TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

MÔN HỌC: KIẾN TRÚC MÁY TÍNH VÀ HỢP NGỮ
LỚP CỬ NHÂN TÀI NĂNG 2017

BÁO CÁO ĐỒ ÁN 1

BIỂU DIỄN VÀ TÍNH TOÁN SỐ HỌC TRÊN MÁY TÍNH

Giảng viên:
ThS. Phạm Tuấn Sơn
ThS. Lê Viết Long

Nhóm thực hiện:
1712152 | Nguyễn Thị Mai Thanh
1712228 | Phạm Việt Nga
1712807 | Nguyễn Thị Minh Thùy

MỤC LỤC

1. YÊU CẦU ĐỒ ÁN

- Thiết kế kiểu dữ liệu QInt lưu trữ số nguyên 16 byte, hỗ trợ các hàm chuyển đổi giữa 3 hệ cơ số nhị phân, thập phân, thập lục phân và các hàm tính toán trên số nguyên QInt.
- Thiết kế kiểu dữ liệu biểu diễn số chấm động có độ chính xác cao (Quadruple-precision), độ lớn 128 bit với 1 bit dấu, 15 bit phần mũ và 112 bit phần định trị. Hỗ trợ các hàm chuyển đổi giữa hệ nhị phân và hệ thập phân, các phép tính cơ bản trên số chấm động QFloat.

2. THÔNG TIN THÀNH VIÊN

Thành viên	MSSV	Mức độ đóng góp
Nguyễn Thị Mai Thanh	1712152	33.(3) %
Phạm Việt Nga	1712228	33.(3) %
Nguyễn Thị Minh Thùy	1712807	33.(3) %

3. ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH

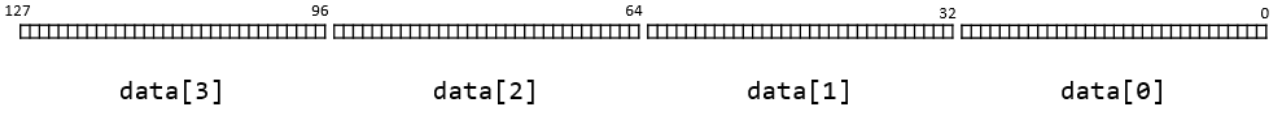
Yêu cầu	Hoàn thành
QInt	
Nhập QInt	CÓ
Xuất QInt	CÓ
Chuyển đổi hệ	CÓ
Toán tử tính toán (+, -, *, /)	CÓ
Toán tử so sánh và gán	CÓ
Toán tử tính toán trên bit (&, |, ^, ~, >>, <<, ror, rol)	CÓ
QFloat	
Nhập QFloat	CÓ
Xuất QFloat	CÓ
Chuyển đổi hệ	CÓ
Toán tử tính toán (+, -, *, /)	CÓ
Yêu cầu khác	
Giao diện người dùng, nhận input từ bàn phím	CÓ
Chương trình thực thi theo tham số dòng lệnh	CÓ
Mức độ hoàn thành đồ án:	**100%**

Đồ án được thực hiện bằng ngôn ngữ C++ trên môi trường lập trình Visual Studio 2015/ 2017, giao diện người dùng sử dụng màn hình console.

4. BIỂU DIỄN DỮ LIỆU

QInt

- QInt được biểu diễn bằng 4 số nguyên 4 byte (kiểu dữ liệu int), tương ứng với 128 bit, số nguyên thứ i biểu diễn dãy bit từ bit thứ $i * 32$ tới bit thứ $(i + 1) * 32$ với $i = 0, 1, 2, 3$.

