THE INFLUENCE OF COLOR ON PROGRAM READABILITY AND COMPREHENSIBILITY

Gerard K. Rambally
Department of Computer Science
University of Regina
Regina, Saskatchewan
Canada S4S 0A2

Abstract

Readability and comprehensibility are among the most important attributes of a program. A program that is easy to read and understand is easier to test, maintain, and modify. Many factors affect program readability and comprehensibility, including variable names, internal documentation, modularity, and so on. This paper investigates the influence of color on program readability and comprehension. Three color schemes were used: Color-scheme-A used different colors to indicate the different blocks in a program; Color-scheme-B used different colors to identify the various statements function in the program; and the third color scheme was the usual black-and-white programs. This study showed that subjects who used programs with Color-scheme-B had the highest mean score for program comprehension, followed by those who used Color-scheme-A. Subjects who used black-and-white programs scored the lowest on the comprehension quiz.

1. Introduction

Readability and comprehensibility (i.e. the degree of ease with which a programmer can read and comprehend a program) are among the most important attributes of a program. A program that is easy to read and understand is easier to test, maintain, and modify. Many aspects of programming style affect readability and comprehensibility, including variable names, internal documentation, modularity, and formatting. It is the influence of formatting on readability and comprehension that is of concern in this paper.

Program readability and comprehension are important areas of research for several reasons. First, it is necessary to fully understand a program in order to select appropriate test data and interpret the output produced with such data. Second, comprehension is essential to debug the logical and semantic aspects of a program. Finally, successful modification of a program requires a thorough understanding of the program. Thus, any factors which can improve program comprehension will have positive effects on (and very likely improve the efficiency of) the software development process.

The objective of this research project is to gather experimental evidence to determine the effects of color on program readability and comprehension. In an attempt to make control structures more visible and easier to follow two different color schemes have been used:

(i) Color-scheme-A (illustrated in Appendix A) used different color codes to bound the scope of loops and conditionals. i.e. the control structures in a particular block were coded in one color; a nested block was coded in another color, and so on.

(ii) Color-scheme-B (illustrated in Appendix A) used different color codes to identify the various statements function in the program. For example, one color was used for I/O statements, another for declaration statements, another for procedure calls, another for repetition statements, while yet another for decision statements, and so on.

2. Techniques for Improving Program Readability and Comprehension

Recent studies on improving program readability and comprehensibility have concentrated on program indentation [7,9, 12] to make the logical structure of programs clearer by allowing visual grouping of statements and visual association between separate parts of control structures. The usefulness of indentation for these purposes, however, is diminished when parts of control

structures are heavily nested or widely separated, for example, when loops cross one or more page boundaries on a listing. Clifton [3] claims that this makes it difficult for a reader to skip around a group of statements or find the path back from the end to the beginning of a loop.

Further studies on improving program readability and comprehensibility have combined indentation with such factors as internal documentation [10,14], blank-line insertion [6,10], control flow [8,14], connector-lines for control structures [3, 11], and solid lines bounding the scope of loops and conditionals [4].

Careful use of internal documentation may help, for example, Norcio [10] found that the use of indentation and one line of interspersed documentation resulted in the highest degree of program comprehension. However, Weissman [14] found that misuse of documentation may actually reduce the clarity of programs.

Some authors consider flowcharts to be very helpful in showing the logical structure of programs [1]. Other authors have questioned this use of flowcharts; one experiment indicated that a combination of flowchart and program listing is at best only slightly easier to understand than a program listing alone [13]. Another technique to improve program readability and comprehension, which was proposed by Clifton [3], involved the use of connector-lines. These lines connect the beginnings to endings of control structures on entire programs.

The author has found no studies (in the computer literature) which investigated the influence of color on program readability and comprehension. This could have been because of the high cost of color printers, however, this excuse is no longer valid. Today, it is only slightly more expensive to print programs in color than in black and white.

3. Experimental Procedures

Hypotheses:

(i) When Color-scheme-A is used, expert and novice Pascal programmers will show only slight increase in program comprehension when compared to programmers who used the identically formatted program without color.

(ii) When Color-scheme-B is used, expert and novice Pascal programmers will show significant increase in program comprehension when compared to programmers who used the identically formatted program without color.

