Politechnika Śląska w Gliwicach
Wydział Informatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

Zarządzanie flotą pojazdów i kierowców

autor	Krzysztof Dybowski
prowadzący	dr inż. Bożena Wieczorek
rok akademicki	2020/2021
kierunek	Teleinformatyka
rodzaj studiów	SSI
semestr	1
termin laboratorium	wtorek, 08:30-10:00
grupa	1
sekcja	1
termin oddania sprawozdania	2020-12-13
data oddania sprawozdania	2015-12-20

1 Treść zadania

Napisać program umożliwiający zapisywanie, organizowanie oraz zarządzanie flotą pojazdów i kierowców.

Program ma umożliwiać:

- dodanie nowego pojazdu,
- usunięcie istniejącego pojazdu,
- dodanie pojazdu,
- usunięcie pojazdu,
- wyświetlenie listy pojazdów (posortowanych wg wszystkich możliwych kryteriów),
- filtrowanie listy pojazdów (wg wszystkich możliwych kryteriów),
- wyświetlenie listy kierowców (posortowanych wg wszystkich możliwych kryteriów),
- filtrowanie listy kierowców (wg wszystkich możliwych kryteriów),
- wygenerowanie raportu zawierającego (także uwzględniające filtrowanie) kierowców i samochodów którymi mogą podróżować.

Zakończenie pracy z programem ma powodować zapisanie na dysku aktualnych danych.
Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:
-k plik wejściowy z kierowcami -p plik wejściowy z pojazdami

2 Analiza zadania

Program jest swoistą bazą danych, wraz z całą jej funkcjonalnością, dodawaniem nowych rekordów, usuwaniem rekordów. Sortowaniem według wybranych pól, wyświetlaniem pól o wybranych parametrach i generowaniem raportu.

2.1 Struktury danych

W programie wykorzystuje listy jednokierunkowe. Wykorzystywane są one do przechowywania danych wczytanych z pliku. Wypisywanie posortowanej listy realizowanie jest poprzez przypisanie elementów z list pierwotnej(wczytanej z pliku), do nowej listy.

2.2 Algorytmy

Generacja raportu oparta jest na pomijaniu nie pasujących pojazdów i wypisywaniu pojazdów zgodnych z kierowcą po polu “licens”. Tworzenie listy z pliku ma średni czas *O(n)*, dodawanie do listy odbywa się w średnim czasie *O(1)*, jest to zasługą zastosowania ogona, czyli wskaźnika na ostatni element listy. Sortowanie jest sortowaniem rosnąco i ma złożoność obliczeniową *O(n)*, w najlepszym przypadku(posortowana lista), *O(n*n!),* dla przypadku pesymistycznego, algorytm jest pewną wariacją sortowania bąbelkowego.

3 Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Należy przekazać do programu nazwy plików, z listą kierowców oraz z listą pojazdów po odpowiednich przełącznikach.
(-k plik wejściowy z kierowcami - p plik wejściowy z pojazdami)
Przełączniki mogą być podane w dowolnej kolejności. Pliki są plikami tekstowymi, jednak użytkownik nie powinien wpisywać rozszerzania .txt w przełączniku. Uruchomienie programu z błędną ilością argumentów skutkuje wpisaniem poradnik jak poprawnie używać programu.Wpisanie błędnych przełączników skutkuje nie wykonaniem programu.

4 Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem programowania strukturalnego, z użyciem wskaźników i referencji.

4.1 Typy zdefiniowane w programie

W programie zdefiniowano następująco typy:

```
1 struct vehicle
2 {
3       std::string type = ""; //Typ pojazdu np. osobowy, rower
4       std::string brand = ""; //Marka pojazdu np. Audi, BMW
5       std::string model = ""; //Model pojazdu np. C6, 150
6       std::string licens = ""; // Kategoria prawo jazdy potrzebna by móc prowadzić pojazd
7       vehicle* next = nullptr; // Następny element w liście
8 };
```

Typ ten służy do zbudowania z niej listy jednokierunkowej.
Analogiczna struktura wykorzystywana jest do tworzenia listy kierowców.

```
1 struct driver
2 {
3       std::string name = ""; // Imię kierowcy
4       std::string surrname = "";// Nazwisko kierowcy
5       std::string licens = "";// Kategoria prawa jazdy kierowcy
6       driver* next = nullptr; // Następny element w liście
7 };
```

