COMS20012: Integer Overflow

Joseph Hallett

bristol.ac.uk

What is all this about then?

- CPUs have registers
- Patterns in bits in registers represent stuff

```c
#include <stdio.h>

int main(void) {
  unsigned int x = -1;
  void *ref = (void *)&x;

  printf("Unsigned int: %u\n", *((unsigned int *)ref));
  printf("Signed int:   %d\n", *((signed int *)ref));
  printf("Float:        %f\n", *((float *)ref));
  printf("Double:       %lf\n", *((double *)ref));
  printf("Pointer:      %p\n", *((void **)ref));

  return 0;
}
```

bristol.ac.uk

It’s just a representation…

```c
#include <stdio.h>

int main(void) {
  unsigned int x = -1;
  void *ref = (void *)&x;

  printf("Unsigned int: %u\n", *((unsigned int *)ref));
  printf("Signed int:   %d\n", *((signed int *)ref));
  printf("Float:        %f\n", *((float *)ref));
  printf("Double:       %lf\n", *((double *)ref));
  printf("Pointer:      %p\n", *((void **)ref));

  return 0;
}
```

```
Unsigned int: 4294967295
Signed int:   -1
Float:        nan
Double:       0.000000
Pointer:      0xffffffff
```

Not Quite Integers

- What a CPU (typically) regards as an integer isn't really an integer in a purely mathematical sense
- A *native* CPU integer typically has *limits* dependent on what the ISA supports
  - (some ISAs support some really weird data types… e.g. BCD)
- Operating systems again impose limits on integers
  - see `limits.h` in your OS libraries and POSIX standards

Binary and 2s Complement Notation

- *Usually* unsigned integers are represented as standard binary
  - unsigned integers typically go from 0 to $2^{wordsize}-1$
  - So for an 8 bit CPU: 0 to 2^8-1 = 255
  - For a 32 bit CPU: 0 to $2^{32}-1$ = 4,294,967,295
- *Usually* signed integers are represented in 2s complement
  - Highest bit's value is negated
  - So $-2^{(wordsize-1)}$ to $2^{(wordsize-1)}-1$
  - So for an 8 bit CPU: -2^7 to 2^7-1 = -128 to 127
  - For a 32 bit CPU: -2^{31} to $2^{31}-1$ = -2,147,483,648 to 2,147,483,647
- So what happens when you go beyond these limits?

What's next?

```
#include <stdio.h>

int main(void) {
  unsigned int x = -1;
  void *ref = (void *)&x;

  printf("Unsigned int: %u\n", *((unsigned int *)ref)+1);
  printf("Signed int:   %d\n", *((signed int *)ref)+1);
  printf("Float:        %f\n", *((float *)ref)+1);
  printf("Double:       %lf\n", *((double *)ref)+1);
  printf("Pointer:      %p\n", *((void **)ref)+1);

  return 0;
}
```

bristol.ac.uk

Wraparound and weirdness!

```c
#include <stdio.h>

int main(void) {
  unsigned int x = -1;
  void *ref = (void *)&x;

  printf("Unsigned int: %u\n", *((unsigned int *)ref)+1);
  printf("Signed int:   %d\n", *((signed int *)ref)+1);
  printf("Float:        %f\n", *((float *)ref)+1);
  printf("Double:       %lf\n", *((double *)ref)+1);
  printf("Pointer:      %p\n", *((void **)ref)+1);

  return 0;
}
```

```
Unsigned int: 0
Signed int:   0
Float:        nan
Double:       1.000000
Pointer:      0x100000000
```

bristol.ac.uk

Whole bunch of weird rules

- Expansion from a smaller data type to a larger one *should* work
  - e.g. char to an int
- Contraction from a larger data type to a smaller one may not!
  - e.g. int to char
- Switching between signed and unsigned types…
  - Do you zero extend or one extend?

bristol.ac.uk

Aren’t there meant to be standards?

