**BÁO CÁO ĐỒ ÁN 1**

**BIỂU DIỄN VÀ TÍNH TOÁN SỐ HỌC TRÊN MÁY TÍNH**

**A. Thành viên nhóm**

|  |  |
| --- | --- |
| **Họ và tên** | **MSSV** |
| Phạm Tấn | 18120082 |
| Nguyễn Thanh Tùng | 18120104 |
| Bùi Văn Vĩ | 18120106 |
| Lê Minh Đức | 18120164 |
| Võ Ngọc Duy | 18120174 |

**B. Báo cáo**

1. **Đánh giá chung đồ án**

Mức độ hoàn thành các yêu cầu: 100%

1. **Số nguyên lớn**
2. Đánh giá mức độ hoàn thành

|  |  |
| --- | --- |
| Yêu cầu | Mức độ hoàn thành |
| Hàm Nhập: *void ScanQInt(QInt &x)* | 100% |
| Hàm xuất: *void PrintQInt( QInt x)* | 100% |
| Hàm chuyển đổi số QInt thập phân sang nhị phân: *bool \* DecToBin (QInt x)* | 100% |
| Hàm chuyển đổi số QInt nhị phân sang thập phân: *QInt BinToDec(bool \*bit)* | 100% |
| Hàm chuyển đổi số QInt nhị phân sang thập lục phân: *char \*BinToHex(bool \*bit)* | 100% |
| Hàm chuyển đổi số QInt thập phân sang thập lục phân: *char \*DecToHex(QInt x)* | 100% |
| Các operator toán tử : “+”, “-” , “\*”, “/” | 100% |
| . Các toán tử so sánh và gán: “”, “==”, “<=”, “>=”, “=” | 100% |
| Các toán tử: AND “&”, OR “|”, XOR “^”, NOT “~” | 100% |
| Các toán tử: dịch trái “<<”, dịch phải “>>” | 100% |
| Xoay trái: “rol” *QInt rol(int k)* | 100% |
| Xoay phải: “ror’ *Qint ror(int k)* | 100% |

1. Phạm vi biểu diễn kiểu dữ liệu

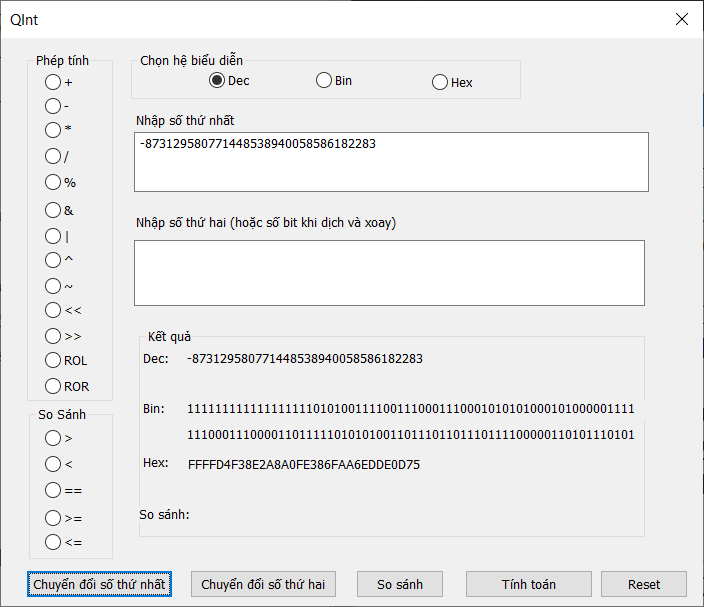
+ Số QInt 16byte = 128 bit

+ 1 bit đầu biểu diễn dấu, phần định trị 127 bit

🡪 miền biểu diễn từ -2127  đến 2127 -1

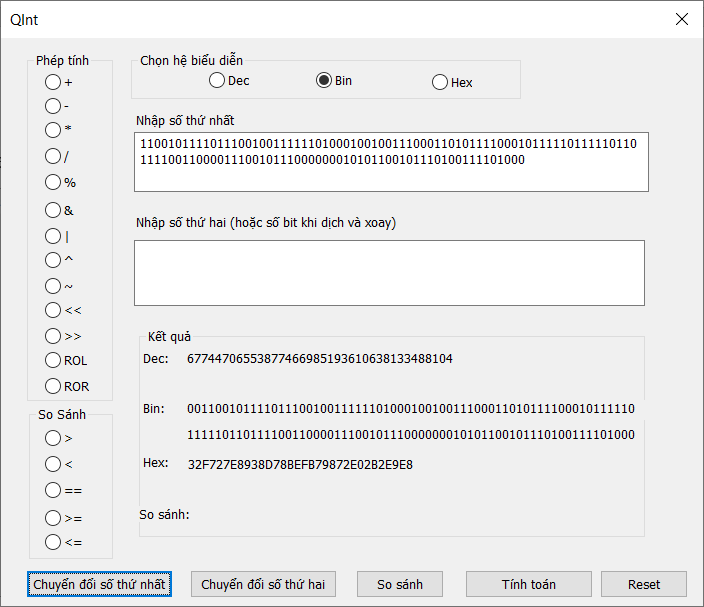
1. Chương trình minh họa với các testcase

