Trong bài này mình sẽ nói về cách cộng/trừ (+/-) `int`'s trong ngôn ngữ C. Cái mà được trình bài ở đây chỉ là những gì mình hình dung được, có thể có sai sốt; mình cũng đang trên đường học tập; nếu bạn đọc nào tìm ra những chỗ bị sai, có thể liên lạc với mình bằng email.

1 Cách biểu thị `int`

Để dễ cho việc thảo luận, mình sẽ coi `int` như luôn được lưu và xử lý trong máy tính bằng 32 bits. Hơn nữa, binary và decimal integers sẽ được biểu thị như, e.g. số bảy, $(111)_2 = (0\cdots0111)_2$ và 7, resp.

Thử giả bộ chúng ta là những người đầu tiên chuẩn bị chế ra tiêu chuẩn cho `int`. Một cách tự nhiên, mình quy định $(0\cdots0)_2 = 0, (0\cdots01)_2 = 1, (0\cdots010)_2 = 2, (0\cdots011)_2 = 3$, etc. Cái này cho đến khi $(01\cdots1)_2 = 2^{31}-1$. Mình tạm dừng ở đây là tại vì (i) đến đây mình đã cho hết một nửa các con số 32-bit; (ii) mình còn chưa assign số âm nào hết.

1.1 Chế tậm bậy

Các con số 32-bit còn lại, thật ra mình cũng có quyền chế tùy ý. Ví dụ như,

$$-1 = (10\cdots01)_2, -2 = (10\cdots010)_2, \ \ldots, -(2^{31}-1) = (1\cdots1)_2.$$

Và cái con số 32-bit cuối cùng sao cũng được:

$$(10\cdots0)_2 = 2^{31} \text{ hoặc } (10\cdots0)_2 = -2^{31}.$$

1.2 Chế đằng hoàng

Thật ra, không có ai bắt con người mình chế tiêu chuẩn cho `int` nhất quyết phải như thế nào cả. Điều quan trọng ở đây là *một tiêu chuẩn được chế ra và sử dụng rộng rãi vì sự thuận thiện nó măng lại.*

Cái tiêu chuẩn mà mình chế đại ở section trước, nó **tiện ở chỗ** là dễ nhận ra con số đó là bao nhiêu (tại vì số dương và số âm chẳng khác gì ngoài bit đầu tiên). Nhưng mà nó lại **bất tiện ở chỗ**, chẳng hạn như $(-1)+1$ sẽ là

$$(10\cdots01)_2 + (0\cdots01)_2.$$

Rất khó để giải thích tại sao cái này tính ra $(0\cdots0)_2$.

Cái tiêu chuẩn những người đi trước mình chế ra, đó là

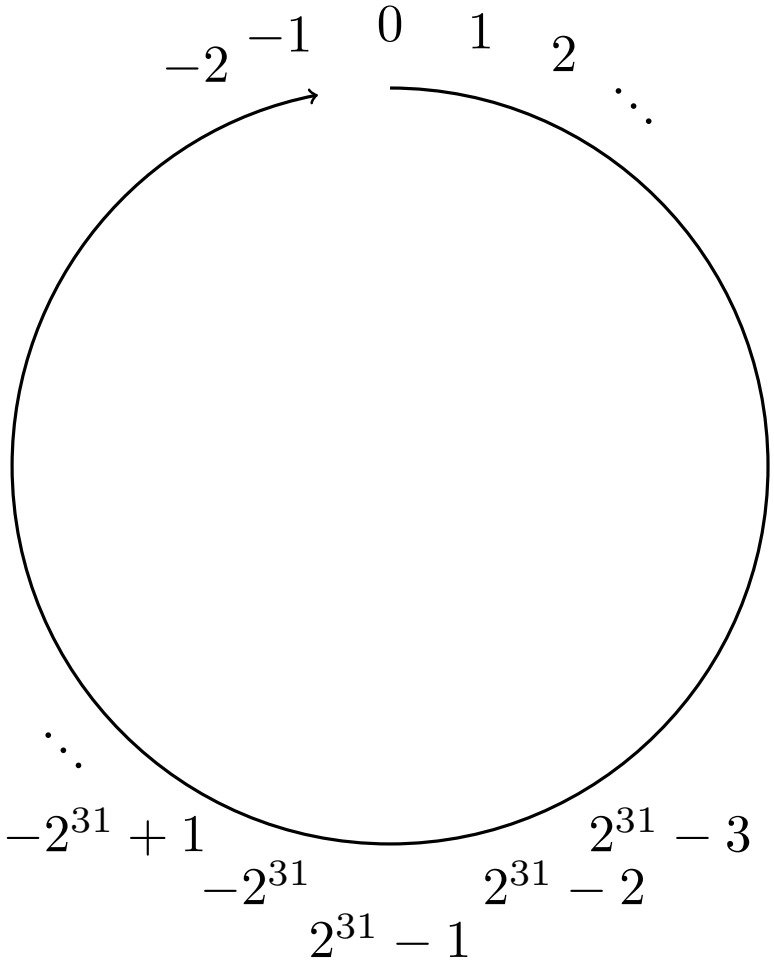
$$(10\cdots0)_2 = -2^{31}, (10\cdots01)_2 = -2^{31}+1, (10\cdots010)_2 = -2^{31}+2, \ \ldots,$$
$$(1\cdots101)_2 = -3, (1\cdots10)_2 = -2, (1\cdots1)_2 = -1.$$

