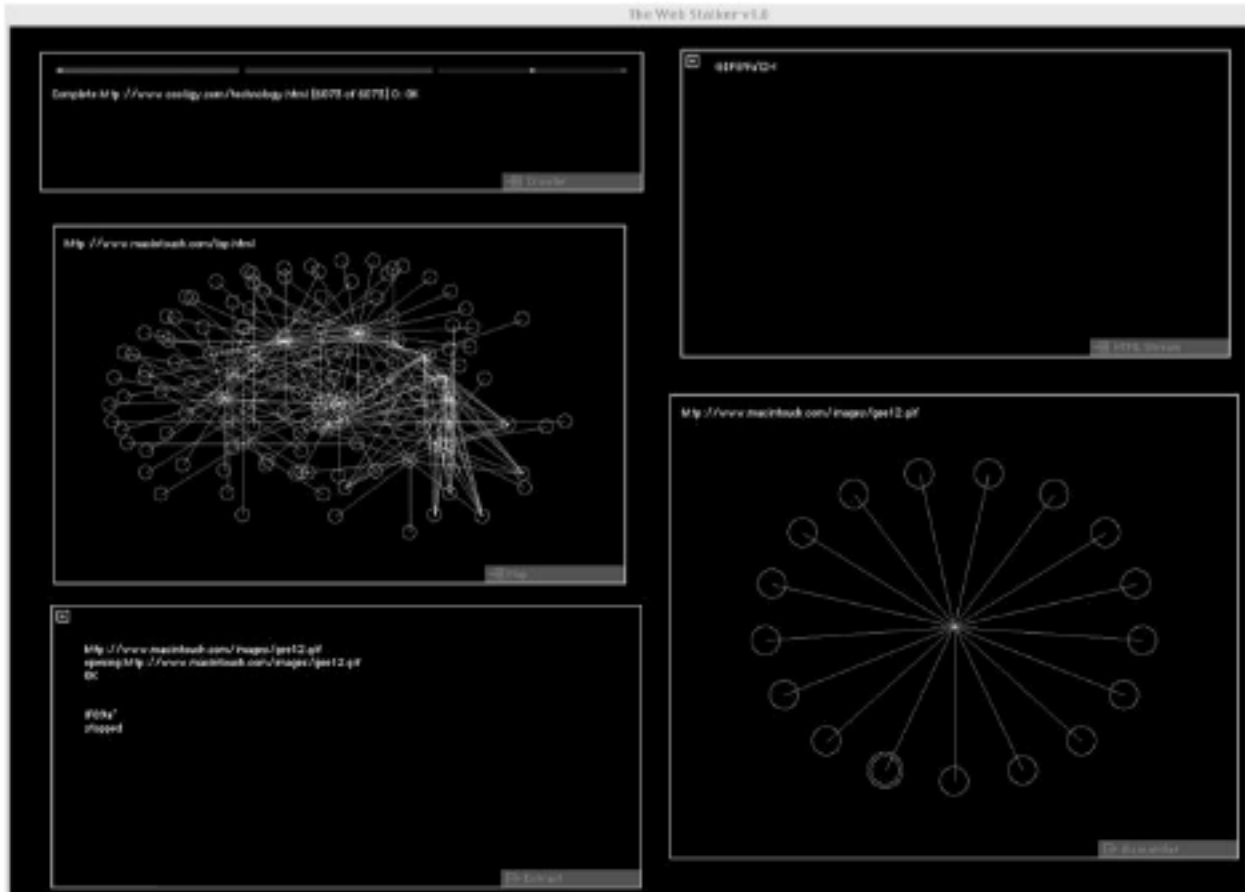


Figure 18. The Web Stalker

Web Stalker is speculative software in that it turns its capability as a browser into an exploration of the principles and codes underlying the web. It harnesses the power of software to illustrate its own structures at a level appropriate for human inspection. Perhaps this is an odd sort of self-referential idea: we must increasingly rely on software code to provide us with the ability to analyze software itself. Then again, perhaps this is not terribly different from Oulipian literature: their works employed language as the primary tool in their effort to analyze language itself.

