Premeštanje resursa

- Pošto većina prevodilaca izostavlja kopiranje čak i kada je to neobavezno, a od verzije C++17 mnoge od navedenih optimizacija postale su obavezne, premeštanje zbog izbegavanja kopiranja resursa iz privremenih objekata gubi na značaju
- Međutim, semantika premeštanja ipak ima svoj značaj (koji postoji oduvek) i koji je možda i važniji od navedenog: premeštanje se može vršiti i na mestima za koje nisu predviđene optimizacije, ukoliko je ono efikasnije i nema potrebe za kopiranjem
- Na primer, bibliotečna šablonska klasa *vector* predstavlja niz promenljivih dimenzija koji se implicitno proširuje po potrebi (operacija *resize*). U tom slučaju za niz elemenata alocira se nov prostor, a elementi vektora se po vrednosti kopiraju na novo alocirano mesto. Ukoliko tip elementa vektora ima semantiku premeštanja, biće upotrebljeno premeštanje umesto kopiranja; ako odgovarajuće operacije premeštanja nisu definisane, vršiće se kopiranje
- Osim toga, neke apstrakcije ne dozvoljavaju kopiranje (jer to nema smisla), ali se njihovi alocirani resursi mogu premeštati; jedan takav primer je apstrakcija ulaznog ili izlaznog znakovnog toka (*istream* i *ostream*)
- U ovakvim situacijama moguće je i eksplicitno zahtevati semantiku premeštanja, iako bi se podrazumevano pozivao konstruktor kopije ili operator dodele kopiranjem: izraz koji je lvrednost se može eksplicitno konvertovati u dvrednost pozivom bibliotečne funkcije *std::move* koja vraća referencu na dvrednost za argument koji može biti i lvrednost; za ovakav rezultat onda se vezuju funkcije čiji parametri primaju reference na dvrednosti, pa i konstruktor premeštanja ili operator dodele premeštanjem:

```
string s1("Hello");
string s2 = std::move(s1);
```

Objekat *s2* biće inicijalizovan konstruktorom premeštanja i preoteće resurse iz objekta *s1*, recimo zato što objekat *s1* više nema potrebe za tim resursom

Premeštanje resursa

- Konstruktor premeštanja i operator dodele premeštanjem treba da preotmu, tj. premeste resurse iz izvorišnog u odredišni objekat, ali tako da izvorišni objekat ostave u konzistentnom stanju, tako da on i dalje bude validan objekat koji se svakako može uništiti (jer se on svakako uništava pozivom destruktora); zato ove operacije po pravilu imaju parametar koji je referenca na nekonstantu
- Za primer klase *string*, to znači da se objekat član *str* u izvorišnom objektu postavi na *null* vrednost, a ne ostavi na staroj vrednosti, jer preuzeti dinamički niz znakova više nije njegov; u svakom slučaju, njegov destruktor ne sme uništiti taj niz jer on pripada drugom objektu (premešten je):

```
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 (string&& s) : string() { move(s); }
  string& operator= (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; }

  void move (string&& s) { str = s.str; s.str = nullptr; }
  …
};

inline string& string::operator= (string&& s) {
  if (this!=&s) {
    release(); move(s);
  }
  return *this;
}
```