

Bent Petersen 351u2001-assignment-02.tex

Your assignment is to compute the unit round for various floating point data type for various compilers for various programming languages. I have provided some sample c code below, but there is no reason to restrict yourself to c.

A direct implementation of the definition of the unit round leads to the simple (overly simple it turns out)

```
/*
** eps1.c
** Mth 351 Spring 2001 - Bent Petersen
**
** Microsoft C/C++ cl 32-bit 12.00.8168      cl /Od eps1.c
** GNU C/C++ gcc egcs-2.91.66 (egcs 1.1.2)  gcc -o eps1 -i eps1.c
**
** This code should compute the float unit round, but it
** will probably fail because the compiler does the
** calculations at the full precision of the FPU or, at
** any rate, at too high precision.
**
** Refer to eps2.c for a possible work around.
*/

#include <stdio.h>
#include <float.h>

/* gcc wants main() to return an int - MS cl doesn't care. */

int main( void ) {
    float x1=1.0;

    while ( 1 + x1 > 1.0 ) {
        x1 /= 2.0;
    }
    printf("\nUnit round - float: %e\t(float.h claims %e)",
        2*x1, FLT_EPSILON );

    /* Return something to shut up the compiler. */

    printf("\n");
    return 0;
}
```

If we compile this code with GNU gcc on i386 Linux and run the executable produced we obtain the output

```
Unit round - float: 1.084202e-19          (float.h claims 1.192093e-07)
```

This result is obviously incorrect! Your result may differ since the compiler will probably generate code to do the calculation at the full precision of the floating point unit, or at any rate, at some higher precision than requested, and that will vary.

In order to obtain the correct value of unit round for each floating point data type, we have to force storing the results of intermediate calculations by assigning the values to a variable of the appropriate type – see the code below. Even this trick may fail to work unless the compiler's optimization routines are turned off. Usually aggressive optimization is off by default, but you may want to check the documentation.

Here is the “corrected” code (with the uncorrected code included for easy comparison):

```
/*
** eps2.c
** Mth 351 Spring 2001 - Bent Petersen
**
** Microsoft C/C++ cl 32-bit 12.00.8168      cl /Od eps2.c
** GNU C/C++ gcc egcs-2.91.66 (egcs 1.1.2)  gcc -o eps2 -i eps2.c
**
** This code looks pretty silly but computing the unit
** round (epsilon) is a bit tricky. The obvious code may
** not work, or may work but yield incorrect results. It
** is necessary to force stores of intermediate results to
** prevent the compiler from doing the calculations at the
** full precision of the FPU or, at any rate, at too high
** precision.
*/

#include <stdio.h>
#include <float.h>

int main( void ) {

/* gcc wants main() to return an int - void main(void) produces */
/* a warning message. MS cl doesn't care. */

    float x1=1.0, z1=2.0;
    double x2=1.0, z2=2.0;
```

```

long double x3=(long double)1.0, z3=(long double)2.0;
float x4=(float)1.0;

while ( z1 > 1.0 ) {
    x1 /= 2.0;
    z1 = 1.0 + x1;
}
printf("\nUnit round - float: %e\t(float.h claims %e)",
        2*x1, FLT_EPSILON );

while ( z2 > 1.0 ) {
    x2 /= 2.0;
    z2 = 1.0 + x2;
}
printf("\nUnit round - double: %e\t(float.h claims %e)",
        2*x2, DBL_EPSILON );

while ( z3 > 1.0 ) {
    x3 /= 2.0;
    z3 = 1.0 + x3;
}
printf("\nUnit round - long double: %e\t(float.h claims %e)",
        2.0*(double)x3, (double)LDBL_EPSILON );

/* In float.h for gcc LDBL_EPSILON is declared as a long double */
/* and therefore the cast to a double is required for printf(). */
/* Likewise x3 must be cast to a double or printf() will print */
/* nonsense. It is a bit curious since LDBL_EPSILON easily fits */
/* in a double. */

while ( 1.0 + x4 > 1.0 )
    x4 /= 2.0 ;

printf("\nUnit round - %e\t\t(bogus, float without stores)\n",
        2.0*x4 );

printf("\n");
return 0;
}

```

If we compile this code with a recent version of GNU gcc on i386 Linux and run the executable produced we obtain the output

```
Unit round - float: 1.192093e-07          (float.h claims 1.192093e-07)
```

```
Unit round - double: 2.220446e-16      (float.h claims 2.220446e-16)
Unit round - long double: 1.084202e-19 (float.h claims 1.084202e-19)
Unit round - 1.084202e-19              (bogus, float without stores)
```

The first three lines agree precisely with the IEEE Standard 754 for short real (single precision real), long real (double precision real), and temporary real (extended precision real). The results, however, may differ with other compilers or other systems. Also for other languages you may not have the equivalent of `float.h` available and you may have to look at the documentation, if available, to find out how floating point is implemented.

Turn in a copy of your code (if different from mine) and a copy of your results. For each result indicate the compiler used and the type of machine you ran it on. There are many different computers available to you on campus, so try several different types.

Include some discussion of your results. For c compilers alone, you may see some surprises in the implementation of long doubles.

The code `eps2.c` for the second routine above may be downloaded from my web page if you want to use it.

Compile and run code successfully	25 pts.
Write your own code (including spreadsheet)	5 pts.
Results including identification of computer and compiler	10 pts.
Discussion of results, cite docs and manuals, etc.	10 pts.

You should have a least two sets of results – different compilers, different programming languages, or significantly different types of computers. Have fun!

### Remarks:

- If you use the Microsoft Visual C/C++ compiler try

```
cl /Ox eps2.c
```

(upper case O) to get a surprise. You may wish to look up the meanings of the switches `/Od` and `/Ox` to figure out what is going on here.

- If you use a recent version of the GNU gcc compiler try the `-O1` switch (or `-O2`, or `-O3`), that is,

```
gcc -O1 -o eps2 -i eps2.c
```

(upper case O) to get a surprise. In recent versions of Linux you can use `info gcc` to get detailed documentation on gcc. You can also use `man gcc` but the documentation may be out of date, excessively concise, or awkward to use.

- On recent Linux systems try `info gcc` and read the documentation on `-ffloat-store` for insightful information relevant to what we are exploring here.