THE STYLIST:
A PASCAL PROGRAM FOR ANALYZING PROSE STYLE

by

Thomas C. Cool

June 1987

Thesis Advisor: Thomas Wu

Approved for public release; distribution is unlimited.
DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.
The Stylist is a Waterloo Pascal program that analyzes the style of English prose. A "style checker", The Stylist pertains to Computational Stylistics and Computer Assisted Composition Instruction (CACI). The Stylist creates an affective model of the text based upon the following characteristics of its component words: etymology, tangibility, difficulty, emotional connotation and vigor. The Stylist then compares this model to the standards of fiction or nonfiction texts and reports results and recommendations to the user. The Stylist also creates a concordance of the user's input text using a new data structure called a Concordance Search Tree (CST). A CST is a binary search tree with a linked list threaded through it recording the order of the use of each word. An inorder traversal of the tree, with a traversal of the linked list during each visit, creates a concordance. This thesis also reviews...
related literature and programs.
The Stylist:
A Pascal Program for Analyzing Prose Style

by

Thomas C. Cool
Lieutenant, United States Navy
B.A., English, The Pennsylvania State University, 1976

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1987
ABSTRACT

The Stylist is a Waterloo Pascal program that analyzes the style of English prose. A style checker, The Stylist pertains to Computational Stylistics and Computer Assisted Composition Instruction (CACI). The Stylist creates an affective model of the text based upon the following characteristics of its component words: etymology, tangibility, difficulty, emotional connotation and vigor. The Stylist then compares this model to the standards of fiction or nonfiction texts and reports results and recommendations to the user.

The Stylist also creates a concordance of the user's input text using a new data structure called a Concordance Search Tree (CST). A CST is a binary search tree with a linked list threaded through it recording the order of the use of each word. An inorder traversal of the tree, with a traversal of the linked list during each visit, creates a concordance.

This thesis also reviews related literature and programs.
# TABLE OF CONTENTS

## I. INTRODUCTION ............................................. 9
A. BACKGROUND .................................................. 9
   1. Computational Stylistics ............................... 9
   2. Computer Assisted Composition Instruction (CACI) .... 11
B. SCOPE OF THE STYLIST ................................... 12

## II. RESEARCH FOR THE STYLIST ............................... 13
A. LITERATURE REVIEW ......................................... 13
   1. Affective Tone ........................................... 13
   2. Readability and Sentence and Word Length ............. 14
B. PRODUCT TESTING ............................................ 15
   1. RightWriter ............................................. 15
   2. PC-Style ................................................ 16
   3. Writer's Workbench .................................... 16
C. CONCLUSIONS BASED UPON RESEARCH ....................... 17

## III. DESIGN OF THE STYLIST .................................. 18
A. CHARACTERISTICS OF WORDS ................................. 18
   1. Etymology .............................................. 18
   2. Tangibility ............................................. 19
   3. Difficulty ............................................. 19
   4. Emotional Connotations ................................ 20
   5. Vigor ................................................... 20
B. THE NEED FOR A DICTIONARY ................................ 21
C. THE COPYDIX PROTOTYPES ................................. 21
   1. The Benefit of Preorder Storage ....................... 22
   2. The Need for AVL ...................................... 22
   3. Storage requirements .................................. 22
   4. Text Processing ........................................ 22
5. Building a Dictionary ........................................... 22

D. CONCORDANCE SEARCH TREE (CST) .......................... 23
   1. Original Concept ............................................. 23
   2. The Tri1 - Tri9 Prototypes .................................. 24

E. DESIGN OF THE STYLIST ...................................... 25
   1. The Reader Module .......................................... 25
   2. The Researcher Module ..................................... 25
   3. The Reporter Module ....................................... 25
   4. Low Level Design ........................................... 25

IV. IMPLEMENTATION OF THE STYLIST ............................ 27
   A. CODING ...................................................... 27
   B. BUILDING THE DICTIONARY .................................. 28

V. TESTING OF THE STYLIST ....................................... 29
   A. INITIAL TEST DATA SET ..................................... 29
   B. FIELD TESTS ............................................... 30
      1. Field Test One: The Ape Test ............................ 31
      2. Field Test Two: The Writing Class ...................... 31

VI. CONCLUSIONS .................................................. 33

APPENDIX A: SUGGESTED EXTENSIONS TO THE STYLIST .......... 35
   1. ABBREVIATIONS ............................................. 35
   2. TRANSLATION TO TURBO PASCAL ............................ 35
   3. IMPROVEMENTS OF THE CONCORDANCE ....................... 35
   4. PARTS OF SPEECH ........................................... 36
   5. PASSIVE VOICE .............................................. 36
   6. CORRECTION OF POOR DICTION AND GRAMMAR ............... 36

APPENDIX B: USER'S MANUAL ....................................... 38
   1. INTRODUCTION .............................................. 38
      a. Etymology .................................................. 38
      b. Tangibility ............................................... 39
      c. Difficulty .................................................. 39
      d. Emotional Connotations ................................. 40

6
2. USING THE STYLIST
   a. Name of Intext
   b. Fiction or NonFiction
   c. Report Frequency
   d. Concordance
   e. Number of Words in Intext
   f. Expand the Dictionary
   g. The End of Execution
3. THE MEANING OF YOUR REPORT
   a. Measures of Length
   b. Measures of Word Characteristics
4. MAINTENANCE OF THE STYLIST
   a. Care of the Dictionary
   b. Changing the Code of The Stylist

APPENDIX C: THE CODE OF THE STYLIST
APPENDIX D: THE CODE OF DIXSPLIT AND DIXJOIN
APPENDIX E: REPRESENTATIVE RUNS
LIST OF REFERENCES
BIBLIOGRAPHY
INITIAL DISTRIBUTION LIST
<table>
<thead>
<tr>
<th>List Number</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HEISE WORDS VALUES</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>EXAMPLES OF NATIVE AND BORROWED WORDS</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>EXAMPLES OF TANGIBLE AND INTANGIBLE WORDS</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>EXAMPLES OF WORDS OF VARIOUS DIFFICULTY</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>EXAMPLES OF WORDS OF VARIOUS EMOTIONAL CONNOTATIONS</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>EXAMPLES OF WORDS OF VARIOUS VIGOR</td>
<td>21</td>
</tr>
<tr>
<td>7</td>
<td>EXAMPLES OF ETYMOLOGY</td>
<td>38</td>
</tr>
<tr>
<td>8</td>
<td>EXAMPLES OF TANGIBILITY</td>
<td>39</td>
</tr>
<tr>
<td>9</td>
<td>EXAMPLES OF DIFFICULTY</td>
<td>40</td>
</tr>
<tr>
<td>10</td>
<td>EXAMPLES OF EMOTIONAL CONNOTATION</td>
<td>40</td>
</tr>
<tr>
<td>11</td>
<td>EXAMPLES OF VIGOR</td>
<td>41</td>
</tr>
<tr>
<td>12</td>
<td>WORD LENGTH</td>
<td>48</td>
</tr>
<tr>
<td>13</td>
<td>SENTENCE LENGTH</td>
<td>48</td>
</tr>
<tr>
<td>14</td>
<td>MODULATION</td>
<td>49</td>
</tr>
<tr>
<td>15</td>
<td>CHARACTERIZATIONS OF ETYMOLOGY</td>
<td>49</td>
</tr>
<tr>
<td>16</td>
<td>CHARACTERIZATIONS OF DIFFICULTY</td>
<td>50</td>
</tr>
<tr>
<td>17</td>
<td>CHARACTERIZATIONS OF TANGIBILITY</td>
<td>50</td>
</tr>
<tr>
<td>18</td>
<td>CHARACTERIZATIONS OF EMOTIONALITY</td>
<td>51</td>
</tr>
<tr>
<td>19</td>
<td>CHARACTERIZATIONS OF VIGOR</td>
<td>52</td>
</tr>
<tr>
<td>20</td>
<td>CODES OF THE DICTIONARY</td>
<td>53</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

A. BACKGROUND

1. Computational Stylistics

Computational Stylistics is the computer-assisted study of literary style. Computational Linguistics is the computer-assisted study of language itself. These disciplines are the automated subsets of Statistical (or Quantitative) Stylistics and Linguistics, which are the mathematical studies of style and language. All of these disciplines involve quantifying aspects of language and then manipulating these quantities in an attempt to gain insight into how and why the language works.

The history of stylistics can be traced back to 1851, when Augustus de Morgan suggested that word-length could prove to be a distinctive trait of a writer's style. [Ref. 1] This suggestion prompted T.C. Mendenhall, an American geophysicist, to investigate whether mean word-length could resolve authorship problems such as those posed by some of the disputed Shakespeare plays or the letters written under the pen name of Junius. Working in the late 19th century, Mendenhall analyzed the word-lengths of some two million words from various periods of English literature, using a primitive tabulating device that spit out reels of paper. His results, however, proved little.

In the early 20th century, another possible characteristic of style, the frequency distribution of words, came under wide-spread investigation. G.K. Zipf postulated a "Rank-Frequency Law", by which a ranking of the use of words in a text would show a constant decrease from the most-used word down to the least-used word. [Ref. 2] Other scholars, such as G. U. Yule, investigated such aspects as the richness of a writer's vocabulary and the length of sentences. [Ref. 3: pp. 363 - 390] By the 1950s and 1960s, the application of statistical methods to the study of literature had reached new heights of sophistication and complexity. Some scholars were investigating the significance of the ratio of verbs to adjectives and others were applying rigorous statistical techniques. Despite the many fascinating insights offered by these lines of inquiry, their final significance and scientific credibility remained a question of much dispute. Enemies of the new disciplines included Norm Chomsky, the pioneer of formal languages, who argued that writing is a very human activity which involves a great deal of chance, and as such is not easily quantified. [Ref. 4]
Computational stylistics began as a natural outgrowth of statistical stylistics. With the spread of computer access some thirty years ago, some scholars quickly realized the potential for their use in literary studies:

"I first heard of computers in 1955 when my wife (a mathematician) told me that she was going to work for an oil company as a program analyst . . . It took me a while to understand her flowcharts and computer programs, but when I did, I realized that a computer could be used to solve other problems as well. Since then, I have used a computer for numerous applications relating to my work in the English Department of Cleveland State University, and have taught many others . . . the power of electronic data processing in the study of language and literature." [Ref. 5]

The first challenge to the field of Computational Stylistics was to translate literary texts to electronic data. Much of the early literature of the field is devoted to this basic problem. After the first two decades, corpora, or bodies of literary texts in machine readable form, had been developed. The Lancaster-Oslo/Bergen (LOB) Corpus is a structured collection of 500 two-thousand-word texts of written British English. An American version of this effort is the Brown Corpus, constructed by Brown University.¹

In addition, many literary classics have been painstakingly entered by keyboard. They are available from the Oxford Computing Laboratory at Oxford University and the American Philological Association. [Ref. 5] With the increasing capability and decreasing cost of optical readers, the problem of translating literary text to electronic data should become trivial.

The usefulness of computers to statistical stylistics quickly became indisputable. Scholars used to labor to copy down on 3" X 5" cards each occurrence of every noun used in a Shakespeare play and then sort these cards by hand. [Ref. 6: pp. 33 - 50] Now a computer program could manipulate the text in any number of ways within minutes. One of the other prominent successes of the use of computers in literary studies was the automation of the process of forming concordances. A concordance is an alphabetical listing of all the significant words used in a text, together with the enclosing phrase. Before the advent of computers, scholars laboriously built concordances of the Bible and the plays of Shakespeare. The power of the computer made possible giant concordance-building projects such as one for the

¹Electronic forms of both of these corpora are available from: The Norwegian Computational Center for Humanistic Research, P.O. Box 53, University of Bergen N-5014, Bergen, Norway
Dead Sea Scrolls. The Centro Atomazione Analisi Linguista in Gallarate, Italy, used computers to build this concordance, which helped to resolve some of the missing or obliterated words. [Ref. 7]

One of the next important challenges to computational linguistics was the issue of disambiguation. Many words have more than one meaning and can belong to more than one parts of speech. An example in English is the word "flies" in the following two sentences:

Time flies like an arrow.
Fruit flies like an apple.

A human reader can easily decide that "flies" in the first sentence is a verb and that "flies" in the second sentence is a noun. This decision is disambiguation. In order to compute such characteristics as verb-noun ratio, it is first necessary to disambiguate the parts of speech. Recent advances in computational linguistics have led to programs that can do this with more than 90% success. [Ref. 8] [Ref. 9: pp. 139 - 150] [Ref. 10]

Scholars continue to argue about the usefulness of statistical stylistics. Efforts to discover and prove laws of distribution have not met with unchallenged success. Nevertheless, the work done in the past 130 years has laid some groundwork for the use of computers to study and analyze written prose. All of the professorial attempts thus have led to success at a humbler level: the tutorial. The history of stylistics is now culminating at this level, in the form of Computer Assisted Composition Instruction.

2. Computer Assisted Composition Instruction (CACI)

CACI is a new discipline which has begun to emerge in the past decade. The success of word processing programs created a growing population of people who expected computers to help them to write. Enterprises such Bell Laboratories and centers of learning such as The Pennsylvania State University began to borrow from the field of stylistics as they developed computer programs that would help students and workers plan, write and edit prose. Several universities, such as Colorado State University, successfully instituted CACI as part of their composition curriculum. [Ref. 11] The earliest CACI programs required a mainframe or at least a minicomputer.
With the expansion of the power of personal computers, however, software houses such as DecisionWare Inc. began to write CACI programs for that environment. Over 10,000 copies of DecisionWare's RightWriter are now in use at corporations, agencies and universities around the world, according to their advertisements. Given that word processing remains the single most common use of personal computers, as one megabyte of main storage becomes commonplace on new personal computers, style checkers such as RightWriter may soon become as popular as spell checkers have become in the past five years.

B. SCOPE OF THE STYLIST

The Stylist is a style checker akin to Bell Laboratory's Writers Workbench and DecisionWare Inc.'s RightWriter. It does not help the user to plan or to compose his product. Instead, it analyzes the finished product and provides that analysis to the user for his consideration toward revision. The Stylist does not determine parts of speech, as does Writer's Workbench; nor does it suggest alternatives to hackneyed phrases, as does RightWriter. What the Stylist does instead is to analyze some of the characteristics of the words used in the text. By doing so, The Stylist can distinguish between short, vigorous, germanic, emotional texts and long, lazy Latin texts. The success of this effort could and should be incorporated into more extensive programs such as Writer's Workbench.

In addition, The Stylist solves the traditional problem of creating a concordance in a novel and elegant way. This solution is optimal for the personal computer environment of the coming few years.
II. RESEARCH FOR THE STYLIST

A. LITERATURE REVIEW

When I first conceived of The Stylist, I believed that a "style checker" was a completely original idea. Little did I know that major universities and great corporations had been working on the problem for decades. My research began with the Encyclopedia Brittanica, where I discovered the existence of the fields of stylistics and statistical linguistics. Using these as subjects, I searched the Dewey decimal system and several automated data bases for titles. The books I discovered referenced the two journals which publish many of the pertinent articles: Computational Linguistics, and Computers and the Humanities. These, in turn, carried advertisements for some of the currently available software. They also identified the current centers of learning, some of which I contacted for guidance and information. The experience of researching the field demoted me from its inventor to its lowliest tyro. It also subjected me to many ideas, some of which I incorporated into The Stylist. The bibliography lists the sources which I unearthed which may benefit other students. In this discussion, I will cover only those articles which directly influenced the development of The Stylist.

1. Affective Tone

C.W. Anderson and G.E. McMaster reported on a program called PSA (Psychological Semantic Analysis) which analyzes the emotional tone of a text. [Ref. 12] They built upon the work of D.R. Heise, who ascribed values of "Evaluation, Activity and Potency" to the thousand most commonly-used words in the English language. [Ref. 13] PSA allows the user to enter his text one line at a time; if any word is ambiguous, PSA asks the user to disambiguate. PSA then matches these words to the 1000 Heise-word dictionary, adding values up to create a profile of the user's text. Table 1 helps clarify the meanings of the three Heise-word categories.

After building and testing PSA, Anderson and McMaster concluded that: "The affective tone of whole passages can be measured by computer-assisted collection of . . . scores of those words . . . for which Heise (1965) has provided semantic meanings." [Ref. 12]
The original idea of The Stylist envisioned doing just such a count of words chosen for their frequency of use and their characteristics such as etymology, emotional connotation and vigor. Anderson and McMaster convinced me that this method had a sound psychological base. Moreover, Heise’s categories of Evaluation and Activity seemed to correspond directly to my envisioned categories of emotional connotation and vigor. His category of Potency, however, struck me and continues to strike me as distressingly Freudian. Because The Stylist would not use that category and because it would use others, I decided not to borrow Heise’s words and values, but rather to build my own dictionary and ascribe my own values. This seemed a reasonable approach, given the statement by Anderson and McMasters that “there is much commonality in the emotional response of different persons to words and objects.”

2. Readability and Sentence and Word Length

Two of the most common stylistic measures are sentence length and word length. The most common use of these measures is to determine readability. The basic idea is that short sentences are easy to read and long sentences are hard to read. Various formulae, such as the Kincaid, the Automated Readability Index (ARI), the
Coleman-Liau, and the Flesch Reading Ease Score, [Ref. 10] attempt to determine readability as a straight-forward function of sentence and word length. These formulae have had considerable impact on the teaching of English. Most style checkers use them. Some, like RightWriter, admonishes the writer any time a sentence grows beyond 22 words or so. Having read a great deal of Victorian novels and Madison Avenue copy, I'm well aware that the ideal 20th century sentence is short. It's vigorous. Easy to understand. Maybe even dispenses with its verb as it rushes toward its punctuation! The mechanization of this modern tendency into simplistic formulae, however, seems to me pernicious. A sentence should be the unit of a complete idea. If we limit ourselves to short sentences, we may be limiting ourselves to small ideas. The average sentence of the prose of the Age of Reason was 45 words. If Samuel Johnson were to live today, no one would let him finish a sentence. My distrust of these formulae redoubled when I read "Readability is a Four-Letter Word," by Jack Selzer. [Ref. 1-] In this article, Selzer forcefully makes the points that readability is a subjective quality which is greatly influenced by factors such as arrangement of ideas, reader background and interest, and difficulty of vocabulary. The Stylist does not use readability formulae. It does flag sentences which seem to be run-on. More importantly, it gauges the difficulty of the vocabulary. If the writer uses hard words, long words, latinate words and long sentences, The Stylist warns him, particularly if the work is fiction.

B. PRODUCT TESTING

1. RightWriter

As part of my research, I procured a copy of RightWriter2 and tested it with some of my own writing. Overall, the program impressed me with its capabilities and its engineering. RightWriter reproduces the users text with inserted comments. Its constant challenge of long or complex sentences forced me to reexamine each case in particular. Several lengthy sentences became two short ones. It never let me begin a sentence with "But," a foible of mine. It applauded my writing when it was strong and it derided it when it was pompous. One attractive feature was the production of an alphabetical word list with frequency of occurrence. This feature demonstrated my over-attachment for the words, "ancient" and "thousand." RightWriter also suggested

---

2RightWriter (tm) is a product of Decisionware Inc., 2033 Wood St., Suite 218, Sarasota, FL 33577, (813) 952-9211.
substitutes for hackneyed or useless phrases such as "the fact that". Overall, RightWriter proved itself extensively useful.

2. PC-Style

PC-Style is a much less ambitious program than RightWriter. PC-Style required only 40K of RAM, compared to 192K for RightWriter. PC-Style is also much cheaper, costing only $29.95 as compared to RightWriter's $95. I tested it with the same test data set I had used on RightWriter. PC-Style has a nice human-factor feature, in that it constantly displays to the user how much it has done and how far it is from finishing. RightWriter is more a coffee-break program: you execute it and then you go make some coffee. PC-Style, however, had little to recommend itself besides this feature. It relies upon a readability formula. It also attempts some affective modeling, based upon its dictionary of 50 action verbs. This minuscule dictionary is inadequate for the task. A match of less than two percent of my input text with these 50 words was typical; such a small sample is inadequate to qualify the vigor of a passage. Another simplistic but more valuable tactic of PC-Style was to count the frequency of "personal" words, such as "I", "you" and "we". Although the conventions of the more stuffy forms of writing forbid them, it is generally accepted that most technical writing benefits from direct, personal pronouns. PC-Style reinforces the clarity and forcefulness of direct rhetoric. Despite these few nice features, PC-Style is a too simplistic to be of lasting utility.

3. Writer's Workbench

Although I was unable to experience Writer's Workbench, I did obtain enough research materials to form an impression of its utility. [Ref. 10] Writer's Workbench is actually a complex of 32 programs. Together, these programs provide more than all the features of RightWriter. STYLE calculates readability, using the previously-discussed formulae. It also analyzes sentence type (simple, complex or compound). STYLE is able to disambiguate the words of the input text with 95% accuracy. It then analyzes the use of verbs and modifiers. If a passage relies too heavily on the passive voice or it is fat with modifiers, STYLE warns the user. The follow-on program, DICTION, detects hackneyed phrases. SUGGEST suggests replacements. Overall, Writer's Workbench appears to be the industry standard.

---

3PC-Style (tm) is a product of ButtonWare Inc., P.O. Box 5786, Bellevue, WA 98006, (206) 454-0479.

4Writer's Workbench (tm) is a product of Bell Laboratories, Murray Hill, New Jersey, 07974.
C. CONCLUSIONS BASED UPON RESEARCH

Now I knew that I would not be writing the first style checker. Nor would I be writing the second, third, fourth or fifth. Some solace could be found in the fact that the previous programs relieved me of the need to incorporate all stylistic features into The Stylist. I could extend the affective modeling of PSA by including other values of words. Three of these new values could be more objective—etymology, difficulty and tangibility. Together with vigor and emotional connotation, these values should be able to describe a profile of the user's input text.

Additionally, I could increase the accuracy of such style checkers by pointing out the need to make allowances for the genre of the user's input text. Any reader of technical as well as fictional writing knows that the characteristics of these two types often vary more widely than the style of writers within the type. Comparative style analysis, therefore, would clearly seem to need to take the genre into account.
ILL. DESIGN OF THE STYLIST

A. CHARACTERISTICS OF WORDS

The fundamental idea of The Stylist is that individual words have power. The denotation of a word is its meaning. The connotation of a word is its emotional impact. For example, "pupil", "student", and "scholar" all denote a person who studies and learns. The emotional connotations, however, range from the humble "pupil" through the familiar "student" to the lofty "scholar". Besides these emotional connotations, words have other intrinsic values which can be quantified. The Stylist would concentrate on the following values: Etymology, Tangibility, Difficulty, Emotional Connotation and Vigor.

