# Nizovi

- Dve potpuno nezavisne implicitne konverzije mogu da dovedu do problema ukoliko se nizovi objekata ne koriste na korektan način:
  - konverzija *T[]* u *T**, koja potiče iz jezika C sa ciljem efikasnosti
  - konverzija pokazivača na izvedenu u pokazivač na dostupnu osnovnu klasu (*Derived** u *Base**), koja je uvedena u jezik C++ kao podrška principu supstitucije
- Ove dve konverzije se mogu raditi i povezano, obe, u lancu implicitnih konverzija, i mogu da dovedu do sledećeg problema: funkcija koja kao parametar ima niz objekata osnovne klase (*Base[]*, odnosno *Base**), može se pozvati sa argumentom koji je niz objekata izvedene klase (*Derived[]*):

*Derived[]* ⇒ *Derived** ⇒ *Base**

```
class Base {
  public: int bi;
};

class Derived : public Base {
  public: int di;
};

void f (Base b[]) { cout<<b[2].bi; }

int main () {
  Derived d[5];
  d[2].bi=77;

  f(d);
}
```

Objekti klase *Base* imaju samo jedan podatak član tipa *int*, pa su veličine jednog *int*

Objekti klase *Derived* su veličine dva *int*

Funkcija *f* očekuje niz objekata klase *Base*, veličine jednog *int*: *b[2]* je **(b+2)*, a pošto je *b* tipa *B[]*, pokazivačka aritmetika dodaje vrednost *2*sizeof(B)*

*d* se konvertuje iz *Derived[]* u *Derived**, a potom u *Base**, pa je poziv ispravan

# Nizovi

- Osim toga, kada se objekti, pa i nizovi objekata, koriste po vrednosti, nema nikakve supstitucije:

```
Base b[5];
Derived d;
b[0]=d;
```

- Zbog ovoga, ukoliko se računa sa nasleđivanjem i supstitucijom, ne treba praviti nizove objekata (po vrednosti), već nizove pokazivača na objekte:

```
void f (Base* b[], int i) { cout<<b[i]->bi; }
…
Base b1,b2; Derived d1,d2,d3; d2.bi=77;
Base* b[5]; b[0]=&d1; b[1]=&b1; b[2]=&d2; ...
f(b,2);
```

- Sada navedena greška nije više moguća, jer se niz tipa *Derived*[]* može konvertovati u *Derived***, ali se to ne može konvertovati implicitno u *Base*** (što bi funkcija sa parametrom tipa *Base*[]* prihvatila):

```
Derived* d[5]; d[0]=&d1; d[1]=&d2; …
f(d,2);
```