Laboratorium

Programowania Komputerów

# Temat: Biuro podróży

Autor: Alicja Lachman

Informatyka, semestr 3, grupa 1

Prowadząca: dr inż. Karolina Nurzyńska

1. **Temat**

Implementacja programu do obsługi biura podróży w języku C++.

1. **Analiza, projektowanie**

Biuro podróży obsługuje wycieczki objazdowe oraz wczasy. Dodatkowo istnieje możliwość łączenia wycieczki objazdowej z wczasami rozpoczynającymi się po jej zakończeniu w tym samym kraju (wycieczka łączona). Oferta biura podróży generowana jest do pliku wyjściowego (jego nazwa podawana jest w linii poleceń jako argument programu) na podstawie ofert zawartych w podanym przez użytkownika folderze z ofertami. W zależności od wybranego rodzaju wycieczki, biuro podróży szuka wycieczek objazdowych, wczasów, wycieczek łączonych lub wycieczek dowolnych.

Działanie programu jest następujące:

- Program sprawdza poprawność argumentów wywołania programu. Jeśli są one poprawne, zostaje uruchomione właściwe biuro podróży,

- Odczytuje kolejno wszystkie pliki z ofertami znajdujące się w podanym folderze,

- W zależności od tego, czy w danym pliku znajduje się wycieczka objazdowa („WO:” na początku pliku) czy wczasy („WP:” na początku pliku), wskaźnik na nowo utworzoną wycieczkę zostaje dodany do wektora wycieczek,

- Program sprawdza w argumentach programu, jakiego typu wycieczki chce szukać użytkownik a następnie przeszukuje wektor wycieczek w poszukiwaniu odpowiedniego typu wycieczki spełniającego zadane przez użytkownika kryteria. Po znalezieniu wczasów sprawdzana jest również możliwość zwielokrotnienia danego turnusu. Jeśli użytkownik wybierze wycieczkę łączoną (-r l), szukanie rozpoczyna się od znalezienia wycieczki objazdowej spełniającej dane kryteria. Następnie dla każdej znalezionej wycieczki objazdowej szukane są wczasy rozpoczynające się w tym samym kraju co ostatni punkt wycieczki objazdowej, a także spełniające kryterium daty i ceny.

1. **Specyfikacja zewnętrzna**
   1. **Obsługa programu**

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:

-k katalog z plikami z ofertą (katalog/)

-o plik wyjściowy (plik.txt)

-r rodzaj wycieczki (o – objazdowa, p – wczasy, l – łączona, d – dowolna)

-t data, od której rozpoczyna się poszukiwanie (RRRRMMDD)

-d długość wycieczki

-c maksymalna cena

Kolejność przełączników jest dowolna. Po uruchomieniu programu program wyszukuje wycieczki spełniające zadane kryteria. W przypadku wystąpienia błędów, na ekranie wypisywane są odpowiednie komunikaty.

* 1. **Format danych wejściowych**

Program wczytuje z argumentów uruchomienia programu ścieżkę do folderu, w którym znajdują się pliki tekstowe z ofertami. Każda oferta znajduje się w osobnym pliku. Przyjęto następujący format danych zgodny z założeniami projektu:

Wycieczka objazdowa:

WO: Nazwa Rok.Miesiąc.Dzień Rok.Miesiąc.Dzień Miasto[Państwo],(….) Cena ŚrodekTransportu

Przykładowy plik:

WO: Objazdowka 2015.06.07 2015.06.21 Warszawa[Polska], Malbork[Polska], Berlin[Niemcy] 3000 Autokar

Wczasy:

WP: Nazwa Rok.Miesiąc.Dzień DługośćTurnusu Miasto Państwo KosztWycieczki ŚrodekTransportu1 JegoCena ŚrodekTransportu2 JegoCena ŚrodekTransportu3 JegoCena

Przykładowy plik:

WP: Egzotyczne 2015.09.12 15 Havana Kuba 4500 Samolot 3000 Autokar 200 Wlasny 100

* 1. **Komunikaty**

Program komunikuje się z użytkownikiem za pomocą komunikatów wyświetlanych w konsoli.

