Łukasz Lepak, 2I4

Projekt 1801
Dokumentacja projektu
Prowadzący: dr inż. Piotr Witoński

1. Treść projektu i jego założenia

Projekt polegał na zaprogramowaniu grafu modelującego abstrakcyjny system kolejek w fabryce samochodów.
Przyjęto następujące założenia dotyczące grafu:
-węzeł grafu odpowiada stanowisku przetwarzania
-stanowisko produkuje obiekt danego typu
-ze stanowiskiem powiązane są kolejki, z których pobiera ono elementy swojej pracy
-skierowana krawędź grafu odpowiada możliwości przekazania przez stanowisko efektów swojej pracy.

Założenia dotyczące ilości stanowisk:
-4 stanowiska produkujące opony
-2 stanowiska produkujące karoserię
-2 stanowiska produkujące silniki
-2 stanowiska malujące karoserię
-2 stanowiska montujące opony do karoserii
-2 stanowiska montujące silniki w karoseriach z oponami
-2 stanowiska montujące elektronikę w karoseriach z silnikiem i oponami.

Założenia dotyczące symulacji:
-stanowisko zwraca efekt swojej pracy po jednostce czasu
-stanowisko przekazuje efekt swojej pracy do najkrótszej kolejki
-stanowisko nie produkuje nowego obiektu, jeśli wszystkie kolejki, do których może oddać stworzony obiekt, są dłuższe niż 10.

Założenia dotyczące testowania i działania programu
-ilość iteracji systemu kolejkowego, częstość zapisu przebiegu testowania do pliku oraz czas wstrzymania programu na zapoznanie się z danymi są podawane przez użytkownika w pliku, którego nazwę wpisuje w programie, a w przypadku braku pliku po wpisaniu komendy *default* zostaną ustawione wartości domyślne
-przebieg testowania jest zapisywany do pliku *przebieg_testowania.txt*, który zostanie stworzony, jeśli nie istniał wcześniej, a w przeciwnym wypadku jego zawartość zostanie usunięta
-do pliku zapisuje się numer iteracji symulacji, stan kolejek oraz mogą być zapisane stworzone obiekty w danej iteracji.

Założenia dotyczące domyślnych wartości w przebiegu testowania:
-ilość iteracji – 40 iteracji (stała DEFAULT_ITERACJE)
-częstość zapisu do pliku – co 5 iteracji (DEFAULT_ZAPIS)
-czas wstrzymania – 2 sekundy. (DEFAULT_WSTRZYMAJ)

Wizualizacja symulacji została zrealizowana za pomocą listy kolejek wraz z ilością elementów w każdej z nich. Dodatkowo istnieje możliwość wyświetlenia stworzonych obiektów oraz przyczyn tego, dlaczego dany obiekt nie został stworzony.

2. Podział na klasy

2.1. Założenia dotyczące przepływu grafu

Aby symulacja była jednoznaczna i nie zależała od losowych czynników, zostały przyjęte założenia dotyczące przepływu grafu:
-pomalowana może zostać tylko „pusta” karoseria
-opony są montowane tylko do pomalowanej karoserii
-silniki są montowane tylko w karoseriach z oponami
-elektronika jest dodawana tylko do karoserii z silnikami
-wszystkie stanowiska pracują równocześnie – w przypadku, gdy w jednej kolejce jest 11 elementów, a w drugiej 10 elementów (zakładając, że są dwie kolejki, do których można oddać gotowy obiekt) wszystkie stanowiska wykonają swoją pracę, więc mogą zdarzyć się kolejki dłuższe niż 11 elementów

Takie założenia sprawiają, że w programie powstaje hierarchia klas, którą będzie można reprezentować za pomocą relacji dziedziczenia.