4.2 Ogólna struktura programu

Funkcja main posiada następującą budowę(nie jest kod źródłowy tylko zarys wnętrza funkcji main):

```
3       int main(int argc, char** argv) //  argc powinno być równe 5
4 {     //zmienne przechowujące nazwy plików(razem z rozszerzeniami) list kierowców i pojazdów
5       std::string v_file_name = "";
6       std::string d_file_name = "";
7       //blok zawierający sprawdzenie, czy przekazano właściwą ilość argumentów
...
38      //blok zawierający zmiennych wewnętrznych funkcji main.
...
48      //blok wczytanie i sprawdzenie poprawności wczytania danych z pliku do struktury wewnątrz
         programu
...
64      //główna pętla programu
65      while (choice != 'e')
66      {
67      //blok wypisywania menu na standardowe wyjście i pobierania znaku z standardowego wejścia
...
80      std::getline(std::cin,trash); // czyszczenie bufora wejściowego
81      //przełącznik wyboru, co program ma zrobić
...
143     }
144}
```

4.3 Szczegółowy opis funkcji obsługi list

Funkcje obsługi list dla obu struktur, są praktycznie takie same, różnią się jedynie ilością wypełnianych pól struktury oraz przyjmowanymi i zwracanymi wartościami. Działanie tych funkcji opisze na podstawie funkcji dla struktury “vehicle”.

Vehicle* creat_vehicle_list(vehicle**beg);

Funkcja tworzy pod wskaźnikiem na który wskazuje **beg, pojedynczy element o typie vehicle i zwraca wskaźnik na niego.

void write_to_vehicle(vehicle** destiny, const vehicle*source);

Funkcja przepisuje wszystkie pola z struktury source do struktury destiny i ustawia pole next na nullptr.

vehicle* add_new_vehicle(vehicle** beg, vehicle** end, vehicle* tmp);

Funkcja dodaje na koniec listy element typu vehicle albo gdy wskaźnik pod wskaźnikiem beg jest pusty tworzy nową listę, w obu przypadkach wykorzystywane są funkcje opisane wyżej. Pola nowo powstałego elementu wypełniane są na podstawie pól elementu na który wskazuje wskaźnik tmp. Pod koniec funkcja zwraca wskaźnik na nowo powstały element. Ciało funkcji:

```
17      vehicle* add_new_vehicle(vehicle** beg, vehicle** end, vehicle* tmp)
18{
19      vehicle* curr = *beg; //wskaźnik tymczasowy służący do modyfikowania listy
20      if (curr == nullptr) //sprawdza czy lista istnieje
21      {
22              *beg = crate_vehicle_list(beg);
23              write_to_vehicle(beg, tmp);
24              *end = *beg;
25              return *beg;
26      }
27      else    //jeśli lista istnieje, na jej koniec dodawany jest element
28      {
29              curr = *end;
30              curr->next = crate_vehicle_list(&curr->next);
31              *end = curr->next;
32              write_to_vehicle(&curr->next, tmp);
33              return *end;
34      }
35}
```

void delete_vehicle(vehicle** beg, vehicle** end)

Funkcja usuwająca jeden element z listy. Podczas jej działania użytkownik wypełnia pola pojedyńczego elementu struktury zwracanego przez funkcję filter_vehicles(** g_filter);, program usunie element o polach odpowiadających temu elementowi. Jeśli program nie znajdzie takiego elementu, powiadomi o tym użytkownika i wróci do głównej pętli programu.

4.4 Szczegółowy opis funkcji wczytania z pliku i filtrowania

---

bool load_vegicle_list(std::string name, vehicle** beg, vehicle** end)

Funkcja odpowiadające za zapisanie do i stworzenie struktury. Jeśli plik podany jako parametr “name” nie udało się otworzyć funkcja zwróci wartość fałsz co wykorzystuje blok w funkcji main odpowiedzialny za poinformowanie użytkownika o tym, że coś jest nie tak z plikiem który próbuje przekazać do programu.
Przy tworzeniu elementów listy funkcja wykorzystuje funkcjie create_vehicle_list i add_new_vehicle.

---

vehicle* filter_vehicles(vehicle** g_filter)

Funkcja zawierająca w sobie zarówno menu w którym wyświetla aktualne pola głównego filtra g_filter jak i możliwość wypełniania pól tego filtru wartościami podanymi przez użytkownika. Po podaniu przez użytkownika znaku ‘e’ pętla menu funkcji jest przerywana i funkcja zwraca wskaźnik na g_filter.