Independent Variables:

1. Color coding schemes:
   (i) Color-scheme-A (described above)
   (ii) Color-scheme-B (described above)
   (iii) No color (i.e. black and white programs).
2. Level of programmer experience:
   (i) Novice: Less than three years of programming experience in school and/or less than two years professionally.
   (ii) Expert: Three or more years of programming experience in school and/or two or more years professionally.

Dependent Variables:

1. Comprehension quiz scores.
2. Subjective rating of the program difficulty.

Subjects:

The novice subjects were selected from an intermediate-level programming class in Pascal at the University of Regina. The experiment was administered in the twelfth week of a thirteen-week semester. By this time, the students had written several Pascal programs beyond the complexity of the program used in the experiment.

The expert subjects were selected from various senior-level Computer Science classes. These students were enrolled in the four-year B.Sc. program majoring in Computer Science.

Materials:

The Pascal program used for both groups was selected from Grogono [5], and contains a wide range of syntactical structures (records, packed-arrays, while-loops, if-then-elses, etc.), making it a challenging program for both novices and experts. This program calculates the frequency count of each unique word in some given text. The program was modified to produce three versions with no blank lines or comments. The three versions (illustrated in Appendix A) consisted of two color versions which were coded using Color-scheme-A and Color-scheme-B, and one regular black and white version. For all versions of the two-page program, the optimal level of indentation [9], namely two spaces was used, and all versions were also divided at the same location when page boundaries were crossed. These programs which were very legible, were distributed to students on computer paper outputted from a near letter-quality color printer.

One of the dependent variables was a subjective rating from 1 to 7 of the difficulty encountered in comprehending the program, with 1 being very easy, 4 moderate, and 7 very hard. The second dependent variable was a comprehension quiz (contained in Appendix B) which the subjects were given thirty minutes to complete.

4. Results

In the final analysis of the data, a total of 79 students were used. Five quizzes were excluded from the analysis for the following reasons: two subjects did not know Pascal, and three subjects were observed not participating in the

	COLOR-SCHEME-A	COLOR-SCHEME-B	NO COLOR	TOTAL
NOVICES	15	14	15	44
EXPERTS	12	12	11	35

Table 1. Breakdown of Subjects per cell

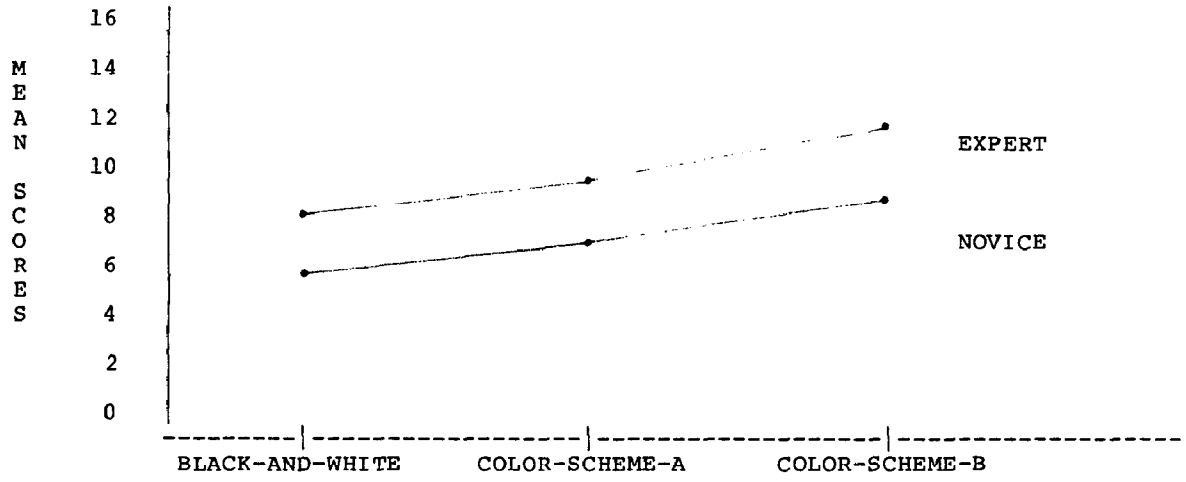

Figure 1. Mean Scores of Novices and Experts

task. Table 1 summarizes the breakdown of subjects per cell.

