JASEM ISSN 1119-8362
All rights reserved

Available Online at
http:// www.bioline.org.br/ja

J. Appl. Sci. Environ. Mgt. 2005
Vol. 9 (3) 87 - 89

Algorithm for Output of Floating-Point Numbers in Fixed-Point Form

NDEEKOR, C B

Maths /statistics/Computer Science Department, Faculty of Science, University of Port Harcourt Nigeria. E-mail: clemtoes@yahoo.com

ABSTRACT: Input/output of data items and information are very common operations performed on computer systems. Various data types require special ways to perform these operations on them. Floating-point numbers are no exceptions. Presented in this paper is an algorithm with which floating-point numbers can be converted to their American Standard Code for Information Interchange (ASCII) equivalents in fixed-point form ready for output. The algorithm is so written that it can be implemented easily and requires just the address of the buffer to contain the ASCII equivalent and the number of fractional digits desired in the result. @JASEM

Input/output of data and results into/from the computer system are very important operations that are performed in most programs. The program written in high level programming language indirectly calls up input/output routines through appropriate input/output statements to perform these operations. The programmer in assembly language has to directly call up routines to perform input/output or writes his own routines (Brey, 1998; Detmer, 1990 and Sanchez, 1990).

The algorithm presented enables the conversion of a floating-point number (in its internal form) to a string of American Standard Code for Information Interchange (ASCII) in fixed-point form ready for output. The output may be done with an appropriate service function (Norton, 1995). The user decides the number of fractional digits to appear in the result and provides a buffer that will hold the ASCII string. This contrasts with what is obtained in existing output routines where the user must specify the width as well as number of fractional digits desired, if the output is formatted. Invariably, the user is forced to provide the number of positions for the whole-number part of the output; but this is unnecessary since the number of digits that make up the whole number part of a floating-point number in fixed-point form is always fixed.

The output from this algorithm is in the form described by the Extended Backus Naur Form (EBNF):

FP = ["-"]{Digit}"."{Digit}.
Digit = "0"/"1"/"2"/"3"/"4"/"5"/"6"/"7"/"8"/"9".

where the minus sign is optional (appears for a negative number). The first recurring Digit represents the whole number part of the floating-point number while the second represents the fractional part. The number of digits that can appear in the whole number as well as the fractional parts are limited by restrictions in the floating-point number data format adopted.

MATERIALS AND METHODS

Below is an algorithm that achieves the aims stated in the introduction above.

Step1: Initialize appropriate variables
NDORM = 0
ROUNDER = 0.5
STRING_POINTER = First byte of Output Buffer
N = Supplied number of fractional digits desired
VALUE = Number to be converted to ASCII characters
Step 2: Divide ROUNDER by 10.0 (N) times.
Step 3: Compare VALUE with 0.0
If (VALUE < 0.0) then
Output minus '-' sign to buffer
Increment STRING-POINTER
Negate VALUE
Elseif (VALUE = 0.0) then

Go to step 6

End if
Step 4: Add ROUNDER to VALUE
Step 5: Compare VALUE with 10.0
If (VALUE > = 10.0) then
While (VALUE > = 10.0) do
Divide VALUE by 10.0
Increment NDORM
Endwhile
Elseif (VALUE < 1.0) then
While (VALUE < 1.0) do
Multiply VALUE by 10.0
Decrement NDORM
Endwhile
Endif.
Step 6: Compare NDORM and 0
If (NDORM < 0) then
Output '0' to output buffer
Increment STRING-POINTER
Output '.' to output buffer
Increment STRING-POINTER

**Corresponding*: E-mail: clemtoes@yahoo.com

```
If (NDORM ≠ -1)then
 Negate NDORM
Decrement NDORM
Repeat until NDORM is 0
Output '0' to output buffer
 Increment STRING-POINTER
Decrement NDORM
End repeat
Else
Nagate NDORM
Endif
Repeat until N is 0
Obtain integer part (INT_PART) of VALUE
Convert INT_PART to ASCII equivalent
 Store ASCII equivalent in output buffer
 Increment STRING-POINTER
 Subtract INT-PART from VALUE
 Multiply VALUE by 10.0
 Decrement N
        End repeat
        Else
COUNTER1 = 1
COUNTER2 = (N + NDORM + 1)
     Repeat until COUNTER2 = 0
If (COUNTER1 = NDORM + 2) then
Store '.' in output buffer
Increment STRING-POINTER
        Endif
Obtain INT_PART of VALUE
Convert INT_PART to ASCII equivalent
Store ASCII equivalent in output buffer
Increment STRING_POINTER
Subtract INT-PART from VALUE
        Multiply VALUE by 10.0
        Increment COUNTER1
        Decrement COUNTER2
        Endif
```

Step 7: Store terminating character ($) in buffer
Step 8: Stop.

