

Intervaly		
Termin odevzdání:	10.04.2023 11:59:59	1252464.190 sec
Pozdní odevzdání s penalizací:	21.05.2023 23:59:59 (Penále za pozdní odevzdání: 100.0000 %)	
Hodnocení:	5.6512	
Max. hodnocení:	5.0000 (bez bonusů)	
Odevzdaná řešení:	1 / 25 Volné pokusy + 20 Penalizované pokusy (-2 % penalizace za každé odevzdání)	
Nápovědy:	2 / 2 Volné nápovědy + 2 Penalizované nápovědy (-10 % penalizace za každou nápovědu)	

Úkolem je realizovat třídy `CRange` a `CRangeList`, která budou reprezentovat celočíselné intervaly a jejich seznamy.

Předpokládáme intervaly celých čísel. Interval může obsahovat pouze jedno číslo, nebo více čísel. Jednoprvkové intervaly budeme zapisovat pouze jako jedno číslo, tedy např. 42. Víceprvkové intervaly pak v lomených závorkách. Např. interval `<1 . . 5>` obsahuje čísla 1, 2, 3, 4, 5. Interval celých čísel implementuje třída `CRange`. Protože chceme pracovat s velkým rozsahem čísel, bude tato třída ukládat meze intervalu jako čísla typu `long long` (chceme pracovat i se zápornými čísly). Rozhraní `CRange` je:

```
konstruktor (lo, hi)
    inicializuje instanci na interval v rozsahu lo až hi. Konstruktor kontroluje, že lo ≤ hi. Pokud podmínka není splněna, bude hozena výjimka std::logic_error (standardní výjimka z hlavičkového souboru stdexcept).
další
    Další rozhraní odvoďte podle konstrukcí z příloženého ukázkového kódu, můžete si samozřejmě přidat i další metody závislé na vaší implementaci. Je pouze potřeba dodržet velikost instance - max. 2x velikost long long.
```

Třída `CRangeList` bude reprezentovat seznam takových intervalů. Při skládání intervalů do instance `CRangeList` budeme ukládat intervaly tak kompaktně, jak to jen lze. Tedy například sousedící nebo překrývající se intervaly sloučíme do jednoho delšího intervalu. Instance `CRangeList` bude nabízet rozhraní pro vkládání intervalů, odebírání intervalů, testování hodnoty a výstup. V bonusovém testu bude třída dále nabízet další rozhraní pro zjednodušení používání (viz dále). Konkrétní požadované rozhraní bude:

```
implicitní konstruktor
    inicializuje prázdný seznam intervalů
destruktor, kopírující konstruktor, op=
    budou implementované, pokud jsou zapotřebí a pokud automaticky generovaná podoba nestačí. Doporučujeme použít vhodný kontejner pro ukládání intervalů a kopírování a přesouvání ponechat na automaticky generovaném kódu.
includes(long long) / includes (CRange)
    metody zjistí, zda seznam intervalů obsahuje zadanou hodnotu / celý zadaný interval hodnot. Pokud ano, vrací true, pokud ne, vrací false.
```

```
+=
    pomocí tohoto operátoru se budou přidávat další intervaly do seznamu. Operátor zajistí, že přidané intervaly budou uloženy kompaktně (slučování intervalů). Protože pracujeme s celými čísly, je slučování intervalů poněkud neobvyklé. Je přirozené, že se sloučí překrývající se intervaly, např.:
        • <10 . . 20> a <15 . . 30>,
        • <10 . . 30> a <15 . . 18>, případně
        • <10 . . 20> a <20 . . 30>,

    Sloučí se ale i intervaly <10 . . 20> a <21 . . 30>, přestože se nedotýkají ani nepřekrývají. Dohromady však pokrývají celý interval <10 . . 30>. Intervaly <10 . . 19> a <21 . . 30> se už ale nesloučí, nepokrývají číslo 20.

-=
    pomocí tohoto operátoru se budou odebírat další intervaly ze seznamu. Odebírání lze chápat i jako množinový rozdíl. Operátor zajistí, že po odebrání budou intervaly uloženy kompaktně (odstraňování nepotřebných intervalů).

=
    pomocí tohoto operátoru lze nastavit obsah na seznam intervalů na pravé straně (samozřejmě kompaktně uložený).

==, !=
    operátory porovnávají obsah dvou instancí. Za shodné považujeme dvě instance, které mají identický seznam intervalů.

<<
    operátor zobrazí obsah seznamu do zadaného výstupního streamu. Seznam bude uzavřen ve složených závorkách, jednotlivé intervaly oddělené čárkou. Jednoprvkové intervaly budou zobrazené jako číslo, delší intervaly pak v lomených závorkách. Formát výpisu je zřejmý z ukázek. Intervaly budou zobrazeny seřazené vzestupně.
```

Úloha nabízí i bonusový test, ve kterém je požadováno další rozhraní pro usnadnění práce se seznamem intervalů. Při řešení se můžete rozhodnout, zda budete toto rozšířené rozhraní implementovat (tedy pokusíte se o bonus), nebo ne:

- pokud rozšířené rozhraní nebudete implementovat, ponechte direktivu preprocesoru `EXTENDED_SYNTAX` v komentáři. Chybějící syntaktické konstrukce se nebudou překládat a bonusový test neudělí body navíc.
- Pokud rozšířenou syntaxi budete implementovat, definujte direktivu preprocesoru `EXTENDED_SYNTAX` (odstraňte komentář). V této situaci pak testovací prostředí bude používat rozšířenou syntaxi a bude hodnotit splnění bonusového testu.
- Pozor, pokud definujete direktivu preprocesoru `EXTENDED_SYNTAX` a potřebné rozhraní nebude implementované, skončí kompilace s chybou.

