# Kod źródłowy Projektu 1 Radosław Podgórski

środa, 24 kwietnia 2024 11:16

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
struct Klient {
    char imie[30];
    char nazwisko[30];
    char PESEL[12];
};
struct Rower {
    int identyfikator;
    char model[30];
    char kolor[20];
    int cena;
};
void dodajKlienta(const Klient &klient) {
    ofstream plik("klienci.bin", ios::out |
ios::binary | ios::app);
    if (plik.is_open()) {
        plik.write((const char*) &klient,
sizeof(Klient));
        plik.close();
    }
    else {
        cout << "Blad otwarcia pliku klienci.bin"
<< endl;
    }
}
void dodajSprzet(const Rower &rower) {
    ofstream plik("rowery.bin", ios::out | ios::binary
| ios::app);
    if(plik.is_open()) {
        plik.write((const char*) &rower,
sizeof(Rower));
        plik.close();
    }
    else {
        cout << "Blad otwarcia pliku rowery.bin"
```

```
<< endl;
    }
}
void usunKlienta(const char* pesel) {
    ifstream plik("klienci.bin", ios::in |
ios::binary);
    ofstream tempPlik("temp.bin", ios::out |
ios::binary);
    if(plik.is_open() && tempPlik.is_open()) {
        Klient klient;
        while(plik.read((char*) &klient,
sizeof(Klient))) {
            if(strcmp(klient.PESEL, pesel) != 0) {
                tempPlik.write((const char*) &klient,
sizeof(Klient));
            }
        }
        plik.close();
        tempPlik.close();
        remove("klienci.bin");
        rename("temp.bin", "klienci.bin");

        cout << "Klient o PESEL: " << pesel << "
zostal usuniety." << endl;
    }
    else {
        cout << "Blad otwierania plikow" << endl;
    }
}
void usunSprzet(int identyfikator) {
    ifstream plik("rowery.bin", ios::in |
ios::binary);
    ofstream tempPlik("temp.bin", ios::out |
ios::binary);
    if(plik.is_open() && tempPlik.is_open()) {
        Rower rower;
        while(plik.read((char*) &rower,
sizeof(Rower))) {
            if(rower.identyfikator != identyfikator) {
                tempPlik.write((const char*) &rower,
sizeof(Rower));
            }
        }
```

```
        plik.close();
        tempPlik.close();
        remove("rowery.bin");
        rename("temp.bin", "rowery.bin");
        cout << "Rower o identyfikatorze "
<< identyfikator << " zostal usuniety." << endl
<< endl;
    }
    else {
        cout << "blad otiwerania plikow" << endl;
    }
}
void modyfikacjaAtrybutowRoweru(int identyfikator,
const char* nowyKolor, int nowaCena) { //nie da sie
zmienic modelu, bo nie mialoby to sensu
    ifstream plik("rowery.bin", ios::in |
ios::binary);
    ofstream tempPlik("temp.bin", ios::out |
ios::binary);
    if(plik.is_open() && tempPlik.is_open()) {
        Rower rower;
        while(plik.read((char*) &rower,
sizeof(Rower))) {
            if(rower.identyfikator == identyfikator) {
                strcpy(rower.kolor, nowyKolor);
                rower.cena = nowaCena;
            }
            tempPlik.write((const char*) &rower,
sizeof(Rower));
        }
        plik.close();
        tempPlik.close();
        remove("rowery.bin");
        rename("temp.bin", "rowery.bin");
        cout << "Atrybuty roweru o identyfikatorze "
<< identyfikator << " zostaly zaktualizowane" << endl;
    }
}
// Funkcja scalająca dwie posortowane części tablicy
void scalanie(Rower tablica[], int lewy, int srodek,
int prawy, int wybor) {
    int i, j, k;
    int n1 = srodek - lewy + 1; // Rozmiar lewej
```

```
części tablicy
    int n2 = prawy - srodek; // Rozmiar prawej części
tablicy
    // Tworzenie pomocniczych tablic
    Rower lewa[n1], prawa[n2];
    // Kopiowanie danych do tablic pomocniczych
    for (i = 0; i < n1; i++)
        lewa[i] = tablica[lewy + i];
    for (j = 0; j < n2; j++)
        prawa[j] = tablica[srodek + 1 + j];
    // Scalanie tablic pomocniczych z powrotem do
tablicy głównej
    i = 0;
    j = 0;
    k = lewy;
    while (i < n1 && j < n2) {
        // Wybór atrybutu sortowania i porównanie
odpowiednich pól
        if (wybor == 1 || wybor == 2) {
            if (wybor == 1) {
                if (strcmp(lewa[i].kolor,
prawa[j].kolor) <= 0) {
                    tablica[k] = lewa[i];
                    i++;
                } else {
                    tablica[k] = prawa[j];
                    j++;
                }
            } else {
                if (strcmp(lewa[i].model,
prawa[j].model) <= 0) {
                    tablica[k] = lewa[i];
                    i++;
                } else {
                    tablica[k] = prawa[j];
                    j++;
                }
            }
        } else if (wybor == 3) {
            if (lewa[i].cena <= prawa[j].cena) {
                tablica[k] = lewa[i];
                i++;
            } else {
```

