Politechnika Śląska w Gliwicach
Wydział Automatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

Loty

autor	Mateusz Piwowarski
prowadzący	dr inż. Krzysztof Simiński
rok akademicki	2017/2018
kierunek	informatyka
rodzaj studiów	SSI
semestr	1
termin laboratorium / ćwiczeń	wtorek, 13:45 – 15:15
grupa	3
sekcja	7
termin oddania sprawozdania	2018-01-25
data oddania sprawozdania	2018-01-25

1 Treść zadania

Korzystając z dynamicznych struktur danych napisać program, który przygotowuje listy pasażerów. Pasażerowie mogą rezerwować bilety na różne loty w różnych biurach i przez internet. Wszystkie rezerwacje są zapisywane w biurze centralnym. Mają one następującą postać:

(symbol lotu) (lotnisko startowe) (data lotu) (nazwisko pasażera) (nr miejsca)

Przykładowy plik z rezerwacjami:

```
KR54R Katowice 2011-12-13 Jaworek 33
TY340 London 2012-02-03 Hastings 2
TY340 London 2012-02-03 Poirot 23
KR54R Katowice 2011-12-13 Matianek 02
TY340 London 2012-02-03 Holmes 11
KR54R Katowice 2011-12-13 Lopez 12
TY340 London 2012-02-03 Lemon 43
KR54R Katowice 2011-12-13 Chavez 43
```

Na podstawie pliku z rezerwacjami należy stworzyć plik z listą pasażerów dla każdego lotu. Każda lista jest tworzona w odrębnym pliku. Nazwą pliku jest symbol lotu. W pliku umieszczona jest nazwa lotniska i data. W kolejnych liniach umieszczone są numery siedzeń i nazwiska pasażerów, posortowane wg numerów.Dla powyższego pliku zostaną utworzone pliki: `KR54R.txt` i `TY340.txt`.
Plik `KR54R.txt`:

```
symbol lotu: KR54R
lotnisko:    Katowice
data lotu:   2011-12-13

lista pasazerow:
02 Matianek
11 Lopez
33 Jaworek
43 Chavez

   liczba rezerwacji: 4.
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następującego przełącznika: `-i plik wejsciowy z rezerwacjami`.

2 Analiza zadania

Zagadnienie przedstawia problem sortowania lotów z pliku wejściowego oraz przydzielanie do odpowiednich lotów pasażerów w sposób posortowany według miejsc.

2.1 Struktury danych

W programie wykorzystano dwie listy jednokierunkowe. Lista nadrzędna przechowuje informacje o: identyfikatorze lotu, lotnisku startowym oraz o dacie lotu. Lista nadrzędna posiada wskaźnik na listę podrzędną, która przechowuje nazwiska pasażerów oraz ich miejsca. Lista podrzędna jest posortowana według miejsc pasażerów rosnąco. Takie struktury danych umożliwiają łatwe sortowanie.

2.2 Algorytmy

Program pobiera z pliku wejściowego symbol lotu, lotnisko startowe oraz datę lotu, przeszukując czy lot o podanych parametrach istnieje. Jeżeli istnieje, program dołącza pasażera do listy pasażerów dla konkretnego lotu. Jeżeli lot, który pobrał program nie znajduje się w liście lotów, program dodaje lot na sam koniec w sposób rekurencyjny.

3 Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Przy wywoływaniu programu możliwe jest użycie przełączników `-i` oraz `-h`. Po wykorzystaniu przełącznika `-i` należy przekazać do programu nazwę wejściowego. Wykorzystanie przełącznika `-h` wyświetla instrukcje dla użytkownika obsługi programu. Instrukcja również jest wyświetlania w wyniku podania niepoprawnych danych. Przełączniki mogą być podane w dowolnej kolejności.
Przykładowe wywołania programu:

```
./main -i ../dat/rezerwacje.txt
./main -h
```

Program zapisuje spis pasażerów w pliku tekstowym w folderze zewnętrznym `dat`. Plik tekstowy dla każdego lotu jest nazwany symbolem lotu. Pliki wejściowe mogą mieć dowolne rozszerzenie (lub go nie mieć.).

Nieprawidłowe wykorzsytanie przełączników powoduje wyświetlenie komunikatu:

```
Niepoprawne wywolanie programu.
```

Jeżeli użytkownik nie poda pliku wejściowego otrzyma komunikat:

```
Nie podano nazwy pliku wejscia.
```

Nieprawidłowe dane w pliku wejścia powodują wyświetlenie komunikatu:

```
Niepoprawne dane w pliku wejsciowym.
```

Jeżeli podany plik przez użytkownika nie istnieje, program wyświetli komunikat:

```
Plik o podanej nazwie nie istnieje.
```

4 Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem strukturalnym. W programie rozdzielono interfejs (komunikację z użytkownikiem) od logiki aplikacji (sortowaniem pasażerów oraz lotów).

4.1 Typy zdefiniowane w programie

W programie zdefiniowano następujący typ:

```
1 struct osoba
2 {
3     string nazwisko;
4     int miejsce;
5     osoba * pNext;
6 };
```

Powyższa struktura jest listą jednokierunkową, która zawiera pasażerów (każdy z nich posiada swoje nazwisko oraz nr miejsca). Struktura przechowuje pasażerów posortowanych według nr miejsca, rosnąco.

Zdefiniowano także strukturę:

```
struct loty
{
    string symbollotu;
    string miasto;
    string data;
    osoba * pPasazerowie;
    loty * pNext;
};
```

Struktura loty jest również listą jednokierunkową. Przechowuje loty: każdy z lotów ma symbol lotu, lotnisko startowe, datę lotu oraz wskaźnik na listę pasażerów.

Zdefiniowano także typ:

```
bool poprawnedane = true;
```

Typ ten służy do przechowania informacji czy dane w pliku wejścia są poprawne. Domyślnie posiada wartość True. Gdy program natrafi na niezgodność w pliku wejściowym, zmienia wartość zmiennej poprawnedane na False.

Wykorzystano również typ klasy **string**, który przechowuje nazwę pliku wejścia podaną przez użytkownika:

```
string wejscie = "";
```

4.2 Ogólna struktura programu

W funkcji głównej wywołana jest funkcja

```
bool Odczytajargumenty(int ile, char ** argumenty,
    string & szInput);
```

Funkcja sprawdza przełączniki wykorzystane przez użytkownika w wywołaniu programu. Jeżeli program został poprawnie wywołany, program przechodzi do funkcji:

```
bool Sprawdzplik(const string & wejscie, bool &
    poprawnedane)
```

Plik który podał użytkownik, zostaje sprawdzany przez program. Jeżeli w pliku znajdują się niepoprawne dane lub plik nie istnieje/ nie posiadamy do niego dostępu, program kończy pracę wywołując funkcję:

```
1 void Pomoc();
```

Funkcja wyświetla instrukcję korzystania z programu.

Jeżeli użytkownik podał plik wejścia, w którym znajdują się poprawne dane, program wywołuje funkcję:

```
1 bool pobieranieDanych(const string & wejscie, loty *&
    pHead);
```

Funkcja pobiera listę pasażerów wraz z ich numerami miejsc, datą lotu, lotniskiem startowym i symbolem lotu. Pobierając dane z pliku, program przydziela pasażerów do konkretnych lotów, na które posiadają rezerwacje. Po podziale każdego pasażera do swojego lotu, program wywołuje funkcję:

```
1 void Wynik(loty * pHead);
```

Funkcja dla każdego lotu tworzy osobny plik tekstowy, zapisując w nim listę pasażerów.
Na koniec program usuwa wszystkie wskaźniki oraz elementy list użytych w programie, aby uniknąć wycieków pamięci. W tym celu uruchamia funkcję

```
1 void usunWszystko(loty *& pHead);
```

Po zakończeniu usuwania, program kończy swoją pracę.

4.3 Szczegółowy opis implementacji funkcji

```
1 bool Odczytajargumenty(int ile, char ** argumenty,
    string & szInput);
```

Funkcja sprawdza, czy program został wywołany wraz z użyciem przełączników `-i`, `-h`. Funkcja zwraca informacje czy użytkownik wykorzystał przełączniki w poprawny sposób. W przypadku błędu zwraca wartość False. Jeżeli przełącznik `-i` został prawidłowo wykorzystany , funkcja w zmiennej szInput przechowa nazwę pliku wejścia. Parametr ile przechowuje informację ile argumentów wykorzystaliśmy wywołując funkcję.

Gdy program został wywołany z użyciem parametru `-h` Funkcja wywołuje funkcję:

```
1 void Pomoc();
```

Funkcja ta wyświetla instrukcję korzystania z programu.

Następnie sprawdzana jest wartość zwracana funkcji:

```
1 bool Sprawdzplik(const string & wejscie, bool &
    poprawnedane)
```

Funkcja po otrzymaniu nazwy pliku wejscia, otwiera plik wejscia i sprawdza czy dane w pliku są poprawne. Zwraca wartość False jeżeli w pliku znajdują się niepoprawne dane. Sprawdzana jest każda linia pliku wywołując funkcję:

```
1 void Sprawdzlinie(const string & linia, bool &
    poprawnedane)
```

Funkcja pobiera linię pliku oraz sprawdza jej poprawność wywołując funkcje:

```
1 void Sprawdzsymbol(const string & linia, bool &
    poprawnedane, int & i);
2 void Sprawdzmiasto(const string & linia, bool &
    poprawnedane, int & i);
3 void Sprawdzdate(const string & linia, bool &
    poprawnedane, int & i);
4 void Sprawdznazwisko(const string & linia, bool &
    poprawnedane, int & i);
5 void Sprawdzmiejsce(const string & linia, bool &
    poprawnedane, int & i);
```

Powyższe funkcje sprawdzają poprawność całej linii pliku. Jako parametry przyjmują linię pliku, wartość poprawności dancyh i informację, który znak linii jest obecnie sprawdzany. Jeżeli program natrafi na niepoprawność, zmienna poprawnedane przyjmuje wartość False. Program wyłapuje niepoprawność gdy:

- Symbol lotu zawiera małe litery
- Nazwa miasto odlotu zawiera cyfry
- Data posiada litery (a-z)
- Nazwisko pasażera posiada cyfry
- Nr miejsca posiada litery (a-z)

Jeżeli program nie natrafi na niepoprawność danych, przejdzie do funkcji

```
1 bool pobieranieDanych(const string & wejscie, loty *&
    pHead);
```

Funkcja przyjmuje jako parametry nazwę pliku wejścia, oraz wkaźnik na pierwszy element listy lotów. Zwraca wartość False jeżeli programowi nie udało się dostać do pliku lub plik nie istnieje. Funkcja pobiera po kolei: Symbol lotu, lotnisko startowe, datę lotu, nazwisko pasażera oraz numer miejsca. Przechowując powyższe dane w zmiennych, wywołuje funkcję

```
1 loty * znajdzLot(loty *& pHead, const string & symbol,
    const string & miasto, const string & data);
```

Funkcja przyjmuje wkaźnik na pierwszy lot w liście lotów, symbol lotu, miasto oraz datę, która była pobrana z pliku. Funkcja sprawdza listę lotów, porównując parametry lotów do parametrów danych, które pobraliśmy z pliku. Jeżeli lot pobrany istnieje, funkcja zwraca jego adres. Jeżeli w całej liście lotów nie ma lotu pobranego z pliku, funkcja dodaje pobrany lot na koniec listy oraz zwraca jego adres.
Program przechodzi do funkcji:

```
1 void dodajPasazeraDoLotu(osoba *& pHead, const string &
    nazwisko, int miejsce);
```

Funkcja wykorzystuje wcześniej zwrócony adres lotu. Znając adres lotu, program dodaje pasażera do lotu w sposób posortowany, według miejsc rosnąco.

Po pobraniu i posortowaniu przez program wszystkich pasażerów z pliku wejściowego, program przechodzi do fukcji:

```
1 void Wynik(loty * pHead);
```

Funkcja tworzy pliki nazywając je symbolami lotu. Każdy plik lotu posiada swój symbol lotu, lotnisko, datę lotu, listę pasażerów oraz liczbę rezerwacji. Dla każdego lotu, listę pasażerów zapisuje funkcja:

```
1 void zapiszOsobe(osoba * pHead, ofstream & plik);
```

Istnieje również funkcja, która zwraca liczbę pasażerów i zapisuje wynik w pliku.

```
1 int liczbaPasazerow(osoba * pHead);
```

Aby unkiknąć jakichkolwiek wycieków pamięci, przed zakończeniem program usuwa wszystkie wcześniej wykorzystywane wskaźniki.

```
1 void usunWszystko(loty *& pHead);
```

Aby wszystko wykonało się w prawidłowej kolejności, najpierw zostają usunięte wskaźniki na pasażerów, następnie wskaźniki na loty. Dlatego też powyższa funkcja wywołuje funkcję usuwającą wskaźniki na pasażerów.

```
1 void usunListe(osoba *& pHead);
```

5 Testowanie

Program został przetestowany na różnego rodzaju plikach. Pliki niepoprawne: `zlysymbol.txt`, `zlemiasto.txt`, `zladata.txt`, `zlenazwisko.txt`, `zlemiejsce.txt` powodują zgłoszenie błędu. Każdy z wymienionych wyżej plików posiadał po jednej nieprawidłowości w zapisie: symbolu lotu, lotniska startowego, daty lotu, nazwiska pasażera, numeru miejsca. Program wywołany dla pustego pliku tworzy plik lotu bez danych.

Pliki, w którym znajdują się dodatkowe znaki białe między wartościami nie powodują zgłoszenia błędu przez program.
Poprawność działania programu została sprawdzona na pliku `rezerwacje.txt`. Program został sprawdzony pod kątem wycieków pamięci tak aby nie dopuścić do ani jednego wycieku.

6 Wnioski

Program Loty jest programem prostym, chociaż wymagał umiejętnego posługiwania się listami jednokierunkowymi. Najbardziej wymagające okazało się zapobieganie przed wyciekami pamięci. Realizacja projektu nauczyła obsługiwać się wskaźnikami oraz listami jednokierunkowymi, które mają wskaźniki na inne listy.