- The C standard says what’s *supposed to happen*
  - But it is full of edge cases and imprecise

- Compilers can and do differ

- …Which means compilers support bugs from old compilers/standards to avoid breaking 50 year old programs

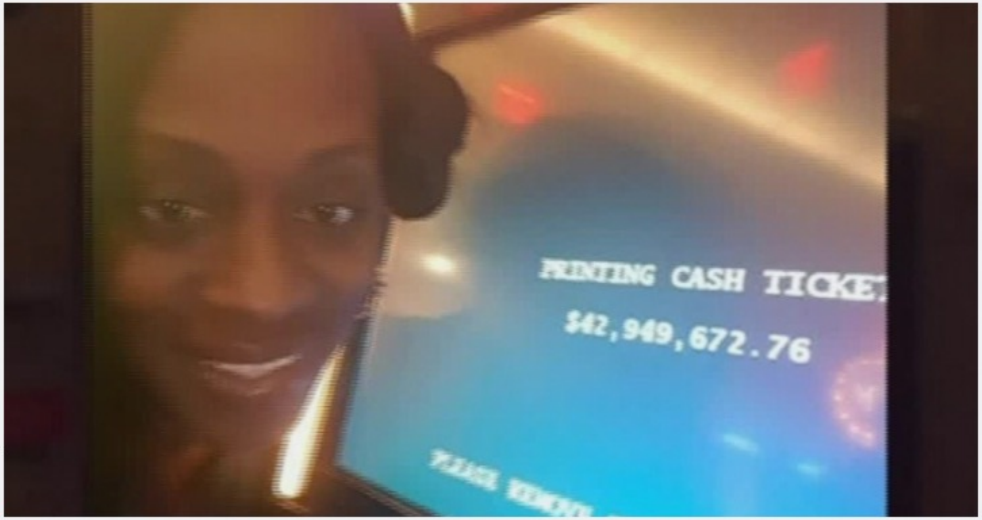
- It’s a mess!

POLICY —

Sorry ma‘am, you didn’t win $43M—there was a slot machine “malfunction”

Casino had apologized "for any inconvenience this may have caused."

DAVID KRAVETS - 6/15/2017, 2:01 PM



Alan Ripka

Enlarge / Katrina Bookman takes a selfie showing she hit the big one.

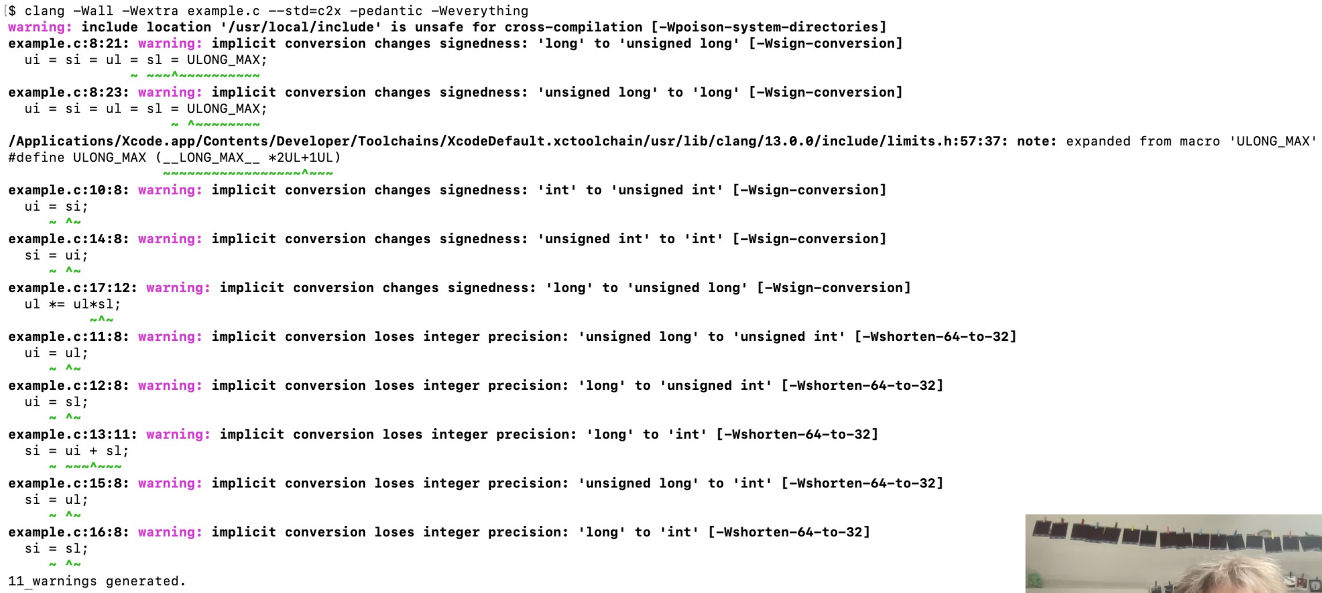
Imagine, if you would, how absolutely giddy you'd be if you won a $43 million jackpot while playing a casino slot machine. You could burn a lot of bridges with that amount of cash.

