Preporučeni načini preklapanja operatora

- Operator *[]* se mora preklopiti kao nestatička funkcija članica; ona može imati parametar bilo kog tipa; taj parametar prima vrednost izraza između uglastih zagrada (drugog operanda):

```
x[expression]
```

tumači se kao:

```
x.operator[](expression)
```

- Ovaj operator se tipično preklapa tako da postoje varijante koje omogućavaju i čitanje iz konstantnih i nekonstantnih objekata, kao i upis u nekonstantne objekte, odnosno varijante koje su konstantna i nekonstantna funkcija članica i koje vraćaju odgovarajuće lvrednosti (reference na konstantu i nekonstantu, respektivno):

```
class X {
  …
        T& operator[] (size_t);
  const T& operator[] (size_t) const;
  …
};
```

Preporučeni načini preklapanja operatora

- Operator *()* se preklapa za klase čiji se objekti mogu posmatrati kao tzv. *funkcijski objekti* (*function object*), tj. objekti koji se mogu koristiti kao funkcije, jer se na njih može primeniti operator poziva funkcije (kao za obične funkcije):

```
x(expression1, expression2)
```

tumači se kao:

```
x.operator()(expression1, expression2)
```

- Za razliku od običnih funkcija, funkcijski objekti, kao i svaki drugi objekti, mogu imati (i tipično imaju) svoje stanje, koje "nose" sa sobom (svaki objekat svoje nezavisno stanje) i mogu da ga menjaju i akumuliraju
- Ovakvi objekti se tipično prenose, slično kao i pokazivači na funkcije, kao argumenti nekih operacija koje sprovode određene postupke, odnosno algoritme, za koji su im potrebne implementacije elementarnih operacija-koraka tog algoritma; zato takav algoritam poziva nazad dostavljenu funkciju, odnosno funkcijski objekat (tzv. *callback* mehanizam), ili ga primenjuje na neke elemente koje obilazi tokom algoritma (npr. obilasci raznih struktura)
- Na primer, funkcija *for_each* iz standardne biblioteke iterira kroz kolekciju, dok kao treći parametar očekuje funkcijski objekat koga primenjuje na svaki posećeni element, dostavljajući mu taj posećeni element kao argument operatora *()*:

```
struct Sum {
    int sum;
    Sum () : sum(0) {}
    void operator() (int n) { sum += n; }
};

std::vector<int> v{...};

Sum s = std::for_each(v.begin(), v.end(), Sum());
```