Bài Tập Lớn 1

TÍNH TOÁN ĐƠN GIẢN TRÊN SỐ NGUYÊN RẤT LỚN

1. GIỚI THIỆU

Ngày xưa ở đất nước Ấn Độ có một ông vua rất ham chơi. Ông nói với các đại thần rằng, ước mong lớn nhất của ông là có được một trò chơi mà chơi mãi vẫn không thấy chán. Ai có thể nghĩ ra trò chơi đó, sẽ được ông trọng thưởng.

Không lâu sau, một vị đại thần thông minh hiến tặng nhà vua một loại cờ. Bàn cờ có 64 ô, quân cờ có khắc chữ "Hậu", "Vua", "Xe", "Ngựa" v.v. Đánh loại cờ này - một loại trò chơi có sự biến hóa vô cùng - quả thật làm cho người ta không cảm thấy chán. Nhà vua vui mừng nói với vị đại thần đó: "Ta muốn trọng thưởng cho nhà ngươi. Nhà ngươi muốn gì, ta sẽ làm cho nhà ngươi được toại nguyện".

Vị đại thần đáp: "Thần chỉ muốn những hạt lúa mạch."

"Lúa mạch ư ? Nhà ngươi muốn bao nhiêu?"

"Thưa bệ hạ, người hãy cho đặt ở ô thứ nhất của bàn cờ một hạt lúa mạch, ô thứ hai hai hạt, ô thứ ba bốn hạt, ô thứ tư tám hạt…, cứ như thế cho đến khi đặt hết 64 ô bàn cờ."

Nhà vua nghĩ bụng: Nếu như vậy thì có đáng là bao nhiêu chứ? Nhiều lắm thì cũng vài trăm đấu! Chuyện nhỏ! Vì thế, ông nói với đại thần quản lí kho lương: "Nhà ngươi đi lấy vài bao lúa để thưởng cho ông ấy."

Vị đại thần quản lí kho lương tính toán một lúc, bỗng nhiên mặt ông ta biến sắc. Ông ta tâu trình với nhà vua: "Kết quả tính toán của thần cho thấy, số lương thực của cả nước ta còn kém xa số lương thực mà ông ấy muốn."

Nói xong liền đưa kết quả tính toán cho nhà vua xem.

$1+ 2 + 2^2 + 2^3 + \ldots + 2^{63} = 2^{64} – 1 = 18446774073709551615$ hạt lúa mạch.

Một mét khối lúa mạch có khoảng 15 triệu hạt, theo cách tính này thì cần phải đưa cho vị đại thần đó 1200 tỉ mét khối lúa mạch. Số lúa mạch này còn nhiều hơn tổng số lúa mạch mà toàn thế giới gieo trồng trong 200 năm.

Sắc mặt nhà vua tái mét, ông vội hỏi đại thần quản lí lương thực: "Làm thế nào bây giờ? Nếu đồng ý cho ông ta số lúa mạch đó, trẫm sẽ vĩnh viễn trở thành kẻ thiếu nợ. Nếu không đồng ý đưa cho ông ta, chẳng phải trẫm sẽ trở thành kẻ nuốt lời hứa sao? Nhà ngươi hãy nghĩ cách giúp ta đi!"

Vị đại thần quản lí kho lương nghĩ ngợi một lát rồi nói: "Chỉ có một cách duy nhất, bệ hạ phải thực hiện lời hứa của mình thì mới làm cho người dân tin tưởng bệ hạ là một ông vua tốt".

"Nhưng trẫm lấy đâu ra nhiều lúa mạch như thế?"

"Xin bệ hạ hãy ra lệnh mở kho lương thực, rồi mời ông ta tự đếm và nhận số lúa mạch đó".

"Như thế thì cần bao nhiêu thời gian?"