Theo mình cách chế này **được cái tiện ở chỗ** cách cộng được tự nhiên và như *group* trong toán học. Ví dụ:

- $(10\cdots0001)_2 + (00\cdots0011)_2 = (10\cdots0100)_2 = -2^{31}+4$ on one side, and $(-2^{31}+1)+3 = -2^{31}+4$ on the other;

- $(1\cdots1)_2 + (0\cdots01)_2 = (10\cdots0)_2$ (**một cái 1 và ba mươi hai cái 0**) = $(0\cdots0)_2$ (**32 cái 0**) ở một phía, và $(-2^{31}+1)+1=0$ ở phía còn lại.

Cái cách cộng hai `int`'s trong hệ thống này giống Hình 1 ở dưới, khi tính `a + b` mình có thể đi từ `a` theo hướng kim đồng hồ số bước từ **số không** tới `b`; nếu cách cộng đó xoay hơn một vòng hình tròn thì mình tính lại từ đầu (i.e. từ **số không**). Các bạn đọc có thể lấy ba ví dụ hình dung trong đầu và verify có thật: (i) cộng hai con số dương; (ii) một con số dương và một con số âm; (iii) hai con số âm.



Hình 1: Để cộng hai `int`'s trong C, chúng ta có thể coi như đi bao nhiêu bước trên hình tròn như trong hình.

Một hệ quả của cách chế này là

- (bit bên tay trái nhất $= 0$) $\implies$ số không âm (i.e. số dương hoặc số 0) ;
- (bit bên tay trái nhất $= 1$) $\implies$ số âm.

Due to the cyclic nature of the addition we just defined, khi mình có một `int a` và mình muốn tìm `-a`, mình sẽ tiến hành như sau:
Nhắc lại

$(10\cdots0)_2$ (**một cái 1 và ba mươi hai cái 0**) $= (0\cdots0)_2$ (**ba mươi hai cái 0**).

Để tìm được `b` sao cho `a` đi thêm `b` bước theo chỉ kim đồng hồ tới $(10\cdots0)_2 = 0$, mình có thể

- tìm trước `c` sao cho `a + c` $= (1\cdots1)_2 = -1$. Việc này có thể đơn giản hoàn tất bằng cách đổi ngược từng bit trong `a`
- rồi `b` $=$ `c` $+ 1$

Bài này đến đây đã hơi hơi bị dài, nếu các bạn đọc cảm thấy hứng thú, sau đây là một vài hướng các bạn có thể tự suy nghĩ tiếp:

1. chứng minh cách cộng này là *commutative* và *associative*
2. `a - b` $:=$ `a + (-b)` theo đúng sense mình hay nghĩ về cách trừ giữa `int`'s. (Hint: It might be easier to think of this geometrically: rotating the angle of `-b` **clockwise** is equiv. to rotating **counter-clockwise** the angle of `b`)