PROI 23L – projekt

Wymagania dla grup 101 i 201 (prowadząca: Agnieszka Malanowska)

Etapy projektu

Projekt składa się z dwóch etapów:

- wstępnego, w ramach którego należy przygotować dokumentację wstępną i omówić ją z prowadzącym podczas konsultacji;
- końcowego, w którym należy zaimplementować wskazany projekt oraz zaktualizować i uzupełnić dokumentację. Całość projektu należy zaprezentować prowadzącemu na ostatnich zajęciach.

Dokumentacja powinna zostać umieszczona w pliku **.pdf** na Gitlabie w repozytorium, w którym znajdować się będzie również kod programu. Nazwa repozytorium: **PROI_23L_NrGrupy_TematProjektu**, np.: PROI_23L_101_Sklep.

Dokumentacja wstępna powinna zawierać:

1. imiona i nazwiska autorów oraz temat projektu (nie całą treść zadania);
2. opis wszystkich założeń przyjętych przez zespół;
3. planowany podział na klasy wraz z hierarchią klas i opisem relacji między klasami (w dowolnej formie, graficznej – np. diagram klas UML – lub tekstowej);
4. planowany podział obowiązków w zespole.

Dokumentacja końcowa powinna zawierać aktualizację informacji podanych w dokumentacji wstępnej oraz:

5. opis działania symulacji oraz jej uruchomienia, w tym wyjaśnienie struktury plików z danymi wejściowymi oraz wymaganego formatu argumentów wywołania programu;
6. wskazanie wykorzystanych elementów biblioteki STL wraz z uzasadnieniem wyboru;
7. opis zidentyfikowanych sytuacji wyjątkowych i ich obsługi;
8. opis przeprowadzonych testów.

Ogólne wymagania dotyczące projektów

1. Ogólne wymagania punktowe są zgodne z podanymi w regulaminie przedmiotu. Każdy członek zespołu jest oceniany indywidualnie – konieczna jest umiejętność przedstawienia pracy wykonanej przez siebie poparta commitami na Gitlabie oraz ogólna znajomość całości projektu.
2. Projekt powinien być realizowany systematycznie i zespołowo, co oznacza, że na Gitlabie powinny pojawiać się regularne commity oraz że powinny one być dodawane przez wszystkich członków zespołu, zgodnie z faktycznym podziałem obowiązków w grupie. Proszę wykorzystać możliwości

Gita do pracy grupowej. W razie potrzeby uzyskania z dodatkowych informacji na temat używania Gita najlepiej skorzystać z oficjalnego podręcznika (https://git-scm.com/book/en/v2).

3. Projekt polega na przeprowadzeniu symulacji działania danej organizacji (instytucji) w czasie. Upływ czasu może być modelowany poprzez zastosowanie pętli, której każdy obrót stanowi umowną jednostkę czasu. Symulacja nie zawsze musi kończyć się „sensownie" w sensie „zamknięcia" organizacji na koniec (tzn. np. obsłużenia wszystkich klientów, zwrócenia wszystkich wypożyczonych przedmiotów). Ma trwać przez zadany czas i może skończyć się w takim stanie, w jakim znajdowała się w ostatniej jednostce czasu. Warto natomiast wprowadzić założenie, że różne czynności mogą trwać różną liczbę jednostek czasu.
4. Projekty są sformułowane w sposób ogólny, a podane założenia stanowią jedynie podstawowy schemat działania organizacji. Jedną z części zadania jest uszczegółowienie sposobu działania organizacji. Zalecane jest skonsultowanie przyjętych założeń przed oddaniem projektu. Możliwa jest też ewentualna modyfikacja założeń sformułowanych w treści zadania po uzgodnieniu z prowadzącym.
5. Każdy obiekt wykorzystywany w symulacji powinien posiadać niepowtarzalny identyfikator umożliwiający rozróżnianie obiektów tego samego typu (np. identyfikator liczbowy, unikalna nazwa obiektu).
6. W projekcie istotne jest wykorzystanie różnorodnych elementów języka C++, dlatego w przypadku każdego z tematów należy:
   a. zaprojektować hierarchię klas odpowiadającą treści zadania i przyjętym szczegółowym założeniom (wykorzystanie mechanizmów **dziedziczenia i polimorfizmu**, w tym funkcji wirtualnych); wykorzystanie polimorfizmu musi być widoczne programie – np. operowanie na kolekcji wskaźników do klas bazowych i wykorzystanie efektu wywołania przedefiniowanych metod wirtualnych;

Uwaga: jeśli w rozwiązaniu zostaną zastosowane wskaźniki, powinny to być tzw. **inteligentne wskaźniki** (ang. smart pointers), a nie zwykłe wskaźniki (ang. raw pointers). Najlepiej jest użyć wskaźników typu `std::unique_ptr`.