1. Etymology

One of the beauties of the English language is that its vocabulary embraces two main sources: native and borrowed. Our native words mainly come to us from the Anglo-Saxon tongues. Borrowed words come from Latin, mostly by way of the French of the Norman conquest. The following table illustrates the differences:

<table>
<thead>
<tr>
<th>Native Words</th>
<th>Borrowed Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Man</td>
<td>Person</td>
</tr>
<tr>
<td>Finger</td>
<td>Digit</td>
</tr>
<tr>
<td>Thinker</td>
<td>Philosopher</td>
</tr>
<tr>
<td>Fire</td>
<td>Conflagration</td>
</tr>
<tr>
<td>Book</td>
<td>Volume</td>
</tr>
<tr>
<td>Dirtbag</td>
<td>Miscreant</td>
</tr>
</tbody>
</table>

Native words are short, strong and rough. Borrowed words tend toward length, gentility and elegance. Good English prose (particularly good fiction) favors native words. Poor English prose (particularly bad technical writing) exhibits a tendency to overutilize latinate etymology. By counting up the number of native and borrowed words in a user's input text, The Stylist could see how it compares to good writing of the appropriate genre.
2. Tangibility

A word can either evoke an image of a thing or it can refer to an idea. I call the former, "tangible" and the latter, "intangible". The following table illustrates the difference:

<table>
<thead>
<tr>
<th>TANGIBLE</th>
<th>INTANGIBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rock</td>
<td>Ethereal</td>
</tr>
<tr>
<td>Lips</td>
<td>Automatically</td>
</tr>
<tr>
<td>Beehive</td>
<td>Then</td>
</tr>
<tr>
<td>Corvette</td>
<td>Rely</td>
</tr>
<tr>
<td>Hammer</td>
<td>Preliminary</td>
</tr>
</tbody>
</table>

Tangible words are concrete, exact and evocative. Intangible words are ideal, general and cognitive. All writing uses both. Good writing usually takes advantage of tangible words. Even the most philosophical writing benefits from the use of tangible words. (See Appendix E for an analysis of a Platonic dialogue). Good fiction writing rarely strays too far toward the intangible. The Stylist could count the tangible and intangible words of the user's input text and compare this count to good writing of the appropriate genre.

3. Difficulty

This is one of the most indisputable characteristics of words. For the purposes of The Stylist, I defined four levels of difficulty: Elementary, High School, Graduate and Postgraduate. The words of each level are those most likely to be used in speech with ease by an average person of that educational level. Although I'm aware that reading, writing and speaking vocabularies are different, I contend that a person reading a word within his reading vocabulary but outside of his speaking vocabulary often must pause for a mental translation into more simple terms. The act of this pausing raises the difficulty level of the text. Table 4 illustrates the categories. These examples are based upon my judgements.

By counting the occurrence of difficult words, The Stylist could determine the overall difficulty of the vocabulary and thus the readability of the text.
TABLE 4
EXAMPLES OF WORDS OF VARIOUS DIFFICULTY

<table>
<thead>
<tr>
<th>ELEMENTARY</th>
<th>HIGH SCHOOL</th>
<th>GRADUATE</th>
<th>POSTGRADUATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big</td>
<td>Tardy</td>
<td>Matrix</td>
<td>Execrable</td>
</tr>
<tr>
<td>Sister</td>
<td>Rendezvous</td>
<td>Immaterial</td>
<td>Parsimony</td>
</tr>
<tr>
<td>Spoon</td>
<td>Transmission</td>
<td>Tonality</td>
<td>Recursive</td>
</tr>
<tr>
<td>Flying</td>
<td>Foreman</td>
<td>Universal</td>
<td>Homomorph</td>
</tr>
<tr>
<td>Wish</td>
<td>Process</td>
<td>Induction</td>
<td>Dilatory</td>
</tr>
<tr>
<td>Done</td>
<td>Joyous</td>
<td>Processor</td>
<td>Zygote</td>
</tr>
<tr>
<td>Handsome</td>
<td>Undergo</td>
<td>Linear</td>
<td>Synergy</td>
</tr>
</tbody>
</table>

4. Emotional Connotations

The Stylist would have five categories of emotional connotation: Sublime, Pleasant, Neutral, Unpleasant and Horrid. The following table illustrates the categories:

TABLE 5
EXAMPLES OF WORDS OF VARIOUS EMOTIONAL CONNOTATIONS

<table>
<thead>
<tr>
<th>SUBLIME</th>
<th>PLEASANT</th>
<th>NEUTRAL</th>
<th>UNPLEASANT</th>
<th>HORRID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beauty</td>
<td>Happy</td>
<td>The</td>
<td>Damage</td>
<td>Cancer</td>
</tr>
<tr>
<td>Sunrise</td>
<td>Food</td>
<td>Which</td>
<td>Loss</td>
<td>Murder</td>
</tr>
<tr>
<td>Victory</td>
<td>Friendly</td>
<td>Brick</td>
<td>Insulting</td>
<td>Whore</td>
</tr>
<tr>
<td>Love</td>
<td>Warm</td>
<td>Is</td>
<td>Loser</td>
<td>Fuck</td>
</tr>
<tr>
<td>God</td>
<td>Helpful</td>
<td>Name</td>
<td>Cost</td>
<td>Death</td>
</tr>
<tr>
<td>Paradise</td>
<td>Sex</td>
<td>When</td>
<td>Wound</td>
<td>Traitor</td>
</tr>
</tbody>
</table>

By counting the use of these types of words, the Stylist could determine the overall emotional tone of the passage. It could also detect flat or emotional writing.

5. Vigor

A related but distinct characteristic is vigor. Sublime and horrid words tend to be highly vigorous, but not all highly vigorous words are emotional. Examples of vigorous but unemotional words are "sprint", "rush" and "cross". This category also tends to be more objective. For example, the word, "soldier" can have widely different connotations for different people. The word "soldier" would please a career Army officer but it would displease a survivor of the Japanese occupation of Canton. Both could agree that "soldier" is a vigorous word.

20
TABLE 6
EXAMPLES OF WORDS OF VARIOUS VIGOR

<table>
<thead>
<tr>
<th>VIOLENT</th>
<th>ENERGETIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destroy</td>
<td>Sprint</td>
</tr>
<tr>
<td>Creation</td>
<td>Dive</td>
</tr>
<tr>
<td>Fire</td>
<td>Discipline</td>
</tr>
<tr>
<td>Atomic</td>
<td>Wedding</td>
</tr>
<tr>
<td>Holocaust</td>
<td>Steam</td>
</tr>
<tr>
<td>Conqueror</td>
<td>Flying</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CALM</th>
<th>INERT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>From</td>
</tr>
<tr>
<td>Write</td>
<td>Into</td>
</tr>
<tr>
<td>Manager</td>
<td>Ocean</td>
</tr>
<tr>
<td>Ocean</td>
<td>Comma</td>
</tr>
<tr>
<td>Blue</td>
<td>Paper</td>
</tr>
<tr>
<td>Sought</td>
<td>Format</td>
</tr>
</tbody>
</table>

By calculating the vigor of the words of the user's input text, The Stylist would be able to estimate its overall strength.

B. THE NEED FOR A DICTIONARY

To quantify the above five aspects of the words of the user's input text, it would be necessary to maintain a dictionary of the most commonly used words and their values. The original idea of The Stylist foresaw just such a dictionary. After reading Anderson [Ref. 12] and testing PC-Style, the idea seemed less original but still valid. The heart of The Stylist, therefore, would be its dictionary. Before I began any top-down designing, I first wanted to explore the technical challenges and possible pitfalls of building such a dictionary. A review of data structure literature convinced me that two approaches were the most feasible: hashing and Binary Search Tree. Since I had already decided that The Stylist would also produce a concordance, I had to take into account the need for efficient alphabetical traversal of the words of the user's input text. My intuition that hashing would be preferable for dictionary look-up but would not lend itself easily to concordance-building was confirmed by a passage in an excellent text by Donald E. Knuth. [Ref. 15; p. 540] Having decided to implement the dictionary with a Binary Search Tree (BST), I began to build a series of prototypes.

C. THE COPYDIX PROTOTYPES

CopyDix1 through CopyDix6 were early experiments in using a Binary Search Trees for dictionary lookup. Naturally, the word was the key value. The experience of the CopyDix series taught me the following lessons:
1. The Benefit of Preorder Storage

The structure of a Binary Search Tree is maintained during execution as a system of pointers. These pointers refer to locations in memory which pertain only to that execution. If a dictionary is modified during execution (for example, if new words are added), then the BST should be stored to its file in preorder. An inorder storage seems the most logical way but it is actually the worst. If a BST is stored using an inorder traversal, the next time it is loaded it will be that least bushy of all BSTs: a linked list. Storage using preorder will cause the next loading of the tree to duplicate the last.

2. The Need for AVL

Even with preorder storage, the dictionary needed the capability to change: to grow or shrink as the user desired. Any particular office or curriculum has its own special vocabulary. I wanted to give The Stylist the ability to adapt its dictionary to the vocabulary of its user's environment. Without some mechanism for rebalancing the BST after the insertion of new words or the deletion of unwanted words, the BST could become increasingly lopsided. Searching for words in such a lopsided tree would become inefficient. Clearly, an AVL scheme was required. I adapted an AVL insertion routine from an excellent text by Niklaus Wirth. [Ref: 16: pps. 220 - 221]

3. Storage requirements

The CopyDix series allowed me to see whether a BST of 5000 words and associated values would fit into the main storage available to a healthy personal computer. I determined that 1.5 megabytes would be sufficient for The Stylist and its data structures. The Copydix series allowed me to create such a structure and to prove that it would not require more than one megabyte.

4. Text Processing

The CopyDix series also identified some unforeseen complexities in processing text. For example, in order to include contractions, it was necessary to include the single quotation mark (') in the dictionary. Waterloo Pascal uses this character to define the beginning and end of user-defined constants and strings. To include contractions, it was necessary to declare "Succ( '@' )" instead of ('). There were a few other similar problems requiring equally unhappy solutions.

5. Building a Dictionary

Finally, the CopyDix series allowed me to begin to build a dictionary of most frequently used words. I ran the text of two novels and several technical articles through the Copydix series, accumulating a dictionary of over 2000 words.
D. CONCORDANCE SEARCH TREE (CST)

1. Original Concept

Based upon the experience of the CopyDix series, I began to play around with the various ways of building a concordance. The literature provided two examples of how the problem has already been solved. One solution was to create a search tree. The key value of the search tree was the word; two associated tables, \text{TOKEN} and \text{TYPE}, recorded the information about the sequence of occurrence of the words. [Ref. 7: pp. 186 - 191] Another solution familiar to most computer scientists was the Key Words in Context (KWIC) program designed by Parnas. [Ref. 17] This solution creates a KWIC listing by circular shifting each line and then sorting each line; such a KWIC listing is similar to a concordance.

I wanted to create a concordance without relying upon any external storage. My experience with the CopyDix series had shown that text processing is both I/O and computation bound: I/O bound because large text files must be loaded; computation bound because each word must be processed character by character. I wanted the solution to eliminate the need for further input or output (such as using secondary storage to build files containing KWIC lines). I also wanted to minimize computation by allowing The Stylist to remember the order of occurrence of each word of the input text, without having to recompute it. I wanted my solution to flow as naturally as possible from the process of looking each word up in the dictionary. I didn’t want to have to process each word twice, or to search for the position of any word twice. Ideally, the Stylist would have enough main storage available that it could retain all important information, without resorting to recomputation or to secondary storage. These ideas led me play to around with various ways of using a BST to record the order of occurrence of the words of the user’s input text. One obvious solution would be to have each node of the BST contain pointers to associated data structures such as an array of linked lists containing the sentences of the input text. Such a solution, however, would require storing the words twice. CopyDix had shown that words take up a great deal of storage space. Storing them twice would be wasteful. With these ideas in mind, I lit upon the idea of the Concordance Search Tree (CST).

A CST is a Binary Search Tree with a linked list threaded through it. This linked list is two-way, connecting each word of the BST to the word used before it and to the word used after it. An inorder traversal of the BST visits each word in alphabetical order. During each visit, a traversal of the linked list in the "backwards"
direction would encounter all the words used before that word. Printing these words, then printing the key word being visited, and finally printing the words in the "forwards" direction creates a concordance.

This solution offers some exciting facilities. It allows the user's input text to be accessed in any number of ways. If a concordance of three, five, seven or thirteen words per line is desired, changing a few global constants immediately fulfills that desire. If only the sentences containing a certain word are desired, a insert search of the CST keying on that word, and then a concordance traversal on that node, provides all those sentences. In short, a CST is a complex structure with a challenging but strong intuitive appeal. It provides a great deal of flexibility to the processing of text.

One added wrinkle of complexity lies in the fact that many words of the input text occur more than once. The word, "the", for example, occurs many times. To guarantee a correct concordance traversal of the CST, each occurrence of the word "the" must be associated with a unique pointer. This requirement is satisfied by creating a unique node of pointers associated with each occurrence of the word. The main node of the CST contains the word, its values, and left and right pointers. It also contains a pointer called "down", which points to the linked list of unique nodes of pointers. The first such node pertains to the first occurrence of the word, "the", in the user's input text. It points to the word used before and the word used after that occurrence of the word "the". (These pointers are called "last" and "next".) It also points "down" to the second occurrence of the word "the". So the Stylist creates a concordance through the following traversals: an inorder traversal of the CST visits each main node in alphabetical order. During that visit, The Stylist goes down the linked list of the occurrences of that word. During each step down the linked list, The Stylist traverses "lastward" and then "nextward" to print out the line of that occurrence. When The Stylist reaches the bottom of the downward list, it continues the inorder traversal.

2. The Tri1 - Tri9 Prototypes

To examine the feasibility of the CST concept, I wrote a series of progressively more capable prototypes called Tri1 through Tri9. (The names refer to a "Trinary Search Tree"). I found that I could implement all the procedures necessary to build and traverse a CST in only 137 lines of code. Moreover, the performance of the concept, both in terms of main storage and processor requirements, seemed extremely satisfactory.
E. DESIGN OF THE STYLIST

Having used a series of prototypes to identify and solve the critical issues, I set them aside and began to design The Stylist. After jotting down some logic flowcharts and data flow diagrams, I decomposed the problem into three main modules which I called “Reader”, “Researcher” and “Reporter”.

1. The Reader Module

Only Reader has access to the user’s input text. It reads that text and passes along a data structure called “readnode”, which is merely the word and its length. The Reader also signals the ends of sentences and the end of the file.

2. The Researcher Module

Researcher receives the “readnodes”, researches the qualities of that word, builds a profile of the user’s input text based upon the tally of the qualities of those words, and then passes that profile on to the Reporter Module. If the user wants one, Researcher also creates the concordance. Only Researcher has access to the dictionary file and to the CST. Neither Reader nor Reporter know how the dictionary is implemented or how the concordance is produced. This is the most crucial instance of information hiding in the design.

3. The Reporter Module

Reporter receives the profile, which it compares to the profiles of other texts. Based upon this comparison, it passes on recommendations and commendations to the user.

4. Low Level Design

Having decomposed the problem into these three modules, I jotted down a plan for the procedures that would comprise them. This plan included the input/output and operations of each procedure. This low level design included all the parameters of the procedures and identified the hierarchy between the procedures and modules.

One of the important features built into this low level design was the human-factor facility of reporting to the user the progress of the execution of The Stylist. The need to send messages to the screen telling the user what was happening required counting steps and further input/output, but my experiences with RightWriter, PC-Style and the CopyDix prototypes had convinced me that the expense was worth it. Psychologically, staring at a blank screen for five minutes seems ten times as long as reading a dynamic screen for seven minutes. To provide the user the option of quicker
but more boring execution over slower but more engaging execution, I designed a facility that asked him whether he wanted frequent reports, occasional reports or one report when done.

With this low level design in hand, I was ready to begin to code.
IV. IMPLEMENTATION OF THE STYLIST

A. CODING

Because of the experience gained from the CopyDix prototypes and because of the top-down, modular design, coding the Reader and Researcher modules took only two weeks. Prototyping and top-down design allowed me to code quickly and with few errors. (The usefulness of these software engineering techniques would also prove itself in the testing phase.) As I had the CopyDix series, I implemented The Stylist in a series of iterative steps called Style1 through Style9.

Style6 brought to light light an unforeseen problem: the interaction between the facilities of adding new words to the dictionary and creating a concordance. The design called for allowing the user to decide whether he wanted to add new words to the dictionary and whether he wanted a concordance. These two choices created four cases: Growing Dictionary and Concordance; Growing Dictionary and no Concordance; Static Dictionary and Concordance; Static Dictionary and no Concordance. In the case of Static Dictionary and Concordance, to create a concordance, new words found in the user's input text would have to be added to the CST. But since the user had chosen not to add new words to the dictionary, The Stylist could not prompt him for the values of these new words. This was not a problem until the same new word was used again. Researcher would find this new word in the CST but would not find any values for the word. This would cause a fatal error: "Word.etymology has unassigned value.″ To avoid this problem, it was necessary to add a new field to each of the nodes: Status. The status of a node could be: "Valuable", which meant that the node contained values for the other fields; "Not Valuable", which meant it didn’t.

This particular problem grew even more gnarly when I decided to extend the Look_Up procedure of Researcher. The original design called for Look_Up to try to match the input word only with its exact equal in the dictionary. This worked fine, but it meant that the dictionary would have to contain all the following variants of "look": "look", "looks", "looked", and "looking". The extension created a new procedure called Look_Up_Variants. If the input word didn’t match any word in the dictionary, Researcher looked up variants of the word. If it found one, the values of the variant
were added to the profile. This extension allowed a more compact dictionary and it increased the probability that most words would be found, but it complicated the problem described in the previous paragraph. In the case of Growing Dictionary and Concordance, a word like “looked” would have to be added to the CST for the sake of a concordance, but now it would also be added to the dictionary. Look_up_variants would thus eventually gum up the dictionary with a lot of useless variants. Solving the problem meant adding a third status. The three statuses were now: “Storable”, which meant the node was a first class citizen, a word which had values and should be stored to the dictionary; “Not_Storable”, which meant that the node was a second class citizen, a variant word which had values, but should not be stored to the dictionary; and “Not_Valuable”, which meant the word was a third class citizen, a new word which had no values, nor should be stored to the dictionary.

The solution of these problems completed the coding phase of the Reader and Researcher modules. The Reporter module remained a stub. The Stylist could not make recommendations and commendations to the user until it had established its dictionary and built up a history of other profiles.

B. BUILDING THE DICTIONARY

The CopyDix series and Style1 through Style7 had built up a dictionary of some 3000 words, which contained about 80% or 85% of the words of any given text. Each of these 3000 words, however, now required the assignment of five values: etymology, tangibility, difficulty, emotional connotation, and vigor. That totals to 15,000 values. After experimenting, I developed a system that allowed me to enter those 15,000 values in two weeks. First, I sorted the dictionary alphabetically and printed out a listing. Using Funk and Wagnall’s Collegiate dictionary, I checked the etymology and highlighted the borrowed words on the print-out. Then I wrote a program called AUTOMATE, which displayed a screen for each of the five values of each word and required one keystroke to assign a value. Using this system, I assigned values to all 3000 words, going at a speed that relied upon a subjective reaction similar to the reaction of a reader scanning a text at a comfortable reading pace. Then I wrote two new programs, DIXSPLIT and DIXJOIN. DIXSPLIT split the dictionary into 17 intersecting subsets: all native words, all borrowed words, all tangible words, all intangible words, and so on. Each of these subsets was like a formation of soldiers. Any word that didn’t belong in that subset stuck out conspicuously. After correcting these mistakes, I executed DIXJOIN, which brought the amended dictionary together.

Now I was ready to begin testing The Stylist.
V. TESTING OF THE STYLIST

A. INITIAL TEST DATA SET

The initial test data set for The Stylist included both fiction and technical writing. Fiction was represented by long passages from two of my novels and 1000-word excerpts from the novels of Ernest Hemingway, Raymond Chandler and Kurt Vonnegut. Technical writing was represented by about a dozen student essays and excerpts from three textbooks on computer science. In both categories, I included both good and poor writing. I ran all the texts through Style8, the Reporter module of which merely printed out the values of their profiles. Style8 performed robustly. I executed it under all four cases pertaining to the dictionary and concordance as well as under all three cases pertaining to the frequency of execution status reports. Style8 revealed no major flaws. Its shortcomings were well within the scope of the original design. (See Appendix A, Suggested Extensions to The Stylist). Style8 suggested some fine-tuning changes.

Two changes dealt with the part of the CST scheme called the FourDix. The FourDix contains the most common function words, all of which are four letters or less in length. Examples of FourDix words are "the", "a", "then", "or", "he", "she", and so on. The FourDix is a separate CST. Words of the user's input text of four or less letters in length are first sought for in the FourDix. This is an economy measure, since a large portion of any text is made up of the function words found in the FourDix. Another savings of the FourDix is that the concordance is created by an inorder traversal of the main CST. This means that the functional words of the FourDix are included in the concordance only in the phrases embracing the more significant words. Otherwise, the concordance would have dozens or perhaps hundreds of lines showing each use of words like "the". The first fine-tuning change suggested by Style8 dealt with the values of the words of the FourDix. These words had values, of course, but they tended to obscure the overall picture. Eliminating their values from the tally was the first solution. I modified this to adding only those values of personal pronouns, which are words of energetic vigor. This is similar to the approach of PC-Style. The second fine-tuning change was not adding the length of the FourDix words to the tally of word lengths. With the FourDix words, a bar graph of word length usually had two
humps: one tall one around three letters and another, shorter one near five letters. By eliminating the length of the FourDix words, both fiction and nonfiction writing always showed a bell-shaped curve. The bell of fiction writing tends to be a tall one centered on five letters/word; the bell of nonfiction writing tends to be a shorter one centered on six letters/word. All this means is that fiction writers use shorter words and words of more uniform length, whereas nonfiction writers use longer words and words of more variable length. As a stylistic measure, its use is limited. As a bar graph, it is a lot of fun.

In any case, the profiles obtained allowed me to examine the true usefulness of The Stylist. Thankfully, The Stylist was able to distinguish between the novels of Ernest Hemingway and the ramblings of computer science graduates. The profiles of each text corresponded closely to my subjective impressions. (This is not surprising since the values of the dictionary were also the product of my subjective decisions. In this sense, The Stylist is an expert system that has automated my own rules and sense of style. The facility of allowing the dictionary to change, however, allows any user to adapt The Stylist to his own taste).

