Laboratorium 5

Stosy (1)

1. Oprogramuj w klasie `Stos` strukturę danych zwaną stosem.
   (a) Wykorzystaj do tego celu tablicę statyczną (o niedużej pojemności).
   (b) Wykorzystaj do tego celu kolekcję `vector`.
   Zdefiniuj metody: `pusty`, `pelny`, `dodaj`, `usun`, `szczytowy`, `oproznij`, oraz metodę wirtualną `pojemnosc`, która podaje bieżącą pojemność stosu.
   Zastosuj mechanizm programowej obsługi wyjątków do rozwiązania problemu nierealizowalnych żądań obsługi stosu (dodawanie elementu do pełnego stosu (a), usuwanie elementu z pustego stosu (a) i (b), odczyt elementu szczytowego ze stosu pustego (a) i (b)).

   Polecenia dodawania i usuwania elementów ze stosu mają postać, odpowiednio:

```
D  liczba   //dodaj element liczba
U          //usun szczytowy element
O          //odczytaj szczytowy element
```

   Przykład (wariant a)

   Dane wejściowe (dla stosu o pojemności 3):

```
D 1
D 2
O
U
D 3
D 4
D 5
U
U
U
U
```

   Dane wyjściowe:

```
Szczytowy: 2
Zdjety: 2
Blad: nie można dodac elementu do stosu pelnego (5)
Zdjety: 4
Zdjety: 3
Zdjety: 1
Blad: stos jest pusty
```

Stosy (2)

2. Dla klasy bazowej `Stos` (zadanie 5.1) zdefiniuj klasę pochodną `Dwa_stosy`, która zawiera metody do działań na dwóch stosach: `usun_pare`, `dodaj_pare`, `szczytowe`.
   Rozwiąż zadanie w dwóch wariantach, stanowiących rozwinięcie zadania 5.1(a) i 5.1(b).
   Zdefiniuj metodę wirtualną `pojemnosc` w taki sposób, by informowała o pojemności tego ze stosów, który w danej chwili zawiera większą liczbę elementów.
   Zdefiniuj w klasie `Stos` operator konwersji jej obiektu do postaci obiektu klasy pochodnej `Dwa_stosy`.
   Na końcu pracy ze stosami, poinformuj o stanie obu stosów.
   Zastosuj mechanizm programowej obsługi wyjątków do rozwiązania problemu nierealizowalnych żądań obsługi stosu (dodawanie elementu do pełnego/pełnych stosu/stosów (a), usuwanie elementu z pustego/pustych stosu/stosów (a) i (b), odczyt elementu szczytowego ze stosu/stosów pustego/pustych (a) i (b)).
   Polecenia dodawania i usuwania elementów ze stosów mają postać, odpowiednio:
   `D1` *liczba*
   `D2` *liczba*
   `D12` *liczba liczba*
   `U1`
   `U2`
   `U12`
   Uruchom program dla różnych zestawów danych.

   Przykład (wariant a)

   Dane wejściowe (dla stosów o pojemności 5):

```
D1  5
D1  9
D1 -12
U1
D1  17
D1 100
D2 10000
D1  0
D1   8
D2  -2
U2
U2
U1
U1
D12   5  8
U12
U12
```

   Dane wyjściowe:

```
Zdjety: (s1, -12)
Blad: nie można dodac elementu do stosu pelnego (s1, 8)
Zdjety: (s2, -2)
Zdjety: (s2, 10000)
Zdjety: (s1, 0)
Zdjety: (s1, 100)
Zdjety: (s1, 5)
Zdjety: (s2, 8)
Zdjety: (s1, 17)
Blad: nie można usunac elementu ze stosu pustego (s2)
S1 zawiera 2 elementy: 9 5
s2 jest pusty
```

Drzewo BST

3. Zaprojektuj i zaimplementuj klasę `BuildandReconstructBST`, przeznaczoną do budowy i rekonstrukcji drzew poszukiwań binarnych z elementami unikalnymi. Klasa ma zawierać metody:
   – `AddElement`,
   – `DeleteElement`,
   – `PrintTree`.
   W metodzie `AddElement` zgłoś wyjątek oznaczający:
   – (a) niemożność wykonania operacji ze względu na występowanie wskazanego elementu w drzewie.
   W metodzie `DeleteElement` zgłoś wyjątki oznaczające:
   – niemożność usunięcia elementu z powodu:
   (b) „pustości" drzewa,
   (c) niewystępowania elementu w (niepustym) drzewie,
   – (d) trudność wynikającą z faktu występowania poszukiwanego elementu w węźle niebędącym liście.
   Wyjątki (a), (b) i (c) obsłuż w metodach, zaś wyjątek (d) propaguj do miejsca wywołania metody. Tam dokonaj jego obsługi polegającej na rekonstrukcji drzewa.
   Zademonstruj działanie wszystkich metod w odpowiednio skonstruowanym programie (funkcja `main`).