مدرس: امین صادقی، رامتین خسروی
طراحان: صادق جزایری، نوید مدنی، بهزاد اوسط
موعد تحویل: جمعه ۱۸ فروردین‌ماه ۱۳۹۶

مقدمه

هدف از این تمرین آشنایی با سربارگذاری عملگرها[1] و استفاده از این قابلیت در حوزه اعداد بزرگ[2] با ممیز شناور است.
همانطور که می‌دانید اعداد از نوع double، بازه هرچند بزرگ ($-5.4*10^{-79} - 7.2*10^{75}$)[3] اما محدودی را پشتیبانی می‌کنند. اعداد بزرگ قرار است که این محدودیت را رفع کنند و بتوانند اعداد با هر تعداد رقم را پشتیبانی کند. اعداد بزرگ در حوزه‌هایی همچون رمزنگاری و کیهان شناسی کاربرد دارد.

اعداد بزرگ

برای مدل کردن اعداد بزرگ کلاس BigNum را طوری تعریف کنید که هر instance از آن، قابلیت نگهداری یک عدد را داشته باشد سپس لازم است که موارد زیر را برای این کلاس پیاده‌سازی کنید :

توابع سازنده:

```
BigNum();
BigNum(string number);
BigNum(double n);

Exapmle:
BigNum("248987124124435.4963516589234633451231451242312312512315457658113124567232515235665878902342246547656");
BigNum(89761123.2334);
BigNum(71234123);
```

سازنده اول یک نمونه جدید ایجاد می‌کند و مقدار اولیه را برابر صفر قرار می‌دهد. مورد دوم نیز عدد n را که به صورت یک رشته دریافت می‌کند به عنوان مقدار اولیه نمونه جدید قرار می‌دهند. مورد سوم نیز مقدار اولیه نمونه را برابر n قرار می‌دهد.
اگر هنگام ساختن عدد بزرگ از رشته، ورودی نامعتبر بود، یک exception را throw کنید.

عملگر جمع :

با سربازگذاری عملگرهای + و =+ این امکان را فراهم کنید که مقدار دو عدد بزرگ را با هم جمع کنیم.

```
BigNum a(1123123);
BigNum b(1.234);
BigNum c();
c = a + b;                    //c=1123124.234
a += b;                       //a=2.234
```

عملگر بین یک عدد بزرگ و یک عدد حقیقی:

عملگر جمع را بین یک عدد بزرگ و یک عدد حقیقی(double) سربارگذاری کنید. دقت کنید که این عملگر **خاصیت جابه‌جایی** دارد. عملگر ++ را نیز برای اضافه کردن مقدار حقیقی ۱ به مقادیر یک عدد بزرگ را نیز سربارگذاری کنید. همچنین دقت داشته باشید که این دو عملگر در **دو حالت** قابل استفاده هستند: عملگر قبل از عملوند و یا عملگر بعد از عملوند. (pre-increment و post-increment) لازم است که هردو نوع را پیاده‌سازی کنید.

[1] Operator Overloading
[2] Big Numbers
[3] https://www.ibm.com/support/knowledgecenter/en/SSEPEK_10.0.0/intro/src/tpc/db2z_numericdatatypes.html

```
BigNum bn1(-2.34);
BigNum bn2(1.789);
BigNum bn3();
bn3 = bn1 + 2.34567
bn3 = 2.34567 + bn1;
bn2++;
++bn2;
```

عملگر انتساب (assignment) :

عملگر = را طوری پیاده سازی کنید که عدد بزرگ سمت راست عملگر جایگزین عدد بزرگ سمت چپ قرار دهد.

عملگر قرینه :

عملگر – را طوری پیاده سازی که بتوان یک عدد مثبت را به منفی تبدیل کرد و برعکس.

عملگر براکت:

عملگر براکت [] را به‌گونه‌ای سربارگذاری کنید که نتیجه‌ی آن یک رقم از عدد بزرگ مورد نظر باشد به طوریکه برای n>=0 ، [n] رقم nام ازقسمت صحیح را بازگرداند و برای n<0، رقم nام اعشار بازگردانده شود. همچنین باید این امکان را فراهم کنید که یک رقم را بتوان استفاده از ین عملگر مقدار دهی کرد.