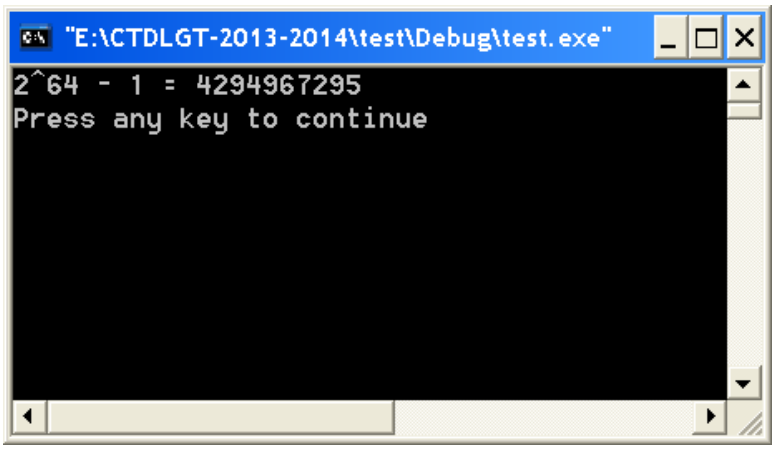
Vị đại thần quản lí kho lương tính toán một lát rồi trả lời: "Giả sử mỗi giây ông ta có thể đếm được hai hạt lúa mạch, mỗi ngày đếm 12 tiếng đồng hồ, tức là 43200 giây, thì

trong vòng 10 năm ông ta mới đếm được 20 mét khối lúa mạch. Để đếm được hết số lúa mạch mà ông ta yêu cầu thì phải mất 290 tỉ năm. Ông ta có thể sống được bao nhiêu năm chứ? Hơn nữa cuộc sống đơn điệu, tẻ nhạt như thế sẽ giày vò con người, cứ như thế chẳng phải ông ta sẽ đoản thọ hay sao? Cho nên, thần trộm nghĩ, ông ta không có ý định muốn bệ hạ trọng thưởng cho ông ta số lúa mạch mà ông ta sẽ không thể nào có được, mà ông ta chỉ muốn thử xem đất nước chúng ta còn ai thông minh hơn ông ta hay không."

Bây giờ chúng ta thử viết một chương trình bằng C++ để tính số lượng hạt lúa mạch.

```
#include <iostream.h>

int main(int argc, char* argv[])
{
	unsigned long	number = 1;

	for(int i = 0; i<64; i++)
		number = number*2;

	number = number - 1;

	cout <<"2^64 - 1 = " << number << endl;

	return 0;
}
```

Tuy nhiên khi chạy, kết quả lại không như mong muốn.



Điều này là vì số nguyên được cung cấp bởi C++ chỉ chứa được giá trị trong khoảng từ 0 đến 4,294,967,295.

Trong bài tập lớn 1, sinh viên sẽ hiện thực danh sách liên kết được dùng để biểu diễn các số nguyên rất lớn cùng với một số phép toán đơn giản trên cấu trúc dữ liệu này.

2. TÀI LIỆU CUNG CẤP CHO SINH VIÊN

Sinh viên download file **assigment1.zip** từ trang Web của môn học. Khi giải nén file này, sẽ có được các file sau:

input.txt	Một file input ví dụ
main.cpp	Chương trình chính
common.h	Định nghĩa lớp **digitList** và **Integer**
common.cpp	Chứa một số hàm và phương thức
assignment1.cpp	Mã nguồn hiện thực bởi sinh viên
Assignment1.pdf	File pdf mô tả nội dung bài tập lớn

*Lưu ý rằng **sinh viên không được phép thay đổi file** main.cpp, common.h, và common.cpp **khi hiện thực chương trình**. Ngoài ra, các hàm do sinh viên viết không được xuất bất kỳ dữ liệu nào ra màn hình khi thực thi.*

Để dịch và thực thi chương trình, sinh viên chứa cả 4 files *main.cpp*, *common.h*, *common.cpp* và *assignment1.cpp* trong cùng một thư mục; sau đó chỉ cần dịch và thực thi **duy nhất** file *main.cpp*. Mọi công việc cần phải làm sẽ được hiện thực trong file *assignment1.cpp*, tuy nhiên không cần dịch và thực thi file này.

