Privremeni objekti

- Privremeni objekti su anonimni (bezimeni) objekti klasa čiji je životni vek najčešće kratak; oni nastaju implicitno, u određenim situacijama, i nestaju implicitno, pod kontrolom prevodioca
- Osnovna potreba za privremenim objektom jeste u tome da se rezultat nekog izraza, uključujući i poziv funkcije (i operatorske funkcije) može dalje iskoristiti kao operand neke druge operacije (ili argument funkcije)
- Bez obzira na to kada se tačno privremeni objekat inicijalizuje i uništava, uvek se pri inicijalizaciji poziva njegov konstruktor, a pri uništavanju njegov destruktor
- Pre verzije jezika C++17, rezultat poziva funkcije (uključujući i operatorske) koja ne vraća referencu (na lvrednost), odnosno čiji je povratni tip klasa (a ne referenca na nju), bio je uvek privremeni objekat: u trenutku povratka iz funkcije, na mestu poziva funkcije, pravi se privremeni objekat koji se inicijalizuje izrazom iza naredbe *return*:

```
class X {...};

X f (X x) {
  return x;
}

int main () {
  X x1;

  ...f(x1)...

  ...f( f(x1) )...
}
```

Do C++17, rezultat ove funkcije u svakom pozivu je bezimeni, privremeni objekat koji se inicijalizuje izrazom iza naredbe *return* u trenutku povratka iz funkcije

Do C++17, rezultat ugnežđenog poziva *f* je privremeni objekat tipa *X* koji prihvata rezultat tog poziva; njime se inicijalizuje stvarni argument poziva okružujućeg poziva funkcije *f*, čiji rezultat ponovo poredstavlja privremeni objekat

Privremeni objekti

- Pre verzije C++17, privremeni objekti obavezno su se pravili i u sledećim situacijama, pored još nekih:
  - konverzija koja ne vraća lvrednost, uključujući i eksplicitan poziv konstruktora; na primer:

```
complex c1(3.0,0.0), c2(0.0,4.0), c3;
c3 = (c1 + c2) * (complex(1.,0.) + c2);
```

  - prilikom inicijalizacije kopiranjem (*copy initialization*, inicijalizacija sa znakom =):

```
comlex c4 = (c1 + c2) * (complex(1.,0.) + c2);
```

  - kada se referenca inicijalizuje izrazom drugog, ali konvertibilnog tipa

- Pritom, za navedeni drugi slučaj inicijalizacije kopiranjem, kao i na nekim drugim mestima, prevodiocu se dopušta (ali se ne obavezuje) optimizacija *izostavljanja kopiranja* (*copy elision*): umesto da se najpre napravi privremeni objekat kao rezultat inicijalizacionog izraza, a potom njime inicijalizuje deklarisani objekat konstruktorom kopije, prevodilac može odmah da inicijalizuje deklarisani objekat rezultatom inicijalizacionog izraza, izostavljajući kopiranje, čak i ako konstruktor kopije ima vidljive bočne efekte, ali samo pod uslovom da su sve odgovarajuće funkcije, uključujući i konstruktor kopije, dostupne na mestu deklaracije (proveravaju se prava pristupa kao da se kopiranje ne izostavlja, tzv. *as-if* pravilo)