Sa proceduralnog na OO programiranje: polimorfizam

- Nešto pregledniji pristup — razdvojiti obradu svake situacije u zaseban potprogram:

```
int canMoveTo (Figure* fig, unsigned col, unsigned row) {
  switch (fig->kind) {

    case pawn: return canPawnMoveTo(fig,col,row);
    case bishop: return canBishopMoveTo(fig,col,row);
    …
  }

int canPawnMoveTo (Figure* fig, unsigned col, unsigned row) {
     if (fig->color==white) …
     else …
}

int canBishopMoveTo (Figure* fig, unsigned col, unsigned row) {
  …
}

…
```

- I dalje moramo menjati *switch* u slučaju da dodajemo novu vrstu objekata, i to na svim ovakvim mestima (polimorfnim operacijama)

Sa proceduralnog na OO programiranje: polimorfizam

- Ideja — osloniti se na strukture podataka i na (dinamičko) ulančavanje pokazivača i pokazivače na funkcije, a onda i na dinamičko vezivanje:

```
// Table of pointers to implementations
// of virtual functions (“virtual table”)

struct Figure_VFTable {
  int (*canMoveTo) (Figure* fig, unsigned col, unsigne
  int (*display) (Figure* fig, …);
  …
};

int canPawnMoveTo (Figure* fig, unsigned col, unsigned row);
int canBishopMoveTo (Figure* fig, unsigned col, unsigned row);
…

Figure_VFTable pawnVFTable;
pawnVFTable.canMoveTo = &canPawnMoveTo;
pawnVFTable.display = &displayPawn;
…

Figure_VFTable bishopVFTable;
bishopVFTable.canMoveTo = &canBishopMoveTo;
bishopVFTable.display = &displayBishop;
…
```