# Domáce zadanie 2 – analýza výkonu ADT zoznam

*Cieľom domáceho zadania je porovnanie dvoch implementácií ADT zoznam. V rámci testovania je nutné porovnať výkonnosť štruktúr v konkrétnych scenároch a odhadnúť časovú zložitosť konkrétnych operácií v závislosti na počte prvkov v zozname.*

## Úloha 1 – implementácia

Implementujte ADT zoznam, ktorý musí podporovať operácie definované v súbore **list.h**. Implementáciu realizujte:

- ***v súvislej pamäti*** (**array_list.h**);
- ***v nesúvislej pamäti pomocou obojsmerného zreťazenia*** (**double_linked_list.h**).

V triedach definovaných v súboroch **array_list.h** a **double_linked_list.h** je nutné implementovať všetky verejné metódy a podľa potreby si môžete pridať ľubovoľné atribúty a ľubovoľné súkromné metódy. **Môžete využiť vlastnú implementáciu z cvičení.** Pre implementáciu obojsmerne zreťazeného zoznamu môžete s veľkou výhodou využiť dedičnosť, ak máte vlastnú implementáciu jednostranne zreťazeného zoznamu z cvičení, ktorý môže byť predkom.

**Správnu implementáciu jednotlivých operácií je nutné otestovať prostredníctvom Vami implementovaných jednotkových testov** (unit testov). Tieto implementujte v súboroch **list_test.h** a **list_test.cpp**.

Súbory list.h, array_list.h, list_test.h a list_test.cpp nájdete v projekte využívanom na cvičeniach. Súbor double_linked_list.h tvorí prílohu tohto zadania.

## Úloha 2 – overenie výkonu v scenári

Realizácie ADT zoznam, ktoré ste implementovali v úlohe 1, otestujte v scenároch definovaných v Tab. 1. V každom scenári vykonajte spolu 100 000 operácií. Jednotlivé operácie sú v jednotlivých scenároch volané náhodne tak, aby na konci súhlasil podiel jednotlivých operácií (neplatí, že najskôr sa zavolajú operácie na vkladanie prvkov, potom operácie na mazanie prvkov a nakoniec operácie na sprístupnenie prvkov). Parametre do operácií sú taktiež náhodné; ako index je možné zvoliť akýkoľvek aktuálne platný index.

Tab. 1 Testovacie scenáre pre ADT zoznam

| Scenár | Podiel operácií | | | |
|---|---|---|---|---|
| | `insert` | `reomoveAt` | `at` | `getIndexOf` |
| **A** | 20 | 20 | 50 | 10 |
| **B** | 35 | 35 | 20 | 10 |
| **C** | 45 | 45 | 5 | 5 |

V rámci analýzy výkonu v scenári je nutné merať len dĺžku trvania vybranej operácie. To znamená, že do merania **sa nesmie započítavať čas potrebný pre generovanie pomocných údajov**.

## Úloha 3 – analýza časových zložitostí

V rámci analýzy časových zložitostí je nutné otestovať rýchlosť operácií `insert`, `at` a `removeAt` v závislosti od počtu prvkov a implementácie zoznamu a na základe nameraných a spracovaných údajov **odhadnúť hornú asymptotickú zložitosť jednotlivých operácií**.

V rámci analýzy časových zložitostí vybraných operácií je nutné merať len dĺžku trvania vybranej operácie. To znamená, že do merania **sa nesmie započítavať čas potrebný pre generovanie pomocných údajov**.

## Úloha 4 – bonus

Cieľom bonusovej úlohy je odhadnúť vplyv počiatočnej kapacity a expanznej stratégie na výkon zoznamu uchovávajúceho prvky v súvislej pamäti. Za týmto účelom pridajte parametrický konštruktor, ktorý ako parameter preberá počiatočnú kapacitu poľa. Pre umožnenie menenia expanznej stratégie môžete využiť niekoľko prístupov, napr.:

1. Vytvorte chránenú virtuálnu metódu

   ```
   virtual size_t newCapacity(size_t currentCapacity)
   ```

   ktorá určí novú kapacitu na základe aktuálnej kapacity. V predkovej verzii môže vrátiť dvojnásobok aktuálnej kapacity. Pre jej zmenu vytvoríte niekoľko potomkov (každý prekryje metódu inak) – testovať a porovnávať budete výkon predka a potomkov.
2. Môžete vytvoriť atribút

   ```
   std::function<size_t(size_t)> expandStrategy_;
   ```

   kde typ `std::function` je definovaný v súbore `<functional>`. Do tohto atribútu jednoducho v teste nastavíte funkciu, ktorá sa má zavolať počas expanzie.
   - Dokumentácia: https://en.cppreference.com/w/cpp/utility/functional/function
   - Príklad použitia (vyskúšajte si sami spustiť príklad kódu v dolnej časti stránky a experimentujte s ním): https://docs.w3cub.com/cpp/utility/functional/function