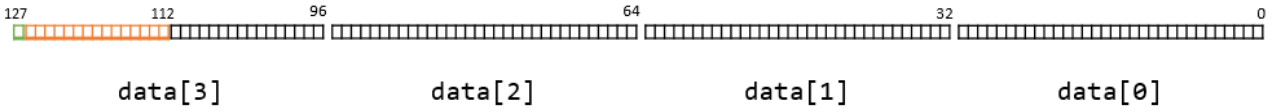


- QInt được biểu diễn dưới dạng số bù 2 nên phạm vi biểu diễn của QInt là:

$$[-2^{127}, 2^{127} - 1]$$
$$= [-170141183460469231731687303715884105728, 170141183460469231731687303715884105727]$$

QFloat

- QFloat cũng được biểu diễn bằng 4 số nguyên 4 byte (kiểu dữ liệu int), tuy nhiên, 16 bit cuối cùng của `data[3]` được dùng để biểu diễn dấu và phần mũ, phần trị được biểu diễn bằng 16 bit đầu của `data[3]` và 96 bit của `data[0]`, `data[1]` và `data[2]`, theo thứ tự từ bit `111` tới bit `0` của dãy bit.



- Phạm vi biểu diễn QFloat:
  - <u>Số dạng chuẩn:</u>
    - Do sử dụng 15 bit để biểu diễn phần mũ dưới dạng số bias nên phạm vi biểu diễn phần mũ là:

$$e_{min} \text{ có dạng } 0 \dots 01 \text{ và } e_{max} \text{ có dạng } 1 \dots 10$$
$$[1 - bias, (2^{15} - 1) - 1 - bias] = [-16382, 16383]$$
$$\text{với } bias = 2^{15-1} - 1 = 2^{14} - 1 = 16383$$

    - Dãy bit dương nhỏ nhất:

$$0\ 000000000000001\ 000 \dots 000_2 = 2^{-16382}{}_{10}$$

    - Dãy bit dương lớn nhất:

$$0\ 111111111111110\ 111 \dots 111_2 = 2^{16383} * (2 - 2^{-112})_{10}$$

    - Dãy bit âm nhỏ nhất:

$$1\ 111111111111110\ 111 \dots 111_2 = -2^{16383} * (2 - 2^{-112})_{10}$$

- Dãy bit âm lớn nhất:

$$1\ 000000000000001\ 000\ldots000_2 = -2^{-16382}{}_{10}$$

- Số không chuẩn:
  - Số không chuẩn có dạng $a * -2^{16382}$
  - Dãy bit dương nhỏ nhất:

$$0\ 000000000000000\ 000\ldots001_2 = 2^{-16382} * 2^{-112}{}_{10} = 2^{-16494}{}_{10}$$

  - Dãy bit dương lớn nhất:

$$0\ 000000000000000\ 111\ldots111_2 = 2^{-16382} * (1 - 2^{-112})_{10}$$

  - Dãy bit âm nhỏ nhất:

$$1\ 000000000000000\ 111\ldots111_2 = -2^{-16382} * (1 - 2^{-112})_{10}$$

  - Dãy bit âm lớn nhất:

$$1\ 000000000000000\ 000\ldots001_2 = -2^{-16494}{}_{10}$$

5. CẤU TRÚC CHƯƠNG TRÌNH

5.1. Các hàm chính

- ***Nhập và xử lý dữ liệu trên từng dòng:***

```
string SolveLine(string line, bool isReal)
```

Hàm đọc một dòng trong input dưới dạng chuỗi, sau đó lần lượt tách chuỗi thành từng phần dựa vào ký tự khoảng trắng, cuối cùng xác định loại yêu cầu và thực hiện yêu cầu.

  - Nếu là yêu cầu chuyển đổi hệ cơ số thì tách giá trị cần chuyển đổi và gọi hàm chuyển đổi tương ứng với kiểu dữ liệu (`IntConvertTool`/ `FloatConvertTool`) với các tham số là giá trị cần chuyển đổi, hệ ban đầu và hệ cần chuyển đổi.
  - Nếu là yêu cầu tính toán, tách hai toán hạng và toán tử, sau đó gọi hàm tính toán tương ứng với kiểu dữ liệu.

