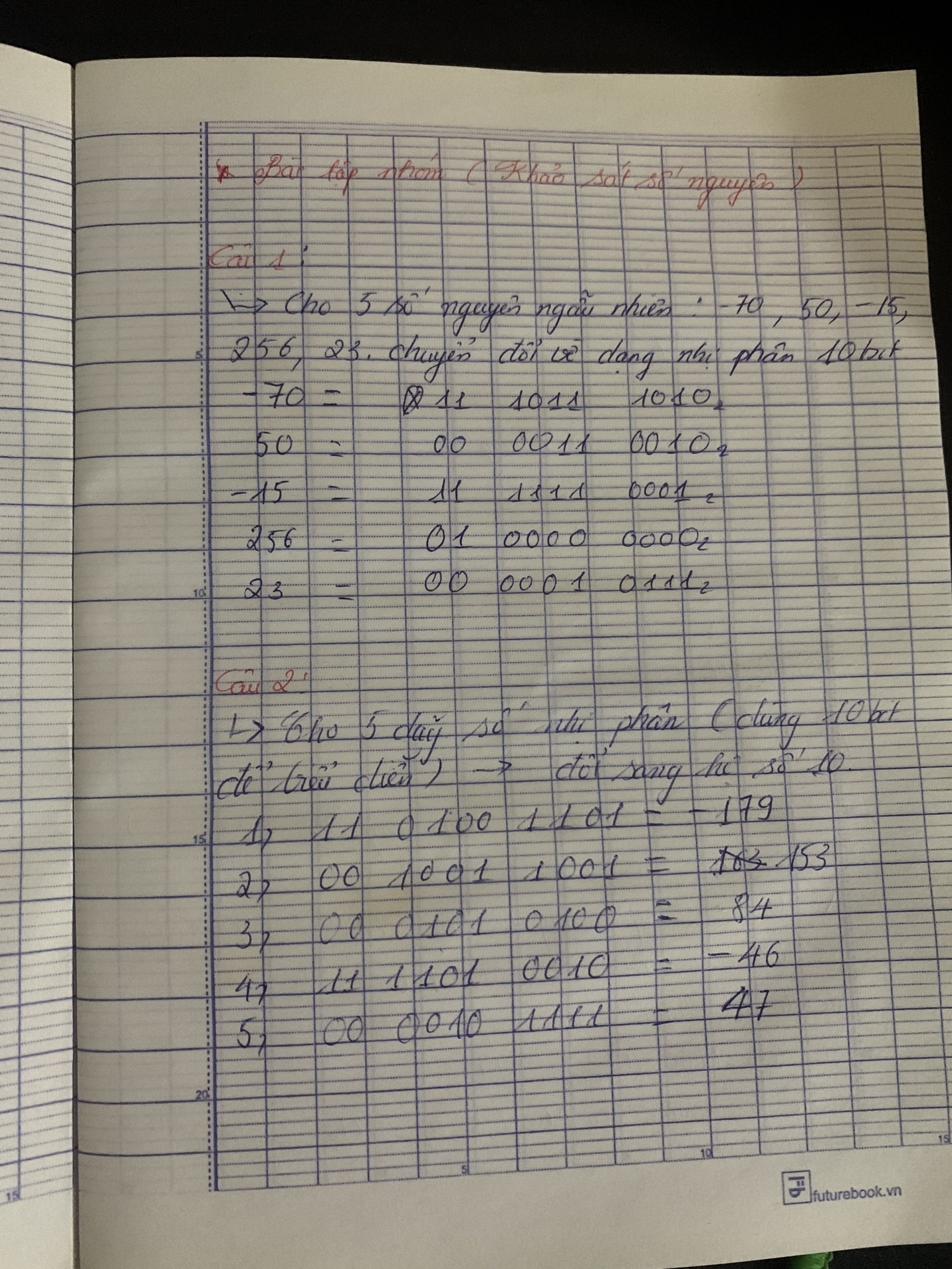
Bài 1 - 2:



Bài 3:

1.Phép toán cộng (addition):

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0 (nhớ 1)

Vd : công 2 số 10001112​ và số 111102111102​

1 0 0 0 1 1 1 (71)

+ 1 1 1 1 0 (30)

---------------

= 1 1 0 0 1 0 1 (101)

2. Phép toán trừ (subtraction):

0 − 0 = 0

0 − 1 = 1 (mượn 1)

1 − 0 = 1

1 − 1 = 0

Vd :

1 1 0 1 1 1 0

− 1 0 1 1 1

---------------

= 1 0 1 0 1 1 1 ← kết quả

------------------------------ vd 2 :

1 1 1 1 (bit mượn)

1 1 0 1 1 1 0

- 1 0 1 1 1

---------------

= 1 0 1 0 1 1 1 ← kết quả

3. Phép toán nhân (multiplication):

1 1 0 1

× 1 0 1

----------------

1 1 0 1

0 0 0 0

1 1 0 1

----------------

= 1 0 0 0 0 0 1

4. Phép toán chia (division):

Vd 1 :

159 | 12

- 12 |----

-- | 13

39 |

-36 |

-- |

3 |

Vd 2:

10011111 | 1100

- 1100 |------

---- | 1101

1111 |

- 1100 |

---- |

1111 |

- 1100 |

---- |

11 |

Bài 4:

Theo e được tìm hiểu thì những số bé nhất và những số lớn nhất của 3 kiểu số nguyên là Char, short và int.

