Objekti sa zauzetim resursima

- Prema tome, u ovakvim situacijama potreban je posebno definisan konstruktor kopije, operator dodele kopiranjem i destruktor; zato postoji preporuka da se, ako za klasu postoji potreba za jednom od ovih operacija, obrati pažnja i verovatno naprave sve tri, jer je u pitanju možda slučaj objekta sa zauzetim resursima koji se koristi i ugrađuje "po vrednosti"
- Treba primetiti da je sve ovo posledica ključne odluke da se objekti klasa mogu koristiti i ugrađivati "po vrednosti", odnosno da se mogu koristiti na isti način kao i objekti ugrađenih (neklasnih) tipova i biti svih kategorija po životnom veku
- Ukoliko to ne bi bio slučaj, kao što i nije u mnogim drugim novijim objektno orijentisanim jezicima, sve ove komplikacije ne bi bilo; u tim jezicima važi:
  - objekti su samo dinamički i uvek anonimni
  - objektima se pristupa samo preko posrednika (pokazivača, odnosno referenci)
  - postoje samo operacije tih klasa koje se pozivaju eksplicitno; nema operatorskih funkcija
  - nema ugrađenih objekata članova klasa (samo pokazivača/referenci), nema automatskih i statičkih objekata (samo pokazivača/referenci) itd.
  - nema implicitnih kopiranja prilikom inicijalizacije, dodele, prenosa argumenata i povratne vrednosti (kopiraju se i prenose samo pokazivači/reference na objekte)

Kopiranje objekata

- Pretpostavimo da smo implementirali operatorsku funkciju *operator+* koja vraća rezultat spajanja dva niza znakova iz operanada tipa *string*; taj rezultat mora biti objekat tipa *string* (po vrednosti), jer je on novonapravljeni objekat koji sadrži poseban dinamički niz znakova:

```
string operator+ (const string&, const string&);
```

- Osim toga, implementirali smo i nestatičku funkciju članicu *substr* koja vraća (ponovo po vrednosti) nov objekat tipa *string* koji sadrži samo podniz datog niza znakova objekta domaćina, počev od zadate pozicije i zadate dužine; za potrebe implementacije ove funkcije, napravili smo još nekoliko funkcija članica klase *string*:

```
inline void allocate (size_t sz) {
  if (sz+1==0) throw std::length_error;
  if (sz) str = new char[sz+1];
}

inline void copy (const char* s, std::size_t count) {
  if (!s || !count) return;
  size_t i = 0;
  while (i<count && *s) str[i++] = *s++;
  str[i] = '\0';
}

inline size_t size () const { return str?std::strlen(str):0; }

inline string substr (size_t pos, size_t count) const {
  size_t sz = size();
  if (pos>=sz) throw std::out_of_range;
  if (pos+count>sz) count = sz - pos;
  string s;
  s.allocate(count);
  s.copy(str+pos,count);
  return s;
}
```