

A FEASIBILITY STUDY OF TYPOGRAPHICAL PROGRAMMING

A FEASIBILITY STUDY IN TYPOGRAPHICAL PROGRAMMING

By

ALBERT CHAN, B. Sc.

A Project

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Science

McMaster University

(c) Copyright by Albert Chan, January 1994

MASTER OF SCIENCE (1994)
(Computer Science)

McMASTER UNIVERSITY
Hamilton, Ontario

TITLE : A Feasibility Study in Typographical Programming

AUTHOR : Albert Chan, B. Sc. (National Taiwan University)

SUPERVISOR : Dr. N. Solntseff

NUMBER OF PAGES : vii, 145

ABSTRACT

Today's computers are much more capable than they were 10 or 20 years ago. Most software developed today uses the WIMP (Window, Icon, Menu, Pointing) interface, and WYSIWYG is not a deluxe requirement anymore. All these advancements lead us into the age of Typographical Programming. The purpose of this project is to carry out a feasibility study of this new programming methodology and present a prototype using FORTH as the programming language, and WordPerfect as the formatting processor.

KEYWORD : FORTH, Literate Programming, Markup Language, Pygmy, SGML, Typographical Programming, WPFORTH, WordPerfect, Word Processor.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Nicholas Solntseff, for suggesting the topic of this project as well as for his guidance of the language FORTH.

Thanks also go to WordPerfect Corporation for their providing information on the structure of WordPerfect 5.1 documents as well as the Beta Document of Developer's Toolkits for WordPerfect 6.0 for DOS.

I also have to thank Mr. Frank C. Sergeant, the author of Pygmy FORTH, for allowing me to use Pygmy FORTH as the basic of WPFORTH. This greatly reduced the work needed to be done.

Finally, I must thank my wife Rita for her tolerance to my working late during the period of this project, and her support for this project. Without her support, the finishing date of the project was not even predictable.

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	LITERATE PROGRAMMING	6
III.	TYPOGRAPHICAL PROGRAMMING	16
IV.	FORTH, PYGMY AND WPFORTH	27
V.	WORDPERFECT DOCUMENT	39
VI.	WPFORTH IMPLEMENTATION DETAILS	53
VII.	MARKUP LANGUAGES AND SGML	72
VIII.	DISCUSSION AND CONCLUSION	84
APPENDICES		
A.	SYSTEM REQUIREMENT	91
B.	SOURCE LISTING OF WPFORTH	92
C.	TESTING	135
	BIBLIOGRAPHY	142

LIST OF FIGURES

Fig. 1 :	Evolution of Programming Methodology	4
Fig. 2 :	Procedure to Produce an Executable Program .	8
Fig. 3 :	An Excerpt from a WEB Program	10
Fig. 4 :	An Excerpt from a WEB Program	11
Fig. 5 :	An Excerpt from a WEB Program	12
Fig. 6 :	A Pascal Program Generated from a WEB File .	13
Fig. 7 :	T _E X Program Generated from a WEB File	14
Fig. 8 :	A T _E X Document	18
Fig. 9 :	The Output of the T _E X Document in Figure 8 .	20
Fig. 10 :	A Sample FORTH Program	29
Fig. 11 :	Structure of a WordPerfect Document	40
Fig. 12 :	Handling of WordPerfect Functions	55
Fig. 13 :	Data Structure Used in Font Segment	65
Fig. 14 :	Data Structure Used in File Segment	68
Fig. 15 :	Segments Used in Original Pygmy FORTH	71
Fig. 16 :	Structure of an SGML Document	77
Fig. 17 :	A Partial DTD for a FORTH Program	79
Fig. 18 :	A Partial DTD for a Colon Definition	79
Fig. 19 :	An Excerpt of an Encoded WPFORTH Program . .	83

LIST OF TABLES

Table 1 : Structure of File Prefix in a WordPerfect	
Document	41
Table 2 : WordPerfect Special Packets	42
Table 3 : Structure of a Special Packet Index	43
Table 4 : Structure of a General Packet Index	43
Table 5 : General Packets Defined in WordPerfect	44
Table 6 : Control Characters Used by WordPerfect	45
Table 7 : ASCII Characters Used by WordPerfect	46
Table 8 : Single-Byte Functions Defined in WordPerfect	47
Table 9 : Fixed-Length, Multi-byte Functions Defined in WordPerfect	48
Table 10 : Variable-Length, Multi-byte Functions Defined in WordPerfect	48
Table 11 : WordPerfect Functions Considered in WPFORTH	56
Table 12 : Structure of WP's Attribute ON/OFF Functions	58
Table 13 : WordPerfect's Attribute Codes	59
Table 14 : Structure of WordPerfect's Font Change Function	63

I. INTRODUCTION

In the earliest stages of the computer age, hardware was expensive so that there could only be a limited amount of resources built into a computer. In fact, the earliest computers were not as powerful as today's personal computer. The old machines were usually time-shared to reduce the capacity and operation costs.

Because of the high cost of hardware and the relatively low cost of manpower, programmers had to optimize their programs to fit the limited available resources. They also had to make their programs run as fast as possible to reduce the CPU time charges. At that time, size and speed of programs had the highest priority. Readability, on the other hand, was not considered as important.

Programmers usually wrote their programs in Assembler and some old languages, such as FORTRAN IV or COBOL 74. Since these languages do not support (or encourage) the concept of structured programming, the resulting code usually looked like spaghetti, making program modification and maintenance a nightmare.

In the 1970s, the cost of hardware dropped, and the cost (wages) of programmers increased. Since the old "spaghetti"-like programs were very difficult (if not impossible) to modify and maintain, the total cost to produce and maintain a program increased. Therefore, the concept of "Structured Programming" appeared. Through the use of structure blocks, program source code became more readable and easier to understand. Although the resulting code may be larger and slower, the increased capacity and speed of the computers by comparison with the old machines meant that the performance loss is not noticeable and the resulting saving in cost encouraged these "imperfect" codes.

At the present, the cost of hardware is still falling, whereas the cost of programmer hours continues to increase. Industries must find ways to further reduce the cost of maintaining programs. This leads to the concept of "Literate Programming". It was first introduced by Dr. D. E. Knuth in 1984 [KNU 84]. In his paper, Dr. Knuth introduced a new programming approach that will make a program look like an essay or a book. However, literate programming is not widely used since it requires the programmer to use special formatting tools (such as T_EX or troff) to format (markup) a program. This made writing programs much more time consuming, although the resulting programs are very easy to understand.

In literate programming, programmers first prepare programs with document formatting tools to produce original documents. The documents are then processed by two routines, *tangle* and *weave*. The purpose of the *tangle* routine is to change the original documents into programs in ASCII format understood by compilers. The purpose of the *weave* routine is to produce a typesetting output file, from which high quality printout can be produced.

Today, the capability of a normal personal computer is usually much better than that of a mid-range mainframe of the 1970s. In addition, the popularity of low-cost laser printers means that desktop publishing is now available to all corporations or even individuals. In fact, many of the current word processor software (e.g., WordPerfect for Windows, Microsoft Word for Windows, AmíPro) are more capable than the desktop publishing software of ten years ago. We can now include more information apart from the structure into our source code. We can format our programs directly on the screen instead of through another formatting program. We can also assign meanings to different typeface or attributes (e.g. use italics to represent comments). However, the compiler must be able to understand the meanings of all the attributes (or at least, we must be able to translate the programs into ASCII format and include all this typeface and attribute information). This introduces the concept of "Typographical

Programming". Therefore, the approach of typographical programming is to utilize all the necessary attribute information available in the word processor. We can say that "Literate Programming" is "Indirect Typographical Programming". Figure 1 illustrates the evolution of the programming method.

The purpose of this project is to investigate the feasibility of the typographical programming

environment, and implement a prototype to illustrate how it works. The prototype is based on Mr. Frank C. Sergeant's Pygmy FORTH, Version 1.4. The resulting product is named as WPFORTH, which uses WordPerfect 5.x as the formatting tools.

In this thesis, I will discuss all the necessary background knowledge and the implementation details about this

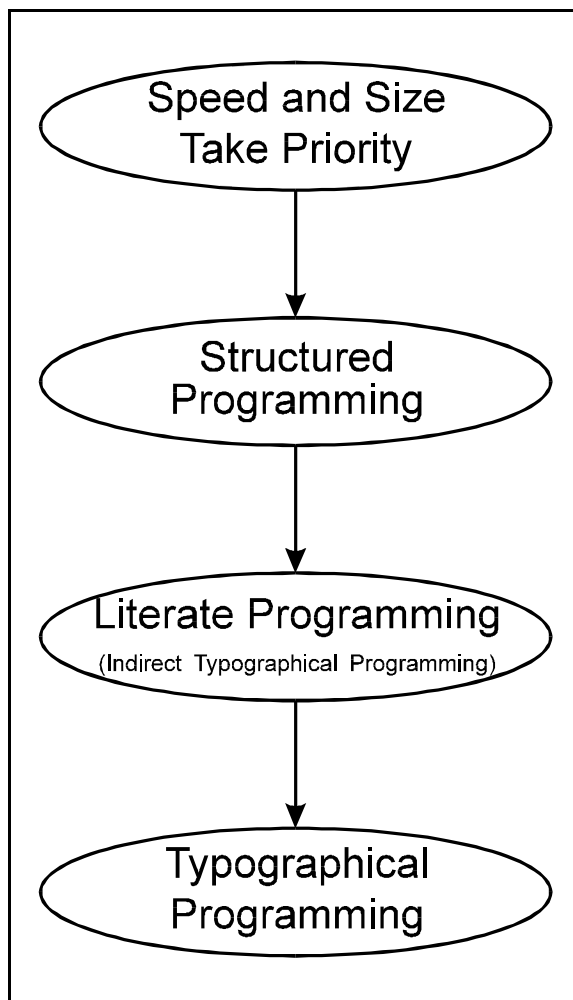


Fig. 1 : Evolution of Programming Methodology

project. Chapter II describes Dr. Knuth's Literate Programming and his WEB system. Chapter III discusses the concept of Typographical Programming in details. Chapter IV describes the features of FORTH – the choice of programming language used in this project. Chapter V describes the internal structure of a WordPerfect document – the choice of word processor used in this project. Chapter VI discusses the implementation details about WPFORTH – the resulting system of this project. Chapter VII contains some discussion about the multi-platform exchange of WPFORTH programs. Chapter VIII will conclude the project.

II. LITERATE PROGRAMMING

2.1 The Concept of Literate Programming

In the past, to produce an executable program, programmers first used a text editor to produce the source code (in ASCII or EBCDIC format), then they called a compiler (or an assembler) to compile the source code into an object module, and finally, they invoked a linkage editor to link the object modules together. If any errors occurred during this process, the whole Edit - Compile - Link procedure must be re-started.

Since the main purpose of the source code is to instruct the computer what to do, it is written in a format recognizable by the computer (or more specifically, the compiler), usually pure ASCII or EBCDIC format. It also contains sections that are only used for communication between people, usually in the form of comments or remarks. These names imply that they are not important parts of programs, so people tend to ignore them. However, a complex program today usually contains thousands of lines of codes, so that these comments and remarks are actually very important for program

maintenance, especially when the person maintaining the program is not the original programmer.

Dr. Knuth's "Literate Programming" changes the roles of program code and comments or remarks. The main purpose of a "literate" program is not to instruct the computer what to do, instead, it is to tell the reader how to solve the given problem. Therefore, the main part of a "literate" program is a description of the solution of the problem. The code is only there to illustrate the solution. Of course, the code also instructs the computer what to do, but this is only its secondary purpose.

As a result of this change of role, the ASCII or EBCDIC format is no longer adequate for the reader. Readers of the programs usually prefer the programs they are reading to look like a book or an essay. Therefore, we need new formatting tools. There are two very popular formatting tools in the Unix world, one is T_EX, another is troff. These formatting tools can prepare a document to a typesetting standard. However, the formatted source cannot be read by the compiler, therefore, we must add one more step into the edit - compile - link cycle. Dr. Knuth called this step as *tangle*, and the programming cycle now becomes edit - tangle - compile - link. Figure 2 illustrates the steps needed to produce a source printout and an executable program from the source code

in Dr. Knuth's WEB system [KNU 84]. The WEB system is described in detail in next section.

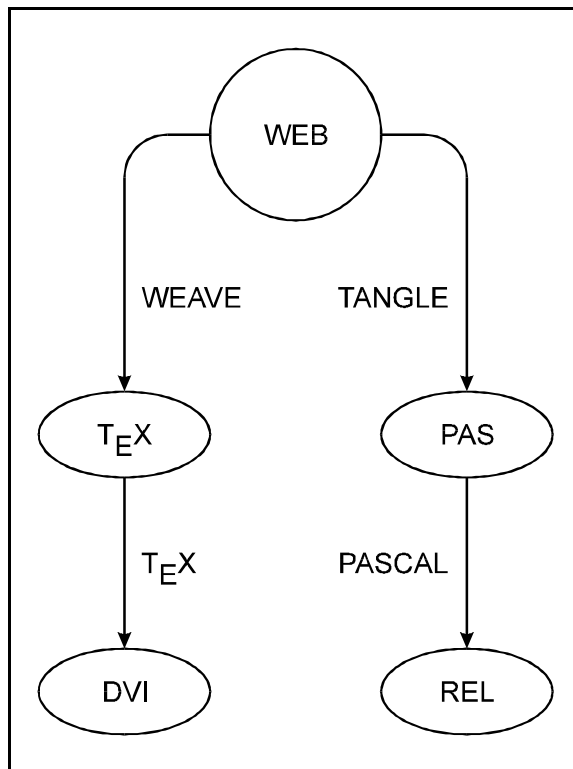


Fig. 2 : Procedure to Produce an Executable Program

2.2 The WEB System

Dr. Knuth has implemented a literate programming system in 1984. He named this system "WEB" because he thought that a program, like a web, should be composed from simple components.

WEB is a combination of two languages: a document formatting language and a programming language. Dr. Knuth chose Pascal as the programming language since it is very well known in the academic world. The document formatting language he used is T_EX. However, as he claimed in his article:

the same principles would apply equally well if other languages were substituted. Instead of T_EX, one could use a language like Scribe or troff; instead of Pascal, one could use ADA, ALGOL, LISP, COBOL, FORTRAN, APL, C, etc., or even assembly language. The main point is that WEB is inherently bilingual, and that such a combination of languages proves to be much more powerful than either single language by itself. WEB does not make the other languages obsolete; on the contrary, it enhances them. [KNU 84]

```

\font\ninerm=cmr9
\let\mc=\ninerm % medium caps
\def\WEB{{\tt WEB}}
\def\PASCAL{{\mc PASCAL}}
\def\[\{\ifhmode\ \fi$\![\![$}
\def\]{$\![$\ }
.
.
.
\hyphenation{Dijk-stra}

@* Printing primes: An example of \WEB. The
following program is essentially the same as
Edsger Dijkstra's @^Dijkstra, Edsger@> ``first
example of step-wise program composition.'' found
on pages 26--39 of his {\sl Notes on Structured
Programming},$^\Dijk$ but it has been translated
into the \WEB\ language. @.WEB@>

\[[Double brackets will be used in what follows to
enclose comments relating to \WEB\ itself,
.
.
.
an informal top-level description.\]

@p @<Program to print the first thousand prime
numbers@>

```

Fig. 3 : An Excerpt from a WEB Program

@ This program has no input, because we want to keep it rather simple. The result of the program will be to produce a list of the first thousand prime numbers, and this list will appear on the |output| file.

Since there is no input, we declare the value |m=1000| as a compile-time constant. The program itself is capable of generating the first |m| prime numbers for any positive |m|, as long as the computer's finite limitations are not exceeded.

\[The program text below specifies the ``expanded meaning'' of '\X2:Program to print \$\ldots\$ numbers\X'; notice that it involves the top-level descriptions of three other sections. When those top-level descriptions are replaced by their expanded meanings, a syntactically correct \PASCAL\ program will be obtained.\]

```
@<Program to print...@>=
program print_primes(output);
const @!m=1000;
@<Other constants of the programs@>@;
var @<Variables of the program@>@;
begin @<Print the first |m| prime numbers@>;
end.
```

Fig. 4 : An Excerpt from a WEB Program

```

@d In order to keep this program reasonably free
of notations that are uniquely \PASCAL esque,
\[and in .
.
.
The first three macro definitions here are
parametric; the other two are simple.\]

@d print_string(##)=write(##)
  { put a given string into the |output| file }

@d print_integer(##)=write(##:1)
  { put a given integer into the |output| file,
    in decimal notation, using only as many
    digit positions as necessary }

@d print_entry(##)=write(##:ww)
  { like |print_integer|, but |ww| character
    positions are filled, inserting blanks at
    the left }

@d new_line==write_ln
  { advance to a new line in the |output| file }

@d new_page==page
  { advance to a new page in the |output| file }

```

Fig. 5 : An Excerpt from a WEB Program

```

{1:}{2:}PROGRAM PRINTPRIMES(OUTPUT);
CONST M=1000;{5:}RR=50;CC=4;WW=10;{5:}{19:}
ORDMAX=30;{19:}VAR{4:}
P:ARRAY[1..M]OF INTEGER;{4:}{7:}
PAGENUMBER:INTEGER;PAGEOFFSET:INTEGER;
ROWOFFSET:INTEGER;C:0..CC;{7:}{12:}J:INTEGER;
K:0..M;{12:}{15:}JPRIME:BOOLEAN;{15:}{17:}
ORD:2..ORDMAX;SQUARE:INTEGER;{17:}{23:}
N:2..ORDMAX;{23:}{24:}
MULT:ARRAY[2..ORDMAX]OF INTEGER;{24:}
BEGIN{3:}{11:}{16:}J:=1;K:=1;P[1]:=2;{16:}
{18:}ORD:=2;SQUARE:=9;{18:}
WHILE K<M DO BEGIN{14:}REPEAT J:=J+2;{20:}
IF J=SQUARE THEN BEGIN ORD:=ORD+1;{21:}
SQUARE:=P[ORD]*P[ORD];{21:}{25:}
MULT[ORD-1]:=J;{25:};END{20:};{22:}N:=2;
JPRIME:=TRUE;
WHILE(N<ORD)AND JPRIME DO BEGIN{26:}
WHILE MULT[N]<J DO MULT[N]:=MULT[N]+P[N]+P[N]
;IF MULT[N]=J THEN JPRIME:=FALSE{26:};N:=N+1;
END{22:};UNTIL JPRIME{14:};K:=K+1;P[K]:=J;
END{11:};{8:}BEGIN PAGENUMBER:=1;
PAGEOFFSET:=1;
WHILE PAGEOFFSET<=M DO BEGIN{9:}
BEGIN WRITE('The First ');WRITE(M:1);
WRITE(' Prime Numbers --- Page ');
WRITE(PAGENUMBER:1);WRITELN;WRITELN;
FOR ROWOFFSET:=PAGEOFFSET TO PAGEOFFSET+RR-1
DO{10:}
BEGIN FOR C:=0 TO CC-1 DO IF ROWOFFSET+C*RR<=
M THEN WRITE(P[ROWOFFSET+C*RR]:WW);WRITELN;
END{10:};PAGE;END{9:};
PAGENUMBER:=PAGENUMBER+1;
PAGEOFFSET:=PAGEOFFSET+RR*CC;END;END{8:}{3:};
END.{2:}{1:}

```

Fig. 6 : A Pascal Program Generated from a WEB File

```

\input webmac
\font\ninerm=arm9
.
.
syntactically correct \PASCAL\ program will
be obtained.\]

\Y\PS\4\X2:Program to print the first
thousand prime numbers\X=S\6
\4\&\{program}\1\ \37$\{\print\_primes}\(%
\{\output}\)$;\6
\4\&\{const} \37$\|m=1000$;\5
\X5:Other constants of the program\X\6
\4\&\{var} \37\X4:Variables of the program\X\6
\&\{begin} \37\X3:Print the first \|m prime
numbers\X;\6
\&\{end}.\par
\U section~1.\fi
.
.
The first three macro definitions here are
parametric; the other two are simple.\]

\Y\PD \37$\{\print\_string}\(\#)\S\{\write}\(%
\#)\$C{put a given string into the %
\{\output} file}\par
.
.
\inx
\:{Bertrand, Joseph, postulate}, 21.
\:\{\boolean}, 15.
.
.
\:\{WEB}, 1.
\:\{\write}, 6.
\:\{\write\_ln}, 6.
\:\{\ww}, \[5], 6.
\fin
.
.
\:\X4, 7, 12, 15, 17, 23, 24:Variables of
the program\X
\U section~2.
\con

```

Fig. 7 : T_EX Program Generated from a WEB File

A WEB program serves as the source of two different system routines. The TANGLE routine reads the WEB program and produces a Pascal program. This program does not have to be human readable since its sole purpose is to be the source of the Pascal compiler. Similarly, the WEAVE routine reads the WEB program and produces a T_EX file. Again it does not have to be human readable as it is used as the input to the T_EX system (Refer to Figure 2 above). Figure 3-5 show excerpts from a WEB program. Figure 6 is the Pascal program generated from the WEB file, and figure 7 is an excerpt of the T_EX file generated from the same WEB file.

III. TYPOGRAPHICAL PROGRAMMING

3.1 Basic Concepts

Although the concept of Knuth's "Literate Programming" sounds very powerful and interesting, it has an inherent disadvantage. The programmer must also be fluent in the document formatting language that, along with the programming language, is embedded into the WEB system. As one can notice from the figures shown earlier, the document formatting language is not easy to use and is not easy to understand (although its output can be very pretty). Figure 8 and 9 illustrates a T_EX document and its output. Since we can only work through the document formatting language, we can call this as "Indirect Typographical Programming", when comparing with "Typographical Programming" discussed below.

With the low cost of computer resources today, it will be much easier to use a word processor (WP) or even a desktop publishing (DTP) software to format our programs. Examples of such programs include PageMaker, Ventura Publisher, AmíPro, Microsoft Word, WordPerfect for Windows and WordPerfect for DOS. Most of these programs provide WYSIWYG (What you see is

what you get) capacity, i.e., as programmers edit a program, they can know immediately on the screen what their program will look like when it is printed. The only product of the above group without WYSIWYG capacity is WordPerfect for DOS. However, its preview mode, as well as its excellent design, makes it a much better working environment than other non-WYSIWYG document formatting languages.

Since working with a DTP software is much better than working with non-WYSIWYG document formatting languages, most programmers will eventually prefer to work directly inside such DTP software. All of these programs can export a document in ASCII format, so that programs written using them can be understood by a compiler. This will lead us back to the same situation as with the WEB system; the program production cycle now becomes: Edit - Export - Compile - Link. However, it will be preferable if the compiler can understand the program even if the program is stored in a word processor's native format. This leads us to the concept of "Typographical Programming".

