# Klasa kao realizacija apstraktnog tipa podataka

- Prenos objekata kao argumenata poziva funkcija po vrednosti, kopiranjem, se rešava na isti način:

```
complex complex::add (complex ca, complex cb);
```

```
...complex::add(c3,c4)...
```

- Ponovo se formalni argument *ca* (*cb*) inicijalizuje stvarnim argumentom *c3* (*c4*) pozivom konstruktora kopije, sa istom semantikom:

```
complex ca = c3;
complex cb = c4;
```

Poziva se konstruktor kopije `complex::complex(complex& other)` za *ca*, pri čemu se referenca - formalni argument *other* vezuje za stvarni argument *c3*

- I povratna vrednost funkcije tj. rezultat poziva funkcije se vrši po vrednosti, takođe kopiranjem, na isti način
- Na mestu poziva funkcije, u trenutku povratka iz funkcije, kreira se *privremeni* (*temporary*) objekat koji se inicijalizuje izrazom iz naredbe *return*; semantika ove inicijalizacije je ista kao i svake druge inicijalizacije:

```
complex complex::add (complex ca, complex cb) {
  ...
  return result;
}
```

```
...complex::add(c1,c2)...
```

Poziv negde u izrazu; kao da je tu kreiran privremeni, bezimeni objekat *temp* tipa *complex* koji prihvata vraćenu vrednost i inicijalizovan sa *result*:
`complex temp = result;`

Rezultat poziva *complex::add* jeste jedan bezimeni, privremeni objekat koji se kreira na mestu poziva ove funkcije, u trenutku povratka iz nje, i inicijalizuje rezultatom izraza iza naredbe *return* - objektom *result*. Poziva se konstruktor kopije, sa sledećim značenjem:

```
complex temp = result;
```

Poziva se konstruktor kopije `complex::complex(complex& other)` za *temp*, pri čemu se referenca - formalni argument *other* vezuje za stvarni argument *result*

# Klasa kao realizacija apstraktnog tipa podataka

- Sledeće pitanje: ako želimo da instance korisničkih tipova koristimo kao i instance ugrađenih tipova, zašto i notacija operacija, recimo za ovakve, matematičke tipove, ne bi bila ista?
- Umesto:

...`complex::sub(complex::add(c3,c4),c5)`...

zašto ne bismo pisali prosto ovako:

...`c3+c4-c5`...

- U skladu sa opredeljenjem na ovakvo korišćenje instanci korisničkih tipova, jezik C++ omogućava i ovo - *preklapanje operatora* (*operator overloading*)
- Umesto "klasičnih" identifikatora, funkcije mogu imati i posebno ime, *operator@*, gde je @ simbol nekog operatora ugrađenog u jezik:

```
class complex {
public:
  complex (double real, double imag);
  complex (complex& other);  // Copy constructor

  friend complex operator+ (complex c1, complex c2);
  friend complex operator- (complex c1, complex c2);

private:
  double re, im;
};

complex::complex (double real, double imag) {…}

complex operator+ (complex c1, complex c2) {
  return complex(c1.re+c2.re,c1.im+c2.im);
}

…
```