Ví dụ 16: Để dịch và thực thi chương trình trên môi trường Cygwin, thực thi các lệnh sau:

```
g++ main.cpp -o main.exe
./main.exe
```

3. NỘP BÀI

Khi nộp bài, sinh viên sử dụng account đã được cấp phát trên hệ thống BK Sakai để nộp bài qua mạng. Sinh viên chỉ nộp đúng một file *assignment1.cpp* (tên file phải được viết thường).**Tất cả các file nộp khác file** *assignment1.cpp* **sẽ bị tự động xoá khi chấm bài.** File được nộp phải là file chương trình gốc, sinh viên không được nén file khi nộp bài. Sinh viên phải kiểm tra chương trình của mình trên **Cygwin** trước khi nộp.

Thời hạn nộp bài:

Thời hạn chót để nộp bài là **17h00 ngày thứ hai (30/09/2013)**. Sinh viên phải dùng account trên hệ thống Sakai để nộp bài. KHÔNG nhận bài được gửi qua mail hoặc bất kỳ hình thức nào khác. Bài nộp trễ sẽ KHÔNG được nhận.

4. XỬ LÝ GIAN LẬN

Bài tập lớn phải được sinh viên **TỰ LÀM**. Sinh viên sẽ bị coi là gian lận nếu:

- Có sự giống nhau bất thường giữa mã nguồn của các bài nộp. Trong trường hợp này, **TẤT CẢ** các bài nộp đều bị coi là gian lận. Do vậy sinh viên phải bảo vệ mã nguồn bài tập lớn của mình.
- Sinh viên không hiểu mã nguồn do chính mình viết, trừ những phần mã được cung cấp sẵn trong chương trình khởi tạo. Sinh viên có thể tham khảo từ bất kỳ nguồn tài liệu nào, tuy nhiên phải đảm bảo rằng mình hiểu rõ ý nghĩa của tất cả những dòng lệnh mà mình viết. Trong trường hợp không hiểu rõ mã nguồn của nơi mình tham khảo, sinh viên được đặc biệt cảnh báo là **KHÔNG ĐƯỢC** sử dụng mã nguồn này; thay vào đó nên sử dụng những gì đã được học để viết chương trình.

Trong trường hợp bị kết luận là gian lận, sinh viên sẽ bị điểm 0 cho toàn bộ môn học (không chỉ bài tập lớn).

KHÔNG CHẤP NHẬN BẤT KỲ GIẢI THÍCH NÀO VÀ KHÔNG CÓ BẤT KỲ NGOẠI LỆ NÀO!

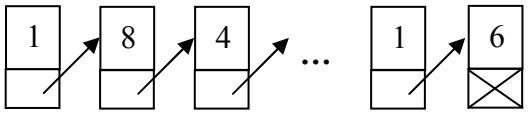
Sau mỗi bài tập lớn được nộp, sẽ có một số sinh viên được gọi phỏng vấn ngẫu nhiên để chứng minh rằng bài tập lớn vừa được nộp là do chính mình làm.

5. HƯỚNG DẪN BAN ĐẦU

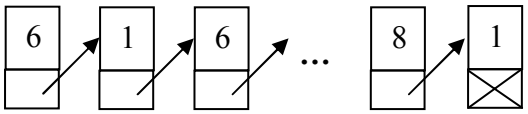
Sinh viên cần phải hiện thực 2 lớp là **class digitList** và **class Integer**. Một số phương thức của 2 lớp này đã được cung cấp sẵn, một số phương thức yêu cầu sinh viên phải hoàn thiện.

