Klasni tipovi

- Na jeziku C++, klasni tipovi su *strukture* (*struct*) i *klase* (*class*). Strukture i klase su skoro potpuno izjednačene na jeziku C++, jer se tretiraju na potpuno identičan način (osim dole navedenih izuzetaka):
  - i strukture i klase mogu imati podatke članove i funkcije članice, uključujući i konstruktore, destruktore i operatorske funkcije
  - i strukture i klase mogu imati javne, zaštićene i privatne članove
  - i jedne i druge mogu se izvoditi, mogu imati polimorfne operacije itd.
- Jedine razlike između strukture i klase su sledeće:
  - ako se u definiciji strukture ne navede specifikator prava pristupa, podrazumeva se *public*; ako se u definiciji klase ne navede specifikator prava pristupa, podrazumeva se *private*
  - ako se u definiciji klase ili strukture, prilikom izvođenja iz druge klase ili strukture (dozvoljeno je sve), za osnovnu strukturu ne navede specifikator prava pristupa, podrazumeva se *public*, dok se za osnovnu klasu podrazumeva *private*
- Ovo su, ipak, krajnje sporedne razlike i ne treba se na njih oslanjati, jer to može da učini program slabije razumljivim: svakako je bolje uvek navoditi specifikatore prava pristupa eksplicitno, radi razumljivosti
- Zbog svega ovoga, strukture (*struct*) se koriste samo u izuzetnim situacijama, kada treba predstaviti sasvim jednostavne apstraktne tipove podataka, po pravilu onda kada se oni koriste samo za implementaciju nekih drugih struktura ili apstrakcija; strukture tada po pravilu imaju samo podatke članove i eventualno konstruktore, retko kada i neke jednostavne operacije ili destruktore
- U svim drugim slučajevima, posebno kada je potrebno da imaju iole složenije operacije ili predstavljaju apstrakciju, treba koristiti klase

Konstantni tipovi i funkcije članice

- Svaki objektni tip (tj. tip koji nije referenca ili funkcija) može da bude kvalifikovan kao *konstantan* (*constant*). Objekti ovakvog tipa ne mogu se menjati: neposredan pokušaj operacije nad ovakvim objektom koja bi promenila taj objekat prevodilac prijavljuje kao grešku, dok indirektan pokušaj izmene takvog objekta, npr. preko pokazivača ili reference na nekonstantan objekat rezultuje nedefinisanim ponašanjem (može izazvati čak i izuzetak od hardvera ako je konstantan objekat smešten u deo memorije zabranjen za upis)
- Konstantni objekti moraju biti inicijalizovani u definiciji, jer kasnije svakako ne mogu da se promene. Na primer:

```
const float pi = 3.14;
const char plus = '+';
pi++;
plus = '-';
```

- Konstantni objekti fundamentalnih tipova mogu da se koriste u konstantnim izrazima koje prevodilac treba da izračuna tokom prevođenja, na primer, u izrazima koje definišu dimenzije nizova:

```
const int MaxNumOfClocks = 100;
…

Clock* clocks[MaxNumOfClocks];
```

- Umesto korišćenja makro zamene *#define*, bolje je koristiti konstantne objekte, jer oni imaju svoj tip koji može biti različit od tipa literala kojim se makro u direktivi *#define* zamenjuje; u određenim situacijama, prevodilac ne mora ni da alocira konstantan objekat za vreme izvršavanja, već da njegovu konstantnu vrednost potpuno iskoristi za vreme prevođenja na svim mestima njenog korišćenja