W pierwszej kolejności sprawdzana jest poprawność wprowadzonych argumentów wywołania programu. W przypadku nieprawidłowej ilości argumentów wyświetlany jest komunikat Zla ilosc argumentow wywolania programu!! Nastepuje zamkniecie programu! oraz następuje zamknięcie programu.

Po poprawnym zakończeniu pracy programu na ekranie wyświetla się następujący komunikat:

Oferta biura podrozy zostala wygenerowana i znajduje sie w pliku: (nazwa pliku)

Znaleziono x wycieczek spelniajacych dane kryteria.

Podczas wczytywania wycieczek z plików program sprawdza poprawność zawartych w nich danych. W przypadku wykrycia niepoprawnych danych, które mogłyby wpłynąć na poprawność działania programu (na przykład zły format daty czy też cena wycieczki podana w postaci słownej), dana wycieczka zostaje pominięta a na ekranie zostaje wyświetlony odpowiedni komunikat, na przykład:

Wykryto plik z nieznanym typem wycieczki

wykryto blad w cenie jednych z wczasow!

Zla cena ktoregos ze srodkow transportu! Te wczasy zostana pominiete!

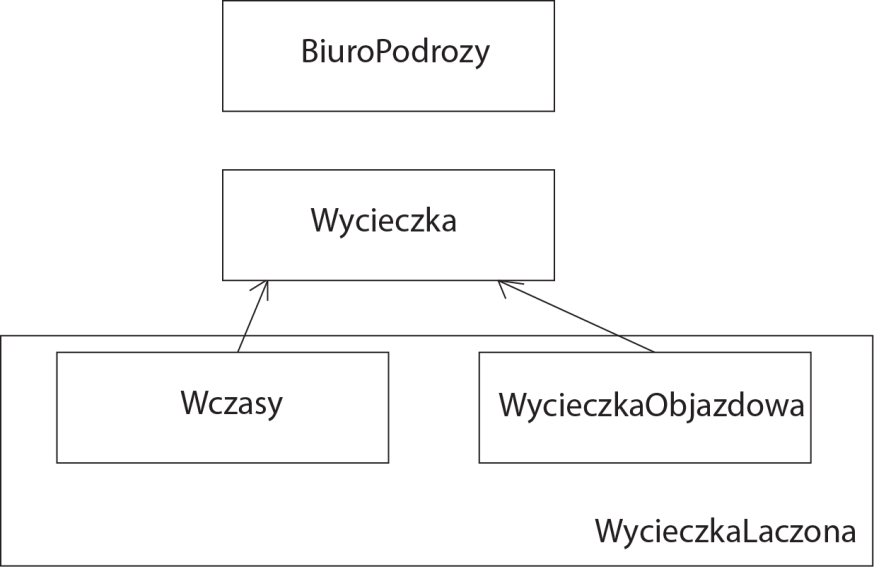
Zly rodzaj transportu! Te wczasy zostana pominiete!

Zly format daty lub dlugosc turnusu jednych z wczasow. Te wczasy zostana pominiete!

Zly format daty! Ta objazdowka zostaje pominieta!

zly rodzaj transportu. Ta objazdowka zostaje pominieta!

1. **Specyfikacja wewnętrzna**
   1. **Hierarchia klas**

****

* 1. **Opis klas**

|  |  |  |
| --- | --- | --- |
| **Klasa** | **Rodzaj** | **Opis** |
| BiuroPodrozy | - | Klasa obsługująca pracę biura podróży. Umożliwia odczyt ofert z plików oraz wyszukiwanie wycieczek o zadanych kryteriach |
| Wycieczka | Abstrakcyjna klasa bazowa | Klasa bazowa dla poszczególnych rodzajów wycieczek. Zawiera pola wspólne dla klas potomnych opisujące nazwę wycieczki, datę jej rozpoczęcia oraz jej koszt. Zawiera dwie czysto wirtualne metody: void sprawdz(BiuroPodrozy \*biuro) oraz void drukuj(BiuroPodrozy \*biuro, int mnoznik). |
| Wczasy | Klasa potomna | Klasa opisująca wczasy. |
| WycieczkaObjazdowa | Klasa potomna | Klasa opisująca wycieczki objazdowe. |
| WycieczkaLaczona | - | Klasa opisująca wycieczkę łączoną, zawiera referencję do obiektu typu wczasy oraz do wycieczki objazdowej. |