- Lớp digitList

Để biểu diễn các số nguyên rất lớn, ta thường lưu chúng dưới dạng danh sách các chữ số. Ví dụ, có thể biểu diễn 2^{64} = 18 446 774 073 709 551 616 bằng danh sách



Mặc dù cách này hoàn toàn giống với cách viết một số thông thường nhưng để tính toán, ta có cách lưu trữ đảo ngược thuận tiện hơn rất nhiều



Có thể thấy được điều này qua định nghĩa đệ quy của một số được biểu diễn bằng các chữ số của nó theo dạng thập phân:

$$n = (n \% 10) + 10 \times (n / 10)$$

Phần tử đầu tiên của danh sách là $n\%10$, phần còn lại là $n/10$. Có thể thay 10 bằng một số khác và biểu diễn con số theo hệ cơ số bất kỳ nhưng hệ thập phân là quen thuộc nhất nên ta vẫn sử dụng hệ này.

Ta sử dụng lớp **digitList** để biểu diễn một số nguyên dương

```
class digitList
{
public:
	int		digit;
	digitList*	nextDigit;
public:
	digitList():digit(0), nextDigit(NULL){}
	digitList(int d, digitList *next):digit(d), nextDigit(next){}
	int getDigit(){
		return digit;
	}
	digitList* getNextDigit(){
		return nextDigit;
	}
	digitList* copy(){
		if(nextDigit == NULL)
			return new digitList(digit, NULL);
		else
			return new digitList(digit, nextDigit->copy());
	}
	int length(){
		if(nextDigit == NULL)
			return 1;
		else
			return 1 + nextDigit->length();
	}
	digitList*	leftDigits(int n){
		//Xem chi tiết trong tập tin common.h
	}
	digitList*	rightDigits(int n){
		//Xem chi tiết trong tập tin common.h
	}
};
```

Lớp này không biểu diễn được các số nguyên âm. Lớp **Integer** phía sau sẽ làm được điều này. Để đơn giản, ta quy ước: các số phải có chữ số đầu tiên khác 0, nghĩa là phần đuôi danh sách không chứa các số không. Cần có quy ước này vì các chữ số 0 ở đầu một số nguyên không ảnh hưởng đến giá trị của số đó nhưng làm giảm tốc độ tính toán vì danh sách dài thêm. Hệ quả của quy ước này là phải biểu diễn số 0 bằng danh sách rỗng (NULL).

Trong lớp **digitList** có các hàm thành phần, như hàm tính độ dài danh sách, hàm copy danh sách, cũng như hàm **digitList* leftDigits(int n)** trả về danh sách chứa *n* phần tử tận cùng bên trái, hàm **digitList* rightDigits(int n)** trả về danh sách chứa *n* phần tử tận cùng bên phải.

Ngoài ra, trong tập tin *common.cpp* (xem để biết chi tiết), có một số hàm như:

- **digitList *digitize(int n):** chuyển một số dương thành một danh sách digitList
- **int compareDigitLists(digitList* L1, digitList* L2):** so sánh hai số nguyên được biểu diễn bởi L1 và L2. Với L1, L2 ở dạng **digitList**, hàm

này sẽ trả về 0 nếu hai số bằng nhau, trả về 1 nếu L1>L2, trả về -1 nếu L1<L2.

- **digitList* addDigitLists(int c, digitList* L1, digitList* L2):** cộng hai danh sách. Hàm này được viết dưới dạng đệ quy. Tuy nhiên, cần xử lý chữ số nhớ khi cộng. Để xử lý số nhớ một cách tổng quát, ta thêm tham số *c* để truyền giá trị nhớ. Lưu ý rằng, nếu không danh sách nào rỗng, ta cộng hai chữ số đơn vị là L1→getDigit() và L2→getDigit() với chữ số nhớ, sau đó cộng chữ số của tổng này với chữ số đơn vị của kết quả. Phần kết quả còn lại được tính bằng cách cộng đệ quy chữ số hàng chục của tổng đó với phần đuôi của hai danh sách.

