Analiza Tematu

Lista dwukierunkowa to struktura, która umożliwia dostęp zarówno od początku jak i końca. Składa się ona z węzłów oraz z dwóch wskaźników (na pierwszy i ostatni element). Każdy węzeł posiada jakieś dane oraz wskaźnik na poprzedni oraz następny element. Zaimplementowana przeze mnie klasa ***List*** korzysta z pomocniczej struktury ***node***, która stanowi węzły. Lista została zaimplementowana przy użyciu szablonów, dlatego działa ona zarówno dla typów standardowych, jak i klas użytkownika. Dodatkowo do testowania stworzonej przeze mnie struktury, zostały napisane dwie klasy testowe: ***Diver*** i ***Flight.***

Wykorzystane zostały biblioteki ***fstream*** aby wykonywać operacje na plikach wejściowych i wyjściowych, ***string*** aby operować na stringach oraz ***memory*** umożliwiająca korzystanie z inteligentnych wskaźników.

Zastosowany został przeze mnie algorytm sortownia szybkiego, który działa na zasadzie „dziel i zwyciężaj”. Jest on bardzo wydajny a jego średnia złożoność obliczeniowa jest rzędu O(nlogn). Zastosowałem wersję, w której pivot ustawiany jest na samym końcu listy. Skutkuje to najgorszą wydajnością dla już posortowanej listy.

Specyfikacja Zewnętrzna

- Program można uruchomić z linii komend. Należy podać 4 argumenty, które są walidowane (-i plik wejściowy dla stringów, -d plik wejściowy wykorzystujący klasę Diver, -f plik wejściowy wykorzystujący klasę Flight oraz -o jako plik wyjściowy). Przy niepodaniu tych argumentów program poinformuje użytkownika o błędzie i zakończy działanie. Kolejność podawanych argumentów jest dowolna, można podać dodatkowe argumenty jednak nie mają one zastosowania. Walidacja odbywa się przy użyciu funkcji ***Inputs***.

- Dodawanie elementu na koniec lub początek struktury. Służą do tego odpowiednio funkcje ***Append*** i ***Prepend***. Jako ich argument należy podać element, który chcemy dodać.

- Usuwanie wybranego elementu jest możliwy przy użyciu funkcji ***Delete***, która jako argument przyjmuje indeks (pierwszy element ma indeks 0).

- Wypisywanie Listy odbywa się przy pomocy bezargumentowej funkcji ***Display***.

- Istnieje możliwość wyszukiwania danego elementy przy pomocy funkcji ***Find***, do której jako argument podajemy klucz, według którego odbywa się porównywanie.

- Przy pomocy funkcji ***Sort*** odbywa się sortowanie listy. Jako pierwszy argument podajemy ***true*** – jeśli chcemy sortować rosnąco, lub ***false*** dla sortowania malejącego.

