Computer Labs: C Integer Conversions

2º MIEIC

Pedro F. Souto (pfs@fe.up.pt)

September 30, 2012

C Integer Conversion Rules

- C supports different integer types, which differ in their:

  Signedness i.e. whether they can represent negative numbers
  Precision i.e. the number of bits used in their representation
- The C standard specifies a set of rules for conversion from one integer type to another integer type so that:
  - The results of code execution are what the programmer expects
- One such rule is that:
  - Operands of arithmetic/logic operators whose type is smaller than `int` are promoted to `int` before performing the operation

  the rational for this is
  - To prevent errors that result from overflow. E.g:

```
signed char cresult, c1, c2, c3;
c1 = 100;
c2 = 3;
c3 = 4;
cresult = c1 * c2 / c3;
```

source: CMU SEI

Problems Source: CMU SEI

Let:

```
uint8_t port = 0x5a;
uint8_t result_8 = ( ~port ) >> 4;
```

Problems Source: CMU SEI

Let:

```
uint8_t port = 0x5a;
uint8_t result_8 = ( ~port ) >> 4;
```

Question: What is the value of `result_8`?

Problems Source: CMU SEI

Let:

```
uint8_t port = 0x5a;
uint8_t result_8 = ( ~port ) >> 4;
```

Question: What is the value of `result_8`?

Answer: Most likely, you'll think in terms of 8-bit integers:

Expr.	**8-bit**
`port`	`0x5a`
`~port`	`0xa5`
`(~port)>>4`	`0x0a`
`result_8`	`0x0a`

Problems Source: CMU SEI

Let:

```
uint8_t port = 0x5a;
uint8_t result_8 = ( ~port ) >> 4;
```

Question: What is the value of `result_8`?

Answer: ... but because of integer promotion, need to think in terms of `sizeof(int)`:

Expr.	**8-bit**	**32-bit**
`port`	`0x5a`	`0x0000005a`
`~port`	`0xa5`	`0xffffffa5`
`(~port)>>4`	`0x0a`	`0xfffffffa`
`result_8`	`0x0a`	`0xfa`

Problems Source: CMU SEI

Let:

```
uint8_t port = 0x5a;
uint8_t result_8 = ( ~port ) >> 4;
```

Question: What is the value of `result_8`?

Answer: ... but because of integer promotion, need to think in terms of `sizeof(int)`:

Expr.	8-bit	32-bit
`port`	`0x5a`	`0x0000005a`
`~port`	`0xa5`	`0xffffffa5`
`(~port)>>4`	`0x0a`	`0xfffffffa`
`result_8`	`0x0a`	`0xfa`

Solution: One way to fix this is to use a cast on the value after the complement:

```
uint8_t port = 0x5a;
uint8_t result_8 = (uint8_t) ( ~port ) >> 4;
```

This truncates the result of the complement to its LSB, and therefore the right shift works as expected

Further Reading

- INT02-C. Understand integer conversion rules