According to Mr. Paul W. Abrahams, there are three different representations of a program: visual representation, internal representation, and interchange representation. In the traditional approach, all the three representations are the same. However, this is not necessary. Actually, in the

```

\begin{document}
\begin{center}
{\bf {\footnotesize W E L C O M E ~T O ~T $_{\rm E}$ X}}
\end{center}
\nwln
\begin{center}
{\bf {\small W E L C O M E ~T O ~T $_{\rm E}$ X}}
\end{center}
\nwln
\begin{center}
{\bf W E L C O M E ~T O ~T $_{\rm E}$ X}
\end{center}
\nwln
\begin{center}
{\bf {\large W E L C O M E ~T O ~T $_{\rm E}$ X}}
\end{center}
\nwln
\begin{center}
{\bf {\Large W E L C O M E ~T O ~T $_{\rm E}$ X}}
\end{center}
\nwln
\begin{center}
{\bf {\LARGE W E L C O M E ~T O ~T $_{\rm E}$ X}}
\end{center}
\nwln
\begin{center}
{\bf {\Large W E L C O M E ~T O ~T $_{\rm E}$ X}}
\end{center}
\nwln
\begin{center}
{\bf {\large W E L C O M E ~T O ~T $_{\rm E}$ X}}
\end{center}
\nwln
\begin{center}
{\bf W E L C O M E ~T O ~T $_{\rm E}$ X}
\end{center}
\nwln
\begin{center}
{\bf {\small W E L C O M E ~T O ~T $_{\rm E}$ X}}
\end{center}
\nwln
\begin{center}
{\bf {\footnotesize W E L C O M E ~T O ~T $_{\rm E}$ X}}
\end{center}
\end{document}

```

Fig. 8 : A T_EX Document

typographical programming approach, all three representations are different. The visual representation is the native format of the word processor (i.e., the format used by the word processor to save the documents). The internal representation is the format that the compiler can understand (usually in ASCII format). The interchange representation is used to exchange programs between platforms (usually in ASCII format also but will include all the formatting codes in the form of tags). These three representation will be discussed in details in the following sections. [ABR 93]



Fig. 9 : The Output of the T_EX Document in Figure 8

3.2 Visual Representation

The visual representation of a program is what one can see about the program. In the past, when programs were restricted to ASCII (or EBCDIC) format, the visual representation was the same as the other (internal and interchange) representations of the program. However, as pointed out by Paul W. Abrahams:

In a programming language that uses extended typography, the visual representation of a program is the representation displayed on a screen or printed on a page. The visual representation is a WYSIWYG one in which the appearance of a construct reflects its meaning. Thanks to the capabilities of most display devices, the visual representation presents many opportunities and few problems. On color terminals we can use different colors as well as different typefaces to represent different syntactic elements; the appropriate choices are determined by human factors considerations rather than by technical limitations. Users can customize the presentation according to their individual preferences. [ABR 93]

One example of separating the visual representation from the other representations is the latest versions of Turbo Pascal and Borland Pascal. When displayed within the Integrated Development Environment(IDE), the different syntactic elements are displayed in different colors and font attributes (such as italic, boldface, etc.).

However, this is not yet a typographical programming environment because the attribute display is done by the IDE itself, whereas the program is still (and can only be) an ASCII file. Any file format other than ASCII will not be understood by the compiler. Although the visual representation is different from other representations, when the program is translated into its internal form, all attributes pertaining to the visual representation will be lost.

The main point of typographical programming is that we make the compiler understand the visual representation, therefore all (necessary) attributes can be translated into the internal representation.

3.3 Internal Representation

The internal representation of a program is what the compiler can see of the program. Again, traditional programs which use ASCII and/or EBCDIC format have the same internal representation as the visual representation. But this is not necessary for a typographical programming environment where visual attributes and typeface information usually present in the form of control codes, must be translated into something that the compiler can understand.

In the WEB system for example, the T_EX document is the visual representation of the WEB program. The Pascal program generated by TANGLE can be thought of as the internal representation of the WEB program. The compiler can understand the generated Pascal program, although it may be difficult for human to read the code.

However, the translation from visual representation to internal representation is an extra step in the Edit - Tangle - Compile - Link cycle, and we must deal with a separate document formatting language instead of the visual representation itself, I have called this an "Indirect Typographical Programming" environment.

In the real Typographical Programming environment, the compiler/interpreter can understand the language used for visual representation and use it DIRECTLY in the translation process. The extra translation step is transparent to the programmers. Also, the programmers can work DIRECTLY with the Word Processor or DTP software, instead of having to deal with a separate document formatting language. This ease of use is the real advantage of Typographical Programming.

3.4 Interchange Representation

In the past, programs were written in pure ASCII and/or EBCDIC format. Interchanging these programs among different platforms does not present any problems, as long as compatible compilers are available on all platforms. However, a Typographical Programming Environment presents more problems because programs are now written in the native format of a word processor or DTP software, so that control codes are now embedded inside the program text. To facilitate interchanging programs among platforms, we have to introduce another representation of the program which we will call the interchange representation. It will contain only a standard printable character set. For example, we can represent the exponential operation " \uparrow " using the notation "`\op{exponential}`". Of course, this kind of explicit tagging is not suitable for humans, but makes it easy to remove potential ambiguities from the interchange representation.

[ABR 93]

Ideally, we should be able to compile a program directly from its interchange representation. But another acceptable approach is to first translate the program from interchange representation back to its internal representation or visual representation. That means, we might need a set of

translators which are responsible for translating the program between different representations.

It would be wise to make the interchange representation as standard as possible. We can adopt some widely available markup scheme. Here are some examples: T_EX, troff, SGML (Standard Generalized Markup Language), ODA (Office Document Architecture) and RTF (Rich Text Format). The markup languages will be discussed in more detail below.

[ISO 86] [SMI 86] [HOL 87] [SMI 87-1] [SMI 87-2] [HOG 88]
[BRO 89] [WU 89] [GOL 90] [HER 90] [BOR-BOR 91]

IV. FORTH, PYGMY AND WPFORTH

4.1 History of FORTH

FORTH was invented by Mr. Charles H. Moore over a period of time starting in the early 1960's. At that time, Mr. Moore was working for the National Radio Astronomy Observatory (NRAO) on a variety of programs for such diverse applications as satellite orbit calculation, chromatography, and business systems. He felt that too much time was wasted using the traditional programming languages such as FORTRAN, ALGOL, and other languages of that day. He decided to invent a tool to help increase his productivity. In 1968, he finished his first FORTH system on an IBM 1130 computer. The system was named FORTH because he felt that it should be the "fourth generation" language for this "third generation" computer, but the IBM 1130 only allowed five-character identifiers, so the letter "U" was removed from the word "FOURTH", and the name "FORTH" created.

In 1971, Mr. Moore added a compiler to his system and, in 1973, multiprogramming capacity. In those days, FORTH was used mainly in controlling astronomy equipment. In this way

FORTH spread throughout the astronomy observatories all over the world.

Because of interest in FORTH shown by astronomers, Mr. Moore and a few other FORTH enthusiasts left NRAO in 1973, and formed FORTH Incorporated. During its first few years, the company mainly developed astronomical applications, but subsequently they also developed many general-purpose commercial FORTH systems.

FORTH was quickly "discovered" by individuals around the world, and several user groups were formed. The most important among them are the European FORTH Users' Group (EFUG) and the FORTH Interest Group (FIG).

Figure 10 is a sample FORTH program that determine whether a given number is a prime number or not.

```

( This word will accept an integer and determine whether it is a prime )
( number or not, by returning a flag )
( )
( It will first test whether the given number is divisible by 2, if )
( so, it is NOT a prime number and a "False" flag will be returned. )
( )
( If the number is not divisible by 2, then it might be a prime )
( number, we start the divisor from 3 and increase it by 2, the loop )
( will continue until one of the following conditions is met : )
( )
( 1. the number is divisible by the divisor, in this case, the number )
( is NOT a prime number, so a "False" flag is returned. )
( )
( 2. the quotient is greater than the divisor, in this case, the )
( number is a prime number, so a "True" flag is returned. )
( )
( Example : 1234 .PRIME )
( 1234 is not a prime number. )
( ok )

: PRIME ( n --- f )
ABS ( n' ) ( Get the absolute value )
DUP 2 U/MOD ( n' r q ) ( Divide by 2 )
DROP ( n' r ) ( We do not need the quotient )
0= ( n' f1 ) ( Is the remainder = 0? )
IF ( n' ) ( Yes, it is )
  DROP ( - ) ( Drop the number )
  0 0= NOT ( f ) ( Return a "False" flag )
ELSE ( n' ) ( No, it is not )
  3 ( n' 3 ) ( Start from 3 )
  BEGIN ( n' t ) ( Loop )
    OVER OVER ( n' t n' t ) ( Duplicate the numbers )
    U/MOD ( n' t r' q' ) ( Perform the division )
    PUSH ( n' t r' ) ( Save the quotient )
    0= NOT ( n' t f2 ) ( Is the remainder <> 0? )
    OVER POP < ( n' t f2 f3 ) ( Is divisor < quotient? )
    AND ( n' t f4 ) ( If both conditions met )
    WHILE ( n' t ) ( Continue the loop )
      2 + ( n' t' ) ( Increase the divisor )
    REPEAT ( n' t' ) ( Loop again )
    U/MOD ( r3 q3 ) ( Loop is over, test again )
    DROP ( r3 ) ( Do not need the quotient )
    0= NOT ( f ) ( Is the remainder <> 0? )
  THEN ( f ) ( Endif )
;

: .PRIME ( n --- )
DUP PRIME ( n f ) ( Test if n is a prime number )
SWAP ( f n ) ( Get the number on Top )
CR ( f n ) ( Print a new line )
. ( f ) ( Print the number )
." is " ( f ) ( Print "is" )
NOT ( f' ) ( Is n a prime number? )
IF ( - ) ( No, it is not )
  ." not " ( - ) ( Print "not" )
THEN ( - ) ( Endif )
." a prime number." ( - ) ( Print the ending text )
CR ( - ) ( Another new line )
;

```

Fig. 10 : A Sample FORTH Program

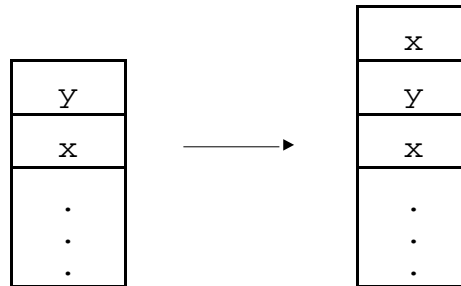
4.2 Features of FORTH

The most noticeable features of FORTH are its uses of two stacks and Reverse Polish Notation. Stacks are the necessary tools for any compiler, but only FORTH makes them available to the users. In FORTH, there are two stacks, a data stack and a return stack. The data stack is used to hold all the parameters needed for a calculation. It is the means of communication between functions. It can also be used to hold temporary data such as the intermediate values created by a calculation to be used in a later calculation.

Since the stack is visible to the users, in order to show the function of a FORTH word, we can use a *stack diagram* to show the net effect of a word on the stack. The several items near the top of the stack before and after the execution of a word is shown. All other items not shown are assume unchanged during the execution. The stack diagram shows two snapshots of the stack, separated by "---", the snapshot on the left represents the stack before the execution, the snapshot on the right represents the stack after the execution. In both snapshots, the rightmost element is at the top of stack. For example, the FORTH word "OVER" which copies the second item on the stack to the top of the stack has the following stack diagram :

(x y --- x y x)

And by executing the word "OVER", the stack is changed as the following diagram :



The return stack is used to hold the returning address of subroutines (words in FORTH terminology). When the operation of a word is completed, it will pop the return stack to obtain the return address. It can also be used to hold temporary values, but the rules for using the return stack to hold temporary values are much more restricted than the rules for data stack usage.

Reverse Polish Notation (RPN) is a natural result of the visibility of the data stack. It is also known as postfix operation. In RPN, we put the operations after the parameters of the operations. For example, if we want to carry out the addition of A and B, instead of saying "A + B", we will say "A B +". The words "A" and "B" will put the value of A and B on the data stack, and the operation "+", will take

two parameters from the data stack (i.e., A and B), add them together, and put the result back on the data stack.

Another feature of Forth is its compact size and fast execution speed. FORTH programs are written as sequences of function calls or words. Each word is a function which, in turn, may be defined as a sequence of words. Internally, a FORTH program is a sequence of subroutine addresses, so that a FORTH program is very compact, and its execution speed is almost the same as that of an assembly language program (but remember, FORTH is a higher level language). This feature makes FORTH especially suitable for use in embedded systems, such as control programs inside a washing machine or a microwave oven.

FORTH uses a unique I/O mechanism to access the storage system. The storage (diskette or tape) is divided into blocks or screens. Each block is 1024 bytes in size and represents a 40X25 or 64X16 screen. The blocks are numbered from 0. The information stored in the blocks can be loaded and executed by the FORTH word LOAD. Information stored in blocks can be modified by a block editor which usually comes with the FORTH system. The change to the block will be saved automatically. Therefore, the details about the filing system of the operating system are transparent to the programmer.

The FORTH words are defined and stored in memory space called a dictionary. The dictionary is structured as a linked list. Search starts from the most recently defined word and goes backwards. The dictionary is divided into vocabularies, words with the same name can have different meanings in different vocabularies. Search of words will start from the current vocabularies, and depending on the nature of the vocabularies, may or may not span to others vocabularies.

There are two different states during a FORTH session: interpretation state and compilation state. In interpretation state, the interpreter will get a word from FORTH's terminal input buffer (TIB), and search for the word in the dictionary. Once found, the word will be executed immediately. After execution, control will pass back to the interpreter, and the cycle starts again. In compilation state, the compiler will get a word from TIB, and search for the word in the dictionary. Once found, the word will be examined to determine whether it is an *immediate* word or not. If it is, the word will be executed immediately, else the address found will be compiled into the dictionary. The control is then passed back to the compiler and the cycle starts again. The immediate word is a very important concept in FORTH's execution. As one may notice, there are some words that must be immediate (e.g. ";" and "[" both signal the end of compilation) or the compiler will never stop.

FORTH is also an extensible system. Users can modify FORTH to suit their unique requirement. The modification is usually done through meta-compilation, a process to create a new FORTH system from an existing one. Most of the available FORTH system comes with a meta-compiler so the users can customise the system to their own needs. The resulting program can be easily saved and become another FORTH system.

4.3 Versions of FORTH and Standardization

In 1979, for the purpose of compatibility, the FORTH standards Team from EFUG released a formal specification describing a set of FORTH words that were to be included in every FORTH system. This became known as the FORTH-79 standard.

The FORTH-79 standard was revised in 1983, and become known as FORTH-83. Unfortunately, neither of these two standards is an official specification, so various dialects of FORTH, like dialects of BASIC, almost always tend to be incompatible with each other.

Recently, the American National Standard Institution has released an ANSI FORTH standard. Since it is a national standard, it is expected to be accept by most FORTH system developers.

There are many different FORTH systems available in the FORTH world. Most of them are public domain software or shareware. Examples of these free (or almost free) systems include: eFORTH, F83 Model, F-PC, FIG FORTH, Pygmy FORTH, and Zen FORTH.

Among these different products, Mr. Frank. C. Sergeant's Pygmy FORTH is very close to the draft proposal of ANSI FORTH. Therefore, it is logical to choose it as the base platform of this project. This choice has a side benefit: it includes an ASCII file loading facility that can be used directly. Some other features of Pygmy FORTH will be discussed in the next section.

WPFORTH is built based on Pygmy FORTH. Therefore, all features appeared in Pygmy FORTH are also available in WPFORTH. In addition, WPFORTH fixes some minor bugs in Pygmy FORTH. WPFORTH also has the capability to load a FORTH program written in WordPerfect format, which is the main concern of this project.

4.4 Overview of Pygmy FORTH

Pygmy FORTH is based on Mr. Moore's cmFORTH for the NOVIX Forth chip. cmFORTH was designed to run on a NOVIX computer connected by a serial line to a host terminal or computer that supplies editing and file storage services. Therefore, cmFORTH did not include an editor. Also, no assembler was needed because the NOVIX's assembly language is (more or less) Forth. [MIL 87] [ROS 87] [SER 89]

Pygmy FORTH is a superior FORTH system. It includes an editor and assembler and still only takes up about 16K bytes. The kernel (without editor and assembler) is less than 8K. Actually, one can have quite a lot of control on just how big it is because one can customize the system as needed. It comes with complete source code, including the meta-compiler, so Pygmy FORTH can recompile itself. The meta-compiler can also be used for target compiling custom applications. In this case one can eliminate the parts (such as the editor, assembler and various utilities) that the final application will not need. And one can also make words headerless to reduce the size or transparency of the final applications.

Pygmy FORTH is direct threaded (i.e., most of the codes are stored "in-line" rather than by jumping to a central routine) with top of stack kept in a register. It has a

comfortable screen oriented block editor. One can move quickly from block to block with the [PgDn] & [PgUp] keys, search across blocks, insert blank blocks, and compress out blank blocks, and switch between related blocks (for documentation or for comparing different versions of an application).

Pygmy FORTH also has an ASCII file loading facility which can allow Pygmy FORTH to load a FORTH program from an ordinary ASCII file. It is this feature and its closeness to the draft proposed ANSI FORTH standard that made me choose it as the base platform for WPFORTH.

V. WORDPERFECT DOCUMENT

5.1 General Description

There are several major versions of WordPerfect available on the market: Version 5.0, 5.1 and 6.0 for DOS, Version 5.1, 5.2 and 6.0 for Windows. Basically, all the 5.x versions are compatible with each other. Version 6.0, however, is a major revision of the product. Although WordPerfect 6.0 can read and save documents in WordPerfect 5.x format, WordPerfect 5.x cannot handle documents saved in WordPerfect 6.0 format. A detailed discussion about the differences between WordPerfect 5.x and 6.0 is presented in a later section. WPFORTH will only handle documents saved in WordPerfect 5.x format. Therefore, unless stated, a WordPerfect document here refers to a WordPerfect 5.x document.

A WordPerfect document is a binary file that contains both ASCII characters and special formatting codes. Each document has two major sections, a prefix area and a document area [WOR 93]. Figure 11 is an illustration of the WordPerfect file structure.

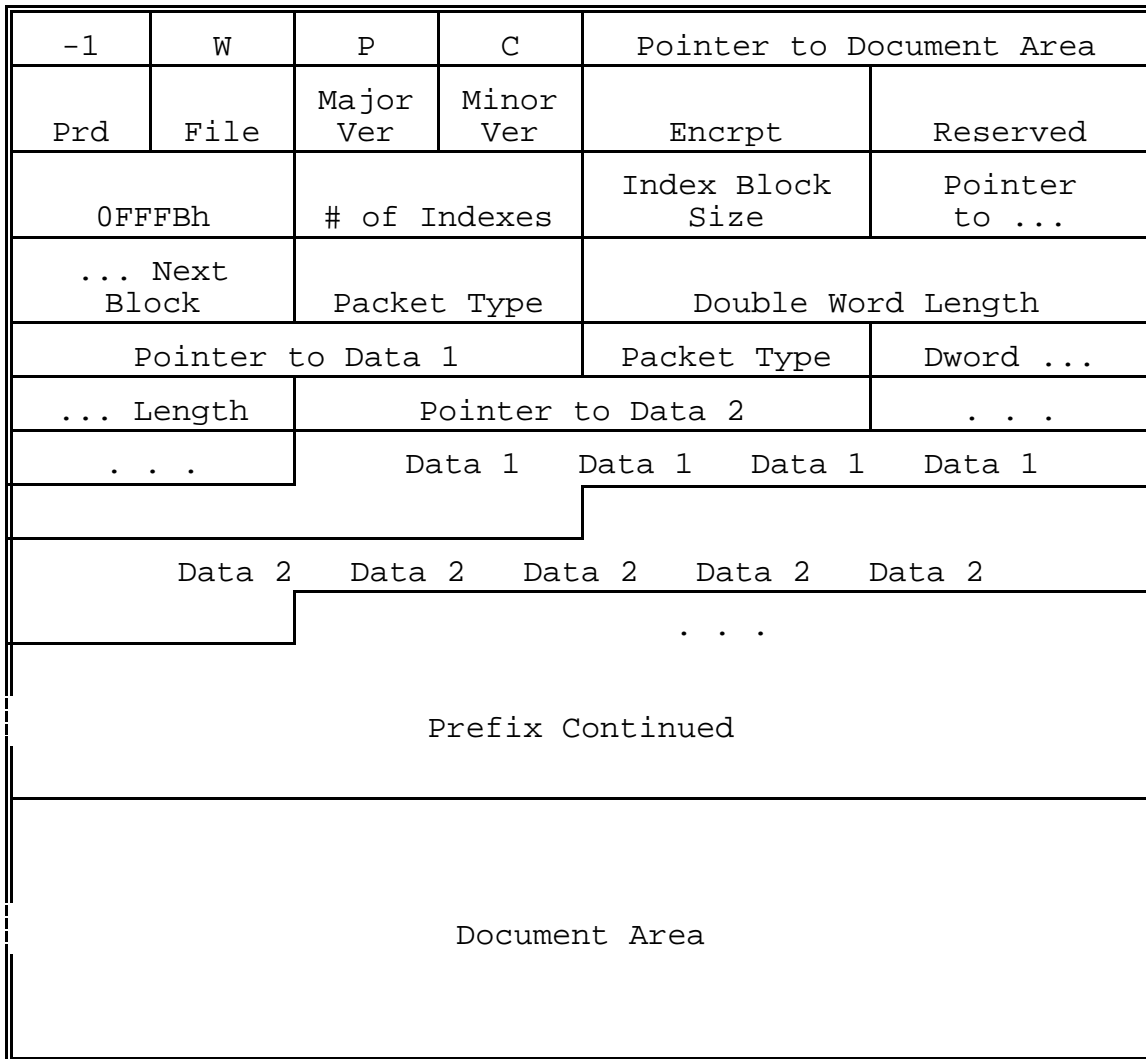


Fig. 11 : Structure of a WordPerfect Document

The prefix area of a WordPerfect document provides information about the document. It consists of a 16-byte file prefix, which is followed by additional prefix information that is specific to the document. The length of the prefix is variable, depending on the WordPerfect document and file prefix contents.

The 16-byte file prefix identifies the file type, product and version, etc. The contents of the file prefix are listed in table 1.

Table 1 : Structure of File Prefix in a WordPerfect Document

<u>Offset</u>	<u>Field</u>	<u>Byte</u>	<u>Value</u>
0	WPCorp File ID	4	0FFH, "WPC"
4	Start of Document	4	Start of document
8	Product Type	1	1h = WP file
9	File Type	1	0Ah = WP Document
10	Major Version	1	0h = Version 5.x
11	Minor Version	1	0h=V5.0 1h=V5.1
12	Encryption Key	2	0h = not encrypted
14	Reserved	2	0h

5.2 Prefix Area

The rest of the prefix area is divided into packets. There are two major kinds of packets: Special Data Packets and General Packets.

The first two bytes in a packet is a packet type which identifies the packet. Table 2 lists the packet types that WordPerfect reserved for use as special packet. All other values are used as general packets.

Table 2 : WordPerfect Special Packets

0000h	= End of Prefix
FFFBh	= Header Index for index block
FFFC - FFFEh	= Reserved
FFFFh	= Deleted Packet

One of the special packet is the index packet (packet type FFFBh). There are always five indexes in an index packet. Since it requires 10 bytes for each index, the length of an index packet is always 50 bytes for five packet indexes. Each index packet contains a special packet index plus four general packet indexes. Table 3 and Table 4 list the structure of an index packet. Table 5 lists the general packets that are defined in WordPerfect.

Table 3 : Structure of a Special Packet Index

<u>Offset</u>	<u>Bytes</u>	<u>Usage</u>
0	2	Packet Type (FFFBh)
2	2	# of indexes (5)
4	2	Size of Block (32h)
6	4	Pointer to next index block

Table 4 : Structure of a General Packet Index

<u>Offset</u>	<u>Bytes</u>	<u>Usage</u>
0	2	Packet Type
2	4	Length of data packet
6	4	Pointer of data packet

A detailed discussion of the packets relevant to the project will be presented in later sections.

