A Computer Programming Method for Handling Different Data Types

by

Brett Bennett

Kansas Geological Survey
1930 Constant Avenue
Lawrence, Kansas 66047-3726

Open-file Report #92-40

December 1992

INTRODUCTION

When a data processing program needs to be able to handle various data type, i.e. integer data or floating point data, the programmer is presented with the problem of being required to convert the data correctly to the internal format used in his programs. The data sets used with the *Eavesdropper* seismic processing package present such a problem. One approach to this problem is for the programmer to create several versions of a given program, each expecting one particular type of input data. This is the straightforward approach, and the easiest to code. There are however, several drawbacks to this approach. One major drawback is maintaining the many different copies of the same program. Any updates or upgrades must be done to all versions of the program. Another major drawback is from the user's point of view. When different versions expect different types of data, the user must correctly match data with the program. A better approach, especially from the user's point of view, is a single program that can handle all pertinent data types. This paper presents a solution to this programming problem.

METHOD

The first thing required in a data set is a 'flag' that can be used by the program to identify the data type. This was done with *Eavesdropper* data by assigning an unused trace header word (#60) a value to represent either integer data or floating point data. The value 55AA (hex) was chosen to be the flag for floating point data, all other values indicate integer data. This was necessary because some early integer data sets created for *Eavesdropper* had random values in this location and could not be relied upon to have a set value. The odds that such a data set would have the value 55AA is only 1 in 65576, so in the other 65575 cases the program would correctly convert the integer data. Establishing this up front allows the program to identify the data type at run time.

The type of data determines how many bytes need to be read for each data point. In the case of integer data, 2 bytes represent a data point, while floating point data uses 4 bytes per data point. Generally speaking, the data would also need to be read into one of two different data buffers depending on its type, and then could be converted to the program's internal format. This method would require data type checking at each READ statement in the program and the waste of memory on a buffer not being used for a given data set. Listing 1 shows a C program fragment that demonstrates a method that will avoid these problems.

The first two lines of the C fragment initialize variables used to read input data. INBUFFER is a floating point array used to receive incoming floating point data. INTINBUFFER is an integer pointer. It has no memory allocated to it, it simply points to some other memory defined in the next statement. The third line performs the 'trick' of this method. INTINBUFFER is assigned to the same

location as the floating point buffer INBUFFER. Doing this allows for the same buffer to do double duty.

The next 5 lines read in the trace header of the current data trace. (Trace headers are of fixed length and format.) The program then inspects the 60th ([59] in C language) element of the trace header. If it finds the flag (55AA) it assigns a value of 4 to a variable called DATATYPE, otherwise it assigns a value of 2 to this variable. This operation only needs to be performed once. The value 2 or 4 is then used to inform subsequent READ statements as to the nature of the data present. The 9th line show the usage of DATATYPE in a FREAD statement. Notice that FREAD always loads the data into the floating point buffer INBUFFER. If the data is indeed floating point, then the program is free to begin using the data, since it is already correctly installed in memory. If the data is integer, then a final step is necessary. The last 4 lines of code fragment show how a conversion from integer to floating point maybe performed using a single data buffer.

DISCUSSION

The trick here is converting the data from the bottom up, rather than from the top down. Notice the FOR loop starts with the number of samples read and decrements towards 0. This insures that longer floating point representation of the integer data never over-writes unconverted data, and at the same time saves memory by using only a single memory buffer. Using this method presents a rare case where floating point data is handled faster than integer data, insofar as no conversion of the data is necessary when floating point is the input type.

In this particular case the internal format needed for the data is floating point. If another type were required (e.g., double precision), the same method could be used. The program need only correctly 'type cast' the input buffer to handle the largest case, and convert accordingly. The method breaks down when the internal format is a smaller type than the possible input types. For instance: integer internal format and floating point input (such a conversion has other difficulties beyond data length, and is generally avoided in practice anyway).

CONCLUSIONS

Presented is one method of handling different input data types with a single program. This method may be found especially useful in the case where an existing program is given the task of reading data with more bytes/elements than it was originally designed for. Use of this method saves memory and instructions when used as described.

```
float inbuffer[16000];        //declare floating point input buffer

int *intinbuffer;             //declare an integer pointer

intinbuffer=&inbuffer[0];  //set the integer pointer = floating point buffer

in2 = fread(trhd,2,120,in);  //read in the trace header

if( trhd[59] != 0x55aa )

datatype=2;

else

datatype=4;
 //set the datatype variable to the number of bytes per sample.  Need only do this once.


in2=fread(inbuffer,datatype,trhd[57],in);
//Use  datatype to control the number of bytes per sample read

if (datatype==2)
//if data type is integer then convert it to floating point, useing the same buffer
{
int jj;  //temp counter integer

for(jj=trh[57]-1;jj>=0;jj--)  //start a bottom an work up

inbuffer[jj]=(float)intinbuffer[jj]; //type cast into floating point
}
```

Listing 1.