Velká čísla

Termín odevzdání:	**05.04.2020 23:59:59**
Pozdní odevzdání s penalizací:	**30.06.2020 23:59:59** (Penále za pozdní odevzdání: 100.0000 %)
Hodnocení:	**7.6435**
Max. hodnocení:	**7.1500** (bez bonusů)
Odevzdaná řešení:	2 / 25 Volné pokusy + 20 Penalizované pokusy (-2 % penalizace za každé odevzdání)
Nápovědy:	1 / 2 Volné nápovědy + 2 Penalizované nápovědy (-10 % penalizace za každou nápovědu)

Úkolem je realizovat třídu `CBigInt`, která bude reprezentovat celá čísla (kladná i záporná) s (téměř) neomezeným rozsahem.

Celá čísla typu `int`, `long long int`, ... mají fixní velikost, tedy omezený rozsah. Pokud potřebujeme výpočty ve větším rozsahu hodnot, musíme si pro ně vytvořit vlastní datový typ. Třída implementující tento datový typ bude číslo ukládat ve vnitřní reprezentaci, kterou bude podle potřeby natahovat (alokovat větší prostor). Vaším úkolem je takovou třídu realizovat. Pro zjednodušení implementace jsou následující omezení:

- Ukládáme pouze celá čísla (kladná, nulu i záporná). Nezabýváme se desetinnou částí.
- Z matematických operací implementujeme pouze sčítání, násobení a porovnávání.

Realizovaná třída tedy musí splňovat následující rozhraní:

konstruktor implicitní
: inicializuje objekt, který bude reprezentovat hodnotu 0,

konstruktor s parametrem celého čísla `int`
: inicializuje objekt, reprezentující toto číslo,

konstruktor s parametrem řetězce (ASCIIZ)
: inicializuje objekt s hodnotou, jejíž desítková reprezentace je v předávaném řetězci. Pokud je zadaný řetězec neplatný (neobsahuje platné desítkové číslo), konstruktor vyhodí výjimku `invalid_argument`. Výjimka je součástí standardní knihovny, její deklarace je v hlavičkovém souboru `stdexcept`. Při vyhazování výjimky `invalid_argument` lze jejímu konstruktoru předat řetězec s podrobnějším popisem příčiny chyby, pro tuto úlohu není obsah tohoto řetězce omezen,

kopírující konstruktor
: bude implementován, pokud to vnitřní struktury Vaší třídy vyžadují,

destruktor
: bude implementován, pokud to vnitřní struktury Vaší třídy vyžadují,

přetížený operátor =
: bude umožňovat přiřazení z celého čísla, řetězce a jiné instance `CBigInt`,

operátor <<
: bude umožňovat výstup objektu do C++ streamu (desítková reprezentace, bez zbytečných úvodních nul).

operátor >>
: bude umožňovat načtení ze vstupního streamu (vstup bude v desítkovém zápisu). Čtení se bude chovat stejně jako načítání celých čísel ve standardní knihovně, tedy zastaví na prvním znaku, který již nemůže být platnou součástí čteného čísla.

operátor +
: umožní sečíst dvě čísla typu:

  - `CBigInt + CBigInt`,
  - `CBigInt + int`,
  - `CBigInt + ASCIIZ řetězec`,
  - `int + CBigInt` a
  - `ASCIIZ řetězec + CBigInt`.

operátor +=
: umožní k číslu `CBigInt` přičíst jiné číslo `CBigInt`, celé číslo nebo číslo v podobě ASCIIZ řetězce.

operátor *
: umožní vynásobit dvě čísla ve stejných kombinacích zápisu jako operátor pro sčítání.

operátor *=
: umožní číslo typu `CBigInt` přenásobit jiným číslem `CBigInt`, celým číslem nebo číslem v podobě ASCIIZ řetězce.

relační operátory (< <=, > >=, == a !=)
umožní porovnávat velká čísla mezi sebou, porovnání si opět musí poradit se všemi kombinacemi jako sčítání a násobení.