I now could finish coding Reporter. Reporter now took the profile and manipulated its values to create an Analysis. This manipulation took into account the genre of the user's input text and involved simple weighting factors rather than any complex statistical methods. Like the original values of the words of the dictionary, these weighting factors depended upon my own subjective impressions, but they also derived from the results of the test data set runs. If a technical paper had four times as many intangible words than tangible words, it read like so much mush. So I incorporated a ratio of 4:1 as the limit of intangible:tangible for nonfiction. I tried to be conservative in these weightings, because I always envisioned The Stylist as a descriptive rather than a prescriptive tool. The Stylist thus rebukes the user only when the qualities of his text fall far outside the bounds of the norm.

B. FIELD TESTS

The Stylist was finally ready for field testing. This final phase was meant to mimic the introduction of the software product for consumer use. The field testing took place in two parts. In the first, two computer science graduates were briefed on the program and asked to use it. In the second, 22 papers for an Administrative Science composition course were run though The Stylist.
1. Field Test One: The Ape Test

The two computer science graduate students evaluated the program by unleashing it on some of their own writing. Their overall reaction was highly positive, possibly because The Stylist seemed to approve of their writing. They praised the human factors and the graphics. They thought that the breakdowns of the values and the concordances were intriguing and offered them insights into their own writing. They complained that the report didn't explain the meanings of some of the values. They also didn't like the need to remove embedded commands from the input text. I addressed these complaints by adding some explanations to the report and by changing Reader so that it didn't signal End_of_Sentence when it encountered an embedded command.

2. Field Test Two: The Writing Class

A Naval Postgraduate School Administrative Science composition class provided 22 short papers for testing. These students are almost all military officers who are considered by their services as top performers. As such, they constitute a somewhat literate and professional test group. Their fairly uniform papers were apparent attempts to incorporate some of the lessons of good business communication. I ran their papers through The Stylist, then provided the reports to the Professor, who gave them to the students. I never met the students; neither did they ever use The Stylist themselves. Of the 22 students, 12 filled out a questionnaire detailing their reactions to the reports.

Most (10 out of 12) had never heard of a style checker before. Most (11 out of 12) understood the reports. One student thought The Stylist was worthless; 11 found some of its features helpful. Six students found the measures of word length and sentence length helpful. Nine found the measures of vigor, etymology and difficulty helpful. Six liked the concordance; four didn't think it was useful. Interestingly, only five thought The Stylist accurately reflected the qualities of their writing; three were sure it didn't; three just weren't sure. One common complaint was that they weren't sure of the meanings of some of the categories.

Overall, this test underscored the need for a user's manual. Appendix B is just such a manual. If this manual had been available for the students or if they had had more experience using The Stylist, then they would have better understood the meaning of the reports. Despite this drawback, it was obvious from the comments in the questionnaire that the majority of the students liked The Stylist and would want to
use it or similar programs to analyze their writing. This test, therefore, indicated that The Stylist would be able to find a place in the classroom and possibly even the market.
VI. CONCLUSIONS

Researching, designing, coding and testing The Stylist was an excellent academic exercise. It brought home many of the lessons of software engineering. It sharpened my skills. The final product, The Stylist, seems a success in that its affective modeling works. As such, The Stylist should serve as a contribution to the development of other style checkers, rather than a stand-alone style checker itself. As to whether writers, teachers and students should use computers to analyze writing, I'm reminded of the following passage from Plato's "Phaedrus":

Socrates:

At the Egyptian city of Naucratis, there was a famous old god, whose name was Theuth... his great discovery was the use of letters. Now is those days the god Thamus was the king of the whole country of Egypt... To him came Theuth and showed his inventions... when they came to letters, This, said Theuth, will make the Egyptians wiser and give them better memories; it is a specific both for the memory and for the wit. Thamus replied: O most ingenious Theuth, the parent or inventor of an art is not always the best judge of the utility or inutility of his own inventions to the users of them. And in this instance, you who are the father of letters, from the paternal love of your own children have been led to attribute to them a quality which they cannot have; for this discovery of yours will create forgetfulness in the learners' souls, because they will not use their memories; they will trust to the external written characters and not remember of themselves. The specific which you have discovered is not an aid to memory, but to reminiscence, and you give your disciples not truth, but only the semblance of truth; they will be hearers of many things and will have learned nothing; they will appear to be omniscient and will generally know nothing; they will be tiresome company, having the show of wisdom without the reality. [Ref. 18]

In the fourth century before Christ, wise Athenians were debating the uses and evils of literature itself. As Plato points out in the above passage toward the end of his dialogue, literature, the act of writing down our ideas, can have its pitfalls. It can weaken our memories. (Let anyone who has never forgotten where he parked his car argue this point.) It can also lead to intellectual cheating. The wisdom of a man capable of piecing together an article with the aid of a library and a long weekend may indeed be less than the wisdom of a man who can stand before a learned crowd and speak an intelligent discourse.
Nevertheless, civilization has embarked on a course inseparable from literature and writing. Wealth, power and knowledge has followed the progress of the written word. Literature has made us strong, but to the extent that we rely upon books instead of our minds, we have grown weak. Now, at what we call the dawn of the information age, we would do well to remember the reservations of Plato. Electronic computation offers us dazzling abilities, but to the extent we rely upon it instead of our minds, we will grow weak.

Therefore, any style checker or CACI program should be used as an interesting tool that provides a fresh perspective on writing. These programs cannot take and should not be put in the place of thoughtful readers, editors, teachers and friends.
APPENDIX A
SUGGESTED EXTENSIONS TO THE STYLIST

1. ABBREVIATIONS

Reader's Read_Intext procedure ignores embedded commands such as "..." by keeping count of the number of words read since the end of the last sentence. If only one word has been read when another period is encountered, Read_Intext does not signal end_of_sentence. This simple feature also allows Read_Intext to ignore the ellipsis ("..."), It doesn't always allow it to ignore abbreviations such as "Mr.", "Dr.", "etc.", or "e.g.". If these abbreviations appear as the first word of this sentence, Read_Intext doesn't signal end_of_sentence. If the abbreviation appears somewhere in the middle of the sentence, Read_Intext does signal end_of_sentence. An example is the following sentence: "You should have told Prof. Wu about this earlier." Read_Intext would signal end_of_sentence at "Prof." and "earlier." This is a shortfall of the Reader module, but one which I haven't corrected for four reasons: first, such occurrences are relatively rare; second, breaking the sentences into two always favors the user; third, the length of sentence measure is not the thrust of The Stylist; fourth, the solution would involve unwanted overhead in computation.

Writer's Workbench solves this problem for at least 48 abbreviations. Presumably, every time a period is encountered, Writer's Workbench checks a dictionary of these 48 abbreviations. If the word preceding the period matches one of these abbreviations, then Writer's Workbench does not count this as an end to a sentence. A similar solution could be implemented for The Stylist.

2. TRANSLATION TO TURBO PASCAL

The design of The Stylist kept in mind the working environment of a personal computer with 1.5 megabytes of main storage. A good follow-on project for The Stylist would be to translate the code from Waterloo Pascal to Turbo Pascal. Copies of this program could then be distributed and possibly even marketed.

3. IMPROVEMENTS OF THE CONCORDANCE

In addition to the main CST and the FourDix, a third CST could be added. This new CST's node's key values would be characters of punctuation. During the
execution of Inorder_Concordance, the procedure that creates the concordance, the
traversals "lastward" and "nextward" could terminate upon encountering a character of
punctuation in this third CST. This would mean the lines of the concordance would
only contain phrases extracted from single sentences. This may or may not be a
desirable feature. I myself found it useful to read entire phrases, even when they
overlapped into preceding or succeeding sentences.

More sophisticated concordances include at the end of each line a note as to
where this line can be found. The third CST outlined above could contains as a field in
its main node just such a note, with the name of the user's input text, the page number
and the line number. Such an extension would be valuable for scholarly research. For
the short, single input texts, such notes are not necessary.

4. PARTS OF SPEECH

A considerably more ambitious extension would be to provide The Stylist the
facility of determining the parts of speech. Other programs [Ref. 8] [Ref. 9] [Ref. 10]
solving this problem use thousands of rules. The disambiguation of parts of speech
would raise The Stylist to a whole new order of complexity. It would also allow more
fine analysis of the characteristics of the words. For example, "like" as a verb has
much different connotations than "like" as a preposition. This facility would also
allow other stylistic measures to figure into the profiles, such as, verb-adjective ratio
and percentage of modifiers.

5. PASSIVE VOICE

The Researcher module could be extended to detect the use of passive voice.
Ideally, this would be accomplished in the context of the extension involving parts of
speech, so that active verbs would also be detected. A simpler solution is readily
available, however. Researcher could raise a flag every time it receives a form of the
verb, "to be": that is, "is", "are", "am", "was", "were", "being", and "been". This flag
would remain raised for the next three words. If any of these three words ended in
"-ed", then Researcher would count this as an example of the use of the passive voice.
Again, I haven't implemented this solution because it is not central to the concept of
affective modeling and because I didn't want to pay for the extra computation.

6. CORRECTION OF POOR DICTION AND GRAMMAR

Another ambitious extension of The Stylist would be to provide it with the
facility of correcting poor diction and grammar. Both RightWriter and DICTION of
Writer's Workbench contain dictionaries of commonly used cliches or repetitive redundancies. Some examples from DICTION are: "a great deal of", "in regards to", "make adjustments to" and so on. One solution for The Stylist would be to include a new field in the dictionary. Words like "deal", "regards" and "adjustments" would contain in this field a pointer to a table containing the poor phrases and their replacements. Such a solution would also require Researcher to maintain a phrase of the three or four most recently received words, so that it could verify that "regards" appeared in the offending context of "in regards to".

Grammatical corrections would require an even more ambitious extension. To perform properly, such a facility would have to be an extension of the parts of speech extension. Determining agreement between subject and verb, for example, would require first the identification of the subject and the verb. Frankly, it would be easier to incorporate the affective modeling of The Stylist into a more extensive program such as Writer's Workbench than it would be to extend The Stylist this far.
1. INTRODUCTION

The Stylist is a Waterloo Pascal program that analyzes the style of English prose, both fiction and nonfiction. The Stylist package contains the following: this User's manual, and the following electronic files: the Waterloo Pascal code of The Stylist itself, the text file dictionary of The Stylist, and two Waterloo Pascal programs for helping to maintain the dictionary, DIXSPLIT and DIXJOIN.

The fundamental idea of The Stylist is that individual words have power. The denotation of a word is its meaning. The connotation of a word is its emotional impact. For example, "pupil", "student", and "scholar" all denote a person who studies and learns. The emotional connotations, however, range from the humble "pupil" through the familiar "student" to the lofty "scholar". Besides these emotional connotations, words have other intrinsic values which can be quantified. The Stylist concentrates on the following values: Etymology, Tangibility, Difficulty, Emotional Connotation and Vigor.

a. Etymology

One of the beauties of the English language is that its vocabulary embraces two main sources: native and borrowed. Our native words mainly come to us from the Anglo-Saxon tongues. Borrowed words come from Latin, mostly by way of the French of the Norman conquest. See Table 7, which illustrates the difference.

<table>
<thead>
<tr>
<th>Native Words</th>
<th>Borrowed Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Man</td>
<td>Person</td>
</tr>
<tr>
<td>Finger</td>
<td>Digit</td>
</tr>
<tr>
<td>Thinker</td>
<td>Philosopher</td>
</tr>
<tr>
<td>Fire</td>
<td>Conflagration</td>
</tr>
<tr>
<td>Book</td>
<td>Volume</td>
</tr>
<tr>
<td>Dirtbag</td>
<td>Miscreant</td>
</tr>
</tbody>
</table>
Native words tend to be short, strong and rough. Borrowed words tend to be long, mild and elegant. Good English prose, particularly good fiction, tends to use native words. Poor English prose, particularly bad technical writing, exhibits a tendency to overutilize latinate etymology. By counting up the number of native and borrowed words in your input text, The Stylist sees how it compares to good writing of the appropriate genre.

b. Tangibility

A word can either evoke an image of a thing or it can refer to an idea. The former words are "tangible" and the latter are "intangible". Table 8 illustrates the difference.

<table>
<thead>
<tr>
<th>TABLE 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAMPLES OF TANGIBILTY</td>
</tr>
<tr>
<td>TANGIBLE</td>
</tr>
<tr>
<td>Rock</td>
</tr>
<tr>
<td>Lips</td>
</tr>
<tr>
<td>Beehive</td>
</tr>
<tr>
<td>Corvette</td>
</tr>
<tr>
<td>Hammer</td>
</tr>
</tbody>
</table>

Tangible words are concrete, exact and evocative. Intangible words are ideal, general and cognitive. All writing uses both. Good writing, however, usually takes advantage of tangible words. Even the most philosophical writing benefits from the use of tangible words. Good fiction rarely strays too far toward the intangible. The Stylist counts the tangible and intangible words of your input text and compare this count to good writing of the appropriate genre.

c. Difficulty

This is one of the most indisputable characteristics of words. The Stylist defines four levels of difficulty: Elementary, High School, Graduate and Postgraduate. The words of each level are those most likely to be used in speech with ease by an average person of that educational level. Table 9 illustrates the categories.

By counting the occurrences of difficult words, The Stylist could determine the overall difficulty of the vocabulary and thus the readability of your text.
d. Emotional Connotations

The Stylist defines five categories of emotional connotation: Sublime, Pleasant, Neutral, Unpleasant and Horrid. Table 10 illustrates the categories.

<table>
<thead>
<tr>
<th>TABLE 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAMPLES OF EMOTIONAL CONNOTATION</td>
</tr>
<tr>
<td>SUBLIME</td>
</tr>
<tr>
<td>Beauty</td>
</tr>
<tr>
<td>Sunrise</td>
</tr>
<tr>
<td>Victory</td>
</tr>
<tr>
<td>Love</td>
</tr>
<tr>
<td>God</td>
</tr>
<tr>
<td>Paradise</td>
</tr>
</tbody>
</table>

By counting the use of these types of words, The Stylist could determine the overall emotional tone of your passage. It could also detect flat or highly emotional writing.

e. Vigor

A related but distinct characteristic is vigor. Sublime and horrid words tend to be highly vigorous, but not all highly vigorous words are emotional. Examples of vigorous but unemotional words are “sprint”, “rush” and “cross”. This category also tends to be more objective. For example, the word, “soldier” can have widely different connotations for different people. The word “soldier” would please a career Army officer but it would displease a survivor of the Japanese occupation of Canton. Both could agree that “soldier” is a vigorous word.
TABLE 11
EXAMPLES OF VIGOR

<table>
<thead>
<tr>
<th>VIOLENT</th>
<th>ENERGETIC</th>
<th>CALM</th>
<th>INERT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destroy</td>
<td>Sprint</td>
<td>Read</td>
<td>From</td>
</tr>
<tr>
<td>Creation</td>
<td>Dive</td>
<td>Write</td>
<td>Into</td>
</tr>
<tr>
<td>Fire</td>
<td>Discipline</td>
<td>Manager</td>
<td>Something</td>
</tr>
<tr>
<td>Atomic</td>
<td>Wedding</td>
<td>Ocean</td>
<td>Comma</td>
</tr>
<tr>
<td>Holocaust</td>
<td>Steam</td>
<td>Blue</td>
<td>Paper</td>
</tr>
<tr>
<td>Conqueror</td>
<td>Flying</td>
<td>Sought</td>
<td>Format</td>
</tr>
</tbody>
</table>

By calculating the vigor the words of your input text, The Stylist would be able to estimate its overall strength.

2. USING THE STYLIST

The first step to using the Stylist is to save your piece of writing as "Input text a". You should be aware of the following principles:

1. Your input text should be at least 500 words long. The Stylist will examine shorter texts, but the statistical sample of shorter texts is too small for valid analysis.

2. Your input text should not be longer than 500 sentences (about 10,000 words). Such long texts require so much computation that most operating systems such as MVS will terminate execution before completion.

3. The Stylist is not meant to analyze non-prose constructions such as tables, lists, references and bibliographies. You should eliminate these from your input text.

In the multiprocessing environment, DEFINE MAIN STORAGE at 1500k. The VM/CMS command is "Define storage 1500k" followed by "I CMS".

Once your input text is properly stored, execute The Stylist. DO NOT interrupt The Stylist during compilation or execution by hitting any keys. Always wait for The Stylist to display a screen with a prompt. After The Stylist is compiled, you will see just such a series of screens that will ask you for information. The following discussions will help you decide how to answer.

a. Name of Intext

The first screen will ask you for the name of your input text. Simply type in the name. You can use any characters. The maximum length of the name is 40 characters. Hit "enter".
b. Fiction or NonFiction

The screen will automatically clear. A new screen will ask you if your input text is fiction or nonfiction. Enter "1" or "2". If you enter any other character, The Stylist will merely ask you again.

c. Report Frequency

The next screen asks whether you want reports "frequently", "seldomly" or "when done". A report is a screen which The Stylist displays to you during execution, telling you what it's doing, how much it's done and how much remains to be done. The act itself of producing such takes time: the more frequent the reports, the longer it takes. You should probably enter "2" for "Seldomly". Most users find this the most agreeable.

d. Concordance

The next screen asks whether or not you want a concordance. A concordance is an alphabetical listing of all the significant words of your input text. ("Significant" words are all words other than functional words such as "the", "and", "as", and so on.) Each line of the concordance contains the key word and the phrases preceding and succeeding it. The following is an example of a concordance.

them and thamus enquired ABOUT their several uses and discovered is not an AID to memory but to other egyptians might be ALLOWED to have the benefit an art is not ALWAYS the best judge of is called by them AMMON to him came theuth learned nothing they will APPEAR to be omniscient and censured others as he APPROVED or disapproved of them many arts such as ARITHMETIC and calculation and geometry or inventor of an ART is not always the the inventor of many ARTS such as arithmetic and blame of the various ARTS but when they came calculation and geometry and ASTRONOMY and draughts and dice have been led to ATTRIBUTE to them a quality in the learners’ souls BECAUSE they will not use your own children have BEEN led to attribute to
Most users find that a concordance offers them interesting insights into their use of words. The only drawback of producing a concordance is that it takes The Stylist additional time. If you elect this option, your concordance will be printed to a separate file called, "Concord text a".

e. Number of Words in Intext

The next screen asks you for the number of words in your text. If you don’t know, make any guess. This number is only used in the screens displaying how far along The Stylist is in reading the text. Your guess will have no effect on the analysis of the text. The Stylist counts the words itself and will report the exact number to you when it finishes.

f. Expand the Dictionary

The next screen will ask whether you want to expand the dictionary of The Stylist or not. This dictionary is a listing of about 3200 of the most commonly used words, together with five values for each word. If you answer “1” for “Yes”, the Stylist will ask you to give it a values for every word in your input text which is not found in the dictionary. It will then add these new words and their values to the dictionary. If you answer “2” for “No”, then The Stylist will write all of the unknown words to a file called “Newwords text a”. Reviewing “Newwords” will give you an idea of the words that you use which aren’t in the dictionary.

The first few times that you use The Stylist, you probably should answer “2” for “No”. Entering the values for the new words can be tedious. You probably want to see how The Stylist works before you begin to expand or modify the dictionary. After you’ve used The Stylist for a while, you’ll probably want to begin adding some words to the dictionary. Words peculiar to your field of writing or the jargon of your profession are examples of the types of words which should be entered. Adding them will tailor The Stylist to your working environment. Keep the following in mind:

*** Every word added to the dictionary is another word that will have to be loaded into memory during each execution of The Stylist. Adding many words will slow down execution and it will decrease the amount of space in main storage. If the dictionary grows beyond 5000 words, you should weed out words that are rarely used in your type of writing. To do this, simply delete that line from the file “Dixonary text a”. ***
If you answer "1" for "Yes", as the Stylist encounters each new word in your text, it will ask if you want to add this particular word to the dictionary. If the word is a misspelling or if it is a word that you rarely use, don't add it. Also, you should know that its best to add the root form of words. For example, neither "burns" nor "burned" nor "burning" should be added. It's much better to add "burn". To keep the dictionary as compact as possible, note down the root form of words you wish to add to the dictionary and append them to the end of the next text you run through The Stylist.

The Stylist will then ask you for the values of the words you do want to add to the dictionary. The following will help you decide how to answer these prompts:

1. **Source**

   If you're uncertain about the etymology of the word, simply look it up in a collegiate dictionary. Old English, Middle English, Norse, German, Old German, Middle German and all Celtic languages are considered "Germanic". Latin, French, Spanish, and all other languages are considered "Latin".

2. **Difficulty**

   Each category contains those words which are most likely to be used in speech by persons of that educational level. If you're still uncertain of the meaning of these categories, the following are examples of the categories:

**POSTGRADUATE DIFFICULTY**

- aggregate
- algorithm
- algorithmic
- ambiguity
- analogue
- applicative
- conceptualize
- conglomerate
- constriction
- consummate
- consummation
- convoluted
- convolution
culminate
cyclical
digitize

GRADUATE DIFFICULTY
accumulate
acrid
acute
adhere
adjoin
allocate
allude
analogy
analysis
analyst
ancestor
ancestral
annihilate
apparatus
appendage
arc
arch
array

HIGH SCHOOL DIFFICULTY
abandon
ability
able
abrupt
absence
absent
absorb
acceptable
access

45
accommodate
accomplish
accomplishment
account
accurate
achievement
activate
adapt
additional
adjust
advantage

ELEMENTARY DIFFICULTY
about
above
accept
acceptance
ache
across
act
action
active
activity
actor
add
addition
address
admit
advance
adventure
afraid
after
afternoon
again

46
3. **Concreteness**

Does this word name a thing that you can touch? If it does, then answer "1" for "Tangible". Otherwise, answer "2".

4. **Emotional Connotation**

What feelings does this word evoke in you? If none, answer "3" for neutral. Otherwise, answer in the appropriate category.

5. **Vigor**

Vigorous words are not necessarily verbs. Words like "conquest" are highly vigorous. "Violent" vigor means "highly" vigorous. Some of the words in the "violent" category are "creation", "triumph" and "epiphany". This category, as does emotional connotation, calls for your subjective reaction. Don't be afraid to "guess" which value is appropriate. Such a "guess" is in fact your subjective reaction.

g. **The End of Execution**

The above are all the screens which prompt you for answers. All the rest of the screens will merely inform you about the progress of the execution. When execution ends, you should look into the following files: "Newwords text a", if you decided not to let the dictionary expand; "Concndnc text a", if you elected to have a concordance produced; and most importantly, "Report text a", which is The Stylist's report on your text.