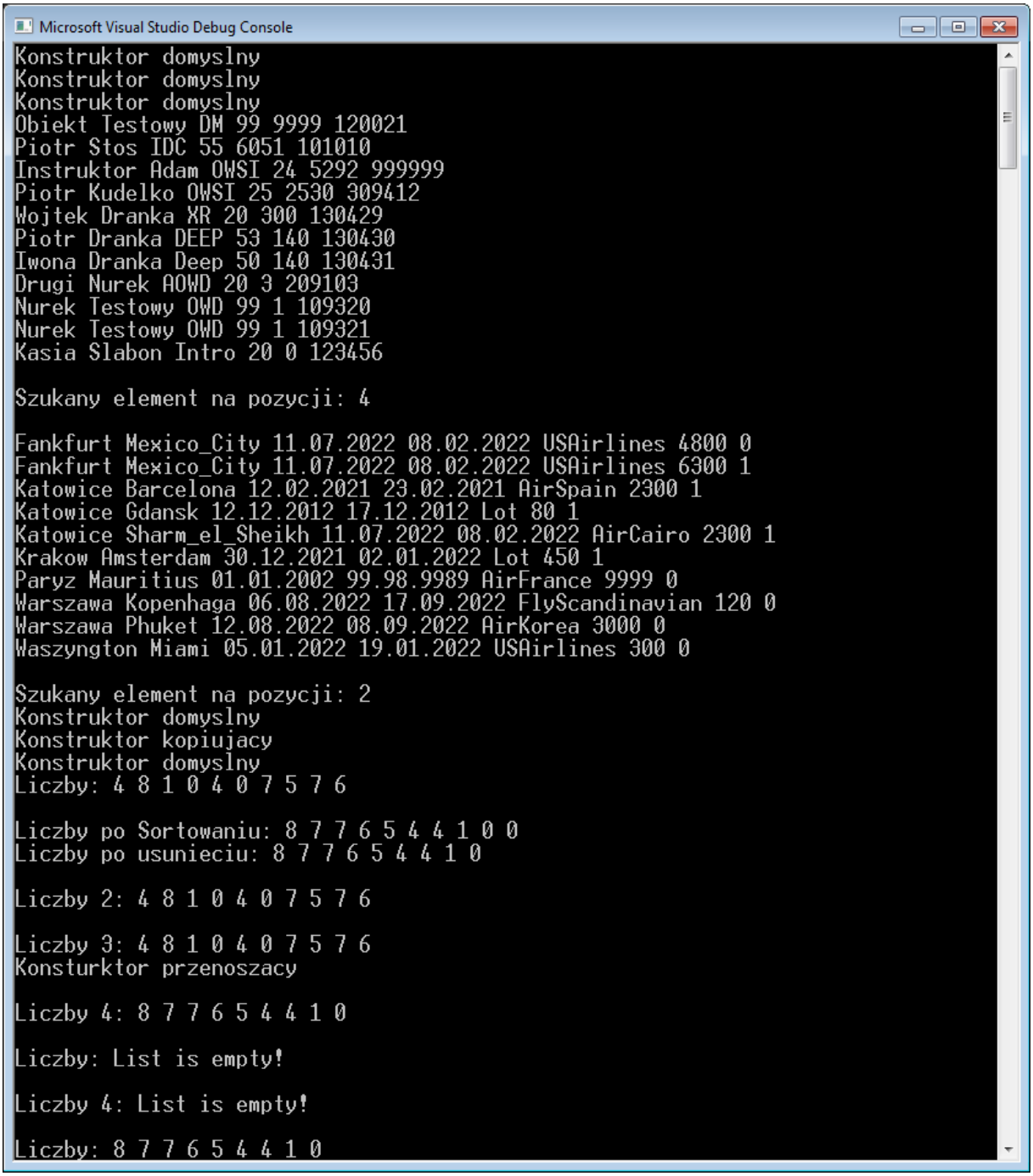
Jako drugi argument podajemy funkcję porównującą, którą należy zdefiniować samemu. Dla typów standardowych, podajemy jedynie pierwszy argument.

- Do operacji na plikach służą funkcje ***Read*** i ***Write***¸ które jako argument przyjmują nazwę pliku wejściowego lub wyjściowego.

- Możemy pobrać rozmiar listy przy używając funkcji ***Size***.

- Istnieje możliwość korzystania z operatorów przypisania i przeniesienia oraz konstruktorów kopiujących i przenoszących.

Przykład działania programu



Specyfikacja Wewnętrzna

Klasa ***List*** posiada następujące atrybuty prywatne:

- `shared_ptr<node> head` - wskaźnik na pierwszy element listy.
- `shared_ptr<node> tail` - wskaźnik na ostatni element listy.
- `struct node` - pomocnicza struktura, stanowiąca kolejne węzły.
  - `shared_ptr<node> prev` – wskaźnik na kolejny węzeł.
  - `shared_ptr<node> next` – wskaźnik na poprzedni węzeł.
  - `T data` – dane przechowywane w konkretnym węźle.
- `int counter` - licznik elementów (węzłów).

Metody prywatne klasy List:

- `void swap(T& data1, T& data2)` – funkcja pomocnicza używana przy sortowaniu, powoduje zamianę ze sobą dwóch zestawów danych, które są przekazane jako argumenty.
- `auto Partition(shared_ptr<node> f, shared_ptr<node> l, bool up, C cmp)` – główna funkcja sortująca, wykonująca sortowanie szybkie. Argumenty f i l to odpowiednio wskaźniki na początek i koniec obszaru, na którym ma być wykonywane sortowanie. Argument up określa
  czy sortowanie ma się odbywać w górę (true) czy w dół (false. Jako cmp jest przekazywana funkcja porównująca, która powinna zostać zdefiniowana przez programistę, chyba że operujemy na typach standardowych.
- `void QuickSort(shared_ptr<node> f, shared_ptr<node> l, bool up, C cmp)` – funkcja pomocnicza używana przy sortowaniu, jest wywoływana rekurencyjnie. Przeciążona, aby mogła być używana dla typów standardowych jak i zdefiniowanych przez użytkownika.