In his essay "The Question Concerning Technology," Martin Heidegger wrote that the "essence of technology is by no means anything technological. We shall never experience our relationship to the essence of technology so long as we merely conceive and push forward the technological" (Heidegger 4). In other words, Heidegger suggested the need to step outside the closed, self-legitimizing world of software to "experience our relationship to the essence of technology." In this study we have turned to ideas from

linguistics, information theory, and literature to understand essential aspects of code and software. It is my hope that this approach will provide others with background, ideas, and inspiration to continue efforts to understand code. If code is to be “the language of our time,” it deserves as thorough an inspection as language has already received.

APPENDIX A: CODEWORK

Mathews' Algorithm

Mathews' Algorithm contrasts with Queneau's *100,000,000,000 Poems* in how it handles the multiple possibilities combinatorial literature can generate. As its name shows, Queneau's approach embraces the vast number of possibilities, but Mathews takes a different tack. His algorithm only produces two variations out of many possible ones. In Mathews' words, combinatoric algorithms for literature "can be divided into two complementary categories. The first category is that of the proliferation of possibilities; the second is that of their reduction" (Motte 139).

In the example cited in this paper, the algorithm uses four sentences of four words each to construct two new sentences – from slightly different applications of the same algorithm. But there are in fact 256 possible combinations of these words, since four separate choices of four possible words yields 4^4 possibilities.

The following code generates all 256 possible sentences, shuffles their order, and presents one per second for inspection.

```
use List::Util 'shuffle';

$sets = [
    ['Truth', 'left', 'him', 'cold.'],
    ['Wealth', 'made', 'her', 'glad.'],
    ['Work', 'turned', 'you', 'sour.'],
    ['Love', 'kept', 'me', 'free.']] ];

for ($i=0; $i<4; $i++) {
    for ($j=0; $j<4; $j++) {
        for ($k=0; $k<4; $k++) {
            for ($m=0; $m<4; $m++) {
                push(@sentences, join(' ', ($sets->[$i]->[0],
$sets->[$j]->[1], $sets->[$k]->[2], $sets->[$m]->[3])));
            }
        }
    }
}

@shuffled = shuffle(@sentences);
foreach (@shuffled) {
    print "$_\n";
    sleep 1;
}
```

The code produces the following output:

```
$ perl ma7.pl
Wealth made her glad.
Wealth turned him sour.
Wealth turned you free.
Truth made me cold.
Work made you glad.
Truth turned me sour.
Work kept you sour.
Truth left him glad.
Work kept you free.
Work made him sour.
Truth turned him glad.
Wealth turned her sour.
Wealth left him free.
Love made me free.
Love made him glad.
Love turned you free.
Love made you glad.
Work kept me cold.
Wealth left me sour.
Wealth kept you glad.
Truth left him sour.
Love made me sour.
.....
```

Perec's Diagonals

I noticed the middle diagonal pattern of the letter *g* in the “Compendium” of Perec’s *Life A User’s Manual*, but I was unable to see any other letter patterns. (Despite the fact that there are two other very similar diagonal letter patterns, they still were not apparent to my eye!) I composed the following code to examine each group of 60 lines, masking all but one letter at a time.

```

# display file as it is
open(IN, "first60.txt");
while (<IN>) {
    print $_;
}
close IN;
sleep 2;

# now display the same file, masking all but one letter at a time
foreach $letter (a..z) {
    open(IN, "second60.txt");
    while (<IN>) {
        /^(\d+\s+)(.*)/;
        $num = $1;
        $sentence = $2;
        $sentence =~ s/[^$letter]/ /gi;
        print $num . $sentence . "\n";
    }
    close IN;
    select undef, undef, undef, 0.5;
}

open(IN, "second60.txt");
while (<IN>) {
    print $_;
}
close IN;
sleep 2;

```

For the first 60 lines, masking all except the letter e, the code produces the following output.

```

1     e                                     e e
2     e           e           e           e e e
3     e e           e           e e           e e           e e
4     e           e           e           e           e
5     e e           e e e e           e           e
6     e e e e           e           e           e e
7     e e e e           e           e e           e
8     e e           e           e           e e           e
9     e           e e           e           e e e e           e
10    e           e e           e           e e           e
11    e           e           e           e           e
12    e           e           e e           e e           e e
13    e           e           e e e e           e           e
14    e           e           e           e e           e e           e
15    e           e           e           e           e
16    e e           e           e           e           e
17    e e           e           e           e e           e
18    e           e           e           e           e
19    e           e e           e ee           e           e e           e
20    e e           e           e           e           e e
....