Testcase 1 : 10 2 -873129580771448538940058586182283



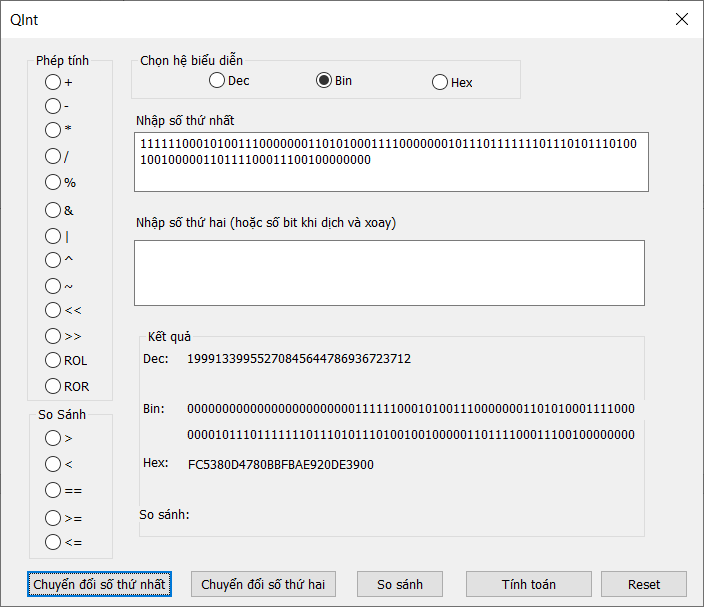
Testcase 2:

2 10 110010111101110010011111101000100100111000110101111000101111101111101101111001100001110010111000000010101100101110100111101000

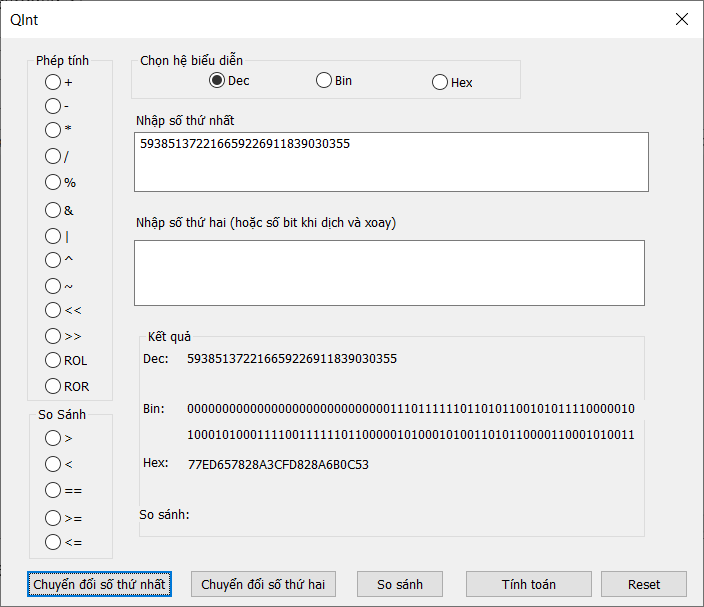


Testcase 3:

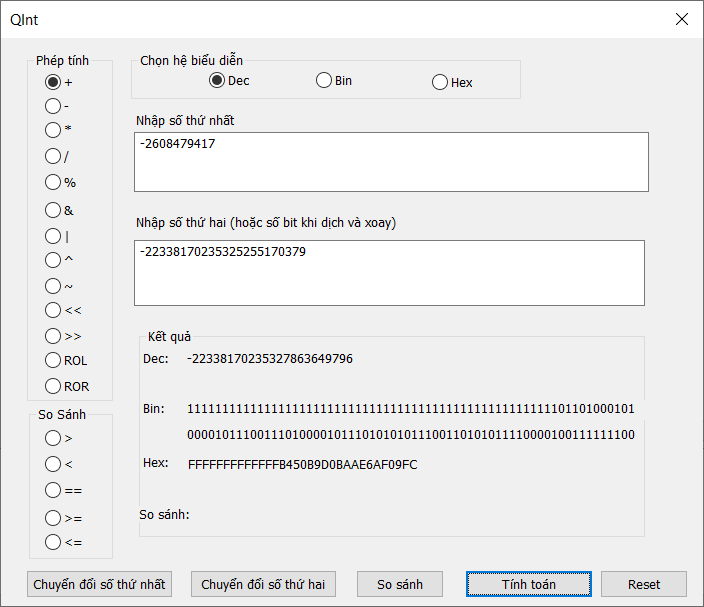
2 16 11111100010100111000000011010100011110000000101110111111101110101110100100100000110111100011100100000000



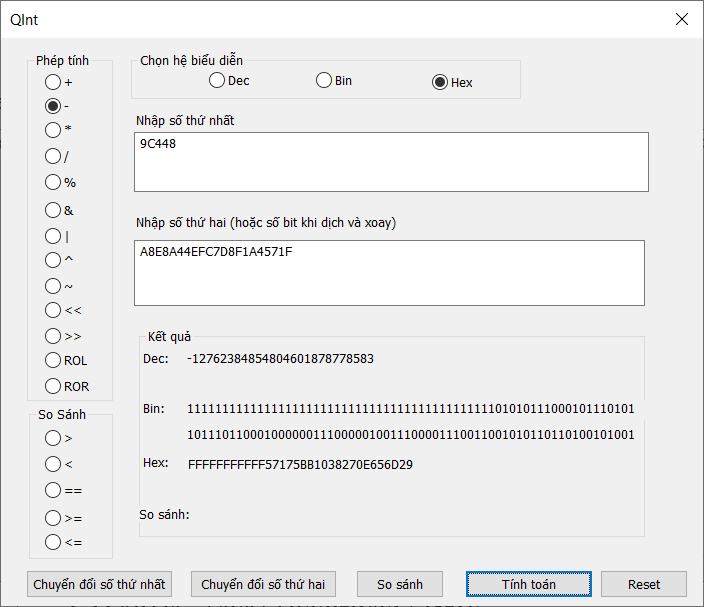
Testcase 4: 10 16 593851372216659226911839030355



