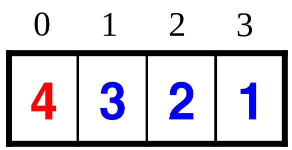
Arithmetic and Algebra

Machine Arithmetics

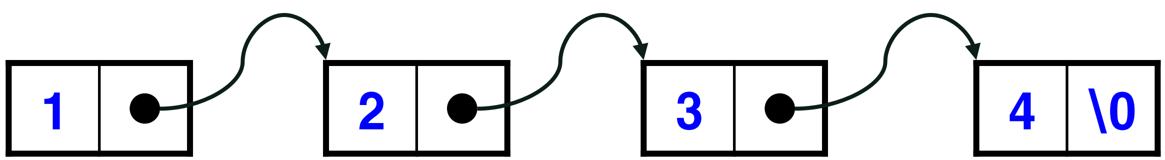
- with 4 bytes
  - an **integer** can be +/- 2^{31}
  - a **long** integer can be +/-2^{63}
  - a **long long** integer can be +/-2^{127}
  - do we need larger numbers?
    - yes, for some scientific calculation
- Libraries
  - stdlib.h : absolute, rand
  - math.h : ceiling, sqrt, exp, ...

Enormous Integer

- array
  - 1234



- linked list



array definitions

```
#define MAXDIGITS       100             /* maximum length bignum */

#define PLUS            1               /* positive sign bit */
#define MINUS           -1              /* negative sign bit */

typedef struct {
        char digits[MAXDIGITS];         /* represent the number */
        int signbit;                    /* PLUS or MINUS */
        int lastdigit;                  /* index of high-order digit */
} bignum;
```

```
print_bignum(bignum *n)
{
        int i;

        if (n->signbit == MINUS) printf("- ");
        for (i=n->lastdigit; i>=0; i--)
                printf("%c",'0'+ n->digits[i]);

        printf("\n");
}
```

now you provide addition and subtraction

연 산	덧 셈	뺄 셈		
		$A>B$	$A<B$	$A=B$
(+A) + (+B)	+(A+B)			
(+A) + (-B)		+(A-B)	-(B-A)	+(A-B)
(-A) + (+B)		-(A-B)	+(B-A)	+(A-B)
(-A) + (-B)	-(A+B)			
(+A) - (+B)		+(A-B)	-(B-A)	+(A-B)
(+A) - (-B)	+(A+B)			
(-A) - (+B)	-(A+B)			
(-A) - (-B)		-(A-B)	+(B-A)	+(A-B)

addition

```
add_bignum(bignum *a, bignum *b, bignum *c)
{
        int carry;                      /* carry digit */
        int i;                          /* counter */

        initialize_bignum(c);

        if (a->signbit == b->signbit) c->signbit = a->signbit;
        else {
                if (a->signbit == MINUS) {
                        a->signbit = PLUS;
                        subtract_bignum(b,a,c);
                        a->signbit = MINUS;
                } else {
                        b->signbit = PLUS;
                        subtract_bignum(a,b,c);
                        b->signbit = MINUS;
                }
                return;
        }

        c->lastdigit = max(a->lastdigit,b->lastdigit)+1;
        carry = 0;

        for (i=0; i<=(c->lastdigit); i++) {
                c->digits[i] = (char) (carry+a->digits[i]+b->digits[i]) % 10;
                carry = (carry + a->digits[i] + b->digits[i]) / 10;
        }

        zero_justify(c);
}
```

Assume that
A>0, B>0

A+B or (-A)+(-B)

(-A)+B=B-A

A+(-B)
A+(-B)
A+(-B)
A+(-B)

자릿수 증가를 위해

00145, -0

subtraction

```
subtract_bignum(bignum *a, bignum *b, bignum *c)
{
        int borrow;                     /* has anything been borrowed? */
        int v;                          /* placeholder digit */
        int i;                          /* counter */

        initialize_bignum(c);

        if ((a->signbit == MINUS) || (b->signbit == MINUS)) {
                b->signbit = -1 * b->signbit;
                add_bignum(a,b,c);
                b->signbit = -1 * b->signbit;
                return;
        }

        if (compare_bignum(a,b) == PLUS) {
                subtract_bignum(b,a,c);
                c->signbit = MINUS;
                return;
        }

        c->lastdigit = max(a->lastdigit,b->lastdigit);
        borrow = 0;

        for (i=0; i<=(c->lastdigit); i++) {
                v = (a->digits[i] - borrow - b->digits[i]);
                if (a->digits[i] > 0)
                        borrow = 0;
                if (v < 0) {
                        v = v + 10;
                        borrow = 1;
                }

                c->digits[i] = (char) v % 10;
        }

        zero_justify(c);
}
```

(+A)−(−B) ➔ A+B

(−A)−(+B) ➔ (−A)+(−B)

(−A)−(−B) ➔ (−A)+B

decide the sign of c

(+A)−(−B) ➔ A+B [ok!]