Table 5 : General Packets Defined in WordPerfect

1h	Document summary packet
2h	List of fonts used in document (WP5.0)
3h	Document initial codes
4h - 5h	Reserved
6h	Document specific flags
7h	Font name string pool
8h	Graphics information
9h	Form hash table
0Ah - 0Bh	Reserved
0Ch	Document printer information
0Dh	Reserved
0Eh	Supplemental dictionary compressed words
0Fh	List of fonts used in document (WP5.1)
10h	DDE link packet (WPWin5.1)
11h	Macro executable code (WPWin5.1)
12h	Reserved
13h	Macro information block (WPWin 5.1)
100h - 1FFh	Style packets
200h - 2FFh	PS table packets

5.3 Document Area

The document area follows the prefix area and contains two kinds of codes: single-byte and multi-byte codes.

Single-byte codes include control characters (01h-1Fh), ASCII characters (20h-7Eh), and single byte functions (80h-BFh). The value 00h and 7Fh can never appear in a WordPerfect document as a single-byte code (but they may appear within a multi-byte code, if necessary). Table 6-8 list the single-byte codes defined in WordPerfect.

Table 6 : Control Characters Used by WordPerfect

<u>Code</u>	<u>Character</u>	<u>Meaning</u>	<u>Code</u>	<u>Character</u>	<u>Meaning</u>
01h	Ctrl-A	Reserved	10h	Ctrl-P	Merge codes P
02h	Ctrl-B	Page number	11h	Ctrl-Q	Merge codes Q
03h	Ctrl-C	Merge codes C	12h	Ctrl-R	Merge codes R
04h	Ctrl-D	Merge codes D	13h	Ctrl-S	Merge codes S
05h	Ctrl-E	Merge codes E	14h	Ctrl-T	Merge codes T
06h	Ctrl-F	Merge codes F	15h	Ctrl-U	Merge codes U
07h	Ctrl-G	Merge codes G	16h	Ctrl-V	Merge codes V
08h	Ctrl-H	Reserved	17h	Ctrl-W	Reserved
09h	Ctrl-I	Reserved	18h	Ctrl-X	Reserved
0Ah	Ctrl-J	Hard return	19h	Ctrl-Y	Reserved
0Bh	Ctrl-K	Soft page break	1Ah	Ctrl-Z	Reserved
0Ch	Ctrl-L	Hard page break	1Bh	Ctrl-[Reserved
0Dh	Ctrl-M	Soft return	1Ch	Ctrl-\	Reserved
0Eh	Ctrl-N	Merge codes N	1Dh	Ctrl-]	Reserved
0Fh	Ctrl-O	Merge codes O	1Eh	Ctrl-^	Reserved
			1Fh	Ctrl-_	Reserved

Table 7 : ASCII Characters Used by WordPerfect

<u>Value</u>	<u>Character</u>	<u>Value</u>	<u>Character</u>	<u>Value</u>	<u>Character</u>
20h	Space	40h	@	60h	`
21h	!	41h	A	61h	a
22h	"	42h	B	62h	b
23h	#	43h	C	63h	c
24h	\$	44h	D	64h	d
25h	%	45h	E	65h	e
26h	&	46h	F	66h	f
27h	'	47h	G	67h	g
28h	(48h	H	68h	h
29h)	49h	I	69h	i
2Ah	*	4Ah	J	6Ah	j
2Bh	+	4Bh	K	6Bh	k
2Ch	,	4Ch	L	6Ch	l
2Dh	-	4Dh	M	6Dh	m
2Eh	.	4Eh	N	6Eh	n
2Fh	/	4Fh	O	6Fh	o
30h	0	50h	P	70h	p
31h	1	51h	Q	71h	q
32h	2	52h	R	72h	r
33h	3	53h	S	73h	s
34h	4	54h	T	74h	t
35h	5	55h	U	75h	u
36h	6	56h	V	76h	v
37h	7	57h	W	77h	w
38h	8	58h	X	78h	x
39h	9	59h	Y	79h	y
3Ah	:	5Ah	Z	7Ah	z
3Bh	;	5Bh	[7Bh	{
3Ch	<	5Ch	\	7Ch	
3Dh	=	5Dh]	7Dh	}
3Eh	>	5Eh	^	7Eh	~
3Fh	?	5Fh	_		

Table 8 : Single-Byte Functions Defined in WordPerfect

<u>Code</u>	<u>Meaning</u>	<u>Code</u>	<u>Meaning</u>
80h	Temporary (always deleted)	A0h	Hard space
81h	Right justification ON	A1h	Do subtotal
82h	Right justification OFF	A2h	Subtotal entry
83h	End center/align	A3h	Do total
84h	Reserved	A4h	Total entry
85h	Temporary (place saver)	A5h	Do grand total
86h	Center page top to bottom	A6h	Calculation column
87h	Columns ON	A7h	Math ON
88h	Columns OFF (at top of page)	A8h	Math OFF
89h	Reserved	A9h	Hard hyphen in line
8Ah	Widow/Orphan ON	AAh	Hard hyphen at end of line
8Bh	Widow/Orphan OFF	ABh	Hard hyphen at end of page
8Ch	Hard return/soft page	ACH	Soft hyphen in line
8Dh	Footnote/Endnote number	ADh	Soft hyphen at end of line
8Eh	Figure number (inside caption)	AEh	Soft hyphen at end of page
8Fh	Hard end of center/align	AFh	Columns OFF at end of line
90h	Deletable return at EOL	B0h	Columns OFF at end of page
91h	Deletable return at EOP	B1h	Math negate
92h	Deleted end of page	B2h	Outline OFF
93h	Invisible return in line	B3h	Reserved
94h	Invisible return at EOL	B4h	Reserved
95h	Invisible return at EOP	B5h	Reserved
96h	Block ON	B6h	Reserved
97h	Block OFF	B7h	Reserved
98h	Table of contents page #	B8h	Reserved
99h	Dormant hard return	B9h	Reserved
9Ah	Cancel hyphenation	BAh	Reserved
9Bh	End of generated text	BBh	Reserved
9Ch	Reserved	BCh	Reserved
9Dh	Reserved	BDh	Reserved
9Eh	Hyphenation OFF	BEh	Reserved
9Fh	Hyphenation ON	BFh	Reserved (unknown)

Multi-byte codes include fixed- and variable-length functions. These functions consist of a group of bytes instead of a single byte and always begin and end with a function code (C0h-FFh) that indicates the type of function. Table 9-10 list the multi-byte codes defined in WordPerfect.

Table 9 : Fixed-Length, Multi-byte Functions Defined in WordPerfect

<u>Code</u>	<u>Bytes</u>	<u>Meaning</u>
C0h	4	Extended Character
C1h	9	Center/Align/Tab/Left Margin Release
C2h	11	Indent
C3h	3	Attribute ON
C4h	3	Attribute OFF
C5h	5	Block Protect
C6h	6	End of Indent
C7h	7	Different Display Character when Hyphenated
C8h-CFh		Reserved

Table 10 : Variable-Length, Multi-byte Functions Defined in WordPerfect

<u>Code</u>	<u>Meaning</u>
D0h	Page Format Group
D1h	Font Group
D2h	Definition Group
D3h	Set Group
D4h	Format Group
D5h	Header/Footer Group
D6h	Footnote/Endnote Group
D7h	Generate Group
D8h	Display Group
D9h	Miscellaneous Group
DAh	Box Group
DBh	Style Group
DCh	Table End of Line Group
DDh	Table End of Page Group
DEh	Enhanced Merged Command Codes Group
DFh	Equation Nested Function Group
E0h-FDh	Reserved
FEh	Reserved (Unknown)
FFh	Reserved

As implied by its name, the fixed-length functions have a predetermined length and take the following structure:

```

Begin function code (C0h-CFh)
data bytes ...
End function code (same as begin function code)

```

Therefore, the smallest fixed-length, multi-byte function is at least 3 bytes long which consists of the function code, a single data byte and the function code again.

Variable-length, multi-byte functions have a much more complex structure and each function is actually a group of similar functions, identified by the subfunction code. The following is the structure of a variable-length function:

```

Begin function code (byte) (D0h-DFh)
Begin subfunction code (byte)
Length (word) (= total length - 4)
Data bytes ...
Length (word) (= total length - 4)
End subfunction code (= begin subfunction code)
End function code (= begin function code)

```

Again, only functions which are relevant to the project will be discussed in later sections.

5.4 Font Information

WordPerfect puts the font information in two kinds of packets: packet 07h stores all the font name strings, where each string ends with a 00h. All other information is stored in another packet, packet 02h in WordPerfect 5.0, and packet 0Fh in WordPerfect 5.1. The structures of the two packets are slightly different, but the lengths of both are the same: 86 bytes for each font.

The font number starts from 0 (this is the initial font used in WordPerfect) and followed by font 1, font 2, etc. There is a pointer (offset 18) which points to name of the font in the font-name string pool (packet 07h). The value of the pointer is the offset from beginning of the font-name string pool.

Also, the font size is contained in the packet. It is offset 28 in packet 02h, and offset 25 in packet 0Fh. Font size is measured in units of 1/1200 inch. To convert this value into point size, we just have to divide it by 10.

Other information is stored in packet 02h and packet 0Fh, but this is not needed for this project, so we can just ignore these items.

5.5 WordPerfect Version 6.0

Recently, WordPerfect Corporation has released a newer version, WordPerfect for DOS 6.0 and WordPerfect for Windows 6.0. Both versions have new capabilities, but they are backwards compatible (they can read and save files in WordPerfect 5.x format).

The structure of a WordPerfect 6.0 document is very different from that of a WordPerfect 5.x document. Most of the functions have different meanings in WP5.x and WP6.0; even the structure of the functions is changed. The following are some examples of the differences:

- * The file header can be longer than 16 bytes (it is always 16 bytes in WordPerfect 5.x)
- * In WordPerfect 6.0, the variable-length multi-byte function is divided into two parts: a non-deletable portion and a deletable portion. The non-deletable portion of a function code is the documented part of the function. The deletable part of the function code follows the non-deletable function code data and is not documented.
- * The function size in WordPerfect 6.0 is the length in bytes of the entire function (it is the function size minus 4 in WordPerfect 5.x).

- * The end subfunction code for a variable-length multi-byte function is eliminated in WordPerfect 6.0 (in WordPerfect 5.x, the end subfunction code must be there). Therefore, a variable-length, multi-byte function in a WordPerfect 6.0 document may look like:

```

<function>      byte
<subfunction>  byte
[size]          word
non-deletable function data ...
deletable function data ...
[size]          word
<function>      byte

```

These are only some of the major differences between these two versions. Therefore, it will not be appropriate to add the capacity of loading a WordPerfect 6.0 document in the prototype implemented in this project. However, it can be considered as a future improvement.

VI. WPFORTH IMPLEMENTATION DETAILS

6.1 General Overview

Although Pygmy FORTH is very suitable for this project, substantial changes have to be made to it before the actual implementation can be made. The first thing is to strengthen its file I/O interface. Like all other FORTH system, the usual method to load a program into Pygmy FORTH is through the block loading facilities. The ASCII file I/O interface, although presented, is not enough for our purpose. Pygmy FORTH also assumes that lines in an ASCII file will not exceed 132 characters in length, so it checks ASCII file input to exclude files with lines longer than 132 characters. This is a reasonable assumption for normal pure ASCII file, but in the case of a WordPerfect document, it becomes unacceptable. The prefix area may span a couple thousands bytes without a single carriage return, the document area can also exceed the limit of 132 when considering "hidden" function codes. The reason for this limitation is that Pygmy FORTH can read the whole line from the input buffer at one time. Since this limitation is unacceptable in WPFORTH, we can only let WPFORTH read from the file one byte at a time. The approach is to ask

the (DOS) system to read a byte into the buffer, then WPFORTH will take the byte from the buffer, carry out any pre-process if necessary, and put it into the WPFORTH's own input buffer.

The implementation of WPFORTH takes advantage of FORTH's top down, modular programming strategy. Many duplicated functions have been factored out to single FORTH words to make the system easy to understand and debug. For example, since many of the values inside WordPerfect packets and functions take the form of a word (two bytes, or a single number in FORTH) in the order of <ll><hh> in the file, we can write a function (FORTH word) READ-WORD to handle this.

The definitions of all the words defined (or redefined) in WPFORTH are grouped into several files. They are linked together with FORTH's INCLUDE facility. This can keep the length of each file short enough to be handled easily, and also make it easy to locate a specific definition. For example, the file WPFORTH.PFX contains all the definitions to handle the WordPerfect prefix area.

To make things as flexible as possible, a table look up technique is used to handle all the WordPerfect functions and packets. This method is used in the implementation of Henry Laxen and Mike Perry's F83 model and is borrowed here [PAS 87]. With this method, a table containing all possible

values of WordPerfect functions is set up. Each element in the table contains a 2-byte address to another FORTH word. If a specific function is to be ignored by WPFORTH, it can easily be done by pointing the function to NOP (No Operation). It will be equally easy to add or change the definition of any of the WordPerfect control codes, functions, and packets.

Since most of the WordPerfect Functions are intended to be ignored by WPFORTH, we should first define a couple of words to handle this kind of functions and codes. They are (SINGLE, (FIX and (VARIABLE. As the name implied, (SINGLE is used to skip a single byte function (01h - 1Fh, and 80h - BFh). (FIX is

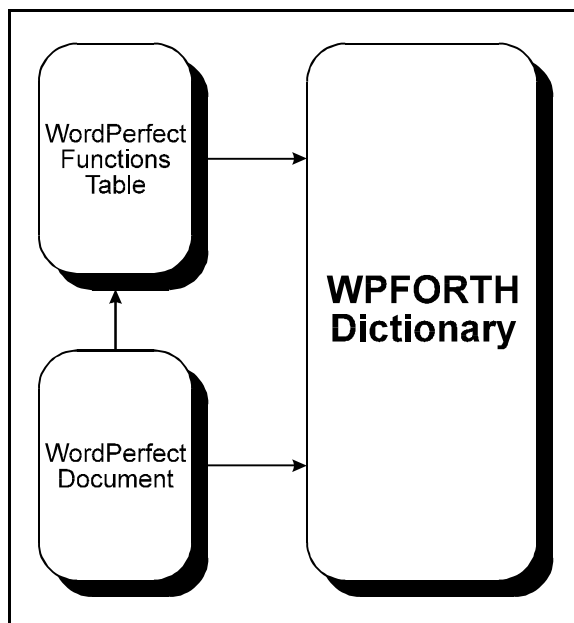


Fig. 12 : Handling of WordPerfect Functions

used to skip a fixed-length, multi-byte function (C0h - CFh); it expects a parameter on the data stack, and will skip bytes according to this parameter. (VARIABLE is used to skip a variable-length, multi-byte function (D0h - FFh); it will read the length from the function itself and, therefore, does not expect anything on the data stack. All these functions start

with a bracket because they are not intended to be used by normal users. The handling of WordPerfect functions through a function table is illustrated in Figure 12. Table 11 is a list of the WordPerfect control codes and functions considered in this project.

Table 11 : WordPerfect Functions Considered in WPFORTH

<u>Function Code</u>	<u>Meanings</u>
00h	Not used in WordPerfect (translated into NOP for safety)
0Ah - 0Ch	Soft/Hard Line/Page Break (Converted into ASCII CR/LF)
1Ah	Not used in WordPerfect but used as EOF for ASCII (translated into NOP for safety)
8Ch	WordPerfect [HRT-SPg] code (translated into ASCII CR/LF)
A9h - AEh	Various kinds of hyphens used in WordPerfect (all will be translated into ASCII "-")
C1h	TAB used in WordPerfect (translated into a space)
C4h - C5h	Attribute On/Off, more details later
D1h	Font change, more details later

In general, all other functions will be translated into a space – the official delimiter used in FORTH.

Because of this translation, we need three levels of input routines to read a byte from a file. The lowest-level **read_word** reads the raw data directly from the file. It will return whatever it got from the file, and pass it to the middle-level **read_word**. The middle-level **read_word** will carry out all necessary translation and put the translated string onto the input stream. FORTH's highest-level **read_word** will read the input from the input stream, without any knowledge about the characters that have been discarded by the middle-level **read_word**.

In order to implement such a multi-level read words, several Pygmy FORTH's file-I/O words have to be modified, including READ-LINE (read a line from a file), and ?REFILL (refill the TIB, terminal input buffer, when necessary).

6.2 Interpretation of Attributes

WordPerfect's functions C3h and C4h are two multi-byte functions both with a fixed length of 3 bytes; they are used to turn on or turn off some attributes, C3h is "Attribute ON" function, and C4h is "Attribute OFF" function. Both functions have similar structures as shown in Table 12.

Table 12 : Structure of WP's Attribute ON/OFF Functions

<u>Offset</u>	<u>Byte</u>	<u>Meaning</u>
0	1	Begin function code (C3h or C4h)
1	1	Attribute to be turned on or off
2	1	End function code (C3h or C4h)

The value in the second byte of the function determines which attribute to be turned on or turned off. There are 16 different attributes used in WordPerfect, as shown in Table 13.

Table 13 : WordPerfect's Attribute Codes

0 = Extra Large	1 = Very Large
2 = Large	3 = Small
4 = Fine	5 = Superscript
6 = Subscript	7 = Outline
8 = Italics	9 = Shadow
A = Redline	B = Double Underline
C = Bold	D = Strikeout
E = Underline	F = Small Caps

In WPFORTH, the attribute value is stored in a single numeric variable, each bit of which represents one of these attributes. If the bit is on (value 1), the corresponding attribute is on. If it is off (value 0), the attribute is off. For example, if the attribute variable has a value of 0, it means no attribute is turned on. On the contrary, if it has the value -1 (or 65535), all the attributes are turned on. If it has the value 20516 (0101 0000 0010 0100b), that means attribute 2 (Large), attribute 5 (Superscript), attribute C (Bold) and attribute E (Underline) are turned on, all others are turned off.

However, we cannot turn the attributes on or off immediately when we read the function code from the file. Instead, a flag must be put into the input stream, to ask the FORTH interpreter and/or compiler to turn on/off the corresponding attribute (store an appropriate value into the attribute variable) at a suitable time. Therefore, the

middle-level **read_word** will put words ATTR-ON and/or ATTR-OFF into the input stream, and these words will turn on/off the corresponding attribute when they are executed.

There is a problem when a change of attribute is encountered during compilation, namely, when the attribute change has to be compiled into the dictionary. At the same time the attribute change should also be carried out immediately to reflect the actual change in the attribute. We can force the words ATTR-ON and ATTR-OFF in the COMPILER vocabulary to compile their run time address into the dictionary, and also execute them immediately. However, the number parameters of ATTR-ON and ATTR-OFF are already compiled into the dictionary and not available for execution. Of course, we can get them back again by reading the value from the dictionary. This method works in most cases, but will fail if the number is 0, 1, 2 or other numeric value which has been redefined as an executable word. For example, if we have defined :

```
: 1 1 ;
```

Then we can only get the address of the executable word. What is worse, we have no method to determine whether we have obtained the value itself or only an address through which we can get the value.

Another problem will arise. If the programmer changes an attribute just after the colon ":" (or after other defining words), WPFORTH will be confused. For example, if the programmer puts the following line in his file:

```
: UNDERLINE 1 1 + . ;
```

WPFORTH will see from its input stream:

```
: 16384 ATTR-ON UNDERLINE 16384 ATTR-OFF 1 1 + . ;
```

therefore, instead of compiling the definition of UNDERLINE, it will redefine the number 16384.

Obviously this is not acceptable. To overcome this problem, WPFORTH plays a small trick. Instead of putting "16384 ATTR-ON" in the input stream, the middle-level read word will put the following in the input buffer :

```
wpf 16384 ATTR-ON
```

"wpf" is a FORTH word which will fetch a number and a FORTH word from the input buffer, execute the FORTH word with the number as its parameter. In the above example, "wpf" will fetch the number 16384 and the FORTH word "ATTR-ON", and execute the word "ATTR-ON" with parameter 16384. The

definition of (HEAD is also modified so that whenever it sees "w`pf`" after a defining word, it will complain and stop. The use of "w`pf`" can also ensure the number after it will be compiled as a numeric literal, because it is always expecting a number from the input buffer, and explicitly converts it into a numeric literal. Therefore, it eliminates the confusion caused by the difference between numeric literal and executable word.

Although the use of "w`pf`" breaks the rule that "everything in FORTH is redefinable", this is worthwhile because it can avoid many problems. In fact the choice of "w`pf`" should be safe enough since Pygmy FORTH (and hence WPFORTH) is case sensitive, while programs are usually written in upper case. If WPFORTH is to be implemented in a case insensitive system (such as F83), other words with rarely used symbols can be chosen (e.g., ">w`pf`<").

6.3 Consideration of Font Attributes

WordPerfect function D1h, subfunction 01h is used for changing fonts. Its structure is listed in Table 14.

Table 14 : Structure of WordPerfect's Font Change Function

<u>Offset</u>	<u>Size</u>	<u>Meaning</u>
0	1 byte	Begin function code (D1h)
1	1 byte	Begin subfunction code (01h)
2	1 word	Length word (23h)
4	1 byte	Old font #
5	24 bytes	Desired font description
29	1 byte	Matched font # (0=default)
30	1 word	Matched font hash value
32	1 word	Point size
34	1 byte	Typeface flags
35	1 word	Length word (23h)
37	1 byte	End subfunction code (01h)
38	1 byte	End function code (D1h)

All of the information above is also included in the font packet (02h for WordPerfect 5.0, and 0Fh for WordPerfect 5.1). Therefore, the only thing we need is the new font number (or matched font number in WordPerfect's terminology) which is at offset 29 in the function. Other information can be obtained by referring to the font table built when processing the font packet (see discussion below for more details).

Similar to the situation of attributes, if the change of fonts appears immediately after a defining word, or the new font number is redefined as an executable word, the compiler will be confused. Therefore, we must use "w_{pf}" to ensure that the number after it is fetched as a numeric literal, and to ensure no font change immediately after a defining word.

6.4 Implementation of a Font Segment

The font information, such as the font name, the font size, etc., should be preserved for later use when the definitions defined in the WordPerfect document are executed. Since the amount of this information may be huge, it can overflow the relatively small dictionary space. To reduce the footprint of this font information in the dictionary, WPFORTH uses another memory segment to store it. The data structure used in the font segment is

similar to the structure used in the dictionary, i.e., a linked list. We can search through the list to locate the information we want for a particular font. Figure 13 illustrates this data structure.

