string1_false.cpp - Beispiel für falsch entworfene Klasse mit C-string

```
#include <iostream>
using namespace std;

class zk {// Beispiel fuer Klasse mit Zeichenkette und was ein
          // Anfaenger alles falsch machen kann !!!

          char *s;                // Zeiger auf Zeichen (Zeichenkette)
    public:
          zk(char *z = 0):s(z){ // unbedingt vermeiden !
          //s und externes z zeigen auf den gleichen Speicher, falls dieser mittels
          //einer Zeigervariablen freigegeben wird, dann erzeugt der Zugriff seitens
          //der anderen Zeigervariablen einen Speicherzugriffschutzfehler
             cout<<"Konstruktor zk, s = "<<(this->s ? this->s : "0")<<endl;
             cout<<"        Adresse s = "<<(int *)s<<endl;
          }

          // delete [] s; ist falsch, falls s nicht mit new angelegt wurde !
          ~zk(){ cout<<"Destruktor zk,  s = "<<s<<endl; delete [] s; s=0; }

         char * get_s(){ return s; } //unbedingt vermeiden, Zeichenkette s (Adresse)
          //wird zurückgegeben, ohne const, s kann damit von ausserhalb des
          //Objektes veraendert und mit delete [] freigegeben werden

         void set_s(char *z = 0){this->s = z; } //unbedingt vermeiden, s und ext. z
          //mit gleichem Speicher, hier entsteht das gleiche Probleme wie oben beim
          //Konstruktor beschrieben

        zk &operator=(zk &zkr){ this->s = zkr.s; return *this; } //von C++ generiert
};

void main(){
           char *z = strcpy(new char[strlen("HTW Dresden")+1], "HTW Dresden");
```

string1_false.cpp - Beispiel für falsch entworfene Klasse mit C-string

```
    cout<<"        Adresse z = "<<(int *)z<<endl;
    zk *s1 = new zk(z); // ab hier: s1->s == z
    // erste Fehlermoeglichkeit:
    delete [] z; z=0;   // z freigeben, worauf zeigt s1->s ???
    cout<<"s1->s = "<<s1->get_s()<<endl;// Abbruch, s1->s ex. nicht mehr!

    // zweite Fehlermoeglichkeit:
    // Rueckgabe des Zeigers s des Objektes
    // *s1 an den Zeiger t, damit zeigt t ausserhalb des Objektes auf
    // den gleichen Speicher, auf den auch s von *s1 zeigt. Hier wird
    // t einfach mit delete [] t freigegeben, damit wird s von *s1
    // ungueltig und s1->get_s() erzeugt einen Fehler:

    char *t = s1->get_s(); delete [] t; t=0;  // Zugriff auf s1->s
    cout<<"s1->s = "<<s1->get_s()<<endl;      // Abbruch !!

    // dritte Fehlermoeglichkeit:
    // kein explizit formulierter Kopierkonstruktor zur Zeigertrennung
    // s2->s und s1->s zeigen auf identischen heap - Speicher:

    zk *s2 = new zk(*s1);
    delete s2; s2 = 0;
    cout<<"s1->s = "<<s1->get_s()<<endl;      // Abbruch !!

    // vierte Fehlermoeglichkeit:
    // kein explizit formulierter Zuweisungsoperator zur Zeigertrennung
    // s3->s und s1->s zeigen auf identischen heap - Speicher:
    zk *s3 = 0; *s3 = *s1;
    delete s3; s3 = 0;
    cout<<"s1->s = "<<s1->get_s()<<endl;      // Abbruch !!
    delete s1; s1=0;
}
```