Programowanie komputerów 2 – Teleinformatyka
Laboratorium 3: Szablony, wyjątki, strumienie

1. **(1pkt)** W przestrzeni nazw `collections` zdeklarować klasę szablonową `ListElem` umożliwiającą reprezentację elementu listy jednokierunkowej (niech typ `T` oznacza parametr szablonu). Element ma przechowywać prywatnie wskaźnik na element następny oraz daną typu `T`. Zdefiniować publiczny konstruktor inicjalizujący składowe klasy przekazanymi argumentami.

2. **(1pkt)** W klasie `ListElem` zdefiniować publiczne:
   - akcesory dla wskaźnika na kolejny element listy,
   - operator dereferencji `*` zwracający referencję na przechowywaną daną typu `T`,
   - metody `bool isLower(const T& ref)` oraz `isEqual(const T& ref)` porównujące daną przechowywaną w elemencie z przekazaną daną wzorcową.

3. **(1pkt)** W przestrzeni nazw `collections` zdeklarować klasę szablonową `OrderedList` reprezentującą sortowaną listę jednokierunkową (ze względu na typ `T`). Wykorzystać klasę `ListElem` do przechowywania pojedynczych elementów. Ukryć pola klasy przed dostępem z zewnątrz. Klasa `OrderedList` ma udostępniać publiczne:
   - konstruktor bezargumentowy inicjalizujący składowe tak, by zapewnić poprawność działania listy.
   - metodę `getHead()` zwracającą adres pierwszego elementu listy lub `nullptr` jeśli takowy nie istnieje,
   - metodę `getSize()` zwracającą aktualną liczbę elementów (nie zliczać ich przy wywołaniu!).

4. **(2pkt)** Zdefiniować metodę `insert()` dodającą element do listy z zachowaniem porządku (pobiera stałą referencję na wartość typu `T`, nic nie zwraca; wykorzystać metodę `isLower` z klasy `ListElem`). Zaprezentować w funkcji `main` działanie klasy `OrderedList`:
   - utworzyć listę dla typu całkowitego,
   - dodać do listy wartości 1, 2, 0, 3 (w takiej właśnie kolejności),
   - prześledzić w debugerze poprawność struktury (wskaźniki).

5. **(1pkt) Zdefiniować metody:**
   - `search()` – poszukiwanie elementu listy (pobiera wartość typu `T`, zwraca wskaźnik na `ListElem` lub `nullptr` jeśli element nie istnieje; wykorzystać metody `isLower/isEqual` z klasy `ListElem`),
   - `reset()` – usunięcie wszystkich elementów z listy (zwolnić pamięci).

   Zapewnić dealokację pamięci przy destrukcji obiektu. W funkcji `main`:
   - wyszukać element istniejący i nieistniejący,
   - wyczyścić listę, dodać kilka nowych wartości, sprawdzić w debugerze poprawność struktury.

6. **(1pkt)** Zdefiniować metody do zapisu/odczytu całej listy do strumienia wyjściowego/wejściowego:
   - `void serialize(std::ostream &s),`
   - `void deserialize(std::istream &s).`

   W funkcji `main`:
   - wyświetlić listę za pomocą metody `serialize` (przekazać `cout` jako argument),
   - dodać kilka elementów z wykorzystaniem metody `deserialize` (przekazać `cin` jako argument, zakończyć wprowadzanie kombinacją CTRL+D) i wyświetlić ponownie.

7. **(2pkt)** Zdefiniować jawną specjalizację (*explicit specialization*) metod `isLower` oraz `isEqual` dla elementu typu `string`. Specjalizacja ta ma zapewnić, że porównywanie łańcuchów znakowych będzie niewrażliwe na wielkie/małe litery (tak jak w słowniku). Przetestować specjalizację w funkcji `main`:
   - utworzyć listę dla typu string,
   - dodać elementy „Belgia”, „Afryka”, „algorytm”, „bisekcja”,
   - wyświetlić listę.

8. **(1pkt)** w przestrzeni nazw `collections` zdefiniować operatory strumieniowe << i >> dla klasy `OrderedList`. W definicjach wykorzystać metody `serialize` i `deserialize`. Pokazać działanie w funkcji `main`.