Metody publiczne klasy List:

- `List()` – konstruktor domyślny przypisujący wskaźnikom head i tail wartość NULL, oraz nadający licznikowi (counter) wartość 0. Przy jego wywołaniu wypisywany jest komunikat „Konstruktor domyślny”.
- `~List()`– Destruktor, przy którego wywołaniu wypisywany jest komunikat „Destruktor”.

- `List(const List<T>& new_list)` – konstruktor kopiujący
- `List(List&& new_list)` – konstruktor przenoszący
- `operator = (const List<T>& new_list)` – operator przypisania.
- `operator = (List&& new_list)` – operator przeniesienia.
- `void Prepend(const T& new_data)` – funkcja dodająca węzeł o zadanej zawartości na początek listy.
- `void Append(const T& new_data)` – funkcja dodająca węzeł o zadanej zawartości na koniec listy.
- `void Delete(int n)` – funkcja usuwająca element znajdujący się na podanym indeksie. Jeżeli indeks jest nieprawidłowy (ujemny lub większy od maksymalnego) to zwracany jest komunikat. Szukanie odbywa się albo zaczynając od tyłu albo od początku w zależności, do którego końca jest bliżej.
- `void Display()` – funkcja wypisująca wszystkie elementy po kolei zaczynając od początku.
- `int Size()`– funkcja zwracająca rozmiar Listy .
- `int Find(T key)` – funkcja zwracająca indeks elementu, który odpowiada podanemu w argumencie kluczowi. W przypadku nie znalezienia danego elementu, zwracana jest informacja oraz wartość -1.
- `void Sort(bool up, C cmp)` – funkcja sortująca, jej pierwszy element określa kierunek sortowania, drugi zaś funkcję porównującą, która powinna zostać wcześniej zdefiniowana przez programistę. Drugi argument nie jest konieczny jeśli sortowane są typy standardowe.
- `void Write(string s)` – funkcja zapisująca listę do pliku. Jej argumentem jest nazwa pliku wyjściowego.
- `void Read(string s)` – funkcja czytująca dane z pliku, które następnie są umieszczane w liście. Kolejne elementy są dodawane na koniec struktury. Za argument przyjmuję nazwę pliku wejściowego.

Klasa Diver – prosta klasa służąca do testowania Listy zawierająca atrybuty:

- `string m_name` – imię osoby.
- `string m_surname` – nazwisko.
- `string m_certificate` – stopień uprawnień.
- `int m_age` – wiek.
- `int m_dives` – ilość nurkowań.
- `int m_ID` – numer identyfikacyjny.

W dodatku posiada metody zwracający każdy z atrybutów oraz przeciążone operatory strumieniowe oraz operator porównania (aby możliwe było użycie funkcji List::Find).

Klasa Flight – prosta klasa służąca do testowania Listy zawierająca atrybuty:

- `string m_from` – miejsce wylotu.
- `string m_to` – miejsce przylotu.
- `string m_departure` – data odlotu.
- `string m_arrival` – data przylotu
- `string m_airlines` – linie lotnicze.
- `int m_price` – cena za bilet.
- `bool m_available` – dostępność biletu.

W dodatku posiada metody zwracający każdy z atrybutów oraz przeciążone operatory strumieniowe oraz operator porównania (aby możliwe było użycie funkcji List::Find()).

Jeśli chcemy zastosować sortowanie dla klasy zdefiniowanej przez użytkownika, należy zadeklarować własną funkcję porównującą, która zostanie przekazana jako argument funkcji Sort np.:

```
bool cmpDiverDives(Diver x1, Diver x2) {
    return x1.GetDives() < x2.GetDives(); // porównujemy ilość nurkowań
}


bool cmpFlightFrom(Flight x1, Flight x2) {

    return x1.GetFrom() < x2.GetFrom();  // porównujemy miejsce wylotu
    }
```

Nie ma sensu rysowanie diagramu hierarchii klas, ponieważ w programie nie jest zastosowane dziedziczenie. Nie było takiej potrzeby.

Program jest na tyle prosty, że z technik obiektowych została wykorzystana jedynie enkapsulacja.

Testowanie i Uruchamianie

Program był testowany na bieżąco. Wszystkie zaimplementowane przeze mnie funkcje działają zarówno dla typów standardowych jak i dla klas pomocniczych. Podczas pisania programu nie natrafiłem na większe problemy ani błędy. Program był również testowany przy pomocy specjalnych gotowych funkcji pod kątem wycieków pamięci, jednak nie miały one miejsca. Jedyny problem sprawiło zapisywanie plików w formacie binarnym. Program kompiluje się zarówno w wersji Release jak i Debug (dla IDE Visual Studio).