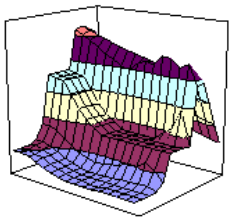

A Note on TMin

Randal E. Bryant and *David R. O'Hallaron*

CS:APP Home Page

Summary

Due to one of the rules for processing integer constants in ANSI C, the numeric constant `-2147483648` is handled in a peculiar way on a 32-bit, two's complement machine.

The problem can be corrected by writing `-2147483647-1`, rather than `-2147483648` in any C code.

Description of Problem

The ANSI C standard requires that an integer constant too large to be represented as a signed integer be ``promoted" to an unsigned value. When GCC encounters the value 2147483648, it gives a warning message: ```warning: decimal constant is so large that it is unsigned`." The result is the same as if the value had been written `2147483648U`.

The compiler processes an expression of the form *-X* by first reading the expression *X* and then negating it. Thus, when the C compiler encounters the constant `-2147483648`, it first processes `2147483648`, yielding `2147483648U`, and then negates it. The unsigned negation of this value is also `2147483648U`. The bit pattern is correct, but the type is wrong!

Writing TMin in Code

The ANSI C standard states that the maximum and minimum integers should be declared as constants `INT_MAX` and `INT_MIN` in the file `limits.h`. Looking at this file on an IA32 Linux machine (in the directory `/usr/include`), we find the following declarations:

```
/* Minimum and maximum values a `signed int' can hold.  */
#define INT_MAX 2147483647
#define INT_MIN (-INT_MAX - 1)
```

This method of declaring `INT_MIN` avoids accidental promotion and also avoids any warning messages by the C compiler about integer overflow.

The following are ways to write TMin_32 for a 32-bit machine that give the correct value and type, and don't cause any error messages:

- `-2147483647-1`
- `(int) 2147483648U`
- `1<<31`

The first method is preferred, since it indicates that the result will be a negative number.

Randy Bryant and Dave O'Hallaron
Last modified: Sun Apr 6 20:34:32 EDT 2003