Gliwice, 25 maja 2019

Programowanie Komputerów

Temat: Pasażerowie autobusu

Autor: Zbigniew Malcherczyk
Semestr: drugi
Grupa: 1
Prowadzący: dr. inż Tomasz Moroń

Analiza rozwiązania

W pierwszej kolejności przystąpiłem do analizy założeń domenowych, aby zdefiniować podstawową funkcjonalność programu. Głównym celem domenowym jest przetworzenie danych, aby agregować zestaw dla poszczególnych autobusów. Następnie przystąpiłem do analizy podanych przykładowych danych wejściowych.

Analiza przykładowych danych wejściowych

kod trasy	**punkt startowy**	**data startu rrrr-mm**	**nazwisko pasażera**	**nr miejsca**
15551	*Katowice*	*2011-12-13*	*Jaworek*	*33*
15551	*Katowice*	*2011-12-13*	*Kowalski*	*2*
15551	*Katowice*	*2011-12-13*	*Szybki*	*12*
15551	*Katowice*	*2011-12-13*	*Biały*	*43*
15651	Warszawa	2012-02-03	Hastings	2
15651	Warszawa	2012-02-03	Poirot	23
15651	Warszawa	2012-02-03	Holmes	11

Z zestawu danych możemy wywnioskować, że każdy autobus posiada pola: **kod trasy**, **punkt startowy** oraz **data startu**. Jednakże domenowy autobus musi posiadać pasażerów, więc aby spełnić wymagania programu utworzymy kolejne pole z listą jednokierunkową pasażerów.

Pozostają nam natomiast pola głównie z pasażerem, które wyodrębnimy jako osobny byt. Na tym etapie zostały utworzone dwie struktury: *pasażera* oraz *autobusu*.

Szczególnie należy zwrócić uwagę pomimo, że posiadamy tutaj pola o wartościach liczbowych, to w prawidłowym zestawie danych nigdy nie przyjmują one wartości ujemnej.

Analiza przykładowych danych wyjściowych

Przejdźmy do analizy przykładowego pliku wyjściowego.

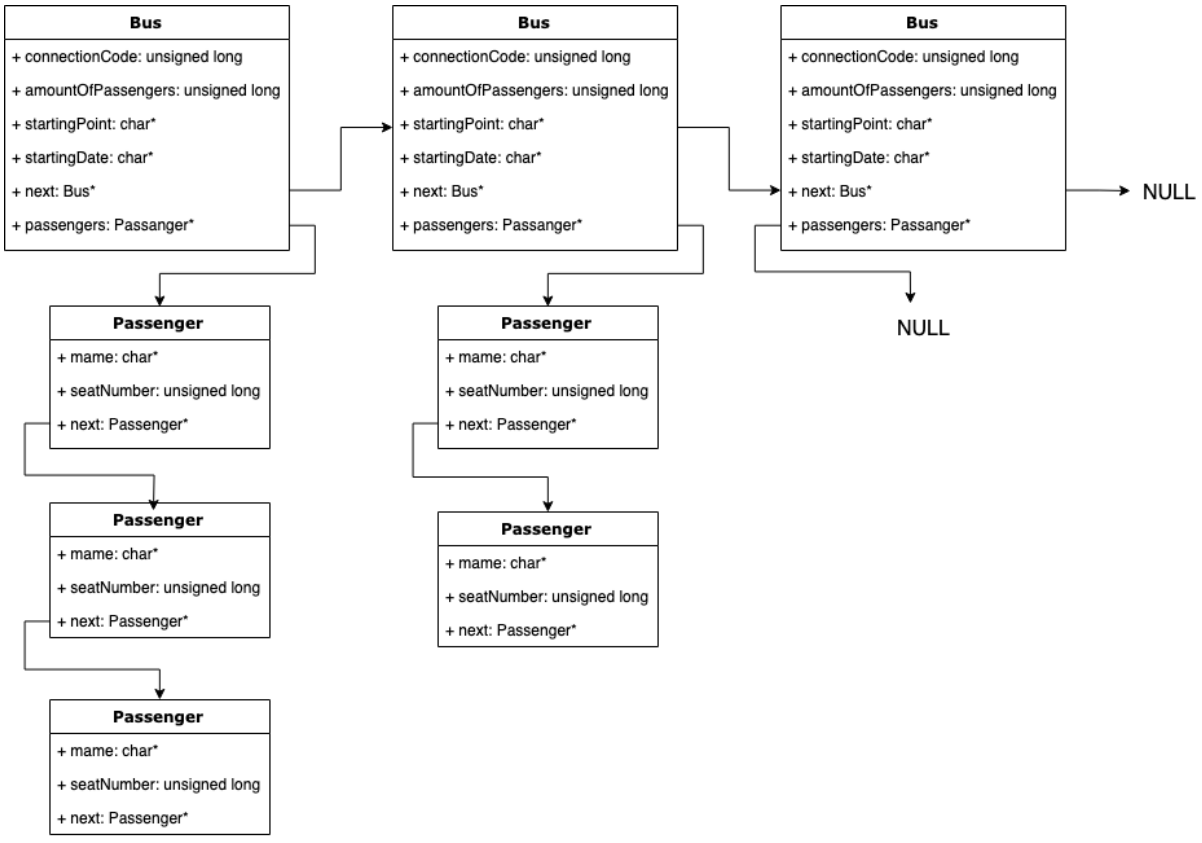
Symbol trasy: 15551
Punkt startowy: Katowice
Data: 2011-12-13
02 Kowalski
10 Nowak
12 Szybki
39 Biały
33 Jaworek
43 Biały
liczba rezerwacji: 6