- ***Xuất dữ liệu:***

```
string QInt/QFloat::toString(int base)
```

Hàm chuyển đổi số QInt hoặc QFloat thành chuỗi để in ra màn hình hoặc ra file. Dựa vào tham số hệ số truyền vào, hàm sẽ trả ra kết quả chuyển đổi hệ số tương ứng.

- ***Chuyển đổi hệ cơ số:***

```
string [Int/Float]ConvertTool
            (string number, int typeOfNumber, int typeChange)
```

  - Bước 1: Chuyển number sang dãy bit nhị phân

- Bước 2: Từ dãy bit nhị phân chuyển sang số QInt/ QFloat
- Bước 3: Từ số QInt/ QFloat, gọi hàm `toString(int typeChange)` để lấy kết quả biểu diễn trong hệ cơ số cần chuyển đổi và trả về kết quả này.

- ***Tính toán:***
  - QInt:
    - `string Calculate1_Int(QInt &number, int numberOfBit, int typeNumber, string typeOperator)`
      Hàm thực hiện các toán tử dịch và xoay trên dãy bit.
      - `numberOfBit`: số bit cần dịch hoặc xoay
      - `typeNumber`: hệ cơ số biểu diễn kết quả
      - `typeOperator`: toán tử
    - `string Calculate2_Int(QInt a, QInt b, int typeNumber, string typeOperator)`
      Hàm thực hiện các toán tử hai ngôi.
      - `a` và `b`: 2 toán hạng
      - `typeNumber`: hệ cơ số biểu diễn hai toán hạng và kết quả
      - `typeOperator`: toán tử
    - `string UnaryOperator(int typeOfNumber, string number)`
      Hàm thực hiện toán tử NOT (~).
      - `number`: toán hạng
      - `typeNumber`: hệ cơ số biểu diễn toán hạng và kết quả
  - QFloat:
    `string Calculate_Float(QFloat a, QFloat b, int typeNumber, string typeOperator)`
    Hàm thực hiện các phép tính +, -, *, /.
    - `a` và `b`: 2 toán hạng
    - `typeNumber`: hệ cơ số biểu diễn hai toán hạng và kết quả
    - `typeOperator`: toán tử

5.2. Các hàm bổ trợ

- ***Chuyển đổi dữ liệu đọc vào từ dạng chuỗi sang dãy bit:***
  - QInt: `vector<bool> ChangeStrToArray(string number, int type)`
    - Dữ liệu đọc vào biểu diễn ở dạng nhị phân: bit thứ *i* = ký tự thứ *i* tính từ phải sang trái.
      Hàm thực hiện: `vector<bool> BinStrToArray(string number)`
    - Dữ liệu đọc vào biểu diễn ở dạng thập phân: Xét dấu, sau đó đổi phần trị sang hệ nhị phân bằng cách chia dần cho 2 (`DecDivide2(number)`)

và thêm số dư vào dãy bit. Nếu là số âm, lấy bù 2 của dãy bit bằng hàm `TwoComplement(number)`.

Hàm thực hiện: `vector<bool> DecStrToArray(string number)`

- Dữ liệu đọc vào ở dạng thập lục phân: Xét lần lượt từng ký tự của chuỗi đọc vào và chuyển thành nhóm 4 bit tương ứng.

  Hàm thực hiện: `vector<bool> HexStrToArray(string number)`

- QFloat:

  `vector<bool> FloatStrToArray(string number, int typeOfNumber)`

  - Bước 1: Xác định bit dấu của dãy bit.
  - Bước 2: Chuẩn hoá dữ liệu đầu vào.
  - Bước 3: Đưa phần định trị và phần mũ vào dãy bit. Phần mũ được biểu diễn dưới dạng số bias.