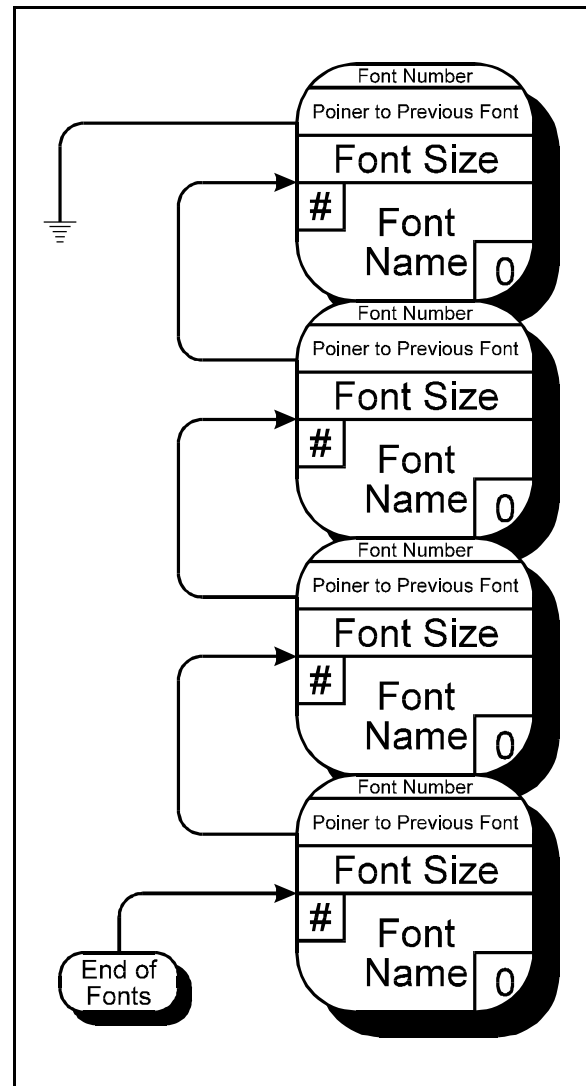


Fig. 13 : Data Strucutre Used in Font Segment

Several words are defined to help WPFORTH handle data in different segments. The most basic one is <SMOVE> which stands for inter-Segment MOVE. It is similar to the FORTH word CMOVE. However, the source and destination address both take the form of <segment> <offset>, so its stack diagram is (fseg fadd tseg tadd length ---).

The word DATA->FONT moves data from the code (dictionary) segment to the font segment, and the word FONT->DATA does the same thing but in opposite direction.

The word READ-FONT-INFORMATION accepts a font number and an address and will store the font name at that address, returning the size of the font on the data stack. If the font is not defined, -1 will be returned on the data stack. The word FONT-SIZE only return the font size on the data stack.

As mention earlier, the font size used by WordPerfect is measured in units of 1/1200 inch. The word FONT-SIZE-IN-POINTS will do the necessary conversion. It will return the point size of the font. Again, if the font is not defined, -1 will be returned on the data stack.

6.5 Implementation of a File Segment

In the original Pygmy FORTH, information about file loading is preserved on the return stack so that we can nest file loading operations (load another file from within a file). This is acceptable because not too many items have to be preserved. In WPFORTH, more information needs to be preserved, e.g., whether the file is a WordPerfect document, the current attribute, the current font, the file position, etc. It will not be wise to store these information on the return stack.

There is another problem with the original Pygmy FORTH, namely, if an error occurs during a file load, only the current file will be closed, all other previously opened files will remain open. The reason is that the return stack is cleared after an error occurs. Even if it is not cleared, there will be no way to tell which items on the return stack are opened file handles. This will not cause damage because Pygmy FORTH will not write anything to these files, so nothing will be overwritten. However, the used DOS file handles will become unreusable. Eventually, all available file handles will be used up and no more files can be loaded unless we restart Pygmy FORTH.

WPFORTH uses another approach. It sets up a file segment for storing the information necessary for loading files. With the use of this file segment, everything needed to be preserved when loading a file can be kept together. Also, the definition of ABORT has been modified so that all opened files are closed automatically whenever an error occurs.

Since the information regarding a file is fixed (it is 22 bytes or 11 single numbers in length), we can use an array as the data structure for the file segment. With the help of a variable CUR-FILE-POINTER, which tells us how many files are open at any time, we can calculate the address of the information we need to access. Figure 14 illustrates this data structure.

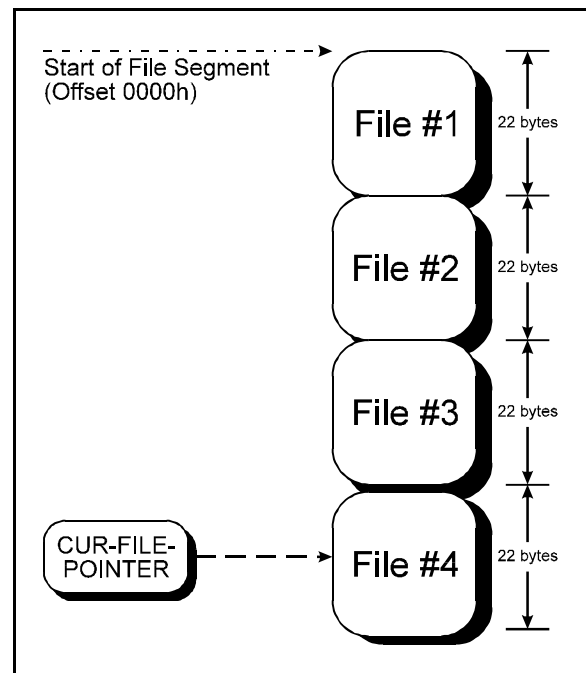


Fig. 14 : Data Structure Used in File Segment

6.6 Save the Result Image in .EXE Format

The original Pygmy FORTH saves its core image as a .COM file, but it is actually a multi-segment program: a dictionary (code) segment and a stack segment. This is workable because saving the image only requires the dictionary segment to be saved, the content of stack segment being discardable. When (a modified) Pygmy FORTH starts, it will always initialize the stack segment so that the data stack and the return stack are empty.

The introduction of the font segment and the file segment cause no trouble in the operation of WPFORTH. However, when programmers want to save a modified image of WPFORTH to disk, especially after loading a WordPerfect file, they must also preserve the contents of the font segment. The file segment is used as a file stack, so its content should be refreshed every time WPFORTH starts, and need not be preserved.

This means we have to save two segments in the image, but the .COM format only accepts one 64K-bytes-segment. Therefore, we have to change the image into an .EXE format which allows multiple segments to be saved.

When WPFORTH is first built, the segments are set up in memory in the following order: dictionary segment, stack segment, font segment, and file segment. In order to save the font information stored in the font segment, WPFORTH has to save the first three segments into its core image. As mentioned above, to save the stack segment is just a waste of disk storage. To reduce storage requirements, we change the load order to dictionary segment, font segment, stack segment, and file segment. Now only the first two segments have to be saved. Furthermore, we do not have to save the whole second segment. The result is an image that is between 64K bytes and 128K bytes in size.

Some extra work must be done to make the saved .EXE image executable. First, unlike a .COM file which always starts execution at 0100h, we must tell the .EXE file where to start. This is done by setting the address of the starting point in the .EXE header which is saved together with the .EXE image. Second, because the locations of the stack segment and font segment are exchanged, before saving the .EXE image, the content of some memory location must be temporarily patched so the saved image can know where to find the correct stack segment and font segment, and this patch must be restored after the save operation because we cannot assume the user will quit after saving the image. Figures 15 illustrates the segment relationship in Pygmy FORTH and WPFORTH.

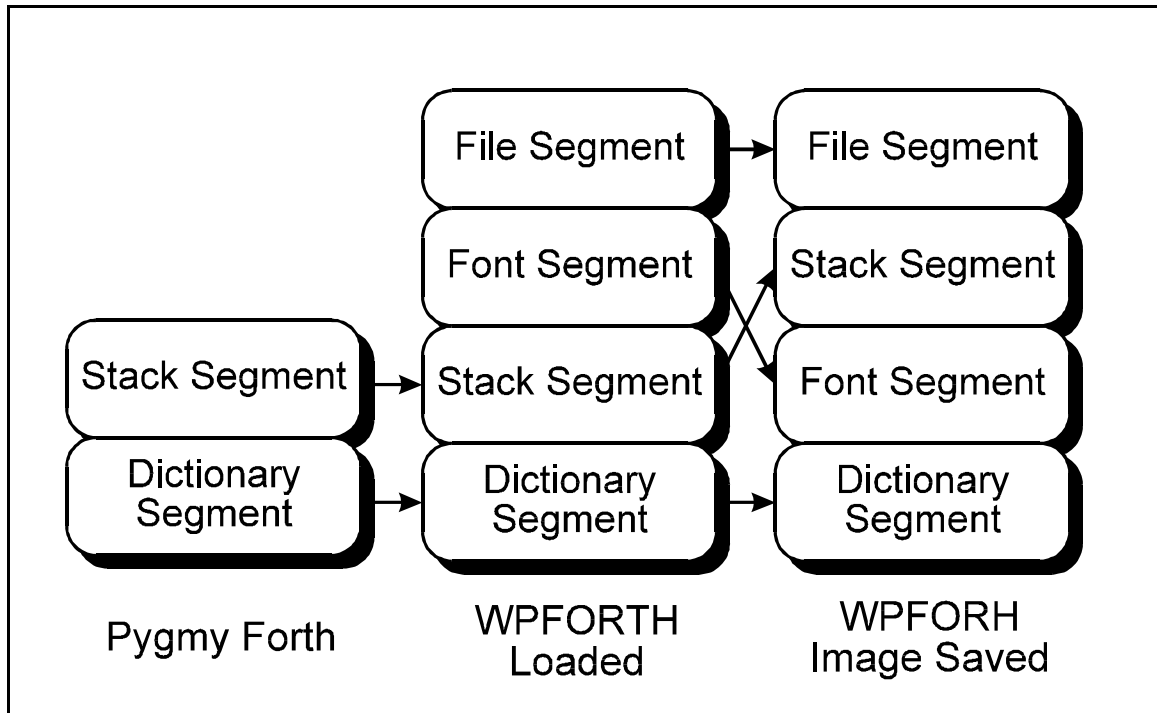


Fig. 15 : Segments Used in Original Pygmy FORTH

VII. MARKUP LANGUAGES AND SGML

7.1 What are Markup Languages

A "markup language" is such a scheme (or set of rules) to mark up programs. Unlike a high level programming language, it is actually a function or procedure that combines data and instructions in the same file. Its purpose is to markup text in order to identify special elements within the text or to specify particular operations [WU 89].

In order to exchange programs between different implementations, we need some scheme to "mark up" the programs. This will produce a version of a program in which only ASCII (or EBCDIC) characters are used, so that transfer from one platform to another can be easily achieved.

There are four kinds of markup languages: presentational markup, punctuational markup, procedural markup and descriptive markup.

We are always using presentational markup and punctuational markup when we write anything, even though we

may not be aware about that. We use different punctuation such as commas, brackets, quotation marks to separate words in our text. This is punctuational markup. We use spaces, cases, blank lines, indentations to separate the words and different parts of our text. This is presentational markup. The punctuational markup and presentational markup are very important to our communication. For example, consider the following two paragraphs:

```
"thestandardgeneralizedmarkuplanguageorsgmlasitismore  
epopularlyknownisbasedontheibmproductdocumentcompositionfaci  
litydcfgeneralizedmarkuplanguagegml"
```

"The Standard Generalized Markup Language, or SGML as it is more popularly known, is based on the IBM product Document Composition Facility (DCF) Generalized Markup Language (GML)."

These two paragraphs are identical except that the latter one has punctuation and case differences to clarify the written expression. Punctuational markup and presentational markup are so important and so common that most people are not aware of them as examples of markup.

The introduction of electronic text-processing systems also introduces the procedural markup. This consists

of commands indicating how text should be formatted. One example is the dot commands used in the WordStar documents: where ".pn" is used for printing page number and ".pa" to force a page break. Other markup systems include troff, T_EX and WordPerfect with its "reveal codes".

These three types of markups have many problems. Punctuational markup is complex and ambiguous. For example, some people put two spaces after a full stop, but some put only one. Also, the full stop sometimes means end of a sentence, but sometimes means abbreviation. Presentational markup and procedural markup are geared to a particular text formatter, so that we cannot change the formatter without explicitly translating all markup codes. They are also device dependent. If someone wants to change the printing format after the text is marked, the entire markup has to be changed. This is often a time-consuming, inefficient and expensive process.

Descriptive markup does not describe the final presentation of the text. It deals with the logical structure of the document. It just marks a portion of text as, for example, a paragraph instead of specifying an indentation of eight characters. The printing format can always be changed easily and more consistently.

7.2 What is SGML

The full name of SGML is Standard Generalized Markup Language. It was created by Charles Goldfarb, and has passed through an excruciatingly slow and painful process known as "standardization". In 1986, the specification for the language was published by the International Organization for Standardization (ISO) as ISO document 8879:1986, the full title is "Information processing — Standard Generalized Markup Language (SGML)". A few fixes to the specification were published in 1988, called Amendment 1, ISO 8879/A1:1988.

The prefix "Standard" implies that SGML is an ISO standard. The prefix "Generalized" indicates the same rigorous techniques are to be used to process documents. It is called a "Language" because it is a formalized description of a generalized markup. The word "Markup", of course, reminds us that it is a descriptive markup language [WU 89].

Under the recent CALS (the Computer-aided Acquisition & Logistics Support) initiative, SGML also became a military standard (MIL-M-28001). CALS is the American Department of Defence's initiative. One of the requirements of CALS is to ensure that any supporting document from a contractor or sub-contractor must be coded in SGML. This initiative greatly increases the importance of SGML.

An SGML document is logically divided into three areas: the SGML declaration, document type definition, and the text instance.

As a formal language, SGML requires a coherent and unambiguous syntax to ensure its consistent and correct usage. However, one of the SGML's objectives is flexibility. In order to achieve this objective, SGML uses an abstract syntax which permits users to define their own delimiters and character set to describe a document. All these definitions are declared in the SGML declaration.

From the point of view of SGML, a document is a hierarchical tree structure with logical elements at different levels. For example, a letter consists of the sender's information, receiver's information, date etc., and the sender's information itself consists of the sender's name and sender's address. On the other hand, the relationship of all these elements is fixed in a specified type of document, some of them must appear before others, some of them may appear in any order, some of them are optional, some of them can only appear once, some of them can appear as many times as necessary. All these relationships are defined in the document type definition (DTD).

The text instance of an SGML document is basically the text of the document with SGML tags. Since the document is treated as a hierarchical collection of elements, the text instance can also be viewed as a hierarchy of SGML elements enclosed by tags which are defined in a DTD.

Figure 16 shows the components of an SGML document.

Since SGML is an ISO standard as well as a military standard, it is most suitable for the use as a markup language for the purpose of interchange of programs. The following section describes the procedures necessary to implement such a system.

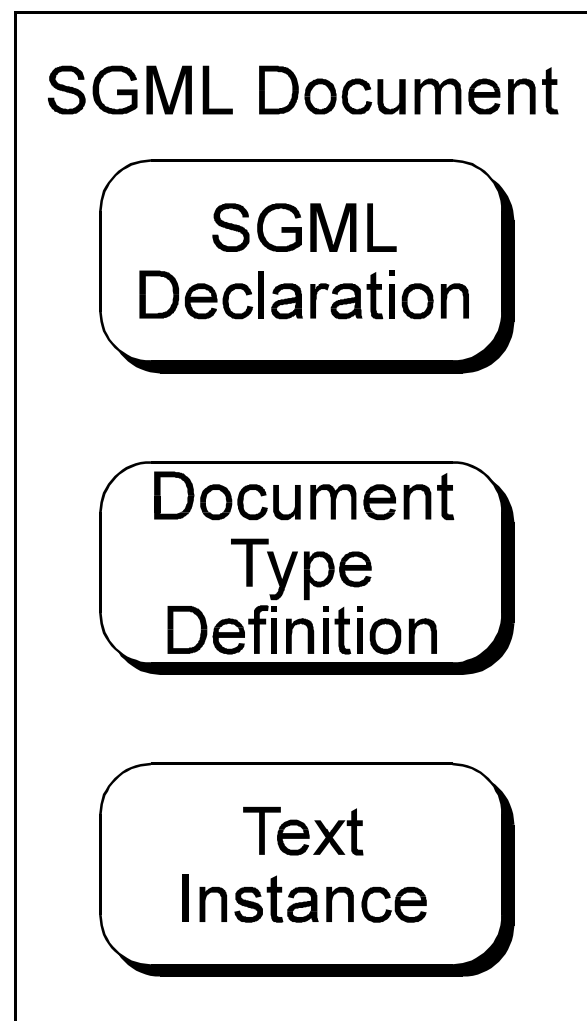


Fig. 16 : Structure of an SGML Document

7.3 Translation between a WordPerfect File and SGML

As mentioned earlier, in order to exchange programs between implementations, programs must first be marked up, and then transferred to another platform (electronically or physically). The transferred programs can then be translated back to normal (WordPerfect) programs or loaded directly into WPFORTH.

Here, we assume using SGML as the markup language. All other markup languages can be used, but SGML is the best choice because of its standardization.

There is another benefit when using SGML as the markup language. Since SGML only marks the logical elements within a document, the resulting marked document will be word processor independent. For example, suppose we create a program on an IBM PC using WordPerfect, and then markup the program and transfer it to a Macintosh, on which we only have Microsoft Word available. We can then translate the marked program into Word format and edit it in Microsoft Word. The use of SGML help us achieve multi-platform typographical programming.

The first thing to be done is to define the SGML document structure, mainly the DTD of a WPFORTH program.

To make things simple, the default SGML declaration can be used. This means we are to accept all the default SGML settings, including the tag delimiter ("`<`"), ASCII character set, etc. Details about the default SGML declarations can be found in the SGML Handbook [GOL 90].

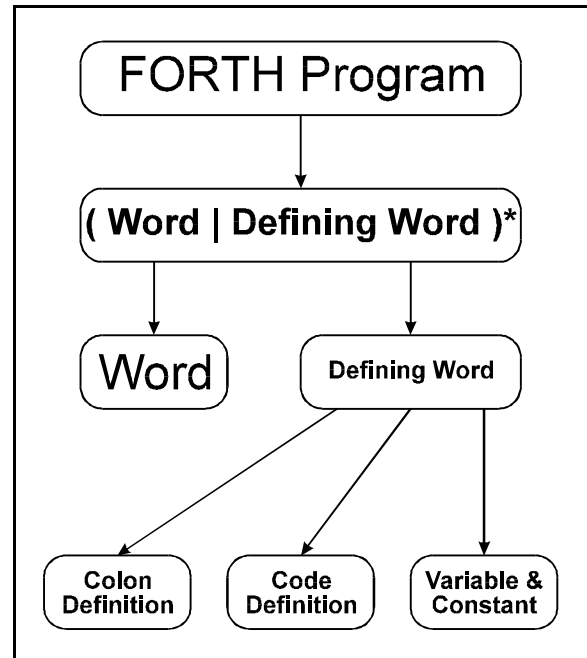


Fig. 17 : A Partial DTD for a FORTH Program

The DTD of a WPFORTH program must be defined. There is no existing DTD that we can use, because a WPFORTH program is different in structure from other general documents. Figure 17 is a partial possible DTD of a FORTH program. Figure 18 is a more detailed DTD for the part of a colon definition.

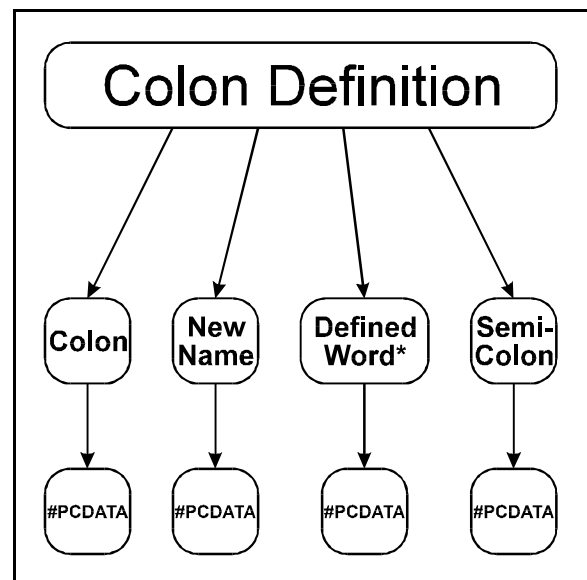


Fig. 18 : A Partial DTD for a Colon Definition

Some notation used in Figure 17

and 18 must be explained: a vertical bar ("|") represents a selection, i.e., we can choose either item beside the bar. For example, in Figure 17, we have "Word | Defining Word", that means, for that item, it may be a Word OR a Defining Word. An asterisk means repeating, i.e. the item can appear any number of times. For example, in Figure 18, we have "Defined Word*", that means in a colon definition, there can be any number of defined words.

After the DTD is defined, one can start the implementation of the WPFORTH to SGML translator. This can be done in any programming language, but the FORTH tradition suggests that it should be done within WPFORTH itself.

The translation should be a word by word (function by function) translation, with all the WordPerfect functions translated into various kinds of tags. For example, the definition

```
: TEST 1 1 + . ; ( A test colon definition )
```

may be translated into:

```
<colon>
    <newname> TEST </newname>
    <defined> 1 </defined>
    <defined> 1 </defined>
    <defined> + </defined>
    <defined> . </defined>
</colon>
<comment> A test colon definition </comment>
```

After the marked program is transferred to another platform, it must be translated back to a WPFORTH programs. Again, the translator can be written in any programming language, but it is preferable to implement it in WPFORTH itself. The purpose of the translation is to change all these tags back into a WordPerfect file. This includes the construction of the prefix area, which can be a very challenging work. One way to do this is to preserve the original prefix in a special tag (which may be called <prefix> and </prefix>).

A better approach would be for marked programs to be directly loadable by WPFORTH. In this case, WPFORTH must be able to read and translate all tags and take proper actions.

7.4 An Alternative Approach

An alternative approach to the SGML translation is to use UUENCODE/UUDECODE or other similar utilities. Since our only need is to exchange programs between platforms, we can encode a WPFORTH program by UUENCODE, resulting in a pure ASCII (although unreadable) file. We can then transmit this encoded file through an appropriate route (e.g. email). The receiving person will use UUDECODE to decode the transmitted file back to a WordPerfect file, and load it into WPFORTH.

This is a quick (and dirty), but easy approach. However, the drawback is that there cannot be any difference between the two implementations. For example, if the WPFORTH on the transmitting platform use italics as comments, and the WPFORTH on the receiving platform use boldface as comments, then the transmitted file is unusable without changing all the comment attributes with WordPerfect. But this will be a time-consuming task as discussed earlier. Figure 19 shows an excerpt of an encoded WordPerfect file.

VIII. DISCUSSION AND CONCLUSION

8.1 Limitation of WPFORTH

As it is a prototype, WPFORTH can be expected to have limitations. In addition to the necessary conversions of page breaks and carriage returns, the current version concentrates on fonts and attributes. All other possibilities are ignored. For example, if the programmer is using styles to write programs, since the information (text, attribute change, etc.) in a style is ignored by WPFORTH, it will not be available in this version. However, it can be added to WPFORTH without great difficulty.

Because WPFORTH is only a prototype, its execution speed is not our main concern. As a result, whenever possible, WPFORTH is written in high-level Pygmy FORTH. The only exceptions are some inter-segment operations which are beyond Pygmy FORTH are written in assembly language (with Pygmy FORTH's built-in assembler). However, the resulting performance is not too bad. When tested on a 286 machine with hard disk of 28ms access time, the time required to load a 13Kb file was around 6 seconds. It can be expected to have a

much better performance when some of the definitions in WPFORTH are re-written in assembler.

The total compiled size of WPFORTH is less than 9Kb in addition to the Pygmy FORTH's original size of 16Kb. This is not a large amount when compared to systems implemented in other programming languages. But in FORTH, if one considers that the original Pygmy FORTH occupies only 16Kb of memory, and the total available memory in the dictionary segment is only 64Kb, then 9K becomes a very large amount. Again, by optimizing some definitions, and by meta-compilation, the total memory requirement can be reduced.