3. **THE MEANING OF YOUR REPORT**

a. **Measures of Length**

The first two pages of your report deals with the number of words, the length of words and the length of sentences. These measures are always interesting but not always illuminating. The first bar graph shows the distribution of the length of the words of your text. Each horizontal column represents the percentage of the words of your next which were of so many letters of length. The overall bar graph should look like a bell-shaped curve. This bar graph does not include function words such as "a", "an", "the", "then" and "or". The blurb below the bar graph explains how your text compares to other texts of the same genre. The only clear indication of prose in trouble would be a dramatic slewing of the curve to the right. This would mean that you're using too many windy words. Table 12 breaks down the characterizations of word length.
TABLE 12
WORD LENGTH

<table>
<thead>
<tr>
<th>Characterization</th>
<th>Fiction</th>
<th>Nonfiction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too Long</td>
<td>.6</td>
<td>7.5</td>
</tr>
<tr>
<td>Long</td>
<td>5.5</td>
<td>6.5</td>
</tr>
<tr>
<td>Medium</td>
<td>5</td>
<td>5.5</td>
</tr>
<tr>
<td>Short</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Too Short</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The second bar graph shows the length of the sentences of your input text. Each bar represents the length of a sentence. You can use this graph to identify run-on sentences. Overall, this bar graph should resemble the skyline of a modest city such as San Francisco. If it resembles New York, then you tend to write long sentences. Table 13 explains the characterizations of sentence length.

TABLE 13
SENTENCE LENGTH

<table>
<thead>
<tr>
<th>Characterization</th>
<th>Lower boundary of length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too Long</td>
<td>22</td>
</tr>
<tr>
<td>Long</td>
<td>18</td>
</tr>
<tr>
<td>Medium</td>
<td>10</td>
</tr>
<tr>
<td>Short</td>
<td>0</td>
</tr>
</tbody>
</table>

One of the measures below this bar graph is “Number of sweet spots”. This is the number of sentences ranging between 9 and 19 words. The idea here is that most ideas can find a home in a sentence of this length. Table 14 breaks down how The Stylist characterizes the modulation of your sentences.

Another count here is the number of run-on sentences. The Stylist defines any sentence over 45 words in length as a run-on. The Stylist realizes that many grand sentences over 45 words in length are not run-ons, but it’s willing to bet that most overlong sentences are.
b. Measures of Word Characteristics

1. Etymology

The next item in the report is the number of Latinate and Germanic words. Notice that these two numbers will not add up to the total number of words in the text unless every word in the text happened to belong to the dictionary. The characterization of your text in this category depends on the ratio of Latin to Germanic words. Good writing generally tends to be native. Table 15 breaks down these characterizations.

<table>
<thead>
<tr>
<th>Characterization</th>
<th>Ratio of Latin to German</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Borrowed</td>
<td>2:1</td>
</tr>
<tr>
<td>Borrowed</td>
<td>1.5:1</td>
</tr>
<tr>
<td>Mixed</td>
<td>1:1</td>
</tr>
<tr>
<td>Native</td>
<td>.5:1</td>
</tr>
<tr>
<td>Very Native</td>
<td>&lt;.5:1</td>
</tr>
<tr>
<td>Fiction</td>
<td>3:1</td>
</tr>
<tr>
<td>Non-Fiction</td>
<td>2:1</td>
</tr>
<tr>
<td></td>
<td>1:1</td>
</tr>
<tr>
<td></td>
<td>&lt;.9:1</td>
</tr>
</tbody>
</table>

2. Difficulty

The next entry shows the tally of the numbers of words of the various levels of difficulty, followed by the percentage of those levels. The difficulty of the text's vocabulary should be appropriate for its intended audience. Moreover, you should remember than many outstanding works of literature convey difficult ideas without resorting to difficult words. Table 16 describes the characterizations of this category.
TABLE 16
CHARACTERIZATIONS OF DIFFICULTY

<table>
<thead>
<tr>
<th>Characterization</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Hard</td>
<td>postgrad &gt; 0</td>
</tr>
<tr>
<td>Hard</td>
<td>grad &gt; 5</td>
</tr>
<tr>
<td>Challenging</td>
<td>highschool &gt; 10</td>
</tr>
<tr>
<td>Easy</td>
<td>above percentages do not pertain</td>
</tr>
</tbody>
</table>

3. Tangibility

The next entries show the total of the tangible and intangible words. As in the etymology category, the characterizations depend upon the ratio between these words and the genre. Generally, good writing is tangible.

TABLE 17
CHARACTERIZATIONS OF TANGIBILITY

<table>
<thead>
<tr>
<th>Characterization</th>
<th>Ratio of Intangible to Tangible</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fiction</td>
</tr>
<tr>
<td>Very tangible</td>
<td>&gt;1.5:1</td>
</tr>
<tr>
<td>Tangible</td>
<td>1.5:1</td>
</tr>
<tr>
<td>Very intangible</td>
<td>3:1</td>
</tr>
</tbody>
</table>

4. Emotional Connotation

The next breakdown is of the number and percentages of the words belonging to the various categories of emotional connotation. Following these values is something called the "Index of Emotion". The following is the formula used to derive this index:

\[
\text{Index} = (\text{Percent Horrid times 5}) + (\text{Percent Unpleasant times 2}) + (\text{Percent Pleasant times 2}) + (\text{Percent Sublime times 5})
\]
The Index of Emotion is simply a number that conveys a sense of the use of strongly emotional words. The most emotional words, Horrid and Sublime, are weighted the most heavily. This Index allows The Stylist to compare your use of emotional words to the use found in other texts. Table 18 shows the thresholds of such uses.

<table>
<thead>
<tr>
<th>Characterization</th>
<th>Lower Boundry of Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rich</td>
<td>20</td>
</tr>
<tr>
<td>Average</td>
<td>10</td>
</tr>
<tr>
<td>Poor</td>
<td>0</td>
</tr>
</tbody>
</table>

Following this characterization is another one, this time for the overall tone of your text. The Stylist compares the frequency of use of positive words to the frequency of use of negative words. If positive words prevail, the tone is positive. If negative words prevail, the tone is negative. If the two balance and the index of emotion is high, then the tone is characterized as a balance of strong positive and strong negative emotions. If the two balance and the index of emotion is low, then the tone is characterized as bland.

5. Vigor

The final breakdown is that of the number and percentage of words of the various categories of vigor. Following this is an “Index of Strength”, which is a measure similar to the Index of Emotion. The following is the formula used to derive the Index of Strength:

\[
\text{Index of Strength} = (\text{Percent of violent words times 10}) + (\text{Percent of energetic words times 5}) + \text{(Percent of calm words)}
\]

This Index allows the Stylist to compare your use of vigorous words to uses found in other texts. Table 19 explains how The Stylist uses the index to characterize your text. Most good writing is strong or lively. Usually only action-packed fiction reaches the upper registers of “very strong”.

51
6. **Recommendations**

The final section of your report includes recommendations and commendations. The Stylist is meant to be a descriptive rather than a prescriptive tool. Therefore, it makes recommendations only when some aspect of your writing seems well outside of the bounds of the normal. For this reason, you should consider the recommendations of The Stylist. They apply only to what appear to be extreme cases. On the other hand, you should never accept these recommendations as the pronouncements of some oracle. The Stylist is merely a machine. When in doubt, trust your own instincts.

4. **MAINTENANCE OF THE STYLIST**

a. **Care of the Dictionary**

Many of the issues regarding the care of the dictionary have already been covered. Here are some other helpful pointers:

1. Always maintain more than one copy of the dictionary. Loss of the dictionary renders The Stylist useless.
2. Beware of changing the values directly. You should never do this to the dictionary itself. While using DIXSPLIT, it is necessary to go into the split files and change the codes for the words which don’t belong in that split file. If you make a mistake, DIXJOIN will merely assign the most common value for that category.

b. **Changing the Code of The Stylist**

The Stylist comes with embedded documentation. Make sure that you read this documentation prior to changing any of its code. You should read the full master’s thesis pertaining to The Stylist prior to making any substantive changes.\(^5\)

---

### TABLE 20
CODES OF THE DICTIONARY

<table>
<thead>
<tr>
<th></th>
<th>First</th>
<th>Second</th>
<th>Third</th>
<th>Fourth</th>
<th>Fifth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Etymology</td>
<td>l(atin)</td>
<td>t(angible)</td>
<td>e(lement.)</td>
<td>h(orrid)</td>
<td>i(nert)</td>
</tr>
<tr>
<td></td>
<td>g(erman)</td>
<td>i(ntang.)</td>
<td>h(igh sc.)</td>
<td>u(nple.)</td>
<td>c(alm)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>g(rad)</td>
<td>m(eutr.)</td>
<td>e(neg.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>p(oat gr.)</td>
<td>p(lease)</td>
<td>v(ile.)</td>
</tr>
</tbody>
</table>

Substantive changes should also bear in mind the modularization of the design. Reader module bears full responsibility for reading the user’s input text. It passes along words (in lower case) and their length. It also signals the ends of sentences and the end of the file. Researcher Module bears full responsibility for maintaining and using the dictionary, for building up the profile of the user’s input text, and for building the concordance. Finally, Reporter module accepts the profile, creates the analysis and writes out the report.

Some fine tuning of The Stylist is readily available. The global constants give you the capability to adapt The Stylist to your operating environment. For example, “Maxsent” allows you to determine the maximum number of sentences in the input text. Moreover, by manipulating the weighting factors in the Calculate procedures of the Reporter module, you can adapt The Stylist to the norms of a particular school of writing, such as submissions to magazines as varied as “Fantasy and Science Fiction” and “Foreign Affairs.”
THE STYLIST

a Pascal Program for Analyzing Prose Style

by Lieutenant Thomas C. Cool, USN

Submitted in partial fulfillment of the requirements
for the degree of
MASTER OF SCIENCE IN COMPUTER SCIENCE

from the
NAVAL POSTGRADUATE SCHOOL

June 1987

program Stylist (input, output);

const
maxsent = 500; (* limit of number of sentences in intext *)
wordlength = 15;
linelength = 40;
phraselength = 4; (* length of phrase around concordance line *)
freqnum = 50; (* the three numbers determine how often *)
selnum = 200; (* Stylist reports to user during execution *)
whandonenum = 1000000; (* frequently, seldomly or when done *)
type

GLOBAL TYPES

GenreType = (Nonfiction, Fiction);
FrequencyType = (Frequently, Saldomly, When_Done);

WordType = packed array (1..wordlength.) of char;
LineType = packed array (1..linelength.) of char;

UserInfoType = record
Name_of_text : linetype;
Genre : GenreType;
Size : Integer;
Frequency : FrequencyType;
Want_Concord : boolean;
Want_Dix_Grow : boolean;
end;

READER MODULE TYPES

Readtype = record
Word : wordtype;
length : integer;
end;
RESEARCHER MODULE TYPES

**BalanceType** = (Plus, Zero, Minus); (**Balance of Node in BST**)

**Statutype** = (Storable, Not_Storable, Not_Valuable);

- (**Storable** - a full node with all values which)
- (**Not_Storable** - a node of a variant (-ing, -ad))
- (**Not_Valuable** - a node of a new word without)

**Storable** - a full node with all values which can be added to the dictionary
**Not_Storable** - a node of a variant (-ing, -ad)
**Not_Valuable** - a node of a new word without

**SourceType** = (Latinate, Germanic);
**DifficultyType** = (PostGrad, Grad, High_School, Elementary);
**ConcretenessType** = (Tangible, Intangible);
**EmotionType** = (Sublime, Pleasant, Neutral, Unpleasant, Horrid);
**VigorType** = (Violent, Energetic, Calm, Inert);

**EntryPointer** = -**EntryType**;
**ConcordPointer** = -**ConcordType**;

**EntryType** is record (X main node in the BST X)

- **Word** : **WordType**;
- **Balance** : **BalanceType**;
- **Status** : **Statutype**;
- **Source** : **SourceType**;
- **Difficulty** : **DifficultyType**;
- **Concreteness** : **ConcretenessType**;
- **Emotion** : **EmotionType**;
- **Vigor** : **VigorType**;
- Left, Right : **EntryPointer**;
- Down : **ConcordPointer**;
end;

**ConcordType** = record (**X a Concord node is a cluster of pointers X**)

- **Up** : **EntryPointer**; (**X which create linked lists from the first**)
- Down, (**X word of the Intext to the last, and from**)
- Next, (**X the last to the first. Used only to**)
- Last : **ConcordPointer**; (**X create the concordance.**)
end;
LtrperWordtype = array (.0..wordlength.) of integer;
(*) array of total number of words of each length

WordsPerSenttype = array (.0..maxsent.) of integer;
(*) array of the length of each sentence

Profiletype = record
(* the profile is the main product *)
(* of the Researcher Module. Here *)
(* Researcher keeps track of all *)
(* counts. *)

TotalWords : integer;
TotalLetters : integer;
TotalSentences : integer;
TotalStructures : integer;
NumWordsThisSent : integer;
NewWords : integer;
KnownWords : integer;
VariantWords : integer;
Structurewords : integer;
UnknownWords : integer;
Numfound : integer;
LtrPerWord : ltrperwordtype;
WordsPerSent : wordsperSenttype;
Ave_ltr_per_word : real;
Ave_words_per_sent : real;
NumGermanic : integer;
NumPostGrad : integer;
NumGrad : integer;
NumHigh_School : integer;
NumElementary : integer;
NumIntangible : integer;
NumSublime : integer;
NumPleasant : integer;
NumNeutral : integer;
NumUnpleasant : integer;
NumHorrid : integer;
NumViolent : integer;
NumEnergetic : integer;
NumCalm : integer;
NumInert : integer;
end;

Phrasetype = packed array (.1..phraselength.) of wordtype;
(*) holds the phrases bracketing the key words of concordance line *)

56
(n)

REPORTER MODULE TYPES

Lengthtype = (TooShort, Short, Medium, Long, TooLong);
Etymologytype = (TooBorrowed, Borrowed, Mixed, Native, TooNative);
Modulationtype = (Bad, Average, Good);
Runonstype = (Unacceptable, Acceptable, Nonexistent);
Tangibilitytype = (Soft, Firm, Solid);
Strengthtype = (Weak, Lively, Strong, VeryStrong);
Hardwordstype = (Easy, Challenging, Hard, VeryHard);
Emotionalitytype = (Poor, Standard, Rich);
Tonetype = (Negative, Bland, Positive);
(* Judgements of intext style Reporter makes based on profile *)

Analysis = record
    Length : Lengthtype;
    SentLength : Lengthtype;
    Modulation : Modulationtype;
    Runon : Runonstype;
    Etymology : Etymologytype;
    Tangibility : Tangibilitytype;
    Strength : Strengthtype;
    Hardwords : Hardwordstype;
    Emotionality : Emotionalitytype;
    Tone : Tonetype;
end;

57
var
  Info : User_InfoType;
  Allswell : boolean; (* global flag that execution can continue *)
  (* used instead of a GOTO and of program *)
  FourRoot, (* Root of the FourDix BST *)
  Root : EntryPointer; (* Root of the Dixonary BST *)
  Lastward : ConcordPointer; (* Pointer to last word read *)
  Intext, (* file that holds users prose *)
  Dixonary, (* dictionary of words and values *)
  FourDix, (* dictionary of short, functional words *)
  Concordso, (* concordance, created by researcher *)
  NewWords, (* list of words not found or added *)
  Report : text; (* main product of Reporter *)

  Tally, (* Researcher scratch pad of profile *)
  Profile : profiletype; (* Researcher input to Reporter *)
  Analysis : analysisType; (* Judgments made by reporter *)

main program

begin (* main program *)
  Interrogate_User;
  if Allswell then Init_Reader;
  if Allswell then Init_Researcher;
  if Allswell then Init_Report;
  if Allswell then Read_Intext;
  if Allswell then Calculate_Profile;
  if Allswell then Analyze_Profile_and_Report;
  if Allswell then if Info.Want_Concord then
  Write_Concordance;
  if Allswell then if Info.Want_Dix_Grow then
  Store_New_Dix;
  if not Allswell then
  Writeln ('Execution terminated until you clear that up.');
end.
procedure Interrogate_User;
var c : char;
good_integer : boolean;
i, j : integer;
begin
Page;
Write; Writeln; Write;
Write ('THE STYLIST':50); Write ('by LT TC Cool':50);
Write ('What is the name of the text to be analyzed?':65);
i := 0;
while (i < linelength) and not (EOLN) do begin
i := i + 1;
Read (Info.Name_of_Text(.i.));
end; Readin;
for j := i + 1 to linelength do Info.Name_of_Text(.j.) := ' ';
Page;
Repeat
Write; Writeln; Write;
Write ('Is it a story? (1) Yes (2) No':45);
Read (c); Readin; Writeln (c:45);
Until (c = '1') or (c = '2');
if (c = '1') then begin Writeln ('That is good.':45);
   Allowell := true; end
else begin Writeln ('That is too bad.':45);
   Allowell := false; end
if Allowell then begin
Page;
Write; Writeln; Write;
Write ('(1) Fiction (2) Nonfiction':45);
Write; Writeln; Write;
Read (c); Readin; Writeln (c:45);
if (c = '1') then Info.Genre := Fiction
else if (c = '2') then Info.Genre := Nonfiction;
Until (c = '1') or (c = '2');
Repeat
Page;
Writeln; Write; Writeln;
Write ('(1) Frequently (2) Seldomly':45);
Writeln ('(3) When Done':45); Writeln;
Read (c); Readin; Writeln; Write (c:45);
if (c = '1') then Info.Frequency := Frequently
else if (c = '2') then Info.Frequency := Seldomly
else if (c = '3') then Info.Frequency := When.Done;
Until (c = '1') or (c = '2') or (c = '3');

59
Repeat
Page; Writeln; Writeln; Writeln;
Writeln ("CONCORDANCE 1:45"); Writeln;
Writeln (" 1) Yes 1:45");
Writeln (" 2) No 1:45");
Read (c); Readln; Writeln (c:45);
if (c = '1') then Info.Want_Concord := true
else if (c = '2') then Info.Want_Concord := false;
Until (c = '1') or (c = '2'))
Repeat
Page; Writeln; Writeln; Writeln;
Writeln ("Approximately how many words are in ");
Writeln (Info.Name_of_text, '1:45");
Info.size := 0;
Good_integer := true;
while not EOLN do begin
Read (c);
if (c in (',0',..9',)) then
  Info.size := Info.size + ord(c) - ord('0')
else begin
  good_integer := false;
end;
Readln;
Until Good_Integer;
Writeln (Info.size:45);
Repeat
Page; Writeln; Writeln; Writeln;
Writeln ("EXPAND DICTIONARY 1:45"); Writeln; Writeln;
Writeln (" 1) Yes 1:45");
Writeln (" 2) No 1:45");
Read (c); Readln; Writeln (c:45);
if (c = '1') then Info.Want_Dix_Grow := true
else if (c = '2') then Info.Want_Dix_Grow := false;
Until (c = '1') or (c = '2'))
end; (X if allwell X)
end;
procedure Init_Reader;
begin
  Reset (Intext, 'Intext text a');
end;

procedure Init_Researcher;
var
  i,
  stepcount,
  threshold,
  dixlength : integer;
  entry : entrytype;
  balanced : boolean;
begin
  Reset (Dixionary, 'Dixionary text a');
  Rewrite (Newwords, 'Newwords text a');
  if Info.Want_Concord then Rewrite (ConcordDoc, 'ConcordDoc text a');
  if (Info.frequency <> When_Done) then begin
    Dixlength := 0;
    while not EOF (Dixionary) do begin
      Readln (Dixionary); Dixlength := dixlength + 1;
    end;
    Reset (Dixionary, 'Dixionary text a');
  end;
  case Info.Frequency of
    Frequently : threshold := frequm;
    Seldomly : threshold := selnum;
    When_Done : threshold := WhenDonenum;
  end; (* cases *)
  New (Root);
  Load (2, Entry);
  Root.left := nil; Root.right := nil;
  Root.down := nil;
  Root.balance := Zero;
  Root.status := Storeable;
  Lastword := nil;
  Stepcount := 0;
  while not EOF (Dixionary) do begin
    Load (2, Entry);
    Entry.balance := Zero; balanced := false;
    AVL_Insert (Entry, Root, balanced);
    if (Info.frequency <> When_Done) then begin
      stepcount := stepcount + 1;
      if (stepcount mod threshold = 0) then begin
        Pigs; WriteIn; WriteIn; WriteIn; WriteIn;
        WriteIn ('Stylist is now loading its dictionary.
           (50); WriteIn; WriteIn;
        WriteIn ('(50); WriteIn (entry.word=100); WriteIn; WriteIn;
        WriteIn ('Stylist has loaded (40, stepcount=5, ' entries.');
        WriteIn ('(40, stepcount=5,' words.');
        WriteIn; WriteIn;
        WriteIn ('Stepcount=100 div Dixlength=33, ' % complete.');
        end; (* if M *)
      end; (* if M *)
    end; (* while not EOF *)
with tally do begin
  totalletters := 0;
  totalwords := 0;
  totalsentences := 0;
  totalstructes := 0;
  numwordsthissent := 0;
  for i := 0 to wordlength do 1straperword(.i.) := 0;
  for i := 0 to maxsent do wordspansent(.i.) := 0;
  KnownWords := 0;
  NewWords := 0;
  Numfound := 0;
  VariantWords := 0;
  StructureWords := 0;
  UnknownWords := 0;
  NumGermanic := 0;
  NumPostGrad := 0;
  NumGrad := 0;
  NumHigh_School := 0;
  NumElementary := 0;
  NumTangible := 0;
  NumIntangible := 0;
  NumSublime := 0;
  NumPleasant := 0;
  NumNeutral := 0;
  NumUnpleasant := 0;
  NumHorrid := 0;
  NumViolent := 0;
  NumEnergetic := 0;
  NumCalm := 0;
  NumInert := 0;
end; (* with tally do *)

Reset (FourDia, 'FourDia_text.a');
New (FourRoot); FourRoot-.right := nil; FourRoot-.left := nil;
Load (1, FourRoot-);
Attach (FourRoot-.right, FourRoot-);
FourRoot-.balance := Zero;
While not EOF (FourDia) do begin
  Load (1, Entry);
  Entry.balance := Zero;
  balanced := false;
  AVL_Insert (Entry, FourRoot, balanced);
end;

end;