W pliku wyjściowym posiadamy trzy wcześniej już zdefiniowane pola struktury autobusu, listę pasażerów. Ukazana w przykładzie suma pasażerów autobusu sygnalizuje, że należy zliczać ilość pasażerów, co czyni tą linię nowym założeniem domenowym.

Relacja struktur

Struktury pasażera oraz autobusu muszą być bezpośrednio powiązane ze sobą, tak aby byt logiczny odzwierciedlał realny przypadek. Dlatego zdefiniujemy, że pasażer jest elementem autobusu, a zbiór wszystkich pasażerów listą. W ten sposób dodane do struktury autobusu pole pasażerów będzie listą jednokierunkową wszystkich pasażerów, którzy zarezerwowali bilet na ten autobus.

Treść zadania mówi jasno, że wynikiem są utworzone pliki o strukturze pokazanej wyżej, których nazwa to **kod trasy**, aby wykonać tego typu funkcjonalność musimy znać pełną listę autobusów. Jednym z sposobów na to jest tablicy dynamiczna, ale w związku z tym, że nasza struktura autobusu posiada już dynamiczną jednokierunkową listę pasażerów, zdecydowałem się na użycie kolejnej listy jednokierunkowej, która połączy byty autobusów. W ten sposób została utworzona architektura listy podwieszanej, ukazana na schemacie poniżej.



Algorytm sortowania przez wstawianie

W opisie zadania znajduje się informacja, że lista pasażerów autobusu ma zostać wypisana do pliku rosnąco w zależności od numeru siedzenia. Tą funkcjonalność można uzyskać poprzez dodanie pasażera do listy w sposób posortowany lub użyć algorytmu sortującego listę jednokierunkową. W związku z tym, że dodawanie elementu do listy w sposób sortujący jest operacją, której implementacja budzi mieszane uczucia wśród osób czytających kod - zdecydowałem się na użycie algorytmu sortowania przez wstawianie, tuż przed wypisaniem danych do pliku wyjściowego. Algorytm ten posiada złożoność czasową $O(n^2)$ oraz pamięciową O(1). Nie jest to algorytm cechujący się najwyższą szybkością (w porównaniu do algorytmu quicksort), jednakże jest stabilny i wydajny dla zbiorów o niewielkiej liczbie elementów.