Rozhraní potřebné pro bonusový test syntaxe:

```
konstruktor pro inicializaci seznamem hodnot
    vyplní instanci zadanými intervaly,
range for loop
    instance CRangeList musí být použitelná v range for cyklu, kde iterovat přes jednotlivé intervaly ve vzestupném pořadí,
operátor pro výstup musí zobrazovat interval vždy v desítkové podobě, pokud je stream nastaven jinak, musí nastavení streamu upravit a následně obnovit.
```

Odevzdávejte zdrojový soubor, který obsahuje Vaši implementaci třídy `CRange` a `CRangeList`. 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. Využijte příložený zdrojový kód jako základ Vaší implementace.

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

Úloha nabízí bonusové testy, které zkoušejí další vlastnosti Vaší implementace:

- První bonusový test dále zkouší, zda implementujete rozhraní pro usnadnění práci se třídou `CRangeList` (viz výše),
- Druhý bonus zkouší rychlost operací `includes`, je potřeba rychlejší než lineární řešení.
- Třetí bonus zkouší rychlost operací vkládání a mazání intervalů, je potřeba rychlejší než lineární řešení. Zvládnutí tohoto testu je dost náročné na délku kódu, proto má referenční řešení více řádků než bývá u domácích úloh zvykem (řešení bez tohoto bonusu má rozsah přibližně poloviční).

Nápověda

- Testovací prostředí kontroluje hodnoty ve Vašich objektech tím, že si je zašle do výstupního proudu a kontroluje jejich textovou podobu. Bez správně fungujícího operátoru pro výstup bude Vaše řešení hodnoceno velmi nízkou (téměř 0 bodů).
- Operátor pro výstup implementujte správně -- neposílejte data na `cout`, posílejte je do předaného výstupního proudu. Za výstupem data 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`.
- Za zamyšlení stojí operátory `+=` a `-=`, které musí správně pracovat i se seznamem intervalů na pravé straně. Navíc mezi intervaly na pravé straně lze vkládat operátory `+` a `-` (viz ukázka).
- Při implementaci můžete použít `std::vector`, `std::list` a `std::string`. Zbývající kontejnery z STL ale nejsou dostupné.
- Hranice intervalů jsou zadávané v rozsahu hodnot typu `long long`. V testu mezních hodnot jsou pak zadávané i intervaly s počátky a konci `LLONG_MIN` a `LLONG_MAX`.
- Nesnažte se používat 128 bitové datové typy. Jejich použití povede k chybě kompilace.

Správné řešení této úlohy, které splní závazné **a bonusové** testy na 100%, může být odevzdáno k code review. (Tedy pro code review musíte zvládnout i bonusové testy.)

Vzorová data:	<div>Download</div>
Odevzdat:	<div>Choose FileNo file chosen</div> <div>Odevzdat</div>

Referenční řešení

- Hodnotitel: automat
 - Program zkompilován
 - Test 'Základní test s parametry podle ukázky': Úspěch
 - Dosaženo: 100.00 %, požadováno: 100.00 %
 - Celková doba běhu: 0.000 s (limit: 8.000 s)
 - Úspěch v závazném testu, hodnocení: 100.00 %
 - Test 'Test mezních hodnot': Úspěch
 - Dosaženo: 100.00 %, požadováno: 50.00 %
 - Celková doba běhu: 1.032 s (limit: 8.000 s)
 - Úspěch v závazném testu, hodnocení: 100.00 %
 - Test 'Test náhodnými hodnotami': Úspěch
 - Dosaženo: 100.00 %, požadováno: 50.00 %
 - Celková doba běhu: 0.625 s (limit: 6.968 s)
 - Úspěch v závazném testu, hodnocení: 100.00 %
 - Test 'Test náhodnými hodnotami + mem dbg': Úspěch
 - Dosaženo: 100.00 %, požadováno: 50.00 %
 - Celková doba běhu: 1.396 s (limit: 3.000 s)
 - Úspěch v závazném testu, hodnocení: 100.00 %
 - Test 'Test literálu C++11': Úspěch
 - Dosaženo: 100.00 %, požadováno: 100.00 %
 - Celková doba běhu: 0.000 s (limit: 8.000 s)
 - Úspěch v bonusovém testu, hodnocení: 120.00 %
 - Test 'Test rychlosti (Includes)': Úspěch
 - Dosaženo: 100.00 %, požadováno: 100.00 %
 - Celková doba běhu: 1.442 s (limit: 8.000 s)
 - Úspěch v bonusovém testu, hodnocení: 120.00 %
 - Test 'Test rychlosti (op +/-/+=/=)': Úspěch
 - Dosaženo: 100.00 %, požadováno: 100.00 %
 - Celková doba běhu: 6.353 s (limit: 9.000 s)
 - Úspěch v bonusovém testu, hodnocení: 150.00 %
 - Celkové hodnocení: 216.00 % (= 1.00 * 1.00 * 1.00 * 1.00 * 1.20 * 1.20 * 1.50)
- Celkové procentní hodnocení: 216.00 %
- Bonus za včasné odevzdání: 0.50
- Celkem bodů: 2.16 * (5.00 + 0.50) = 11.88

SW metriky:

	Celkem	Průměr	Maximum	Jméno funkce
Funkce:	27	--	--	--
Řádek kódu:	236	8.74 ± 6.08	21	operator
Cykломatická složitost:	56	2.07 ± 1.84	6	CNode: :next