62
procedure Load (filename : integer; var entry : entrytype);
var
  source : char;
  difficulty, concreteness, emotion, vigor : char;
begin
  if (filename = 1) then Readln (FourDix, Entry.word, difficulty, source, concreteness, emotion, vigor)
  else Readln (Dictionary, Entry.word, source, difficulty, concreteness, emotion, vigor);
  with entry do begin
    if (source = 'l') then source := Latinate
    else source := Germanic;
    case difficulty of
      'p' : difficulty := Postgrad;
      'g' : difficulty := Grad;
      'h' : difficulty := High_School;
      'e' : difficulty := Elementary;
    end;
    if (concreteness = 't') then concreteness := Tangible
    else concreteness := Intangible;
    case emotion of
      's' : emotion := Sublime;
      'p' : emotion := Pleasant;
      'n' : emotion := Neutral;
      'u' : emotion := Unpleasant;
      'h' : emotion := Horrid;
    end;
    case vigor of
      'v' : vigor := Violent;
      'e' : vigor := Energetic;
      'c' : vigor := Calm;
      'i' : vigor := Inert;
    end;
  end; (* with entry do *)
end;
procedure AVL_Insert (Entry : entrytype; var p : Entrypointer;
var balanced : boolean);

var
pl, p2 : entrypointer;
begin
  if (p = nil) then begin
    Attach (Entry, p);
    p-.balance := Zero;
    balanced := true;
  end
  else if (Entry-.word < p-.word) then begin
    AVL_Insert (Entry, p-.left, balanced);
    if Balanced then (as left pointer has grown higher)
      case p-.balance of
        Plus : begin p-.balance := Zero; balanced := false; end;
        Zero : p-.balance := Minus;
        Minus : begin (as rebalance)
          pl := p-.left;
          if (pl-.balance = Minus) then begin (as single LL rotate)
            p-.left := pl-.right;
            pl-.right := p;
            p-.balance := Zero;
            p := pl;
          end (as else)
        end; (as double LR rotation)
        else begin (as)
          p2 := p-.right;
          pl-.right := p2-.left;
          p2-.left := pl;
          p-.left := p2-.right;
          p2-.right := p;
          if (p2-.balance = Minus) then p-.balance := Plus
          else p-.balance := Zero;
          if (p2-.balance = Plus) then pl-.balance := Minus
          else pl-.balance := Zero;
          p := p2;
        end (as if)
        p-.balance := Zero; balanced := false;
      end (as case of Minus)
  end (as of cases)
end (as if
else if (Entry.word > p-.word) then begin
    AVL_Insert (Entry, p-.right, balanced);
    if balanced then (* right pointer has grown higher *)
    case p-.balance of
        Minus : begin p-.balance := Zero;
            balanced := false; end;
        Zero : p-.balance := Plus;
        Plus : begin (* rebalance *)
            pl := p-.right;
            if (pl-.balance = Plus) then begin
                (* single RR *)
                p-.right := pl-.left;
                pl-.left := p;
                p-.balance := Zero;
                p := pl;
            end (* of cases *)
        end (* of if Entry.word < p-.word *)
    end (* of balanced *)
end (* of procedure AVL_Insert *)

procedure Init_Report;
begin
    Rewrite (Report, 'Report text a!);
    Writeln (Report); Writeln(Report); Writeln(Report);
    Writeln (Report, Info.Name_of_Text, 45);
    Writeln (Report, 'PROFILE145);
    Writeln (Report), Writeln(Report); end;
procedure Read_Intext;
var
  letternum, wordnum, threshold, stepcount, offset : integer;
  c : char;
  n, readtype;
begin
  letternum := 0;
  wordnum := 0;
  stepcount := 0;
  case Info.Frequency of
    Frequently : threshold := freenum;
    Seldomly : threshold := selnum;
    When_Done : threshold := WhenDoneunum;
  end; (* case *)
  offset := ord ('A') - ord ('a');
  while not EOF (Intext) do begin
    while not EOLN (Intext) and Allswol do begin
      Read (Intext, c);
      if (c in ('.', 'Z', 'J', 'Z')) then c := chr(ord(c) - offset);
      if (c in ('a', 'z', 'j', 'z')) or (c in ('A', 'Z', 'J', 'Z')) then begin
        letternum := letternum + 1;
        if (letternum <= wordlength) then n.word(letternum) := c;
      end;
      if (letternum > 0) then begin
        if (c = ' ') or (c = '!') or EOLN (Intext) or
          (c = '#'') or (c = '!') or (c = ';') then begin
          for i := letternum + 1 to wordlength do n.word(i) := ' ';
          if (letternum > wordlength) then n.length := wordlength
          else n.length := letternum;
          Store_Intext (n);
          Wordnum := Wordnum + 1;
          letternum := 0;
          if (c = ' ') or (c = '#') or (c = '!') or (c = ';') then begin
            if (wordnum > 1) then begin (* not embedded command *)
              Signal_End_of_Sentence;
              Wordnum := 0;
            end;
            stepcount := stepcount + 1;
            if (stepcount mod threshold = 0) then begin
              Page; Writeln; Writeln; Writeln; Writeln;
              Write ('Stylist is now reading ');
              Write (Info.Name_of_text);
              Writeln ('Last word read was ' , n.word);
              Writeln; Writeln;
              Write ('Stylist has read ', stepcount);5);
              Writeln ('words. ');Writeln;
              Writeln; Writeln;
              Writeln (Stepcount*100 div Info.Size);1, % complete.');
            end; (* if *)
          end;
        end; (* while not EOF *)
        Signal_End_of_File;
      end;
      if (n.readtype = 0) then begin
        Page; Writeln; Writeln; Writeln; Writeln;
        Write ('Stylist is now reading ');
        Write (Info.Name_of_text);
        Writeln ('Last word read was ' , n.word);
        Writeln; Writeln;
        Write ('Stylist has read ', stepcount);5);
        Write ('words. ');Writeln;
        Writeln; Writeln;
        Writeln (Stepcount*100 div Info.Size);1, % complete.');
      end; (* if *)
    end; (* Page *)
  end; (* while *)
end; (* Read_Intext *)
procedure Store_Intext (n : readtype);
  var cp : concordpointer;
    found : boolean;
    entry : entrytype;
begin
  tally.totalwords := tally.totalwords + 1;
  tally.totalletters := tally.totalletters + n. length;
  tally.numwordsthissent := tally.numwordsthissent + 1;
  found := false;
  if (n. length < 5) then Look_Up_Four (n, FourRoot, found);
  if not found then
    tally.LtPerWord(n. length, ) := tally.LtPerWord(n. length, ) + 1;
  if not found then Look_Up_Four (n, Root);
  end;

procedure Signal_End_of_Sentence;
begin
  tally.totalsentences := tally.totalsentences + 1;
  if (tally.totalsentences > maxsent) then begin
    Write ("INTEXT TOO BIG.");
    Allwell := false;
  end;
  else tally.WordsPerSent(tally.totalsentences, ) :=
    tally.numwordsthissent;
  tally.numwordsthissent := 0;
end;

procedure Signal_End_of_File;
begin
  if Info.Want_Concord then Lastword := nil;
  tally.numfound := tally.totalwords - tally.unknownwords;
  Page := WriteIn; WriteIn; WriteIn; WriteIn; WriteIn;
  Write ("Stylist has finished reading " );
  Write (Info.Name_of_text);
end;

procedure Look_Up_Four (n : readtype; var ancestor : entrypointer);
  var cp : concordpointer;
    entry : entrytype;
begin
  if (ancestor = nil) then found := false
  else if (ancestor-..word = n. word) then begin
    found := true;
    if (ancestor-.vigor <> inert) then Add_Values (ancestor);
    tally.totalstrucltrs := tally.totalstrucltrs + n. length;
    tally.structurewords := tally.structurewords + 1;
    if Info.Want_Concord then
      Attach_Concord_Node (ancestor, cp);
  end;
  else if (ancestor-.word > n. word) then
    Look_Up_Four (n, ancestor-.left, found)
  else Look_Up_Four (n, ancestor-.right, found);
end;
procedure Look_Up (n : readtype; var ancestor : entrypointer);
var cp : concordpointer;
  entry : entrytype;
p : entrypointer;
found : boolean;
char : char;
begin
  if ancestor = nil then begin
    Look_Up_Variants (n, p, found);
    if found then begin
      tally.variantwords := tally.variantwords + 1;
      New (ancestor);
      with ancestor do begin
        word := n.word;
        left := nil; right := nil;
        down := nil;
      end; (* with ancestor do *)
      if (p-.status <> not_valueable) then begin
        Add_Values (p);
        with ancestor do begin
          status := not_storable;
          source := p-.source;
          difficulty := p-.difficulty;
          concreteness := p-.concreteness;
          emotion := p-.emotion;
          vigor := p-.vigor;
        end; (* with ancestor do *)
      end else (* p-.status is other than not_valueable *)
      ancestor-.status := not_valueable;
    end if Info.Want_Dix_Grow then begin
      Repeat
        Read (c); Readln;
        Until (c = '1') or (c = '2');
      if (c = '1') then begin
        Entry.word := n.word;
        Get_Values_for_New_Entry (Entry);
        Attach (Entry, ancestor);
        tally.newwords := tally.newwords + 1;
        Add_Values (ancestor);
        end else Attach_No_Value (ancestor, n);
      end if Info.Want_Concord then
      Attach_Concord_Node (ancestor, cp);
      end (* in ancestor = nil *)
    else if (ancestor-.word = n.word) then begin
      if (ancestor-.status <> not_storable) then
        tally.knownwords := tally.knownwords + 1
      else tally.unknownwords := tally.unknownwords + 1;
      if (ancestor-.status <> not_valueable) then
        Add_Values (ancestor);
      if Info.Want_Concord then
      Attach_Concord_Node (ancestor, cp);
      end (* if *)
    else if (ancestor-.word > n.word) then
      Look_Up (n, ancestor-.left);
    else Look_Up (n, ancestor-.right);
  end;
procedure Attach (Entry : entrytype; var p : entrypointer);
begin
  New (p);
  with p do begin
    word := entry.word;
    status := storables;
    source := entry.source;
    difficulty := entry.difficulty;
    concretaness := entry.concretaness;
    emotion := entryemotion;
    vigor := entry.vigor;
    left := nil; right := nil;
    down := nil;
  end;
end;

procedure Attach_concord_node (p : entrypointer;
                             var cp : concordpointer);
var Bottom,
    Bottommost : concordpointer;
begin
  Bottom := p.down;
  while (Bottom <> nil) do begin
    Bottommost := Bottom;
    Bottom := Bottom_.down;
  end;
  New (cp);
  with cp do begin
    Up := p;
    if (Lastward = nil) then last := nil
    else last := Lastward;
    Down := nil;
  end;
  if (P_.down = nil) then p.down := cp
  else Bottommost_.down := cp;
  if (Lastward <> nil) then
    cp_.last_.next := cp;
  lastward := cp;
end;

procedure Attach_no_value (var ancestor : entrypointer;
                          n : readtype);
begin
  writeln (newwords, n.word);
  New (ancestor); Ancestor_.word := n.word;
  tally.unknownwords := tally.unknownwords + 1;
  Ancestor_.right := nil; Ancestor_.left := nil;
  Ancestor_.down := nil;
  Ancestor_.status := not_valueable;
end;

69
procedure Look_Up_Variants (original : readtype; var p : entrypointer; var variant : readtype; var found : boolean);
begin
found := false;
if (original.length > 3) then
if (original.word(original.length.) = 's') then begin
variant := original;
variant.word(original.length.) := 's';
variant.length := original.length - 1;
Look_Up_Variant (variant, Root, found, p);
end;
if not found and (variant.word(variant.length.) = 'e') then begin
variant.word(variant.length.) := 'e';
variant.length := variant.length - 1;
Look_Up_Variant (variant, Root, found, p);
end;
end;
else if (original.length > 4) then
if (original.word(original.length.) = 'g') and
(original.word(original.length - 1) = 'n') and
(original.word(original.length - 2) = 'i') then begin
variant := original;
for i := original.length - 2 to original.length do
variant.word(i) := 'i';
variant.length := original.length - 3;
Look_Up_Variant (variant, Root, found, p);
end;
end;
end;
else if (original.length > 4) then
  if (original.word(original.length) = 'd') and
    (original.word(original.length - 1) = 'e') then begin
    variant.word := original.word;
    for i := original.length - 1 to original.length do
      variant.word(i) := 'i';
    variant.length := original.length - 2;
    Look_Up_Variant (variant, Root, found, p);
  end;
else if (original.length > 4) then
  if (original.word(original.length) = 'y') and
    (original.word(original.length - 1) = 'i') then begin
    variant.word := original.word;
    for i := original.length - 1 to original.length do
      variant.word(i) := 'i';
    variant.length := original.length - 2;
    Look_Up_Variant (variant, Root, found, p);
  end;
end;
end;
procedure Look_Up_Variant (variant : readtype;
var ancestor : Entrypointer;
var Found : Boolean;
var p : Entrypointer);

begin
  if (ancestor = nil) then found := false
else if (ancestor.word = variant.word) then begin
  found := true;
p := ancestor;
end
else if (ancestor.word > variant.word) then
  Look_Up_Variant (variant, ancestor.left, found, p)
else Look_Up_Variant (variant, ancestor.right, found, p)
end;

procedure Add_Values (p : entrypointer);
begin
  if (p-.source = Latinate) then
    Tally.NumLatinate := Tally.NumLatinate + 1
  else Tally.NumGermanic := Tally.NumGermanic + 1;
  case p-.difficulty of
    PostGrad : Tally.NumPostGrad := Tally.NumPostGrad + 1;
    Grad : Tally.NumGrad := Tally.NumGrad + 1;
    High_School : Tally.NumHigh_School := Tally.NumHigh_School + 1;
    Elementary : Tally.NumElementary := Tally.NumElementary + 1;
  end;
  if (p-.concreteness = Intangible) then
    Tally.NumIntangible := Tally.NumIntangible + 1
  else Tally.NumTangible := Tally.NumTangible + 1;
  case p-.emotion of
    Sublime : Tally.NumSublime := Tally.NumSublime + 1;
    Pleasant : Tally.NumPleasant := Tally.NumPleasant + 1;
    Neutral : Tally.NumNeutral := Tally.NumNeutral + 1;
    Unpleasant : Tally.NumUnpleasant := Tally.NumUnpleasant + 1;
    Horrid : Tally.NumHorrid := Tally.NumHorrid + 1;
  end;
  case p-.vigor of
    Violent : Tally.NumViolent := Tally.NumViolent + 1;
    Energetic : Tally.NumEnergetic := Tally.NumEnergetic + 1;
    Calm : Tally.NumCalm := Tally.NumCalm + 1;
    Inert : Tally.NumInert := Tally.NumInert + 1;
  end;
end;
procedure Get_Vvalues_for_New_Entry (var Entry : entrytype);
var a : char;
good_char : boolean;
begin
Repeat
  a := chr (128);
  writeln; writeln; writeln;
  writeln ('SOURCE :45); writeln; writeln;
  writeln (Entry.word:45);
  writeln; writeln;
  writeln ('1) LATINATE :45);
  writeln ('2) GERMANIC :45);
  read (c); writeln (c:45); readin;
  until (c = '1') or (c = '2');
  if (c = '1') then Entry.source := Latinate
    else Entry.source := Germanic;
Repeat
  writeln; writeln; writeln;
  writeln (Entry.word:45);
  writeln; writeln;
  writeln ('1) POSTGRADUATE :45);
  writeln ('2) GRADUATE :45);
  writeln ('3) HIGH SCHOOL :45);
  writeln ('4) ELEMENTARY :45);
  read (c); writeln (c:45); readin;
  until (c = '1') or (c = '2') or (c = '3') or (c = '4');
  case c of
    '1' : Entry.difficulty := PostGrad;
    '2' : Entry.difficulty := Grad;
    '3' : Entry.difficulty := High_School;
    '4' : Entry.difficulty := Elementary;
  end;
Repeat
  writeln; writeln; writeln;
  writeln (Entry.word:45);
  writeln; writeln;
  writeln ('1) TANGIBLE :45);
  writeln ('2) NOT TANGIBLE :45);
  read (c); readin; writeln; writeln (c:45);
  until (c = '1') or (c = '2');
  if (c = '1') then Entry.concreteness := Intangible
    else Entry.concreteness := Tangible;
Repeat
  writeln; writeln; writeln;
  writeln (Entry.word:45);
  writeln; writeln;
  writeln ('1) SUBLIME :45);
  writeln ('2) PLEASANT :45);
  writeln ('3) NEUTRAL :45);
  writeln ('4) UNPLEASANT :45);
  writeln ('5) HORRID :45);
  read (c); readin; writeln; writeln (c:45);
  until (c = '1') or (c = '2') or (c = '3') or (c = '4') or (c = '5');
  case c of
    '1' : Entry.emotion := Sublime;
    '2' : Entry.emotion := Pleasant;
    '3' : Entry.emotion := Neutral;
    '4' : Entry.emotion := Unpleasant;
    '5' : Entry.emotion := Horrid;
  end;
end;
Repeat
Page; Writeln; Writeln; Writeln;
Writeln ('VIGOR'); Writeln; Writeln;
Writeln ('Entry.word'); Writeln; Writeln;
Writeln ("1" VIOLENT '1':45));
Writeln ('2" ENERGETIC '2':45));
Writeln ("3" CALM '3':45));
Writeln ("4" NONE '4':45));
Read (c); Writeln (c:45)); Readln;
Until (c = '1') or (c = '2') or (c = '3') or (c = '4'))
  case c of
    '1': Entry.vigor := Violent;
    '2': Entry.vigor := Energetic;
    '3': Entry.vigor := Calm;
    '4': Entry.vigor := Inert;
  end;
end;

procedure Write_Concordance;
beg
Page; Writeln; Writeln; Writeln; Writeln;
Write ('Stylist is now writing the concordance for ');
Write (Info.Name_of_text));
Writeln; Writeln;
Write ('Concordance will be written to "Concordance text a"');
Writeln; Writeln;
Write (Concord); Writeln (Concord);
Write (Concord, 'CONCORDANCE':55);
Write (Concord, Info.Name_of_text:55);
Write (Concord); Writeln (Concord);
Inorder_Concordance (Root);
end;
procedure Inorder_Concordance (var Root : entrypointer);
begin
  if (Root <> nil) then begin
    Inorder_Concordance (Root-.left);
    w := 0;
    Lastward := Downward-.last;
    while (Lastward <> nil) and (w < Phraselength) do begin
      w := w + 1;
      Phrasel (.w.) := Lastward-.up-.word;
      Lastward := Lastward-.last;
    end;
    for i := w downto 1 do begin
      l := 1;
      while (l <= wordlength) do begin
        if (Phrasel (.i.)(.1.) <> ' ') then
          Write (Concondno, Phrasel (.i.)(.1.));
        l := l + 1;
      end;
      Write (Concondno, ' '); l := l + 1;
    end;
    Write (Concondno, ' ');
    c := Root-.word(.1.);
    if (c in ('a','i','u')) then
      c := chr(ord(c) + offset);
    if (Root-.word(.1.) <> ' ') then
      Write (Concondno, c);
    l := l + 1;
    Write (Concondno, ' ');
  end;
end;
}
for i := 1 to w do begin
  i := i;
  while (i <= wordlength) do begin
    if (Phrase2(i,i) <> ' ') then
      Write (Concord, Phrase2(i,i));
    i := i + 1;
  end;
  Write (Concord, ', ');
end;
Write (Concord, ' ');
Write (Concord);
Downward := Downward . down;
end;
Inorder_Concordance (Root . right);
end; (* if Root <> nil *)
end; (* procedure Inorder_Concordance *)
procedure Calculate_Profile;
var numsigltras : integer; (* number of significant letters *)
numsigwrds : integer; (* number of significant words *)
(num in this context, structural words are insignificant *)
begin
  if (tally.totalsentences = 0) then tally.totalsentences := 1;
  if (tally.totalwords = 0) then tally.totalwords := 1;
  numsigwrds := tally.totalwords - tally.structurewords;
  numsigltras := tally.totalletters - tally.totalstructltras;
  Profile := tally;
  profile.Ave_1trs_per_word := numsigltras/numsigwrds;
  profile.Ave_wrds_per_sent := tally.totalwords/tally.totalsentences;
end;