8.2 Further Improvement

Since this project is only a prototype, it can be improved in many way, the following lists only some possibilities:

- a. Rewrite the code in assembly language to boost performance.
- b. Use meta-compilation to reduce the code size.
- c. Include some other WordPerfect functions, such as styles and graphics.
- d. Upgrade the system so that it has WordPerfect 6.0 compatibility.
- e. Include a WordPerfect to SGML translator and an SGML to WordPerfect translator.
- f. Implement an SGML loader so a WPFORTH program which is translated into SGML can be loaded directly.
- g. Rewrite the system for other word processors, such as Microsoft Word, AmíPro, etc.
- h. Rewrite the system so that it is based on other FORTH system (such as F83).
- i. Rewrite the system for other programming languages such as Pascal and C/C++. This will be a much more difficult task than any of the above. One possible approach is to follow Dr. Knuth's method: write a translator to translate Pascal code, which is written in a word processor's native format, into a

"normal" Pascal program, where all the format codes become system library call. The development cycle then becomes: Edit (in a WP or DTP program) - Translate (to a normal ASCII program) - Compile - Link (with WP or DTP system library).

8.3 Conclusion

Although the prototype presented here is quite limited, it can improve the readability of our programs and can also reduce the time to develop a program. For example, let us assume Pygmy FORTH can display bitmap graphics. The programmer is required to code all the bitmap pixels as program code. However, with the introduction of typographical programming in WPFORTH, the programmer needs only include a graphic inside the display string. Of course, in this case, WPFORTH is also required to be modified to accept graphical information from the WordPerfect file instead of ignore it, but it will not be a difficult task.

In the past, program code and the presentation document were two different things. We can, of course, put comments into the code, but the purpose of these comments is to explain the code. Also, the code cannot be printed with satisfactory quality unless it is formatted with some word processor or typesetter.

With the introduction of typographical programming, the gap between a program and its presentation document is reduced. We can foresee that in the future programmers will only have to prepare the presentation document which includes the requirements of the program. They can then compile the

document to produce the required executable code. Programming will be much more fun than it is today.

A P P E N D I C E S

A. SYSTEM REQUIREMENT

1. IBM PC, PC/XT, PC/AT compatible or higher machines.
2. At least 256K bytes of free (available) main memory after DOS and TSR's are loaded.
3. DOS 2.0 or higher.
4. At least one floppy drive (hard disk recommended).
5. Any kind of text display monitor capable for displaying at least 80 columns of text.

B. SOURCE LISTING OF WPFORTH

1. WPFORTH.F

```
( ***** )
( * Program : WPFORTH * )
( * Version : 1.0 * )
( *-----* )
( * Author : Albert Chan (9226386) * )
( *      Master of Science * )
( *      McMaster University * )
( *      Department of Computer Science and Systems * )
( *-----* )
( * File : WPFORTH.F * )
( * Last Modified : December 8, 1993. * )
( *-----* )
( * Description : Main module of WPFORTH, it will load all the * )
( *      sub-modules and then make necessary patches to * )
( *      the original Pygmy. * )
( ***** )

( The file WPFORTH.VAR defines and initializes all )
( the variables and constants used in the system )

INCLUDE WPFORTH.VAR

( The file WPFORTH.DBL defines some double number operations )

INCLUDE WPFORTH.DBL

( The file WPFORTH.UTL contains some general utilities )

INCLUDE WPFORTH.UTL

( The file WPFORTH.WPF contains definitions for )
( handling WordPrefect codes and functions )

INCLUDE WPFORTH.WPF

( The file WPFORTH.FON contains some utilities for handling fonts )

INCLUDE WPFORTH.FON

( The file WPFORTH.FIL contains definitions for handling File I/O )

INCLUDE WPFORTH.FIL
```

```

( The file WPFORTH.PFX contains definitions )
(   for handling WordPerfect prefix       )

INCLUDE WPFORTH.PFX

( The file WPFORTH.LOD redefines loading words for )
(   loading other definitions from either a block, )
(   an ASCII file or a WordPerfect document       )

INCLUDE WPFORTH.LOD

( The file WPFORTH.DEF redefines all necessary defining words )

INCLUDE WPFORTH.DEF

( The file WPFORTH.SAV contains all definitions )
(   for saving a WPFORTH image to an .EXE file   )

INCLUDE WPFORTH.SAV

( Patch the definition of HEAD to point to NEW-(HEAD )

' NEW-(HEAD      ( Address of NEW-(HEAD          )
' HEAD           ( Address of HEAD              )
19 +            ( 1st Offset of (HEAD in HEAD )
!              ( Store the patch                )

' NEW-(HEAD      ( Address of NEW-(HEAD          )
' HEAD           ( Address of HEAD              )
27 +            ( 2nd Offset of (HEAD in HEAD )
!              ( Store the patch                )

( Patch the definition of SOURCE to point to NEW-?REFILL )

' NEW-?REFILL    ( Address of NEW-?REFILL        )
' SOURCE         ( Address of SOURCE            )
33 +            ( Offset of ?REFILL in SOURCE )
!              ( Store the patch                )

```

2. WPFORTH.VAR

```
( ***** )
( * Program : WPFORTH * )
( * Version : 1.0 * )
( *-----* )
( * Author : Albert Chan (9226386) * )
( *      Master of Science * )
( *      McMaster University * )
( *      Department of Computer Science and Systems * )
( *-----* )
( * File : WPFORTH.VAR * )
( * Last Modified : December 8, 1993. * )
( *-----* )
( * Description : This file contains definitions for all variables * )
( *      and constants used in the system. * )
( ***** )
```

```
( Definitions of boolean constant TRUE and FALSE )
```

```
0 0= CONSTANT TRUE
TRUE NOT CONSTANT FALSE
```

```
( The following constants define the      )
( sixteen attributes used by WordPerfect )
```

```
1 CONSTANT EXTRA-LARGE
2 CONSTANT VERY-LARGE
4 CONSTANT LARGE
8 CONSTANT SMALL
16 CONSTANT FINE
32 CONSTANT SUPERSCRIP
64 CONSTANT SUBSCRIPT
128 CONSTANT OUTLINE
256 CONSTANT ITALICS
512 CONSTANT SHADOW-FONT
1024 CONSTANT REDLINE
2048 CONSTANT DOUBLE-UNDERLINE
4096 CONSTANT BOLD
8192 CONSTANT STRIKEOUT
16384 CONSTANT UNDERLINE
32768 CONSTANT SMALL-CAPS
```

```
( Variables used in handling attributes and fonts )
```

```
VARIABLE ?EXPAND      ?EXPAND      ON      ( Expand only outside "..." )
VARIABLE ?WPLOADING  ?WPLOADING  OFF
VARIABLE ATTRIBUTE    ATTRIBUTE    OFF
VARIABLE ?COMMENT     ITALICS      ?COMMENT !
VARIABLE ?REMARK      ?REMARK      OFF
VARIABLE CUR-FONT     0              CUR-FONT !
VARIABLE DEFAULT-FONT
VARIABLE DEFAULT-ATT
```

```

( Implementation of a file stack )

11 2* CONSTANT FILE-BUFFER-SIZE      ( 11 items of file must be saved )

VARIABLE CUR-FILE-POINTER             CUR-FILE-POINTER OFF

VARIABLE FILE-SEGMENT

( Constants used when saving files in .EXE format )

          ( Length of .EXE Header )
          $20 CONSTANT HEADER-LENGTH

          ( Starting address of Header )
$100 HEADER-LENGTH - CONSTANT HEADER-ADDRESS

          ( FILE LENGTH MOD 512 )
HEADER-ADDRESS 2 + CONSTANT FILE-LENGTH-BYTE

          ( FILE LENGTH IN BLOCK )
FILE-LENGTH-BYTE 2 + CONSTANT FILE-LENGTH-BLOCK

( WordPerfect Control Variables )

VARIABLE WPC          ( WordPerfect Control Code Table )
VARIABLE WPF          ( WordPerfect Function Table )
VARIABLE WPP          ( WordPerfect Packet Table )

( File Loading Control Variables )

VARIABLE K-BUF        ( Keyboard buffer )
VARIABLE CUR-POS 0 , ( CUR-POS is defined as a double variable )

( Variables used in handling fonts )

VARIABLE FONT-SEGMENT
CREATE FONT-NAME-POOL 4 ALLOT          0 0 FONT-NAME-POOL 2!
CREATE FONT-POOL 4 ALLOT              0 0 FONT-POOL 2!
CREATE FONT-POOL-LENGTH 4 ALLOT       0 0 FONT-POOL-LENGTH 2!
VARIABLE FONT-SEARCH-POINTER         -1 FONT-SEARCH-POINTER !
VARIABLE FONT-POINTER                0 FONT-POINTER !
VARIABLE FONT-PACKET                 0 FONT-PACKET !

( The following variable indicates whether )
( we are returning from another file )

VARIABLE RETURN-FROM-ANOTHER-FILE    RETURN-FROM-ANOTHER-FILE OFF

( The variable SHOW-TIB control whether the content )
( in the input buffer, TIB, will be displayed on )
( screen while loading a file, it is off by default )

VARIABLE SHOW-TIB                    SHOW-TIB OFF

```

3. WPFORTH.DBL

```

( ***** )
( * Program : WPFORTH * )
( * Version : 1.0 * )
( *-----* )
( * Author : Albert Chan (9226386) * )
( *      Master of Science * )
( *      McMaster University * )
( *      Department of Computer Science and Systems * )
( *-----* )
( * File : WPFORTH.DBL * )
( * Last Modified : December 8, 1993. * )
( *-----* )
( * Description : The file will defines some operations for double * )
( *      number operations that is not presented in PYGMY * )
( *      but necessary for WPFORTH. Note that not all * )
( *      double number operations are defined but only * )
( *      the necessary ones. * )
( ***** )

: D+      ( dl1 dh1 d12 dh2 --- d13 dh3 )
  PUSH    ( dl1 dh1 d12 )
  ROT     ( dh1 d12 dl1 )
  OVER    ( dh1 d12 dl1 d12 )
  + SWAP  ( dh1 d13 d12 )
  OVER    ( dh1 d13 d12 d13 )
  SWAP U< ( dh1 d13 f )
  IF      ( dh1 d13 )
    1     ( dh1 d13 1 )
  ELSE    ( dh1 d13 )
    0     ( dh1 d13 0 )
  THEN    ( dh1 d13 carry )
  ROT     ( d13 carry dh1 )
  POP     ( d13 carry dh1 dh2 )
  + +     ( d13 dh3 )
;

: D-      ( dl1 dh1 d12 dh2 --- d13 dh3 )
  PUSH    ( dl1 dh1 d12 )
  ROT     ( dh1 d12 dl1 )
  DUP ROT ( dh1 d11 dl1 d12 )
  - SWAP  ( dh1 d13 dl1 )
  OVER    ( dh1 d13 dl1 d13 )
  U<      ( dh1 d13 f )
  IF      ( dh1 d13 )
    -1    ( dh1 d13 1 )
  ELSE    ( dh1 d13 )
    0     ( dh1 d13 0 )
  THEN    ( dh1 d13 carry )
  ROT     ( d13 carry dh1 )
  POP     ( d13 carry dh1 dh2 )
  - +     ( d13 dh3 )
;

```

```

: D0=          ( dl dh --- f )
OR 0=         ( f )
;

: D=          ( dl1 dh1 dl2 dh2 --- f )
ROT          ( dl1 dl2 dh1 dh2 )
=           ( dl1 dl2 f1 )
ROT ROT     ( f1 dl1 dl2 )
=           ( f1 f2 )
AND         ( f )
;

: UD<         ( dl1 dh1 dl2 dh2 --- f )
ROT SWAP    ( dl1 dl2 dh1 dh2 )
2DUP        ( dl1 dl2 dh1 dh2 dh1 dh2 )
U<          ( dl1 dl2 dh1 dh2 f1 )
IF          ( dl1 dl2 dh1 dh2 )
  DROP DROP ( dl1 dl2 )
  DROP DROP ( - )
  TRUE      ( true )
ELSE       ( dl1 dl2 dh1 dh2 )
  =        ( dl1 dl2 f2 )
  IF      ( dl1 dl2 )
    U<    ( f )
  ELSE    ( dl1 dl2 )
    DROP DROP ( - )
    FALSE  ( false )
  THEN    ( f )
THEN      ( f )
;

```

4. WPFORTH.UTL

```
( ***** )
( * Program : WPFORTH * )
( * Version : 1.0 * )
( *-----* )
( * Author : Albert Chan (9226386) * )
( *      Master of Science * )
( *      McMaster University * )
( *      Department of Computer Science and Systems * )
( *-----* )
( * File : WPFORTH.UTL * )
( * Last Modified : December 8, 1993. * )
( *-----* )
( * Description : This file contains definitions for general * )
( *      utilities that cannot be classified. * )
( ***** )
```

```
( The following definition change a number to a boolean value )
( 0 = FALSE, all others = TRUE, this is not necessary in most )
( version of FORTH, but it is included just for safety )
```

```
: N>BOOL      ( n --- f )
0=           ( f' )
0=           ( f )
;
```

```
( Change a value to a string, store it in the given address, )
( and return the final length. For example if n = 512, then )
( the string "512" will be stored in the address starting )
( with a, and n' will be 3. This definition uses the )
( technique similar to the sequence <# #S #> except it )
( stores the value in the address instead of prints it out, )
( and some additional housekeeping work. )
```

```
: N>STR      ( n a --- a n' )
PUSH        ( n )      ( R: a )
<# #S DROP R@ SWAP PUSH R@ ( .. a n' ) ( R: a n' )
FOR SWAP    ( .. a n' )
OVER C! 1+  ( .. a' )
NEXT        ( a' )
DROP POP POP SWAP ( a n' )
;
```

```
( This definition preform the bitwise-not operation, since PYGMY do )
( not have bitwise-not operation. Note : -1 is same as 65535 or )
( 1111111111111111b, when the given value is subtracted from 65535 )
( all the 1-bits will become 0, and the 0-bits will become 1 )
```

```
: BITWISE-NOT ( n --- n' )
-1 SWAP -     ( n' )
;
```

(The following can be used to retrieve the actual code segment)

```

CODE GET-CODE-SEGMENT          ( x --- x s )
  BX PUSH,                     ( x x )
  CS PUSH,                      ( x s x )
  BX POP,                       ( x s )
  NXT,                          ( x s )
END-CODE

```

(Push a number to the second top position of the return stack)

```

: PUSH2ND                      ( n --- ) ( R: r --- n r )
                                ( n )      ( R: r r' )
  POP POP                       ( n r' r ) ( R: - )
  ROT                           ( r' r n ) ( R: - )
  PUSH                          ( r' r )   ( R: n )
  PUSH PUSH                     ( - )      ( R: n r r' )
;                               ( - )      ( R: n r )

```

(Pop a number from the second top position of the return stack)

```

: POP2ND                        ( --- n ) ( R: n r --- r )
                                ( - )      ( R: n r r' )
  POP POP                       ( r' r )   ( R: n )
  POP                           ( r' r n ) ( R: - )
  SWAP PUSH                     ( r' n )   ( R: r )
  SWAP PUSH                     ( n )      ( R: r r' )
;                               ( n )      ( R: r )

```

(Read one byte from the current file, if end of file, return Ctrl-Z)

```

: CHAR-READ                     ( --- c )
  K-BUF 1 FBLK @                ( a l h )
  FILE-READ                     ( - )
  #BYTES-READ @ 0=              ( f )
  IF                             ( - )
    26                          ( c )
  ELSE                          ( - )
    K-BUF @                     ( c )
  THEN                          ( c )
;

```

(Read two bytes and convert it into a number)

```

: READ-WORD                     ( --- n )
  CHAR-READ                     ( cl )
  CHAR-READ                     ( cl ch )
  256 * +                       ( n )
;

```

(Drop some bytes from the file)

```

: DROP-CHAR                     ( n --- )
  FOR                          ( - )
    CHAR-READ DROP              ( - )
  NEXT                          ( - )
;

```

```

( Read a string into the given address, )
( returning the length of the string   )

: READ-STRING          ( a --- n )
  DUP                  ( a a )
  BEGIN                ( a a' )
    CHAR-READ          ( a a' c )
    DUP                ( a a' c f )
  WHILE                ( a a' c )
    OVER C!            ( a a' )
    1+                  ( a a" )
  REPEAT               ( a a" )
  DROP                 ( a a' )
  SWAP -                ( n )
;

( Patch a number in code segment )

: PATCH                ( new address --- old )
  DUP @ PUSH           ( new address )
  !                    ( - )
  POP                  ( old )
;

( Transfer data between different segments )
( )
( x - data that are in some depth under the top of the stack )
( fs - source segment )
( fa - source address )
( ts - target segment )
( ta - target address )
( # - number of bytes to be moved )
( DS - saved current data segment value )

CODE <SMOVE>          ( x fs fa ts ta # --- x )
                     ( x fs fa ts ta # )
                     ( x fs fa ts ta # )
                     ( x fs fa ts ta # )
                     ( x fs fa ts ta DS )
                     ( x fs fa ts DS )
                     ( x fs fa DS )
                     ( x fs fa DS )
                     ( x fs DS )
                     ( x DS )
                     ( x DS )
                     ( x DS )
                     ( x DS )
                     ( x DS )
                     ( x DS )
                     ( x DS )
                     ( x )
                     ( x )
END-CODE

```

5. WPFORTH.WPF

```
( ***** )
( * Program : WPFORTH * )
( * Version : 1.0 * )
( *-----* )
( * Author : Albert Chan (9226386) * )
( *      Master of Science * )
( *      McMaster University * )
( *      Department of Computer Science and Systems * )
( *-----* )
( * File : WPFORTH.WPF * )
( * Last Modified : December 8, 1993. * )
( *-----* )
( * Description : This file contains definitions for handling all * )
( *      WordPerfect functions * )
( ***** )
```

```
( Reserved definition for changing font/attribute, cannot )
( be redefined. It also ensures the number after it being )
( compiled as a literal, not a constant. )
( )
( It is defined in both the vocabularies FORTH and COMPILER )
```

```
: wpf ( --- )
  32 WORD ( addr )
  NUMBER ( n )
  32 WORD ( n addr )
  2 ( n addr forth )
  -FIND ( n addr true | n pfa false )
  ABORT" Error occurred while parsing, Aborted."
  EXECUTE ( - )
;
```

COMPILER

```
: wpf ( --- )
  32 WORD ( addr )
  NUMBER ( n )
  \ LITERAL ( n )
  32 WORD ( addr )
  4 ( addr compiler )
  -FIND ( addr true | n pfa false )
  ABORT" Error occurred while parsing, Aborted."
  EXECUTE ( - )
;
```

FORTH

```
( This definition decides whether we should replace the attribute )
( with a space or adjust its address : )
( )
( If the attribute indicates that the character is a comment, the )
( byte will be replace with a space with no address adjustment, )
( the space will be discarded later by NEW-REAS-LINE, since there )
( is no address adjustment both here and in NEW-READ-LINE, there )
( will not any bytes be lost. However, if the byte is not a )
( comment, and the address will be adjusted, and the previous )
( byte will be return, this adjustment will be compensated by )
( NEW-READ-LINE when it try to store the previous byte in the )
( the previous location. )
```

```
: ?>FIB ( a --- a' c' )
?REMARK @ ( f )
IF ( a )
  32 ( a c' )
ELSE ( a )
  1- ( a' )
  DUP C@ ( a' c' )
THEN ( a' c' )
;
```

```
( The following definitions will insert the definitions )
( ATTR-ON and ATTR-OFF in the input buffer, so action )
( can be taken when the input is being processed. )
```

```
: (ATTRIB-ON ( a c --- a' c' )
DROP ( a )
1 ( a 1 )
CHAR-READ ( a 1 c" )
FOR ( a n )
  2* ( a n*2 )
NEXT ( a n1 )
1 DROP-CHAR ( a n1 )
DUP ?COMMENT @ AND ( a n1 f1 )
IF ( a n1 )
  ?REMARK ON ( a n1 )
THEN ( a n1 )
DUP ?COMMENT @ AND NOT ( a n1 f2 )
?EXPAND @ AND ( a n1 f3 )
IF ( a n1 )
  SWAP DUP ( n1 a a )
  " wpf " COUNT ( n1 a a "_wpf_" 5 )
  ROT SWAP ( n1 a "_wpf_" a 5 )
  CMOVE ( n1 a )
  5 + ( n1 a1 )
  N>STR ( a1 n2 )
  + DUP ( a2 a2 )
  " ATTR-ON" ( a2 a2 a3 )
  COUNT ( a2 a2 a3+1 n3 )
  PUSH SWAP R@ ( a2 a3+1 a2 n3 )
  CMOVE ( a2 )
  POP ( a2 n3 )
  + ( a' )
  32 ( a' c' )
ELSE ( a n1 )
  DROP ( a )
  ?>FIB ( a' c' )
THEN ( a' c' )
;
```

```

: (ATTRIB-OFF ( a c --- a' c' )
DROP ( a )
1 ( a 1 )
CHAR-READ ( a 1 c" )
FOR ( a n )
  2* ( a n*2 )
NEXT ( a n1 )
1 DROP-CHAR ( a n1 )
DUP ?COMMENT @ AND ( a n1 f1 )
IF ( a n1 )
  ?REMARK OFF ( a n1 )
THEN ( a n1 )
DUP ?COMMENT @ AND NOT ( a n1 f2 )
?EXPAND @ AND ( a n1 f3 )
IF ( a n1 )
  SWAP DUP ( n1 a a )
  " wpf " COUNT ( n1 a a "_wpf_" 5 )
  ROT SWAP ( n1 a "_wpf_" a 5 )
  CMOVE ( n1 a )
  5 + ( n1 a1 )
  N>STR ( a1 n2 )
  + DUP ( a2 a2 )
  " ATTR-OFF" ( a2 a2 a3 )
  COUNT ( a2 a2 a3+1 n3 )
  PUSH SWAP R@ ( a2 a3+1 a2 n3 )
  CMOVE ( a2 )
  POP ( a2 n3 )
  + ( a' )
  32 ( a' c' )
ELSE ( a n1 )
  DROP ( a )
  ?>FIB ( a' c' )
THEN ( a' c' )
;

```

```

( The following definitions actually handle the )
( turn on and off of the attribute variable. )
( )
( It is defined in both the vocabularies FORTH and COMPILER )

```

```

: ATTR-ON ( n --- )
ATTRIBUTE @ ( n att )
OR ( att' )
ATTRIBUTE ! ( - )
;

```

```

: ATTR-OFF ( n --- )
BITWISE-NOT ( n' )
ATTRIBUTE @ ( n' att )
AND ( att' )
ATTRIBUTE ! ( - )
;

```

COMPILER

```

: ATTR-ON          ( --- )
  HERE 2 - @      ( n )
  ATTRIBUTE @     ( att )
  OR              ( att' )
  ATTRIBUTE !     ( - )
  COMPILE ATTR-ON ( - )
;

```

```

: ATTR-OFF        ( --- )
  HERE 2 - @      ( n )
  BITWISE-NOT    ( n' )
  ATTRIBUTE @     ( att )
  AND            ( att' )
  ATTRIBUTE !     ( - )
  COMPILE ATTR-OFF ( - )
;

```

FORTH

```

( The words defined by SUB-SCRIPT and SUPER-SCRIPT has the capacity )
( to return either an address or the value of that address          )
( depending whether it is in subscript/superscript mode or not      )

```

```

: SUB-SCRIPT      ( --- )      ( COMPILE TIME )
                  ( --- a|n )  ( RUN TIME )
  CREATE 0 ,      ( - )        ( COMPILE TIME ACTION )
  DOES>           ( a )         ( RUN TIME ACTION )
  ATTRIBUTE @     ( a att )
  SUBSCRIPT AND N>BOOL ( a f )
  IF              ( a )
    @             ( n )
  THEN            ( a|n )
;

