ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ
*Одсек за СИ* Београд 27.11.2015.

## Други колоквијум из
# Објектно оријентисаног програмирања I

**1)** *(30 поена)* Одговорити концизно (по једна или две реченице) и прецизно на следећа питања:

**а)** Колико операнада и у ком случају може да има преклопљени оператор `operator-`?

**б)** Да ли је дозовољено писати и зашто:
`class A{}; class B: protected A{}; int main(){A&r=*(new B());}` ?

**в)** Чему служи апстрактни деструктор, да ли се декларише, да ли се дефинише, како и зашо?

**2)** *(укупно 70 поена)* Написати на језику *C++* следеће класе (класе опремити оним конструкторима, деструктором и операторима доделе који су потребни за безбедно и ефикасно коришћење класа):

- *(10 поена)* ***Датум*** се задаје помоћу редног броја дана, месеца и године (подразумевано 27.11.2015, исправност не треба проверавати). Може да се одреди да ли је датум после другог датума (`datum1>datum2`) и да се испише на главном излазу у облику *дан.месец.година*..
- *(30 поена)* Апстрактна ***полиса осигурања*** има јединствен, аутоматски генерисан целобројан идентификатор (број полисе), датум почетка важења, датум истека, податак да ли је активна и тип. Тип полисе може бити `STANDARD`, `KOMFORT` и `EKSKLUZIV` (подразумевано `STANDARD`). При стварању, полиса је активна. Може да се утврди да ли је активна и дохвати датум истека, као и да се полиса означи неактивном. Важење полисе је могуће продужити задавањем новог датума истека (неактивна полиса постаје активна, а продужење се игнорише ако је нови датум истека пре текућег). Може да се дохвати максимална премија при реализацији полисе и да се полиса упише у излазни ток (`it<<polisa`) у облику *ид* **-** *тип* **[***премија***eur][***почетак***-***крај***].**
- Максимална премија при реализацији ***полисе путног осигурања*** је 5000 eyp, 7000 eyp и 10000 eyp, за типове `STANDARD`, `KOMFORT` и `EKSKLUZIV`, респективно. Полиса путног осигурања може да се упише у излазни ток у облику **`Putno_osiguranje:`** *полиса*.
- *(20 поена)* ***Осигураник*** има име и јединствен, аутоматски генерисан, целобројан идентификатор осигураника. Садржи произвољан број полиса чији је власник. Ствара се задавањем имена, без полиса осигурања, након чега се полисе додају појединачно (`osiguranik+=polisa`). Могуће је означити неактивним све полисе осигураника чији је датум истека пре задатог датума, и продужити важење до задатог датума свих активних полиса осигураника. Осигураник не може да се копира ни премешта, ни иницијализацијом ни доделом. Може да се упише у излазни ток (`it<<osiguranik`) у облику *име* **(***ид***)**, након чега се у пару витичастих заграда (**{ }**) уписују полисе осигураника, свака у засебном реду.

*(10 поена)* Написати на језику *C++* **програм** који направи осигураника, дода му неколико полиса путног осигурања и испише га на главном излазу. Након тога означи неактивним полисе осигураника истекле пре 27.11.2015., продужи важење полиса осигураника до краја године и испише осигуранике.

---

**НАПОМЕНЕ: а)** Колоквијум траје **120** минута.

**б)** Рад се предаје искључиво у вежбанци за испите (-5 поена за неадекватну вежбанку). Није дозвољено имати поред себе друге листове папира, нити уз себе имати мобилни телефон, без обзира да ли је укључен или искључен.

**в)** Водити рачуна о уредности. Нечитки делови текста ће бити третирани као непостојећи. Решења задатака навести по горњем редоследу (-1 поен за лош редослед). Препоручује се рад обичном графитном оловком.

**г)** Решење задатка не треба раздвајати у датотеке. Довољно је за сваку класу навести дефиницију класе и одмах иза ње евентуалне дефиниције метода које нису дефинисане у самој класи.