Odevzdávejte zdrojový soubor, který obsahuje Vaší implementaci třídy `CBigInt`. V odevzdávaném souboru nenechávejte vkládání hlavičkových souborů, Vaše testovací funkce a funkci `main`. Pokud v souboru chcete ponechat `main` nebo vkládání hlavičkových souborů, vložte je do bloku podmíněného překladu.

V tomto příkladu není poskytnutý předpis pro požadované rozhraní třídy. Z textového popisu, ukázky použití níže a znalostí přetěžování operátorů byste měli být schopni toto rozhraní vymyslet.

Nápověda

- Testovací prostředí kontroluje hodnoty ve Vašich objektech tím, že si je zašle do výstupního proudu a kontroluje jejich textovou podobu. Dokud Vám nebude správně fungovat výstup, budou všechny testy negativní.
- Operátor pro výstup implementujte správně -- neposílejte data na `cout`, posílejte je do předaného výstupního proudu. Za výstupem čísla do proudu nepřidávejte odřádkování ani jiné bílé znaky.
- Pokud Vám program nejde zkompilovat, ujistěte se, že máte správně přetížené operátory. Zejména si zkontrolujte kvalifikátory `const`.
- V jednoduché variantě lze velká čísla reprezentovat uvnitř třídy jako řetězec jejich desítkové absolutní hodnoty a oddělené znaménko. Sčítání a násobení lze provádět tak, jak jsme se jej učili na základní škole. Tento postup není moc rychlý, ale správně implementované řešení na tomto principu vyhoví všem testům kromě bonusových testů rychlosti.
- Bonusový test 1 (rychlost) vyžaduje rychlé násobení. Test spočívá ve výpočtu faktoriálů velkých čísel (např. 5000!). Desítková reprezentace z minulého bodu není vhodná. Lepší výsledky dává binární reprezentace.
- Bonusový test 2 (rychlost) vyžaduje rychlé násobení velkých čísel. Násobí se čísla s řádově desítkami až stovkami tisíc desítkových cifer. Nutností je binární reprezentace a efektivní algoritmus násobení (algoritmus Karatsuba, násobení založené na FFT). Jedná se o bonus, implementace tohoto rozšíření citelně prodlužuje délku implementace.
- Načítání ze streamu musí správně manipulovat s příznakem chyby (fail bit). Chování `CBigInt` má být stejné jako chování pro celá čísla (čtou se desítkové cifry dokud to lze, úvodní bílé znaky se přeskakují, pokud je načtena alespoň jedna cifra, čtení uspěje). Pokud je na vstupu nečíselná hodnota (nelze načíst ani jednu cifru), musíte nastavit fail bit a nechat v proměnné její nezměněnou původní hodnotu (viz ukázka, hodnota 999). Pro nastavení příznaku chyby se hodí `is . setstate (ios::failbit)`. Dále, pro načtení jednoho znaku dopředu bez jeho faktického odstranění ze streamu se hodí metoda `is . peek()`. Pozor - chování se liší od načítání ze stringu v konstruktoru.
- Pokud se v operandu objeví celé číslo reprezentované jako ASCIIZ řetězec (např. operand pro +, +=, *, *=, =, ==, ...), předpokládá se, že jej zpracujete konstruktorem `CBigInt`. Pokud zadané číslo nebude správné, vyhodíte výjimku `invalid_argument`. U konstruktoru předpokládáme, že se musí načíst celý řetězec. Tedy například inicializace `CBigInt ("12x")` musí vyhodit výjimku (načítání ze streamu by proběhlo bez chyb, zastavilo by se na znaku x).
- Použití třídy předpokládá operátory přetížené pro mnoho datových typů. Navrhněte rozhraní tak, abyste neměli zbytečně dlouhou implementaci. Využijte vlastností C++ (konstruktor uživ. konverze, implicitní parametry). Rozumný návrh dokáže ušetřit mnoho práce.
- Při implementaci můžete použít `std::vector` a `std::string`.
- Instance `CBigInt` se při provádění aritmetických operací často kopírují (či přesouvají). Pokud se rozhodnete, že budete ve vlastní režii implementovat dynamické alokování prostoru pro ukládaná čísla, budete muset implementovat vlastní kopírující konstruktor, operátor = a destruktor (případně i jejich přesouvací varianty). Takový postup je možný, ale doporučujeme se mu vyhnout. Využijte skládání objektů tak, aby postačovaly kompilátorem automaticky generované konstruktory/operátory=/destruktor (**rule of zero**).
- Funkční řešení této domácí úlohy může být použito pro code review (řešení musí projít povinnými a nepovinnými testy na 100%, nemusí projít bonusovými testy).

