NPRG041 – 2022/23 ZS – Cvičení MS – **Malý úkol C12**

Gumové pole II

V posledním úkolu dokončíme naši rozpracovanou reprezentaci generického kontejneru gumového pole. Konkrétně převezmeme veškerý stávající kód a rozšíříme jej o implementaci pokročilejších konstruktorů a operátorů přiřazení, stejně jako pro něj naprogramujeme vlastní dopředný iterátor obohacený o některé další nepovinné funkce iterátorů vyšších kategorií.

Pokud jde konkrétně o zmíněné pokročilé konstruktory a operátory přiřazení, implementujeme následující standardní čtveřici:

- `Array(const Array<element>& other)`: kopírovací konstruktor (copy constructor)
- `Array(Array<element>&& other) noexcept`: přesouvací konstruktor (move constructor)
- `Array<element>& operator=(const Array<element>& other)`: kopírovací operátor přiřazení (copy assignment)
- `Array<element>& operator=(Array<element>&& other) noexcept`: přesouvací operátor přiřazení (move assignment)

Pokud jde o vlastní iterátor našeho kontejneru gumového pole, naprogramujeme jej v podobě veřejné vnitřní třídy `iterator` v rámci třídy našeho pole. Nejprve do našeho kódu přidáme knihovnu `<iterator>`, následně do třídy iterátoru přidáme následující tagy, prostřednictvím kterých zpřístupníme požadované informace o chování a možnostech našeho iterátoru.

- `using iterator_category = std::forward_iterator_tag`
- `using difference_type = std::ptrdiff_t`
- `using value_type = element`
- `using pointer = element*`
- `using reference = element&`

Uvnitř našeho iterátoru využijeme dvě datové položky, a sice ukazatel na příslušnou instanci gumového pole a logické číslo aktuální pozice. V návaznosti na tuto představu přidáme jediný konstruktor v podobě `iterator(Array<element>* array, size_t position)`.

Pro naplnění očekávané funkcionality našeho iterátoru implementujeme konkrétně následující nezbytně nutné operátory dopředného iterátoru:

- `bool operator==(const iterator& other) const`: otestujeme rovnost našeho iterátoru vůči jinému iterátoru nad stejným gumovým polem, tedy že oba ukazují na stejnou logickou pozici
- `bool operator!=(const iterator& other) const`: analogicky otestujeme rozdílnost obou iterátorů nad stejným polem, tedy že ukazují na rozdílné pozice
- `iterator& operator++()`: provedeme preinkrementaci našeho iterátoru aneb posuneme aktuální logickou pozici o jedna dál směrem dopředu
- `iterator operator++(int)`: analogicky provedeme postinkrementaci našeho iterátoru
- `reference operator*() const`: provedeme dereferencování našeho iterátoru aneb vrátíme referenci na prvek, na který iterátor aktuálně ukazuje
- `pointer operator->() const`: vrátíme ukazatel na prvek, na který iterátor aktuálně ukazuje

Dodejme, že za všech okolností očekáváme jen korektní používání iterátorů. V opačném případě bude chování vždy nedefinováno. Např. tehdy, když se pokusíme testovat rovnost nebo nerovnost iterátorů patřících jiným instancím gumových polí, když se pokusíme jít za logický konec pole (analogicky před jeho začátek), stejně jako když se pokusíme dereferencovat neplatnou pozici.

Nad rámec již diskutovaných operátorů dobrovolně přidáme ještě několik dalších:

- `iterator& operator--()`: provedeme predekrementaci našeho iterátoru
- `iterator operator--(int)`: analogicky provedeme postdekrementaci našeho iterátoru

- `iterator operator+(const difference_type& movement) const`: posuneme náš iterátor o daný počet pozic, a to dopředu v případě kladného čísla a dozadu v případě záporného
- `iterator operator-(const difference_type& movement) const`: analogicky, akorát v opačném směru

Abychom mohli s našimi iterátory v praktické rovině pracovat, potřebujeme ještě rozšířit implementaci třídy gumového pole jako takového:

- `iterator begin()`: vrátí instanci iterátoru ukazujícího na první prvek našeho gumového pole (je-li takový, jinak za konec)
- `iterator end()`: analogicky vrátí instanci iterátoru ukazujícího za aktuální logický konec našeho pole

Na závěr přemístíme veškerou implementaci našeho gumového pole i iterátorů do vlastního jmenného prostoru s názvem `lib`.

Opět odevzdejte pouze a jenom hlavičkový soubor `Array.h` a dodržte obvyklé požadavky na úkoly kladené. Cílem tohoto úkolu je ilustrovat schopnost návrhu a práce s copy / move konstruktory / operátory přiřazení, návrh vlastního iterátoru a také vytvoření vlastního jmenného prostoru.