```
                tablica[k] = prawa[j];
                j++;
            }
        }
        k++;
    }
    // kopiuje pozostałe elementy z lewej tablicy
    while (i < n1) {
        tablica[k] = lewa[i];
        i++;
        k++;
    }
    // kopiuje pozostale z prawej
    while (j < n2) {
        tablica[k] = prawa[j];
        j++;
        k++;
    }
}
// Funkcja sortująca przez scalanie
void sortowaniePrzezScalanie(Rower tablica[], int
lewy, int prawy, int wybor) {
    if (lewy < prawy) {
        int srodek = lewy + (prawy - lewy) / 2; //
Środek tablicy
        // Sortowanie lewej i prawej części tablicy
        sortowaniePrzezScalanie(tablica, lewy, srodek,
wybor);
        sortowaniePrzezScalanie(tablica, srodek + 1,
prawy, wybor);
        // Scalanie posortowanych części
        scalanie(tablica, lewy, srodek, prawy, wybor);
    }
}
// Funkcja wyświetlająca posortowaną listę rowerów
void wyswietlPosortowanaListe(Rower rowery[], int n,
int wybor) {
    sortowaniePrzezScalanie(rowery, 0, n - 1, wybor);
// Sortowanie tablicy rowerów
    // Wyświetlanie posortowanej listy
    cout << "Posortowana lista rowerów:" << endl;
    for (int i = 0; i < n; i++) {
        cout << "Identyfikator: "
```

```
<< rowery[i].identyfikator << ", Model: "
<< rowery[i].model
             << ", Kolor: " << rowery[i].kolor << ",
Cena: " << rowery[i].cena << endl;
    }
}
int main() {
    int wybor;
    const int MAX_ROWERY = 100; // Maksymalna ilość
rowerów
    Rower rowery[MAX_ROWERY]; // Tablica przechowująca
rowery
    int n = 0; // Licznik aktualnej liczby rowerów w
tablicy
    while(true) {
        // Wyświetlanie opcji i pobieranie wyboru
użytkownika
        cout << "Wybierz funkcje, ktorej chcesz uzyc
(1-8) lub 0 aby zakonczyc program: " << endl;
        cout << "0. Zakoncz dzialanie programu"
<< endl;
        cout << "1. Dodaj nowego klienta" << endl;
        cout << "2. Dodaj nowy sprzet" << endl;
        cout << "3. Usun klienta" << endl;
        cout << "4. Usun obiekt, ktory jest
przedmiotem dzialanosci wypozyczalni" << endl;
        cout << "5. Modyfikuj atrybuty obiektu, ktory
jest przedmiotem dzialalnosci wypozyczalni" << endl;
        cout << "6. Wyswietl posortowana liste po
wybranych atrybutach" << endl;
        cout << "Twoj wybor: ";
        cin >> wybor;
        // Wybór operacji w zależności od wyboru
użytkownika
        switch(wybor) {
            case 0: // Zakończenie działania programu
                cout << "Koniec programu" << endl;
                return 0;
            case 1: // Wywołanie funkcji dodającej
nowego klienta
                Klient nowyKlient;
                cout << "Podaj imie, nazwisko i PESEL
klienta: " << endl;
```

```
                cin >> nowyKlient.imie >>
nowyKlient.nazwisko >> nowyKlient.PESEL;
                dodajKlienta(nowyKlient);
                break;
            case 2: // Wywołanie funkcji dodającej
nowy sprzęt
                if (n < MAX_ROWERY) {
                    Rower nowyRower;
                    cout << "Podaj identyfikator,
model, kolor i cene roweru oddzielone spacja: "
<< endl;
                    cin >> nowyRower.identyfikator >>
nowyRower.model >> nowyRower.kolor >> nowyRower.cena;
                    dodajSprzet(nowyRower);
                    rowery[n++] = nowyRower; // Dodaje
nowy rower do tablicy i inkrementuje licznik n
                } else {
                    cout << "Maksymalna ilość rowerów
osiągnięta." << endl;
                }
                break;
            case 3: // Wywołanie funkcji usuwającej
klienta
                char pesel[12];
                cout << "Podaj pesel klienta do
usuniecia: " << endl;
                cin >> pesel;
                usunKlienta(pesel);
                break;
            case 4: // Wywołanie funkcji usuwającej
obiekt z działalności wypożyczalni
                int identyfikator;
                cout << "Podaj identyfikator roweru do
usuniecia: " << endl;
                cin >> identyfikator;
                usunSprzet(identyfikator);
                break;
            case 5: // Wywołanie funkcji modyfikującej
atrybuty obiektu
                int id;
                char nowyKolor[30];
                int nowaCena;
                cout << "Podaj identyfikator roweru
```

```
ktorego atrybuty chcesz zmodyfikowac: ";
                cin >> id;
                cout << "Podaj nowy kolor: ";
                cin >> nowyKolor;
                cout << "Podaj nowa cene: ";
                cin >> nowaCena;
                modyfikacjaAtrybutowRoweru(id,
nowyKolor, nowaCena);
                break;
            case 6: // Wywołanie funkcji
wyswietlajacej posortowana liste na podstawie
wybranego atrybutu
                cout << "Wybierz atrybut do sortowania
(1 - kolor, 2 - model, 3 - cena): ";
                cin >> wybor;
                wyswietlPosortowanaListe(rowery, n,
wybor);
                break;
            default: //default
                cout << "Niepoprawny wybor. Sprobuj
ponownie." << endl;
        }
    }
    return 0;
}
```