As was expected, the experts did better on the quizzes than the novices. The mean score was 10.6 for experts and 7.2 for novices, out of a possible 15 points. The highest mean scores were obtained by both the novices and experts who had programs with Color-scheme-B; 11.8 for experts and 8.3 for novices. Both groups also had the lowest mean scores on the black-and-white programs, with 8.0 for experts and 5.8 for novices. These results are summarized in figure 1.

On the subjective rating from 1 to 7, with 1 being very easy, and 7 very hard; novices rated all versions of the program to be more difficult to comprehend than the experts. The average rating of novices was 5.7, while that for experts was 3.6. The black-and-white programs were rated the most difficult to comprehend by both experts and novices. The results of the subjective ratings are summarized in figure 2.

Analysis of the combined results yield similar results as did the groups separately. Those subjects who received the black-and-white programs had a lower mean score than other subjects, while those who received programs with Color-scheme-B had the highest mean score. The subjective program rating for the combined subjects ran about the same as for the separate groups.

The analysis of variance (ANOVA) of the comprehension quiz scores indicated that programming experience had an effect on program comprehension at the $p<0.001$ significance level. The ANOVA also showed that the color coding schemes had a significant effect on the mean scores at the $p=0.015$ level. Approximately 38 percent of the variance of the quiz scores were explained. The ANOVA of the program difficulty ratings also showed that programming experience and color coding schemes had effects on program comprehension at significance levels $p<0.001$ and $p=0.075$, respectively. Approximately 42 percent of the variance in the subjective ratings were explained.

5. Discussion

The results indicate that color coding schemes have a statistically significant effect on program comprehension. The color coding scheme that

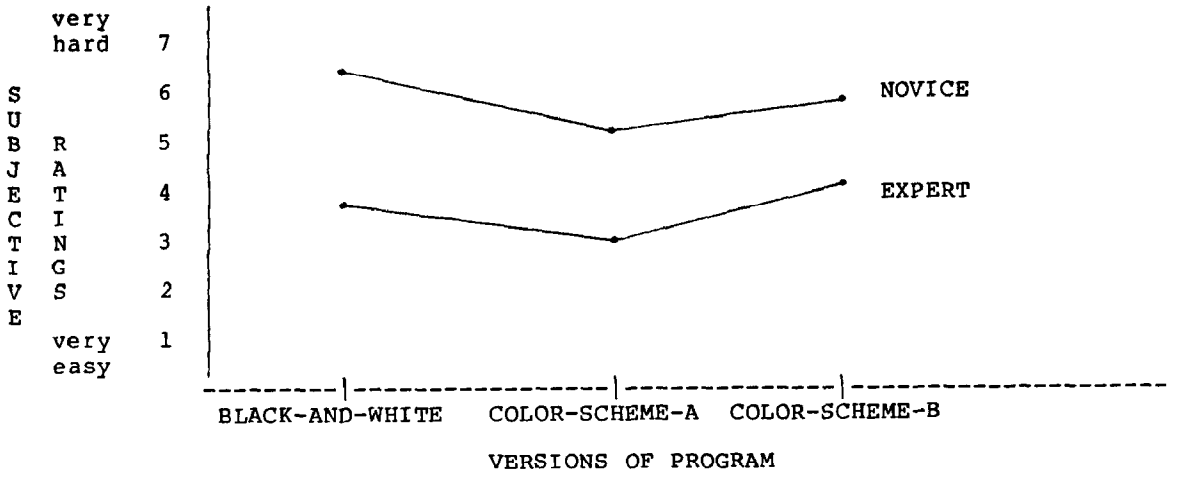

Figure 2. Mean Subjective Program Rating of Novices and Experts

produced optimal results in comprehension is when different colors were used to indicate the various statements function in the program (i.e. Color-scheme-B). The comprehension level decreased when different color codes were used to indicate block structures (i.e. color-shceme-A). This decrease in level of comprehension might be attributed to the fact that the subjects were only accustomed to working with black-and-white programs, and getting a program completely coded with seven different colors might have been confusing to them in the early minutes of the quiz. This remark was in fact voiced by many students who got programs in Color-scheme-A.

