Programowanie obiektowe - Projekt **103B-INxxx-ISP-PROI**		
Projekt zaliczeniowy: **Symulacja restauracji**	**Student:** Ciarka Jakub 270849 Czerwiński Paweł 309431	
Prowadzący: **mgr inż. Agnieszka Malanowska**	**Data:** 1**4.01.2021**	**Ocena:**

Spis treści

Spis treści 1

1 Temat projektu 2

2 Założenia projektowe 3

3 Konstrukcja programu 4

3.1 Logika odliczania czasu 4

3.2 Logika klientów 5

3.3 Logika dań 6

4 Obsługa programu 8

5 Testowanie 9

6 Obsługa błędów 10

7 Użyte elementy biblioteki STL 10

Załączniki 10

1 Temat projektu

Celem projektu jest wykonanie symulatora restauracji. W ramach projektu zostanie kompleksowo odwzorowana logika restauracji, wraz z jej najważniejszymi elementami. Stworzone zostaną klasy różnych klientów, grup klientów, dań, stolików, kelnerów, kuchni i inne. Zostanie odwzorowana logika zachowania klientów (siadanie przy stoliku, wybór dań, jedzenie, płacenie), kelnerów (obsługa klientów, przynoszenie dań z kuchni), kuchni (przygotowywanie zamówień).

Działanie programu nie zakłada interakcji z użytkownikiem. Sterowane będzie ono przez pliki konfiguracyjne. Ponadto niektóre zdarzenia takie jak przybywanie klientów do restauracji, czy wybór dań przez klientów będzie bazował na pseudolosowości.

Projekt będzie realizowany w języku C++ w środowisku Visual Studio jako aplikacja konsolowa. Wynik działania programu będzie prezentowany na terminalu. Ponadto poszczególne zdarzenia rejestrowane będą w plikach tekstowych.

Program zostanie wykonany w oparciu o szczegółowe założenia wydane wraz z tematem projektu. Podsumowanie założeń projektowych wraz z dodatkowymi ustaleniami nie objętymi w poleconym zakresie przedstawiono w kolejnym rozdziale.

2 Założenia projektowe

Symulator restauracji zostaje wykonany w oparciu o następujące założenia:

- W działaniu programu należy przyjąć określony sposób odliczania czasu. Założono, że jednostka czasu odpowiada wykonaniu pojedynczej czynności przez kelnera. Każdy z kelnerów w jednej jednostce czasu może zrealizować jedno z poniższych zadań:
    - zanieść jedno danie z kuchni do stolika;
    - podać menu osobom zasiadającym przy jednym stoliku;
    - przyjąć zamówienia od osób zasiadających przy jednym stoliku;
    - przyjąć opłatę od wszystkich osób zasiadających przy jednym stoliku;

  Czas wykonania wszystkich pozostałych zadań określa się za pomocą ilości zdefiniowanych w powyższy sposób jednostek. Przykładowo ustala się określoną ilość jednostek przygotowania danego typu dania, czas spożywania tego dania, czas wyboru dań przez klientów i inne.

- Liczba kelnerów pracujących w restauracji określana jest są przez odpowiedni plik konfiguracyjny
- Klienci mogą przybywać do restauracji pojedynczo lub w grupach. W programie klienci zawsze zostają opakowani klasą grupy. Grupa może być wiec jednoosobowa lub wieloosobowa,
- Grupa, która przybyła do restauracji może nie być kompletna i oczekiwać jeszcze na kilku członków. Grupa taka oznaczona zostaje stanem WAITING_FOR_FRIENDS i zajmuje stolik większy, niż zawiera w danej chwili członków. Ponadto nie rozpoczyna ona procesu obsługi, dopóki nie zostanie ona skompletowana. Do grupy takiej mogą dosiadać się kolejne grupy,
- Jeśli dana grupa jest kompletna, rozpoczyna ona procedurę obsługi. Nie mogą dosiadać się do niej kolejne osoby,
- Zdefiniowano cztery typy klientów, którzy zamawiają określona strukturę dań:
    - *StandardClient* zamawia jedynie napój i danie główne;
    - *HungryClient* zamawia napój, zupę i danie główne;
    - *DessertLovingClient* zamawia napój, danie główne i deser;
    - *HungryDessertLovingClient* zamawia napój, zupę, danie główne i deser;
- Danie mogą zostać spożywane w określonej kolejności: zupa, danie główne, deser. Napój jest spożywany równolegle do pozostałych dań,
- Zakres menu określany jest przez zestaw plików konfiguracyjnych,
- Liczba stolików w restauracji oraz ich wielkości określone są przez odpowiedni plik konfiguracyjny,
- Poszczególne zdarzenia raportowane są na terminalu oraz do pliku tekstowego.