Specyfikacja zewnętrzna

Jeżeli w katalogu programu istnieje plik z rozszerzeniem .exe program należy uruchomić z linii poleceń. W przeciwnym przypadku program należy skompilować. Aby podać plik wejściowy należy uruchomić program z flagą **-i** oraz podać **pełną** ścieżkę do pliku:

<nazwa-programu.exe> -i nazwa-pliku.txt

Uwaga! Program obsługuje wyłącznie pliki wejściowe o rozszerzeniu *.txt

Gdy plik (wraz z ścieżką) nie zostanie odnaleziony, ukaże się następujący komunikat:

Nie znaleziono pliku z danymi!
Pamietaj, ze potrzebna jest absolutna (pelna) sciezka pliku

Gdy program wykona się poprawnie wystąpi komunikat:

Success: Dane rejestru zostały przetworzone poprawnie

Wynikiem tego w folderze */test/output* zostaną utworzone pliki wynikowe dla przekazanych danych wejściowych.

Specyfikacja wewnętrzna

Opis funkcji realizujących cele programu:

int isBusDuplicated(struct Bus list, unsigned long connectionCode, char* city, char* date);*
Funkcja zwraca **1**, jeżeli w podanej liście autobusów występuje taki o podanym kodzie trasy, mieście i dacie. W przeciwnym razie zwraca **0**.

int isBusForConnectionCode(struct Bus list, unsigned long connectionCode);*
Funkcja zwraca **1**, jeżeli w podanej liście autobusów występuje taki o podanym kodzie trasy. W przeciwnym razie zwraca **0**.

void addPassengerToBus(struct Bus list, unsigned long connectionCode, char* name, unsigned long seatNumber);*
Funkcja tworzy pasażera dla podanej nazwy oraz numeru siedzenia, a następnie dodaje go na początek listy pasażerów dla obiektu autobusu z zadanym kodem trasy.

void printBus(struct Bus busList, unsigned long connectionCode);*
Funkcja **testowa**, która wyświetla w terminalu nieuporządkowany obiekt autobusu wraz z listą pasażerów.

void freeData(struct Bus bus);*
Funkcja zwalniająca pamięć listy dynamicznej autobusów wraz z pasażerami na koniec programu, aby nie wystąpił żaden wyciek danych.

void generateResult(struct Bus bus)*
Funkcja zapisująca do plików o nazwie kodu trasy strukturą danych zgodnie z celem programu.

FILE loadSourceFile(const char* argv[])*
Funkcja odpowiedzialna za ładowanie pliku wejściowego. Jeżeli podany jest argument -i oraz nazwa pliku to funkcja zwróci ten właśnie plik. W przeciwnym przypadku załadowany zostanie statycznie zdefiniowany plik testowy.

struct Bus getBusByConnectionCode(struct Bus* bus, unsigned long connectionCode);*
Funkcja zwraca strukturę autobusu z podanej listy oraz o zadanym kodzie trasy

struct Bus createBus(unsigned long code, char* city, char* date);*
Funkcja zwraca utworzoną strukturę autobusu dla zadanych danych.

struct Passenger sortList(struct Passenger* passenger);*
Funkcja korzystająca z algorytmu sortowania przez wstawianie w celu posortowania listy pasażerów autobusu w kolejności rosnącej.

Testowanie

W katalogu głównym programu w katalogu test znajdują się katalogi: **input**, który zawiera pliki z danymi testowymi oraz **examples**, który to zawiera podkatalogi o nazwie pliku wejściowego, które zawierają pliki wynikowe programu dla tego właśnie pliku wejściowego. Dla przykładu, uruchamiając program poleceniem:
program.exe -i in_example_not_sorted.txt

Zostają utworzone dwa pliki wynikowe: 155.txt oraz 156.txt

Plik 155.txt
Symbol trasy: 155
Punkt startowy: Gdynia
Data: 2011-12-13
1 A_name
2 B_name
3 C_name
4 D_name
5 E_name
6 F_name
7 G_name
8 H_name
9 I_name
10 J_name
Liczba rezerwacji: 10

Plik 156.txt
Symbol trasy: 156
Punkt startowy: Katowice
Data: 2018-12-13
1 A_name
2 B_name
3 C_name
4 D_name
5 E_name
6 F_name
7 G_name
8 H_name
9 I_name
10 J_name
Liczba rezerwacji: 10