---

bool vehicle_display_flag(vehicle* curr, vehicle* filter)

Zbiór instrukcji warunkowych, mających na celu sprawdzić czy chociaż jedno pole elementu curr jest zgodne z polem elementu filter. Jest używana do wystawienia flagi czy dany element ma być wyświetlony czy nie w funkcji show_filtered_vehicle_list. funkcja bierze pod uwagę czy porównywane pole filtru jest puste.

---

void show_filtred_vehicle_list(vehicle* beg, vehicle* filter)

Funkcja dostępna z głównego menu, zawierają w sobie podmenu wyboru czy po prostu wyświetlić elementy listy, czy wyświetlić posortowane elementy listy. Filtr główny jest przekazywany do tej funkcji i wypływa na wyświetlanie przez funkcje elementów listy.

4.5 Szczegółowy opis funkcji sortowania listy

Funkcje te opierają się na dwóch funkcjach is_grater(std::string s1, std::string s2) i is_less(std::string s1, std::string s2). Które zwracają czy ciąg znakowy s2 był większy od ciągu s1 w przyapdku funkcji is_grater, czy był mniejszy w przypadku funkcji is_less.

---

int sort_vehicles_by()

Jest to funkcja pełniąca menu w której użytkownik wybiera po którym polu ma odbyć się sortowanie listy. funkcja zwraca wartość int, przy czym zwraca 0 gdy sortowanie ma zostać anulowane, anulowanie sortowania. Jestu używana sort_vehicle.

zbiór funkcji find_place_by(vehicle ** sorted_list, vehicle* base_next)

Wszystkie funkcje w tym zbiorze działają na tej samej zasadzie, różnią się jedynie polem po którym mają sortować oraz tym czy mają posortować malejąco czy rosnąco. Każda funkcja z tego zbioru sprawdza, czy element pod wskaźnikiem base_next jest większy/mniejszy od każdego elementu z listy posortowanej sorted_list, jeśli element jest większy/ mniejszy jest wstawiane przed/za element z którym jest porównywany. funkcje te nic nei zwracają za to modyfikują listę sorted_list.

vehicle*sort_vehicles(vehicle* base)

Funkcja bierze pierwszy element z listy pod wskaźnikiem base i tworzy nowa liste sorted_list, a potem korzystając z funkcji menu sort_veficle_by i odpowiedniych funkcji ze zbioru find_place_by wypełnia ją, iterując się przez całą listę base i znajdując w liście sorted_list miejsce dla nich. Zawiera też w sobie menu wyboru w którym użytkownik wybiera czy posortować malejąco czy rosnąco elementy listy base. Funkcja zwraca wskaźnik na posortowaną listę, który jest wykorzystywane w funkcji show_filtred_vehicle_list. Po jej wypisaniu lista posortowana jest usuwana funkcją delete_sorted_v.

4.6 Szczegółowy opis funkcji specjalnych

Funkcje specjalne to funkcje potrzebne do generacji raportu, funkcja generująca raport i funkcje końcowe programu, czyli funkcja zapisu list do plików z danymi z których zostały one utworzone oraz funkcja czyszcząca i usuwająca obie listy.

bool can_drive(driver* c_driver, vehicle* c_vehicle)

Funkcja sprawdza, czy element reprezentujący kierowcę przekazany do funkcji ma pole “licens” zgodne z polem elementu reprezentującego pojazd. Jeśli tak zwraca true w przeciwnym razie fasle.

void generate_report(vehicle* v_beg, driver* d_beg, vehicle* v_filter, driver* d_filter)

Generuje raport , to znaczy wypisuje z listy kierowców oraz samochody którymi mogą jeździć. wykorzystuje do tego funkcje can_driver.

void save_database(vehicle* v_beg, driver* d_beg, std::string d_name, std::string v_name)

Funkcja zapisuje zarówno listę kierowców jak i pojazdów do odpowiednich im plików. Pliki te to te same pliki wykorzystane do stworzenia i wypełnienia listy kierowców i pojazdów tworzonej przez program. Dane w tych plikach są nadpisywane danymi z list programu. Wywoływana jest przy wychodzeniu z programu.

void clear(vehicle** v_beg, driver** d_beg, vehicle** v_end, driver** d_end)

Funkcja usuwa wszystkie elementy z obu list kierowców i pojazdów. Wywoływana jest przy wychodzeniu z programu.