Program semestralny z przedmiotu Programowanie 2

I. Przebieg realizacji i zaliczania projektu:

1. Zrealizować program semestralny w określonym terminie:
- zademonstrowanie projektu (dwa tygodni przed zajęciem zaliczeniowym),
- naniesienie ewentualnych poprawek albo zaliczanie (tydzień przed zajęciem zaliczeniowym);
- zaliczanie programu semestralnego i przedmiotu.

2. Zaliczyć program semestralny. Zaliczenie przebiega indywidualnie. W czasie zaliczenia programu semestralnego student/ka ma potrafić zademonstrować :
a) umiejętność posługiwania się wybranym narzędziem lub narzędziami do realizacji programu semestralnego;
b) zrozumienie kodu – umieć tłumaczyć użytą składnię, w tym w kontekście jej użycia;
c) udowodnić, że pod czas zaliczenia samodzielnie potrafi modyfikować program semestralny.

3. Student(ka) otrzymuje wpis dopiero po wgraniu programu semestralnego (plik *.zip) w nieprzekraczalnym terminie podanym w zadaniu Zaliczenie przedmiotu na platformie Team.
4. Wgrany plik *.zip programu semestralnego powinien zawierać tylko kod źródłowy, niezbędne pliki danych oraz dokumentację.

Na końcową ocenę mają wpływ : aktywność w czasie semestru (oceny cząstkowe), zaangażowanie do rozbudowy kodu oraz dokumentacji, użycie technologii zorientowanych na programowanie w C++ oraz tematów wykraczających poza treści programowe (ANSI C++), obecność, zaliczenie programu semestralnego w terminie.

II. Założenia ogólne

1. Treść programu semestralnego ma pokrywać tematy poruszone na wykładach.
2. Temat programu semestralnego może reprezentować dowolną dziedzinę, którego to wersja końcowa będzie podstawą do zaliczenia laboratoriów.
3. Każdy etap realizacji programu semestralnego jest związany z realizacją konkretnego tematu (składni języka C++) zleconego przez prowadzącego zajęcia.
4. W programie, nazwy zmiennych, funkcji, plików, itd powinni nawiązywać do dziedziny wybranego tematu programu semestralnego.
5. Rozwiązania nie powinny być sztuczne, tzn wszystkie mechanizmy językowe C++ mają być dopasowane do kontekstu (wybranego tematu).
6. Funkcjonalność programu ma demonstrować przetwarzanie danych zgodnie z kontekstem.
7. Każdy nowy etap implementacja programu nie może redukować użytej składni języka C++ z poprzednich etapów, jeśli to nie zostało określono przez prowadzącego zajęcia.
8. Każdy następny etap stanowi temat laboratorium, który zalicza się po zaliczeniu poprzedniego etapu. Tzn., że niewolno łączyć kilka zadań w jednym etapie, w razie niezaliczenia poprzednich laboratoriów.
9. Kod programu powinien być zrozumiały, co osiąga się za pomocą komentarzy, zrozumiałych nazw funkcji (metod), class, zmiennych oraz dokumentacji.
10. Kod nie może generować błędów kompilacji i powinien działać bezbłędnie.
11. Dokumentacja powinna się składać z opisu dziedziny wybranej do tworzenia oprogramowania, powinna demonstrować i wyjaśniać związek opisu dziedziny ze strukturą i funkcjonalnością kodu, wyjaśniać składnię C++ użytej do tych celów (nawiązując przede wszystkim do składni wymaganej do realizacji programu) i zawierać instrukcję użytkownika.

III. Wymogi do składni programu (ocena od 3.0 do 3.5) (*minimum podstawowe)*

1. Program powinien składać się z wielu plików, klas i ***namespace***'ów
2. Klasy powinny głownie mieć pola prywatne i tylko niekiedy chronione
3. Program powinien demonstrować użycie:
   - klas zagnieżdżonych, agregacji, przyjaźni, konstruktorów z listą inicjalizacyjną (domyślnych, kopiujących, delegujących), pola **mutable** w klasach, funkcje przeładowane, przeciążanie operatorów (<<, >>, [], (), =), prywatnego wskaźnika do typu statycznego, odczyt danych z plików i zapis stanu danych programu do pliku.
   - tablic lub kontenerów STL z możliwością zarządzania ich rozmiarami w czasie wykonania. Elementami tablic powinni być wskaźniki do abstrakcji.
   - wielowarstwowej architektury dziedziczenia i polimorfizmu.
4. Wszędzie gdzie jest to tylko możliwe zademonstrować i udowodnić konieczność użycia **const**.

Tematy na większą ocenę:

Użycie kontenerów i algorytmów biblioteki STL różnych typów wraz z własnymi obiektami funkcyjnymi; zarządzanie wyjątkami, wskaźniki inteligentne, semantykę przeniesienia, funkcji oraz klas szablonowych.