* 1. **Klasa BiuroPodrozy**

|  |  |
| --- | --- |
| **Metoda** | **Opis** |
| void czytajPliki(); | Funkcja czytająca kolejne pliki znajdujące się w podanym folderze. Korzysta ze struktury dirent z biblioteki POSIX. |
| void interpretujDane(string zawartosc); | Funkcja sprawdzająca, czy wczytany plik zawiera opis wczasów czy wycieczki objazdowej. Uzyskany wskaźnik na odpowiedni obiekt zostaje dodany do wektora wskaźników na wycieczki. |
| Wczasy\* tworzWczasy(vector <char\*> opis); | Funkcja zwracająca wskaźnik na nowo utworzone wczasy. Na podstawie przekazanego z pliku opisu wypełniane są kolejne pola wczasów. |
| WycieczkaObjazdowa\* tworzObjazdowke(vector <char\*> opis); | Funkcja zwracająca wskaźnik na nowo utworzną objazdówkę. Na podstawie przekazanego z pliku opisu wypełniane są kolejne pola objazdówki. |
| void szukajWycieczek(); | Funkcja sprawdzająca jakiego typu wycieczek należy szukać. |
| void szukajWczasow(); | Funkcja przeszukująca wektor wskaźników na wycieczki w poszukiwaniu wskaźników wskazujących na wczasy. |
| void szukajObjazdowek(); | Funkcja przeszukująca wektor wskaźników na wycieczki w poszukiwaniu wskaźników wskazujących na objazdówki. |
| int getIloscWycieczek(); | Funkcja zwracająca ilość znalezionych wycieczek spełniających zadane kryteria. |
| void zwiekszLicznikWycieczek(); | Funkcja inkrementująca licznik wycieczek spełniających kryteria. |

|  |  |
| --- | --- |
| **Zmienna** | **Opis** |
| vector <Wycieczka\*> lista\_wczasow; | Wektor wskaźników na obiekty typu Wycieczka, zawiera wszystkie wycieczki poprawnie wczytane z plików wejściowych |
| int liczbaWycieczek | Zawiera liczbę wszystkich wycieczek spełniających zadane kryteria |

* 1. **Klasa Wycieczka**

|  |  |
| --- | --- |
| **Metoda** | **Opis** |
| virtual void sprawdz(BiuroPodrozy \*biuro)=0; | Czysto wirtualna metoda sprawdzająca czy dana wycieczka spełnia kryteria podane przez użytkownika |
| virtual void drukuj(BiuroPodrozy \*biuro, int mnoznik=1)=0; | Czysto wirtualna metoda drukująca daną wycieczkę do pliku wyjściowego |
| bool porownajDate(struct tm data1, struct tm data2); | Funkcja sprawdzająca czy data1 jest większa lub równa data2 |
| int obliczIloscDni(struct tm data1, struct tm data2); | Funkcja obliczająca czas w dniach pomiędzy dwiema datami |
|  |  |
| **Zmienna** | **Opis** |
| string nazwa; | Zawiera nazwę wycieczki |
| struct tm data\_rozpoczecia; | Zawiera datę rozpoczęcia wycieczki |
| float kosztWycieczki; | Zawiera koszt wycieczki |

* 1. **Klasa Wczasy**

|  |  |
| --- | --- |
| **Metoda** | **Opis** |
| void sprawdz(BiuroPodrozy \*biuro); | Metoda sprawdzająca, czy dane wczasy spełniają kryteria biura podróży. |
| void drukuj(BiuroPodrozy \*biuro, int mnoznik=1); | Metoda drukująca wczasy do pliku. Zmienna mnożnik pozwala na uwzględnienie wielokrotności turnusów. |
|  |  |
| **Zmienna** | **Opis** |
| float koszt\_autokar; | Zawiera koszt dojazdu autokarem |
| float koszt\_samolot; | Zawiera koszt dojazdu samolotem |
| float koszt\_wlasny; | Zawiera koszt dojazdu własnego |
| int dlugosc\_turnusu; | Informuje o długości turnusu |
| string destynacja\_kraj; | Przechowuje informację o kraju wczasów |
| string destynacja\_miasto; | Przechowuje informację o mieście wczasów |

