Ogólne wymagania dotyczące projektu drugiego

1. Zadanie polega na zaimplementowaniu szablonu klasy oraz przygotowaniu testów jednostkowych dla stworzonego szablonu. Nie ma potrzeby korzystania z żadnych dodatkowych narzędzi do testów jednostkowych, w zupełności wystarczą testy przygotowane i przeprowadzone „ręcznie”. Należy przetestować przygotowany szablon dla dwóch typów danych: dowolnego typu wbudowanego oraz typu złożonego zdefiniowanego przez Państwa (może to być nawet klasa z pierwszego projektu). Każda zaimplementowana metoda powinna być przetestowana przynajmniej raz dla każdego typu danych. Poniżej podane są zalecenia dotyczące przygotowania testów jednostkowych.
2. W tym projekcie zawartość programu testującego sprowadza się do uruchomienia testów jednostkowych – nie ma potrzeby interakcji z użytkownikiem. Trzeba jedynie wyświetlić wyniki wszystkich przeprowadzonych testów w sposób czytelny dla użytkownika.
3. Należy samodzielnie zarządzać pamięcią przydzieloną dynamicznie. Proszę nie używać elementów biblioteki STL, np. klasy `vector`. Proszę też pamiętać o zwalnianiu pamięci przydzielonej dynamicznie.
4. Operacje drukowania i porównywania wymienione w treści zadania powinny zostać zaimplementowane za pomocą przeciążonych operatorów.

Zalecany sposób tworzenia testów jednostkowych

1. Jednej testowanej klasie odpowiada jedna klasa grupująca testy – tzn. oprócz szablonu z treści zadania należy zaimplementować klasę służącą do testowania tego szablonu.
2. Testy są zawarte w metodach klasy testującej, jedna metoda odpowiada jednemu testowi. Nazwa metody testowej powinna odzwierciedlać cel testu. Metoda ta powinna być widoczna publicznie.
3. W każdym teście należy sprawdzić zachowanie operacji klasy szablonowej i stwierdzić, czy po ich wykonaniu wynik zgadza się z przewidywanym. Jeden test powinien sprawdzać poprawność jednej wartości – takie podejście ma ułatwić wykrywanie błędów w testowanym kodzie (w przeciwnym razie stwierdzenie, które instrukcje powodują błąd, jest dodatkowo utrudnione). Oczywiście, może być konieczne wywołanie różnych metod testowanej klasy w celu uzyskania zamierzonego efektu (np. dodawanie i usuwanie elementów kolekcji i sprawdzenie wynikowego rozmiaru).
4. Wszystkie parametry wykorzystywane do tworzenia testowanych obiektów oraz oczekiwane wyniki operacji powinny być jawnie zapisane w kodzie testu. Metody testowe nie powinny przyjmować żadnych argumentów.
5. Do sprawdzania zgodności uzyskiwanych wyników z oczekiwaniami można użyć makra `assert` z pliku `<cassert>` lub wykorzystać wyrażenie warunkowe `if-else`.
6. W testach warto uwzględnić, oprócz typowych przypadków, także sytuacje skrajne – np. dla przypadku kolekcji pustej, jednoelementowej i o stosunkowo dużym rozmiarze.
7. Uruchomienie testów w określonej kolejności powinno następować w funkcji `main`.
8. Przydatne mogą być dodatkowe pola i metody w klasie testującej, np. do wykonania przed pierwszym testem lub po ostatnim teście (czyszczenie).

Przykładowa metoda testowa

Załóżmy, że testowaniu podlega klasa `IntList`, która reprezentuje listę liczb całkowitych. Niech udostępnia ona m.in. operacje:

- `IntList()` – konstruktor bezargumentowy, tworzy pustą listę
- `void add(int number)` – dodaje liczbę na koniec listy
- `int getElement(int index)` – zwraca liczbę znajdującą się na zadanej pozycji (numeracja rozpoczyna się od zera)

Test sprawdzający poprawność dodawania elementów do listy może wyglądać następująco:

```
//metoda testowa ma nazwę określającą cel testu i nie przyjmuje żadnych
//argumentów
void additionTest()
{
	//wykonanie operacji na testowanej klasie
	IntList list;
	list.add(7); //wszystkie parametry są zawarte w kodzie testu
	list.add(2);
	list.add(90);
	list.add(1);
	int expected = 90; //wartość oczekiwana jest zapisana w kodzie testu
	assert(list.getElement(2) == expected); //test sprawdza jedną rzecz
	//wynik testu można wypisać w metodzie testowej lub po powrocie z jej
	//wywołania w funkcji main
}
```