3 Konstrukcja programu

3.1 Logika odliczania czasu

Jednym z najbardziej podstawowych zadań do realizacji w programie jest odliczanie czasu. Jest to zadanie realizowane przez wszystkie klasy w zbliżony sposób, dlatego warto wydzielić je z ich logiki. W ten sposób powstały interfejsy ITrigger oraz ITriggered oraz odpowiadające im implementacje Trigger i Triggered. Zadania tych klas są następujące:

- Obiekt typu Trigger będzie globalnym licznikiem w programie. Będą w nim rejestrowane obiekty implementujące interfejs ITriggered. Trigger zawiera metodę publiczną execute_iteration(), której zadaniem jest wywołanie metody o tej samej nazwie wszystkich zarejestrowanych obiektów ITriggered.
- Interfejs ITriggered powinien być implementowany przez wszystkie klasy wykonujące pewną logikę w zależności od czasu, a więc praktycznie prawie przez wszystkie klasy w programie. Obiekty te powinny być rejestrowane w liczniku Trigger, aby mogły być przez niego pobudzane w każdej iteracji. Ponadto należy pamiętać, aby ten obiekt wyrejestrować przed zniszczeniem. Interfejs ITriggered zawiera metodę abstrakcyjną execute_iteration(). Należy ją zdefiniować w klasach pochodnych umieszczając w niej pożądaną logikę zależną od czasu.
- Klasa abstrakcyjna Triggered dziedziczy po ITriggered i realizuje logikę rejestracji zgodnie z założeniami RAII. Obiekt dziedziczący po Triggered będzie automatycznie rejestrowany w liczniku Trigger podanym jako argument konstruktora oraz wyrejestrowany podczas jego niszczenia. Klasa Triggered nie implementuje abstrakcyjnej metody execute_iteration(). Musi ona zostać zdefiniowana w klasie pochodnej.

Ponadto niektóre klasy realizują zadania przeciągające się na wiele jednostek czasu. Muszą one zliczać impulsy generowane przez Trigger. Logikę realizacją to zadanie wydzielono do klasy abstrakcyjnej TriggeredCounter, dziedziczącej po Triggered. Zawiera ona metody:

- set_counter(unsigned value) – nastaw odliczaną wartość
- start() – rozpocznij odliczanie
- pause() – zatrzymaj odliczanie
- is_counting() – czy dolicza?

Klasa TriggeredCounter zmniejsza wartość wewnętrznego licznika przy każdym wywołaniu execute_iteration() przez globalny Trigger. Po zliczeniu do zera wywoływana jest metoda abstrakcyjna OnCounted(). Metoda ta musi zostać zaimplementowana przez określone pochodne i może zostać wykorzystana m.in. do realizacji przygotowania dań lub ich spożywania.

Wszystkie klasy wykonujące zadania w reżimie czasowym muszą dziedziczyć po Triggered i TriggeredCounter. Pokazane zostało to na diagramie dziedziczenia przedstawionym w załączniku 1.

3.2 Logika klientów

Schemat „życia” kilenta jest następujący:

1. Obiekt GroupGenerator w sposób losowy tworzy zbiór klientów. Losowy jest typ klienta oraz czas wyboru dania. Klienci tworzeni są w sposób dynamiczny i opakowywani są w unique_ptr, aby zapewnić zniszczenie klienta wraz z klasą Grupy, w której się znajdzie. Zbiór klientów umieszczany jest w grupie. Grupa również jest opakowywana w unique_ptr i trafia do kolejki grup. Pole stanu grupy i stanu wszystkich klientów w grupie nastawiane jest na identyczną wartość z dwóch możliwych: WAITING_FOR_FRIENDS lub READY_TO_BEGIN, w zależności czy jest kompletna, czy będzie oczekiwała na dołączenie kolejnej grupy;
2. Kolejka grup w każdej iteracji stara się przydzielić do stolików jak największą liczbę grup zaczynając od początku kolejki;
3. Grupa jest przydzielana do stolika za pomocą metody place_group klasy stolika. Jeśli stolik jest pusty, metoda ta nastawia odpowiednie wskaźniki przechowywane przez stolik i grupę. Jeśli przy stoliku sieci grupa oczekująca na przyjaciół, rozpoczyna jest proces scalania. Klienci z dosiadającej się grupy są przenoszeni do tej pierwszej, po czym pusta grupa jest niszczona. Stan scalonej grupy zależy od stanu grupy dosiadającej się, więc może ona rozpocząć proces lub oczekiwać na kolejnych klientów;
4. Grupa i zawarci klienci zmieniają stan WAITING_FOR_CARD. Grupa zgłasza w kolejce ServiceQueue chęć wezwania obsługi;
5. Kelner pobiera z kolejki ServiceQueue informacje o grupie i dla wszystkich zawartych w niej klientów wywołuje metodę *take_card*. Klienci zapamiętują wskaźnik na globalne menu restauracji oraz nastawiają licznik na czas wyboru dań. Zmieniają również stan na CHOOSING_DISHES. Potem wywołują metodę *on_client_state_change* grupy, do której należą. Skutkuje to również zmianą stanu grupy na CHOOSING_DISHES;
6. Każdy z klientów w grupie po odliczeniu czasu wyboru dań dynamicznie generuje obiekty dań i przypisuje je do przechowywanych wskaźników (dania będą niszczone wraz z niszczeniem obiektu klienta). Następnie klient zmienia swój stan na READY_TO_ORDER i wywołuje metodę *on_client_state_change* grupy, do której należy. Grupa natomiast czeka, aż ostatni z klientów zmieni swój stan, a następnie również zmienia swój stan i zgłasza chęć obsługi do kolejki serwisów;
7. Kelner wywołuje metodę give_order każdego z klientów, a następnie przekazuje pozyskane zamówienia do kuchni. Klienci następnie zmieniają swój stan na WAITING_FOR_DISHES. Podobnie zmienia stan grupa zawierająca klientów;
8. Kelner przynosi do klientów dania zaraz po ich przygotowaniu w kuchni. Klient po otrzymaniu pierwszego dania zmienia swój stan na EATING. Klient spożywa dania w określonej kolejności: zupa, danie główne, deser. Napój jest spożywany równolegle do pozostałych dań;
9. Rozpoczęcie spożywania dania ma miejsce przez wywołanie metody begin_eating() danego dania. Po ukończeniu jedzenia danie wywołuje metodę klienta *on_dish_state_change*. Po ukończeniu wszystkich dań stan klienta zmienia się na FINISHED_EATING. Wywołuje również metodę *on_client_state_change* grupy, do której należy. Grupa natomiast czeka aż ostatni z klientów zmieni swój stan, a następnie również zmienia swój stan i zgłasza chęć obsługi do kolejki serwisów. Następnie grupa oczekuje na wystawienie rachunku przez kelnera.

10. Kelner wywołuje metodę pay() wszystkich klientów należących do grupy. Klienci oraz grupa zmieniają swój stan na LEAVING. Grupa zgłasza do stolika chęć opuszczenia restauracji wywołując metodę on_group_state_change.
11. Stolik w następnej iteracji usuwa odwołanie do uniqe_ptr opakowującego grupę. Grupa jest niszczona.

Uwaga!

Prawie cała logika funkcjonowania klienta zdefiniowana jest w klasie bazowej StandardClient, stanowiącej podstawowy typ klienta. Pozostałe typy klientów dziedziczą po StandardCllient i jedynie redefiniują metodę chronioną choose_disches, która decyduje które typy dań są wybierane z menu.

3.3 Logika dań

Schemat „życia" dań jest następujący:

1. Obiekt menu wyposażony jest w metody do pobierania ilości dań danego typu w karcie np. *get_main_course_size* oraz pobrania dania danego typu *get_main_course.* Ostatnia z tych metod tworzy instancję danego dania, której zawartość jest przenoszona przez konstruktor przenoszący do obiektu tworzonego dynamicznie w obiekcie klienta. Dzięki takiemu zabiegowi utrzymana zostaje konwencja, że dany obiekt jest tworzony i niszczony przez ten sam obiekt. Jest to alternatywą dla zastosowania inteligentnych wskaźników,
2. Klient losuje numer dania w zakresie pobranym z menu, a następnie za pomocą konstruktora przenoszącego tworzy instancje wybranych dań. Każdy klient zawiera wskaźniki na wszystkie typy dań, ale przypisuje do wybranych z nich typy tylko te dań, które wynikają z typu tego klienta. Przykładowo DesserLovingClient wybiera tylko napój danie główne i deser,
3. Utworzone danie przyjmuje stan CHOOSEN.
4. Kelner pobiera wskaźnik na dania w procesie pobierania zamówień. Następnie przekazuje ten wskaźnik obiektowi klasy Kitchen. Kuchnia wywołuje metodę begin_preparing i przechowuje danie w wektorze on_preparation. Danie zmienia stan na PREPARATION oraz nastawia wewnętrzny licznik, który będzie odliczał czas przygotowania dania,
5. Po mininięciu czasu przygotowania dania jego stan zmieniany jest na PREPARED oraz wywoływana jest metoda on_dish_state_change Obiektu klasy Kitchen. Klasa kitchen przemieszcza wskaźnik na dania do kolejki *prepared*, z której mogą być pobierane przez Kelnerów.
6. Kelnerzy pobierają dania z kolejki *prepared* w kuchni wywołując metodę *deliver_prepared*. Następnie dostarczają dania do klientów.
7. Logikę odbioru dania przez klienta realizuje metoda *pick_up_order*. Logika jedzenia dań we właściwej kolejności znajduje się w tej metodzie oraz w metodzie on_dish_state_change. W pierwszej z nich sprawdzane jest czy danie, które zostało dostarczone spełnia warunki kolejności jedzenia. Jeśli tak, rozpoczyna się jego jedzenie, a danie przyjmuje stan CONSUMPTION. Jeśli nie to danie musi zaczekać na swoją kolej, więc przejmuje stan DELIVERD. Jedzenie dania dostarczonego musi rozpocząć się po skończeniu jedzenia dania poprzedniego w kolejności, więc zadanie to realizuje metoda on_dish_state_change.
8. Rozpoczęcie jedzenia dania skutkuje zmianą jego stanu na CONSUMPTION oraz nastawieniu wewnętrznego licznika dania na czas jego konsumpcji.

9. Po odliczeniu czasu jedzenia stan dania zmieniany jest na EATEN i wywoływana jest metoda klienta on_dish_state_change.
10. Destruktor klasy dania jest wywoływany przez destruktor klasy kilenta.

4 Obsługa programu

Program przy uruchomieniu oczekuje 7 argumentów. Są to kolejno:

1. Plik csv z zupami
2. Plik csv z daniami głównymi
3. Plik csv z desertami
4. Plik csv z napojami
5. Plik csv z stolikami
6. Liczba kelnerów
7. Plik csv z prawdopodobieństwem

Pliki csv z częściami menu mają następującą strukturę kolumn:

1. Czas przygotowania
2. Czas jedzenia
3. Nazwa
4. Cena

Kolumny są oddzielone za pomocą przecinków, a jedno danie zajmuje jedną linijkę pliku.

Plik z stolikami to ilości miejsc przy stoliku oddzielone przecinkami.

Liczba kelnerów to dowolna dodatnia liczba naturalna.

Plik csv z prawdopodobieństwem to 10 wartości oddzielonych przecinkami, mających następujące znaczenie:

1. Szanse na wygenerowanie grupy w jednostce czasu to 1/x
2. Szanse na to, że grupa czeka na znajomych 1/x
3. Maksymalna ilość standardowych klientów w grupie (szanse na każdą poprawną wartość wynoszą 1/x)
4. Maksymalna ilość jednostek czasu przez którą standardowy klient będzie wybierał z menu (szanse na każdą poprawną wartość wynoszą 1/x)
5. Maksymalna ilość głodnych klientów w grupie (szanse na każdą poprawną wartość wynoszą 1/x)
6. Maksymalna ilość jednostek czasu przez którą głodny klient będzie wybierał z menu (szanse na każdą poprawną wartość wynoszą 1/x)
7. Maksymalna ilość głodnych klientów lubiących desery w grupie (szanse na każdą poprawną wartość wynoszą 1/x)
8. Maksymalna ilość jednostek czasu przez którą głodny klient lubiący desery będzie wybierał z menu (szanse na każdą poprawną wartość wynoszą 1/x)
9. Maksymalna ilość klientów lubiących desery w grupie (szanse na każdą poprawną wartość wynoszą 1/x)
10. Maksymalna ilość jednostek czasu przez którą klient lubiący desery będzie wybierał z menu (szanse na każdą poprawną wartość wynoszą 1/x)