```

Blowfish Algorithm

The following code examines the raw bit manipulation of the Blowfish algorithm. Our example showed the difficulty of examining software at the level of raw bits; it is very difficult to gain understanding of software by examining it at this level.

```
use Crypt::Blowfish_PP;

my $key = 'cookiesaretasty';
my $plaintext = 'greeting';

$blowfish=new Crypt::Blowfish_PP($key);
$ciphertext=$blowfish->encrypt($plaintext);

print "key: $key\n";
print "ciphertext: $ciphertext\n";
```

The code also relies upon *print* statements placed within the encryption subroutines of the Blowfish module to output the state of the bits at each round of encryption. The code produces the following output.

```
begin:      01100111011100100110010101100101 01110100011010010110111001100111
round 1:    11010101110011110001010111101000 00010111110011100110010000101001
round 2:    00010111110011100110010000101001 11001111000010011111110110100101
round 3:    11001111000010011111110110100101 11001101001101001011000101101111
round 4:    11001101001101001011000101101111 11011111010101110110111010101001
round 5:    11011111010101110110111010101001 11010011011010110100011011110011
round 6:    11010011011010110100011011110011 10100101011010100001110000011110
round 7:    10100101011010100001110000011110 00100110001011101100011000001111
round 8:    00100110001011101100011000001111 00110110111001011010010001101011
round 9:    00110110111001011010010001101011 01001111110000111100100111110011
round 10:   01001111110000111100100111110011 01110000110101100100111010111010
round 11:   01110000110101100100111010111010 11010001011000000001111110001101
round 12:   11010001011000000001111110001101 11110011011101001110011010000101
round 13:   11110011011101001110011010000101 00101010000011011000100110011111
round 14:   00101010000011011000100110011111 00111101010011000000001110101001
round 15:   00111101010011000000001110101001 00000101110101001100001000001000
round 16:   00000101110101001100001000001000 00011100010011100101010110011010
end:        01101110111001101101000001100111 00011100010011100101010110011010
key: cookiesaretasty
ciphertext: n??gNU?
```

WORKS CITED

- Ars Electronica Center Linz. *Code — The Language of Our Time: Ars Electronica 2003*. Ostfildern: Hatje Cantz, 2003.
- AT&T Advertising. web. Available: <http://www.att.com/presskit/worldsnetcom/>. March 17 2004.
- Carnivore. web. Available: <http://rhizome.org/carnivore/>. March 16 2004.
- Web Stalker. web. Available: <http://bak.spc.org/iod/iod4.html>. March 17 2004.
- Bellos, David. *Georges Perec : A Life in Words : A Biography*. 1st U.S. ed. Boston: D.R. Godine, 1993.
- Bénabou, Marcel. *Why I Have Not Written Any of My Books*. Trans. David Kornacker. French Modernist Library. Lincoln: University of Nebraska Press, 1996.
- Calvino, Italo. *If on a Winter's Night a Traveler*. Trans. William Weaver. 1st ed. New York: Harcourt Brace Jovanovich, 1981.
- Deleuze, Gilles. *Negotiations, 1972-1990*. Trans. Martin Joughin. European Perspectives. New York: Columbia University Press, 1995.
- Enzensberger, Hans Magnus, Reinhold Grimm, and Bruce Armstrong. *Critical Essays*. New York: Continuum, 1982.
- Foucault, Michel, and Paul Rabinow. *The Foucault Reader*. 1st ed. New York: Pantheon Books, 1984.
- Friedman, William F. *Six Lectures Concerning Cryptography and Cryptanalysis*. Laguna Hills, California: Aegean Park Press.
- Fromkin, Victoria, and Robert Rodman. *An Introduction to Language*. 5th ed. Fort Worth: Harcourt Brace Jovanovich College Publishers, 1993.
- Fuller, Matthew. *Behind the Blip : Essays on the Culture of Software*. Brooklyn, NY, USA: Autonomedia, 2003.
- Galloway, Alex. *Protocol, or, How Control Exists after Decentralization*. Available: <http://slash.interactivist.net/analysis/03/02/03/1223228.shtml>.
- Garfinkel, Simson. *PGP: Pretty Good Privacy*. Sebastopol, CA: O'Reilly & Associates, 1995.
- Gere, Charlie. *Digital Culture*. London: Reaktion Books, 2002.

