

Einführung in die Programmiersprachen C und C++

Prof. Dr. Ulf Rehmann, Fakultät für Mathematik

Übungsblatt 10 (4 Seiten)

Hier ist die Implementation der Klasse `string`, die schon mehrfach verwendet worden ist.

Sie arbeitet mit sogenannten Referenzzählern, die Buch darüber führen, wie oft ein Objekt zur Laufzeit vom Programm gerade verwendet wird.

Die Klasse verwendet neben den üblichen Konstruktoren auch sogenannte "Destruktoren" – das sind Member-Funktionen, die das Verschwinden von Variablen z. B. am Ende eines Anweisungs-Blocks automatisieren. Eine Destruktor-Funktion trägt – ähnlich wie eine Konstruktor-Funktion – den Namen der Klasse, jedoch mit einer vorangestellten Tilde `~`, in diesem Fall also `~string()`. Solche Destruktor-Funktionen werden in der Regel automatisch und implizit aufgerufen, z. B. beim Verlassen einer Funktion oder eines "Compound Statements". Z. B. können sie mit `delete` oder `delete[]` Speicher freigeben. (Hinweis: Die Befehle `new[...]`, `delete[]` können Arrays von Objekten allokalieren oder de-allokalieren.)

Die Klasse `string` besteht aus einer Komponente `p`, ein Zeiger auf die Struktur `srep`, die aus den Komponenten `char *s` und `int n` besteht. `s` zeigt auf einen ganz gewöhnlichen C-String. `n` ist ein Referenzzähler. Da ein String sehr lang sein kann, ist es nicht zweckmäßig, ihn bei Zuweisungen, Funktionsaufrufen oder Werterückgaben jedesmal zu kopieren, daher arbeitet man in diesem Programm mit Referenzen: Jedesmal werden in solchen Situationen nur Referenzen auf den String, also Adressen, übergeben, und der String jeweils nur in einfacher Kopie gehalten.

Um den Speicher, den er verbraucht, dennoch "dynamisch" verwalten zu können, also den Speicherplatz für einen String genau dann freizugeben, wenn der String selbst vom Programm nicht mehr gebraucht wird, führt man in der Komponente `n` Buch über die Anzahl der Referenzen, die gerade auf den String zeigen.

Ein Code

```
string s = "Hallo!";
string t;
t = s;
```

wird also in der ersten Zeile den Referenzzähler von `s` auf 1 setzen. In der dritten Zeile wird er auf 2 gesetzt, und es wird nicht, wie bei gewöhnlichen Zuweisungen, sein Wert kopiert.

Das Beispiel-Programm `test.cc` demonstriert dies etwas ausführlicher.

Dazu ist in `string-class.h` die Hilfs-Member-Funktion `string::check()` definiert worden.

```
// test.cc
#include <iostream.h>
#include <stdio.h>
#include "string-class.h"

string s = "Hallo Welt";

string reverse(string u) {
    u.check("reverse: u");
    string t = u;
    t.check("reverse: t");
    int i,j;
    for(i=0, j = strlen(t)-1; i<j ; i++, j--) {
        char c = t[i];
        t[i] = t[j];
        t[j] = c;
    }
    t.check("reverse: t");
    return u;
}
```

```

main() {
    string t;
    s.check("s");t.check("t");
    t = s;
    s.check("s");t.check("t");
    t = reverse(s);
    s.check("s");t.check("t");
    s = s + " Welt!";
    s.check("s");t.check("t");
    s = t + " " + (t = s);
    s.check("s");t.check("t");
    s.check("s");t.check("t");
}

```

Aufgabe 10.1: Lesen Sie string-class.h und interpretieren Sie die Implementationen der einzelnen Klassenfunktionen. Erweitern Sie das Test-Programm, um die Wirkungsweise von Kon-/Destructoren zu verstehen.