   b. dobrać odpowiednie typy wykorzystywanych **kolekcji z biblioteki STL** (wektory, listy, kolejki, zbiory, słowniki itp.);
   c. wyświetlać przebieg symulacji w konsoli, a jednocześnie **zapisywać go do pliku** – warto przeciążyć `operator<<` i użyć go do tych dwóch zastosowań. Wyświetlanie w konsoli powinno następować w takim tempie, by użytkownik zdążył przeczytać wypisywany komunikat. W komunikatach należy umieścić identyfikatory obiektów biorących udział w danej akcji (np. „Klient nr 20 kupuje książkę nr 34 od pracownika nr 7");
   d. **wczytywać** parametry programu **z pliku oraz z argumentów wywołania programu** (ale żadnego parametru nie należy wczytywać w procesie interakcji z użytkownikiem);
   e. zastanowić się nad możliwymi przypadkami, w których program nie będzie działał poprawnie i zastosować do ich obsługi **mechanizm wyjątków**. Może to dotyczyć zarówno sposobu działania organizacji (jakaś sytuacja nie powinna się zdarzyć), jak i sposobu działania samego programu (np. nieudany odczyt/zapis do pliku).

Uwaga: poprawne wykorzystanie wszystkich ww. mechanizmów jest ważniejsze niż stopień skomplikowania symulacji. Wymyślna symulacja, która zawiera tylko szczątkowe elementy

wspomnianych konstrukcji, nie zrównoważy braku ich użycia. Graficzna ilustracja symulacji również nie ma wpływu na ocenę.

7. Użytkownik powinien mieć możliwość ustawiania parametrów symulacji, np. długości czasu symulacji (liczby iteracji), liczby obiektów danego typu biorących udział w symulacji (np. liczba obsługujących pracowników, liczba kas w sklepie).
8. **Nazwy plików** (ani ścieżki do plików) **nie mogą być zapisane na sztywno w kodzie**. Powinny zostać wczytane jako argumenty wywołania programu.
9. Należy oddzielić podstawowe klasy od klas odpowiedzialnych za wczytywanie parametrów i wypisywanie danych czy przeprowadzenie symulacji. W funkcji `main` powinno znajdować się uruchomienie symulacji, ale sama symulacja powinna zostać wyodrębniona do osobnej klasy.
10. Program powinien zwalniać wszystkie zaalokowane przez siebie zasoby, w tym pliki.
11. W symulacji korzysta się z generatorów liczb pseudolosowych. Można ich użyć w dowolnych miejscach w programie (np. do losowania, jakie czynności będą wykonane w danej jednostce czasu), ale ważne, by się pojawiły. Do generacji tych liczb proszę nie używać funkcji `rand` z biblioteki języka C, tylko **skorzystać z elementów dostępnych w pliku nagłówkowym `<random>`**. Generatory liczb pseudolosowych powinny być inicjowane innym ziarnem przy każdym uruchomieniu (uwaga na `random_device`, którego losowość zależy od implementacji!) – można wykorzystać do tego aktualny czas, najlepiej posługując się elementami zawartymi w pliku `<chrono>`. Przykład zastosowania elementów biblioteki standardowej znajdujących się w plikach nagłówkowych `<chrono>` i `<random>` do generacji liczb pseudolosowych można znaleźć np. pod adresem http://www.cplusplus.com/reference/random/mersenne_twister_engine/operator()/. Potrzebne może się okazać zastosowanie generatora liczb pseudolosowych w połączeniu z określonym rozkładem, do czego również należy wykorzystać rozkłady dostępne w pliku `<random>`.
12. Klasy przygotowane w ramach projektu należy przetestować za pomocą **testów jednostkowych**.
13. Do projektu należy dołączyć **dokumentację** projektową **w postaci pliku pdf**. Dokumentacja powinna być krótka, ale przygotowana starannie – napisana **poprawnie w języku polskim**. Proszę pamiętać, że jakość dokumentacji również podlega ocenie. Dokumentacja w formacie innym niż **.pdf** nie będzie oceniana. W dokumentacji nie należy umieszczać kodu. Dokumentacja powinna znaleźć się w repozytorium zawierającym kod projektu.
14. Projekt należy umieścić w repozytorium o nazwie zbudowanej wg następującego schematu: **PROI_23L_NrGrupy_TematProjektu**, np.: PROI_23L_101_Sklep.
15. Ocena jest wystawiana po osobistej prezentacji projektu przez cały zespół oraz po zapoznaniu się z dokumentacją i kodem źródłowym przez prowadzącego.