```

```

: SUPER-SCRIPT   ( --- )      ( COMPILE TIME )
                  ( --- a|n )  ( RUN TIME )
  CREATE 0 ,      ( - )        ( COMPILE TIME ACTION )
  DOES>           ( a )         ( RUN TIME ACTION )
  ATTRIBUTE @     ( a att )
  SUPERScript AND N>BOOL ( a f )
  IF              ( a )
    @             ( n )
  THEN            ( a|n )
;

```

```

( The following definition will insert the definitions )
( FONT-CHANGE in the input buffer, so action can be )
( taken when the input is being processed. )

: (FONT-CHANGE ( a c --- a' c' )
DROP ( a )
CHAR-READ ( a c" )
1 = ( a f1 )
IF ( a )
  27 DROP-CHAR ( a )
  CHAR-READ ( a font )
  9 DROP-CHAR ( a font )
  ?EXPAND @ ( a font f2 )
  IF ( a font )
    SWAP DUP ( font a a )
    " wpf " COUNT ( font a a "_wpf_" 5 )
    ROT SWAP ( font a "_wpf_" a 5 )
    CMOVE ( font a )
    5 + ( font a1 )
    N>STR ( a1 n1 )
    + DUP ( a2 a2 )
    " FONT-CHANGE" ( a2 a2 a3 )
    COUNT ( a2 a2 a3+1 n3 )
    PUSH SWAP R@ ( a2 a3+1 a2 n3 )
    CMOVE ( a2 )
    POP ( a2 n3 )
    + ( a' )
    32 ( a' c' )
  ELSE ( a font )
    DROP ( a )
    ?>FIB ( a' c' )
  THEN ( a' c' )
ELSE ( a )
  CHAR-READ ( a c1 )
  CHAR-READ ( a c1 ch ) ( READ THE LENGTH BYTES )
  256 * + ( a n ) ( CALCULATE THE LENGTH )
  BEGIN ( a n )
    DUP 0= NOT ( a n f )
  WHILE ( a n ) ( WHILE THERE ARE MORE BYTES )
    1 DROP-CHAR ( a n ) ( DROP A BYTE )
    1- ( a n-1 )
  REPEAT ( a n' )
  DROP ( a )
  ?>FIB ( a' c' )
THEN
;

```

```

( The following definitions actually handle the change of font. )
( )
( It is defined in both the vocabularies FORTH and COMPILER )

```

```

: FONT-CHANGE ( font --- )
CUR-FONT ! ( - )
;

```

COMPILER

```

: FONT-CHANGE          ( --- )
  HERE                 ( here )
  2 - @                ( font )
  CUR-FONT !           ( - )
  COMPILER FONT-CHANGE ( - )
;

```

FORTH

```
( Change the tab to a space )
```

```

: (TAB                  ( c --- [space] )
  DROP                 ( - )
  8 DROP-CHAR          ( - )
  32                   ( [space] )
;

```

```
( Change the page break etc. into a new line )
```

```

: (NEWLINE             ( c --- [cr] )
  DROP 13              ( [cr] )
;

```

```
( Change all kind of hyphen to a minus sign "-" )
```

```

: (HYPHEN              ( c --- [-] )
  DROP 45              ( [-] )
;

```

```
( Drop a single byte function )
```

```

: (SINGLE               ( a c --- a' c' )
  DROP                 ( a )
  ?>FIB                ( a' c' )
;

```

```
( Drop a fixed length function )
```

```

: (FIX                 ( a c n --- a' c' )
  SWAP DROP            ( a n ) ( DROP THE ORIGINAL BYTE )
  1+ DROP-CHAR         ( a )
  ?>FIB                ( a' c' )
;

```

```

: (1FIX 1 (FIX ;
: (2FIX 2 (FIX ;
: (3FIX 3 (FIX ;
: (4FIX 4 (FIX ;
: (5FIX 5 (FIX ;
: (6FIX 6 (FIX ;
: (7FIX 7 (FIX ;
: (8FIX 8 (FIX ;
: (9FIX 9 (FIX ;

```

(Drop a variable length function)

```

: (VARIABLE ( a c --- a' c' )
  DROP ( a ) ( DROP THE FUNCTION BYTE )
  1 DROP-CHAR ( a ) ( DROP THE SUBFUNCTION BYTE )
  CHAR-READ ( a cl )
  CHAR-READ ( a cl ch ) ( READ THE LENGTH BYTES )
  256 * + ( a n ) ( CALCULATE THE LENGTH )
  BEGIN ( a n )
    DUP 0= NOT ( a n f )
  WHILE ( a n ) ( WHILE THERE ARE MORE BYTES )
    1 DROP-CHAR ( a n ) ( DROP A BYTE )
    1- ( a n-1 )
  REPEAT ( a n' )
  DROP ( a )
  ?>FIB ( a' c' )
;

```

(WordPerfect function tables)

CREATE WP-CONTROL

WP-CONTROL WPC !

```

] ( 00H ) NOP ( SINGLE ( SINGLE ( SINGLE
  ( 04H ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE
  ( 08H ) ( SINGLE ( SINGLE ( NEWLINE ( NEWLINE
  ( 0CH ) ( NEWLINE ( NEWLINE ( SINGLE ( SINGLE

  ( 10H ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE
  ( 14H ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE
  ( 18H ) ( SINGLE ( SINGLE NOP ( SINGLE
  ( 1CH ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE [

```

CREATE FUNCTION-TABLE

FUNCTION-TABLE WPF !

```

] ( 80H ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE
  ( 84H ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE
  ( 88H ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE
  ( 8CH ) ( NEWLINE ( SINGLE ( SINGLE ( SINGLE

  ( 90H ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE
  ( 94H ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE
  ( 98H ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE
  ( 9CH ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE

  ( A0H ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE
  ( A4H ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE
  ( A8H ) ( SINGLE ( HYPHEN ( HYPHEN ( HYPHEN
  ( ACH ) ( HYPHEN ( HYPHEN ( HYPHEN ( SINGLE

  ( B0H ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE
  ( B4H ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE
  ( B8H ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE
  ( BCH ) ( SINGLE ( SINGLE ( SINGLE ( SINGLE

  ( C0H ) ( 2FIX ( TAB ( 9FIX ( ATTRIB-ON
  ( C4H ) ( ATTRIB-OFF ( 3FIX ( 4FIX ( 5FIX
  ( C8H ) NOP NOP NOP NOP
  ( CCH ) NOP NOP NOP NOP

```

(D0H)	(VARIABLE	(FONT-CHANGE	(VARIABLE	(VARIABLE
(D4H)	(VARIABLE	(VARIABLE	(VARIABLE	(VARIABLE
(D8H)	(VARIABLE	(VARIABLE	(VARIABLE	(VARIABLE
(DCH)	(VARIABLE	(VARIABLE	(VARIABLE	(VARIABLE
(E0H)	(VARIABLE	(VARIABLE	(VARIABLE	(VARIABLE
(E4H)	(VARIABLE	(VARIABLE	(VARIABLE	(VARIABLE
(E8H)	(VARIABLE	(VARIABLE	(VARIABLE	(VARIABLE
(ECH)	(VARIABLE	(VARIABLE	(VARIABLE	(VARIABLE
(F0H)	(VARIABLE	(VARIABLE	(VARIABLE	(VARIABLE
(F4H)	(VARIABLE	(VARIABLE	(VARIABLE	(VARIABLE
(F8H)	(VARIABLE	(VARIABLE	(VARIABLE	(VARIABLE
(FCH)	(VARIABLE	(VARIABLE	(VARIABLE	(VARIABLE [

6. WPFORTH.FON

```
( ***** )
( * Program : WPFORTH * )
( * Version : 1.0 * )
( *-----* )
( * Author : Albert Chan (9226386) * )
( *      Master of Science * )
( *      McMaster University * )
( *      Department of Computer Science and Systems * )
( *-----* )
( * File : WPFORTH.FON * )
( * Last Modified : December 8, 1993. * )
( *-----* )
( * Description : This file contains definitions for handling * )
( *      fonts, it uses a font segment to keep all the * )
( *      font information. Initially, the font segment * )
( *      is set to two segments above the code segment. * )
( *      However, if WPFORTH is loaded from the saved * )
( *      .EXE image, the font segment will be only one * )
( *      segment above the code segment. * )
( ***** )
```

```
( Initially the font segment is set to two )
( whole segments above the code segment. )
```

```
GET-CODE-SEGMENT $2000 + FONT-SEGMENT !
```

```
( Transfer data to and from font segment )
( and code segment, act like CMOVE. )
```

```
: DATA->FONT ( addr1 addr2 # --- )
  PUSH PUSH PUSH ( - )
  GET-CODE-SEGMENT ( cs )
  POP ( cs addr1 )
  FONT-SEGMENT @ ( cs addr1 fs )
  POP POP ( cs addr1 fs addr2 # )
  <SMOVE> ( - )
;
```

```
: FONT->DATA ( addr1 addr2 # --- )
  PUSH PUSH PUSH ( - )
  FONT-SEGMENT @ ( fs )
  POP ( fs addr1 )
  GET-CODE-SEGMENT ( fs addr1 cs )
  POP POP ( fs addr1 cs addr2 # )
  <SMOVE> ( - )
;
```

```
( The structure of font segment is : )
( Offset : Description )
( ----- )
( 00 : font number )
( 01 - 02 : pointer to previous font )
( 03 - 04 : font size )
( 05 : # = length of font name )
( 06 - #+5 : font name )
( #+6 : an ending zero )
```

```

( Read font name from the file and save it to the font segment )

: READ-FONT-NAME ( font np size --- font+1 )
  SWAP 0 ( font size np 0 )
  FONT-NAME-POOL 2@ D+ ( font size font-offset )
  FBLK @ >POSITION ( font size )
  PAD 3 + ! ( font )
  DUP ( font )
  PAD C! ( font )
  FONT-SEARCH-POINTER @ PAD 1+ ! ( font )
  PAD 6 + READ-STRING ( font n )
  DUP PAD 5 + C! ( font n )
  DUP PAD + 6 + 0 SWAP C! ( font n )
  FONT-POINTER @ ( font n fp )
  FONT-SEARCH-POINTER ! ( font n )
  7 + ( font n' )
  DUP ( font n' n' )
  PAD FONT-POINTER @ ROT ( font n' pad fp n' )
  DATA->FONT ( font n' )
  FONT-POINTER +! ( font )
  1+ ( font+1 )
;

( Search for a font given its font number, )
( returning offset in the font segment )

: SEARCH-FONT ( font --- offset )
  PUSH ( - )
  FONT-SEARCH-POINTER @ DUP ( fp fp)
  -1 ( fp fp -1 )
  BEGIN ( fp' pfp font' )
    OVER -1 = NOT ( fp' pfp font' f1 )
    OVER R@ = NOT AND ( fp' pfp font' f2 )
  WHILE ( fp' pfp font' )
    DROP ( fp' pfp )
    SWAP DROP ( pfp )
    DUP PAD 3 FONT->DATA ( pfp )
    PAD 1+ @ ( pfp pfp' )
    PAD C@ ( pfp pfp' font" )
  REPEAT ( pfp pfp' font" )
  POP = ( fp' pfp f3 )
  IF ( fp' pfp )
    DROP ( offset )
  ELSE ( fp' -1 )
    SWAP DROP ( offset )
  THEN ( offset )
;

( Read font information from the font )
( segment, returning the font size )

: (READ-FONT-INFORMATION ( offset address --- size )
  PUSH ( offset )
  DUP PAD 6 FONT->DATA ( offset )
  PAD 3 + @ SWAP ( size offset )
  5 + ( size offset+5 )
  PAD 5 + C@ ( size offset+5 len )
  2 + ( size offset+5 len+2 )
  POP SWAP ( size offset+5 addr len+2 )
  FONT->DATA ( size )
;

```

```

( Read font information from the font segment, given the font )
( number, returning the font size, -1 if font cannot be found )

: READ-FONT-INFORMATION          ( font address --- size|-1 )
  SWAP                          ( address font )
  SEARCH-FONT                   ( address offset )
  DUP -1 = NOT                  ( address offset f )
  IF                             ( address offset )
    SWAP                        ( offset address )
    (READ-FONT-INFORMATION      ( size )
  ELSE                            ( address offset )
    DROP DROP                   ( - )
    -1                          ( -1 )
  THEN                           ( size|-1 )
;

( Return the font size given the font number, -1 if font not found )

: FONT-SIZE                      ( font --- size )
  PUSH                          ( - )
  FONT-SEARCH-POINTER @ DUP     ( fp fp)
  -1                             ( fp fp -1 )
  BEGIN                          ( fp' pfp font' )
    OVER -1 = NOT               ( fp' pfp font' f1 )
    OVER R@ = NOT AND          ( fp' pfp font' f2 )
  WHILE                          ( fp' pfp font' )
    DROP                       ( fp' pfp )
    SWAP DROP                  ( pfp )
    DUP PAD 5 FONT->DATA        ( pfp )
    PAD 1+ @                   ( pfp pfp' )
    PAD C@                     ( pfp pfp' font" )
  REPEAT                         ( pfp pfp' font" )
  POP =                         ( fp' pfp f3 )
  IF                             ( fp' pfp )
    DROP DROP                  ( - )
    PAD 3 + @                  ( size )
  ELSE                            ( fp' -1 )
    SWAP DROP                  ( -1 )
  THEN                           ( size | -1 )
;

( Return the point size of font, -1 if font not found )

: FONT-SIZE-IN-POINTS           ( font --- point )
  FONT-SIZE                     ( size )
  DUP -1 = NOT                  ( size f )
  IF                             ( size )
    5 + 10 /                   ( point )
  THEN                           ( point )
;

```

7. WPFORTH.FIL

```
( ***** )
( * Program : WPFORTH * )
( * Version : 1.0 * )
( *-----* )
( * Author : Albert Chan (9226386) * )
( *      Master of Science * )
( *      McMaster University * )
( *      Department of Computer Science and Systems * )
( *-----* )
( * File : WPFORTH.FIL * )
( * Last Modified : December 8, 1993. * )
( *-----* )
( * Description : This file contains definitions for handling file * )
( *      I/O. It uses the font segment to preserve all * )
( *      the information that is necessary to be restored * )
( *      after the file is loaded. * )
( ***** )
```

```
( The file segment is always 3 segments above the code segment )
```

```
GET-CODE-SEGMENT $3000 + FILE-SEGMENT !
```

```
( Middle level definition to "READ a byte" from a file, )
( special actions will be taken for some special byte. )
( Note : the stack diagram for this definition is : )
( a --- a' c )
( where a' = a + bytes read - 1 )
( c is the last character read )
( in normal case, a' will be equal to a and that )
( means one character is read )
```

```
: CH-READ ( a --- a' c )
?WPLOADING @ ( a f )
IF ( a )
CHAR-READ ( a c' )
DUP 32 < ( a c' f1 )
IF ( a c' )
DUP 2* WPC @ + @ ( a c' a1 )
EXECUTE ( a' c )
ELSE ( a c' )
DUP 127 > ( a c' f2 )
IF ( a c' )
DUP 128 - 2* WPF @ + @ ( a c' a2 )
EXECUTE ( a' c )
THEN ( a' c )
THEN ( a' c )
ELSE ( a )
CHAR-READ ( a' c )
THEN ( a' c )
;
```

```

( Push file information onto the file stack )
(
( a[0]          is ?WPLOADING          )
( a[1]          is BLK                 )
( a[2]          is >IN                 )
( a[3]          is FBLK                )
( a[4]          is >FIN                )
( a[5]          is #FIB                )
( a[6] & a[7]   is CUR-POS             )
( a[8]          is CUR-FONT            )
( a[9]          is FONT-SEARCH-POINTER )
( a[10]         is ATTRIBUTE           )

: PUSH-FILE          ( --- )
  PAD                ( a[0] )
  ?WPLOADING @ OVER ! 2 + ( a[1] )
  BLK @ OVER ! 2 +    ( a[2] )
  >IN @ OVER ! 2 +   ( a[3] )
  FBLK @ OVER ! 2 +  ( a[4] )
  >FIN @ OVER ! 2 +  ( a[5] )
  #FIB @ OVER ! 2 +  ( a[6] )
  PUSH CUR-POS 2@ R@ 2! POP 4 + ( a[8] )
  CUR-FONT @ OVER ! 2 + ( a[9] )
  FONT-SEARCH-POINTER @ OVER ! 2 + ( a[10] )
  ATTRIBUTE @ SWAP ! ( - )
  GET-CODE-SEGMENT PAD ( cs addr1 )
  FILE-SEGMENT @      ( cs addr1 fs )
  CUR-FILE-POINTER @ FILE-BUFFER-SIZE * ( cs addr1 fs addr2 )
  FILE-BUFFER-SIZE   ( cs addr1 fs addr2 size )
  <SMOVE>             ( - )
  1 CUR-FILE-POINTER +! ( - )
;

( Pop information from the file stack )

: POP-FILE          ( --- )
  CUR-FILE-POINTER @ 0= ( f )
  ABORT" No File on Stack, Aborted."
  -1 CUR-FILE-POINTER +! ( - )
  FILE-SEGMENT @      ( fs )
  CUR-FILE-POINTER @ FILE-BUFFER-SIZE * ( fs addr1 )
  GET-CODE-SEGMENT PAD ( fs addr1 cs addr2 )
  FILE-BUFFER-SIZE   ( fs addr1 cs addr2 size )
  <SMOVE>             ( - )
  PAD                ( a[0] )
  DUP @ ?WPLOADING ! 2 + ( a[1] )
  DUP @ BLK ! 2 +      ( a[2] )
  DUP @ >IN ! 2 +     ( a[3] )
  DUP @ FBLK ! 2 +    ( a[4] )
  DUP @ >FIN ! 2 +    ( a[5] )
  DUP @ #FIB ! 2 +    ( a[6] )
  DUP 2@ CUR-POS 2! 4 + ( a[8] )
  DUP @ CUR-FONT ! 2 + ( a[9] )
  DUP @ FONT-SEARCH-POINTER ! 2 + ( a[10] )
  @ ATTRIBUTE !      ( - )
;

```

```

( This is a replacement of PYGMY's READ-LINE in ?REFILL. )
( It will take care of the comment attribute, and turn   )
( on/off the variable ?EXPAND switch when necessary.   )

: NEW-READ-LINE      ( --- a # )
  ?EXPAND ON        ( - )
  FBLK @            ( h )
  DUP 0=            ( h f )
  ABORT" No File is Loading." ( h )
  POSITION@ CUR-POS 2! ( - )
  FIB @ DUP         ( a a )
  BEGIN            ( a a )
    CH-READ        ( a a' c )
    DUP 13 = NOT    ( a a' c f1 )
    OVER 26 = NOT AND ( a a' c f2 )
  WHILE            ( a a' c )
    DUP 34 =        ( a a' c f3 )
    IF              ( a a' c )
      ?EXPAND @ NOT ?EXPAND ! ( a a' c )
    THEN            ( a a' c )
    ?REMARK @ NOT   ( a a' c f4 )
    IF              ( a a' c )
      OVER C!       ( a a' )
      1+            ( a a'+1 )
    ELSE            ( a a' c )
      DROP          ( a a' )
    THEN            ( a a' )
  REPEAT           ( a a' )
  13 =             ( a a' f5 )
  ?WPLOADING @ AND ( a a' f6 )
  IF               ( a a' )
    10 OVER C! 1+  ( a a'+1 )
  THEN             ( a a" )
  OVER -           ( a # )
  DUP #FIB !       ( a # )
  SHOW-TIB @      ( a # f7 )
  IF               ( a # )
    OVER OVER TYPE ( a # )
    DUP 0=         ( a # f8 )
    IF             ( a # )
      CR           ( a # )
    ELSE           ( a # )
      13 EMIT      ( a # )
    THEN           ( a # )
  THEN             ( a # )
;

( If we are returning from another file, the input buffer pointer )
( must be reserved, otherwise, it must be reset to zero.         )

: RESET->IN      ( --- )
  RETURN-FROM-ANOTHER-FILE @ ( f )
  IF              ( - )
    RETURN-FROM-ANOTHER-FILE OFF ( - )
  ELSE            ( - )
    >IN OFF       ( - )
  THEN            ( - )
;

```

```

( This is a replacement of PYGMY's ?REFILL in SOURCE. )
( It will take care of whether we are returning from )
( another file and take necessary action. )

: NEW-?REFILL ( h --- a # )
  PUSH ( - )
  #FIB @ >IN @ > NOT ( f1 )
  FIBH @ R@ - ( f1 f2 )
  OR ( f3 )
  FIB @ ( f3 a )
  SWAP ( a f3 )
  IF ( a )
    RESET->IN ( a )
    BEGIN ( a )
      DROP ( - )
      R@ FIBH ! ( - )
      NEW-READ-LINE ( a # )
      >IN @ ( a # n )
      IF ( a # )
        OVER >IN @ 32 FILL ( a # )
        >IN OFF ( a # )
      THEN ( a # )
      DUP ( a # f4 )
      IF ( a # )
        2DUP ( a # a # )
        -CTRL ( a # )
        -TRAILING ( a # )
        DUP #FIB ! ( a # )
      ELSE ( a false )
        NOT ( a true )
      THEN ( a f6 )
    UNTIL ( a )
  THEN ( a )
  POP DROP ( a )
  #FIB @ ( a # )
;

```

8. WPFORTH.PFX

```
( ***** )
( * Program : WPFORTH * )
( * Version : 1.0 * )
( * ----- * )
( * Author : Albert Chan (9226386) * )
( *      Master of Science * )
( *      McMaster University * )
( *      Department of Computer Science and Systems * )
( * ----- * )
( * File : WPFORTH.PFX * )
( * Last Modified : December 8, 1993. * )
( * ----- * )
( * Description : This file contains definitions for processing * )
( *      the Prefix of a WordPerfect document * )
( ***** )
```

```
( All WordPerfect Files start with WordPerfect Signature : )
( "FF 57 50 4C" or "ΔWPC" at the first four byte )
```

```
: CHECK-WORDPERFECT ( --- f )
  FBLK @ >BOF ( - )
  CHAR-READ 255 ( FFH ) = ( f1 )
  CHAR-READ 87 ( "W" ) = AND ( f2 )
  CHAR-READ 80 ( "P" ) = AND ( f3 )
  CHAR-READ 67 ( "C" ) = AND ( f )
;
```

```
( All WordPerfect Files put the length of )
( Prefix at byte 4-7 as a double number )
```

```
: READ-PREFIX-LENGTH ( --- dl dh )
  READ-WORD ( dl )
  READ-WORD ( dl dh )
;
```

```
( WordPerfect documents have "01 0A" at byte 8 and byte 9 )
```

```
: CHECK-DOCUMENT ( --- f )
  CHAR-READ 1 ( 01H ) = ( f1 )
  CHAR-READ 10 ( 0AH ) = AND ( f2 )
;
```

```
( WordPerfect document have the version number at byte 10 and )
( byte 11 for version 5.x, the number in byte 10 should be 0. )
```

```
: CHECK-VERSION ( --- version )
  CHAR-READ ( version )
  1 DROP-CHAR ( - )
;
```

```

( Byte 12-13 indicate whether the file is encrypted or not )

: CHECK-ENCRYPT                ( --- f )
  CHAR-READ 0=                ( f1 )
  CHAR-READ 0= AND            ( f )
;

( Check whether the file is a WordPerfect Document file or not )
( the beginning of the document area will be returned )

: CHECK-SIGNATURE              ( --- dl dh )
  CHECK-WORDPERFECT           ( f1 )
  IF                           ( - )
    READ-PREFIX-LENGTH        ( dl dh )
    CHECK-DOCUMENT             ( dl dh f2 )
    NOT ABORT" is not a WordPerfect Document file."
    CHECK-VERSION              ( dl dh version )
    0= NOT                     ( dl dh f3 )
    ABORT" is not a WordPerfect version 5.x document."
    CHECK-ENCRYPT               ( dl dh f4 )
    NOT ABORT" cannot be read because it is encrypted."
    2 DROP-CHAR                ( dl dh )
  ELSE                          ( - )
    0 0                        ( dl dh )
  THEN                          ( dl dh )
;

( Skip a uninterested packet )

: SKIP-PACKET                  ( pl ph ll lh --- )
  DROP DROP                    ( pl ph )
  FBLK @ >POSITION             ( - )
;

( Packet 02 is the font packet in WordPerfect 5.0 )
( Action will be postponed until end of prefix )

: READ-PACKET-02              ( pl ph ll lh --- )
  FONT-POOL-LENGTH 2!          ( pl ph )
  FONT-POOL 2!                 ( - )
  2 FONT-PACKET !              ( - )
;