اگر مقداری که به یک رقم داده می‌شود بزرگتر از ۹ و یا کوچکتر از ۰ بود، یک exception را throw کنید.

```
BigNum bn("123.456");
cout << bn[0];                  // Output:3
cout << bn[2];                  // Output:1
cout << bn[-2];                 // Output:5
cout << bn[14];                 // Output:0
b[-1] = 8;                      // bn=123.856
b[2] = 13;                      // throw exception
```

عملگرهای منطقی:

عملگر == را طوری پیاده‌سازی کنید که در صورت برابر بودن هر دو عدد بزرگ سمت چپ و راست عملگر، مقدار true برگرداند و در غیر این صورت false بازگرداند.

عملگر کوچکتر < را نیز طوری سربارگذاری کنید که اگر عدد بزرگ سمت چپ کوچکتر از عملوند سمت راست بود، مقدار true و در غیر این‌صورت مقدار false بازگردانده شود.

```
BigNum bn1("123.456");
BigNum bn2("256.789432");
BigNum bn3("256.789");
BigNum bn4("256.789");
bn1==bn2;                       //false
bn2==bn3;                       //false
bn3==bn4;                       //true
bn1<bn2;                        //true
bn3<bn2;                        //true
bn4<bn1;                        //false
```

خروجی و ورودی اعداد بزرگ:

برای چاپ یک عدد بزرگ عملگر >> را برای ostream سربارگذاری کنید. هم چنین برای دریافت عددبزرگ به عنوان ورودی نیز عملگر << را برای istream سربارگذاری کنید. در نمایش عددبزرگ به عنوان خروجی توجه کنید که نباید صفرهای سمت چپ قسمت صحیح و سمت راست قسمت اعشار را چاپ کنید.

```
BigNum bn();
BigNum bn1("123.4567");
BigNum bn2("-0567.5600");
BigNum bn3("0.1234");
cin >> bn;
cout << bn1;                    // Output: 123.4567
cout << bn2;                    // Output: -567.56
cout << bn3;                    // Output: 0.1234
```

نکات پایانی

- برای این تمرین نیازی به نوشتن تابع main **نیست** و کدهای شما با استفاده از تابع main دستیاران آموزشی تست خواهند شد. به این صورت که یک فایل main.cpp در کنار دوفایل شما (bignum.h و bignum.cpp) قرار می‌گیرد که فایل header در آن include شده است.
- در مورد هرگونه خطای احتمالی دیگر، exception مناسبی throw کنید.
- مواردی که باید پیاده سازی کنید در جدول زیر آمده است:

توابع سازنده	BigNum(double n);	BigNum(string n);	BigNum();
جمع دو عدد بزرگ	Bn3 = bn1+bn2;	bn1+=bn2;	
جمع یک عدد بزرگ و یک عدد حقیقی	bn3 = bn1+d;	bn3=d+bn1;	
Increment	bn1++;	++bn1;	
Assignment	bn1 = bn2;		
قرینه	bn2 = -bn1;		
براکت	x<0 , x>=0	int n = bn[x];	b[x] = n;
عملگرهای منطقی مساوی و کوچکتر	bn1<bn2;	bn1==bn2;	
ورودی و خروجی	cin >> bn1;	cout << bn2;	

دقت کنید

- برنامه‌ی شما باید در سیستم‌عامل لینوکس نوشته و با کامپایلر g++ کامپایل شود.
- به فرمت و نام فایل‌های خود دقت کنید.
- تحویل این تمرین به صورت حضوری است و در هنگام تحویل باید به تمام قسمت‌های کد خود مسلط باشید.
- پروژه شما باید multifile باشد و makefile داشته باشد. (makefile را بر فرض داشتن main.cpp بنویسید.)
- نیازی به تقلب ندارید. در صورت کشف هرگونه تقلب مطابق سیاست‌های درس با آن برخورد خواهد شد.

نحوه‌ی تحویل

فایل‌های مربوط به برنامه‌ی خود را در پوشه‌ای به نام با نام A5-SID.zip را در سایت درس آپلود کنید. (SID پنج رقم آخر شماره‌ی دانشجویی شماست. به عنوان مثال اگر شماره‌ی دانشجویی شما ۸۱۰۱۹۵۱۲۳ است، نام فایل شما باید A5-95123.zip باشد.)