Programmieren C++: Operatoren: String-Klasse

Klaus Kusche

Wir tun so, als ob es die vordefinierte C++-Klasse **`string`** nicht gäbe, und wollen eine *ganz einfache String-Klasse* **`String`** schreiben. Unsere Klasse enthält nur *ein Member*, nämlich einen **`char *`**. Dieser zeigt auf einen ganz normalen, mit **`new`** *dynamisch angelegten C-String* (d.h. auf ein **`char`**-Array mit einem **`'\0'`** als Endemarkierung).

Weiters soll die Klasse folgende Methoden haben:

- Einen **Konstruktor**, der mit der *Länge* des anzulegenden Strings aufgerufen wird (zähl beim Anlegen des Arrays zur Länge eins dazu, damit die Endemarkierung zusätzlich zur angegebenen Anzahl von Zeichen Platz hat!).

  Der String soll auf den *Leerstring* initialisiert werden (erstes Zeichen **`'\0'`**).

  Dieser Konstruktor soll zugleich *Standard-Konstruktor* (Ergebnis Leerstring) sein: Gibt man keine Länge an, wird er mit *Länge 0* aufgerufen, liefert also ein Objekt, dessen C-String Länge 1 hat und nur aus der Endemarkierung besteht.
- Einen **Copy-Konstruktor** (wie üblich: **`new`** in der richtigen Länge plus **`strcpy`**).
- Einen dritten **Konstruktor**, mit einem C-String (**`char *`**) aufgerufen wird und ein Objekt mit einer *Kopie dieses Strings* liefert.
- Einen **Destruktor** (was tut der?).
- Einen **+-Operator**, dessen Ergebnis die *zusammengehängten* Strings beider Operanden enthält.

  Tipp: Rufe für das Ergebnis-Objekt den Konstruktor mit der richtigen Länge auf und kopiere dann die Strings der beiden Operanden mit **`strcpy`** und **`strcat`** in das Array des Ergebnis-Objektes.
- Einen **+=-Operator**, der den rechten String an den linken *anhängt*.

  Tipp: Leg zuerst mit **`new`** ein Array in der richtigen Länge für das Ergebnis an, kopiere wie oben beide Strings hinein, gib den alten eigenen String frei, und lass den Pointer des eigenen Objektes dann auf das zuvor neu angelegte Array zeigen!
- Die beiden **Vergleichs-Operatoren** **<** und **==** (verwende **`strcmp`**!).
- Einen unären **Operator `!`** (“Not”) mit **`bool`**-Ergebnis:
  Er liefert **`true`**, wenn der Operand der *Leerstring* ist, und sonst **`false`**.
- Einen **Zuweisungsoperator** mit der üblichen Funktion.

  Tipp: Prüfe wie üblich auf *Selbstzuweisung* und vergiss nicht, vor dem **`strcpy`** den alten eigenen String *freizugeben* und in der richtigen Größe *neu anzulegen*!

Versuche, *völlig ohne Schleifen* auszukommen:
Die C-String-Funktionen **`strcpy`**, **`strcat`**, **`strlen`** und **`strcmp`** sollten reichen.
(*Achtung:* Für das **`new`** muss man zum Ergebnis von **`strlen`** *eins dazuzählen*! Warum?)

Implementiere auch den üblichen **Ausgabe-Operator <<** für Objekte deiner String-Klasse als *Friend-Funktion* (C-Strings kann man ganz normal mit **<<** ausgeben).

Ich liefere dazu ein **Hauptprogramm**, um alle deine Methoden kurz zu testen.

Erweiterung für Studenten:
Index- und Typumwandlungs-Operator

Erweitere deine String-Klasse um *zwei weitere Operatoren*:

- Einen **Index-Operator `[]`**: Er soll wie bei normalen Strings *das i-te Zeichen liefern*, und zwar so, dass man das Zeichen *sowohl lesen als auch ändern* kann (d.h. die **`[]`** sollen auch auf der *linken Seite einer Zuweisung* verwendbar sein).

  Eine Variante für *konstante* String-Objekte ist *nicht gefragt*, der Normalfall genügt.

  Wenn der Index **`i`** *negativ oder zu groß* ist, soll statt einem Zeichen des Strings eine in der Methode *lokal angelegte* **`char`**-Variable zurückgegeben werden. Diese Variable ist *vor jeder Rückgabe* auf die Ende-Markierung **`'\0'`** zu setzen.

  Wie musst du diese zusätzlich Variable *deklarieren*, damit sie beim **`return`** *nicht verschwindet*?

- Ein **Typumwandlungs-Operator** auf einen *Standard-C++-String* (Typ **`string`**).

  Es gibt bereits einen *vordefinierten Typumwandlungs-Operator* von **`char *`** auf **`string`**. Verwende diesen, um den internen **`char *`**-String umzuwandeln, und gib das Ergebnis zurück.