Korisnički definisane konverzije

- Standardne konverzije mogu da se rade implicitno, i to tranzitivno (u nizu), pa jedna korisnički definisana konverzija iz tipa *X* u ugrađeni tip može da znači i implicitnu konverziju u neki drugi ugrađeni tip; ako se ovo želi sprečiti, ta druga konverzija deklariše se posebno, čime ona nadjačava tranzitivne konverzije, ali se označi kao *obrisana* (*deleted*), specifikatorom *=delete* iza zagrada, pa je prevodilac neće dozvoliti. Ovo može da spreči neke nepredviđene upotrebe objekata tipa *X*, na primer kada se želi upotreba tih objekata kao Bulovih vrednosti (tzv. "problem sigurnog tipa *bool*", *safe bool problem*):

```
class Assertion {
public:

  operator bool ();
  operator int () = delete;

};

void f (Assertion x) {

  x << 1;

  if (x) …

}
```

Greška u prevođenju: ne postoji deklarisan operator << za ovu klasu, dok je operator implicitne konverzije u tip *int* deklarisan kao obrisan. Da ovaj operator nije tako deklarisan, prevodilac bi izvršio implicitnu, korisnički definisanu konverziju iz tipa *Assertion* u tip *bool*, pa potom i implicitnu, standardnu konverziju iz tipa *bool* u tip *int* i ovakav kod bez smisla bi bio moguć

Implicitna konverzija iz *Assertion* u *bool* je dozvoljena: objekti ove klase su namenjeni za korišćenje kao Bulovi izrazi, pa je zato definisana ova implicitna konverzija u *bool*

- Slično se može postići i deklarisanjem operatora konverzije u tip *bool* kao *explicit*, jer naredba *if* i naredbe petlji dozvoljavaju korisnički definisanu konverziju koja je eksplicitna:

```
class Assertion {
public:

  explicit operator bool ();
};

void f (Assertion x) {

  x << 1;

  if (x) …

}
```

Greška u prevođenju: nije dozvoljena implicitna konverzija iz tipa *Assertion* u tip *int*, čak ni tranzitivno, jer konverzija iz *Assertion* u *bool* nije implicitna

Ovo je u redu, jer je definisana konverzija iz tipa *Assertion* u tip *bool*

Korisnički definisani literali

- Kada neki apstraktni tip podataka implementira neku fizičku veličinu, numeričke vrednosti te veličine u programu ne mogu da sadrže informaciju o jedinici u kojoj je izražena ta vrednost. Takav program može biti osetljiv na greške ukoliko se na različitim mestima koriste vrednosti, date kao literali, izražene u različitim jedinicama mere. Na primer:

```
class Pressure {
public:

  Pressure (long double); // Pressure in Pa
  …

};

Pressure atm = 10.e5;

Pressure p0 = 1.0;
```

- Za ovakve potrebe, jezik C++ omogućava upotrebu *korisnički definisanih literala* (*user-defined literals*), za koje se mogu definisati *literalski operatori* (*literal operator*), kao operatorske funkcije koje pretvaraju korisničke literale u neki tip:

```
class Pressure {
private:

  friend Pressure operator"" _pa (long double);
  friend Pressure operator"" _bar (long double);
  Pressure (long double);
  …

};

Pressure operator"" _pa (long double val) {
  return Pressure(val);
}

Pressure operator"" _bar (long double val) {
  return Pressure(val*10.e5);
}

Pressure atm = 10.e5_pa;

Pressure p0 = 1.0_bar;
```