procedure Analyze_Profile_and_Report; (* main Reporter procedure *)
begin
  Page; Writeln; Writeln; Writeln; Writeln;
  Write ('Stylist is now analyzing the style of ');
  Writeln ('Info.Name_of_text');
  Writeln; Writeln;
  Writeln ('Report of analysis will be written to Report text as"Report text as"');
  Writeln (Report);
  Writeln (Report); Writeln (Report);
  Writeln (Report); Writeln (Report);
  Calculate_Size_of_Text (profile, analysis);
  Calculate_Length_of_Words (profile, analysis);
  Calculate_Length_of_Sentences (profile, analysis);
  Calculate_Phyiology (profile, analysis);
  Calculate_Difficulty_of_Vocabulary (profile, analysis);
  Calculate_Tangibility (profile, analysis);
  Calculate_Emoational_Tone (profile, analysis);
  Calculate_Vigor_of_Words (profile, analysis);
  Make_Recomandations (analysis);
end;
procedure Calculate_Size_of_Text (profile : profiletype);  
var analysis : analysisistype);  
begin  
PercentFound : integer;  
Write (Report), Write (Report), Write (Report);  
Write (Report, 'Total of sentences : ');  
Write (Report, profile.Totalsentences);  
Write (Report, 'Total of words : ');  
Write (Report, profile.Totalwords);  
Write (Report, 'Total of letters : ');  
Write (Report, profile.totalletters);  
Write (Report, profile.totalwords);  
Write (Report, 'Total of words ');  
Write (Report, profile.totalwords, ' words are ');  
Write (Report, ' too few for valid statistical analysis. ');  
Write (Report, profile.totalwords, ' words are ');  
Write (Report, ' enough for valid statistical analysis. ');  
Write (Report, profile.totalwords, ' words are ');  
Write (Report, ' plenty for valid statistical analysis. ');  
Write (Report, 'Of the ', profile.totalwords, ' words in ');  
Write (Report, Info.Name_of_Text, ' ');  
Write (Report, profile.numFound);  
Write (Report, ' were matched to words in the Stylist dictionary. ');  
Write (Report, profile.numFound, ' % were matched. ');  
Write (Report, info.Name_of_Text, ' ');  
Write (Report, ' Statistical analysis is not valid. ');  
Write (Report, ' This is enough for statistical analysis. ');  
Write (Report, profile.newwords);  
Write (Report, ' to the dictionary during this session. ');  
e  
end;  
end;  
end;  
end;  
end;  
end;  
end;
procedure Calculate_Length_of_Words (profile : profiletype;
     var analysis : analysistype);
begin
  Graph_LtrsPerWord (profile.ltrspeword);
  Write (Report, 'The average number of letters per word: \');
  Write (Report, profile.Ave_Ltrs_per_word);
  Write (Report, 'The length of the words is: '); case analysis.WordLength of
    TooShort : Write (Report, 'too short. ');
    Short : Write (Report, 'short. ');
    medium : Write (Report, 'medium. ');
    Long : Write (Report, 'long. ');
    Toolong : Write (Report, 'too long. ');
  end;
  Write (Report); Write (Report); Write (Report);
end;
procedure Graph_LtrPerWord (LtrPerWord : LtrPerWordType);  
var perLtrWords : LtrPerWordType;  
i, j : integer;  
begin  
  Page (Report);  
  Writeln (Report);  
  Writeln (Report);  
  Writeln (Report);  
  Writeln (Report);  
  Writeln (Report);  
  for i := 0 to wordlength do  
    PerLtrWords (.i.) := (LtrPerWord(.i.) x 100) div  
      (tally.totalwords - tally.structuredwords);  
  for i := 15 downto 1 do begin  
    Write (Report,'1:5');  
    for j := 1 to wordlength do  
      if (perLtrWords(j) >= 1) then  
        Write (Report,'1:5')  
      else if (perLtrWords(j) >= (i x 2) - 1) then  
        Write (Report,'1:5')  
      else  
        Write (Report,'1:5');  
    Write (Report);  
    Write (Report,'1:4');  
    for i := 1 to wordlength do  
      Write (Report,'1:5');  
  end;  
end;
procedure Calculate_Length_of_Sentences ( profile : profiletypen; 
var analysis : analysisiype); 
var 
  i, 
  Sweetspots, 
  Runons, 
  PercentRunons, 
  PercentSweetSpots : integer; 
begin 
  Graph_WordsParSent (profile.WordsPerSent); 
  Write (Report, 'The average number of words per sentence : '); 
  Write (Report, profile.Ave_words_per_sent); 
  Write (Report); 
  Write (Report, 'A typical modern text sentences average '); 
  if (profile.ave_words_per_sent > 22) then 
    Analysis.SentLength := TooLong 
  else if (profile.ave_words_per_sent > 18) then 
    Analysis.SentLength := Long 
  else if (profile.ave_words_per_sent > 15) then 
    Analysis.SentLength := Medium 
  else if (profile.ave_words_per_sent > 10) then 
    Analysis.SentLength := Short 
  else Analysis.SentLength := TooShort; 
  for i := 1 to profile.totalSentences do 
    if (profile.wordsperSent(i,i) > 45) then 
      runons := runons + 1; 
    else if (profile.wordsperSent(i,i) > 8) and 
      (profile.wordsperSent(i,i) < 20) then 
      Sweetspots := sweetspots + 1; 
  PercentRunons := runons*100 div profile.totalSentences; 
  if (PercentRunons > 5) then Analysis.Runons := unacceptable 
  else if (PercentRunons > 0) then Analysis.Runons := acceptable 
  else Analysis.Runons := nonexistent; 
  for i := 1 to profile.totalSentences do 
    if (profile.wordsperSent(i,i) > 8) and 
      (profile.wordsperSent(i,i) < 20) then 
      Sweetspots := sweetspots + 1; 
  PercentSweetSpots := sweetspots*100 div profile.totalSentences; 
  if (PercentSweetSpots > 50) then Analysis.Modulation := Good 
  else if (PercentSweetSpots > 20) then Analysis.Modulation := Average 
  else Analysis.Modulation := bad; 
  Write (Report); 
  Write (Report, 'Number of medium length sentences : ', sweetspots); 
  Write (Report, 'Percent of medium length sentences : '); 
  Write (Report, percentSweetspots); 
  Write (Report);
Write (Report, 'Modulation is ');
  case Analysis.Modulation of
    Good : Writeln (Report, ' good.');
    Average : Writeln (Report, ' average.');
    Bad : Writeln (report, 'bad');
  end;
end; (* procedure analyze length of sentences *)
procedure Graph_WordsPerSent (Graph : WordsPerSenttype);
var i, j, pagenum : integer;
begin
  Pagenum := 1;
  While (Tally.totalsentences > (Pagenum - 1) * 70) do begin
    Page (Report);
    Writeln (Report, 'BREAKDOWN OF NUMBER OF WORDS PER SENTENCE':60);
    Writeln (Report, 'Number of words');
    for i := 1 * ((Pagenum - 1) * 70) to 70 * (Pagenum) do
      Graph(i) := (Graph(i) div 2);
    for i := 50 downto 1 do begin
      Write (Report, 'BREAKDOWN OF NUMBER OF SENTENCES':60);
      Write (Report, 'Number of words');
      for j in 1 * ((Pagenum - 1) * 70) to 70 * (Pagenum) do
        if (Graph(j) >= i) then
          Write (Report, 'X':1) else Write (Report, ' ':1);
      Write (Report, 'BREAKDOWN OF NUMBER OF SENTENCES':60);
      Write (Report, 'Number of words');
      for i := 70 downto 1 do begin
        Write (Report, i * 10 + ((Pagenum - 1) * 70):10);
      end;
    end;
  end;
end;

procedure Calculate_Etymology (profile : profiletype);
var analysis : analysistype;
begin
  Writeln (Report); Writeln(Report); Writeln(Report);
  Writeln(Report, 'ETYMOLOGY OF WORDS':45);
  Writeln(Report); Writeln(Report);
  Write (Report, 'Number of Latinate words ':1);
  Writeln (Report, profile.NumLatinate);
  Write (Report, 'Number of Germanic words ':1);
  Writeln (Report, profile.NumGermanic);
  if (Info.Genre = NonFiction) then begin
    if (profile.NumLatinate > profile.NumGermanic * 3) then
      Analysis.Etymology := TooBorrowed
    else if (profile.NumLatinate > profile.NumGermanic * 2) then
      Analysis.Etymology := Borrowed
    else if (profile.NumLatinate > profile.NumGermanic) then
      Analysis.Etymology := Mixed
    else if (profile.NumLatinate > profile.NumGermanic * 0.9) then
      Analysis.Etymology := Native
    else Analysis.Etymology := TooNative;
  end
  else if (genre = Fiction) begin
    if (profile.NumLatinate > profile.NumGermanic * 2) then
      Analysis.Etymology := TooBorrowed
    else if (profile.NumLatinate > profile.NumGermanic * 1.5) then
      Analysis.Etymology := Borrowed
    else if (profile.NumLatinate > profile.NumGermanic) then
      Analysis.Etymology := Mixed
    else if (profile.NumLatinate > profile.NumGermanic * 0.5) then
      Analysis.Etymology := Native
    else Analysis.Etymology := TooNative;
  end;
  Write (Report, 'Etymology is ':1);
  case Analysis.Etymology of
    TooBorrowed : Writeln (Report, 'very borrowed.');
    Borrowed : Writeln (Report, 'borrowed.');
    Mixed : Writeln (report, 'mixed');
    Native : Writeln (Report, 'native.');
    TooNative : Writeln (Report, 'very native.');
  end;
end; (* procedure calculate etymology *)
procedure Calculate_Difficulty_of_Vocabulary (profile : profiletype;
   var analysis : analysistype);
var
PercentPostGrad,
PercentGrad,
PercentHigh : integer);
begin
  Writeln (Report); Writeln(Report); Writeln(Report);
  Writeln(Report, 'DIFFICULTY OF VOCABULARY', '45');
  Writeln(Report); Writeln(Report);
  Writeln(Report, 'PostGraduate difficulty : ');  
  Writeln(Report, profile.NumPostGrad);
  Writeln(Report, 'Graduate difficulty : '); 
  Writeln(Report, profile.NumGrad);
  Writeln(Report, 'High School difficulty : '); 
  Writeln(Report, profile.NumHigh_School);
  Writeln(Report, 'Elementary difficulty : '); 
  Writeln(Report, profile.NumElementary);
  Writeln(Report); Writeln(Report);
  parcentPostGrad := profile.numpostgrad*100 div profile.numfound;
  percentGrad := profile.numgrad*100 div profile.numfound;
  percentHigh := profile.numhigh_school*100 div profile.numfound;
  if (parcentPostGrad > 0) then analysis.HardWords := veryhard 
  else if (parcentgrad > 5) then analysis.HardWords := hard
  else if (percenthigh > 10) then analysis.HardWords := challenging
  else analysis.HardWords := easy;
  Writeln(Report, 'Percent of Postgraduate difficulty ');
  Writeln(Report, 'Percent of Graduate difficulty ', percentgrad);
  Writeln(Report, 'Percent of High School difficulty ', percenthigh);
  Writeln(Report); Writeln(Report);
  case Analysis.HardWords of
    Veryhard: Writeln (Report, 'very hard');
    hard: Writeln (Report, 'hard');
    challenging: Writeln (report, 'challenging');
    easy: Writeln (report, 'easy');
  end;
end; (* Calculate_Difficulty_of_Vocabulary *)
procedure Calculate_Tangibility (profile : profiletype;  
                          var analysis : analysisistype); 
begin 
  Writeln (Report); Writeln(Report); Writeln(Report); 
  Writeln (Report, 'TANGIBILITY':45) ; 
  Write (Report, 'Number of Tangible words :'); 
  Writeln (Report, profile.NumTangible); 
  Write (Report, 'Number of Intangible words :'); 
  Writeln (Report, profile.NumIntangible); 
  if (Info.Genre = 'NonFiction') then begin 
    if (profile.NumIntangible > profile.NumTangible * 4) then 
      Analysis.Tangibility := 'Soft' 
    else if (profile.NumIntangible > profile.NumTangible * 2) then 
      Analysis.Tangibility := 'Firm' 
    else Analysis.Tangibility := 'Solid'; 
  end; 
  Writeln (Report); 
  Write (Report, 'Tangibility is '); 
  case Analysis.Tangibility of 
    'Solid' : Writeln (Report, 'very tangible.'); 
    'Firm' : Writeln (Report, 'tangible.'); 
    'Soft' : Writeln (Report, 'very intangible.'); 
  end; 
end; (* calculate tangibility *)
procedure Calculate_Emotional_Tone (profile : profiletype);
begin
  var
    PerCentSublime,
    PerCentPleasant,
    PerCentUnpleasant,
    PerCentHorrid,
    EmotionIndex : integer;
  begin
    Writeln (Report); Writeln(Report); Writeln(Report);
    Writeln (Report, 'EMOTIONAL CONNOTATIONS':45);
    Writeln (Report, 'Percent of sublime connotations ',PercentSublime);
    Writeln (Report, 'Percent of pleasant connotations ',PercentPleasant);
    Writeln (Report, 'Percent of unpleasant connotations ',PercentUnpleasant);
    Writeln (Report, 'Percent of horrid connotations ',PercentHorrid);
    EmotionIndex := (PercentSublime * 5) +
                    (PercentPleasant * 2) +
                    (PercentUnpleasant * 2) +
                    (PercentHorrid * 5);
    if (EmotionIndex > 20) then Analysis.emotionality := rich
    else if (EmotionIndex > 10) then Analysis.emotionality := standard
    else Analysis.emotionality := poor;
    Write (Report, 'Index of Emotionality ', EmotionIndex);
    Writeln (Report, 'Emotionality is '); case Analysis.Emotionality of
      Rich: Writeln (Report, 'Rich');
      Standard : Writeln (Report, 'Average');
      Poor: Writeln (report, 'Poor');
    end;
    if (((PercentSublime * 5) +PercentPleasant) >
        (PercentHorrid * 5) + percentUnpleasant) * 1.2) then
      Analysis.Tone := positive
    else if (((PercentSublime * 5) +PercentPleasant) * 1.2 <
             (PercentHorrid * 5) + percentUnpleasant)) then
      Analysis.Tone := negative
    else Analysis.Tone := bland;
    Write (Report, 'Tone is '); case Analysis.Tone of
      Positive: Writeln (Report, 'Positive');
      Bland: if (analysis.emotionality= rich) then begin
        Write (Report, 'a balance of strong positive ');
        Write (Report, 'and strong negative emotions.'); end else Writeln (Report, 'bland.'));
      negative: Writeln (report, 'negative');
    end;
end; (* procedure calculate emotional tone *)
procedure Calculate_Vigor_of_Words (profile : profiletype;
    var analysis : analysisistype);
var
    ParCentViolent,
    PercentEnergetic,
    PercentCalm,
    StrengthIndex : integer;
begin
    Writein (Report); Writeln(Report); Writeln(Report);
    Writain (Reports 'VIGOR OF WORDS't45)
    WriteinCRoport); WritainCReport);
    Writein (Report, profile.NumViolent); Writeln;
    Write (Raeport, 'Words of Extreme Vigor : ');
    Write (Report, profile.NumEnergetic);
    Write (Report, profile.NumCalm);
    Write (Report, 'Words of Much Vigor : ');
    Write (Report, profile.NumCalm);
    Write (Report, 'Words of Some Vigor : ');
    Write (Report, profile.NumCalm);
    Write (Report, 'Words of Little Vigor : ');
    Write (Report, profile.NumCalm);
    Writein (Report, profile.NumCalm);
    parcentViolent := profile.numviolent*100 div profile.numfound;
    percentEnergetic := profile.numenergetic*100 div profile.numfound;
    percentCalm := profile.numcalm*100 div profile.numfound;
    StrengthIndex := percentviolent * 10 +
                   percentenergetic * 5 +
                   percentcalm;
    if (StrengthIndex > 60) then Analysis.Strength := VeryStrong
    else if (StrengthIndex > 50) then Analysis.Strength := strong
    else if (StrengthIndex > 20) then Analysis.Strength := lively
    else Analysis.Strength := weak;
    Writein (Report, 'Percent of words of extreme vigor : ');
    Writein (Report, percentViolent);
    Write (Report, 'Percent of words of much vigor : ');
    Writein (Report, percentEnergetic);
    Write (Report, 'Percent of words of some vigor : ');
    Writein (Report, percentCalm);
    Write (Report, 'Index of Vigor : ', StrengthIndex);
    Write (Report, 'Vigor is ');
    case Analysis.Strength of
        VeryStrong: Writein (Report, 'very strong');
        strong: Writein (Report, 'strong');
        lively: Writein (report, 'lively');
        weak : Writein (Report, 'weak.'));
    end;
end; (* procedure calculate vigor of words *)

procedure Write_NumRecom (Var Numrecom : integer);
begin
    Numrecom := Numrecom + 1;
    Write (Report); Writeln(Report);
    Writein (Report, 'RECOMMENDATION NUMBER :55, Numrecom:2);
    Writeln(Report); Writeln(Report);
end;
procedure Make_Recommendations (analysis : analysisistype); 
begin
  NumRecs := 0;
  with analysis do begin
    if (Runons = Unacceptable) then begin
      Write (NumRecs, (NumRecs);
      Write (Report, 'You tend to write run-on sentences.');
      Write (Report, 'Check your longest sentences for run-ons.');
      Write (Report, 'Break them up into units of single ideas.');
    end;
    if (SentLength = TooLong) and (HardWords > Easy) then begin
      Write (NumRecs, (NumRecs);
      Write (Report, 'Your average sentences are too long');
      Write (Report, 'for the difficulty of your vocabulary.');
      Write (Report, 'Use simpler words or shorter sentences.');
    end;
    if (Modulation = Bad) then begin
      Write (NumRecs, (NumRecs);
      Write (Report, 'Your sentences tend to be too short or too');
      Write (Report, 'long. Try to moderate and modulate');
      Write (Report, 'the length of your sentences.');
    end;
    if (Etymology = TooBorrowed) then begin
      Write (NumRecs, (NumRecs);
      Write (Report, 'Use shorter, more native English words.');
    end;
    if (Emotionality = Poor) then begin
      Write (NumRecs, (NumRecs);
      Write (Report, 'Use more words that provoke emotions.');
    end;
    if (Info.Genre = Fiction) then begin
      if (Strength < Strong) then begin
        Write (NumRecs, (NumRecs);
        Write (Report, 'Use more evocative, sensory words.');
      end;
      if (Etymology < Mixed) and (Hardwords > Challenging) and
      (SentLength > Medium) then begin
        Write (NumRecs, (NumRecs);
        Write (Report, 'Your fiction reads like non-fiction.');
        Write (Report, 'Unless you are targeting a highly');
        Write (Report, 'literate audience, keep it simple.');
      end;
      if (Tangibility = Soft) then begin
        Write (NumRecs, (NumRecs);
        Write (Report, 'A narrative should be concrete and');
        Write (Report, 'detailed. Describe things, not ideas.');
      end;
    end else begin
      if (Strength = VeryStrong) then begin
        Write (NumRecs, (NumRecs);
        Write (Report, 'Your non-fiction reads like fiction.');
        Write (Report, 'Ask yourself if it is too vigorous');
        Write (Report, 'for the audience that will read it.');
      end;
      if (Strength = Weak) then begin
        Write (NumRecs, (NumRecs);
        Write (Report, 'Use more evocative, sensory words.');
      end;
    end;
  end;
end;
if (Strength > Lively) and (Emotionality > Standard) and
  (Modulation = Good) then begin
  Write (Report, 'Congratulations! You write with ');
  Writeln (Report, 'strength and grace. ');
  end;
if (Runons = nonexistent) then begin
  Write (Report, 'Congratulations! You never seem ');
  Writeln (Report, 'to write run-on sentences. ');
  end;
if (NumRscm = 0) then begin
  Writein (Report), Write (Report, 'This is a solid piece of writing ');
  Writein (Report, 'well within the traditions of its genre.'));
  Writein (Report);
  Write (Report, 'You are as able to understand the meaning of')
  Writein (Report, 'the above characteristics as The Stylist.');
  Writein (Report, 'Stand the course!');
  end;
end; (* with analysis do *)
end; (* procedure make recommendations *)
procedure Store_New_Dix;
begin
 Rewrite (Dixionary, 'Dixionary text a');
 Page; Writeln; Writeln; Writeln; Writeln;
 Write ('Stylist is now storing the new dictionary,';55);
 Store_BST (Root);
end;

procedure Store_BST (p : entrypointer);
begin
 if (p <> nil) then begin
  if (p-.status = storable) then Store (p-);
   Store_BST (p-.left);
   Store_BST (p-.right);
 end;
end;

procedure Store (Entry : entrytype);
var
 sourceltr,
   difficultyltr,
   concretenessltr, emotionltr,
   vigorltr : char;
begin
 with entry do begin
  if (source = Latinate) then sourceltr := 'l'
   else sourceltr := 'g';
   case difficulty of
    Postgrad : difficultyltr := 'p';
    Grad : difficultyltr := 'g';
    High_School : difficultyltr := 'h';
    Elementary : difficultyltr := 'e';
   end;
  if (concreteness = Tangible) then concretenessltr := 't'
    else concretenessltr := 'i';
   case emotion of
    Sublime : emotionltr := 's';
    Pleasant : emotionltr := 'p';
    Neutral : emotionltr := 'n';
    Unpleasant : emotionltr := 'u';
    Horrid : emotionltr := 'h';
   end;
   case vigor of
    Violent : vigorltr := 'v';
    Energetic : vigorltr := 'e';
    Calm : vigorltr := 'c';
    Inert : vigorltr := 'i';
 end;
end; (* with entry do *)
 Writeln (Dixionary, Entry.word, sourceltr, difficultyltr,
 concretenessltr, emotionltr, vigorltr);
end;
APPENDIX D
THE CODE OF DIXSPLIT AND DIXJOIN

(* NOTE: THE FOLLOWING IS NOT IN EXECUTABLE ORDER.
IT IS IN THE ORDER OF GREATEST CONCEPTUAL CLARITY. *)

(*$S00000*)
program Dixsplit (input, output);
const
  maxsent = 1000;
  wordlength = 15;
  linelen = 60;
  phraselength = 4;
  freenum = 50;
  solnum = 200;
type
  Genre-type = (Nonfiction, Fiction);
  Frequency-type = (Frequently, Seldomly, When_Done);
  Duration-type = (Permanent, Temporary);
  Source-type = (Latinate, Germanic);
  Difficulty-type = (PostGrad, Grad, High_School, Elementary);
  Concreteness-type = (Tangible, Intangible);
  Emotion-type = (Sublime, Pleasant, Neutral, Unpleasant, Horrid);
  Vigor-type = (Violent, Energetic, Calm, Inert);
  Word-type = packed array (1..wordlength.) of char;
  Linetype = packed array (1..linelen.) of char;
  Phrasetype = packed array (1..phraselength.) of wordtype;
  EntryPointer = -Entrytype;
  Concord-type = -Concordtype;
  Entrytype = record
    Word : Word-type;
    Balance : Balancetype;
    Duration : Duration-type;
    Source : Source-type;
    Difficulty : Difficulty-type;
    Concreteness : Concreteness-type;
    Emotion : Emotion-type;
    Vigor : Vigor-type;
    Left, Right : EntryPointer;
    Down : Concord-pointer;
  end;
  Concord-type = record
    Up : EntryPointer;
    Down,
    Next, Last : Concordpointer;
  end;

