Sveučilište u Zagrebu
PMF – Matematički odjel

Objektno programiranje (C++)

Vježbe 04 – Međuklase i optimizacije nekih operatora

Vinko Petričević

- Ako imamo klasu s funkcijom koja vraća referencu:

```
class klasa {
    int neko_svojstvo;
public:
    int& svojstvo() { return neko_svojstvo; }
};
```

- referenca nam je potrebna da bi mogli i mijenjati vrijednost svojstva

```
klasa k;
int x = k.svojstvo();
k.svojstvo() = 10;
```

- želimo napraviti razliku u očitavanju i mijenjanju svojstva

Međuklasa

- umjesto int& neka funkcija vraća novu klasu koja će se brinuti za čitanje/pisanje svojstva

```
class klasa {
     int neko_svojstvo;
public:
     struct pomocna {
            klasa *pThis;
            pomocna(klasa *p) : pThis(p) {}
            operator int() {
                   cout << "svojstvo se cita" << endl;
                   return pThis->neko_svojstvo;
            }
            int& operator=(int x) {
                   cout << "svojstvo se pise" << endl;
                   return pThis->neko_svojstvo = x;
            }
     };
     pomocna svojstvo() { return pomocna(this); }
//   int& svojstvo() { return neko_svojstvo; }
};
```

Međuklasa

- sada klasa može razlikovati kada se svojstvo mijenja, a kada se samo očitava, je se na kôdu

```
int x = k.svojstvo();
```

u stvari izvrši:

```
klasa::pomocna temp(k);
int x = (int)temp;
```

a umjesto

```
k.svojstvo() = 10;
```

se izvrši:

```
klasa::pomocna temp(k);
temp.operator=(10);
```

Zadatak

- koristeći međuklasu, poboljšajte rad s mapom, tako da ako imamo npr.

```
map<string, int> m;

int x = m[“123”];
// vrati 0 i ne ubacuje novi element u mapu

m[“321”] = 10;
// ubaci novi par u mapu (“321”, 10)
```

Zadatak - pojašnjenje

- sljedeći kôd ispiše 2, a trebao bi ispisati 1

```
#include <iostream>
#include <map>
#include <string>
using namespace std;

class MAP : public map<string, int> {
public:
};
```

-
```
void main() {
    MAP m;
    int x = m["123"]; // ubaci se m["123"]=0;
    m["321"] = 10;
    cout << m.size() << endl;
}
```

Zadatak - pojašnjenje

- trebalo bi preraditi ljedeći operator. Ovo je ono što on po defaultu napravi:

```
 class MAP : public map<string, int> {
public:
   int& operator[](const string& k) {
        map<string, int> &m = *this;
//castanje u bazni tip je ubačeno da
se ne bi operator[] pozvao ponovo na
klasi MAP, pa bi ostali u beskonačnoj
petlji

        return m[k];
   }
};
```

Zadatak - pojašnjenje

- umjesto takvog operatora koji vraća int&, treba napraviti međuklasu koja će pamtiti kojoj mapi i o kojem ključu se radi, te da ta međuklasa onda izvršava očitavanje/pridruživanje vrijednosti

Zadatak - rješenje

```
class MAP : public map<string, int> {
public:
     struct pom {
            map<string,int>& mThis;
            const string &indeks;
            pom(map<string, int> *pThis, const string& index) :
 mThis(*pThis), index(index) {}

            operator int() {
               map<string,int>::iterator f = mThis.find(index);
               if (f == mThis.end()) return 0;
               return f->second;
            }

            int& operator=(int x) {
                   return mThis[index] = x;
            }
     };

     pom operator[](const string& k) {
            return pom(this, k);
     }
};
```

Zadatak

```
template<class K, class V> class MAP : public map<K, V> {
public:
     struct pom {
            map<K,V>& mThis;
            const K &indeks;
            pom(map<K, V> *pThis, const K& index) : mThis(*pThis),
index(index) {}

            operator V() {
                  map<K,V>::iterator f = mThis.find(index);
                  if (f == mThis.end()) return V();//def.konst.
                  return f->second;
            }

            V& operator=(const V& x) {
                  return mThis[index] = x;
            }
     };

     pom operator[](const K& k) {
            return pom(this, k);
     }
};
```

Zadatak

- Pretpostavimo da imamo strukturu broj koja predstavlja niz znamenaka velike duljine i C-ovsku funkciju
```
void zbroj(broj* rez, const broj* l, const broj* d);
```
koja računa zbroj brojeva l i d i rezultat sprema u rez;
- želimo li napraviti klasu Broj, koja će također pamtiti niz znamenaka kao broj, a da na njoj imamo operator+, ona bi izgledala otprilike:

```
struct Broj{
    broj* b;

    Broj operator+(const broj& d) const {
        Broj ret;
        zbroj(ret.b, this->b, d.b);
        return ret;
    }
};
```

Zadatak

- Ako bi dobro napravili kontrolu kopiranja za klasu, imali bi nekoliko nepotrebnih kopiranja prilikom operacije oblika:

```
Broj a, b, c;
c = a+b;
```

- Optimizirajte operacije zbrajanja, tako da je rezultat međuklasa (koja pamti oba operanda zbrajanja), te da se tek prilikom pridruživanja vrši stvarni izračun zbroja. Na taj način ćemo izbaciti nepotrebna kopiranja objekata

Zadatak

- Uz pretpostavku da imamo i C-ovsku funkciju
  ```
  void oduzmi(broj* rez, const broj* l, const broj* d);
  ```
  koja računa razliku brojeva l i d i rezultat sprema u rez.
  optimizirajte i oduzimanje

Zadatak

- Budući da je kôd i novoubačena međuklasa za zbrajanje vrlo slična onoj za oduzimanje, pokušajte parametrizirati sve takve binarne operacija/i njihove međuklase