The students who received the program with Color-scheme-B were not overwhelmed initially since most of the program was in black-and-white, with only certain reserved words color coded. This scheme was the greatest asset to comprehension because students were able to see quickly and clearly when a procedure call should be made, which statements were in a loop, which were decision satements, which were declaration statements, which were I/O statements, and so on. Consequently, as the students traced the program (to complete the quiz), the color codes made the function of every statement so much more obvious.

These results should not be surprising since in our everyday lives we frequently use color codes to convey information (e.g. red indicates danger, and a green light indicates it is safe to go, etc.).

Overall, experts did better on the comprehension quiz and rated the program less difficult than the novices. These results were reassuring because we expected the experts to do better and to rate this type of task less difficult than novices.

Finally, the combined results of the expert and novice subjects indicated the highest mean scores in the Color-scheme-B programs. It is interesting to note, however, that in the subjective ratings the Color-scheme-A programs were rated as the least difficult to understand. We feel that this result occured because the subjects found the widespread use of colors in the programs visually pleasing, and made the block structures very obvious. However, during the comprehension task, the function of the various statements was not as obvious as in programs with Color-scheme-B, thus resulting in lower scores.

6. Conclusion

The use of color codes in programs is not restricted to Pascal. Any language allowing structured programming could use this technique to make it's programs easier to read and understand. The technique would be less useful in unstructured programs because unrestricted GOTO statements could not be represented by color codes without allowing the colors to overlap. This technique presents practical implications for the teaching of programming; color codes may be used to clearly demonstrate the functions and relationships of the various statements in a program.

In this paper, a color-coded formatting methodology was applied to study its influence on program readability and comprehension. In general, the results indicate that both novices and experts displayed the highest level of program comprehension when different color codes were used to identify the various statements function in a program. The second highest level of program comprehension was achieved (again by both groups) when different color codes were used to indicate different blocks in the program. The lowest level of program comprehension was achieved by novices and experts who used black-and-white programs.

References

[1] Bohl, M. Flowcharting Techniques. Scientific Research Associates, Chicago, 1971.

[2] Boysen, J. and Keller, R. "Measuring Computer Program Comprehension". ACM SIGCSE Bulletin 12, 1 (Feb. 1980), 92-102.

[3] Clifton, M.H. "A Technique for Making Structured Programs more Readable". ACM SIGPLAN Notices 13, 4 (April 1978), 58-63.

[4] Gimpel, J.F. "Contour, A Method of Preparing Structured Flowcharts". ACM SIGPLAN Notices 15, 10 (Oct. 1980), 35-41.

[5] Grogono, P. Programming in Pascal. Addison-Wesley Publishing Company, Inc., Reading, MA, 1978, 186-188.

[6] Hueras, J. and Ledgard, H. "An Automatic Formatting Program for Pascal". ACM SIGPLAN Notices 12, 7 (July 1977), 82-84.

[7] Leinbaugh, D.W. "Indenting for the Compiler". ACM SIGPLAN Notices 15, 5 (May 1980), 41-48.

[8] Love, T. "An Experimental Investigation of the Effect of Program Structure on Program Understanding" Proc. ACM Conference on Language Design for Reliable Software. March 1977, 105-113.

[9] Miara, R. et al. "Program Indentation and Comprehensibility". Communications of the ACM 15, 11 (Nov. 1983), 861-867.

[10] Novcia, A.F. "Indentation Documentation and Programmer Comprehension". Proc. of Human Factors in Computer Systems. ACM Washington, D.C. 1981, 118-120.

[11] Ramsdell, J. "Pretty Printing Structured Programs with Connector Lines". ACM SIGPLAN Notices 14, 9 (Sept. 1979), 74-75.

[12] Shneiderman, B. and McKay, D. "Experimental Investigations of Computer Program Debugging and Modification". Proc. 6th International Congress of the International Ergonomics Association. July 1976, College Park, MD.

[13] Shneiderman, B. et al. "Experimental Investigations of the Utility of Detailed Flowcharts in Programming", Communications of the ACM 20, 6 (June 1977), 373-381.