- **Lớp Integer**

Ta sử dụng lớp **digitList** vừa mô tả ở trên để định nghĩa lớp **Integer**. Một đối tượng của lớp này sẽ chứa một danh sách **digitList** (*digits*) cùng với một số nguyên (*sign*) dùng để biểu diễn dấu, nhờ vậy lớp **Integer** có thể biểu diễn được cả số nguyên âm.

```
class Integer{
public :
      digitList*  digits;
      int         sign;
public:
      ~Integer(){}
      Integer(): sign(1), digits(NULL){}
      Integer(digitList *L): sign(1), digits(L) {}
      Integer(int i, digitList *L):sign(sgn(i)), digits(L) {}
      Integer(int i): sign(sgn(i)), digits(digitize(abs(i))) { }
      Integer(char str[]) {
            if(str[0] == '-'){
                  sign = -1;
                  for(int i = 0; i<strlen(str)-1; i++)
                        str[i] = str[i+1];
                  digits = digitize(str);
            }
            else{
                  sign = 1;
                  digits = digitize(str);
            }
      }
      int   length(){
            if (digits == NULL)
                  return 0;
            else
                  return digits->length();
      }
      Integer copy(){
            if(digits == NULL)
                  return Integer(0);
            else
                  return Integer(sign, digits->copy());
      }
      int sgn(int i){
            if(i < 0)         return -1;
```

```
            else                return 1;
      }
      Integer trimDigit(){
            return Integer(sign, trimDigitList(digits));
      }

      //Sinh viên phải thực hiện các phương thức này

      Integer operator +(Integer L);
      Integer operator -(Integer L);

      Integer leftDigits(int n);
      Integer rightDigits(int n);
      Integer     shift(int n);
      Integer operator *(Integer L);
};
```

Lớp đã cung cấp sẵn một số phương thức như:

- **int length():** tính chiều dài của số nguyên, tức là tính số lượng các chữ số
- **Integer copy():** copy số nguyên
- **int sgn(int i):** trả về dấu của i
- **Integer trimDigit():** loại bỏ các chữ số 0 ở cuối danh sách digits(tức là các số không ở đầu số nguyên)
- 5 hàm khởi tạo, trong đó hàm khởi tạo **Integer(char str[])** được dùng để tạo ra danh sách liên kết từ một chuỗi ký tự. Hàm khởi tạo này gọi hàm **digitList *digitize(char str[80]).**

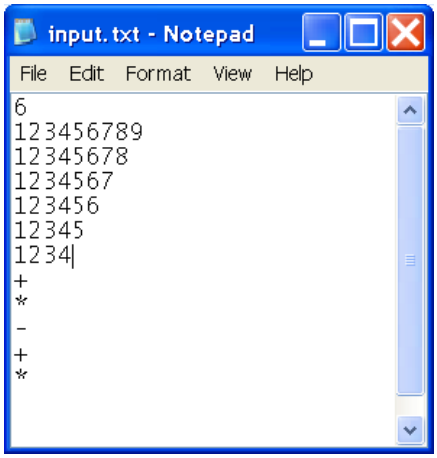
6. ĐỊNH DẠNG CỦA FILE SỐ LIỆU NHẬP

Dữ liệu nhập của chương trình được chứa trong file mang tên *input.txt*. File này chứa các thông tin như sau:

- Hàng đầu tiên: cho biết tập tin có bao nhiêu toán hạng
- Các hàng từ thứ 2 đến 2+*n*, mỗi hàng chứa một toán hạng
- *n*-1 hàng tiếp theo mỗi hàng chứa một toán hạng

Phép toán được thực hiện không xét đến độ ưu tiên, tức là lấy toán hạng thứ nhất thao tác với toán hạng thứ hai sau đó kết quả sẽ thao tác với toán hạng thứ ba, v.v… Kết quả cuối cùng được ghi vào file *output.txt*

Ví dụ 1:

