Deklaracija i definicija funkcije

- Povratni tip funkcije ne može biti funkcija ili niz (ali može biti pokazivač ili referenca na funkciju ili niz)
- Povratni tip funkcije može da se navede i iza parametara i znaka ->, što olakšava pisanje i čitanje deklaracija ako je povratni tip složen, ili ako se ne može odrediti, npr. zato što zavisi od tipova argumenata unutar šablona:

```
auto redirect (int(*)(int)) -> int(*)(int(*)(int));
```

```
template <typename U, typename V>
auto combine (U u, V v) -> decltype(u+v);
```

Ovo bi bilo prilično teško napisati klasičnom notacijom

- Povratni tip ne mora da se navodi eksplicitno, nego da se ostavi prevodiocu da ga sam izvede, na osnovu tipa izraza iza naredbe *return*; svi tipovi iza višestrukih naredbi *return* moraju biti konzistentni; ova mogućnost nije dozvoljena za virtuelne funkcije:

Povratni tip ove funkcije je *decltype(u*scalar1 + v*scalar2)*

```
template <typename U, typename V>
auto combine (U u, V v, double scalar1, double scalar2) {
  return u*scalar1 + v*scalar2;
}
```

Inline funkcije

- Kada se u složenom programu temeljno sprovedu svi elementi objektne dekompozicije, tipična situacija jeste ta da većina metoda (tela fukcija) ima vrlo malo linija koda, vrlo retko više od 5-10 linija; sve preko toga može biti signal da je propuštena prilika za apstrakcijom ili dekompozicijom (makar algoritamskom). Duža tela funkcija svakako nisu preporučljiva, jer smanjuju razumljivost
- Tipični ekstremni primeri su metode koje rade vrlo proste operacije, na primer:
  - samo vraćaju ili postavljaju vrednost atributa (*getter* i *setter* operacije):

```
string Person::getName () const { return this->name; }
Person& Person::setName (const string& newName) { this->name = newName; return *this; }
```

  - predstavljaju "omotače" (*wrapper*) oko neke druge operacije, sa ciljem enkapsulacije:

```
Clock* Clock::create (…) { return new Clock(…); }
```

  - delegiraju poziv jednoj ili nekim drugim operacijama, uz eventualne konverzije argumenata, kao posedica dekompozicije i lokalizacije zajedničkih delova, odnosno svođenja na već postojeće:

```
Clock::Clock (int hh, int mm, int ss) { setTime(hh,mm,ss); }
bool operator!= (const complex& c1, const complex& c2) { return !(c1==c2); }
```

- Nije neobičan utisak koji se može steći o dobro dekomponovanom programu, da na neki način "većina metoda ne radi ništa posebno, već sve prepušta drugima, samo delegira pozive drugima", a ceo program ipak radi složen posao!
- Ovo je posledica dobre dekompozicije, ali i puno implicitne semantike koja je sadržana u konstruktima jezika, kao što su npr. polimorfni pozivi, pozivi konstruktora osnovnih klasa, konverzije i slično