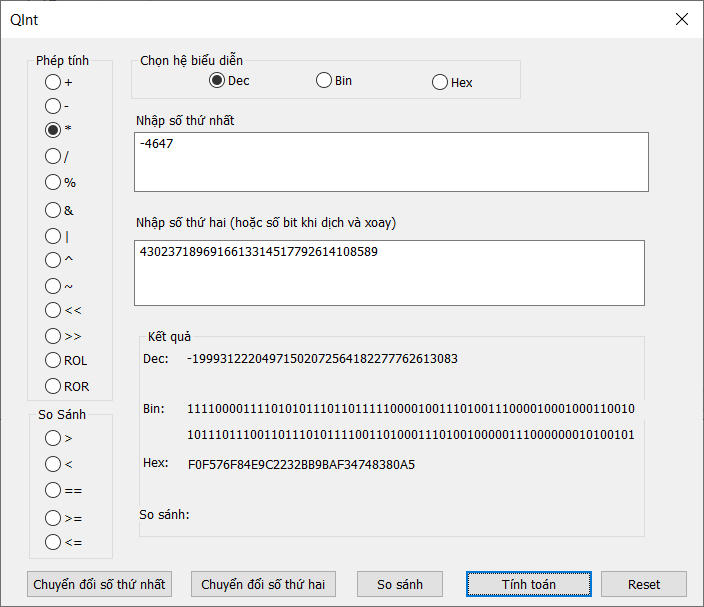
Testcase 5: 10 -2608479417 + -22338170235325255170379



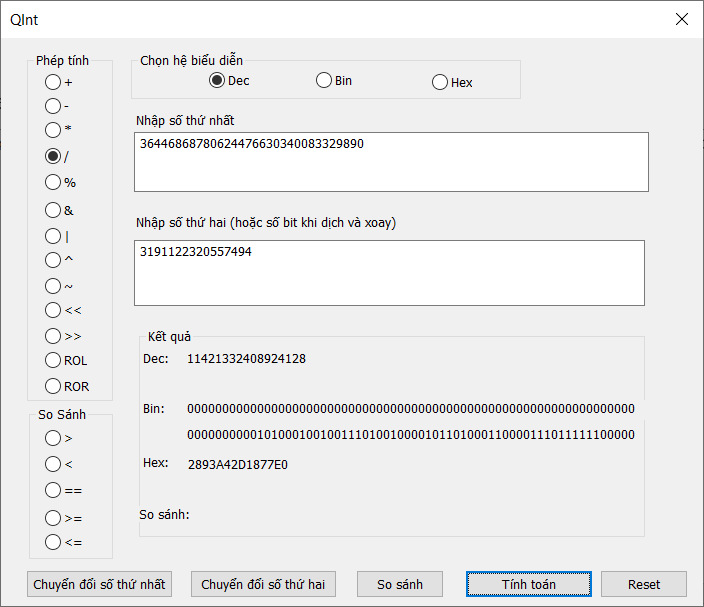
Testcase 6: 16 9C448 - A8E8A44EFC7D8F1A4571F



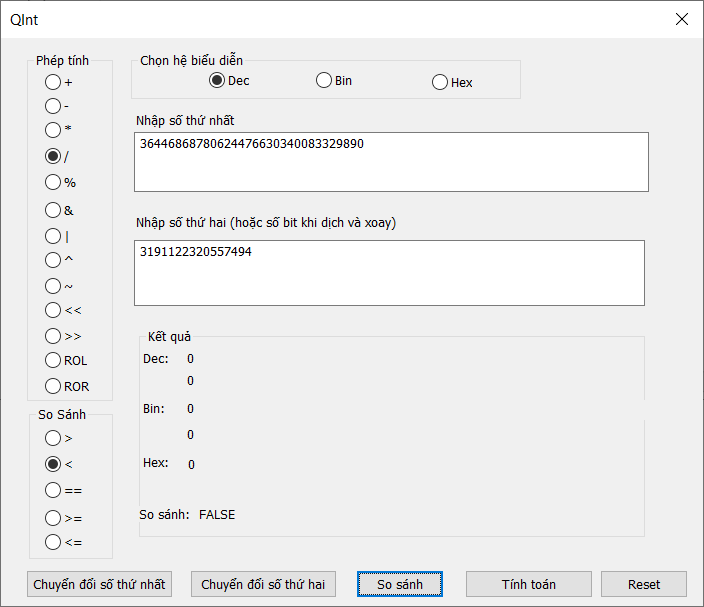
Testcase 7: 10 -4647 \* 4302371896916613314517792614108589



