Vaje 23

Simona Perčič

4.5.

Polimorfizem

Predmeti iz razreda skladov imajo lastnost, da lahko na njih:

- nalagamo predmete in
- z njih zlagamo predmete.

Primeri so:

- razred skladov celih števil:
  - `Nalozi(int elem)` naloži element na sklad
  - `int Zlozi()` vzame zgornji element iz sklada in ga vrne
- razred skladov racionalnih števil:
  - `Nalozi(float elem)` naloži element na sklad
  - `float Zlozi()` vzame zgornji element iz sklada in ga vrne
- razred skladov znakov:
  - `Nalozi(char elem)` naloži element na sklad
  - `char Zlozi()` vzame zgornji element iz sklada in ga vrne
- razred skladov elementov tipa `tSklElt`:
  - `Nalozi(tSklElt elem)` naloži element na sklad
  - `tSklElt Zlozi()` vzame zgornji element iz sklada in ga vrne

Ogledali smo si iste operacije, vendar za sklade predmetov različnimi tipov. Sedaj bomo uporabili princip polimorfizma in sicer tako, da se bomo znebili skladov predmetov različnih tipov (tudi vseh datotek) in jih nadomestili z enim samim, ki bo definiran za *posplošen tip* – *večobličen* ali *polimorfen tip*.

Ad-hoc polimorfizem

Najprej bomo uporabili polimorfizem, ki se imenuje *ad-hoc polimorfizem* ali drugače – za silo narejen polimorfizem.

Razlika med `int`, `float`, ... polimorfizmi je v velikosti obsega in tako prostora v pomnilniku, ki ga zaseda en predmet posameznega razreda (tipa). Če vemo kje je element in kako velik je, lahko naredimo izvedbeni modul neodvisno od `int`, `float`, ...

Izhajali bomo iz razreda skladov celih števil `SKLAD_NO.h` in opisali potrebne spremembe na njem. Najprej kreatorju dodamo parameter, ki pove, koliko prostora zavzame en element (predmet) naložen na sklad.

Zakaj imamo sedaj dva kreatorja? (ponovitev)

- S kreatorjem `Sklad()` še ne vemo koliko prostora lahko zasedemo za potrebe sklada.
- S kreatorjem `Sklad(int min_vel_sklad)` določimo koliko prostora najmanj bo zasedel sklad.

Ta dva kreatorja sedaj razširimo na:

- `Sklad(velElt)` in
- `Sklad(velElt, min_vel_sklad)`

kjer `velElt` opisuje koliko prostora zasede en predmet naložen na sklad. To velikost dodamo med privatne stvari razreda skladov.

Poleg tega spremenimo še ostale funkcije:

- `Nalozi(naslov)`: ker je velikost za vse predmete, ki jih nalagamo na sklad enaka, potrebujemo za izvedbo nalaganja samo še podatek o mestu, kjer se nahaja nalagani predmet – njegov *naslov*.
- `Zlozi(naslov_zlaganja)`: podobno za zlaganje potrebujemo samo prostor, kam naj zložimo predmet – potrebujemo *referenco* predmeta v katerega bomo prepisovali.
- `Vrh(naslov_prepisovanja)`: isto velja za vrh.
- Funkciji `Prazen` in `Poln` sta nespremenjeni.

V izvedbenem modulu sta spremenjene funkcije za nalaganje, zlaganje in vračanje vrha. Za razliko od starega nalaganja, kjer je bil predmet prenešen po vrednosti in smo ga lahko v enem kosu naložili na sklad, moramo pri novem nalaganju to početi po koščkih. En košček je veliko kot en znak (`char`), njihovo število pa je določeno ob kreiranju – *velElt*:

```
for(i= 0; i < velElt; i++)
  ptr_skladisce[zadnji][i]=element[i];
```

Sedaj lahko uporabljamo naš *ad-hoc* polimorfni razred skladov predmetov na naslednji način:

```
int main() {
  Sklad s1(sizeof(int));      // kliče: Sklad(velElt)
  Sklad s2(sizeof(int), 3);   // kliče: Sklad(velElt, min_vel_sklad)
  Sklad s3(sizeof(float));
  Sklad s4(sizeof(Crac));
  int i;
  float f;
  Crac r;

  s1.Nalozi(&i);
  s2.Nalozi(&i);
  s3.Nalozi(&f);
  s4.Nalozi(&r);

  return 0;
}; // main
```

Žal po nesreči lahko napišemo tudi:

```
  s4.Nalozi(&i);
```

in naš program bo deloval napačno. Formalno gledano smo kršili pogodbo, saj predpogoj zahteva, da je sklad `s4` sklad racionalnih števil in ne celih.

Polimorfizem z vzorci / kalupi – *templates*

Kako delujejo kalupi(vzorci)? Ko naredimo funkcijo, razred ali kaj podobnega, lahko rečemo, da bo to le vzorec (kalup), ki bo sicer kot tak neuporaben in bo postal resnična funkcija (razred, ...) šele, ko bomo določili dodatne parametre.

Sintaksa definicije vzorca je:

`template` < template-argument-list > declaration

primer:

`template` < `type tSklElt` > `classSklad`...

Za naše potrebe, kjer bomo definirali vzorec razreda skladov predmetov iz nekega drugega razreda, boma namesto `type` zapisali `class` – parameter našega vzorca je pač *razred* predmetov, ki so naloženi na sklad:

```
template <class tSklElt> Sklad { ... };
```

Rezervirana besedica *template* (kalup, vzorec) nakazuje, da definiramo kalupni razred `Sklad`. Ta kalup ima kot parameter razred `tSklElt` – to je razred predmetov naloženih na sklad. Njegova deklaracija seže **samo do konca deklaracijskega kalupa razreda**. Ko je konec deklaracije razreda, prevajalnik razreda `tSklElt` ne pozna več!

Seveda takšen kalup razreda ni takoj uporaben, saj ne ve, kaj `tSklElt` sploh predstavlja. Zato moramo pred uporabo povedati s kakšen razred (tip) predstavlja `tSklElt`:

```
Sklad <int>.
```

S tem iz kalupa naredimo razred skladov celih števil. Kako izgleda celoten program si lahko ogledate na:

`Z:\PREDAVANJA\POLDE\code\Sklad\Skladi.h`

To ni samo definicijski modul, ampak vključuje tudi izvedbeni modul.

Primer uporabe:

```
int main() {
  Sklad<int>   s1;
  Sklad<int>   s2(3);
  Sklad<float> s3;
  .
  .
  .
};
```

Ada

V programski jeziku ada, se vzorci imenujejo *generični tipi*.

Programski jezik ada nosi ime po *Lady Ada Lovelace*, nečakinji Lord Byrona. Ada je bila prva programerka v zgodovini in sicer je programirala prvi ,,resnični" računalnik. Le tega je okoli leta 1850 naredil *Charles Babbage* ter ga imenoval *analitični stroj*.