RESULTS AND DISCUSSION

In the above algorithm, a ROUNDER is used to take care of rounding error. The final value of ROUNDER (originally set to 0.5) is determined by the number of fractional digits desired by the user.

Step 1 initializes some variables. These variables and what they stand for are:

NDORM:	Number of divisions or multiplications done on the floating-point number to be converted to make it have only one digit (that is non-zero) before decimal point.
COUNTER1:	Used in determining the position o f decimal point in the result.
COUNTER2:	Used in determining number of digits to be in the output.
ROUNDER:	As explained above.
STRING_POINTER:	A pointer to the buffer that will contain the output.
N :	This stands for the supplied desired number of fractional digits.
VALUE:	The floating-point number to be converted

Step 2 divides ROUNDER by 10.0 as many times as the number of fractional digits desired in the result. This gives the eventual value of ROUNDER that is used to round off the floating-point number to be displayed.

In *step 3*, the floating-point number to be converted is checked against 0.0 to know if it is a negative or positive number. If negative, minus sign is output to the output buffer, STRING_POINTER incremented and the original VALUE negated. However, if the floating-point number is zero, control goes to step 6.

Step 4 adds ROUNDER to the floating-point number.

Step 5 compares VALUE (floating-point number to be converted) against 10.0. If it is greater than or equal to 10.0, it is repeatedly divided by 10.0 until VALUE is less than 10.0. At each division, NDORM is incremented by 1. However, if VALUE is not greater than or equal to 10.0 and it is less than 0.1, it is repeatedly multiplied by 10.0 until it is greater than or equal to 1.0. At each multiplication, NDORM is decremented by 1 (Detmer, 1990).

In *step 6*, the digits of the floating-point number are outputted to the output buffer. The value of NDORM is tested against 0 to know if it is greater than or equal to it. If it is not, the implication is that the original floating-point number had leading zero digits before and after decimal point (recall the continuous multiplication by 10.0 in step 5). So the leading '0' is output to the output buffer followed by a point ('.'). The next test and the actions that follow (NDORM tested against -1) determine the number of '0's' to be output before the first non-zero digit. When NDORM is equal to -1, there will be no leading non-zero digit after decimal point but if NDORM is less than -1, then there will be as many as (-NDORM -1) leading '0's' before the first non-zero digit. After storing the leading zeros, other digits are

stored according to the desired number of fractional digits.
If NDORM is not less than 0, then the floating-point number is output containing (NDORM + N + 1) ASCII characters. The position of decimal point is at (NDORM + 2).

Step 7 stores the terminating character ($) while step 8 ends the algorithm.

Conclusion: In this work, an algorithm with which an internal floating-point number can be converted to fixed-point form requiring just the desired number of fractional digits and the output buffer is developed. The algorithm can be used to converted floating-point numbers declared with the Institute of Electrical and Electronic Engineers (IEEE) floating-point data formats or any possible data format adopted. The algorithm can also be implemented for customized applications. The dollar sign is used to terminate the obtained ASCII string.

Acknowledgement: I am indepted to Dr O Owolabi and Dr I. U. Mbeledogu for their contributions to the research work that lead to this paper.

REFERENCES

Sanchez, J [1990]: Assembly language Tools And Techniques For The IBM Microcomputers. Prentice-Hall, Inc., New Jersey.

Detmer R C [1990]: Fundamentals Of Assembly Language Programming Using The IBM PC And Compatibles. D. C. Heath And Company, Massachusetts.

Norton P [1995]: Programmer's Guide To The IBM PC. Microsoft Press, USA.

Brey B B [1998]: Embedded Controllers 80186, 80188, And 80386EX. Prentice-Hall, Inc., New Jersey.