2.2. Opis klas

W programie klasy są zadeklarowane w plikach *Karoseria.h, Silnik.h, Opona.h* oraz *MyException.h,* a definicje metod do nich należących znajdują się w odpowiadających im plikach *Karoseria.cpp, Silnik.cpp, Opona.cpp* oraz *MyException.cpp*

W programie są zadeklarowane 3 klasy abstrakcyjne:
-klasa *Silnik* – zawiera wspólne dla każdego silnika pola określające masę, moc, pojemność, rodzaj używanego paliwa oraz metody czysto wirtualne zwracające parametry silnika

-klasa *Opona* – zawiera wspólne dla każdej opony pola określające masę, średnicę, rodzaj bieżnika oraz metody czysto wirtualne zwracające parametry opony
-klasa *Karoseria* – zawiera wspólne dla każdego typu karoserii pola określające masę, rodzaj nadwozia, markę samochodu oraz metody czysto wirtualne zwracające parametry danego rodzaju karoserii.

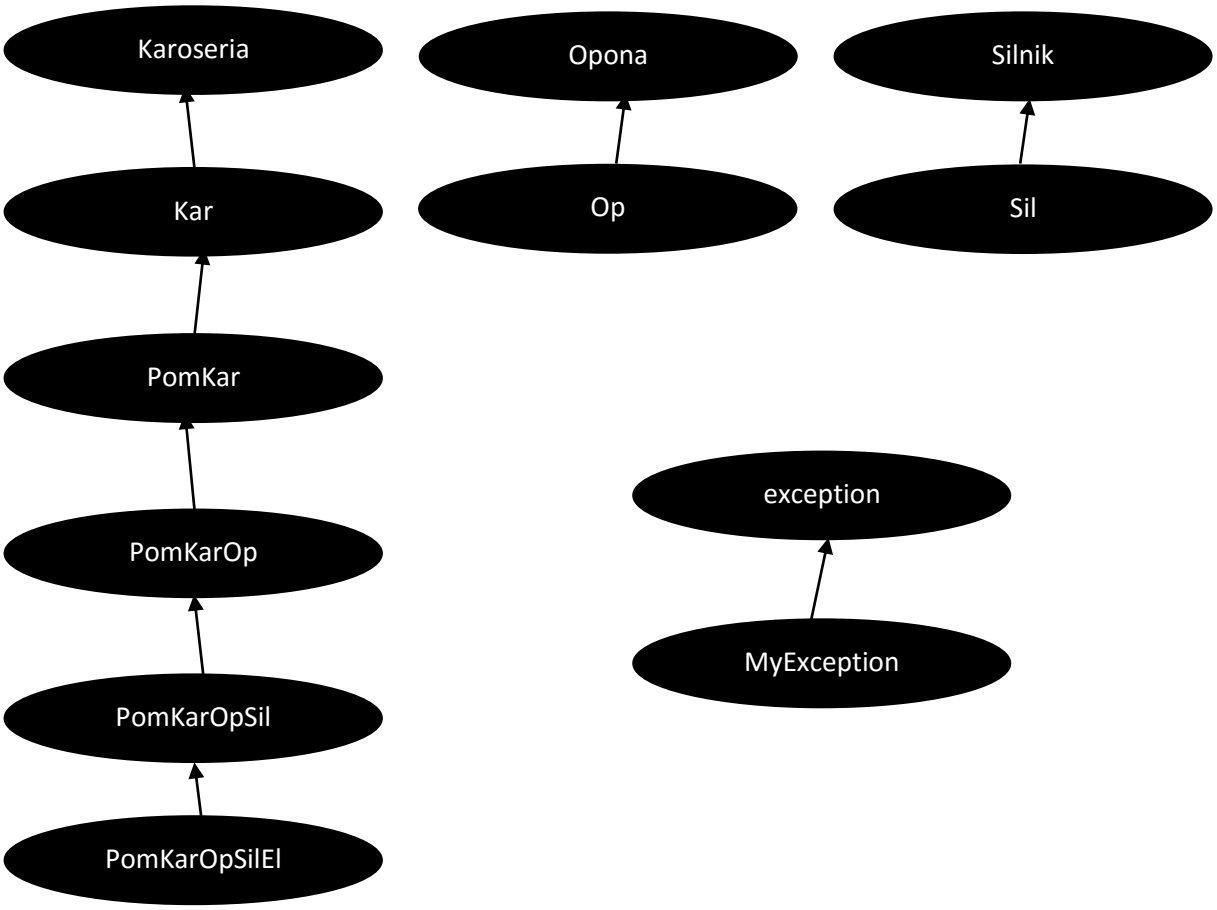
Po klasie *Silnik* dziedziczy klasa *Sil* reprezentująca obiekt będący silnikiem, po klasie *Opona* dziedziczy klasa *Op* reprezentująca obiekt będący oponą, a po klasie *Karoseria* dziedziczy klasa *Kar* reprezentująca obiekt będący karoserią. Każda z tych klas implementuje metody wirtualne zadeklarowane jako czysto wirtualne w odpowiednich klasach abstrakcyjnych.