(−A)−(+B) ➔ (−A)+(−B) [ok!]

(−A)−(−B) ➔ (−A)+B ➔ B−A

(in the add function)

```c
compare_bignum(bignum *a, bignum *b)
{
        int i;                          /* counter */

        if ((a->signbit == MINUS) && (b->signbit == PLUS)) return(PLUS);
        if ((a->signbit == PLUS) && (b->signbit == MINUS)) return(MINUS);

        if (b->lastdigit > a->lastdigit) return (PLUS * a->signbit);
        if (a->lastdigit > b->lastdigit) return (MINUS * a->signbit);

        for (i = a->lastdigit; i>=0; i--) { 같은 자릿수
                if (a->digits[i] > b->digits[i]) return(MINUS * a->signbit);
                if (b->digits[i] > a->digits[i]) return(PLUS * a->signbit);
        }

        return(0);
}
```

(−A)−B = −(B−(−A))

A−(−B)

same sign bits

if B>A in A−B, then −(B−A)

if A>B in A−B, then A−B

(+15) + (-9)의 경우

a +

0				99
5	1	0	…	0

b −

0			99
9	0	…	0

- add_bignum(+15, -9, +C)

C +

0				99
0	…	…	…	0

  - *b*->signbit = +
  - subtract_bignum(+15, +9, +C)
  - compare_bignum(+15, +9)
    - *a*->lastdigit > b->lastdigit, so return the sign of (-*a)
  - i=0; i<=1 까지
    - v = -4 + 10 = 6, borrow=1

C +

0
6

Multiplication

- loop
  - 135 * 24
    - add 135 4 times
    - add 1350 2 times
    - add all the results

```c
multiply_bignum(bignum *a, bignum *b, bignum *c)
{
        bignum row;                     /* represent shifted row */
        bignum tmp;                     /* placeholder bignum */
        int i,j;                        /* counters */

        initialize_bignum(c);

        row = *a;

        for (i=0; i<=b->lastdigit; i++) {
                for (j=1; j<=b->digits[i]; j++) {
                        add_bignum(c,&row,&tmp);
                        *c = tmp;
                }
                digit_shift(&row,1);
        }

        c->signbit = a->sign
        
        zero_justify(c);
}
```

해당 수를 숫자
회수 만큼 더함!

해당 수를
10배 증가!

```c
digit_shift(bignum *n, int d)           /* multiply n by 10^d */
{
        int i;                          /* counter */

        if ((n->lastdigit == 0) && (n->digits[0] == 0)) return;

        for (i=n->lastdigit; i>=0; i--)
                n->digits[i+d] = n->digits[i];

        for (i=0; i<d; i++) n->digits[i] = 0;

        n->lastdigit = n->lastdigit + d;
}
```

Real Numbers

- floating point numbers
- IEEE standard
  - Single precision

s	exponent(8 bits)	significand(23 bits)

  - Double precision: 11 bit exponent (why?)
- two numbers may look the same though they are really NOT
- What about 4/7 or 3.141592....
  - fractions are better represented as (4, 7) if precision is needed

Real Numbers on Computers

- they are not contiguous
- (a + b) + c may differ from a + (b + c)

  - do not use them in conditions
  - errors accumulate
  - use decimals whenever possible
    - 0.0123123..... => 123/9990

Polynimials

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$$

- most scientific applications use them
- power function implementation?
- naive approach
  - (n+1) + n + (n-1) + (n-2) + ... multiplications
- how about $$((a_n x + a_{n-1})x + \ldots)x + a_0$$
- representations
  - sparse ones?
  - multivariate ones?

Math Libraries in C/C++

```
#include <math.h>            /* include the math library */

double floor(double x);     /* chop off fractional part of x */
double ceil (double x);     /* raise x to next largest integer */
double fabs(double x);      /* compute the absolute value of x */

double sqrt(double x);      /* compute square roots */
double exp(double x);       /* compute e^x */
double log(double x);       /* compute the base-e logarithm */
double log10(double x);     /* compute the base-10 logarithm */
double pow(double x, double y);   /* compute x^y */
```

use them whenever possible since they are much more efficient than yours

A Multiplication Game

Stan and Ollie play the game of multiplication by multiplying an integer p by one of the numbers 2 to 9. Stan always starts with $p = 1$, does his multiplication, then Ollie multiplies the number, then Stan, and so on. Before a game starts, they draw an integer $1 < n < 4,294,967,295$ and the winner is whoever reaches $p \geq n$ first.

assuming that both of them play perfectly.