91
var
FourRoot,
Root : EntryPointer;
Lastward : ConcordPointer;
Latinat,
Germanat,
Tangable,
Intangible,
Postcard,
Graduate,
HighSchol,
Elements,
Sublime,
Pleasant,
Naevus,
Unpleasant,
Horrid,
Violent,
Energetic,
Cim,
Inrt,
Dictionary : text;

procedure Attach (Entry : entrytype; var p : entrypointer);
begin
  New (p);
  with p do begin
    word := entry.word;
    duration := entry.duration;
    source := entry.source;
    difficulty := entry.difficulty;
    concretteness := entry.concreteness;
    emotion := entry.emotion;
    vigor := entry.vigor;
    left := nil; right := nil;
    down := nil;
  end;
end;
procedure Encode (its : entrytype; var xsource : char; 
    var xdifficulty : char; 
    var xconcreteness : char; var xemotion : char; 
    var xvigor : char);
begin 
    if (its.source = Latinate) then xsource := 'l'
      else xsource := 'g'; 
    case its.difficulty of 
      Postgrad : xdifficulty := 'p'; 
      Grad : xdifficulty := 'g'; 
      High_School : xdifficulty := 'h'; 
      Elementary : xdifficulty := 'e'; 
    end; 
    if (its.concreteness = Tangible) then Xconcreteness := 't'
      else Xconcreteness := 'i'; 
    case its.emotion of 
      Sublime : xemotion := 's'; 
      Pleasant : xemotion := 'p'; 
      Nuetral : xemotion := 'n'; 
      Unpleasant : xemotion := 'u'; 
      Horrid : xemotion := 'h'; 
    end; 
    case its.vigor of 
      Violent : xvigor := 'v'; 
      Energetic : xvigor := 'e'; 
      Calm : xvigor := 'c'; 
      Inert : xvigor := 'i'; 
    end; 
end;

procedure Decode (var its : entrytype; 
    xsource : char; xdifficulty : char; 
    xconcreteness : char; xemotion : char; 
    xvigor : char);
begin 
    if (Xsource = 'l') then its.source := Latinate 
      else its.source := Germanic; 
    case Xdifficulty of 
      'p' : its.difficulty := Postgrad; 
      'g' : its.difficulty := Grad; 
      'h' : its.difficulty := High_School; 
      'e' : its.difficulty := Elementary; 
    end; 
    if (Xconcreteness= 't') then its.concreteness := Tangible 
      else its.concreteness := Intangible; 
    case Xemotion of 
      's' : its.emotion := Sublime; 
      'p' : its.emotion := Pleasant; 
      'n' : its.emotion := Nuetral; 
      'u' : its.emotion := Unpleasant; 
      'h' : its.emotion := Horrid; 
    end; 
    case XVigor of 
      'v' : its.vigor := Violent; 
      'e' : its.vigor := Energetic; 
      'c' : its.vigor := Calm; 
      'i' : its.vigor := Inert; 
    end; 
end;
procedure AVL_Insert (Entry : entrytype; var p : Entrypointer;
  var balanced : boolean);
var
  p1, p2 : entrypointer;
begin
  if (p = nil) then begin
    Attach (Entry, p);
    p-.balance := Zero;
    balanced := true;
  end
  else if (Entry.word = p-.word) then Writeln ('INSERT COLLISION')
  else if (Entry.word < p-.word) then begin
    AVL_Insert (Entry, p-.left, balanced);
    If Balanced then (X left pointer has grown higher X)
    case p-.balance of
      Plus : begin p-.balance := Zero; balanced := false; end;
      Zero : p-.balance := Minus;
      Minus : begin (X rebalance X)
        p1 := p-.left;
        if (p1-.balance = Minus) then begin (X single LL rotatX)
          p1-.left := p1-.right;
          p1-.right := p;
          p-.balance := Zero;
        p := p1;
        end; (X if X)
      else begin (X double LR rotation X)
        p2 := p1-.right;
        p1-.right := p2-.left;
        p2-.left := p1;
        p-.left := p2-.right;
        p2-.right := p;
        if (p2-.balance = Minus) then p-.balance := Plus
        else p-.balance := Zero;
        if (p2-.balance = Plus) then p1-.balance := Minus
        else p1-.balance := Zero;
        p := p2;
      end; (X else X)
    end; (X if Entry.word < p-.word X)
  end
  else begin (X single RR rotation X)
    if p-.balance = Minus then begin (X rebalance X)
      p-.balance := Zero; balanced := false;
    end; (X if p-.balance = Minus X)
  end; (X of cases X)
end; (X of cases X)
else if (Entry.word > p-.word) then begin
    AVL_Insert (Entry, p-.right, balanced);
    if balanced then (% right pointer has grown higher %)
        case p-.balance of
            Minus : begin p-.balance := Zero;
                balanced := false; end;
            Zero : p-.balance := Plus;
            Plus : begin (% rebalance %)
                pl := p-.right;
                if (pl-.balance = Plus) then begin
                    (% single RR rotation %)
                    p-.right := pl-.left;
                    pl-.left := p;
                    p-.balance := Zero;
                    p := pl;
                end (% if %)
                else begin (% double RL rotation %)
                    p2 := pl-.left;
                    pl-.left := p2-.right;
                    p2-.right := pl;
                    p-.right := p2-.left;
                    p2-.left := p;
                    if (p2-.balance = Plus) then
                        p-.balance := Minus
                    else p-.balance := Zero;
                    if (p2-.balance = Minus) then
                        pl-.balance := Plus
                    else pl-.balance := Zero;
                    p := p2;
                end (% double RL rotation %);
            end (% case of balance = Plus %)
        end (% of if Entry.word > p-.word %)
    end (% of if Entry.word > p-.word %)
else balanced := false;
end; (% of procedure AVL_Insert %)
procedure Init_Researcher;
var
  i, stepcount, threshhold,
  dixilength : integer;
  s, d, c, e, v : char;
  entry : entrytype;
  balanced : boolean;
begin
  Reset (Dixonary, 'Dixonary text a');
  New (Root);
  Readin (Dixonary, Root-.word, s, d, c, e, v);
  Decode (Root-.s, d, c, e, v);
  Root-.left := nil; Root-.right := nil;
  Root-.balance := Zero;
  Root-.duration := Permanent;
  Lastward := nil;
  Stepcount := 0;
  while not EOF (Dixonary) do begin
    Reset (Dixonary, Entry-.word, s, d, c, e, v);
    Decode (Entry-.s, d, c, e, v);
    Entry-.balance := Zero; balanced := false;
    AVL_Insert (Entry, Root, balanced);
    stepcount := stepcount + 1;
    if (stepcount mod 200 = 0) then begin
      Page; Writeln; Writeln; Writeln;
      Writeln ('Stylist is now loading its dictionary.');
      Writeln ('Last word loaded was ', entry-.word); Writeln;
      Writeln ('Stylist has loaded ', stepcount, ' entries.');
      Writeln; Writeln;
    end;
  end; (* if x)
end; (* while not EOF x)
procedure Inorder (var F : t : entrypointer);
begin
  if (Root <> nil) then begin
    Inorder (Root-.left);
    case Root-.Source of
      Latinat : Writeln (Latinat, Root-.word, 'l':1);
      Germanic : Writeln (Germanic, Root-.word, 'g':1);
    end;
    case Root-.Concreteness of
      Tangible : Writeln (Tangible, Root-.word, 't':1);
      Intangible : Writeln (Intangible, Root-.word, 'i':1);
    end;
    case Root-.difficulty of
      Postgrad : Writeln (Postgrad, Root-.word, 'p':1);
      Grad : Writeln (Graduate, Root-.word, 'g':1);
      High_School : Writeln (HighSchl, Root-.word, 'h':1);
      Elementary : Writeln (Elementary, Root-.word, 'e':1);
    end;
    case Root-.emotion of
      Sublime : Writeln (Sublime, Root-.word, 's':1);
      Pleasant : Writeln (Pleasant, Root-.word, 'p':1);
      Neutral : Writeln (Neutral, Root-.word, 'n':1);
      Unpleasant : Writeln (Unpleasant, Root-.word, 'u':1);
      Horrid : Writeln (Horrid, Root-.word, 'h':1);
    end;
    case Root-.vigor of
      Violent : Writeln (Violent, Root-.word, 'v':1);
      Energetic : Writeln (Energetic, Root-.word, 'e');
      Calm : Writeln (Calm, Root-.word, 'c');
      Inert : Writeln (Inert, Root-.word, 'i');
    end;
  end;
  Inorder (Root-.right);
end;
begin (main program)
Init_Researcher;
Rewrite ([Latinat, 'Latinat text a']);
Rewrite ([Germano, 'Germano text a']);
Rewrite ([Tangble, 'Tangble text a']);
Rewrite ([Intngble, 'Intngble text a']);
Rewrite ([Postgrd, 'Postgrd text a']);
Rewrite ([Graduate, 'Graduate text a']);
Rewrite ([HighSchl, 'HighSchl text a']);
Rewrite ([Elementa, 'Elementa text a']);
Rewrite ([Sublime, 'Sublime text a']);
Rewrite ([Pleasnt, 'Pleasnt text a']);
Rewrite ([Nuetrl, 'Nuetrl text a']);
Rewrite ([Unpleasn, 'Unpleasn text a']);
Rewrite ([Horrd, 'Horrd text a']);
Rewrite ([Violnt, 'Violnt text a']);
Rewrite ([Energeti, 'Energeti text a']);
Rewrite ([Cim, 'Cim text a']);
Rewrite ([Inrt, 'Inrt text a']);
Inorder (Root);
end.
program Dixjoin (input, output);
const
maxsent = 1000;
wordlength = 15;
linelength = 60;
phraselength = 4;
freqnum = 50;
seilnum = 200;

type
Genre type = (Nonfiction, Fiction);
Frequency type = (Frequently, Seldomly, When_Done);
Balance type = (Plus, Zero, Minus);
Duration type = (Permanent, Temporary);
Source type = (Latinate, Germanic);
Concreteness type = (Tangible, Intangible);
Emotion type = (Sublime, Pleasant, Neutral, Unpleasant, Horrid);
Vigor type = (Violent, Energetic, Calm, Inert);

Word type = packed array (.i..wordlength.) of char;
Linetype = packed array (.i..linelength.) of char;
Phrasetype = packed array (.i..phraselength.) of wordtype;

EntryPointer = ~Entrytype;
ConcordPointer = ~Concordtype;

Entrytype = record
  Word : Wordtype;
  Balance : Balancetype;
  Duration : Duratiotype;
  Source : Sourcetype;
  Difficulty : Difficulty type;
  Concreteness : Concratenesstype;
  Emotion : Emotiontype;
  Vigor : Vigortype;
  Left, Right : EntryPointer;
  Down : Concordpointer;
end;

Concordtype = record
  Up : EntryPointer;
  Down;
  Next, Last : Concordpointer;
end;
var
    FourRoot,
    Root : EntryPointer;
    Lastward : ConcordPointer;

    Latinat,
    Germaine,
    Tangble,
    Intngble,
    Postcard,
    Graduate,
    HighSchl,
    Elementa,
    Sublma,
    Pleasnt,
    Nustrl,
    Unpleaszn,
    Horrd,
    Violnt,
    Energeti,
    Clm,
    Inrt,
    Dictionary : text;
    word : wordtype;
    p : entrypointer;
    c : char;

procedure Attach (Entry : entrytype; var p : entrypointer);
begin
    New (p);
    with p do begin
        word := entry.word;
        duration := permanent;
        source := entry.source;
        difficulty := entry.difficulty;
        concreteness := entry.concreteness;
        emotion := entryemotion;
        vigor := entry.vigor;
        left := nil; right := nil;
        down := nil;
    end;
end;
procedure Encode (its : entrytype; var xsource : char;  
    var xdifficulty : char;  
    var xconcreteness : char; var xemotion : char;  
    var xvigor : char);  
begin  
    if (its.source = Latinate) then xsource := 'l'  
        else xsource := 'g'  
    case its.difficulty of  
        Postgrad : xdifficulty := 'p';  
        Grad : xdifficulty := 'g';  
        High_School : xdifficulty := 'h';  
        Elementary : xdifficulty := 'e';  
    end;  
    if (its.concreteness = Tangible) then Xconcreteness := 't'  
          else Xconcreteness := 'i';  
    case its.emotion of  
        Sublime : xemotion := 's';  
        Pleasant : xemotion := 'p';  
        Neutral : xemotion := 'n';  
        Unpleasant : xemotion := 'u';  
        Horrid : xemotion := 'h';  
    end;  
    case its.vigor of  
        Violent : xvigor := 'v';  
        Energetic : xvigor := 'e';  
        Calm : xvigor := 'c';  
        Inert : xvigor := 'i';  
    end;  
end;  

procedure Decode (var its : entrytype;  
    xsource : char; xdifficulty : char;  
    xconcreteness : char; xemotion : char;  
    xvigor : char);  
begin  
    if (Xsource = 'l') then its.source := Latinate  
          else its.source := Germanic;  
    case Xdifficulty of  
        'p' : its.difficulty := Postgrad;  
        'g' : its.difficulty := Grad;  
        'h' : its.difficulty := High_School;  
        'e' : its.difficulty := Elementary;  
    end;  
    if (Xconcreteness = 't') then its.concreteness := Tangible  
          else its.concreteness := Intangible;  
    case Xemotion of  
        's' : its.emotion := Sublime;  
        'p' : its.emotion := Pleasant;  
        'n' : its.emotion := Neutral;  
        'u' : its.emotion := Unpleasant;  
        'h' : its.emotion := Horrid;  
    end;  
    case XVigor of  
        'v' : its.vigor := Violent;  
        'e' : its.vigor := Energetic;  
        'c' : its.vigor := Calm;  
        'i' : its.vigor := Inert;  
    end;  
end;  

procedure AVL_Insert (Entry : entrytype; var p : Entrypointer; 
var balanced : boolean);
var
p1, p2 : Entrypointer;
begin
if (p = nil) then begin
  Attach (Entry, p);
  p-.balance := Zero;
  balanced := true;
end
else if (Entry.word = p-.word) then Writeln ('INSERT COLLISION');
else if (Entry.word < p-.word) then begin
  AVL_Insert (Entry, p-.left, balanced);
  if Balanced then begin
    case p-.balance of
      Plus : begin p-.balance := Zero; balanced := false; end;
      Zero : p-.balance := Minus;
      Minus : begin (* rebalance *)
        pl := p-.left;
        if (pl-.balance = Minus) then begin (* single LL rotation *)
          p-.left := pl-.right;
          pl-.right := p;
          p := pl;
        end (* if *)
        else begin (* double LR rotation *)
          p2 := pl-.right;
          pl-.right := p2-.left;
          p2-.left := pl;
          p-.left := p2-.right;
          p2-.right := p;
          if (p2-.balance = Minus) then p-.balance := Plus
          else p-.balance := Zero;
          if (p2-.balance = Plus) then pl-.balance := Minus
          else pl-.balance := Zero;
          p := p2;
        end (* else *)
        p-.balance := Zero; balanced := false;
      end (* case of Minus *)
  end (* if Balanced *)
end (* of cases *)
end (* if Entry.word < p-.word *)
else if (Entry.word > p-.word) then begin
    AVL_Insert (Entry, p-.right, balanced);
    if balanced then (* right pointer has grown higher *)
        case p-.balance of
            Minus : begin p-.balance := Zero;
                balanced := false; end;
            Zero : p-.balance := Plus;
            Plus : begin (* rebalance *)
                pl := p-.right;
                if (pl-.balance = Plus) then begin
                    (* single RR *)
                    p-.right := pl-.left;
                    pl-.left := p;
                    p-.balance := Zero;
                    p := pl;
                end (* if *)
                else begin (* double RL rotation *)
                    p2 := pl-.left;
                    pl-.left := p2-.right;
                    p2-.right := pl;
                    pl-.right := p2-.left;
                    if (p2-.balance = Plus) then
                        p-.balance := Minus
                    else p-.balance := Zero;
                    if (p2-.balance = Minus) then
                        pl-.balance := Plus
                    else pl-.balance := Zero;
                    p := p2;
                end (* double RL rotation *)
                end (* case of balance = Plus *)
            end (* case of cases *)
        end (* of if Entry.word > p-.word *)
    else balanced := false;
    end (* of procedure AVL_Insert *)
procedure Init_Researcher;
var
i,
stepcount,
threshold,
dixlength : integer;
s, d, c, e, v : char;
entry : entrytype;
balanced : boolean;
begin
Reset (Dixonary, 'Dixonary text a');
New (Root);
Readin (Dixonary, Root-.word, s, d, c, e, v);
Decode (Root-.word, s, d, c, e, v);
Root-.left := nil; Root-.right := nil;
Root-.down := nil;
Root-.balance := Zero;
Root-.duration := Permanent;
Lastword := nil;
Stepcount := 0;
while not EOF (Dixonary) do begin
Readin (Dixonary, Entry-.word, s, d, c, e, v);
Decode (Entry-.word, s, d, c, e, v);
Entry-.balance := Zero; balanced := false;
AVL_Insert (Entry, Root, balanced);
stepcount := stepcount + 1;
if (stepcount mod 200 = 0) then begin
Page; Writeln; Writeln; Writeln; Writeln;
Writeln ('Stylist is now loading its dictionary.');
Writeln ('Last word loaded was '); Writeln;
Writeln ('The list has loaded ', stepcount, ' entries.');
Writeln; Writeln;
end; (* if *)
end; (* while not EOF *)
end;

procedure Inorder (var Root : entrypointer;
word : wordtype;
var P : entrypointer);
begin
if (Root-.word = word) then P := root
else if (Root-.word > word) then Inorder (Root-.left, word, P)
else Inorder (Root-.right, word, P);
end;

procedure Store_BST (p : entrypointer);
var s, d, c, e, v : char;
begin
if (p <> nil) then begin
Encode (p-.word, s, d, c, e, v);
Writeln (Dixonary, p-.word, s, d, c, e, v);
Store_BST (p-.left);
Store_BST (p-.right);
end;
end;

procedure Store_New_Dix;
begin
Rewrite (Dixonary, 'Dixonary text a');
Store_BST (Root);
end;
begin (X main program X)
Init_Researcher;
Reset (Latinat, 'Latinat text a');
While not EOF (Latinat) do begin
  Readln (Latinat, word, c);
  if (c <> 'l') then begin Inorder (root, word, p);
    p-.source := germanic; end;
end;
Reset (Germano, 'Germano text a');
While not EOF (Germano) do begin
  Readln (Germano, word, c);
  if (c <> 'g') then begin Inorder (root, word, p);
    p-.source := latinate; end;
end;
Reset (Tangble, 'Tangble text a');
While not EOF (Tangble) do begin
  Readln (Tangble, word, c);
  if (c <> 't') then begin Inorder (root, word, p);
    p-.concreteness := intangible; end;
end;
Reset (Intngble, 'Intngble text a');
While not EOF (Intngble) do begin
  Readln (Intngble, word, c);
  if (c <> 'i') then begin Inorder (root, word, p);
    p-.concreteness := tangible; end;
end;
Reset (Postgrd, 'Postgrd text a');
While not EOF (Postgrd) do begin
  Readln (Postgrd, word, c);
  if (c <> 'p') then begin Inorder (root, word, p);
    if (c = 'g') then p-.difficulty := grad;
    if (c = 'h') then p-.difficulty := high_school;
    if (c = 'a') then p-.difficulty := elementary; end;
end;
Reset (Graduate, 'Graduate text a');
While not EOF (Graduate) do begin
  Readln (Graduate, word, c);
  if (c <> 'g') then begin Inorder (root, word, p);
    if (c = 'p') then p-.difficulty := postgrad;
    if (c = 'h') then p-.difficulty := grad;
    if (c = 'e') then p-.difficulty := elementary; end;
end;
Reset (HighSchl, 'HighSchl text a');
While not EOF (HighSchl) do begin
  Readln (HighSchl, word, c);
  if (c <> 'h') then begin Inorder (root, word, p);
    if (c = 'p') then p-.difficulty := postgrad;
    if (c = 'g') then p-.difficulty := grad;
    if (c = 'e') then p-.difficulty := elementary; end;
end;
Reset (Elementa, 'Elementa text a');
While not EOF (Elementa) do begin
  Readln (Elementa, word, c);
  if (c <> 'e') then begin Inorder (root, word, p);
    if (c = 'p') then p-.difficulty := postgrad;
    if (c = 'g') then p-.difficulty := grad;
    if (c = 'h') then p-.difficulty := high_school; end;
end;
Reset (Sublime, 'Sublime text a');
While not EOF (Sublime) do begin
  Readin (Sublime, word, c);
  if (c <> 's') then begin Inorder (root, word, p);
    if (c = 'p') then p.emotion := pleasant;
    if (c = 'n') then p.emotion := neutral;
    if (c = 'u') then p.emotion := unpleasant;
    if (c = 'h') then p.emotion := horrid; end;
end;

Reset (Pleasant, 'Pleasant text a');
While not EOF (Pleasant) do begin
  Readin (Pleasant, word, c);
  if (c <> 'n') then begin Inorder (root, word, p);
    if (c = 's') then p.emotion := sublime;
    if (c = 'p') then p.emotion := pleasant;
    if (c = 'n') then p.emotion := neutral;
    if (c = 'u') then p.emotion := unpleasant; end;
end;

Reset (Neutral, 'Neutral text a');
While not EOF (Neutral) do begin
  Readin (Neutral, word, c);
  if (c <> 'n') then begin Inorder (root, word, p);
    if (c = 's') then p.emotion := sublime;
    if (c = 'p') then p.emotion := pleasant;
    if (c = 'n') then p.emotion := neutral;
    if (c = 'u') then p.emotion := unpleasant; end;
end;

Reset (Unpleasant, 'Unpleasant text a');
While not EOF (Unpleasant) do begin
  Readin (Unpleasant, word, c);
  if (c <> 'n') then begin Inorder (root, word, p);
    if (c = 's') then p.emotion := sublime;
    if (c = 'p') then p.emotion := pleasant;
    if (c = 'n') then p.emotion := neutral;
    if (c = 'u') then p.emotion := unpleasant; end;
end;

Reset (Horrid, 'Horrid text a');
While not EOF (Horrid) do begin
  Readin (Horrid, word, c);
  if (c <> 'h') then begin Inorder (root, word, p);
    if (c = 's') then p.emotion := sublime;
    if (c = 'p') then p.emotion := pleasant;
    if (c = 'n') then p.emotion := neutral;
    if (c = 'u') then p.emotion := unpleasant; end;
end;

Reset (Violent, 'Violent text a');
While not EOF (Violent) do begin
  Readin (Violent, word, c);
  if (c <> 'v') then begin Inorder (root, word, p);
    if (c = 's') then p.vigor := energetic;
    if (c = 'p') then p.vigor := calm;
    if (c = 'n') then p.vigor := inert; end;
end;

Reset (Energetic, 'Energetic text a');
While not EOF (Energetic) do begin
  Readin (Energetic, word, c);
  if (c <> 'e') then begin Inorder (root, word, p);
    if (c = 's') then p.vigor := violent;
    if (c = 'p') then p.vigor := calm;
    if (c = 'n') then p.vigor := inert; end;
end;
Reset (Clm, 'Clm text a');
While not EOF (Clm) do begin
  Readln (Clm, word, c);
  if (c <> 'c') then begin Inorder (root, word, p);
    if (c = 'v') then p-.vigor := violent;
    if (c = 'e') then p-.vigor := energetic;
    if (c = 'i') then p-.vigor := inert; end;
end;

Reset (Inrt, 'Inrt text a');
While not EOF (Inrt) do begin
  Readln (Inrt, word, c);
  if (c <> 'c') then begin Inorder (root, word, p);
    if (c = 'v') then p-.vigor := violent;
    if (c = 'e') then p-.vigor := energetic;
    if (c = 'i') then p-.vigor := calm; end;
end;

Store_New_Dix;
end.

end.
APPENDIX E
REPRESENTATIVE RUNS

PROFILE

Plato's Phaedrus

<table>
<thead>
<tr>
<th>Total of sentences</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total of words</td>
<td>360</td>
</tr>
<tr>
<td>Total of letters</td>
<td>1595</td>
</tr>
</tbody>
</table>

360 words are too few for valid statistical analysis.