- ***Chuẩn hóa dữ liệu số chấm động:***

```
void ToNormalize(string &int_part, string &point_part,
                 string number, int typeOfNumber, int &exp)
```

  - Bước 1: Tách dữ liệu ban đầu `number` thành 2 phần: phần nguyên `int_part` và phần thập phân `point_part` dựa vào ký tự phân cách là dấu chấm (.).
  - Bước 2: Nếu số chấm động ban đầu đang được biểu diễn ở hệ thập phân thì chuyển sang hệ nhị phân (hàm `DecToBinFloat`). Chuẩn hóa phần nguyên của số chấm động ở hệ nhị phân (hàm `NormalizeInt`) và cập nhật số mũ `exp`.
  - Bước 3: Xác định dạng của số chấm động (chuẩn/ không chuẩn). Nếu số dạng chuẩn thì `int_part = "1"`, nếu số dạng không chuẩn thì `int_part = "0"` và số mũ `exp = -16382`.

- ***Chuyển chuỗi số chấm động thập phân sang nhị phân:***

```
void DecToBinFloat(string &int_part, string &point_part, int &exp)
```

  - Bước 1: Chuyển phần nguyên sang hệ nhị phân, cách làm tương tự với số nguyên, sử dụng hàm `DecDivide2(int_part)`.
  - Bước 2: Chuẩn hóa phần nguyên.
  - Bước 3: Chuyển phần nhị phân sang biểu diễn bit bằng cách nhân lần lượt phần thập phân với 2 (`PointMultiply2(point_part)`), bit thu được là phần nguyên của kết quả và gán `point_part` bằng phần thập phân của kết quả để tiếp tục nhân. Lặp lại quá trình cho tới khi phần thập phân của kết quả bằng 0 hoặc độ dài dãy bit biểu diễn phần thập phân bằng 112.

- ***Các toán tử của lớp QInt:***

  - Hàm cộng/ trừ dãy bit:

```
QInt BitAdd(QInt a, QInt b, int numberOfBit)
```

```
QInt BitSub(QInt a, QInt b, int numberOfBit)
```

- Ý nghĩa: Cộng/ trừ dãy bit của 2 số QInt `a` và `b`. `a` và `b` là các số nguyên `numberOfBit` bit.
- Thực hiện: Cộng/ trừ (có nhớ) lần lượt dãy bit bắt đầu từ bit 0. Nếu đã cộng hết `numberOfBit` nhưng vẫn còn nhớ thì thêm bit 1 vào đầu kết quả. Thêm các bit 0 (vô nghĩa) ở đầu cho tới khi dãy bit kết quả đủ 128 bit (để có thể chuyển sang QInt và trả về).

- Toán tử +:
  - Gọi hàm `BitAdd` với tham số `numberOfBit = 128`.
  - Kiểm tra tràn số: Nếu hai toán hạng khác dấu thì không thể tràn số, nếu hai toán hạng cùng dấu với nhau và khác dấu kết quả thì thông báo tràn số (bằng cách throw exception).
- Toán tử -:

  Coi phép trừ `a - b` là phép cộng `a + (-b)`. Lấy bù 2 của `b` và thực hiện phép cộng. Thông báo tràn số (bằng cách throw exception) nếu bắt được exception từ phép cộng.
- Toán tử *:
  - Trường hợp hai toán hạng đều dương: thực hiện nhân không dấu.

    ```
    QInt UnsignedMultiply(QInt &M)
    ```

    Để tối ưu hóa chương trình, hàm coi hai toán hạng như hai số nguyên `n` bit với `n = max(this->MSBpos(), M.MSBpos()) + 1`. Do đó, các hàm cộng trung gian khi nhân sẽ gọi hàm `BitAdd` với tham số `numberOfBit = n`.

    Kiểm tra tràn số: Nếu bit dấu của kết quả bằng 1 (nghã là kết quả âm) thì thông báo tràn số (bằng cách throw exception).
  - Trường hợp có ít nhất 1 toán hạng âm: thực hiện nhân có dấu bằng thuật toán Booth.