Inne klasy reprezentujące obiekty symulacji to:
-klasa *PomKar* – reprezentuje pomalowaną karoserię, dziedziczy po klasie *Kar* oraz zawiera pole reprezentujące kolor
-klasa *PomKarOp* – reprezentuje pomalowaną karoserię z oponami, dziedziczy po klasie *PomKar* oraz zawiera 4 obiekty klasy *Op*
-klasa *PomKarOpSil* – reprezentuje pomalowaną karoserię z oponami i silnikami, dziedziczy po klasie *PomKarOp* oraz zawiera 1 obiekt klasy *Sil*
-klasa *PomKarOpSilEl* – reprezentuje pomalowaną karoserię z oponami, silnikiem i elektroniką, dziedziczy po klasie *PomKarOpSil* oraz zawiera informację o występowaniu oraz masie elektroniki.

Każda z tych klas implementuje metody wirtualne zadeklarowane w odpowiednich klasach wirtualnych z odpowiednimi zmianami, np. klasa *PomKarOp* zwraca swoją masę jako masę karoserii oraz czterech opon.

W programie została zdefiniowana również klasa *MyException* dziedzicząca po wbudowanej klasie *exception,* która została użyta do przechwytywania wyjątków, które mogły wystąpić w programie wraz z wiadomością informującą o tym, dlaczego dany wyjątek wystąpił.

2.3. Hierarchia klas



3. Koncepcja rozwiązania

Deklaracje funkcji związanych z symulacją znajdują się w pliku *Symulacja.h,* ich definicje znajdują się w odpowiadającym pliku *Symulacja.cpp*. Funkcje te są wywoływane z pliku *main.cpp.*

Działanie programu jest podzielone na następujące etapy:
-deklaracja używanych zmiennych oraz kontenerów
-wczytanie danych dotyczących przebiegu symulacji z pliku
-wykonywanie oraz testowanie symulacji, zapis do pliku

3.1. Deklaracje zmiennych oraz kontenerów

Na początku programu zostają zadeklarowane zmienne, które będą przechowywały informacje o ilości iteracji symulacji, częstości zapisu do pliku, czasie wstrzymania oraz o tym, czy dana iteracja zostanie zapisana do pliku. Następnie otwierany jest plik *przebieg_testowania.txt.* Jeśli plik ten nie istniał, to zostanie utworzony, a w przeciwnym wypadku jego zawartość zostaje usunięta. Potem zadeklarowane są kontenery do przetrzymywania stworzonych obiektów. Zostają utworzone kontenery kolejek z biblioteki STL, które będą odpowiadały za system kolejkowy w symulacji. Są one stworzone w tablicach dwuelementowych – kolejki o indeksach 0 będą dołączone do stanowisk nr 1, a te o indeksie 1 – do stanowisk nr 2. Zostaje również stworzony kontener tablicy o dynamicznie zwiększającym się rozmiarze, który będzie przechowywać gotowe obiekty klasy PomKarOpSilEl.

