

# TOOLWORKS™ C/80 MATHPAK FLOATS AND LONGS FOR C/80 COMPILER

*The Software Toolworks*

SHERMAN OAKS, CALIFORNIA

**C/80 MATHPAK.** Adds true 32 bit float and long data types to the Toolworks C/80 language. Includes fast assembly language arithmetic library, plus C source code for formatted I/O routines and transcendental function library. Configuration program modifies C compiler to compile float and long declarations. Requires C/80 version 3.

*Programming Language Series*

*The Software Toolworks®*

SEE DISKETTE LABEL FOR MACHINE REQUIREMENTS



# The Software Toolworks®

15233 VENTURA BOULEVARD, SUITE 1118, SHERMAN OAKS, CALIFORNIA 91403 (818) 986-4885

## TOOLWORKS C/80 MATHPAK

### Float and Long Data Types for Toolworks C/80 Version 3

Version 3.1

March 1984

Walt Bilofsky

## Contents

|  |    |
|--|----|
| 1. INTRODUCTION.....                           | 2  |
| 2. THE MATHPAK DISTRIBUTION DISK.....          | 3  |
| 3. INSTALLING AND USING THE MATHPAK.....       | 4  |
| 3.1. Installing the MATHPAK.....               | 4  |
| 3.2. Using the MATHPAK with AS.....            | 4  |
| 3.3. Using the MATHPAK with Macro-80 or RMAC.4 |    |
| 3.4. Object Program Size Considerations.....   | 5  |
| 4. ADDITIONS TO THE C/80 LANGUAGE.....         | 6  |
| 5. ADDITIONS TO THE FUNCTION LIBRARY.....      | 7  |
| 5.1. Printing Floats and Longs.....            | 7  |
| 5.2. Using Printf Without Floats and Longs...8 |    |
| 5.3. Using Scanf with Floats and Longs.....    | 8  |
| 5.4. Float and String Conversion.....          | 8  |
| 5.5. Transcendental Function Library.....      | 9  |
| 6. IMPLEMENTATION OF FLOATS AND LONGS.....     | 10 |

Copyright (c) 1983, 1984 Walter Bilofsky. Sale of this software conveys a license for its use on a single computer at a time, owned or operated by the purchaser. Copying this software or documentation by any means whatsoever for any other purpose is expressly prohibited. C/80 and Toolworks are trademarks of The Software Toolworks.

## 1. INTRODUCTION

The Toolworks C/80 MATHPAK adds true 32 bit long and float data types to the C/80 Version 3 compiler and runtime library. It includes the CCONFIGF program to patch the C/80 compiler to recognize long and float, a runtime library to perform 32 bit arithmetic, routines to convert between ASCII and floating point representation, an augmented printf, and a transcendental function library written in C.

Long integers in C/80 are 32 bit signed numbers in the range -2,147,483,648 to 2,147,483,647. Floating point numbers have 24 bits of precision, or about 7 decimal digits, and an exponent in the range 10 to the plus or minus 38th power.

The C/80 MATHPAK requires the C/80 compiler, Version 3.0 or later.

Note: This document describes C/80 implementations for both CP/M and HDOS. Where program names, devices, etc., differ for the two systems, the CP/M names will be used, with the HDOS equivalent in brackets, [like this].

Copyright (c) 1983, 1984 Walter Bilofsky. Sale of this software conveys a license for its use on a single computer at a time, owned or operated by the purchaser. Copying this software or documentation by any means whatsoever for any other purpose is expressly prohibited. C/80 and Toolworks are trademarks of The Software Toolworks.

## 2. THE MATHPAK DISTRIBUTION DISK

The C/80 MATHPAK distribution disk contains the following files:

CCONFIG.COM [or .ABS] Program to patch the C/80 compiler, C.COM [or C.ABS on HDOS], to compile floats and longs. An augmented version of the CCONFIG program supplied with C/80. (Section 3.1.)

FLOATLIB.ASM The C/80 floating point library in assembler form. When the AS assembler is used, this file is automatically included in C/80 programs which use floating point operations. Usually it should reside on A: [SY0: on HDOS] at assembly time. (Section 3.2.)