```

```

( Packet 03 is the initial instruction packet )

: READ-PACKET-03          ( pl ph ll lh --- )
  PUSH PUSH OVER OVER    ( pl ph pl ph )
  FBLK @ >POSITION        ( pl ph )
  POP POP                 ( pl ph ll lh )
  D+                      ( pl' ph' )
  BEGIN                  ( pl' ph' )
    OVER OVER             ( pl' ph' pl' ph' )
    FBLK @ POSITION@       ( pl' ph' pl' ph' pl" ph" )
    D= NOT                ( pl' ph' f )
  WHILE                  ( pl' ph' )
    PAD                   ( pl' ph' a )
    CHAR-READ             ( pl' ph' a c' )
    DUP 32 <              ( pl' ph' a c' f1 )
    IF                    ( pl' ph' a c' )
      DUP 2* WPC @ + @    ( pl' ph' a c' a1 )
      EXECUTE              ( pl' ph' a' c )
    ELSE                  ( pl' ph' a c' )
      DUP 127 >           ( pl' ph' a c' f2 )
      IF                  ( pl' ph' a c' )
        DUP 128 - 2* WPF @ + @ ( pl' ph' a c' a2 )
        EXECUTE            ( pl' ph' a' c )
      THEN                ( pl' ph' a' c )
    THEN                  ( pl' ph' a' c )
    DROP DROP            ( pl' ph' )
  REPEAT                 ( pl' ph' )
  DROP DROP              ( - )
;

( Packet 07 is the Font Name String Pool )
( Action will be postponed until end of prefix )

: READ-PACKET-07          ( pl ph ll lh --- )
  DROP DROP              ( pl ph )
  FONT-NAME-POOL 2!      ( - )
;

( Packet 0F is the font packet in WordPerfect 5.1 )
( Action will be postponed until end of prefix )

: READ-PACKET-0F          ( pl ph ll lh --- )
  FONT-POOL-LENGTH 2!    ( pl ph )
  FONT-POOL 2!           ( - )
  15 FONT-PACKET !      ( - )
;

```

```

( Packet 02 contains the following information in group of 86 bytes )
( Offset      Description                                         )
( -----      -----                                         )
(  0 -  1      Font instant pointer, ignore this font if = -1    )
(  2 - 17      Uninterested                                         )
( 18 - 19      Offset of font name in the font name pool         )
( 20 - 27      Uninterested                                         )
( 28 - 29      Font size                                           )
( 30 - 85      Uninterested                                         )

: (PROCESS-PACKET-02      ( --- )
  FONT-POOL 2@           ( pl ph )
  FBLK @ >POSITION      ( - )
  FONT-POOL-LENGTH 2@   ( ll lh )
  0                      ( ll lh 0 )
  BEGIN                 ( ll' lh' font )
    READ-WORD           ( ll' lh' font ip)
    -1 = NOT            ( ll' lh' font fl )
    IF                  ( ll' lh' font )
      16 DROP-CHAR     ( ll' lh' font )
      READ-WORD        ( ll' lh' font np )
      8 DROP-CHAR      ( ll' lh' font np )
      READ-WORD        ( ll' lh' font np size )
      FBLK @ POSITION@ PUSH PUSH ( ll' lh' font np size )
      READ-FONT-NAME   ( ll' lh' font' )
      POP POP FBLK @ >POSITION ( ll' lh' font' )
      56 DROP-CHAR     ( ll' lh' font' )
    ELSE                ( ll' lh' font )
      84 DROP-CHAR     ( ll' lh' font' )
    THEN                ( ll' lh' font' )
    PUSH                ( ll' lh' )
    86 0                ( ll' lh' 86 0 )
    D-                  ( ll" lh" )
    OVER OVER D0=      ( ll" lh" f2 )
    POP SWAP           ( ll" lh" font' f2 )
  UNTIL                ( ll" lh" font' )
  DROP DROP DROP      ( - )
;

```

```
( Packet 0F contains the following information in group of 86 bytes )
( Offset      Description                                     )
( -----      - )
( 0 - 1      Font instant pointer, ignore this font if = -1 )
( 2 - 17     Uninterested                                     )
( 18 - 19    Offset of font name in the font name pool      )
( 20 - 24    Uninterested                                     )
( 25 - 26    Font size                                       )
( 27 - 85    Uninterested                                     )
```

```
: (PROCESS-PACKET-0F      ( --- )
FONT-POOL 2@             ( pl ph )
FBLK @ >POSITION         ( - )
FONT-POOL-LENGTH 2@     ( ll lh )
0                         ( ll lh 0 )
BEGIN                    ( ll' lh' font )
  READ-WORD               ( ll' lh' font ip)
  -1 = NOT                ( ll' lh' font f1 )
  IF                      ( ll' lh' font )
    16 DROP-CHAR          ( ll' lh' font )
    READ-WORD             ( ll' lh' font np )
    5 DROP-CHAR           ( ll' lh' font np )
    READ-WORD             ( ll' lh' font np size )
    FBLK @ POSITION@ PUSH PUSH ( ll' lh' font np size )
    READ-FONT-NAME        ( ll' lh' font' )
    POP POP FBLK @ >POSITION ( ll' lh' font' )
    59 DROP-CHAR          ( ll' lh' font' )
  ELSE                    ( ll' lh' font )
    84 DROP-CHAR          ( ll' lh' font' )
  THEN                    ( ll' lh' font' )
  PUSH                   ( ll' lh' )
  86 0                   ( ll' lh' 86 0 )
  D-                     ( ll" lh" )
  OVER OVER D0=          ( ll" lh" f2 )
  POP SWAP               ( ll" lh" font' f2 )
UNTIL                    ( ll" lh" font' )
DROP DROP DROP          ( - )
;
```

```
( Postponed action for font processing )
( Process font depending on the packet type we found )
```

```
: PROCESS-FONT          ( - )
FONT-NAME-POOL 2@ OR 0= NOT ( f1 )
IF                      ( - )
  FONT-PACKET @ DUP     ( n n )
  2 =                   ( n f2 )
  IF                    ( n )
    DROP                ( - )
    (PROCESS-PACKET-02 ( - )
  ELSE                  ( n )
    15 =                ( f3 )
    IF                  ( - )
      (PROCESS-PACKET-0F ( - )
    THEN                ( - )
  THEN                  ( - )
THEN                    ( - )
0 0 FONT-POOL 2!       ( - )
0 0 FONT-POOL-LENGTH 2! ( - )
0 0 FONT-NAME-POOL 2! ( - )
0 FONT-PACKET !       ( - )
;
```

```
( WordPerfect packet tables )
( All the words in this table should have a stack diagram of : )
( pl ph ll lh --- )
( where pl ph is the pointer to the start of packet and ll lh )
( is the length of the packet, both values are double numbers )
```

```
CREATE WP-PACKET
```

```
WP-PACKET WPP !
```

```
] ( 00H ) SKIP-PACKET          SKIP-PACKET
  ( 02H ) READ-PACKET-02      READ-PACKET-03
  ( 04H ) SKIP-PACKET          SKIP-PACKET
  ( 06H ) SKIP-PACKET          READ-PACKET-07

  ( 08H ) SKIP-PACKET          SKIP-PACKET
  ( 0AH ) SKIP-PACKET          SKIP-PACKET
  ( 0CH ) SKIP-PACKET          SKIP-PACKET
  ( 0EH ) SKIP-PACKET          READ-PACKET-0F [
```

```
( We ignore all packet type larger than 0Fh )
```

```
( Read a regular packet )
```

```
: READ-PACKET          ( pl ph ll lh t --- )
  DUP 15 >            ( pl ph ll lh t f1 )
  IF                  ( pl ph ll lh t )
    DROP SKIP-PACKET ( - )
  ELSE                ( pl ph ll lh t )
    DUP 0 <          ( pl ph ll lh t f2 )
    IF              ( pl ph ll lh t )
      DROP SKIP-PACKET ( - )
    ELSE            ( pl ph ll lh t )
      2* WPP @ + @ EXECUTE ( - )
    THEN           ( - )
  THEN             ( - )
;                  ;
```

```

( Read a special packet )
( Pointer to next special packet will be returned )
( Note : the prefix D on the stack diagram represents )
( double number, for example Dp = p1 ph )

: READ-SPECIAL-PACKET ( --- nl nh )
  READ-WORD ( t )
  DUP 0= ( t f1 )
  IF ( 0 )
    0 ( 0 0 )
  ELSE ( t )
    -1 = ( f2 )
    IF ( - )
      8 DROP-CHAR ( - )
    ELSE ( - )
      READ-WORD ( n )
      1- ( n-1 )
      2 DROP-CHAR ( n-1 )
      READ-WORD ( n-1 nl )
      READ-WORD ( n-1 nl nh )
      ROT ( nl nh n-1 )
      DUP ( nl nh n-1 n-1 )
      FOR ( nl nh n-1 )
        POP ( nl nh n-1 I )
        READ-WORD PUSH ( nl nh n-1 I )
        READ-WORD READ-WORD PUSH PUSH ( nl nh n-1 I )
        READ-WORD READ-WORD PUSH PUSH ( nl nh n-1 I )
        PUSH ( nl nh n-1 )
      NEXT ( nl nh n-1 )
      DUP ( nl nh n-1 n-1 )
      FOR ( nl nh ... n-1 )
        POP ( nl nh ... n-1 I )
        POP ROT ROT POP ROT ROT ( nl nh ... Dp D1 n-1 I )
        POP ROT ROT POP ROT ROT ( nl nh ... Dp D1 t n-1 I )
        POP ROT ROT ( nl nh ... Dp D1 t n-1 )
        PUSH ( nl nh ... Dp D1 t n-1 )
      NEXT ( nl nh ... Dp D1 t n-1 )
      FOR ( nl nh ... Dp D1 t )
        READ-PACKET ( nl nh )
      NEXT ( nl nh )
    THEN ( nl nh )
  THEN ( nl nh )
;

( Skip the prefix of a WordPerfect file )

: SKIP-PREFIX ( dl dh --- )
  BEGIN ( dl dh )
    READ-SPECIAL-PACKET ( dl dh nl nh )
    OVER OVER ( dl dh nl nh nl nh )
    OR ( dl dh nl nh f )
  WHILE ( dl dh nl nh )
    FBLK @ >POSITION ( dl dh )
  REPEAT ( dl dh )
  PROCESS-FONT ( dl dh )
  DROP DROP ( dl dh )
  FBLK @ >POSITION ( - )
;

```

9. WPFORTH.LOD

```
( ***** )
( * Program : WPFORTH * )
( * Version : 1.0 * )
( *-----* )
( * Author : Albert Chan (9226386) * )
( *      Master of Science * )
( *      McMaster University * )
( *      Department of Computer Science and Systems * )
( *-----* )
( * File : WPFORTH.LOD * )
( * Last Modified : December 8, 1993. * )
( *-----* )
( * Description : This file will redefine all loading word in * )
( *      PYGMY, so we can mix with any combination of * )
( *      LOAD/FLOAD/WPLOAD/INCLUDE. * )
( ***** )
```

```
( Load word definitions from a WordPerfect or ASCII file )
( Usage : " filename.ext" (WPLOAD
```

```
: (WPLOAD ( a -- )
FBLK @ ( a f1 )
DUP PUSH ( a f1 )
IF ( a )
  PUSH-FILE ( a )
ELSE ( a )
  POP ( a f1 )
  >IN 2@ PUSH PUSH ( a f1 )
  PUSH ( a )
THEN ( a )
0 0 CUR-POS 2! ( a )
0 0 >FIN 2! ( a )
0 0 >IN 2! ( a )
0 CUR-FONT ! ( a )
-1 FONT-SEARCH-POINTER ! ( a )
ATTRIBUTE OFF ( a )
?REMARK OFF ( a )
FOPEN ( h f2 )
ABORT" File cannot be found." ( - )
FBLK ! ( - )
CHECK-SIGNATURE ( dl dh )
OVER 0= OVER 0= AND NOT ( dl dh f3 )
DUP ?WLOADING ! ( dl dh f3 )
IF ( dl dh )
  SKIP-PREFIX ( - )
ELSE ( dl dh )
  DROP DROP ( - )
  FBLK @ >BOF ( - )
THEN ( - )
>IN 2@ INTERPRET ( - )
10 BASE ! ( - )
FBLK @ FCLOSE ( - )
RETURN-FROM-ANOTHER-FILE ON ( - )
POP ( f1 )
IF ( - )
  POP-FILE ( - )
  FBLK @ FIBH ! ( - )
  CUR-POS 2@ FBLK @ >POSITION ( - )
ELSE ( - )
  0 0 >FIN 2! ( - )
  FIBH OFF ( - )
  POP POP >IN 2! ( - )
  -1 FONT-SEARCH-POINTER ! ( - )
THEN ( - )
;
```

```
( Usage : WLOAD filename.ext, this definition will use )
( less spaces than (WLOAD, in which the filename )
( will also be compiled into the dictionary in )
( the form of a string but never be used again )
```

```
: WLOAD ( --- )
32 WORD ( a )
0 ( a 0 )
OVER COUNT ( a 0 a # )
+ C! ( a )
(WLOAD ( - )
;
```

```

( Redefine FLOAD so it can load a WordPerfect file )

: FLOAD                      ( a --- )
  (WPLOAD                    ( - )
;

( Redefine INCLUDE so it can handle all kind of files )

: INCLUDE                    ( --- )
  WPLOAD                    ( - )
;

( Redefine LOAD and THRU so it will save )
( file information when necessary )

: LOAD                      ( n --- )
  FBLK @                    ( n f )
  DUP PUSH                  ( n f )
  IF                        ( n )
    PUSH-FILE              ( n )
  ELSE                      ( n )
    POP                    ( n f )
    >IN 2@ PUSH PUSH      ( n f )
    PUSH                   ( n )
  THEN                     ( n )
  0 0 >FIN 2!             ( n )
  0 INTERPRET              ( - )
  10 BASE !                ( - )
  POP                      ( f )
  IF                        ( - )
    RETURN-FROM-ANOTHER-FILE ON ( - )
    POP-FILE               ( - )
    FBLK @ FIBH !         ( - )
    CUR-POS 2@ FBLK @ >POSITION ( - )
  ELSE                      ( - )
    POP POP >IN 2!       ( - )
  THEN
;

```

```

: THRU ( n1 n2 --- )
FBLK @ ( n1 n2 f )
DUP PUSH ( n1 n2 f )
IF ( n1 n2 )
  PUSH-FILE ( n1 n2 )
ELSE ( n1 n2 )
  POP ( n1 n2 f )
  >IN 2@ PUSH PUSH ( n1 n2 f )
  PUSH ( n1 n2 )
THEN ( n1 n2 )
0 0 >FIN 2! ( n1 n2 )
OVER - 1+ ( n1 # )
FOR ( n1' )
  DUP PUSH ( n1' )
  0 INTERPRET ( - )
  POP 1+ ( n1" )
NEXT ( n1" )
DROP ( - )
10 BASE ! ( - )
POP ( f )
IF ( - )
  RETURN-FROM-ANOTHER-FILE ON ( - )
  POP-FILE ( - )
  CUR-POS 2@ FBLK @ >POSITION ( - )
ELSE ( - )
  POP POP >IN 2! ( - )
THEN
;

( Redefine ABORT so it will close all opened files, )
( a capacity that is lack in the original PYGMY. )

: NEW-(ABORT ( ... --- blk|- )
['] DEFAULT-EMIT 1+ @ ['] EMIT 1+ ! ( ... )
POP POP BLK @ PUSH PUSH PUSH ( ... )
FBLK @ ?DUP ( ... f f | ... 0 )
IF ( ... f | ... )
  FCLOSE ( ... )
THEN ( ... )
CUR-FILE-POINTER @ ( ... n )
FOR ( ... )
  POP-FILE ( ... )
  FBLK @ FCLOSE ( ... )
NEXT ( ... )
0 0 >FIN 2! ( ... )
0 0 >IN 2! ( ... )
-1 FONT-SEARCH-POINTER ! ( ... )
0 CUR-FONT ! ( ... )
0 ATTRIBUTE ! ( ... )
FIBH OFF ( ... )
HERE TYPE$ ( ... )
SPACE ( ... )
POP POP ( ... n1 n2 )
TYPE$ ( ... n1 )
SP! ( - )
POP ( blk )
?DUP DROP ( blk|- )
QUIT ( blk|- )
;

' NEW-(ABORT IS ABORT

```

10. WPFORTH.DEF

```
( ***** )
( * Program : WPFORTH * )
( * Version : 1.0 * )
( * ----- * )
( * Author : Albert Chan (9226386) * )
( *      Master of Science * )
( *      McMaster University * )
( *      Department of Computer Science and Systems * )
( * ----- * )
( * File : WPFORTH.DEF * )
( * Last Modified : December 8, 1993. * )
( * ----- * )
( * Description : This file will redefine all the defining words * )
( *      so they will do some checking before defining * )
( *      a new definition. It also defines the words * )
( *      ":" and ";" which is used for defining an * )
( *      attributes reserved definition. * )
( * ----- * )
( * Note : Most of the definitions in this file are not documented * )
( *      with the stack diagrams, since almost all of these * )
( *      definitions are system dependent, and the works * )
( *      presented in this file is only re-works of the original * )
( *      system. * )
( ***** )
```

```
( All defining words will now save the attributes and fonts )
( information for the corresponding end-definition to check )
```

```
: CODE
  CUR-FONT @ DEFAULT-FONT !
  ATTRIBUTE @ DEFAULT-ATT !
  CODE
;

: END-CODE
  ATTRIBUTE @
  DEFAULT-ATT @
  = NOT
  ABORT" Attribute not reset before end of CODE definition, Aborted."
  CUR-FONT @
  DEFAULT-FONT @
  = NOT
  ABORT" Font not reset before end of CODE definition, Aborted."
;
```

COMPILER

```

: ;
  ATTRIBUTE @
  DEFAULT-ATT @
  = NOT
  ABORT" Attribute not reset before end of colon definition, Aborted."
  CUR-FONT @
  DEFAULT-FONT @
  = NOT
  ABORT" Font not reset before end of colon definition, Aborted."
  \ RECURSIVE
  POP DROP
  COMPILE EXIT
;

```

FORTH

```

: :
  CUR-FONT @ DEFAULT-FONT !
  ATTRIBUTE @ DEFAULT-ATT !
  :
;

```

COMPILER

```

: [
  ATTRIBUTE @
  DEFAULT-ATT @
  = NOT
  ABORT" Attribute not reset."
  CUR-FONT @
  DEFAULT-FONT @
  = NOT
  ABORT" Font not reset."
  POP DROP
;

```

FORTH

```

: ]
  CUR-FONT @ DEFAULT-FONT !
  ATTRIBUTE @ DEFAULT-ATT !
  ]
;

```

```

( Simulate the docol and lit in the original PYGMY )
( which are not visible after metacompilation )

```

CODE (DOCOL

```

    SWITCH,
    SI PUSH,
    SWITCH,
    3 #, AX ADD,
    AX SI MOV,
    NXT,

```

END-CODE

```

CODE (LIT
    BX PUSH,
    AX LODS,
    AX BX MOV,
    NXT,
END-CODE

: SAVE-FONT-AND-ATTRIBUTE
  FONT-SEARCH-POINTER @ PUSH2ND
  CUR-FONT @ PUSH2ND
  ATTRIBUTE @ PUSH2ND
;

: RESTORE-FONT-AND-ATTRIBUTE
  POP2ND ATTRIBUTE !
  POP2ND CUR-FONT !
  POP2ND FONT-SEARCH-POINTER !
;

: ::
  CUR-FONT @ DEFAULT-FONT !
  ATTRIBUTE @ DEFAULT-ATT !
  HEAD $E9 C, (LIT
    (DOCOL HERE 2 + - ,
  COMPILER SAVE-FONT-AND-ATTRIBUTE
  FONT-SEARCH-POINTER @ \ LITERAL
  FONT-SEARCH-POINTER \ LITERAL COMPILER !
  CUR-FONT @ \ LITERAL CUR-FONT \ LITERAL COMPILER !
  ATTRIBUTE @ \ LITERAL ATTRIBUTE \ LITERAL COMPILER !
  SMUDGE ]
;

COMPILER

: ;;
  ATTRIBUTE @ ( att1 )
  DEFAULT-ATT @ ( att2 )
  = NOT ( f1 )
  ABORT" Attribute not reset before end of colon definition, Aborted."
  CUR-FONT @ ( font1 )
  DEFAULT-FONT @ ( font2 )
  = NOT
  ABORT" Font not reset before end of colon definition, Aborted."
  COMPILER RESTORE-FONT-AND-ATTRIBUTE
  \ RECURSIVE POP DROP COMPILER EXIT
;

```

FORTH

```

( Redefine the (HEAD so it will never redefine the word "wpf" )
(   which is reserved for the system use   )

: NEW-(HEAD ( --- )
  BLK @ , ( - )
  HERE 0 , ( here )
  32 WORD ( here addr )
  DUP " wpf" 4 COMP 0= ( here addr f1 )
  IF ( here addr )
    DROP ( here )
    BEGIN ( here )
      32 WORD DROP 32 WORD DROP ( here )
      32 WORD ( here addr' )
      " wpf" 4 COMP ( here f2 )
    UNTIL ( here )
    TRUE ( here true )
    ABORT" cannot change font/attribute immediate after defining word."
  THEN ( here addr )
  CONTEXT @ 2DUP -FIND NIP NOT ( here addr context f2 )
  IF ( here addr context )
    OVER TYPE$ ( here addr context )
    ." not unique " ( here addr context )
  THEN ( here addr context )
  HASH ( here addr hash )
  2DUP ( here addr hash addr hash )
  @ ( here addr hash addr n )
  SWAP ( here addr hash n addr )
  2 - ! ( here addr hash )
  SWAP ( here hash addr )
  C@ ( here hash length )
  1+ ALLOT ( here hash )
  ! ( - )
;

```

11. WPFORTH.SAV

```
( ***** )
( * Program : WPFORTH * )
( * Version : 1.0 * )
( *-----* )
( * Author : Albert Chan (9226386) * )
( *      Master of Science * )
( *      McMaster University * )
( *      Department of Computer Science and Systems * )
( *-----* )
( * File : WPFORTH.SAV * )
( * Last Modified : December 8, 1993. * )
( *-----* )
( * Description : This file contains definitions for saving * )
( *      WPFORTH to an .EXE image. * )
( ***** )
```

```
( Entry point of the system )
```

```
: EXE-BOOT ( --- )
GET-CODE-SEGMENT ( cs )
$1000 + DUP FONT-SEGMENT ! ( fs )
$2000 + FILE-SEGMENT ! ( - )
." *** WPFORTH *** Version 1.0 "
." copyright Winter 1993 by Albert Chan" CR CR
." This project is supervised by Dr. N. Solntseff and based on"
(BOOT ( - )
;
```

(EXE Header)

CREATE EXE-HEADER

Offset	Value	Description
0h	\$5A4D	.EXE file signature
2h	\$0000	File length mod 512, fill in later
4h	\$0000	File size in 512-byte blocks, fill in later
6h	\$0000	Number of relocation tables, set to none
8h	\$0002	Size of header in paragraphs
Ah	\$0000	Minimum paragraphs needed
Ch	\$FFFF	Maximum paragraphs needed, set to all
Eh	\$0000	SS, set when start up by system
10h	\$0000	Initial stack pointer, set by system
12h	\$0000	Word Checksum, ignored
14h	\$0100	Starting point of IP, set to always 0100h
16h	\$FFF0	CS - 100h below code start
18h	\$001C	Offset of 1st relocation tables
1Ah	\$0000	Number of overlays, zero here
1Ch	\$0000	Null relocation item, filled to end of header
1Eh	\$0000	. . . D i t t o . . .

