Aufgabe – Dynamischer Stack

Einen Stack haben Sie in Aufgabe 5 bereits implementiert. Programmieren Sie nun einen Stack, der seine Elemente dynamisch verwaltet. Dieses Mal sind die verwalteten Daten Objekte der Klasse string (C++-Bibliothek string).

Wir unterscheiden in dieser Aufgabe einen Stack (Klasse CStackDyn) von den Elementen, die der Stack speichert (Klasse CElement). So kommen wir ohne ein sogenanntes Root-Element aus.

Implementieren Sie den dynamischen Stack in der entsprechenden *.cpp-Datei zur gegebenen Header-Datei.

Die Main-Funktion zum Testen Ihrer Implementierung können sie aus Aufgabe 5 übernehmen. Da der Stack hier voll dynamisch ist, entfällt die Angabe der (maximalen) Größe.

Geben Sie analog zu Aufgabe 5 Screendumps ab, aus denen Ihre sinnvoll gewählten Testfälle ersichtlich sind.

Datei „CStackDyn.h“:

```
1   #pragma once
2   #include <string>
3   using namespace std;
4
5   class CElement
6   // Objekte dieser Klasse repräsentieren die Elemente, die vom Stack
7   // verwaltet werden.
8   {
9   public:
10      CElement(string data);  // Konstruktor mit Angabe eines Strings (Daten).
11
12                              // Der Default-Konstruktor (ohne Parameter)
13                              // steht nach obiger Deklaration nicht mehr
14                              // automatischzur Verfügung (wird hier aber auch
15                              // nicht zwingend benötigt).
16
17                              // Auch einen eigenen Destruktor benötigen wir
18                              // nicht, da CElement keine dyn. Elemente hat.
19      string    mData;
20      CElement* mPtrNext;
21  };
22
23  class CStackDyn
24  // Ein Objekt dieser Klasse CStackDyn repräseniert den Stack, bestehend aus
25  // Objekten der Klasse CElement.
26  // Ein Stack hat einen Konstruktor und die bekannten Methoden push und pop.
27  // Darüber hinaus hat der Steck einen Zeiger auf das zuletzt mit push
28  // eingefügte Element (Objekt der Klasse CElement).
29  // Ist dieser Zeiger gleich NULL (Nullpointer), ist der Stack leer.
30  // Somit benötigen wir kein Root-Element, wie wir es in der letzten
31  // Imlementierung eines Stacks verwendet haben.
32  {
33  public:
34          CStackDyn();               // Konstruktor, erzeugt einen leeren Stack.
35          ~CStackDyn();              // Destruktor.
36      bool push(const string& data); // Daten auf dem Stack ablegen.
37      bool pop(string& data);        // Daten vom Stack herunternehmen.
38      void display(void);            // Gibt den gesamten aktuellen Inhalt des
39                                     // Stacks auf dem Bildschirm aus. Der
40                                     // Stack selbst bleibt dabei unverändert.
41  private:
42      CElement* mPtrHead;            // Zeiger auf den "Kopf" des Stacks (zuletzt
43                                     // eingefügtes Element). NULL, wenn leer.
44  };
```

Viel Spaß und Erfolg bei der Bearbeitung!