LONGLIB.ASM The C/80 long arithmetic library in assembler form. Included in AS assemblies of C/80 programs which use long arithmetic.

LGFLTLIB.ASM Common 32 bit arithmetic routines. Included in AS assemblies of C/80 programs which use either floats or longs.

FLIBRARY.REL A relocatable version of the complete float and long arithmetic library, for use with Microsoft's Link-80 or RMAC from Digital Research.

FPRINTF.C The C/80 formatted output routines, augmented for float and long data types. (See Section 5.1.)

FPRINTF.H #Include file for use when fprintf is used with a linking loader. (See Section 5.1.)

MATHLIB.C Transcendental function library, written in C. (See Section 5.5.)

MATHLIB.REL Relocatable version of MATHLIB, for use with a linking loader.

### 3. INSTALLING AND USING THE MATHPAK.

#### 3.1. Installing the MATHPAK.

To install the MATHPAK, run the CCONFIGF program and tell it to patch the C/80 compiler file C.COM [or C.ABS]. The menu of configurable options will be displayed, including a line which reads:

N: Floats and longs available: NOT AVAILABLE

Type the letter N and press RETURN, to change this option to AVAILABLE. Then type Y and RETURN to make the change in the compiler.

To use the scanf function with float or long conversions, you will need to edit file SCANF.C (supplied with C/80). See Section 5.3.

#### 3.2. Using the MATHPAK with AS.

If you use the AS assembler, copy the three .ASM library files from the MATHPAK distribution disk to your A: disk [HDOS: SY0:] (or to the disk on which you keep CLIBRARY.ASM from the C/80 distribution disk). Then compile and assemble your C/80 programs just as before. During assembly of programs which use floats and longs, the required library files will automatically be assembled in.

If you do not have room on your A: disk for the .ASM library files, remember that the compiler can be told to look on another disk for the library files. This is done by the -v switch when running the compiler, or by using the CCONFIGF program to patch the compiler; see the C/80 manual.

#### 3.3. Using the MATHPAK with Macro-80 or RMAC.

If you use either the Macro-80 or RMAC relocatable assembler, copy FLIBRARY.REL to a convenient disk. During the load process, load it with the S switch, so that it is selectively loaded as a library. For example, the following commands will compile, assemble with Macro-80 and load a program from file TESS.C, assuming that all files are on the current drive:

```
c -m tess
m80 =tess
l80 tess,flibrary/s,clibrary,tess/n/e
```

You will probably wish to use Microsoft LIB-80 to create a single library, combining FLIBRARY.REL and CLIBRARY.REL, and perhaps FPRINTF.REL and your own personal library files. Be sure to load FLIBRARY.REL and CLIBRARY.REL last, in that order.

Because of object program size considerations (as noted

in the following section), when compiling programs that do not use floats or longs you may want to use the integer version `printf` provided with your C/80 distribution. You may include `.REL` files generated from both `PRINTF.C` and `FPRINTF.C` in your library; the global symbols in each are distinct. Including `PRINTF.H` or `FPRINTF.H` in your program source file will determine which version of `printf` is linked.

A similar distinction may be made with `scanf`, but there is not a unique floating point version of `SCANF.C`, so you will have to prepare one yourself.

### 3.4. Object Program Size Considerations.

When AS is used, object programs which use longs will contain about 1K bytes of long runtime library (in addition to the 1.9K required by the basic C/80 library). Floats will require about 2.3K bytes. Because a few routines are common to both, use of both floats and longs requires 2.9K bytes of library.

When a relocatable assembler is used, the linking loader is capable of loading only those portions of the float and long libraries which are actually required. Thus, you can expect a somewhat smaller overhead for library routines when using RMAC or Macro-80. We recommend that one of these assemblers be used for large projects or serious software development.

#### 4. ADDITIONS TO THE C/80 LANGUAGE.

With the MATHPAK installed, the only change to the C/80 language is that it recognizes the long, float and double keywords in declarations, and long and float constants. Although double is recognized, it is just a synonym for long.