- Gibson, William. *Neuromancer*. New York: Ace Books, 1984.
- Harel, David. *Computers Ltd. : What They Really Can't Do*. Oxford ; New York: Oxford University Press, 2000.
- Heidegger, Martin. *The Question Concerning Technology, and Other Essays*. 1st ed. New York: Harper & Row, 1977.
- Hillis, W. Daniel. *The Pattern on the Stone : The Simple Ideas That Make Computers Work*. 1st ed. New York: Basic Books, 1998.
- Jakobson, Roman, and Stephen Rudy. *Selected Writings*. The Hague: Mouton, 1971.
- Kay, Lily E. *Who Wrote the Book of Life? : A History of the Genetic Code*. Writing Science. Stanford, Calif.: Stanford University Press, 2000.
- Levy, Steven. *Crypto : How the Code Rebels Beat the Government--Saving Privacy in the Digital Age*. New York: Viking, 2001.
- Manovich, Lev. *The Language of New Media*. Cambridge, Mass.: MIT Press, 2001.
- Mathews, Harry, Alastair Brotchie, and Raymond Queneau. *Oulipo Compendium*. Atlas Arkhive 6. London: Atlas Press, 1998.
- Motte, Warren F. *Oulipo : A Primer of Potential Literature*. French Literature Series. 1st Dalkey Archive ed. Normal, Ill.: Dalkey Archive Press, 1998.
- Perec, Georges. *La Vie, Mode D'emploi*. Paris: Hachette, 1978.
- Perec, Georges. *Life, a User's Manual*. Boston: D.R. Godine, 1987.
- Perec, Georges. *A Void*. London: Harvill, 1994.
- Plan of 11, Rue Simon-Crubbellier, La Vie Mode D'emploi*. web. Available: <http://130.238.50.3/ilmh/Ren/dig-perec-plan.htm>. March 21 2004.
- Poe, Edgar Allen. *The Project Gutenberg Etext of The Works of Edgar Allan Poe V. 1. Project Gutenberg*. web. Available: <http://www.gutenberg.net/browse/BIBREC/BR2043.HTM>. March 21 2004.
- Queneau, Raymond, and Barbara Wright. *Exercises in Style*. London: J. Calder: Distributed by Calder and Boyars, 1979.
- Queneau, Raymond, and Oulipo (Association). *Oulipo Laboratory : Texts from the Bibliothèque Oulipienne*. London: Atlas, 1995.
- Rosenheim, Shawn. *The Cryptographic Imagination : Secret Writing from Edgar Poe to the Internet*. Baltimore, Md.: Johns Hopkins University Press, 1997.

- Saussure, Ferdinand de, et al. *Course in General Linguistics*. 1st McGraw-Hill paperback ed. New York: McGraw-Hill Book Co., 1966.
- Schneier, Bruce. *Applied Cryptography : Protocols, Algorithms, and Source Code in C*. 2nd ed. New York: Wiley, 1996.
- Shannon, Claude Elwood, et al. *Claude Elwood Shannon : Collected Papers*. New York: IEEE Press, 1993.
- Shannon, Claude Elwood, and Warren Weaver. *The Mathematical Theory of Communication*. Urbana,: University of Illinois Press, 1964.
- Singh, Simon. *The Code Book : The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. 1st Anchor Books ed. New York: Anchor Books, 2000.
- Sturrock, John. *Structuralism. Paladin Movements and Ideas*. London, England: Paladin, 1986.
- Thomas, Douglas. *Hacker Culture*. Minneapolis: University of Minnesota Press, 2002.
- Verisign. *Verisign's Sitemap Implementation*. web. Available: www.verisign.com/resources/gd/sitemap/implementation.pdf. March 21 2004.
- Visual Thesaurus*. web. Available: <http://www.visualthesaurus.com/>. March 21 2004.
- What Is Machine Language?* web. Available: http://www.webopedia.com/TERM/m/machine_language.html. March 21 2004.