[14] Weissman, L.M. "Psychological Complexity of Computer Programs: An Experimental Methodology". ACM SIGPLAN Notices 15, 6 (June 1974), 25-36.

```
PROGRAM TEST (INPUT,OUTPUT);
  CONST
    TABLESIZE = 1000;
    MAXWORDLEN = 20;
  TYPE
    CHARINDEX = 1 .. MAXWORDLEN;
    COUNTTYPE = 1 .. MAXINT;
    TABLEINDEX = 1 .. TABLESIZE;
    WORDTYPE = PACKED ARRAY [CHARINDEX] OF CHAR;
    ENTRITYPE =
      RECORD
        WORD : WORDTYPE;
        COUNT : COUNTTYPE
      END;
    TABLETYPE = ARRAY [TABLEINDEX] OF ENTRITYPE;
  VAR
    TABLE : TABLETYPE;
    ENTRI, NEXTENTRI : TABLEINDEX;
    TABLEFULL : BOOLEAN;
    LETTERS : SET OF CHAR;
PROCEDURE READWORD (VAR PACKEDWORD : WORDTYPE);
  CONST
    BLANK = ' ';
  VAR
    BUFFER : ARRAY [CHARINDEX] OF CHAR;
    CHARCOUNT : 0 .. MAXWORDLEN;
    CH : CHAR;
  BEGIN
    IF NOT EOF
      THEN
        REPEAT
          READ(CH)
        UNTIL EOF OR (CH IN LETTERS);
    IF NOT EOF
      THEN
        BEGIN
          CHARCOUNT := 0;
          WHILE CH IN LETTERS DO
            BEGIN
              IF CHARCOUNT < MAXWORDLEN
                THEN
                  BEGIN
                    CHARCOUNT := CHARCOUNT + 1;
                    BUFFER[CHARCOUNT] := CH
                  END;
              IF EOF
                THEN
                  CH := BLANK
                ELSE
                  READ(CH)
            END;
          FOR CHARCOUNT := CHARCOUNT+1 TO MAXWORDLEN DO
            BUFFER[CHARCOUNT] := BLANK;
          PACK(BUFFER,1,PACKEDWORD)
        END
  END;

PROCEDURE PRINTWORD (PACKEDWORD : WORDTYPE);
  CONST
    BLANK = ' ';
  VAR
    BUFFER : ARRAY [CHARINDEX] OF CHAR;
    CHARPOS : 1 .. MAXWORDLEN;
  BEGIN
    UNPACK(PACKEDWORD,BUFFER,1);
    FOR CHARPOS := 1 TO MAXWORDLEN DO
      WRITE(BUFFER[CHARPOS])
  END;
BEGIN
  LETTERS := ['A' .. 'Z'];
  TABLEFULL := FALSE;
  NEXTENTRI := 1;
  WHILE NOT (EOF OR TABLEFULL) DO
    BEGIN
      READWORD(TABLE[NEXTENTRI].WORD);
      IF NOT EOF
        THEN
          BEGIN
            ENTRI := 1;
            WHILE TABLE[ENTRI].WORD<>TABLE[NEXTENTRI].WORD DO
              ENTRI := ENTRI + 1;
            IF ENTRI < NEXTENTRI
              THEN
                TABLE[ENTRI].COUNT := TABLE[ENTRI].COUNT + 1
              ELSE
                IF NEXTENTRI < TABLESIZE
                  THEN
                    BEGIN
                      NEXTENTRI := NEXTENTRI + 1;
                      TABLE[ENTRI].COUNT := 1
                    END
                  ELSE
                    TABLEFULL := TRUE
          END
    END;
  IF TABLEFULL
    THEN
      WRITELN('THE TABLE IS NOT LARGE ENOUGH')
    ELSE
      FOR ENTRI := 1 TO NEXTENTRI-1 DO
        WITH TABLE[ENTRI] DO
          BEGIN
            PRINTWORD(WORD);
            WRITELN(COUNT)
          END
END.
```