Long constants are (1) a decimal constant outside the range -32768 to 32767, or (2) an octal or hexadecimal constant outside the range 0 to 65535, or (3) any constant which has the letter 'L' or 'l' immediately following it. Examples of long constants are:

```
32768
31415926
0L
0xabcd1
```

A floating constant consists of a decimal number, with an optional decimal point, followed by an optional exponent (power of ten scaling factor) consisting of the letter 'E' or 'e', perhaps a sign, and an integer number. Either the exponent or the decimal point may be omitted, but not both. The value of the constant is the number times ten to the exponent power. Examples of floating constants are:

```
3.1415926
1e12
.712
0.
2.71828e-31
```

The last number is 2.71828 times 10 to the -31 power.

With the C/80 MATHPAK, the compiler will accept long and float data types almost anywhere char and int are allowed: arrays, pointers, structure elements, and initializers for static data. The only restriction is that constant expressions involving floats or longs are not evaluated at compile time.

## 5. ADDITIONS TO THE FUNCTION LIBRARY.

### 5.1. Printing Floats and Longs.

The `printf` supplied with the MATHPAK is an extension of the version supplied with C/80. It contains conversions to print float and long data types. These conversions are compatible with the description of `printf` in The C Programming Language, by Kernighan and Ritchie. For conversion of values other than float and long, see the C/80 manual.

Longs can be converted by prefixing `l` (lower case 'L') immediately before the format conversion. The conversions for longs are:

|                  |                               |
|------------------|-------------------------------|
| <code>%ld</code> | Long decimal notation         |
| <code>%lo</code> | Long octal notation           |
| <code>%lx</code> | Long hexadecimal notation     |
| <code>%lu</code> | Long unsigned (but see below) |

Due to a limitation in the arithmetic library, unsigned long conversion is not really supported, but is taken to be the same as signed long decimal conversion.

Floating point conversion is determined by the format letter and optional precision. Recall that the conversion consists of a `%`, an optional `-` (left justify), an optional width, an optional `.` and precision, and a conversion letter. Floating point numbers are printed using the conversions:

|                 |   |
|-----------------|---|
| <code>%e</code> | Exponent: <code>[-]d.ffffffe[+/-]xx</code>                        |
| <code>%f</code> | Floating: <code>[-]ddd.ffffff</code>                              |
| <code>%g</code> | The shorter of the <code>%e</code> and <code>%f</code> conversion |

The precision, if specified, determines the number of digits in the fractional part (shown as "ffffff" above). The default precision is 6. The `[-]` indicates an optional minus sign; `[+/-]` is either a `+` or `-`. Rounding is done in the last digit printed. Non-significant zeros are printed in `%e` and `%f` formats, but not in `%g`. The number 0 is always printed as 0, to distinguish it from a very small nonzero quantity.

When printing floating point numbers, remember that only about seven digits of significance are available.

## 5.2. Using Printf Without Floats and Longs

If your program uses `printf`, the float and long arithmetic in `printf` will force the float and long library routines to be loaded with your program, even if you don't use them. This will make your program bigger, and, if you use AS, slower to assemble. Since long divide is relatively slow, decimal output will also take longer.

You can avoid this by `#define`-ing the constants `NOFLOAT` and/or `NOLONG`, either in the file `FPRINTF.C`, or in your source file before including `FPRINTF.C`. The `#define` lines for these constants are included in file `FPRINTF.C`, so all you have to do is remove the initial `/*` on each line.

If `NOFLOAT` is defined, no floating point conversions or arithmetic will be used by `printf`. Similarly, defining `NOLONG` suppresses long conversions and arithmetic.

If you prefer, you can simply use the version of `printf` supplied with your original C/80 compiler when floats and longs are not required.

Users of Macro-80 or RMAC can select the proper version automatically from a relocatable library; see Section 3.3.

## 5.3. Using Scanf with Floats and Longs

The file `SCANF.C` (supplied with C/80) has the same `#defined` symbols `NOFLOAT` and `NOLONG` as in file `FPRINTF.C`. As supplied, these lines are commented out. In order to use `scanf` with float and/or long conversions, you will need to comment out the `NOFLOAT` and/or `NOLONG` lines, respectively.

