Opseg važenja klase

- Kao desni operand operatora :: kom je levi operand ta klasa ili klasa izvedena iz te klase. Na primer, čest je slučaj da redefinisana operacija u izvedenoj klasi treba da uradi isto što i metoda osnovne kalse, ali i još nešto (u bilo kom redosledu); drugim rečima, da treba da pozove metodu iz osnovne klase:

```
class Base {
public:

  virtual void f();

};
```

```
class Derived : public Base {
public:

  virtual void f();

};


void Derived::f () {
  …
  Base::f();
  …
}
```

Poziv po nekvalifikovanom imenu *f* bi značio rekurzivni poziv iste ove funkcije, što nije korektno

- Operator ::, koji je inače asocijativan sleva nadesno, navodi da prevodilac vrši *kvalifikovanu potragu* (*qualified lookup*) na sledeći način: *Base::f* znači da se najpre sprovodi nekvalifikovana potraga za imenom *Base*; pronađe se to ime i njegova deklaracija, odnosno deklaracija klase *Base*; zatim se u oblasti važenja te klase traži deklaracija za ime *f*; ako se ne nađe takva deklaracija, traži se u njenoj osnovnoj klasi, ako je ima itd.

Opseg važenja klase

- Unutar definicije klase, odnosno u oblasti važenja klase mogu biti deklarisani drugi tipovi, uključujući i klase. Svi oni se nazivaju *članovima klase* (*class members*)
- Takve ugnežđene klase, osim što su u oblasti važenja okružujuće klase, nemaju nikakve druge posebne veze; takvo ugnežđivanje ne znači nikako ugrađivanje objekata, već samo logičko "pakovanje" klase
- Na primer, za implementaciju liste potrebna je struktura koja predstavlja jedan element liste i skladišti pokazivač(e) na sledeći (i prethodni) takav element liste. Ovakva struktura, dakle, bitna je samo za klasu liste i ni za koga drugog. Štaviše, bitna je samo za njenu implementaciju, pa je logično da bude definisana unutar klase liste, i to kao privatna, da ne bi bila deo njenog interfejsa:

```
template<typename T>
class List {
public:
  List ();

  void put (T);
  …

private:

  struct ListElement {
    ListElement (const ListElement* next);
    …

  };

};

void List::put (T t) { … }

List::ListElement::ListElement (const List::ListElement* next) { … }
```