   Ako funkciu môžete použiť napr. lambda funkciu:
   - Dokumentácia: https://en.cppreference.com/w/cpp/language/lambda
   - Príklad: https://blogs.embarcadero.com/lambda-expressions-for-beginners/

Navrhnite spôsob testovania, ktorým objektívne porovnáte vplyv expanznej stratégie a počiatočnej kapacity na výkon implementácie zoznamu uchovávajúceho prvky v súvislej pamäti. Tieto testy následne vykonajte a vyhodnoťte. V rámci testovania je nutné porovnať aspoň dve odlišné expanzné stratégie a pri každej z nich preskúmať vplyv počiatočnej kapacity na výkon zoznamu.

## Výstupy domáceho zadania

Domáce zadanie bude mať nasledujúce výstupy:

1. súbory (*odovzdávate všetky súbory bez ohľadu na to, ktoré časti domáceho zadania riešite*):
    - **array_list.h** – implementácia ADT zoznam uchovávajúceho prvky v súvislej pamäti (úloha 1);
    - **double_linked_list.h** – implementácia ADT zoznam uchovávajúceho prvky v nesúvislej pamäti pomocou obojsmerného zreťazenia (úloha 1);
    - **list_test.h**, **list_test.cpp** – implementácia jednotkových testov ADT zoznam (úloha 1) vrátane výkonnostných testov definovaných v úlohách 2, 3 a 4;
2. CSV súbory s údajmi zaznamenanými počas testov (*odovzdávate len v prípade riešenia úloh 2, 3 a 4*);
3. dokumentácia (*odovzdávate len v prípade riešenia úloh 2, 3 a 4*), ktorá obsahuje:
    - popis realizácie priebehu jednotlivých testov – v prípade úlohy 2 je nutné vysvetliť, ako budete realizovať jednotlivé scenáre, t. j. ako sa budete v danom scenári rozhodovať, ktorú operáciu vykonáte;
    - popis formátu údajov a CSV súborov v testoch – je nutné popísať údaje, zaznamenávané počas testov, a formát CSV súboru (teda, čo sa bude v CSV súbore nachádzať a kde), do ktorého budete tieto údaje ukladať;
    - metodiku spracovania a vyhodnotenia výsledkov testov (CSV súborov) – je nutné popísať, ako budete analyzovať výsledné CSV súbory;
    - prezentáciu výsledkov a záverov vyplývajúcich z testovania.

**V kóde používajte namiesto číselných konštánt symbolické.** Údajové štruktúry a testy musia byť naprogramované tak, aby plne dodržiavali rozhrania špecifikované v súboroch **array_list.h**, **double_linked_list.h** a **list_test.h**, čím ich bude možné bez problémov zakomponovať do projektu AUS používanom na cvičeniach. Údajové štruktúry a testy musia byť Vami naprogramované a **efektívne** implementované z pohľadu výpočtovej a pamäťovej zložitosti. Po ukončení behu projektu AUS s Vašimi súbormi musí byť pamäť **preukázateľne čistá** (nevznikli „memory leak-y“).

Testy vyprodukujú CSV súbor so zaznamenanými údajmi. Vyprodukované CSV súbory je potrebné spracovať (akokoľvek – ručne, pomocou tabuľkového editora, pomocou ďalšej alebo tej istej aplikácie). **Metodické popísanie spôsobu spracovania údajov je súčasťou dokumentácie.** Pomocou popísaného postupu musí byť možné nové údaje spracovať rovnako ako pôvodné. Výsledky spracovania (grafy, priemery, odhady, atď.) je potrebné zdokumentovať a interpretovať (vyvodiť závery). Identifikácia relevantných veličín je súčasťou domáceho zadania.

## Bodovanie domáceho zadania

| Úloha | Zoznam uchovávajúci prvky v súvislej pamäti | Obojstranne zreťazený zoznam | Podmienka |
|---|---|---|---|
| **1 – implementácia** | 2 | 2 | - |
| **2 – overenie výkonu v scenári** | 1 | 1 | správna implementácia štruktúry |
| **3 – analýza časových zložitostí** | 2 | 2 | správna implementácia štruktúry |
| **4 – bonus** | 1 | | musí byť všetko vyššie pre obe štruktúry |

**Aby bolo možné získať uvedený počet bodov za jednotlivé funkcionality, je nutné s každou funkcionalitou odovzdať aj príslušnú časť dokumentácie** (okrem bodu 1).