Appendix A1. Program Listing in Color-scheme-A

```
PROGRAM TEST (INPUT,OUTPUT);
  CONST
    TABLESIZE = 1000;
    MAXWORDLEN = 20;
  TYPE
    CHARINDEX = 1 .. MAXWORDLEN;
    COUNTTYPE = 1 .. MAXINT;
    TABLEINDEX = 1 .. TABLESIZE;
    WORDTYPE = PACKED ARRAY [CHARINDEX] OF CHAR;
    ENTRITYPE =
      RECORD
        WORD : WORDTYPE;
        COUNT : COUNTTYPE
      END;
    TABLETYPE = ARRAY [TABLEINDEX] OF ENTRITYPE;
  VAR
    TABLE : TABLETYPE;
    ENTRI, NEXTENTRI : TABLEINDEX;
    TABLEFULL : BOOLEAN;
    LETTERS : SET OF CHAR;
PROCEDURE READWORD (VAR PACKEDWORD : WORDTYPE);
  CONST
    BLANK = ' ';
  VAR
    BUFFER : ARRAY [CHARINDEX] OF CHAR;
    CHARCOUNT : 0 .. MAXWORDLEN;
    CH : CHAR;
  BEGIN
    IF NOT EOF
      THEN
        REPEAT
          READ(CH)
        UNTIL EOF OR (CH IN LETTERS);
    IF NOT EOF
      THEN
        BEGIN
          CHARCOUNT := 0;
          WHILE CH IN LETTERS DO
            BEGIN
              IF CHARCOUNT < MAXWORDLEN
                THEN
                  BEGIN
                    CHARCOUNT := CHARCOUNT + 1;
                    BUFFER[CHARCOUNT] := CH
                  END;
              IF EOF
                THEN
                  CH := BLANK
                ELSE
                  READ(CH)
            END;
          FOR CHARCOUNT := CHARCOUNT+1 TO MAXWORDLEN DO
            BUFFER[CHARCOUNT] := BLANK;
          PACK(BUFFER,1,PACKEDWORD)
        END
  END;
PROCEDURE PRINTWORD (PACKEDWORD : WORDTYPE);
  CONST
    BLANK = ' ';
  VAR
    BUFFER : ARRAY [CHARINDEX] OF CHAR;
    CHARPOS : 1 .. MAXWORDLEN;
  BEGIN
    UNPACK(PACKEDWORD,BUFFER,1);
    FOR CHARPOS := 1 TO MAXWORDLEN DO
      WRITE(BUFFER[CHARPOS])
  END;
BEGIN
  LETTERS := ['A' .. 'Z'];
  TABLEFULL := FALSE;
  NEXTENTRI := 1;
  WHILE NOT (EOF OR TABLEFULL) DO
    BEGIN
      READWORD(TABLE[NEXTENTRI].WORD);
      IF NOT EOF
        THEN
          BEGIN
            ENTRI := 1;
            WHILE TABLE[ENTRI].WORD<>TABLE[NEXTENTRI].WORD DO
              ENTRI := ENTRI + 1;
            IF ENTRI < NEXTENTRI
              THEN
                TABLE[ENTRI].COUNT := TABLE[ENTRI].COUNT + 1
              ELSE
                IF NEXTENTRI < TABLESIZE
                  THEN
                    BEGIN
                      NEXTENTRI := NEXTENTRI + 1;
                      TABLE[ENTRI].COUNT := 1
                    END
                  ELSE
                    TABLEFULL := TRUE
          END
    END;
  IF TABLEFULL
    THEN
      WRITELN('THE TABLE IS NOT LARGE ENOUGH')
    ELSE
      FOR ENTRI := 1 TO NEXTENTRI - 1 DO
        WITH TABLE[ENTRI] DO
          BEGIN
            PRINTWORD(WORD);
            WRITELN(COUNT)
          END
END.
```