**д)** Резултати колоквијума биће објављени на *Web*-у на адреси:
`http://rti.etf.bg.ac.rs/rti/ir2oo1/index.html/`

```
#include <iostream>
#include <string>
#include <cstdlib>
using namespace std;
class Datum {
public:
   Datum(int dd=27,int mm=11,int gg=2015)
      { d = dd; m = mm; g = gg;}
   friend bool operator>(const Datum& d1,
                         const Datum& d2) {
      if (d1.g != d2.g) return d1.g > d2.g;
      if (d1.m != d2.m) return d1.m > d2.m;
      return d1.d > d2.d;
   }
   friend ostream& operator<<(ostream& os,
                         const Datum& d) {
   return os<<d.d<<'.'<<d.m<<'.'<<d.g<<'.';}
private:
    int d, m, g;
};
```

```
class Osiguranje {
public:
   enum Tip {STANDARD, KOMFORT, EKSKLUZIV};
   Osiguranje(const Datum& p, const Datum&
                    k, Tip t = STANDARD) {
      if (p>k) exit(1);
      poc = p; kraj = k; tip = t;
   }
   virtual ~Osiguranje() {}
   Osiguranje(const Osiguranje &o)
      { poc=o.poc; kraj=o.kraj; tip=o.tip; }
   Osiguranje& operator=(const Osiguranje&
                                        o) {
      poc=o.poc; kraj=o.kraj; tip=o.tip;
      return *this; }
   bool dohvAkt() {return aktivna;}
   const Datum& dohvTr() { return kraj;}
   void oznaci() { aktivna = false;}
   Osiguranje& produzi(const Datum& d) {
      if(d>kraj){ kraj = d; aktivna = true; }
      return *this; }
   virtual int premija() const = 0;
   friend ostream &operator<<(ostream &os,
                       const Osiguranje &o)
      { o.pisi(os); return os; }
protected:
   static int pos_id; int id = ++pos_id;
   Datum poc, kraj; Tip tip;
   bool aktivna = true;
   virtual void pisi(ostream &os) const {
      static string tipovi[]={"S","K","E"};
      os << id << " - " << tipovi[tip] <<
      '[' << premija() << "eur]" << '['
      << poc << '-' << kraj << ']';
   }
};
int Osiguranje::pos_id = 0;
```

```
class PutnoO: public Osiguranje {
public:
   PutnoO(const Datum& p, const Datum& k,
   Tip t = STANDARD): Osiguranje(p, k, t) {}
   int premija() const override {
      static int prem[]={5000,7000,10000};
      return prem[tip];
   }
protected:
   void pisi(ostream &os) const override {
      os << "Putno_osiguranje : ";
      Osiguranje::pisi(os);
   }
};
```

```
class Osiguranik {
public:
   Osiguranik(string i) { ime = i;}
   Osiguranik(const Osiguranik& o) = delete;
   void operator=(const Osiguranik& o) =
                                      delete;
   ~Osiguranik();
   Osiguranik& operator+=(Osiguranje *o){
      posl = (!prvi ? prvi : posl->sled)
                                = new Elem(o);
      return *this;
   }
   void oznaci(const Datum& d);
   void produzi(const Datum& d);
   friend ostream &operator<<(ostream &os,
                        const Osiguranik &k);
private:
   struct Elem {
      Osiguranje *osig; Elem *sled;
      Elem(Osiguranje *o, Elem *sl =
                nullptr): osig(o), sled(sl){}
      ~Elem() { delete osig; }
   };
   Elem *prvi = nullptr, *posl = nullptr;
   static int pos_id; int id = ++pos_id;
   string ime;
};
int Osiguranik::pos_id = 0;
Osiguranik::~Osiguranik() {
   while (prvi) {
      Elem* stari = prvi; prvi = prvi->sled;
      delete stari;
   } posl = nullptr;
}
void Osiguranik::oznaci(const Datum& d) {
   Elem *tek = prvi;
   while (tek) {
      if (d > tek->osig->dohvTr())
         tek->osig->oznaci();
      tek = tek->sled;
   }
}
void Osiguranik::produzi(const Datum& d) {
   Elem *tek = prvi;
   while (tek) {
      if (tek->osig->dohvAkt())
        tek->osig->produzi(d);
      tek = tek->sled;
   }
}
ostream &operator<<(ostream &os, const
                              Osiguranik &o) {
   os << o.ime << '(' << o.id << "){";
   Osiguranik::Elem *tek = o.prvi;
   while (tek) {
      os << *tek->osig;
      if (tek->sled) os<< endl;
      tek = tek->sled;
   }
   return os << '}';
}
```

```
int main() {
   Osiguranik o("Marko");
   o+=new PutnoO(Datum(23,7,2015),
Datum(13,8,2015), Osiguranje::KOMFORT);
   o+=new PutnoO(Datum(10,11,2015),
Datum(29,11,2015), Osiguranje::EKSKLUZIV);
   cout << o << endl;
   o.oznaci(Datum(27,11,2015));
   o.produzi(Datum(31,12,2015));
   cout << o << endl; return 0;
}
```

```
Marko(1){Putno_osiguranje : 2 - k[7000eur][23.7.2015.-
13.8.2015.]
Putno_osiguranje : 3 - e[10000eur][10.11.2015.-29.11.2015.]}
Marko(1){Putno_osiguranje : 2 - k[7000eur][23.7.2015.-
13.8.2015.]
Putno_osiguranje : 3 - e[10000eur][10.11.2015.-31.12.2015.]}
```