    ```
    QInt SignedMultiply(QInt &M)
    ```

    Kiểm tra tràn số: Nếu bit dấu của kết quả khác kết quả *xor* 2 bit dấu của 2 toán hạng thì thông báo tràn số (bằng cách throw exception).
- Toán tử /:
  - Nếu giá trị tuyệt đối của số bị chia nhỏ hơn giá trị tuyệt đối của số chia thì trả về 0.

    Tương tự như phép nhân, để tối ưu hóa, hàm coi hai toán hạng là hai số nguyên `n` bit với `n = this->MSBpos() + 1`. Các hàm cộng trừ

trung gian sẽ gọi hàm `BitAdd/ BitSub` với tham số `numberOfBit = n`. Sau khi có kết quả, chuyển kết quả từ biểu diễn n-bit sang 128 bit bằng cách thêm vào đầu dãy bit các bit có giá trị bằng bit trái nhất (bit ở vị trí `n - 1`).

- Nếu số bị chia và số chia khác dấu thì đổi dấu thương.
- Nếu giá trị tuyệt đối của số dư bằng giá trị tuyệt đối của thương thì cộng thêm 1 vào thương (nếu thương dương) hoặc trừ thương đi 1 (nếu thương âm).

- Toán tử gán: Gán lần lượt các data.
- Toán tử dịch bit >>, <<: Thực hiện phép dịch số học trên dãy bit.
- Toán tử xoay bit ror, rol:
  - Xoay trái n bit: Đưa n bit đầu (từ bit o tới bit `n - 1`) xuống cuối dãy bit theo đúng thứ tự.
  - Xoay phải n bit: Đưa n bit cuối lên đầu dãy bit theo đúng thứ tự.
- Các toán tử so sánh: So sánh lần lượt từ `data[3]` tới `data[0]`, kết quả trả kiểu `bool` (`true`: phép so sánh đúng/ `false`: phép so sánh sai).
- Toán tử &, |, ^, ~: Thực hiện toán tử trên từng data.

- ***Các toán tử lớp QFloat:***
  - Toán tử +:
    - Bước 1: Xét trường hợp đặc biệt, hai số hạng cùng giá trị và khác dấu → tổng bằng 0.
    - Bước 2: Tính số mũ của từng số hạng, so sánh độ chênh lệch hai số mũ (`diff_e`) và thực hiện việc dịch dãy bit phần định trị của số có số mũ nhỏ hơn tương ứng với giá trị `diff_e`.
    - Bước 3: Xử lý phần định trị: thêm bit 0 vào đầu dãy bit nếu là số không chuẩn, thêm bit 1 vào đầu dãy bit nếu là số chuẩn.
      - Nếu hai số trái dấu, tính bù 2 một trong hai dãy bit theo điều kiện: nếu số mũ hai số bằng nhau, tính bù 2 của số có trị tuyệt đối nhỏ hơn; nếu số mũ hai số khác nhau, tính bù 2 của số có số mũ nhỏ hơn.
      - Thực hiện việc cộng hai dãy bit.
      - Từ đầu dãy bit kết quả, lấy 113 bit.
      - Định lại phần trị theo chuẩn $IEEE754$.
    - Bước 4: Xử lý phần dấu:
      - Hai số cùng dấu → dấu kết quả cùng dấu hai số.

- Hai số khác dấu → dấu kết quả là dấu của số có trị tuyệt đối lớn hơn.

▪ Bước 5: Xử lý phần mũ và trả kết quả: Mũ của kết quả là tổng của số mũ lớn hơn ban đầu và độ dời dấu "." khi chuẩn hóa phần trị.