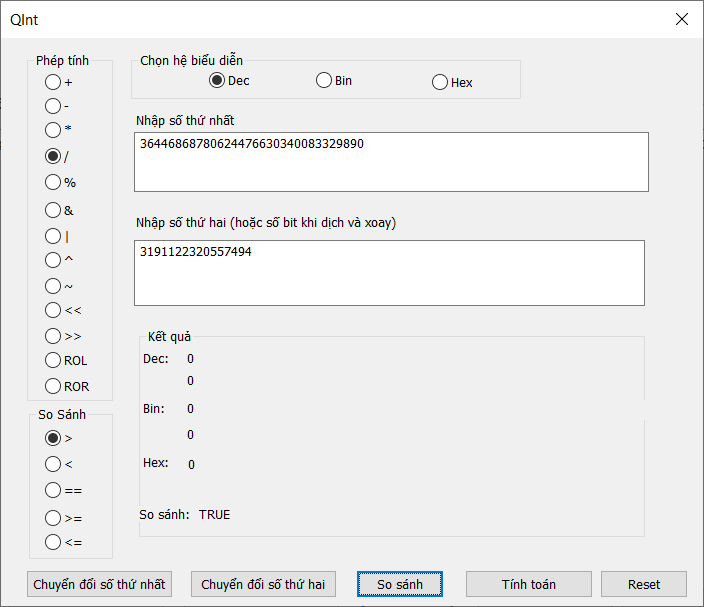
Testcase 8: 10 36446868780624476630340083329890 / 3191122320557494



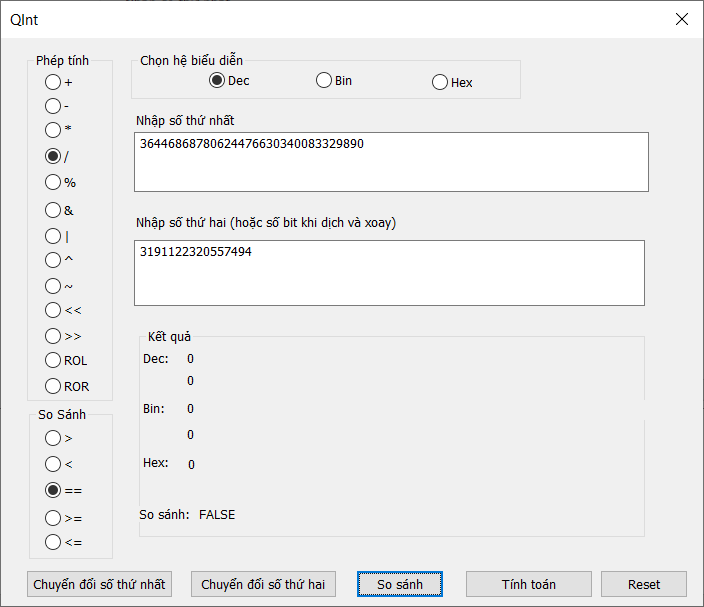
Testcase 9: 10 36446868780624476630340083329890 < 3191122320557494



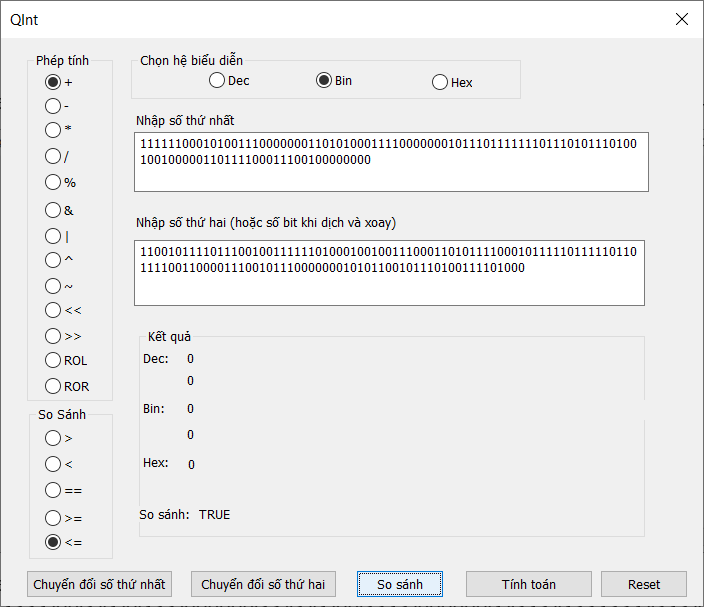
Testcase 10: 10 36446868780624476630340083329890 > 3191122320557494



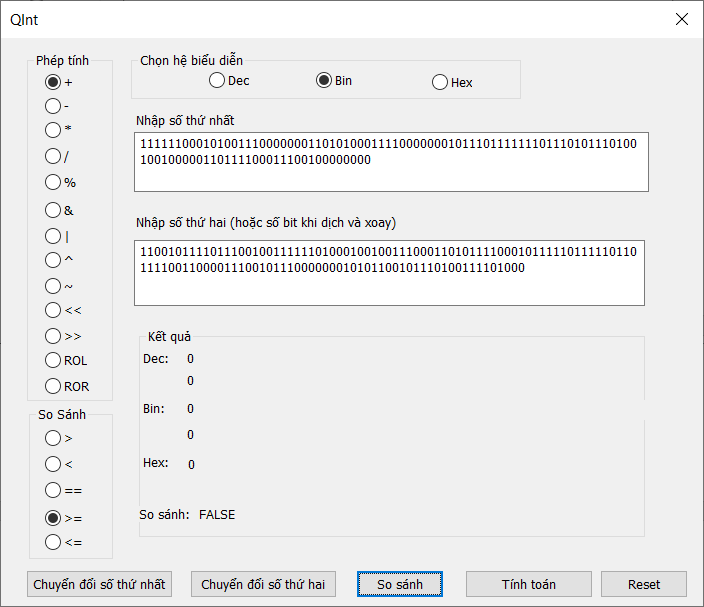
Testcase 11: 10 36446868780624476630340083329890 == 3191122320557494



