Some Computers will not work after 2038…

Problems & solutions
Regarding Date and Time

Y2K problem

The **Year 2000 problem** (also known as the **Y2K problem**, the **millennium bug**, the **Y2K bug**, or simply **Y2K**) was a notable computer bug resulting from the practice in early computer program design of representing the year with two digits. This caused some date-related processing to operate incorrectly for dates and times on and after January 1, 2000 and on other critical dates which were billed "event horizons". This fear was fueled by the attendant press coverage and other media speculation, as well as corporate and government reports. People recognized that long-working systems could break down when the "...97, 98, 99..." ascending numbering assumption suddenly became invalid. Companies and organizations world-wide checked and upgraded their computer systems.

Reasons:

1. In the 1960s, computer memory was scarce and expensive, and most data processing was done on punch cards which represented text data in 80-column records. Programming languages of the time, such as COBOL and RPG, processed numbers in their ASCII or EBCDIC representations. They occasionally used an extra bit called a "zone punch" to save one character for a minus sign on a negative number, or compressed two digits into one byte in a form called binary-coded decimal, but otherwise processed numbers as straight text. Over time the punch cards were converted to magnetic tape and then disk files and later to simple databases like ISAM, but the structure of the programs usually changed very little.

2. Saving two characters for every date field was significant in the 1960s. Since programs at that time were mostly short-lived affairs programmed to solve a specific problem, or control a specific hardware setup, neither managers nor programmers of that time expected their programs to remain in use for many decades. The realization that databases were a new type of program with different characteristics had not yet come, and hence most did not consider fixing two digits of the year a significant problem.

Solution

There were exceptions, of course; the first person known to publicly address the problem was Bob Bemer who had noticed it in 1958, as a result of work on genealogical software. He spent the next twenty years trying to make programmers, IBM, the US government and the ISO aware of the problem, with little result. This included the recommendation that the COBOL PICTURE clause should be used to specify four digit years for dates. This could have been done by programmers at any time from the initial release of the first COBOL compiler in 1961 onwards. **However, lack of foresight, the desire to save storage space, and overall complacency prevented this advice from being followed.**

Storage of a combined date and time within a fixed binary field is often considered a solution, but the possibility for software to misinterpret dates remains, because such date and time representations must be relative to a defined origin. Rollover of such systems is still a problem but can happen at varying dates and can fail in various ways. For example:

> The typical Unix timestamp (time_t) stores a date and time as a 32-bit signed integer number representing, roughly speaking, the number of seconds since January 1, 1970, and will roll over (exceed 32 bits) in 2038 and cause the Year 2038 problem. To solve this problem, many systems and languages have switched to a 64-bit version, or supplied alternatives which are 64-bit.

Year 2038 problem

The year 2038 problem (also known as "Unix Millennium bug", or "Y2K38" by analogy to the Y2K problem) may cause some computer software to fail before or in the year 2038. The problem affects all software and systems that store system time as a signed 32-bit integer, and interpret this number as the number of seconds since 00:00:00 January1,1970.The latest time that can be represented this way is 03:14:07 UTC on Tuesday, 19 January 2038. Times beyond this moment will "wrap around" and be stored internally as a negative number, which these systems will interpret as a date in 1901 rather than 2038. This will likely cause problems for users of these systems due to erroneous calculations.

Range for 32 bit processor:

***Signed:* −2,147,483,648 to +2,147,483,647**

***Unsigned:* 0 to +4,294,967,2**

For the uninitiated, `time_t` is a data type used by C and C++ programs to represent dates and times internally. `time_t` is actually just an integer, a whole number, that counts the *number of seconds* since January 1, 1970 at 12:00 AM Greenwich Mean Time. A `time_t` value of 0 would be 12:00:00 AM (exactly midnight) 1-Jan-1970, a `time_t` value of 1 would be 12:00:01 AM (one second after midnight) 1-Jan-1970, etc..

Date & time	`time_t` representation
1-Jan-1970, 12:00:00 AM GMT	0
1-Jan-1970, 12:00:01 AM GMT	1
1-Jan-1970, 12:01:00 AM GMT	60
1-Jan-1970, 01:00:00 AM GMT	3 600
2-Jan-1970, 12:00:00 AM GMT	86 400
3-Jan-1970, 12:00:00 AM GMT	172 800
1-Feb-1970, 12:00:00 AM GMT	2 678 400
1-Mar-1970, 12:00:00 AM GMT	5 097 600
1-Jan-1971, 12:00:00 AM GMT	31 536 000
1-Jan-1972, 12:00:00 AM GMT	63 072 000
1-Jan-2003, 12:00:00 AM GMT	1 041 379 200
1-Jan-2038, 12:00:00 AM GMT	2 145 916 800
19-Jan-2038, 03:14:07 AM GMT	2 147 483 647

By the year 2038, the `time_t` representation for the current time will be over 2 140 000 000. And that's the problem. A modern 32-bit computer stores a "signed integer" data type, such as `time_t`, in 32 bits. The first of these bits is used for the positive/negative sign of the integer, while the remaining 31 bits are used to store the number itself. The highest number these 31 data bits can store works out to exactly 2 147 483 647. A `time_t` value of this exact number, 2 147 483 647, represents January 19, 2038, at 7 seconds past 3:14 AM Greenwich Mean Time. So, at 3:14:07 AM GMT on that fateful day, every `time_t` used in a 32-bit C or C++ program will reach its upper limit.

By: Er. Deepak Balana
Lecturer E.C. Deptt.