Of the 360 words in Plato's Phaedrus, 312 were matched to words in the Stylist dictionary. 86% were matched. This is enough for statistical analysis.
BREAKDOWN OF PERCENT OF LETTERS PER WORD

The average number of letters per word : 6.131428571E+00

A typical nonfiction texts distribution resembles a low bell-shaped curve centered around six letters/word.

The length of the words is medium
**BREAKDOWN OF NUMBER OF WORDS PER SENTENCE**

| Number of words | 100 | 98 | 96 | 94 | 92 | 90 | 88 | 86 | 84 | 82 | 80 | 78 | 76 | 74 | 72 | 70 | 68 | 66 | 64 | 62 | 60 | 58 | 56 | 54 | 52 | 50 | 48 | 46 | 44 | 42 | 40 | 38 | 36 | 34 | 32 | 30 | 28 | 26 | 24 | 22 | 20 | 18 | 16 | 14 | 12 | 10 | 8  | 6  | 4  | 2  |
|-----------------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Sentence Number | 10  | 20 | 30 | 40 | 50 | 60 | 70 |

The average number of words per sentence: \(2.117647059 \times 10^1\)

A typical modern text's sentences average between fifteen and twenty words.

Sentences are long.

Number of run ons: 0

Percent of run ons: 0

Run ons are nonexistent.
Number of medium length sentences: 9
Percent of medium length sentences: 52

Modulation is good.

**ETYMOLOGY OF WORDS**

<table>
<thead>
<tr>
<th>Latinate words</th>
<th>51</th>
</tr>
</thead>
<tbody>
<tr>
<td>Germanic words</td>
<td>95</td>
</tr>
</tbody>
</table>

Etymology is very native.

**DIFFICULTY OF VOCABULARY**

<table>
<thead>
<tr>
<th>Postgraduate difficulty</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graduate difficulty</td>
<td>0</td>
</tr>
<tr>
<td>High School difficulty</td>
<td>1</td>
</tr>
<tr>
<td>Elementary difficulty</td>
<td>120</td>
</tr>
</tbody>
</table>

Percent of Postgraduate difficulty: 0
Percent of Graduate difficulty: 0
Percent of High School difficulty: 8

Difficulty is easy.

**TANGIBILITY**

<table>
<thead>
<tr>
<th>Tangible words</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intangible words</td>
<td>111</td>
</tr>
</tbody>
</table>

Tangibility is tangible.

**EMOTIONAL CONNOTATIONS**

<table>
<thead>
<tr>
<th>Connotations</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sublime</td>
<td>7</td>
</tr>
<tr>
<td>Pleasant</td>
<td>44</td>
</tr>
<tr>
<td>Neutral</td>
<td>87</td>
</tr>
<tr>
<td>Unpleasant</td>
<td>8</td>
</tr>
<tr>
<td>Horrid</td>
<td>0</td>
</tr>
</tbody>
</table>

Percent of sublime connotations: 2
Percent of pleasant connotations: 14
Percent of unpleasant connotations: 2
Percent of horrid connotations: 0

Index of Emotionality: 42

Emotionality is Rich
Tone is Positive
VIGOR OF WORDS

<table>
<thead>
<tr>
<th>Type of Vigor</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Words of Extreme Vigor</td>
<td>0</td>
</tr>
<tr>
<td>Words of Much Vigor</td>
<td>17</td>
</tr>
<tr>
<td>Words of Some Vigor</td>
<td>75</td>
</tr>
<tr>
<td>Words of Little Vigor</td>
<td>54</td>
</tr>
</tbody>
</table>

Percent of words of extreme vigor: 0
Percent of words of much vigor: 5
Percent of words of some vigor: 24

Index of Vigor: 49

Vigor is lively

Congratulations! You never seem to write run on sentences. This is a solid piece of writing well within the traditions of its genre.
Phaedrus

them and thamus inquired about their several uses and discovered is not an aid to memory but to other egyptians might be allowed to have the benefit of an art is not always the best judge of is called by them ammon to him came theuth learned nothing they will appear to be omniscient and censured others as he approved or disapproved of them many arts such as arithmetic and calculation and geometry or inventor of an art is not always the art in praise or blame of the various arts but when they came calculation and geometry and astronomy and draughts and dice have been led to attribute to them a quality in the learners' souls because they will not use your own children have been led to attribute to allowed to have the benefit of them he enumerated is not always the best judge of the utility wiser and give them better memories it is a name was theuth the bird which is called the ibis sacred the god himself is called by them ammon to them ammon to him came theuth and showed his arts but what theuth said is a quality which they cannot have for this discovery some of them and censured others as he approved to the external written characters and not remember of love of your own children have been led to socrates at the egyptian city of naucratis there was dwelt in that great city of upper egypt which they will be tiresome company having the show of king of the whole country of egypt and he discovery of yours will create forgetfulness in the learners' letters now is those days the god thamus was and showed that the other egyptians astronomy and draughts and dice but his great discovery as he approved or disapproved of them it would and you give your disciples not truth but only specific which you have discovered is not an aid dice but his great discovery was the use of cannot have for this discovery of yours will create geometry and astronomy and draughts and dice but his of egypt and he dwelt in that great city the whole country of egypt and he dwelt in great city of upper egypt which the hellenes call egyptian thebes and the god desiring that the other egyptians might be allowed to theuth will make the egyptians wiser and give them enumerated them and thamus enquired about their several uses benefit of them he enumerated them and thamus enquired will trust to the external written characters and not naucratis there was a famous old god whose name you who are the father of letters from the of yours will create forgetfulness in the learners' souls be omniscient and will generally know nothing they will arithmetic and calculation and geometry and astronomy and draughts the egyptians wiser and give them better memories it to reminiscence and you give your disciples not truth
was a famous old GOD whose name was theuth
as theuth was the GOD thamus was the king
egyptian thebes and the GOD himself is called by
and dice but his GREAT discovery was the use
he dwelt in that GREAT city of upper egypt
will be tiresome company HAVING the show of wisdom
truth they will be HEARERS of many things and
upper egypt which the HELLENES call egyptian thebes and
themselves and the god is called by them
which is called the IBIS is sacred to him
thamus replied o most INGENIOUS theuth the parent or
them and in this INSTANCE you who are the
of the utility or INUTILITY of his own inventions
thath and showed his INVENTIONS desiring that the other
integrity of his own INVENTIONS to the users of
and he was the INVENTOR of many arts such
theuth the parent or INVENTOR of an art is not
always the best JUDGE of the utility or
and thamus was the KING of the whole country
omniscient and will generally KNOW nothing they will be
things and will have LEARNED nothing they will appear
create forgetfulness in the LEARNERS' souls because they will
own children have been LED to attribute to them
was the use of LETTERS; now in those days
when they came to LETTERS this said thauth will
are the father of LETTERS from the paternal love
it would take a LONG time to repeat all
letters from the paternal LOVE of your own children
this said theuth will MAKE the egyptians wiser and
was the inventor of MANY arts such as arithmetic
will be hearers of MANY things and will have
and give them better MEMORIES it is a specific
will not use their MEMORIES they will trust to
specific both for the MEMORY and for the wit
not an aid to MEMORY but to reminiscence and
that the other egyptians MIGHT
thamus replied o MOST ingenious theuth the parent
famous old god whose NAME was theuth the bird
the egyptian city of NAUCRATIS there was a famous
and will have learned NOTHING they will appear to
and will generally know NOTHING they will be tiresome
thamus replied o most ingenious theuth the
there was a famous OLD god whose name was
will appear to be OMNISCIENT and will generally know
inventions desiring that the OTHER egyptians might be allowed
of them and censured OTHERS as he approved or
or inutility of his OWN inventions to the users
paternal love of your OWN children have been led
most ingenious theuth the PARENT or inventor of an
of letters from the PATERNAL love of your own
said to theuth in PRAISE or blame of the
their several uses and praised some of them and
attribute to them a QUALITY which they cannot have
of wisdom without the REALITY
written characters and not REMEMBER of themselves the specific
to memory but to REMINISCENCE and you give your
a long time to REPEAT all that thamus said
for the wit thamus REPLIED o most ingenious theuth
called the ibis is SACRED to him and he
repeat all that thamus SAID to theuth in praise
came to letters this SAID thauth will make the
truth but only the SEMBLANCE of truth they will
thamus enquired about their SEVERAL uses and praised some
times having the SHOW of wisdom without the
him came thauth and SHOWED his inventions desiring that
SOCRATES at the egyptian city
several uses and praised SOME of them and censured forgetfulness in the learners' SOULS because they will not remember of themselves the SPECIFIC which you have discovered of them it would TAKE a long time to those days the god THAMUS was the king of he enumerated them and THAMUS enquired about their several to remember all that THAMUS said to theuth in and for the wit THAMUS replied o most ingenious the hellenes call egyptian THEBES and the god himself and thamus enquired about THEIR several uses and praised they will not use THEIR memories they will trust and not remember of THEMSELVES the specific which you egyptian city of naucratis THERE was a famous old god whose name was THEUTH the bird which is ammon to him came THEUTH and showed his inventions that thamus said to THEUTH in praise or blame to letters this said THEUTH will make the egyptians replied o most ingenious THEUTH the parent or inventor be hearers of many THINGS and will have learned of letters now is THOSE days the god thamus would take a long TIME to repeat all that nothing they will be TIRESOME company having the show their memories they will TRUST to the external written give your disciples not TRUTH but only the semblance of TRUTH they will be hearers of many THINGS and will have learned of letters now is THOSE days the god thamus great discovery was the USE of letters now is because they will not USE their memories they will own inventions to the USERS of them and in enjoined about their several USES and praised some of best judge of the UTILITY or inutility of him or blame of the VARIOUS arts but when they the various arts but WHEN they came to letters was theuth the bird WHICH is called the ibis city of upper egypt WHICH the hellenes call egyptian to them a quality WHICH they cannot have for of themselves the specific WHICH you have discovered is the king of the WHOLE country of egypt and a famous old god WHOSE name was theuth the having the show of WISDOM without the reality will make the egyptians WISER and give them better memory and for the WIT thamus replied o most the show of wisdom WITHOUT the reality disapproved of them it WOULD take a long time trust to the external WRITTEN characters and not remember for this discovery of YOURS will create forgetfulness in
Student Paper. Example of Poor Writing

PROFILE

Total of sentences : 62
Total of words : 2026
Total of letters : 10561

2026 words are plenty for valid statistical analysis.

Of the 2026 words in Student Paper, Example of Poor Writing, 1839 were matched to words in the Stylist dictionary.

90% were matched. This is enough for statistical analysis.
BREAKDOWN OF PERCENT OF LETTERS PER WORD

<table>
<thead>
<tr>
<th>Number of Letters in a Word</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The average number of letters per word: \(6.92188713E+00\)

A typical nonfiction text distribution resembles a low bell-shaped curve centered around six letters/word.

The length of the words is long.
The average number of words per sentence is 3.267741935E+01.

A typical modern texts sentences average between fifteen and twenty words. Sentences are too long.

Number of run ons: 13
Percent of run ons: 20

Run ons are unacceptable.
Number of medium length sentences: 14
Percent of medium length sentences: 22

Modulation is average

**ETYMOLOGY OF WORDS**

<table>
<thead>
<tr>
<th>Latinate words</th>
<th>708</th>
</tr>
</thead>
<tbody>
<tr>
<td>Germanic words</td>
<td>402</td>
</tr>
</tbody>
</table>

Etymology is mixed

**DIFFICULTY OF VOCABULARY**

<table>
<thead>
<tr>
<th>Difficulty</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Postgraduate</td>
<td>26</td>
</tr>
<tr>
<td>Graduate</td>
<td>141</td>
</tr>
<tr>
<td>High School</td>
<td>405</td>
</tr>
<tr>
<td>Elementary</td>
<td>538</td>
</tr>
</tbody>
</table>

Difficulty is very hard

**TANGIBILITY**

<table>
<thead>
<tr>
<th>Tangible words</th>
<th>1343</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intangible words</td>
<td>767</td>
</tr>
</tbody>
</table>

Tangibility is tangible.

**EMOTIONAL CONNOTATIONS**

<table>
<thead>
<tr>
<th>Connotation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sublime</td>
<td>7</td>
</tr>
<tr>
<td>Pleasant</td>
<td>166</td>
</tr>
<tr>
<td>Neutral</td>
<td>861</td>
</tr>
<tr>
<td>Unpleasant</td>
<td>75</td>
</tr>
<tr>
<td>Horrid</td>
<td>1</td>
</tr>
</tbody>
</table>

Percent of sublime connotations: 0%
Percent of pleasant connotations: 9%
Percent of unpleasant connotations: 4%
Percent of horrid connotations: 0%

Index of Emotionality: 26

Emotionality is Rich

Tone is Positive
VIGOR OF WORDS

Words of Extreme Vigor : 1
Words of Much Vigor : 94
Words of Some Vigor : 464
Words of Little Vigor : 551

Percent of words of extreme vigor : 0
Percent of words of much vigor : 5
Percent of words of some vigor : 25

Index of Vigor 50
Vigor is lively

RECOMMENDATION NUMBER 1

You tend to write run-on sentences.
Check your longest sentences for run-ons.
Break them up into units of single ideas.

RECOMMENDATION NUMBER 2

Your average sentences are too long for the difficulty of your vocabulary.
Use simpler words or shorter sentences.
Excerpt from A Farewell to Arms  PROFILE

Total of sentences : 65
Total of words : 1349
Total of letters : 15431

1349 words are plenty for valid statistical analysis.

Of the 1349 words in Excerpt from A Farewell to Arms, 1197 were matched to words in the Stylist dictionary.

88 % were matched. This is enough for statistical analysis.
### Breakdown of Percent of Letters per Word

<table>
<thead>
<tr>
<th>Number of Letters in a Word</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The average number of letters per word: \( 5.100817439 \times 10^0 \)

A typical fiction texts distribution resembles a tall bell-shaped curve centered around five letters/word.

The length of the words is medium.
A typical modern text sentences average between fifteen and twenty words. Sentences are long.

Number of run ons 3
Percent of run ons 4

Run ons are acceptable.
Number of medium length sentences  26
Percent of medium length sentences  40

Modulation is average

ETYMOLOGY OF WORDS

Number of Latinate words  137
Number of Germanic words  484

Etymology is very native.

DIFFICULTY OF VOCABULARY

PostGraduate difficulty  0
Graduate difficulty  0
High School difficulty  25
Elementary difficulty  596

Percent of Postgraduate difficulty  0
Percent of Graduate difficulty  0
Percent of High School difficulty  2

Difficulty is easy.

TANGIBILITY

Number of Tangible words  179
Number of Intangible words  442

Tangibility is tangible.

EMOTIONAL CONNOTATIONS

Sublime connotations  1
Pleasant connotations  135
Neutral connotations  450
Unpleasant connotations  34
Horrid connotations  1

Percent of sublime connotations  0
Percent of pleasant connotations  11
Percent of unpleasant connotations  2
Percent of horrid connotations  0

Index of Emotionality  26

Emotionality is Rich
Tone is Positive
VIGOR OF WORDS

<table>
<thead>
<tr>
<th>Type of Vigor</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Words of Extreme Vigor</td>
<td>4</td>
</tr>
<tr>
<td>Words of Much Vigor</td>
<td>73</td>
</tr>
<tr>
<td>Words of Some Vigor</td>
<td>324</td>
</tr>
<tr>
<td>Words of Little Vigor</td>
<td>220</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type of Vigor</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Words of extreme vigor</td>
<td>0</td>
</tr>
<tr>
<td>Words of much vigor</td>
<td>6</td>
</tr>
<tr>
<td>Words of some vigor</td>
<td>27</td>
</tr>
</tbody>
</table>

Index of Vigor: 57

Vigor is strong

This is a solid piece of writing well within the traditions of its genre.
You are as able to understand the meaning of the above characteristics as The Stylist!
Stand the course!
Total of sentences : 56
Total of words : 1178
Total of letters : 5868

1178 words are plenty for valid statistical analysis.

Of the 1178 words in Computer Science Text 1, 1026 were matched to words in the Stylist dictionary. 87% were matched. This is enough for statistical analysis.
### Breakdown of Percent of Letters Per Word

<table>
<thead>
<tr>
<th>Number of Letters in a Word</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The average number of letters per word: 6.500000000E+00

A typical nonfiction texts distribution resembles a low bell-shaped curve centered around six letters/word.

The length of the words is medium.
The average number of words per sentence: 2.103571428E+01

A typical modern text's sentences average between fifteen and twenty words. Sentences are long.

Number of run ons: 1
Percent of run ons: 1

Run ons are acceptable.
Number of medium length sentences 28
Percent of medium length sentences 50

Modulation is average

**ETYMOLOGY OF WORDS**

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latinate words</td>
<td>422</td>
</tr>
<tr>
<td>Germanic words</td>
<td>227</td>
</tr>
</tbody>
</table>

Etymology is mixed

**DIFFICULTY OF VOCABULARY**

<table>
<thead>
<tr>
<th>Difficulty Level</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostGraduate</td>
<td>1</td>
</tr>
<tr>
<td>Graduate</td>
<td>55</td>
</tr>
<tr>
<td>High School</td>
<td>273</td>
</tr>
<tr>
<td>Elementary</td>
<td>320</td>
</tr>
</tbody>
</table>

Percent of Postgraduate difficulty 0
Percent of Graduate difficulty 5
Percent of High School difficulty 26

Difficulty is challenging

**TANGIBILITY**

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tangible words</td>
<td>106</td>
</tr>
<tr>
<td>Intangible words</td>
<td>543</td>
</tr>
</tbody>
</table>

Tangibility is very intangible.

**EMOTIONAL CONNOTATIONS**

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sublime connotations</td>
<td>1</td>
</tr>
<tr>
<td>Pleasant connotations</td>
<td>50</td>
</tr>
<tr>
<td>Neutral connotations</td>
<td>562</td>
</tr>
<tr>
<td>Unpleasant connotations</td>
<td>32</td>
</tr>
<tr>
<td>Horrid connotations</td>
<td>4</td>
</tr>
</tbody>
</table>

Percent of sublime connotations 0
Percent of pleasant connotations 4
Percent of unpleasant connotations 3
Percent of horrid connotations 0

Index of Emotionality 14

Emotionality is average
Tone is Positive
VIGOR OF WORDS

| Words of Extreme Vigor | 7 |
| Words of Much Vigor    | 34 |
| Words of Some Vigor    | 301 |
| Words of Little Vigor  | 307 |

Percent of words of extreme vigor : 0
Percent of words of much vigor : 3
Percent of words of some vigor : 29

Index of Vigor : 44

Vigor is lively

This is a solid piece of writing well within the traditions of its genre.
You are as able to understand the meaning of the above characteristics as The Stylist.
Stand the course!
542 words are enough for valid statistical analysis.

Of the 542 words in Excerpt from *Galapagos* by Vonnegut, 449 were matched to words in the Stylist dictionary. 82% were matched. This is enough for statistical analysis.
### Breakdown of Percent of Letters per Word

<table>
<thead>
<tr>
<th>Number of Letters in a Word</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

The average number of letters per word: \( 5.7602339189E+00 \)

A typical fiction texts distribution resembles a tall bell-shaped curve centered around five letters/word.

The length of the words is long.
The average number of words per sentence: \(2.463636364E+01\)

A typical modern text's sentences average between fifteen and twenty words. Sentences are too long.

Number of run ons: 2
Percent of run ons: 9

Run ons are unacceptable
Number of medium length sentences: 10
Percent of medium length sentences: 45

Modulation is average.

ETYMOLOGY OF WORDS

Number of Latinate words: 86
Number of Germanic words: 184

Etymology is very native.

DIFFICULTY OF VOCABULARY

PostGraduate difficulty: 0
Graduate difficulty: 3
High School difficulty: 49
Elementary difficulty: 218

Percent of Postgraduate difficulty: 0
Percent of Graduate difficulty: 0
Percent of High School difficulty: 10

Difficulty is easy.

TANGIBILITY

Number of Tangible words: 76
Number of Intangible words: 194

Tangibility is tangible.
EMOTIONAL CONNOTATIONS

Sublime connotations : 2  
Pleasant connotations : 79  
Neutral connotations : 170  
Unpleasant connotations : 15  
Horrid connotations : 6

Percent of sublime connotations 0  
Percent of pleasant connotations 17  
Percent of unpleasant connotations 2  
Percent of horrid connotations 1

Index of Emotionality 43  
Emotionality is Rich  
Tone is Positive

VIGOR OF WORDS

Words of Extreme Vigor : 3  
Words of Much Vigor : 39  
Words of Some Vigor : 123  
Words of Little Vigor : 105

Percent of words of extreme vigor : 0  
Percent of words of much vigor : 8  
Percent of words of some vigor : 27

Index of Vigor 67  
Vigor is very strong

RECOMMENDATION NUMBER 1

You tend to write run-on sentences.  
Check your longest sentences for run-ons.  
Break them up into units of single ideas.
LIST OF REFERENCES


BIBLIOGRAPHY


Bruno, Agnes M., Toward a Quantitative Methodology for Stylistic Analysis, University of California Publ. in Modern Philology Vol. 109, Bks Demand UMI.


Grishman, Ralph, Computational Linguistics, Cambridge University Press.


### INITIAL DISTRIBUTION LIST

<table>
<thead>
<tr>
<th>No.</th>
<th>Copies</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>Defense Technical Information Center&lt;br&gt;Cameron Station&lt;br&gt;Alexandria, VA 22304-6145</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Library, Code 0142&lt;br&gt;Naval Postgraduate School&lt;br&gt;Monterey, CA 93943-5002</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>Associate Professor T. Wu&lt;br&gt;Computer Science Department, Code 52&lt;br&gt;Naval Postgraduate School&lt;br&gt;Monterey, CA 93943</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>Associate Professor B. MacLennan&lt;br&gt;Computer Science Department, Code 52&lt;br&gt;Naval Postgraduate School&lt;br&gt;Monterey, CA 93943</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>Director, Writing Program&lt;br&gt;English Department&lt;br&gt;Burrowes Building&lt;br&gt;The Pennsylvania State University&lt;br&gt;University Park, PA 16802</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>ButtonWare Inc.&lt;br&gt;P.O. Box 5786&lt;br&gt;Bellevue, WA 98006&lt;br&gt;Attn : PC-Style Project Manager</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>Bell Laboratories&lt;br&gt;Murray Hill&lt;br&gt;New Jersy 07974&lt;br&gt;Attn : Writer's Workbench Project Manager</td>
</tr>
</tbody>
</table>