(The file size is determined by the following formulae :)
 ()
 (Total file size = 64K for code segment)
 (- 256 bytes PSP)
 (+ 32 bytes .EXE Header)
 (+ font-pointer or fp)
 ()
 (Therefore, Total file size = 64K - 256 + 32 + fp)
 (= 64K - 224 + fp)
 ()
 (Since 64K is divisible by 512, so file size mod 512)
 (will be same as : fp-224 mod 512)
 ()
 (Also, 64K / 512 = 128, therefore, total file size in)
 (fp - 224 + 511 fp + 287)
 (block will be 128 + ----- = 128 + -----)
 (512 512)
 ()
 (The value 511 in the above formulae is used to ensure)
 (the result will be rounded up to next 512-byte block)
 (instead of rounded down.)

```

: TOTAL-FILE-SIZE          ( --- block byte )
  FONT-POINTER @          ( fp )
  DUP                      ( fp fp )
  287 + 512 U/ 128 +      ( fp block )
  SWAP                     ( block fp )
  224 - 512 U/MOD         ( block byte q )
  DROP                     ( block byte )
;

```



```

( Save the .EXE image, usage : " TEST.EXE" (SAVE-EXE          )
(                                                                 )
( There are two tricks in this definitions :                   )
(                                                                 )
( 1. It will peek at the current value of the stack segment, save it )
(   and patch with the value we want, and restore after save   )
(                                                                 )
( 2. It will get the deferred address of BOOT, save it and change it )
(   to EXE-BOOT, ana restore after save                       )
(                                                                 )
( Both tricks are to prevent the user to run BOOT after saving, in )
(   that case, the system will crash if the original values are not )
(   restored.                                                  )
(                                                                 )
( Usage : "TEST.EXE" (SAVE-EXE                                )
)

: (SAVE-EXE          ( addr --- )
  FMAKE              ( handle flag )
  ABORT" file?"
  $2000 $010D PATCH PUSH          ( handle )
  ['] BOOT 1+ @ PUSH              ( handle )
  ['] EXE-BOOT IS BOOT           ( handle )
  PUSH                          ( - )
  EXE-HEADER HEADER-ADDRESS      ( addr1 addr2 )
  HEADER-LENGTH CMOVE            ( - )
  TOTAL-FILE-SIZE                ( block byte )
  FILE-LENGTH-BYTE !             ( block )
  FILE-LENGTH-BLOCK !           ( - )
  HEADER-ADDRESS $FFFF OVER - 1+ ( cfrom clength )
  GET-CODE-SEGMENT              ( cfrom clength cseg )
  R@ WRITE-EXE                  ( - )
  $0 FONT-POINTER @ FONT-SEGMENT @ ( ffrom flength fseg )
  R@ WRITE-EXE                  ( - )
  POP FCLOSE                    ( - )
  POP IS BOOT                   ( - )
  POP $010D PATCH DROP          ( - )
;

( Usage : SAVE-EXE TEST.EXE )

: SAVE-EXE          ( --- )
  32 WORD           ( a )
  0                 ( a 0 )
  OVER COUNT       ( a 0 a # )
  + C!             ( a )
  (SAVE-EXE        ( - )
;

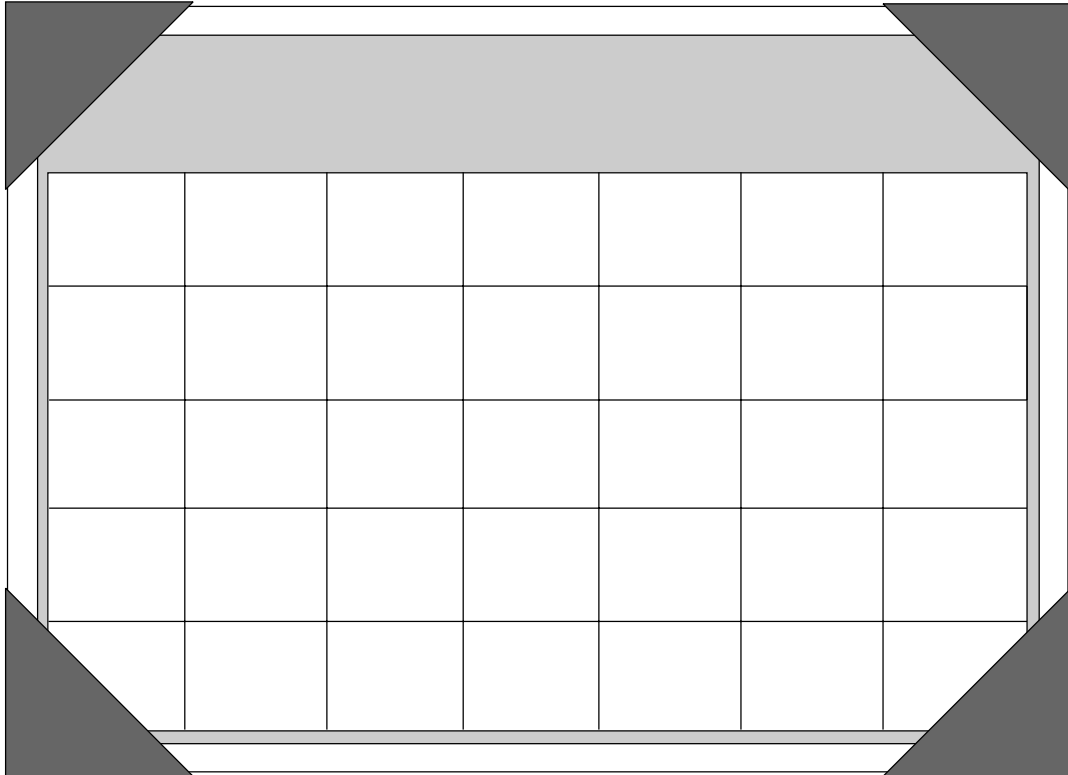
( Redefine SAVE so it always save an .EXE image )

: SAVE              ( --- )
  SAVE-EXE         ( - )
;

```

C. TESTING

1. Test Program



```
*****
* PURPOSE : THIS PROGRAM WILL PRINT THE CALENDAR OF A      *
*           MONTH ON THE TERMINAL SCREEN                    *
*                                                         *
* BY : ALBERT CHAN - 9226386                               *
*       DEPARTMENT OF COMPUTER SCIENCE AND SYSTEMS        *
*       McMASTER UNIVERSITY, HAMILTON, ONTARIO, CANADA    *
*                                                         *
* DATE : MAY 20, 1993                                     *
*                                                         *
* USAGE : year month CALENDAR                             *
*                                                         *
* EXAMPLE : 1993 5 CALENDAR                               *
*****
```



```

: INITIALIZE          ( y m --- f )
DEPTH 1 > DUP        ( y m f f ) IF ENOUGH PARAMETERS
IF                  ( y m f )
    SWAP MONTH !    ( y f )      STORE THE MONTH
    SWAP DUP YEAR ! ( f y )      STORE THE YEAR
    ARRAY-INIT      ( f y )      INIT. MDAY ARRAY
    CHECK-LEAP      ( f f1 )     IS IT A LEAP YEAR
    IF              ( f )
        29 2 MDAY ! ( f )      YES, FEBRUARY = 29
    THEN            ( f )
THEN              ( f )
;                  END OF INITIALIZE

: VALIDATE           ( f1 --- f )
DUP                ( f1 f1 )     IF ENOUGH PARAMETER
IF                ( f1 )
    DROP MONTH @    ( m )      FETCH THE MONTH
    DUP 0 > SWAP    ( f2 m )    MONTH > 0
    13 < AND        ( f )      AND < 13
THEN              ( f )
;                  END OF VALIDATE

: JAN-1ST           ( --- w )
YEAR @ 1-          ( y )      FETCH THE YEAR
DUP 4 /            ( y q1 )     LEAP EACH 4 YEARS
OVER 100 / NEGATE  ( y q1 -q2 ) NO LEAP AT 100 YEARS
PUSH OVER POP      ( y q1 y -q2 )
SWAP 400 /         ( y q1 -q2 q3 ) LEAP EACH 400 YEARS
+ + +             ( w )      ADD THEM TOGETHER
1+                 ( w )      FINAL ADJUSTMENT
;                  END OF JAN-1ST

: DAYS-FROM-1/1     ( --- w )
MONTH @            ( m )      FETCH THE MONTH
DUP 1 =            ( m f )     IF IT IS JANUARY
IF                ( m )
    DROP 0          ( t )      RETURN 0
ELSE              ( m )
    0                ( m 0 )    SET UP TOTAL = 0
    SWAP 1-          ( w m-1 )  SET UP DO LOOP LIMIT
    FOR              ( w )      FOR I = MONTH-2 DOWNT0 0
        I 1+ MDAY @ + ( w )    TOTAL += MDAY[I+1]
    NEXT            ( w )      NEXT I
THEN              ( w )      TOTAL IS ON STACK
;                  END OF DAYS-FROM-1/1

: CALC-CALENDAR     ( --- w )
JAN-1ST           ( w )      DAY ON JAN 1ST
DAYS-FROM-1/1     ( w )      DAYS FROM JAN 1ST
+ 7 UMOD          ( w )      COMPUTE DAY OF WEEK
;                  END OF CALC-CALENDAR

```

```

: PRINT-MONTH      ( --- )
  30 SPACES        ( - )           LEAVE SPACES
  MONTH @          ( m )           FETCH THE MONTH
    DUP 1 = IF ." JANUARY "        PRINT THE MONTH
  ELSE DUP 2 = IF ." FEBRUARY "
  ELSE DUP 3 = IF ." MARCH "
  ELSE DUP 4 = IF ." APRIL "
  ELSE DUP 5 = IF ." MAY "
  ELSE DUP 6 = IF ." JUNE "
  ELSE DUP 7 = IF ." JULY "
  ELSE DUP 8 = IF ." AUGUST "
  ELSE DUP 9 = IF ." SEPTEMBER "
  ELSE DUP 10 = IF ." OCTOBER "
  ELSE DUP 11 = IF ." NOVEMBER "
  ELSE              ." DECEMBER "
  THEN THEN THEN THEN THEN THEN THEN THEN THEN THEN THEN
  DROP              ( - )
  YEAR @            ( y )           FETCH THE YEAR
  11 .R             ( - )           PRINT THE YEAR
  CR CR             ( - )           LEAVE BLANK LINES
;                                                            END OF PRINT-MONTH

: PRINT-HEADING    ( --- )
  30 SPACES        ( - )           LEAVE SPACES
  ." SUNDAY MONDAY TUESDAY WEDNESDAY " FIRST HEADING
  ." THURSDAY FRIDAY SATURDAY "
  CR                                                        NEXT LINE
  30 SPACES        ( - )           LEAVE SPACES
  ." -----"                                             NEXT HEADING
;                                                            END OF PRINT-HEADING

: PRINT-DAY        ( d --- )
  DUP 0 >          ( d f )         DAY > 0 ?
  IF               ( d )
    3 .R           ( - )           YES, PRINT THE DAY
  ELSE             ( d )
    DROP           ( - )           NO, DROP THE "DAY"
    3 SPACES       ( - )           AND PRINT 3 SPACES
  THEN             ( - )
;

: NEXT-WEEK        ( - )
  CR               ( - )           START NEXT WEEK
  29 SPACES        ( - )           LEAVE ENOUGH SPACES
;

```

```

: PRT-CALENDAR      ( w --- )
CR CR              ( w )          LEAVE BLANK LINES
PRINT-MONTH        ( w )          PRINT THE MONTH
PRINT-HEADING      ( w )          PRINT THE HEADING
MONTH MDAY @       ( w dmax )     GET NUMBER OF DAYS
+ 0 SWAP           ( 0 w+dmax )   SET UP LIMITS
FOR                ( dw )         FOR I = w+dmax-1 DOWNT0 0
  DUP 0=           ( dw f1 )      IS FIRST DAY OF WEEK ?
  IF              ( dw )
    NEXT-WEEK      ( dw )         YES, START A NEW WEEK
  THEN            ( dw )
  1+ DUP 7 =      ( dw+1 f2 )     IS LAST DAY OF WEEK ?
  IF              ( dw+1 )
    DROP 0        ( dw' )        YES, NEW WEEK NEXT TIME
  THEN
    MONTH MDAY @   ( dw' dmax )   GET NUMBER OF DAYS
    I -            ( dw' d )      CALCULATE d = dmax - I
    PRINT-DAY      ( dw' )        PRINT THE DAY
NEXT              ( dw' )        NEXT I
DROP              ( - )          DROP dw'
CR CR CR          ( - )          LEAVE BLANK LINES
;                ( - )          END OF PRT-CALENDAR

: PRINT-ERROR      ( --- )
CR CR              ( - )          LEAVE BLANK LINES
." INPUT ERROR"   ( - )          ERROR MESSAGE
CR CR              ( - )          LEAVE BLANK LINES
;                ( - )          END OF PRINT-ERROR

: CALENDAR         ( y m --- )
INITIALIZE        ( f1 )         INITIALIZATION
VALIDATE          ( f )         VALIDATEION
IF                ( - )
  CALC-CALENDAR   ( w )         CALCULATE CALENDAR
  PRT-CALENDAR    ( - )         PRINT CALENDAR
ELSE              ( - )
  PRINT-ERROR     ( - )         PRINT ERROR
THEN
;                ( - )         END OF CALENDAR

: YEAR-CALENDAR    ( y --- )
12                ( y 12 )       SET UP FOR 12 MONTHS
FOR               ( y )          FOR I = 11 DOWNT0 0
  DUP             ( y y )         DUPLICATE YEAR
  12 I -         ( y y m )       MONTH = 12 - I
  CALENDAR        ( y )         DISPLAY CALENDAR
  KEY DROP        ( y )         WAIT FOR A KEY
NEXT              ( y )         NEXT
DROP              ( - )         DROP THE YEAR
;                ( - )         END OF YEAR-CALENDAR

```

2. Testing Result

C:\>PYGMY

PYGMY Forth v1.4 copyright 1989-1992 by Frank Sergeant

UNIT	1ST	LAST	HANDLE	FILE
0	0	191	5	PYGMY.SCR
1	1000		-1	PYGMY.DOW
2	2000	2007	6	YOURFILE.SCR
3			-1	
4			-1	
5			-1	
6			-1	
7			-1	
8			-1	
9			-1	
10			-1	
11			-1	
12			-1	
13			-1	
14			-1	
15			-1	

hi

INCLUDE WPFORTH.F

FLOAD not unique INCLUDE not unique LOAD not unique THRU not unique

CODE not unique END-CODE not unique ; not unique : not unique

[not unique] not unique SAVE not unique ok

SAVE WPFORTH.EXE ok

BYE

C:\>WPFORTH

*** WPFORTH *** Version 1.0 copyright Winter 1993 by Albert Chan

This project is supervised by Dr. N. Solntseff and based on
PYGMY Forth v1.4 copyright 1989-1992 by Frank Sergeant

UNIT	1ST	LAST	HANDLE	FILE
0	0	191	5	PYGMY.SCR
1	1000		-1	PYGMY.DOW
2	2000	2007	6	YOURFILE.SCR
3			-1	
4			-1	
5			-1	
6			-1	
7			-1	
8			-1	
9			-1	
10			-1	
11			-1	
12			-1	
13			-1	
14			-1	
15			-1	

hi

INCLUDE CALENDAR ok

SAVE CALENDAR.EXE ok

BYE

C:\>CALENDAR

*** WPFORTH *** Version 1.0 copyright Winter 1993 by Albert Chan

This project is supervised by Dr. N. Solntseff and based on
PYGMY Forth v1.4 copyright 1989-1992 by Frank Sergeant

UNIT	1ST	LAST	HANDLE	FILE
0	0	191	5	PYGYM.SCR
1	1000		-1	PYGYM.DOW
2	2000	2007	6	YOURFILE.SCR
3			-1	
4			-1	
5			-1	
6			-1	
7			-1	
8			-1	
9			-1	
10			-1	
11			-1	
12			-1	
13			-1	
14			-1	
15			-1	

hi

1994 1 CALENDAR

JANUARY 1994

SU	MO	TU	WE	TH	FR	SA
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

ok
BYE

BIBLIOGRAPHY

- [KLE-MAY 64] : M. Klerer and J. May, "An Experiment in a User-oriented Computer System", **Comm. ACM** 7(5), May 1964.
- [BRO 81] : Leo Brodie, **Starting FORTH**, Prentice-Hall, Englewood Cliffs, NJ., 1981.
- [SCA 82] : Leo J. Scanlon, **FORTH Programming**, H. W. Sams, Indianapolis, In., 1982.
- [LAX 83] : Henry Laxen, "Meta Compiling", **FORTH Dimensions IV**(6), March/April 1983, **V**(2), July/August 1983 and **V**(3), September/October 1983.
- [MCC 83] : C. Kevin McCabe, **FORTH Fundamentals**, Dilithium Press, Beaverton, Or., 1983.
- [KNU 84] : Donald E. Knuth, "Literate Programming", **The Computer Journal** 27(2), February 1984.
- [OAK 84] : Steve Oakey, **Forth for Micros**, Newnes Technical Books, Boston, Ma., 1984.
- [FEI-THO 85] : Gary Feierbach and Paul Thomas, **FORTH Tools and Applications**, Reston Pub. Co., Reston, Va., 1985.
- [ROL 85] : Dan Rollins, **8088 Marco Assembler Programming**, Macmillan Publishing Company, New York, NY., 1985.
- [TIN 85] : C. H. Ting, **Inside F83**, Offete Enterprises, Inc., San Mateo, Ca., 1985.
- [DUN 86] : Ray Duncan, **Advanced MS DOS**, Microsoft Press, Redmond, Wa., 1986.

- [ISO 86] : International Organization for Standardization, **ISO 8879-1986(E) - Information Processing - Text and Office systems - Standard Generalized Markup Language**, 1986.
- [MIL 86] : Alan R. Miller, **Assembly Language Techniques for the IBM PC**, Sybex, Berkeley, Ca., 1986.
- [REY 86] : A. J. Reynolds, **Advanced FORTH**, Sigma Press, Wilmslow, England, 1986.
- [SMI 86] : Joan M. Smith, **The Standard Generalized Markup Language and Related Issues**, British National Bibliography Research Fund, London, England, 1986.
- [HOL 87] : Henry L. Holloway, **An Introduction to Generic Coding and SGML for the Office for Humanities Communication**, British Library, London, England, 1987.
- [JAM 87] : Kris Jamsa, **DOS The Complete Reference**, Osborne McGraw-Hill, Berkeley, Ca., 1987.
- [MIL 87] : Daniel L. Miller, "STACK MACHINES AND COMPILER DESIGN", **Byte Magazine 12(4)**, April 1987.
- [PAS 87] : Alberto Pasquale, "F83 Compiles Text - Letters to the Editor", **FORTH Dimensions, VIII(5)**, January/February 1987.
- [ROS 87] : Douglas W. Ross, "Concept for FORTH Engine Based Workstation", **Journal of Forth Application and Research, 5(1)**, 1987.
- [SMI 87-1] : Joan M. Smith, **The Standard Generalized Markup Language (SGML): Guidelines for Authors**, British National Bibliography Research Fund, London, England, 1987.
- [SMI 87-2] : Joan M. Smith, **The Standard Generalized Markup Language (SGML) : Guidelines for Editors and Publishers**, British National Bibliography Research Fund, London, England, 1987.
- [HOG 88] : Thom Hogan, **The Programmer's PC Sourcebook**, Microsoft Press, Redmond, Wa., 1988.
- [BRO 89] : Heather Brown, "Standards for Structured Documents", **The Computer Journal, 32(6)**, 1989.

- [HAY 89] : Glen B. Haydon, "Formatting Source Code", **FORTH Dimensions**, X(6), March/April 1989.
- [SER 89] : Frank C. Sergeant, **Pygmy FORTH Documentation**, 1989.
- [TRA-AND 89] : Martin Tracy and Anita Anderson, **Mastering FORTH**, Brady, New York, NY., 1989.
- [WU 89] : Gilbert S.K. Wu, **SGML Theory and Practice**, British Library Research and Development Department, London, England, 1989.
- [ACE 90] : Karen L. Acerson, **WordPerfect 5.1 The Complete Reference**, Osborne McGraw-Hill, Berkeley, Ca., 1990.
- [BAE-MAR 90] : Ronald Baecker and Aaron Marcus, **Human Factors and Typography for More Readable Programs**, ACM Press/Addison-Wesley, Reading, Ma., 1990.
- [GOL 90] : Charles F. Goldfarb, **The SGML handbook**, Clarendon Press, Oxford, England, 1990.
- [HER 90] : Eric Van Herwijnen, **Practical SGML**, Kluwer Academic Publishers, Boston, Ma., 1990.
- [ANI 91] : American National Standard Institution, **Draft Proposed American National Standard for Information Systems - Programming Languages - FORTH (X3J14 dpANS-3)**, August 1991.
- [BOR-BOR 91] : Ute Bormann and Carsten Bormann, "Standards for Open Document Processing: Current State and Future Developments", **Computer Networks and ISDN Systems** 21(3), 1991.
- [WOR 91] : WordPerfect Corporation, **The WordPerfect Corporation Developer's Toolkit for PC Product**, WordPerfect Corporation, Orem, Ut, 1991.
- [ROD 92-93] : B. J. Rodriguez, "Principles of Meta-compilation", **FORTH Dimensions**, XIV(3), September/October 1992, XIV(4), November/December 1992 and XIV(6), January/February 1993.

- [ABR 93] : Paul W. Abrahams, "Typographical Extensions for Programming Languages: Breaking out of the ASCII Straitjacket", **ACM SIGPLAN Notices**, **28**(2), February 1993.
- [TIN 93] : C. H. Ting, **eFORTH and Zen**, Offete Enterprises, Inc., San Mateo, Ca., 1993.
- [WOR 93] : WordPerfect Corporation, **Developer's Toolkit Beta Documentation for WordPerfect 6.0 for DOS**, WordPerfect Corporation, Orem, Ut, 1993.