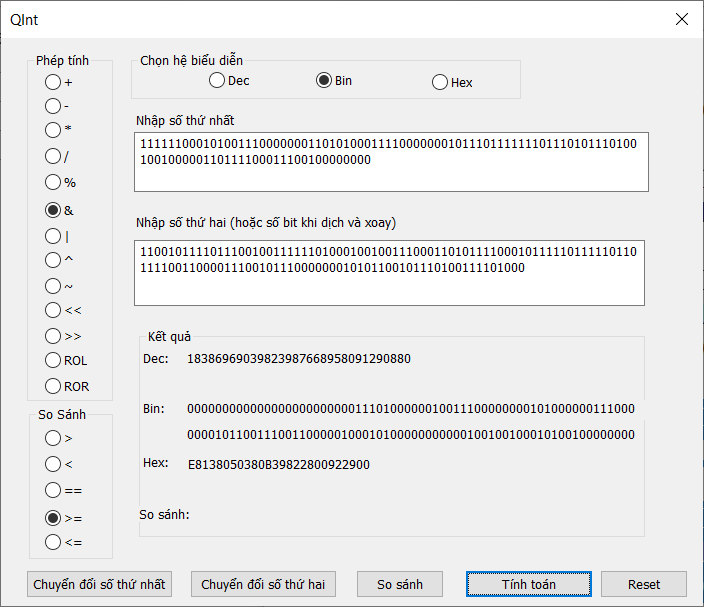
Testcase 12: 2 11111100010100111000000011010100011110000000101110111111101110101110100100100000110111100011100100000000 <= 110010111101110010011111101000100100111000110101111000101111101111101101111001100001110010111000000010101100101110100111101000



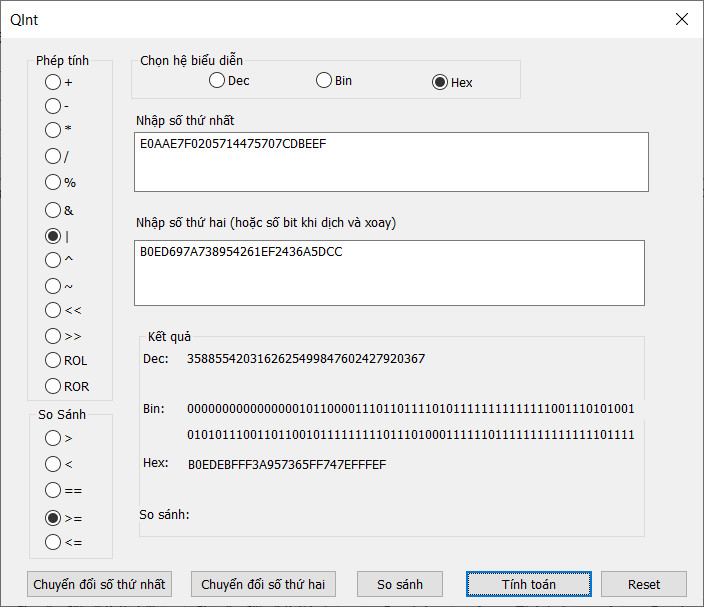
Testcase 13: 2 11111100010100111000000011010100011110000000101110111111101110101110100100100000110111100011100100000000 >= 110010111101110010011111101000100100111000110101111000101111101111101101111001100001110010111000000010101100101110100111101000



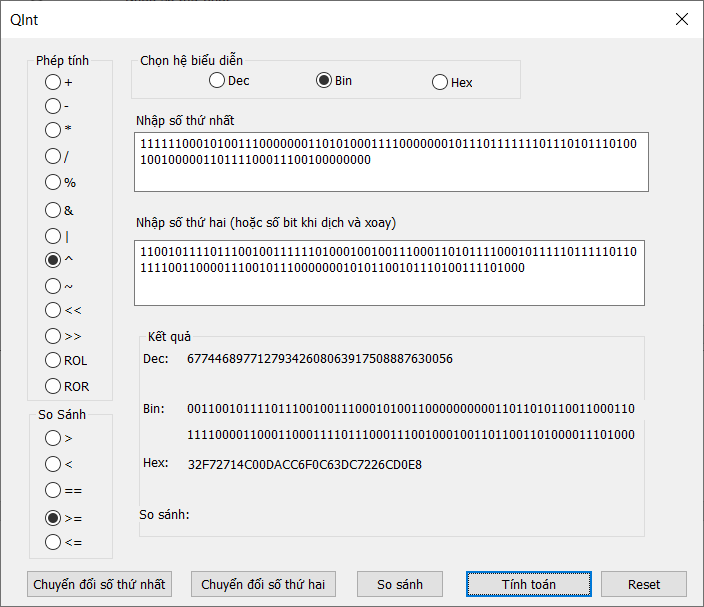
Testcase 14: 2 11111100010100111000000011010100011110000000101110111111101110101110100100100000110111100011100100000000 & 110010111101110010011111101000100100111000110101111000101111101111101101111001100001110010111000000010101100101110100111101000