Preventing overflows

```c
#include <limits.h>
int main(void) {
  unsigned int ui;
  signed int si;
  unsigned long ul;
  signed long sl;

  ui = si = ul = sl = ULONG_MAX;

  ui = si;
  ui = ul;
  ui = sl;
  si = ui + sl;
  si = ui;
  si = ul;
  si = sl;
  ul *= ul*sl;

  return 0;
}
```

```
[$ gcc -Wall -Wextra example.c --std=c2x -pedantic
[$ # ...wat.
```

The C compiler doesn't have enough warnings by default



```
[$ clang -Wall -Wextra example.c --std=c2x -pedantic -Weverything
warning: include location '/usr/local/include' is unsafe for cross-compilation [-Wpoison-system-directories]
example.c:8:21: warning: implicit conversion changes signedness: 'long' to 'unsigned long' [-Wsign-conversion]
  ui = si = ul = sl = ULONG_MAX;
               ~ ~~~^~~~~~~~~~~
example.c:8:23: warning: implicit conversion changes signedness: 'unsigned long' to 'long' [-Wsign-conversion]
  ui = si = ul = sl = ULONG_MAX;
                    ~ ^~~~~~~~~
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/lib/clang/13.0.0/include/limits.h:57:37: note: expanded from macro 'ULONG_MAX'
#define ULONG_MAX (__LONG_MAX__ *2UL+1UL)
                   ~~~~~~~~~~~~~~~~~~^~~~
example.c:10:8: warning: implicit conversion changes signedness: 'int' to 'unsigned int' [-Wsign-conversion]
  ui = si;
     ~ ^~
example.c:14:8: warning: implicit conversion changes signedness: 'unsigned int' to 'int' [-Wsign-conversion]
  si = ui;
     ~ ^~
example.c:17:12: warning: implicit conversion changes signedness: 'long' to 'unsigned long' [-Wsign-conversion]
  ul *= ul*sl;
          ~^~
example.c:11:8: warning: implicit conversion loses integer precision: 'unsigned long' to 'unsigned int' [-Wshorten-64-to-32]
  ui = ul;
     ~ ^~
example.c:12:8: warning: implicit conversion loses integer precision: 'long' to 'unsigned int' [-Wshorten-64-to-32]
  ui = sl;
     ~ ^~
example.c:13:11: warning: implicit conversion loses integer precision: 'long' to 'int' [-Wshorten-64-to-32]
  si = ui + sl;
     ~ ~~~^~~~
example.c:15:8: warning: implicit conversion loses integer precision: 'unsigned long' to 'int' [-Wshorten-64-to-32]
  si = ul;
     ~ ^~
example.c:16:8: warning: implicit conversion loses integer precision: 'long' to 'int' [-Wshorten-64-to-32]
  si = sl;
     ~ ^~
11 warnings generated.
```

bristol.ac.uk

Do your own checking

- Use the limits.h header to find limits
-
```
unsigned int a, b;
if (a < UINT_MAX – b) { return a + b; }
else {
  /* Look maybe you want to use a higher level
programming language if this is likely to be an
issue.  There are a bunch of libraries and non-
standard ways of dealing with this but at the end
of the day you have better things to be doing
with your lives.
*/
return -1; // ;-)
```

bristol.ac.uk