Dodatek, 23.3.2020:

- Instance `CBigInt` lze vytvářet i z řetězců se zbytečnými úvodními nulami (např. `CBigInt ( "0012122" )`). Takové řetězce považujte za platná **desítková** čísla.
- Vstupní a výstupní konverze vždy pracuje jen s desítkovými čísly (nezabývá se manipulátory `hex`, `oct`, ...).
- Zvládnutí druhého bonusu je v této úloze poměrně obtížné. Je potřeba zvolit vhodnou reprezentaci čísel a implementovat rychlý algoritmus násobení. Pokud zvolíte reprezentaci jako pole hodnot (např. `vector<int>`) a do prvků pole budete ukládat hodnoty od 0 do 1000000000 (modulo 10^9, de facto budete čísla reprezentovat v číselné soustavě o základu 1 miliarda), nebude Váš program dostatečně rychlý pro druhý bonusový test. Tato reprezentace si vynucuje časté operace dělení/modulo 1 miliarda, které celý výpočet významně zpomalí. Ke zvládnutí druhého bonusového testu vede např. následující:
  - pole hodnot je `vector<uint32_t>`,
  - je využita celá kapacita `uint32_t`,
  - čísla jsou de facto ukládána v soustavě o základu 2^{32},
  - místo dělení/modula 10^9 se použije dělení/modulo 2^{32}, tyto operace se ale dají implementovat mnohem rychleji pomocí bitových posunů, maskování, nebo jsou dokonce provedené implicitně (např. přiřazení z `uint64_t` do `uint32_t` zkopíruje pouze dolních 32 bitů, tedy vlastně provede operaci modulo 2^{32},
  - tato reprezentace si vyžádá změnu algoritmů sčítání a odčítání,
  - operace konverze z/do desítkové soustavy se rovněž zkomplikují,
  - rychlost násobení je ale vyšší, v kombinaci s algoritmem Karatsuba pak vede k úspěchu ve druhém bonusu,
  - druhý bonus je v této úloze náročný, je ale hodnocen 50% body navíc.
- Do ukázkových testů přibyl jeden další test.

Vzorová data: **Download**

☐ **Referenční řešení**

2	**24.03.2020 17:15:10**	**Download**
Stav odevzdání:	Ohodnoceno	
Hodnocení:	7.6435	