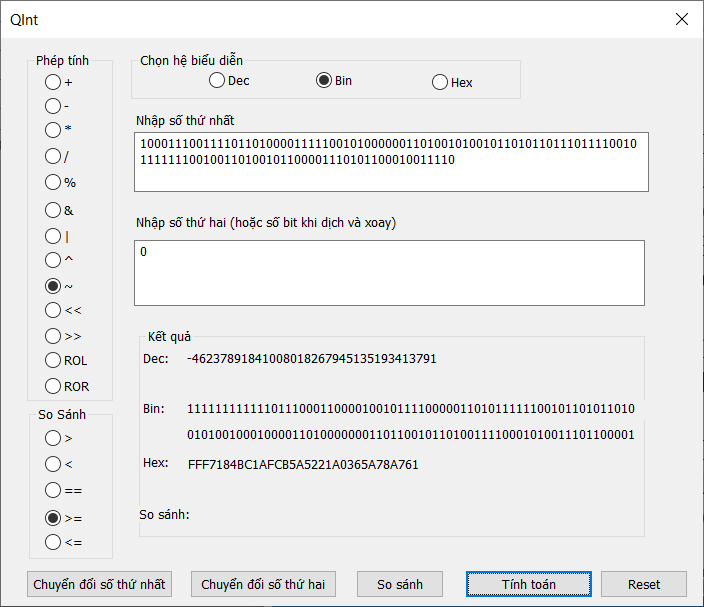
Testcase 15: 16 E0AAE7F0205714475707CDBEEF | B0ED697A738954261EF2436A5DCC



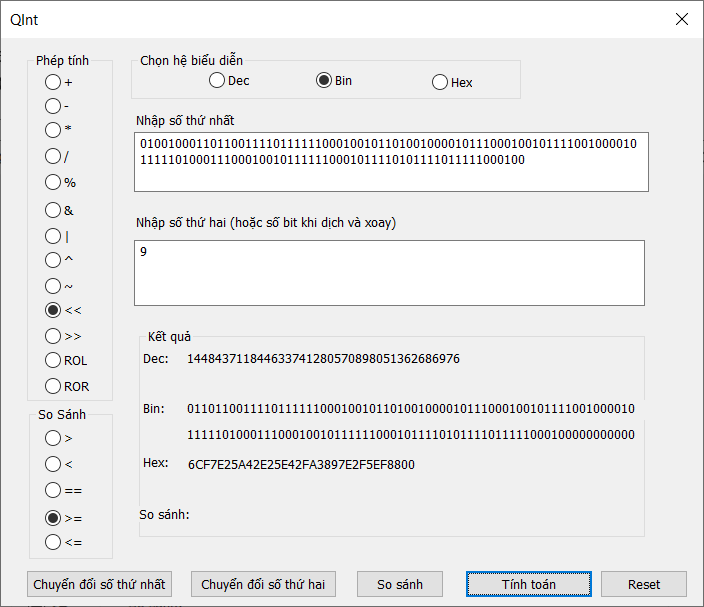
Testcase 16: 2 11111100010100111000000011010100011110000000101110111111101110101110100100100000110111100011100100000000 ^ 110010111101110010011111101000100100111000110101111000101111101111101101111001100001110010111000000010101100101110100111101000



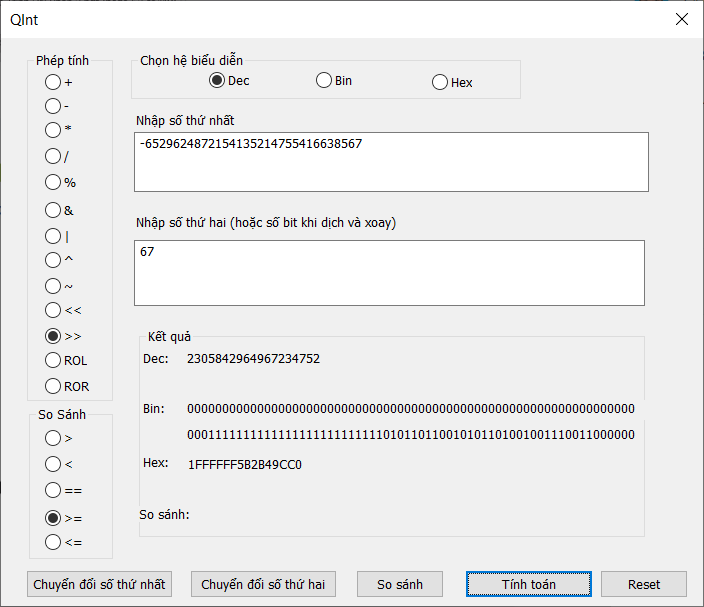
Testcase 17: 2 ~ 10001110011110110100001111100101000000110100101001011010110111011110010111111100100110100101100001110101100010011110



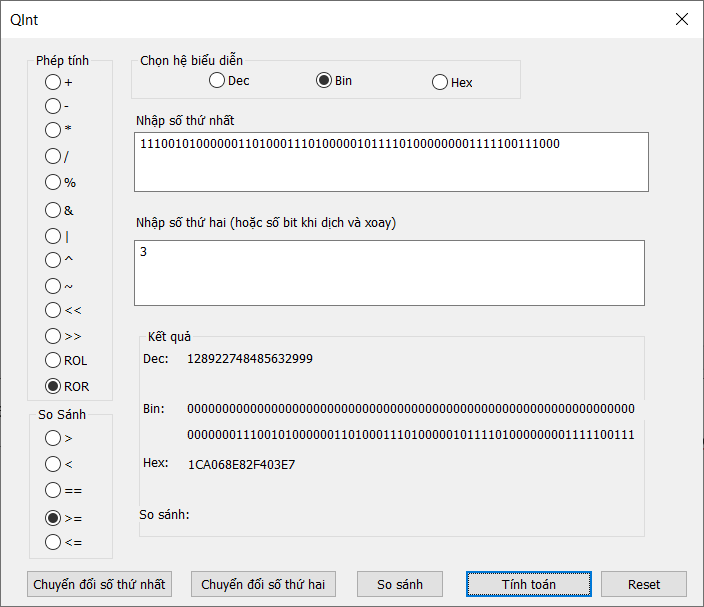
Testcase 18: 2 010010001101100111101111110001001011010010000101110001001011110010000101111101000111000100101111110001011110101111011111000100 << 9



