Objekti sa zauzetim resursima

- Potpuno analogno, ako se predviđa korišćenje ovih objekata po vrednosti (svih kategorija životnog veka), korisnici ove klase očekivaće i mogućnost dodele kopiranjem. Na primer:

```
int main () {
  string s1(“Hello”), s2;
  s2 = s1;
  …
}
```

- Prema već navedenim pravilima, ako se u klasi ne navede eksplicitno operator dodele kopiranjem, prevodilac će generisati implicitni operator dodele kopiranjem koji vrši dodelu kopiranjem objekata članova pozivom njihovih operatora dodele kopiranjem; međutim, za članove koji su ugrađenih tipova, vrši se prosto kopiranje vrednosti, što je ovde slučaj, pošto je član *str* pokazivač. Prema tome, i u ovom slučaju objekti klase *string* biće dodelom plitko kopirani

- Zato nam je ovde neophodan i korisnički definisan operator dodele kopiranjem koji vrši duboko kopiranje, ali za razliku od konstruktora kopije, on mora najpre da oslobodi postojeći dinamički niz (ono što radi destruktor), pa onda alocira i kopira novi niz (ono što radi konstruktor kopije):

```
string& string::operator= (const string& s) {
  if (this == &s) return *this;
  delete [] str; str = nullptr;
  if (!s.str) return;
  str = new char[std::strlen(s.str)+1];
  std::strcpy(str,s.str);
  return *this;
}
```

U slučaju poziva *s=s*, ne radi dalje, jer bi to obrisalo dinamički niz objekta *s*

Zadatak:
Ova implementacija obezbeđuje osnovnu garanciju sigurnosti od izuzetaka. Objasniti zašto i prepraviti je tako da obezbeđuje jaku garanciju.

Objekti sa zauzetim resursima

- Pritom, ove operacije imaju neke zajedničke delove, pa je dobro *refaktorisati* (*refactor*) ovu klasu tako da se ti zajednički delovi izdvoje u pomoćne potprograme i tako eliminišu dupliranja koda:
  - konstruktor kopije radi alokaciju i kopiranje
  - destruktor radi dealokoaciju
  - operator dodele radi najpre dealokaciju (kao destruktor), pa onda alokaciju i kopiranje (kao konstruktor kopije)

```
class string {
public:
  string () : str(nullptr) {}
  string (const char* s) : string() { allocate(s); copy(s); }
  string (const string& s) : string(s.str) {}
  string& operator= (const string& s);

  ~string () { release(); }
  …
protected:
  void allocate (const char* s) { if (s) str = new char[std::strlen(s)+1]; }
  void copy (const char* s) { if (s) std::strcpy(str,s); }
  void release () { delete [] str; str = nullptr; }
  …
};

inline string& string::operator= (const string& s) {
  if (this!=&s) {
    release(); allocate(s.str); copy(s.str);
  }
  return *this;
}
```