## 5.4. Float and String Conversion

The following routines are provided in the floating point library to convert between string and floating point representations of float values:

`atof(s)` - `S` is a string containing the representation of a floating point constant. `Atof` returns the float value of the constant.

`ftoa(fmt,digits,f,where)` - Converts the floating point number `f` to a string in the char array `where`. `Fmt` is the character `'e'` or `'f'` to specify exponential or floating conversion, or `'E'` or `'F'` to specify suppression of trailing zeros. `Digits` is the number of digits after the decimal place. Note that as many as `digits+45` bytes may be needed to store a very large or very small number in `'f'` representation.

### 5.5. Transcendental Function Library.

The file MATHLIB.C contains the following functions. Note that all values are computed using 7 significant digits, so the accuracy of the results may be no more than about 6 significant digits. All arguments and results are floating point. Arguments outside the proper range for a function (i.e.,  $\text{sqrt}(-1.0)$ ) will produce undefined results. Angles are represented in radians. E is the natural log base (approximately 2.71828...).

|                   |                     |
|-------------------|---------------------|
| $\sin(f)$         | sine                |
| $\cos(f)$         | cosine              |
| $\text{atan}(f)$  | arc tangent         |
| $\text{sqrt}(f)$  | square root         |
| $\exp(f)$         | e to the f power    |
| $\text{pow}(g,f)$ | g to the f power    |
| $\text{pow}10(f)$ | 10 to the f power   |
| $\ln(f)$          | log base e of f     |
| $\log(f)$         | log base 10 of f    |
| $\text{fabs}(f)$  | absolute value of f |

The file MATHLIB.C contains `#ifneed` directives. Thus, if it is included at the end of your source file, only the functions actually called will be compiled. However, you must declare the functions used to be of type float before first using them, or compile errors will result. An example of such a declaration is:

```
float sin(),pow();
```

Alternatively, you can edit out the `#ifneed` directives and include MATHLIB.C at the beginning of your source file.

Macro-80 or RMAC users may use the file MATHLIB.REL, which contains a relocatable version of MATHLIB. It should be loaded with the S switch so it is searched as a library.



Figure 1. Float and Long Representations  
(32 is floating point sign bit.)

## 6. IMPLEMENTATION OF FLOATS AND LONGS.

This section is presented for advanced programmers who wish to interface C/80 floats and longs to assembly language routines or other programs. It is not necessary to understand this material in order to use the C/80 MATHPAK.

Longs are stored as four bytes, sign extended, with the low order byte first (i.e., lowest memory address), as shown in Figure 1.

Floats are stored as four bytes, with 23 bits of unsigned mantissa in the first three bytes, low order byte first, and one byte of power of two exponent in the fourth byte. The high order bit of the mantissa is always 1 and is not stored. The high bit of the third byte is the sign bit. The exponent is scaled by adding 128; thus an exponent of 127 means a factor of  $2^{-1}$ , 131 means  $2^3$ , etc. The value 0 is represented by a number with exponent byte 0, regardless of the other three bytes.

When floats and longs are used as subroutine arguments, they are pushed on the stack in the same order as stored in memory. That is, the high byte or exponent is pushed first. Subroutines return float or long values in the BCDE registers, with the low byte in E and the high byte or exponent in B, as shown in Figure 1.

If you need to know how to call the C/80 library routines which perform float and long arithmetic, the best way to discover the calling sequences is to write a simple C/80 program invoking the operations in question, and inspect the code it generates.

|               |      |        |        |        |
|---------------|------|--------|--------|--------|
| In registers: | E    | D      | C      | B      |
| In memory:    | +0   | +1     | +2     | +3     |
| On stack:     | (SP) | (SP+1) | (SP+2) | (SP+3) |

  

|       |             |     |     |              |
|-------|-------------|-----|-----|--------------|
| Long: | Low<br>Byte | ... | ... | High<br>Byte |
|-------|-------------|-----|-----|--------------|

  

|        |             |          |        |              |       |
|--------|-------------|----------|--------|--------------|-------|
| Float: | Low<br>Byte | ...      | S<br>B | High<br>Byte | Expt. |
|        | ...         | Mantissa | ...    |              |       |

Figure 1. Float and Long Representations  
(SB is floating point sign bit.)