Testcase 19: 10 -6529624872154135214755416638567 >> 67

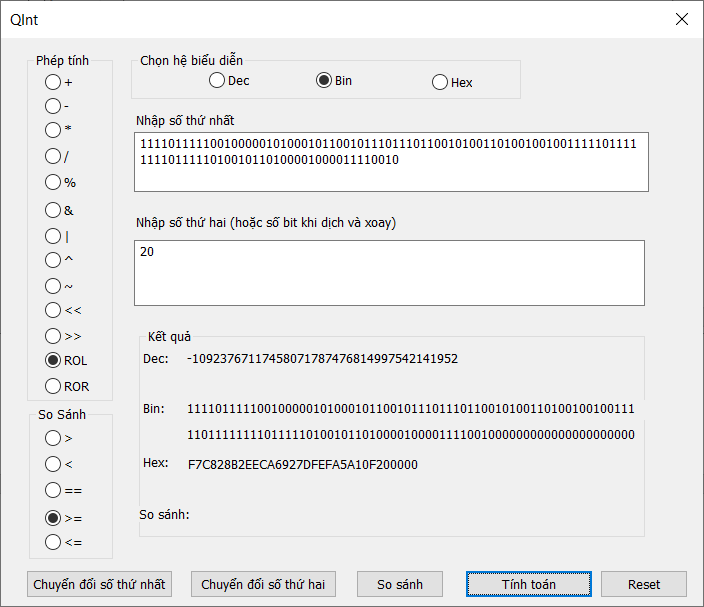


Testcase 20: 2 111001010000001101000111010000010111101000000001111100111000 ror 3



Testcase 21:

2 111101111100100000101000101100101110111011001010011010010010011111011111111011111010010110100001000011110010 rol 20



**III Số chấm động chính xác cao**

1. Đánh giá mức độ hoàn thành

|  |  |
| --- | --- |
| **Yêu cầu** | **Mức độ hoàn thành** |
| Hàm nhập | 100% |
| Hàm xuất | 100% |
| Hàm chuyển đổi số Qfloat nhị phân sang thập phân | 100% |
| Hàm chuyển đổi số Qfloat thập phân sang nhị phân | 100% |
| Các trường hợp đặc biệt từ nhị phân sang thập phân | 100% |
| Kiểm tra dữ liệu đầu vào, quá kiểu dữ liệu | 100% |
| Làm tròn khi chuyển từ thập phân sang nhị phân | 100% |

1. Phạm vi biểu diễn của kiểu dữ liệu

- Nhị phân: bắt buộc phải đủ 128 bit

+ Các trường hợp đặc biệt:

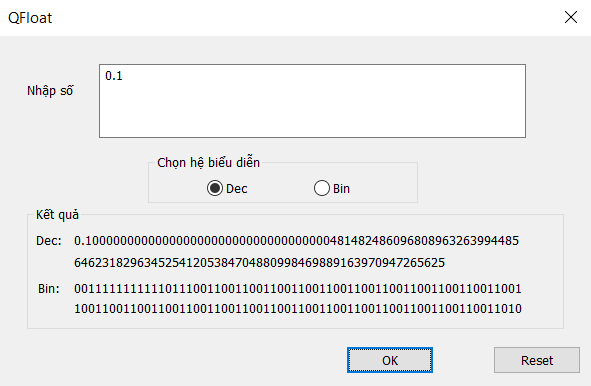
* Tất cả các bit bằng 0 thì là số 0
* 15 bit phần Exponent bằng 0 và 112 bit phần Significand khác 0 thì đó là số không thể chuẩn hóa
* 15 bit phần Exponent bằng 1 và 112 bit phần Significand bằng không thì đó là số vô cùng
* 15 bit phần Exponent bằng 1 và 112 bit phần Significand khác không thì đó là số báo lỗi

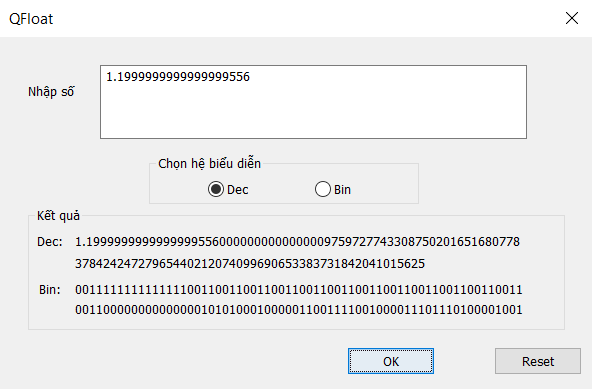
- Thập phân: dữ liệu nhập vào là chuỗi số thập phân có dấu

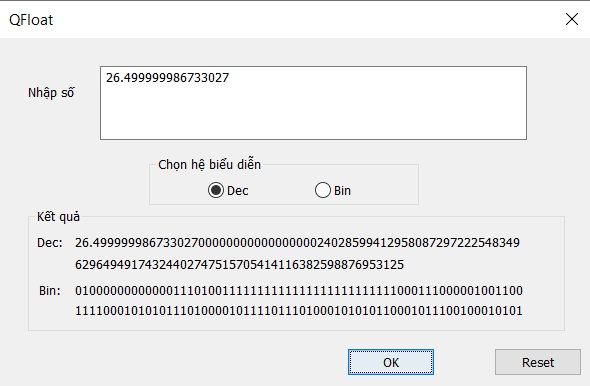
+ Phạm vi kiểu dữ liệu: khi chuyển về dạng 1.F \* 2E và chuyển Exp = E + 16383