- Toán tử -:
  - Bước 1: Xét trường hợp đặc biệt, số bị trừ và số trừ cùng dấu và cùng độ lớn →hiệu bằng 0.
  - Bước 2: Cộng số bị trừ và số đối của số trừ (thực hiện việc đổi dấu bit cuối cùng) theo toán tử "+".
- Toán tử *:
  - Bước 1: Xử lý trường hợp có toán hạng bằng 0 → Tích bằng 0.
  - Bước 2: Dịch chuyển dấu phẩy để lấy hai dãy bit nguyên nhân với nhau (chỉ lấy tối đa 60 bit có nghĩa của mỗi toán hạng để nhân nhằm tránh tràn số và do chỉ lưu được tối đa 112 bit phần định trị nên không cần lấy nhiều hơn). Thêm bit 0 vào đầu dãy bit nếu là số không chuẩn, thêm bit 1 vào đầu dãy bit nếu là số chuẩn.
  - Bước 3: Tính số mũ của mỗi toán hạng sau khi dịch chuyển dấu phẩy và tính số mũ của kết quả.
  - Bước 4: Nhân hai đãy bit nguyên với nhau. Dịch chuyển dấu phẩy trên dãy bit kết quả (ban đầu kết quả có phần thập phân bằng 0) dựa vào số mũ của kết quả: số mũ dương -> dịch dấu phẩy sang phải, số mũ dương -> dịch dấu phẩy sang trái.
  - Bước 5: Từ số chấm động biểu diễn ở hệ nhị phân thu được, chuyển sang QFloat và trả về kết quả.
- Toán tử /:
  - Bước 1: Xử lý trường hợp phép chia $a/0$ hoặc $0/a$.
  - Bước 2: Xác định dấu kết quả.
  - Bước 3: Xử lý trường hợp hai số bằng nhau hoặc đối nhau → Xác định phần mũ, phần định trị bằng 0 → trả về kết quả.
  - Bước 4: Xác định phần mũ kết quả: $e_{result} = e_{dividend} - e_{divisor}$
  - Bước 5: Chia phần định trị → Chuẩn hóa lại phần định trị → Trả kết quả.

6. CÁC TRƯỜNG HỢP THỬ NGHIỆM

QInt

- Chuyển đổi hệ cơ số:

input.txt

```
10 2 81985529216486895
16 2 123456789ABCDEF
16 10 123456789ABCDEF
2 10 100100011010001010110011110001001101010111100110111101111
10 16 81985529216486895
2 16 100100011010001010110011110001001101010111100110111101111
10 2 -4891240192392183981923 8
10 16 -48912401923921839819238
16 2 FFFFFFFFFFFFF5A474109F3481C1761A
16 10 FFFFFFFFFFFFF5A474109F3481C1761A
10 2 -170141183460469231731687303715884105728
10 2 170141183460469231731687303715884105727
```

output.txt

```
100100011010001010110011110001001101010111100110111101111
100100011010001010110011110001001101010111100110111101111
81985529216486895
81985529216486895
123456789ABCDEF
123456789ABCDEF
11111111111111111111111111111111111111111111111111110101101001000111010000010000100111110011010010000001110000010111011000011010
FFFFFFFFFFFFF5A474109F3481C1761A
11111111111111111111111111111111111111111111111111110101101001000111010000010000100111110011010010000001110000010111011000011010
-48912401923921839819238
10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
```

- Tính toán:
  - Trên hệ nhị phân:

input.txt

```
2 110101010010100101010111111 + 10101011111010011110101011111111
2 101011110011111010011010110111110 - 10101011111010011110101011111111
2 1 + 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
2 1 - 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
2 1101001010111111101111111 * 1010010111111010111111111
2 10001000101001000000111001100111110101010010000001 / 1101001010111111101111111
2 111 / 110
2 100 / 111
2 0 / 11
2 11 / 0
```

output.txt

```
101011110011111010011010110111110
110101010010100101010111111
0
10
10001000101001000000111001100111110101010010000001
1010010111111010111111111
1
0
0
INVALID DIVISOR
```