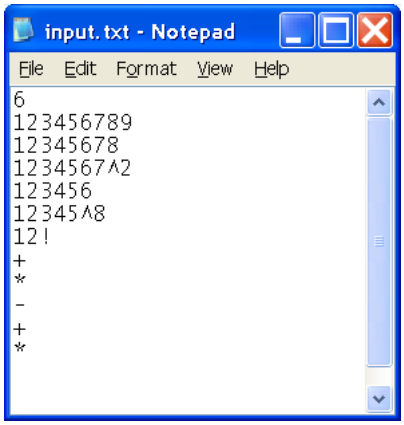


File số liệu nhập này chứa 6 toán hạng và 5 toán tử, chương trình cần phải tính ra kết quả của phép toán sau:

((((123456789 + 12345678) * 1234567) – 123456) + 12345) * 1234

Ví dụ 2:

Toán hạng ngoài việc chứa các chữ số từ 0 đến 9 cũng có thể chứa các ký hiệu (!) biểu diễn giai thừa, (^) biểu diễn số mũ.



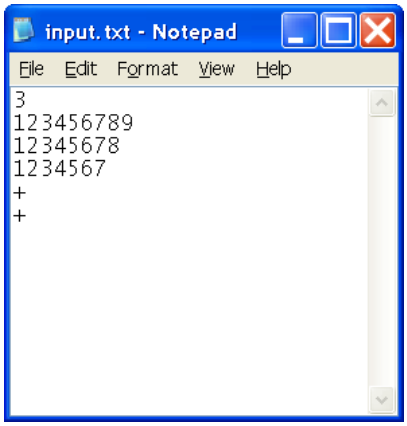
Trong trường hợp này, mỗi khi đọc đến toán hạng chứa (!) hoặc (^), chương trình sẽ tính ra giá trị của toán hạng này. Chương trình cần phải tính ra kết quả của phép toán sau:

((((123456789 + 12345678) * (1234567^2)) – 123456) + (12345^8)) * (12!)

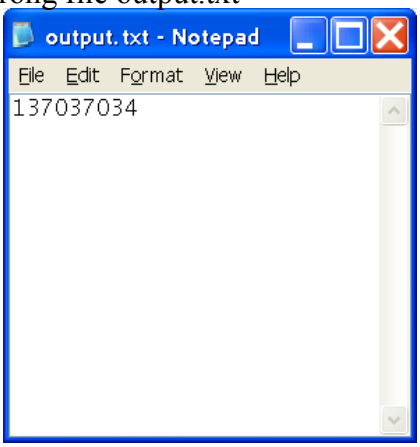
7. CÁC BƯỚC TIẾN HÀNH

BƯỚC I: Đọc hiểu và chạy thử mã nguồn được cung cấp

- Trong tập tin *main.cpp*, có các hàm **int readFile(char* filename, int&operandNum, int& operatorNum)** dùng để đọc file *input.txt*, hàm **void writeFile(char* filename, Integer L)** dùng để xuất kết quả ra file *ouput.txt*
- Trong hàm main(), các bước tiến hành như sau: đầu tiên đọc file *input.txt*, sau đó gọi hàm **computeValue(…)** để tính toán, cuối cùng ghi kết quả ra file *output.txt*. Sinh viên sẽ dần hoàn thiện hàm **computeValue(…)** ở những bước sau
- Chạy thử với file *input.txt* có nội dung như sau.



-Kiểm tra kết quả trong file output.txt



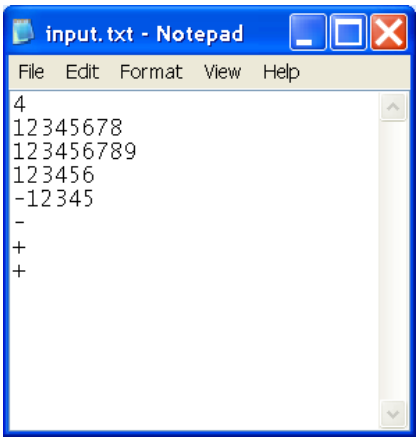
BƯỚC 2: Hiện thực phép trừ (Sinh viên tự thực hiện)