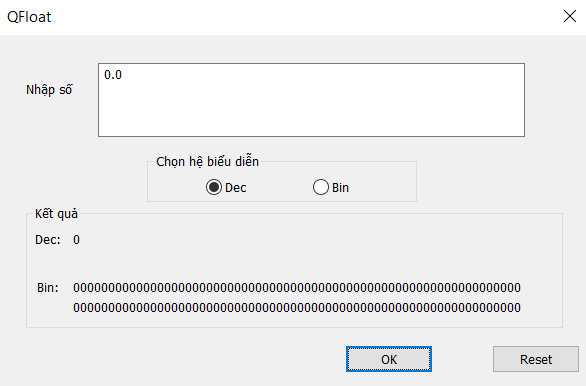
Nếu Exp < 0 hoặc Exp > 32766 thì số đó quá phạm vi biểu diễn

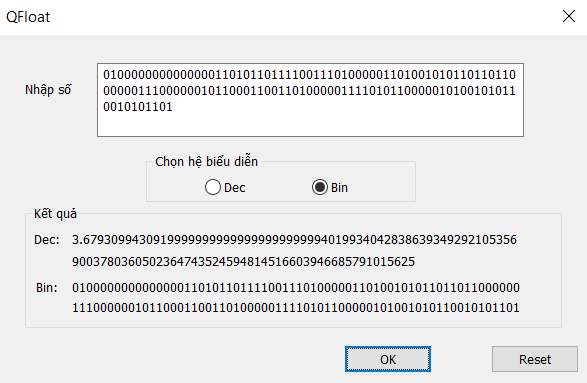
1. Chương trình minh họa với các testcase:

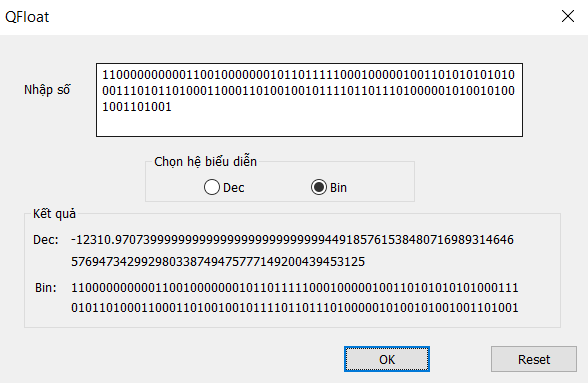


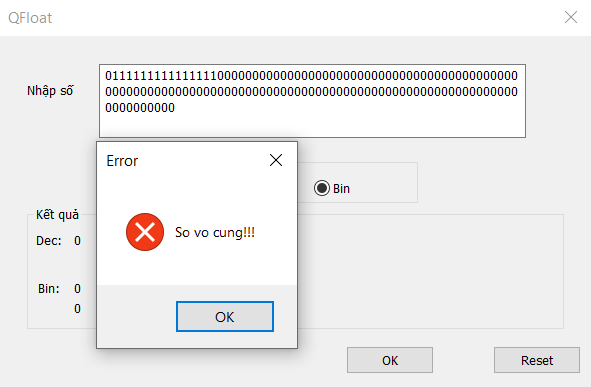


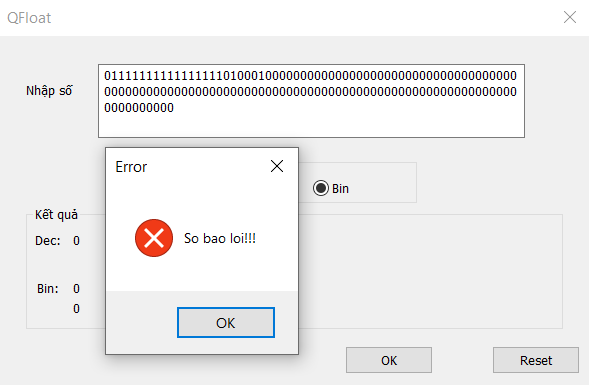


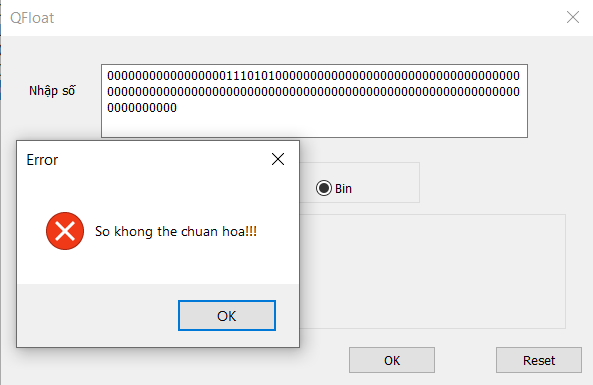












**III. Tài liệu tham khảo**

1. <https://www.tutorialspoint.com/mfc/mfc_strings.htm?fbclid=IwAR23L8zbdsnRtD2eDAIITwmqK4MMspNmlrB_sCc89Y5JqkhrLLDAF6co_BE>
2. <https://www.tutorialspoint.com/mfc/mfc_strings.htm?fbclid=IwAR23L8zbdsnRtD2eDAIITwmqK4MMspNmlrB_sCc89Y5JqkhrLLDAF6co_BE>
3. <https://www.geeksforgeeks.org/divide-large-number-represented-string/>