- Trên hệ thập phân:

input.txt

```
10 41178321894819328 + 89283188721893921
10 -41178321894819328 + -89283188721893921
10 130461510616713249 - 89283188721893921
10 -130461510616713249 + 89283188721893921
10 -130461510616713249 - -89283188721893921
10 123456789 + -123456789
10 123456789 - 123456789
10 123456789 + 0
```

output.txt

```
130461510616713249
-130461510616713249
41178321894819328
-41178321894819328
-41178321894819328
0
0
123456789
```

input.txt

```
10 274818278132 * 3192318392
10 -274818278132 * -3192318392
10 274818278132 * -3192318392
10 877307443738555003744 / 3192318392
10 100000000000000000000 / 33
10 100000000000000000000 / -33
10 7012 * 0
10 0 / 7012
```

output.txt

```
877307443738555003744
877307443738555003744
-877307443738555003744
274818278132
30303030303030303030
-30303030303030303030
0
0
```

- Trên hệ thập lục phân:

input.txt

```
16 FFFFFFFF8 + AB736CBD2
16 1AB736CBCA - AB736CBD2
16 ABCDEF - 1
16 ABCDEF + 1
16 FFFFFFFFFFAA787382 * 1
16 ABC1289129 * 948EFCDABA
16 63AB9A3D4F6E857261CA / ABC1289129
16 0 / 127AC8DB
```

output.txt

```
1AB736CBCA
FFFFFFFF8
ABCDEE
ABCDF0
FFFFFFFFFFAA787382
63AB9A3D4F6E857261CA
948EFCDABA
0
```

- Một số trường hợp gây tràn số (kết quả phép tính nằm ngoài phạm vi biểu diễn):

input.txt

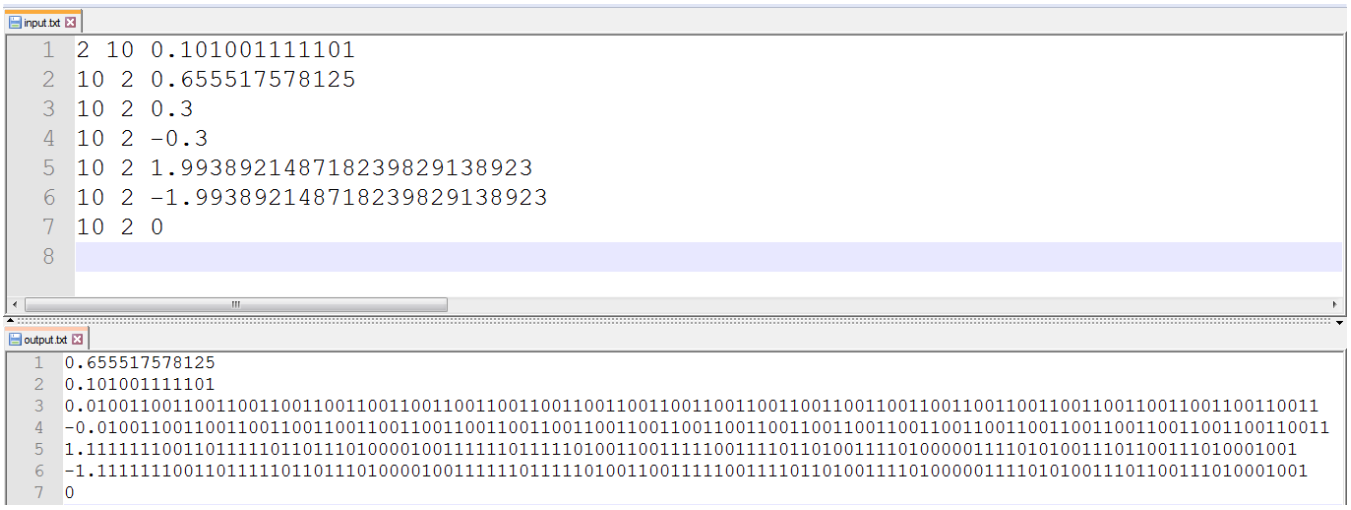
```
10 170141183460469231731687303715884105727 + 1
10 -170141183460469231731687303715884105727 - 100
10 85070591730234615865843651857942052865 * 3
10 85070591730234615865843651857942052865 + 85070591730234615865843651857942052866
```