Trước tiên em khai báo thư viện và làm thuật toán chuyển tất cả các số thập phân thành nhị phân.

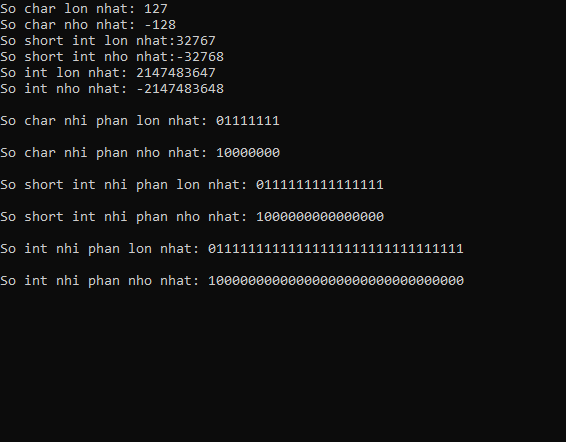
Em khai báo using namespace std để có thể viết được hàm cout

Em khai báo void ChuyenNhiPhan

Em thấy được các hàm min của char, short và int có giá trị âm và số rất nhiều cho nên em sử dụng sizeof để đếm được kích cỡ của từng dòng char, short, int.

Em khai báo hàm Bool GiaTri, dùng phép toán dịch bit (bit shift) và & (bit and) để lấy bit tại một vị trí n của số m. Trả về true nếu là 1, false nếu là 0.

Và cuối cùng em khai báo hàm main để chạy tất cả chương trình

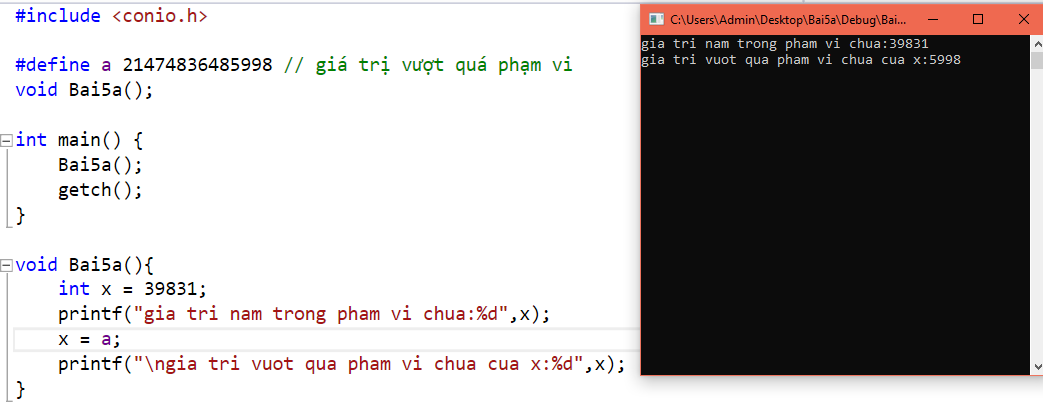
Như các bạn thấy

Bài 5a: Khi gán x=a, với a là hằng số vượt ra ngoại phạm vi có thể chứa của biến nguyên x thì kết quả nhận được sẽ như thế nào?

Như chúng ta đã biết hằng số là 1 số cố định không bao giờ thây đổi ở bài toán trên chúng ta đã cho hằng số a một giá trị **21474836485998** và gán cho biến x có kiểu dữ liệu “int” vùng giá trị là -**2147483648 -> 2147483647** như vậy giá trị của a đã vượt quá vùng giá trị của biến x.

Vậy cho 1 hằng số vượt ngưỡng giá trị của x thì kết quả ra sao?

Đối với kiểu dữ liệu “int” kích thước của nó có **4byte = 32bit** => xảy ra hiện tượng lỗi tràn số do đó khi chúng ta đưa giá trị a vượt ngưỡng **32bit** thì biến x ban đầu sẽ lấy đủ kích cỡ **32bit** sau đó nó mới tiếp tục nhận những 32bit tiếp theo và giá trị còn lại là: 5998.