Appendix A2. Program Listing in Color-scheme-B

```
PROGRAM TEST (INPUT , OUTPUT);
  CONST
    TABLESIZE = 1000;
    MAXWORDLEN = 20;
  TYPE
    CHARINDEX = 1 .. MAXWORDLEN;
    COUNTTYPE = 1 .. MAXINT;
    TABLEINDEX = 1 .. TABLESIZE;
    WORDTYPE = PACKED ARRAY [CHARINDEX] OF CHAR;
    ENTRITYPE =
      RECORD
        WORD : WORDTYPE;
        COUNT : COUNTTYPE
      END;
    TABLETYPE = ARRAY [TABLEINDEX] OF ENTRITYPE;
  VAR
    TABLE : TABLETYPE;
    ENTRI, NEXTENTRI : TABLEINDEX;
    TABLEFULL : BOOLEAN;
    LETTERS : SET OF CHAR;
PROCEDURE READWORD (VAR PACKEDWORD : WORDTYPE);
  CONST
    BLANK = ' ';
  VAR
    BUFFER : ARRAY [CHARINDEX] OF CHAR;
    CHARCOUNT : 0 .. MAXWORDLEN;
    CH : CHAR;
  BEGIN
    IF NOT EOF
      THEN
        REPEAT
          READ(CH)
        UNTIL EOF OR (CH IN LETTERS);
    IF NOT EOF
      THEN
        BEGIN
          CHARCOUNT := 0;
          WHILE CH IN LETTERS DO
            BEGIN
              IF CHARCOUNT < MAXWORDLEN
                THEN
                  BEGIN
                    CHARCOUNT := CHARCOUNT + 1;
                    BUFFER[CHARCOUNT] := CH
                  END;
              IF EOF
                THEN
                  CH := BLANK
                ELSE
                  READ(CH)
            END;
          FOR CHARCOUNT := CHARCOUNT+1 TO MAXWORDLEN DO
            BUFFER[CHARCOUNT] := BLANK;
          PACK(BUFFER,1,PACKEDWORD)
        END
  END;
PROCEDURE PRINTWORD (PACKEDWORD : WORDTYPE);
  CONST
    BLANK = ' ';
  VAR
    BUFFER : ARRAY [CHARINDEX] OF CHAR;
    CHARPOS : 1 .. MAXWORDLEN;
  BEGIN
    UNPACK(PACKEDWORD,BUFFER,1);
    FOR CHARPOS := 1 TO MAXWORDLEN DO
      WRITE(BUFFER[CHARPOS])
  END;
BEGIN
  LETTERS := ['A' .. 'Z'];
  TABLEFULL := FALSE;
  NEXTENTRI := 1;
  WHILE NOT (EOF OR TABLEFULL) DO
    BEGIN
      READWORD(TABLE[NEXTENTRI].WORD);
      IF NOT EOF
        THEN
          BEGIN
            ENTRI := 1;
            WHILE TABLE[ENTRI].WORD<>TABLE[NEXTENTRI].WORD DO
              ENTRI := ENTRI + 1;
            IF ENTRI < NEXTENTRI
              THEN
                TABLE[ENTRI].COUNT := TABLE[ENTRI].COUNT + 1
              ELSE
                IF NEXTENTRI < TABLESIZE
                  THEN
                    BEGIN
                      NEXTENTRI := NEXTENTRI + 1;
                      TABLE[ENTRI].COUNT := 1
                    END
                  ELSE
                    TABLEFULL := TRUE
          END
    END;
  IF TABLEFULL
    THEN
      WRITELN('THE TABLE IS NOT LARGE ENOUGH')
    ELSE
      FOR ENTRI := 1 TO NEXTENTRI - 1 DO
        WITH TABLE[ENTRI] DO
          BEGIN
            PRINTWORD(WORD);
            WRITELN(COUNT)
          END
END
```

Appendix A3. Program Listing in Black-and-White

APPENDIX B

COMPREHENSION QUIZ

1) Circle the global variables in the following list:
   TABLE CHARCOUNT BUFFER ENTRI LETTERS TABLEFULL

2) What is the maximum number of words the input file can hold?
   _____

3) Assume that the input file starts as follows:
   PROTONS, WHICH ARE NONPHOTOMICROGRAPHICAL, ARE ...
   Give the values of the following variables:
   TABLE[1].WORD _____
   TABLE[2].WORD _____
   TABLE[3].WORD _____
   TABLE[4].WORD _____

4) Why does the variable ENTRI, in the FOR loop of the main segment of the program, stop at NEXTENTRI - 1, and not at NEXTENTRI?

5) What output is produced with the following input?
   "HE", HE SAID, SAID "WHAT". WHAT SAID HE?
   HE SAID, "WHAT".

6) In less than 25 words, describe what this program does.

7) Circle the number indicating the difficulty encountered in comprehending the program.

1	2	3	4	5	6	7
very easy			moderate			very hard

8) List the Computer Science classes which you have taken.

_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____