output.txt

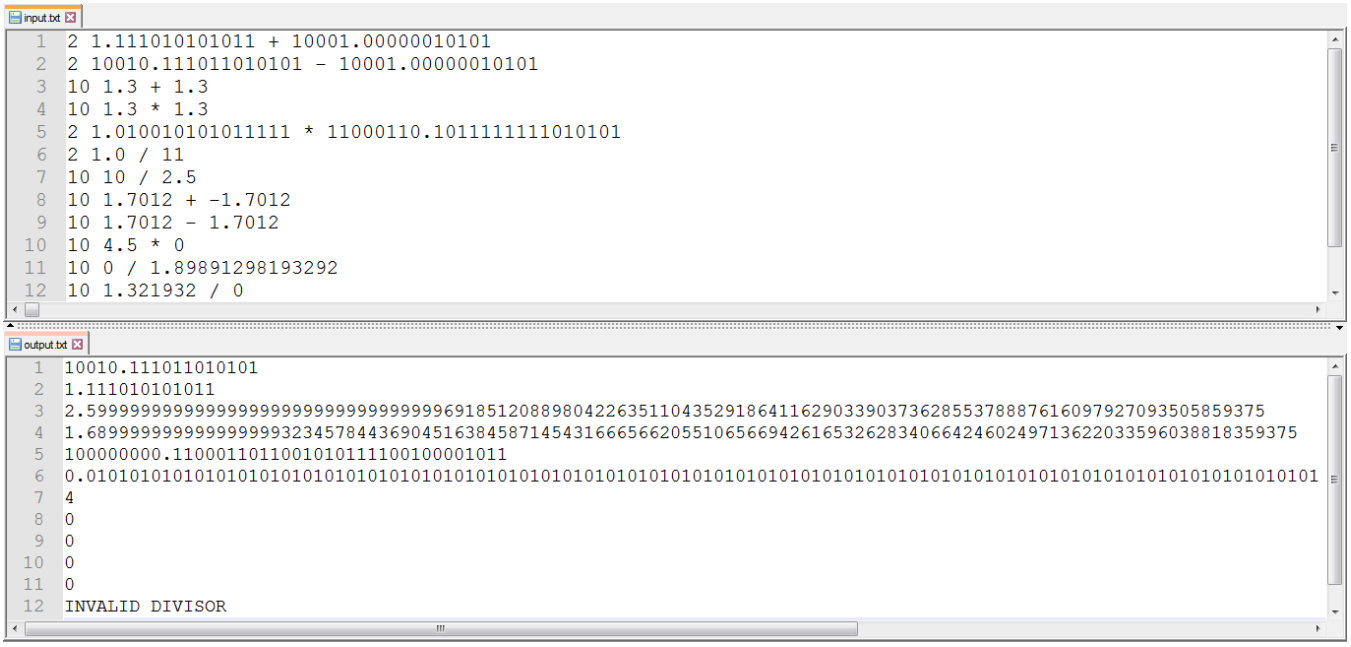
```
OVERFLOW
OVERFLOW
OVERFLOW
OVERFLOW
```

QFloat

- Chuyển đổi hệ cơ số:



- Tính toán:



7. TÀI LIỆU THAM KHẢO

- Bài giảng **Số nguyên**, **Số chấm động** – ThS. Phạm Tuấn Sơn, ĐH Khoa học Tự nhiên, ĐHQG TP.HCM
- **Quadruple-precision floating-point format** - https://en.wikipedia.org
- Bài viết **Chapter 7 -- floating point arithmetic** - http://pages.cs.wisc.edu