Sa proceduralnog na OO programiranje: polimorfizam

- Prema tome, efikasnost poziva virtuelne funkcije na jeziku C++ je ista kao i u ekvivalentnom C kodu sa dinamičkim vezivanjem, i tek nešto malo manja nego poziv statičkim vezivanjem (skok na adresu memorijski direktnim adresiranjem), ali je ceo mehanizam sakriven od programera i program je apstraktniji, kompaktniji i lakši za razumevanje i održavanje:

```
struct Figure_VFTable {
  int (*canMoveTo) (Figure* fig,…);
  int (*display) (Figure* fig, …);
  …
};

int canPawnMoveTo (Figure* fig,…);
int canBishopMoveTo (Figure* fig,…);
…

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

struct Figure {
  Figure_VFTable* vtp;
  FigureKind kind;
  …
};

void initPawn (Figure* fig,…) {
  fig->vtp = &pawnVFTable;
  fig->kind = pawn;
  …
}

int canMoveTo (Figure* fig,…) {
  return fig->vtp->canMoveTo(fig,col,row);
}
```

```
class Figure {
public:
  Figure ();
  virtual int canMoveTo (…);
  virtual int display (…);
  …
};

class Pawn : public Figure {
public:
  Pawn ();
  virtual int canMoveTo (…);
  virtual int display (…);
  …
};

…

…aFig->canMoveTo(…)…
```

Klasa kao realizacija apstraktnog tipa podataka

- Želimo da realizujemo *apstraktni tip podataka* (*abstract data type*), kompleksan broj: strukturu sa pridruženim operacijama koja predstavlja *korisnički definisani tip* (*user-defined type*) - tip koji ne postoji ugrađen u jezik (*built-in type*), već ga definiše programer (kao korisnik jezika)
- Možemo kao i ranije, na jeziku C:

```
struct _complex {
  double re, im;
};
typedef struct _complex complex;

void complex_init (complex* this, double real, double imag);
complex complex_add (complex c1, complex c2);
complex complex_sub (complex c1, complex c2);
…

void complex_init (complex* this, double real, double imag) {
  this->re = real; this->im = imag;
}

complex complex_add (complex c1, complex c2) {
  complex result;
  result.re = c1.re + c2.re;
  result.im = c1.im + c2.im;
  return result;
}
…
```