- **Hodnotitel: automat**
  - Program zkompilován
  - Test 'Zakladni test s parametry podle ukazky': Úspěch
    - Dosaženo: 100.00 %, požadováno: 100.00 %
    - Celková doba běhu: 0.000 s (limit: 1.000 s)
    - Úspěch v závazném testu, hodnocení: 100.00 %
  - Test 'Test nahodnymi vstupy (op=, op <<)': Úspěch
    - Dosaženo: 100.00 %, požadováno: 50.00 %
    - Celková doba běhu: 0.008 s (limit: 1.000 s)
    - Úspěch v závazném testu, hodnocení: 100.00 %
  - Test 'Test nahodnymi vstupy (op=, op <<, op +)': Úspěch
    - Dosaženo: 100.00 %, požadováno: 50.00 %
    - Celková doba běhu: 0.014 s (limit: 2.000 s)
    - Úspěch v nepovinném testu, hodnocení: 100.00 %
  - Test 'Test nahodnymi vstupy (op=, op <<, op +, op*)': Úspěch
    - Dosaženo: 99.19 %, požadováno: 50.00 %
    - Celková doba běhu: 0.212 s (limit: 1.986 s)
    - Úspěch v nepovinném testu, hodnocení: 99.19 %
    - Nesprávný výstup **[Zpřístupnit nápovědu (328 B)]**
  - Test 'Test nahodnymi vstupy (op=, op <<, op +, op*, op >': Úspěch
    - Dosaženo: 97.97 %, požadováno: 50.00 %
    - Celková doba běhu: 0.111 s (limit: 1.774 s)
    - Úspěch v nepovinném testu, hodnocení: 97.97 %
    - Nesprávný výstup **[Zpřístupnit nápovědu (75 B)]**
    - Nesprávný výstup **[Zpřístupnit nápovědu (210 B)]**
    - Nesprávný výstup **[Zpřístupnit nápovědu (276 B)]**
  - Test 'Test relacnich operatoru (<, <=, ...)': Úspěch
    - Dosaženo: 100.00 %, požadováno: 50.00 %
    - Celková doba běhu: 0.088 s (limit: 1.663 s)
    - Úspěch v nepovinném testu, hodnocení: 100.00 %
  - Test 'Test kopirujiciho konstruktoru': Úspěch
    - Dosaženo: 100.00 %, požadováno: 50.00 %
    - Celková doba běhu: 0.001 s (limit: 1.575 s)
    - Úspěch v nepovinném testu, hodnocení: 100.00 %
  - Test 'Test operatoru=': Úspěch
    - Dosaženo: 100.00 %, požadováno: 50.00 %
    - Celková doba běhu: 0.001 s (limit: 1.574 s)
    - Úspěch v nepovinném testu, hodnocení: 100.00 %
  - Test 'Test operatoru v meznich podminkach': Úspěch
    - Dosaženo: 100.00 %, požadováno: 50.00 %
    - Celková doba běhu: 0.003 s (limit: 1.573 s)
    - Úspěch v nepovinném testu, hodnocení: 100.00 %
  - Test 'Test rychlosti (faktorial)': Program překročil přidělenou maximální dobu běhu
    - Vyčerpání limitu na celý test, program násilně ukončen po: 2.002 s (limit: 2.000 s)
    - Neúspěch v bonusovém testu, hodnocení: Bonus nebude udělen
  - Test 'Test rychlosti (nasobeni pro velka cisla)': Nebylo testováno
    - Neúspěch v bonusovém testu, hodnocení: Bonus nebude udělen
  - Celkové hodnocení: 97.18 % (= 1.00 * 1.00 * 1.00 * 0.99 * 0.98 * 1.00 * 1.00 * 1.00 * 1.00)
- Použité nápovědy: 1
- Penalizace za vyčerpané nápovědy: Není (1 <= 2 limit)
- Celkové procentní hodnocení: 97.18 %
- Bonus za včasné odevzdání: 0.72
- Celkem bodů: 0.97 * (7.15 + 0.72) = 7.64

SW metriky:

	Celkem	Průměr	Maximum	Jméno funkce
Funkce:	**27**	--	--	--
Řádek kódu:	**318**	**11.78 ± 17.34**	**93**	`main`
Cyklomatická složitost:	**64**	**2.37 ± 1.70**	**7**	`CBigInt::CBigInt`

1	24.03.2020 13:44:14	Download
Stav odevzdání:	Ohodnoceno	
Hodnocení:	0.0000	

- **Hodnotitel: automat**
  - ☐ Chyba při základní kompilaci
- Celkové procentní hodnocení: 0.00 %
- Bonus za včasné odevzdání: 0.72
- Celkem bodů: 0.00 * (7.15 + 0.72) = 0.00