Pokazivači na objekte

- Na primer:

```
int i=0, j=0;
int* pi = &i;
*pi = 1;
pi = &j;
*pi = 2;
```

pi ukazuje na *i*

i dobija vrednost *1*

pi sada ukazuje na *j*

j dobija vrednost *2*

- Pošto su i pokazivači i nizovi objekti, pokazivač može ukazivati i na njih:

```
int i=0, j=0;
int* pi=&i;
int** ppi;
ppi=&pi;
*pi=1;
**ppi=2;
*ppi=&j;
ppi=&i;

int a[10];
int (*pa)[10] = &a;
```

ppi je tipa *pokazivač na - pokazivač na - int*

ppi ukazuje na *pi*

i dobija vrednost *1*

Izračunava se: **(*ppi); i* dobija vrednost *2*

pi sada ukazuje na *j*

Greška u prevođenju: *ppi* je tipa *int***, a *&i* je tipa *int**; ne postoji ova implicitna konv.

pa je tipa *pokazivač na - niz od 10 elemenata tipa - int* i ukazuje na niz *a* koji je baš takvog tipa

- Ako pokazivač ukazuje na objekat klasnog tipa, može se pristupiti članu tog objekta preko operatora ->; ukoliko se radi o objektu polimorfne klase (sa bar jednom virtuelnom funkcijom), aktivira se polimorfizam:

```
Clock* pc = new ClockWithDates;
pc->tick();
(*pc).tick();
```

Ovo je isto

Pokazivači na objekte

- Pokazivač tipa *void** može da ukazuje na bilo koji objekat: postoji implicitna konverzija iz bilo kog pokazivačkog tipa u tip *void**; sa objektom na koji ukazuje ovaj pokazivač ne može se raditi ništa
- Ovakav pokazivač se vrlo ograničeno upotrebljava, tipično u starim C interfejsima, gde se prihvata pokazivač na bilo šta; na primer, nit (*thread*) u biblioteci POSIX kreira se zadavanjem argumenta funkcije koji je pokazivač tipa *void**
- Da bi se preko tog pokazivača uradilo nešto sa onim na šta on ukazuje, mora se izvršiti eksplicitna konverzija na odredišni tip, s tim da odgovornost za ispravnost te konverzije nosi programer:

```
void DocumentSaver::run (void* p) {
  DocumentSaver* ds = static_cast<DocumentSaver*>(p);
  if (ds) ds->save();
  delete ds;
}
```