3.2. Wczytywanie danych z pliku

Użytkownik zostaje poproszony o podanie nazwy pliku, w którym znajdują się wpisane przez niego dane początkowe dotyczące przebiegu symulacji. Podczas tego etapu jest możliwych kilka scenariuszy:
-podany plik będzie zawierał poprawne dane
-podany plik nie będzie zawierać żadnych danych
-podany plik nie będzie zawierać kompletnych danych
-podany plik będzie zawierać błędne dane
-podany plik będzie zawierać zbyt wiele danych
-plik nie będzie istniał
-użytkownik nie będzie chciał podać pliku.

W pierwszym przypadku funkcja oczywiście wykona się bez żadnych komplikacji. Jeśli plik nie będzie zawierać żadnych danych, funkcja przypisze zmiennym odpowiadającym za dane symulacji wartości domyślne. Z pliku zawierającego niekompletne dane zostaną wczytane tylko te, które istnieją – reszta zostanie ustawiona na wartości domyślne. Z pliku o błędnych danych zostaną wczytane tylko prawidłowo wpisane dane, a reszta będzie domyślna. Z pliku o zbyt wielu danych zostaną wczytane tylko te z początku. Jeśli dany plik nie istnieje, użytkownik zostanie poproszony ponownie o wpisanie nazwy pliku. Jeśli użytkownik jednak nie będzie chciał podać pliku, to będzie mógł wpisać komendę *default*, która ustawi wartości zmiennych na domyślne.

3.3. Przebieg symulacji oraz zapis do pliku

Na początku symulacji zostają zadeklarowane wskaźniki na obiekty odpowiedniego rodzaju, na których potem będą tworzone obiekty odpowiednich klas. Przed próbą utworzenia obiektu do każdego wskaźnika zostaje przypisany NULL po to, aby żaden z nich nie został niezainicjalizowany. Następnie symulacja próbuje stworzyć obiekt według założeń programu. Jeśli wszystkie kolejki, do których dany obiekt może trafić, są dłuższe niż 10, zostanie wygenerowany wyjątek, a obiekt nie zostanie utworzony. Podobna sytuacja zdarzy się, jeśli kolejki, z których dany obiekt pobiera elementy są puste albo nie zawierają wystarczającej ilości elementów. W przeciwnym wypadku obiekt zostanie stworzony i informacja o tym pojawi się na ekranie jeśli stała STWORZENIE jest ustawiona na 1. Po stworzeniu elementów te, które udało się utworzyć, są dodawane do odpowiednich kolejek z tym, że zawsze wybierają krótszą kolejkę, a w przypadku tych samych długości – kolejkę z indeksem 0. Potem na konsoli zostaje wyświetlony stan kolejek po danej iteracji i – jeśli zmienna *czy_do_pliku* ma wartość *true* – zostaje on zapisany do pliku. Do pliku zapisywane są też informacje o stworzonych obiektach, jeśli stała STWORZENIE jest ustawiona na 1. Ustawienie stałej WYJATEK na 1 spowoduje wypisywanie przy każdej iteracji symulacji informacji o powodach niestworzenia niestworzonych obiektów. Po zakończeniu symulacji użytkownik jest pytany o to, czy chce powtórzyć symulację przez odpowiednie wpisanie komendy *tak* lub *nie.*

4. Kompilacja

Kompilacja programu odbywa się za pomocą pliku *makefile* znajdującego się w katalogu plików programu. Wpisanie w terminalu komendy *make* spowoduje powstanie pliku *projekt3.out* za pomocą kompilatora *g++*. Wpisanie *make clean* spowoduje usunięcie wszystkich plików z rozszerzeniem *.o* powstałych w trakcie kompilacji programu.

5. Obserwacje i wnioski

Po ustawieniu odpowiednio dużej ilości iteracji można zauważyć, że stan kolejek w fabryce zaczyna skakać między dwoma stanami – dąży on do stanu stabilnego. Jest to spowodowane tym, że w pewnych miejscach tworzą się większe kolejki niż w innych.

Po wielu próbach testowania programu z różnymi danymi okazało się, że największe kolejki robią się w kolejkach karoserii do pomalowania, kolejkach pomalowanej karoserii do montowania opon oraz w kolejkach silników do zamontowania ich w karoseriach.