

PROGRAMMING METHODOLOGY.

© 2000 Rob Lucas and Keylink Computers Ltd
All rights reserved

1. INTRODUCTION	2
2. PROGRAM DOCUMENTATION	3
3. PSEUDO CODE	5
4. TOP DOWN DESIGN METHOD OF PROGRAM DESIGN.....	7
5. FUNCTIONAL SPECIFICATION.....	8
6. DEBUGGING	9
7. EXAMPLES OF THE TOP-DOWN METHOD.....	10
8. SIMPLE EXAMPLE.....	10
APPENDIX A – SOURCE CODE LISTING.....	16
APPENDIX B – EXAMPLE OUTPUT FROM THE 'LIFE' PROGRAM.	24
APPENDIX C – DEBUG TRACE	25
APPENDIX D – GLOSSARY.....	29

Keylink Computers Limited
2 Woodway house
Common Lane
Kenilworth
Warwickshire
CV8 2ES

1. Introduction

This document sets out those standards that are recommended for the design, coding and testing of computer programs using third generation languages such as Pascal, C, Fortran and Basic. This particular version of the document uses examples from Pascal and there are some comments for 'C' users.

This paper is not meant to be the last word on design and documentation, rather it is a starting point for others to build on. The methodology comes from what I was taught when programming for a software house. I will be very appreciative of all suggestions made as to how it might be improved.

February 1987.

I've made a few improvements and corrections. Most notably the documentation for a procedure separates parameter I/O, file I/O, and standard I/O. These had been confusing some people. I've added a glossary as an appendix. Some of the definitions are a little imprecise, as I've tried to make it readily understandable to the first year.

September 1991.

I've updated this to reflect requirements at Aberdeen.

April 2000

Changed to make the document relevant to Visual Basic. Standard input and output have now become Form input and output.

2. Program Documentation

Programs are given documentation so that they can be maintained. That is, so that they can be changed from time to time, either to remove bugs, or to improve/extend the functionality. A common reason given for this is that someone, other than the original programmer, will need to change the program and it is obvious that he will need some guidance, in the form of documentation, to do this. It has been found, through bitter experience, that the original programmer needs this documentation every bit as much. It should be emphasized that we are talking about programmers who write applications, that is professional programmers. There is as much difference between a ten line Basic program and an industrial application as between a paper plane and Concorde. If you have never documented your programs, then it is the case that either, you have never written a substantial program or you have never written a program that entirely worked.

Each program, in its specification and in its implemented form, will have the following documentation:

Name:

The name of the procedure

Purpose:

An accurate description of the function of the procedure. This should be clear and concise, do not include information on parameters or algorithms.

Calling format:

What an actual call to the procedure looks like using those parameter names which are given under parameter input and output. If the procedure is the main (driving) program, say so.

If the procedure has parameters or uses/modifies global values then include one or both of the following sections:

Parameter input:

invar1 (type)	-description.
invar2 (type)	-description.
.	.
.	.

Parameter output:

outvar1 (type)	-description.
outvar2 (type)	-description.
.	.
.	.

From these descriptions and the purpose, it should be clear how the output is related to the input. That, is the reader should have a clear idea of what values the output variables take given values for the input variables. If it isn't, you should think about respecifying the procedure. See notes on top-down design in this document. If any global data structures are used or modified these should be included under the input and output section respectively. If a program performs any input or output activity to the screen, or from the keyboard via a form, then include one or both sections:

Form input

invar1 (type)	-description.
invar2 (type)	-description.
.	.
.	.

Form output

outar1 (type)	-description.
outvar2 (type)	-description.
:	:
.	.

If a program performs any input or output activity to any files then include a section that describes the file i/o in a similar fashion to the sections above.

Processing notes:

A reference to the source of the algorithm.

A description of any data structures used inside the program. Types may be specified as in Pascal/C or more informally, clarity is the main thing. The top level program's processing notes can describe here data structures used by lower level procedures. The lower level procedures' documentation may refer to any types defined here. The same is true for globally defined constants.

Author/Date

Give the author and date on a single line.

Modifications:

Finally give a list of modifications under the subheadings of *Purpose*, *Date* and *Notes*.

In the case of smaller examples, such as student exercises where the code is all compiled at one time from one file, the last few entries, i.e. author, date and modifications can be omitted from all procedures bar the main driving program.

The calling format is only required for the specification of the procedure, it does not need to be part of the documentation that goes with the actual code, as this includes the calling format as part of the procedure declaration.

In the case of the top-most program, input or output may be expressed more informally than stating actual variable names and may refer to i/o activity that occurs in a subprocedure rather than literally in the main procedure.

Having a clearly written statement of what you intended a procedure to do can be a saver of hours of your time.

Examples of this documentation can be found with the example design and appendix A.

3. Pseudo Code

Pseudo code:

- (1). offers an alternative method to flowcharts for specifying algorithms.
- (2). Imitates the high level constructs of programming languages.
- (3). Is easier to change than flowcharts.
- (4). Allows a straightforward progression from an informal design in stylized English to a formal specification in pseudo code. This greatly facilitates the 'top-down' design method.
- (5). The final pseudo code design is more specific, and therefore easier to code from.

Pseudo Code Conventions.

- (1) Written in lower case.
- (2) Language elements of the code are underlined.
- (3) Variable, constant and procedure names may be any length, if they are more than one word then they are joined with an underscore, e.g. top_of_form.
- (4) Comments or remarks are written after a semicolon.
- (5) Strings are written between single quotes.
- (6) Scope of loops and ifs are indented. (A single indent is four spaces).
- (7) All code between start and end of procedure is indented.
- (8) Abbreviations in English are allowed where corresponding code is obvious.
- (9) Where a design is hand written, the space character is denoted by either a delta or a 'b' and '/' superimposed.

The Language elements.

begin	start of pseudo code.
debug (var1, var2,...)	output of variables for debugging purposes.
if condition then statement(s)	structured if as fortran77 etc.
else if condition then statement(s)	optional else if clause(s).
else statement(s)	optional else clause. ends scope of the if.
endif	
case of x label:statement(s)	case statement
else statement(s)	for no matching label
endcase	end of scope for case
goto label	a label is specified by a name Followed by a colon.
loop for index = m to n step statement(s)	for loop, step is optional