Program zapisuje wypisane przez niego informacje do pliku restaurantraport.txt

5 Testowanie

W celu potwierdzenia poprawności działania programu wykonane zostały proste testy funkcjonalne. W pierwszym przykładzie tworzymy tylko jeden stolik dla 10 osób oraz tylko jednego kelnera. Załączono plik test1.txt jest raportem z działania restauracji. Znajdują się w nim następujące informacje:

- Do 4 iteracji nie wygenerowała się grupa, która zmieściłaby się na 10 miejscach;
- W iteracji 5 do restauracji przyszła grupa sześcioosobowa i usiadła przy pierwszym (jedynym) stoliku. Grupa 1 poprosiła kelnera o menu, a kelner je przyniósł. Klienci zaczynają wybierać dania;
- W 6 iteracji klient 6 zadecydował co będzie jeść, w iteracji 7 klienci 1,2,3 oraz 5 zadecydowali co chcą zjeść. W iteracji 9 klient 4 zdecydował co chce zjeść, i jako że cała grupa zdecydowała, proszą kelnera o przyjęcie ich zamówienia;
- W 10 iteracji kelner przychodzi do grupy 1 przy stoliku 1 i przyjmuje ich zamówienie. Kuchnia rozpoczyna przygotowywanie zamówionych dań.
- W następnych iteracjach kuchnia tworzy kolejne dania, kelner je zanosi, a klienci po kolei je jedzą;
- Wszyscy klienci spożywają dania we właściwej kolejności. Warto zwrócić uwagę, że spożycie napoi i zup rozpoczyna się od razu po dostarczeniu, a jedzenie pozostałych dań można rozpocząć dopiero po zjedzeniu poprzednich np. deser w rozważanym przykładzie często był dostarczany przed pozostałymi daniami, ale klient spożywał go dopiero jako ostatnie danie w kolejności;
- Po zjedzeniu wszystkich zamówionych przez grupę dań, grupa woła kelnera, aby zapłacić (34 iteracja);
- Klienci płacą za jedzenie i opuszczają restauracje;
- Następna grupa czekała już w kolejce i jak tylko stolik jest wolny siadają do stolika i cykl czynności się powtarza. Klienci są numerowani po kolei, tak jak dania co pozwala śledzić poszczególne czynności każdego klienta.

Test działania restauracji na przykładzie konfiguracji z 5 stolikami różnych wielkości oraz 5 kelnerami. Załączony plik test2.txt jest raportem z działania restauracji. Znajdują się w nim między innymi następujące informacje:

- W trzeciej iteracji zaczęła posiłek grupa 2 siedząca przy stoliku 2. Stolik 1 jest w tym czasie zajęty przez grupę 1, która czeka na znajomych;
- Dopiero w 13 iteracji grupa pierwsza składająca się z klientów o numerach 1-7 łączy się ze znajomymi oznaczonymi jako klienci 69-70. Gdy są już razem proszą o menu kelnera i otrzymują je w 21 iteracji, a następnie wydarzenia następują normalnym tokiem;
- Kelnerzy wiedzą którzy klienci najdłużej czekają na obsługę i zgodnie z kolejnością wykonują polecenia.

6 Obsługa błędów

Sytuacje wyjątkowe mogą wydarzyć się w następujących sytuacjach:

- Realizacji logiki programu - w poszczególnych funkcjach sprawdzane jest czy stan przetwarzanego obiektu spełnia wymagania związane z algorytmem działania restauracji np. czy danie przyniesione przez kelnera jest w stanie PREPARED. Pojawienie się takiego błędu świadczy, że w kodzie programu pojawił się błąd logiczny. Sytuacja taka powoduje rzucenie wyjątku typu logic_error oraz powoduje przerwanie przetwarzania programu i wypisanie komunikatu wskakującego jakiego elementu dotyczy niewłaściwe funkcjonowanie, a więc który fragment kodu wymaga poprawienia;
- Wprowadzona przez użytkownika ścieżka do pliku konfiguracyjnego nie jest poprawna - zakończenie pracy programu i wypisanie komunikatu do terminala;
- Zawartość plików konfiguracyjnych nie jest zgodna ze spodziewaną zawartością (argumentów jest za mało, litery na miejscu liczb) - zakończenie pracy programu i wypisanie komunikatu do terminala.

7 Użyte elementy biblioteki STL

W programie wykorzystano następujące elementy biblioteki stl:

- vector
- queue
- unique_ptr
- stringstream
- logic_errors
- funkcje: find, find_if, remove
- chrono::system_clock
- mt19937
- move
- stoul

Załączniki

1. diagram_dziedziczenia.png
2. test1.txt
3. test2.txt