* 1. **Klasa WycieczkaObjazdowa**

|  |  |
| --- | --- |
| **Metoda** | **Opis** |
| void sprawdz(BiuroPodrozy \*biuro); | Metoda sprawdzająca, czy dana objazdówka spełnia kryteria biura podróży. Jeśli użytkownik wybierze dowolny rodzaj wycieczki lub wycieczki łączone, sprawdzane są również wycieczki łączone. |
| void drukuj(BiuroPodrozy \*biuro, int mnoznik=1); | Metoda drukująca objazdówkę do pliku. |
|  |  |
| **Zmienna** | **Opis** |
| string dojazd; | Zawiera informację o rodzaju transportu |
| struct tm data\_zakonczenia; | Zawiera datę zakończenia wycieczki |
| vector <string> lista\_miast; | Wektor zawierający nazwy miast odwiedzonych podczas wycieczki |
| set <string> lista\_krajow; | Zbiór (niepowtarzających się) krajów odwiedzonych podczas wycieczki |
| string ostatni\_kraj; | Nazwa ostatniego odwiedzonego kraju, służy do porównywania z krajem wczasów podczas poszukiwania wycieczek łączonych |

* 1. **Klasa WycieczkaLaczona**

|  |  |
| --- | --- |
| **Metoda** | **Opis** |
| void sprawdz(BiuroPodrozy \*biuro); | Funkcja sprawdzająca, czy dana wycieczka łączona spełnia kryterium ceny oraz daty rozpoczęcia |
| void drukujLaczona(BiuroPodrozy \*biuro); | Funkcja drukująca wycieczkę łączoną do pliku wyjściowego |
|  |  |
| **Zmienna** | **Opis** |
| Wczasy &wczasy; | Referencja do obiektu wczasy. Wczasy odbywają się w kraju zakończenia wycieczki objazdowej |
| WycieczkaObjazdowa &objazd; | Referencja do obiektu wycieczka objazdowa. |

Dla klas Wycieczka, Wczasy oraz WycieczkaLaczona pominięto opis metod zwracających wartość pól prywatnych <typ> Klasa::get<nazwaPola>(); ze względu na ich oczywistość.

1. **Testowanie i uruchamianie**

Program został przetestowany dla dużej ilości plików z ofertami. Sprawdzano również działanie programu w przypadku błędnego formatu danych zawartych w niektórych plikach. Dzięki zastosowaniu odpowiednich zabezpieczeń (obsługa wyjątków) program właściwie reaguje na błędne dane i nie zawiesza się, a jedynie komunikuje wystąpienie błędu i w zależności od jego wagi kontynuuje działanie lub informuje użytkownika o zakończeniu działania.

Program został również sprawdzony pod kątem wycieków pamięci przy użyciu narzędzia Visual Leak Detector. Żadne wycieki pamięci nie występują.

Podczas pracy nad programem korzystano z systemu kontroli wersji GitHub, a wszystkie zmiany dokumentowane były na koncie <http://github.com/alicja-lachman>.

1. **Podsumowanie**

W programie zawarto następujące cechy programowania orientowanego obiektowo:

* **Dziedziczenie**

Klasy Wczasy oraz WycieczkaObjazdowa dziedziczą po klasie Wycieczka,

* **Polimorfizm**

Klasa Wycieczka jest klasą abstrakcyjną z dwiema czysto wirtualnymi metodami: void sprawdz(BiuroPodrozy \*biuro) oraz void drukuj(BiuroPodrozy \*biuro, int mnoznik).

* **Hermetyzacja**

Wszystkie pola klas Wycieczka, Wczasy, WycieczkaObjazdowa i WycieczkaLaczona są prywatne. Obiekty z innych klas niż dana nie mają możliwości modyfikowania tych pól, a jedynie dostęp do odczytu ich wartości poprzez funkcje typu getWartosc().

Dodatkowo uwzględniono obsługę wyjątków.