Hier ist string-class.h :

```

#include <string.h>
void error(char *s, int i, int j, char *w) {    // Fehlerausgabe
    cerr << s << i << " " << j << " " << w << "...\\n";
}
#define STRCMP strcoll // statt: strcmp

class string {
    struct srep {                // String repraesentiert durch
        char *s;                // Zeiger auf char
        int n;                  // Referenz-Zaehler
        srep() { n = 1; }       // initiiert auf 1;
    };
    srep *p;                    // und Zeiger auf srep
public:
    string();                   // string x;
    string(const char *);       // string x = "abc";
    string(const string &);     // string y = x;
    string& operator=(const char *); // Zuweisung x = "abc";
    string& operator=(const string& ); // Zuweisung y = x;
    ~string();                  // Destruktor;
    char& operator[](int i);     // Ueberladung des Index-Operators []
    char operator[](int i) const; // wird nicht genommen. ??
    const char * cstring() { return p->s; }
    operator char *() const { return p->s; }
    operator unsigned char *() const { return (unsigned char *)p->s; }
    friend ostream& operator<<(ostream&, const string&);
    friend istream& operator>>(istream&, string&);

    friend int operator==(const string& x, const char *s)
    { return STRCMP(x.p->s, s) == 0; }

    friend int operator==(const string& x, const string &y)
    { return STRCMP(x.p->s, y.p->s) == 0; }
}

```

```

friend int operator!=(const string& x, const char *s)
{ return STRCMP(x.p->s, s) != 0; }

friend int operator!=(const string& x, const string &y)
{ return STRCMP(x.p->s, y.p->s) != 0; }
friend int operator<(const string& x, const string &y)
{ return STRCMP(x.p->s, y.p->s) < 0; }
friend int operator>(const string& x, const string &y)
{ return STRCMP(x.p->s, y.p->s) > 0; }

friend string operator+(const string& x, const string &y) {
    string s;
    s.p->s = new char[strlen(x.p->s) + strlen(y.p->s) + 1];
    strcpy(s.p->s, x.p->s); strcat(s.p->s, y.p->s);
    return s;
}
int len() { strlen(p->s); }
friend int strlen(const string& u) {
    return strlen(u.p->s);
}
void check(char *u) { // Test-Funktion
    printf("%s : -->n = %3d, s = %s\n", u, p->n, p->s);
}
};

string::string() {
    p = new srep;
    p->s = 0;
}

string::string(const string& x) {
    x.p->n++;
    p = x.p;
}

string::string(const char *s) {
    p = new srep;
    p->s = new char[ strlen(s) + 1];
    strcpy(p->s, s);
}

string::~~string() { // Wenn nicht mehr benoetigt,
    if (--p->n == 0) { // wird aufgeraeumt.
        delete[] p->s;
        delete p;
    }
}

string& string::operator=(const char *s) { // Kopie C-string nach string
    if (p->n > 1) { // Ggf. wird ein vorhandener String ueberschrieben
        p->n--;
        p = new srep; // Neuer srep noetig
    }
}

```

```

    }
    else
        delete[] p->s;    // Loeschen, falls nicht mehr gebraucht
        p->s = new char[strlen(s)+1]; // Platz fuer neuen C-String
        strcpy(p->s ,s);
        return *this;
}

string& string::operator=(const string& x) { // Kopie string nach string
    x.p->n++;           // Eine neue Instanz verwendet die "rechte Seite"
    if (--p->n == 0) { // Die "alte" linke Seite wird einmal weniger benoetigt
        delete[] p->s;    // und damit vielleicht gar nicht mehr
        delete p;
    }
    p = x.p;
    return *this;
}

ostream& operator<<(ostream& s, const string& x) {
    // return s << x.p->s << " { Ref-count = " << x.p->n << "}\n";
    return s << x.p->s;
}

istream& operator>>(istream& s, string& x) {
    char buf[1024];
    s.width(1024);
    s >> buf;           // ohne width : Ueberlaufgefahr
    x = buf;
    return s;
}

char& string::operator[](int i) { // Ueberladen von []
    //cout << "first\n";
    if ( (i<0) || (strlen(p->s) < i) )
        error(" 1. Index ausser Bereich : ", i, strlen(p->s), p->s );
    if (p->n > 1) {        // Lege neues Array an zugewiesen werden soll
        srep* np = new srep;
        np->s = new char[ strlen(p->s) + 1 ];
        strcpy(np->s, p->s);
        p->n--;
        p = np;
    }
    return p->s[i];
}

char string::operator[](int i) const {
    //cout << "second\n";  wird anscheinen nicht genommen
    if ( (i<0) || (strlen (p->s)) < i)
        error("2. Index ausser Bereich", i, strlen(p->s), p->s );
    return p->s[i];
}

```