AISDI - Mini-projekt “asocjacyjne”

Cel ćwiczenia

Zapoznanie się poprzez implementację i testowanie z różnymi strukturami asocjacyjnymi (słownikami).

Zadanie do wykonania

1. Implementacja słownika opartego o drzewo binarne.
2. Implementacja słownika opartego o funkcję mieszającą (hashmapa - tablica z kubełkami).
3. Porównanie efektywności czasowej tych dwóch kolekcji dla co najmniej dwóch różnych operacji.

Instrukcja

Po skopiowaniu plików projektu, należy uruchomić skrypt `create_configs.sh` aby powstały pliki projektów Make i CodeBlocks.

Na przykład:

```
cd KatalogNaProjektyAisdi
cp -r /materialy/AISDI/Mapy .
cd Mapy
./create_configs.sh
```

Opis plików mini-projektu

Katalogi:

- src - katalog zawierający pliki źródłowe klas do zaimplementowania.
- tests - katalog zawierający pliki źródłowe testów jednostkowych.
- Debug - katalog zawierający Makefile’e w trybie Debug i opakowujący je projekt CodeBlocks.
- Release - katalog zawierający Makefile’e w trybie Release i opakowujący je projekt CodeBlocks.

Pliki źródłowe:

- src/TreeMap.h - wydmuszka implementacji struktury drzewa binarnego.
- src/HashMap.h - wydmuszka implementacji hashmapy.
- src/main.cpp - wydmuszka aplikacji do profilowania wybranych struktur.
- tests/TreeMapTests.cpp - testy jednostkowe klasy TreeMap (można dopisywać nowe).

- tests/HashMapTests.cpp - testy jednostkowe klasy HashMap (można dopisywać nowe).
- tests/test_main.cpp - plik wymagany do stworzenia aplikacji wykonującej testy jednostkowe.

Uwagi

- Testy nie są testami czarnej-skrzynki, ale testami "od programisty dla programisty", mającymi pomóc w implementacji. Nie są więc złośliwe, ale jak najbardziej zbliżone do takich, jakie powinny powstać podczas normalnej implementacji. Oznacza to, że można stworzyć kod "złośliwy", zaliczający testy a jednocześnie nie robiący tego co wymagane. Ale prościej będzie skorzystać z pomocy testów i wykonać poprawnie ćwiczenie - Prowadzący będą oceniać kod.
- Wszystkie pliki wolno modyfikować (np. wykomentowywać część testów na czas implementacji jakiejś funkcji). Nie wskazane jest trwałe usunięcie testów - one wychwytują różne sytuacje graniczne, które zapewne będą drobiazgowo weryfikowane przez Prowadzących.
- Dopisywanie własnych testów jest mile widziane.
- Interfejsy klas TreeMap i HashMap przypominają lekko te z biblioteki standardowej C++, ale tylko w zakresie dotyczącym iteracji (aby ułatwić użycie niektórych operacji dostępnych w `std::`). Z premedytacją pozostałe operacje mają często inną syntaktykę i semantykę - proszę czytać testy, by zapoznać się z ich oczekiwanym zachowaniem.
- Warto zweryfikować program testujący narzędziem do wykrywania wycieków pamięci (np. `valgrind`).
- Wycieki pamięci najlepiej wykrywać badając aplikację wykonującą testy jednostkowe. W ten sposób większość operacji będzie zweryfikowana na obecność wycieków.
- Chcąc profilować konkretną operację na kolekcji, warto wykonać ją wielokrotnie dla np. rosnących wielkości kolekcji, żeby zauważyć różnicę.
- Profilowanie ma sens wyłącznie dla kompilacji zoptymalizowanej (`Release`).
- Domyślny tryb budowania (np. `make`, `make all` czy konfiguracja `all` w CodeBlocks) buduje testy, uruchamia je i tylko gdy one przejdą - buduje aplikację do profilowania.
- Niestety CodeBlocks może mieć problemy z parsowaniem wyjścia z testów - wygodniejsze niż przeglądanie widoku błędów "Build messages" powinno być sprawdzenie zawartości okna "Build Log".
- Przełączanie się pomiędzy trybami budowania "Debug" i "Release" wymaga otwarcia dwóch instancji CodeBlocks, ale obydwie wykorzystywać będą te same pliki źródłowe.