Algorithmen & Datenstrukturen: Stack als Liste implementiert

Klaus Kusche

Gesucht ist ein C++-Programm, das einen *beliebigen Text* von **`cin`** einliest
und *in umgekehrter Wortreihenfolge* wieder ausgibt
(wobei "Wort" für uns eine Folge sichtbarer Zeichen getrennt durch White Space ist,
also genau das, was **>>** pro Aufruf in eine **`string`**-Variable einliest).

Dazu benötigt man einen *Stack von Strings*, in dem man die Worte beim Einlesen
speichert und aus dem man sie zum Ausgeben wieder herausholt.
Wir wollen aber *nicht* die vordefinierte Template-Klasse **`stack`** verwenden,
sondern unseren *eigenen Stack* schreiben.
Intern soll unser Stack eine *einfach verkettete Liste* verwenden.

Als erstes brauchen wir eine *Klasse* **`Elem`** für *ein einzelnes Listen-Element*.
Sie soll *ausschließlich für unseren Stack verwendbar* sein (wie schafft man das?)
und enthält:

- *Zwei Member*: Eines für den **`string`** (die "Nutzdaten" unserer Liste)
  und eines für die *Listen-Verkettung*.
- Einen *Konstruktor mit zwei Parametern*,
  die in die beiden Member des neuen Objektes gespeichert werden
  (d.h. man legt gleich beim Anlegen des Elementes seine Nutzdaten
  und seine Verkettung fest).

Set- oder Get-Methoden braucht die Klasse nicht,
da die Stack-Klasse die beiden *Member direkt lesen und schreiben* kann.

Dann kommt die *Stack-Klasse* (ich habe sie **`StrStack`** genannt). Sie enthält:

- Ein *rein internes Member*: Den *Listenkopf-Pointer*
  (bei einem Stack braucht man keinen Pointer auf das Listenende).
- Einen *Konstruktor ohne Parameter*, der die Liste auf *"leer"* setzt
  (was muss er dazu wo speichern?).
- Eine *Methode* **`push`** mit einem **`string`**-Referenz-Parameter und *ohne* Returnwert,
  die einen String im Stack *speichert*:
  Sie legt *dynamisch ein neues Listenelement* mit diesem String an
  und hängt es *als vorderstes Element* in die Liste.
- Eine *Methode* **`pop`** mit **`string`**-Referenz-Parameter und **`bool`**-Returnwert,
  die das *oberste Element aus dem Stack* entfernt und dessen String im Parameter
  zurückliefert.

  Sie prüft zuerst, ob die *Liste leer* ist. Wenn ja, returniert sie sofort **`false`**.

  Sonst *entnimmt* sie das *vorderste* Element aus der Liste (wie?),
  speichert dessen String im Parameter, *vernichtet* das Element,
  und kehrt mit **`true`** zurück.

Das **Hauptprogramm** legt zuerst einen *Stack* an (*als lokale Variable*, nicht dynamisch).

Dann liest es in einer *Schleife* immer wieder einen **`string`** von **`cin`**,
bis das Lesen fehlschlägt, und *speichert* jeden String mit **`push`** im Stack.

Erinnere dich:

- Ein Einlesen mit **>>** kann man *direkt als Schleifenbedingung* verwenden:
  Es liefert einen Wert, der als **`true`** gilt, wenn es *erfolgreich* gelesen hat,
  und **`false`**, wenn es *nichts lesen* konnte.
- Zum *Beenden der Eingabe* kann man im DOS-Fenster mit **`Strg/Z`**
  am Beginn einer Zeile das "Dateiende" signalisieren, unter Linux mit **`Strg/D`**.

Nach der Eingabeschleife kommt die *Ausgabeschleife*:
Mit **`pop`** wird immer wieder ein **`string`** *vom Stack genommen* und auf **`cout`** *ausgegeben*,
bis **`pop`** mit **`false`** anzeigt, dass der Stack *leer* ist.