Bài 5b:   
  
\_Ở bài tập này, chúng ta thực nghiệm với 2 kiểu dữ liệu là int và long (long long)  
Ban đầu, chúng ta có biến **addMaxIntVariable** với giá trị tối đa của kiểu dữ liệu Integer (kiểu dữ liệu mặc định **32 bit = 4 bytes**). Ta có thể thấy, lúc này giá trị của biến **addMaxIntVariable** = **2147483647**. Tiếp đó, ta khai báo một biến **checkMaxIntVariable** có kiểu long long (kiểu dữ liệu có giá trị tối thiểu là **64 bit = 8 bytes**) và gán cho giá trị này bằng giá trị **addMaxIntVariable + 1**. Lúc này, giá trị của biến **checkMaxIntVariable** nên là 2147483648 (2147483647 + 1), nhưng thực tế chúng ta nhận được kết quả là -2147483648.   
\_Lại tiếp tục thực nghiệm, gán giá trị cho biến checkMaxIntVariable các giá trị lần lượt là 2147483648 và LLong\_MAX(9223372036854775807). Tại đây ta có thể thấy giá trị lúc này lớn hơn rất nhiều so với addMaxIntVariable và vẫn hiển thị đúng giá trị ta đã gán.  
\_Lúc này ta có thể đưa ra kết luận: vì biến addMaxIntVariable có giá trị ô nhớ nằm trong vùng 8 Bytes này (tối đa là 2147483647) và được hiển thị với giá trị nhị phân 1111111111111111111111111111111 và có phần bù là 0, khi ta tiến hành + 1, tất cả giá trị trong vùng 8 Bytes sẽ chuyển về 0 và cộng thêm 1. Lúc này, phần cộng thêm 1 sẽ được cộng vào phần bù, do giá trị của biến addMaxIntVariable lúc này chỉ có 8 bytes, và phần 1 đã được thêm vào byte kế tiếp đó (kết quả : (1)0000000000000000000000000000000), ta nhận thấy 0000000000000000000000000000000 chính là bù 1 của 2147483647 và bù 2 = bù 1 + 1, nên lúc này(1)0000000000000000000000000000000 sẽ được biểu diễn dưới dạng bù 2, và kết quả sẽ cho ra là -2147483648.

Bài 5c:

MIN:

Ta có thể thấy đối với 2 số unsigned int(q) và unsigned char(r), thì số nhỏ nhất sẽ là 0, và số nhỏ nhất của int(p) là -2147483648, nên kết quả của phép toán cộng trên sẽ là -2147483648. Tuy nhiên, kiểu dữ liệu của phép toán + trên sẽ là unsigned int, mà theo ta đã biết, unsigned int là kiểu dữ liệu có số min là 0, nên kiểu dữ liệu từ kết quả bài toán là sai. Để có thể thấy rõ lỗi sai, thay vì dùng long x, ta có thể chuyển sang kiểu dữ liệu lần lượt là unsigned và signed int.

MAX:  
Tuy nhiên ở phép cộng, các thứ tự phép toán sẽ làm thay đổi kiểu dữ liệu của kết quả, với phép toán x = p + q + r, ta có thể hiểu gồm 2 kết quả:  
\_ t = p + q và v = t + r  
  
Ta có thể thấy được cho dù đổi vị trí của số int p trong phép toán trên, ta đều sẽ có công thức là một số unsigned int + signed int. Và lúc này, p sẽ bí ép kiểu từ signed int sang unsigned int. Tuy nhiên kết quả vẫn sẽ dẫn đến hiện tượng tràn số và dẫn tính phép tính không còn đúng đắn nữa.

**Mở rộng**

1. Tìm hiểu khái niệm căn lề các biến (word align). Trong Visual Studio từ khoá nào làm việc

này?

Căn lề các biến (word align) là quá trình sắp xếp các biến trong bộ nhớ của máy tính sao cho mỗi biến được lưu trữ ở một vị trí bắt đầu trên bộ nhớ là bội số của độ dài của từ (word length). Độ dài của từ là số bit tối thiểu có thể được xử lý đồng thời bởi CPU và các bộ vi xử lý khác trong hệ thống.

Việc căn lề các biến giúp cải thiện hiệu suất của chương trình bằng cách tối ưu hoá việc truy cập và xử lý dữ liệu trên bộ nhớ, giảm thiểu thời gian truy cập dữ liệu và tăng tốc độ xử lý.

Trong Visual Studio, từ khóa `\_\_declspec(align(n))` được sử dụng để chỉ định độ căn lề của biến, trong đó n là số byte cần căn lề. Ví dụ, để căn lề một biến kiểu double với độ dài từ là 4 byte, ta sử dụng từ khóa như sau:

```c++

\_\_declspec(align(4)) double myDouble;

```

Khi biên dịch chương trình, Visual Studio sẽ sắp xếp các biến sao cho mỗi biến có độ dài bằng bội số của độ dài từ.

2. Biểu diễn số nguyên trong phạm vi lớn hơn.

Một byte biểu diễn được số nguyên có dấu trong phạm vi từ -127 đến 127. Một byte 8 bit có thể biểu thị được 256 giá trị khác nhau (28 = 256) – đủ để lưu trữ một số nguyên không dấu từ 0 đến 255, hay một số nguyên có dấu từ -127 đến 127.