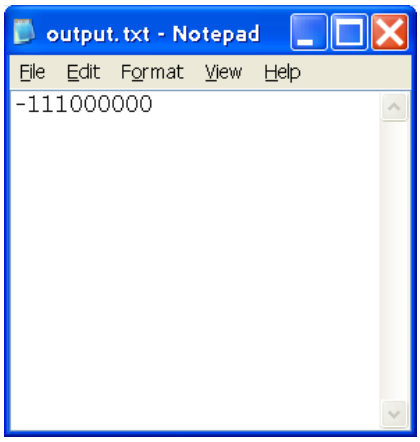
Chương trình mẫu mới thực hiện được phép cộng các số cùng dấu. Ở bước này, sinh viên sẽ tự hiện thực phép trừ.

Việc hiện thực phép trừ cũng gần giống như phép cộng nhưng có hai điểm khác biệt quan trọng. Thứ nhất, ý nghĩa của giá trị nhớ khác trước. Thứ hai, do phép trừ có thể tạo ra các số 0 đứng đầu (nghĩa là chúng ở đuôi danh sách) nên trong kết quả cần bỏ chúng đi. Ví dụ, nếu lấy 11 021 948 trừ cho 11 021 932, thì các chữ số 0 thừa ở kết quả sẽ phải bỏ đi.

Các công việc cụ thể cần làm:

- Hiện thực hàm **digitList *trimDigitList(digitList* L)** trong file *assignment1.cpp*. Hàm này có chức năng loại bỏ những chữ số 0 ở cuối danh sách
- Hiện thực hàm **digitList *subDigitLists(int b, digitList* L1, digitList* L2)** (trong file *assignment1.cpp*). Hàm này có chức năng trừ hai danh sách cho nhau
- Hoàn thiện hàm **Integer computeValue(int operatorNum)** (trong file *assignment1.cpp*).
- Hoàn thiện phương thức **Integer Integer::operator +(Integer L)** của lớp **Integer** (trong file *assignment1.cpp*). Hiện tại phương thức này mới chỉ thực hiện được phép cộng các số cùng dấu.
- Hiện thực phương thức **Integer Integer::operator -(Integer L)** của lớp **Integer** (trong file *assignment1.cpp*).
- Tự tạo file *input.txt* để kiểm chứng công việc đã làm. Hình dưới là một ví dụ minh họa. ((12345678 – 123456789) + 123456) + (-12345) = -111000000





BƯỚC 3: Hiện thực phép nhân (Sinh viên tự thực hiện)

Trong bước này, sinh viên hiện thực phép nhân cho lớp Integer. Để nhân x và y, giả sử x có $l \geq 2$ chữ số và y có nhiều nhất l chữ số. Gọi $x_0, x_1, x_2, \ldots, x_{l-1}$ là các chữ số của x và $y_0, y_1, y_2, \ldots, y_{l-1}$ là các chữ số của y (trong trước hợp số lượng chữ số của y nhỏ hơn của x, ta sẽ thêm các chữ số 0 vào cuối danh sách biểu diễn y, để cho x và y đều có cùng số chữ số). Ta có:

$$x = x_0 + 10x_1 + 10^2x_2 + \ldots + 10^{l-1}x_{l-1}$$

và

$$y = y_0 + 10y_1 + 10^2y_2 + \ldots + 10^{l-1}y_{l-1}$$

chia x thành hai phần: n chữ số bên trái và các chữ số còn lại:

$$x = x_{left} + 10^n x_{right}$$

trong đó $n = l/2$. Tương tự:

$$y = y_{left} + 10^n y_{right}$$

Vì số chữ số của y nhỏ hơn hoặc bằng x nên y_{right} có thể bằng 0. Tích số $x \times y$ có thể được viết thành:

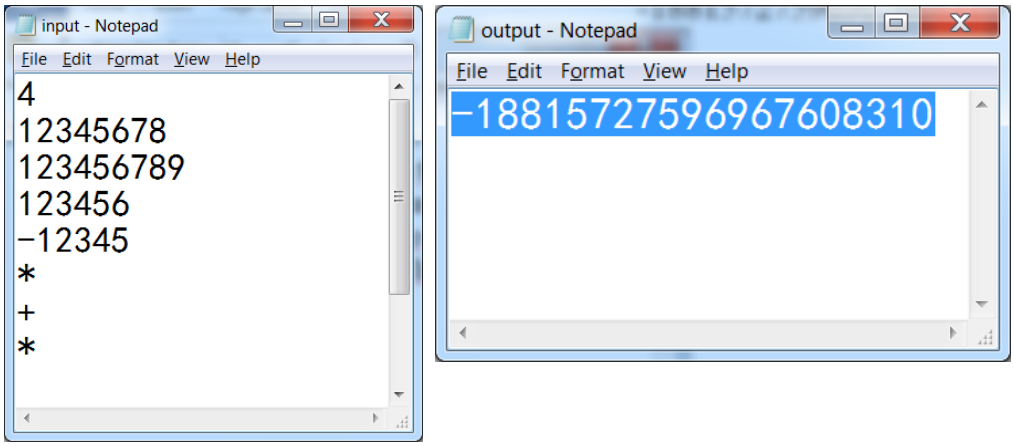
$$x \times y = (x_{left} + 10^n x_{right}) \times (y_{left} + 10^n y_{right})$$
$$= x_{left} \times y_{left} + 10^n(x_{right} \times y_{left} + x_{left} \times y_{right}) + 10^{2n} x_{right} \times y_{right}$$

Với phân tích ở trên có thể thấy việc sử dụng giải thuật đệ quy sẽ đơn giản và ngắn gọn.

Các công việc cụ thể cần làm:

- Hiện thực phương thức **Integer Integer::leftDigits(int n)** của lớp **Integer** (trong file *assignment1.cpp*). Phương thức này trả về đối tượng **Integer** có n chữ số là n chữ số bên trái của đối tượng được gọi.
- Hiện thực phương thức **Integer Integer:: rightDigits (int n)** của lớp **Integer** (trong file *assignment1.cpp*). Phương thức này trả về đối tượng **Integer** có n chữ số là n chữ số bên phải của đối tượng được gọi.
- Hiện thực phương thức **Integer Integer:: shift (int n)** của lớp **Integer** (trong file *assignment1.cpp*). Phương thức này trả về đối tượng **Integer** là kết quả của phép dịch n chữ số của đối tượng được gọi. Chức năng của phương thức này là thực hiện phép tính nhân với 10^n.

- Hiện thực phương thức **Integer Integer::operator *(Integer Y)** của lớp **Integer** (trong file *assignment1.cpp*) dựa trên phân tích ở trên với sự hỗ trợ của 3 phương thức **leftDigits(…)**, **rightDigits (…)** và **shift (…)**
- Hoàn thiện hàm **Integer computeValue(int operatorNum)** (trong file *assignment1.cpp*).
- Tự tạo file *input.txt* để kiểm chứng công việc đã làm. Hình dưới là một ví dụ minh họa. ((12345678 * 123456789) + 123456) * (-12345) = -18815727596967608310



<u>BƯỚC 4: Cho phép đọc được chuỗi số biểu diễn giai thừa và số mũ (Sinh viên tự thực hiện)</u>

- Hoàn thiện phương thức **digitList *digitize(char str[80])** (trong file *assignment1.cpp*) cho phép đọc được chuỗi ký tự biểu diễn giai thừa và số mũ.
- Tự tạo file *input.txt* để kiểm chứng công việc đã làm. Hình dưới là một ví dụ minh họa. (((123^3) * (12!)) + 123456) * (12345^4) = 207022083718920907977141123600000