ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ
*Одсек за РТИ и Одсек за СИ* Београд 16.01.2014.

# Испит из Објектно оријентисаног програмирања I

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

**а)** Да ли може да се преклопи бинарни аритметички оператор глобалном пријатељском функцијом са два целобројна аргумента и зашто?

**б)** Ако је: `class I:public O{…};` и ако је: `I i[10];` описати проблем који ће наступити приликом позива `m(i);` за дефиницију `void m(O *o){…o[3]…}`. Да ли проблем може да се открије у време превођења и зашто?

**в)** Ако постоји шаблонска класа `template<typename A, typename B> class X{…};` чија имплементација не одговара за замену формалног аргумента (параметра) `A` стварним аргументом `int`, написати декларацију шаблонске класе која решава проблем.

**2)** *(укупно 70 поена)* Реализовати на језику *C++* следећи систем класа (класе опремити оним конструкторима, деструктором и оператором за доделу вредности, који су потребни; грешке пријављивати изузецима типа једноставних класа које су опремљене писањем текста поруке):

- *(30 поена)* ***Дете*** има задат пол и име који могу да се дохвате. Не може да се копира ни на који начин. Уписује се у излазни ток (`it<<dete`) у облику *ime*`:`*pol*, где је ознака пола слово **`M`** или **`Z`**.
- Апстрактан ***поклон*** задате цене има јединствен, аутоматски генерисан, целобројан идентификатор. Може да се дохвати једнословна ознака врсте поклона и једнословна ознака по̂ла детета којем је намењен, да се дохвати цена, да се направи полиморфна копија и да се упише у излазни ток (`it<<pok`) у облику *vrsta*`.`*id*`(`*cena*`)`.
- ***Аутић***, ***лутка*** и новогодишњи ***украс*** су поклони. Ознаке врсте производа су **`A`**, **`L`** и **`U`**, а намењени су мушкој, женској односно и једној и другој деци, респективно. Ознаке по̂лова којима су поклони намењени су: мушки – **`M`**, женски – **`Z`** или оба по̂ла – **`?`**
- *(30 поена)* ***Збирка*** може да садржи произвољан број показивача на ствари произвољног типа које могу полиморфно да се копирају. Може да се дода ствар збирци, да се извади ствар која је прва стављена (грешка је ако нема ствари у збирци), да се дохвати број ствари у збирци и да се збирка упише у излазни ток (`it<<zbirka`) у облику **`[`***ствар*`,…,`*ствар*`]`.
- ***Магацин*** је збирка поклона. Не може да се копира ни на који начин.
- Новогодишњи ***пакетић*** је збирка поклона за мушко или женско дете, чија укупна цена не сме да премаши задату дозвољену цену, у којој могу бити поклони само за пол детета којем је намењен пакетић и поклони за оба пола. Може да се дохвати дозвољена цена и стварна цена пакетића. Грешка је ако се при стављању поклона премаши дозвољена цена пакетића, као и ако се покуша да се у пакетић стави поклон за супротан пол.
- *(10 поена)* ***Деда Мраз*** формира новогодишњи пакетић задате дозвољене цене задатом детету тако што га напуни, што више може, поклонима одговарајуће врсте из задатог магацина. Ако поклон узет из магацина не одговара текућем пакетићу, враћа се у магацин.

Написати на језику *C++* **програм** који створи магацин, напуни га са неколико поклона и испише га на главном излазу, створи дете и Деда Мраза који формира пакетић за то дете и испише на главном излазу дете, пакетић и завршни садржај магацина. Сви параметри треба да буду константе (не треба ништа учитати).

---

**НАПОМЕНЕ:** **а)** Испит траје **180** 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 испита биће објављени на *Web*-у на адреси: `home.etf.rs/~kraus/` (одреднице: *настава* | <име предмета> | *оцене* | *испити*).

```
#include <iostream>
#include <cstring>
using namespace std;

class Dete {
  char pol; char* ime;
  Dete(const Dete&) {}
  void operator=(const Dete&) {}
public:
  Dete(const char* iime, char ppol) {
    ime = new char [strlen(iime)+1];
    strcpy(ime, iime);
    pol = ppol;
  }
  ~Dete() { delete [] ime; }
  const char* dohIme() const
    { return ime; }
  char dohPol() const { return pol; }
  friend ostream& operator<<
      (ostream& it, const Dete& d)
    { return it<<d.ime<<':'<<d.pol; }
};
```

```
class Poklon {
  static int posId; int id; float cena;
public:
  Poklon(float c) {cena=c; id=++posId;}
  Poklon(const Poklon& p)
    { cena = p.cena; id = ++posId; }
  virtual ~Poklon() {}
  Poklon& operator=(const Poklon& p)
    { cena = p.cena; return *this; }
  virtual char dohVrs() const =0;
  virtual char dohPol() const =0;
  virtual float dohCen() const
    { return cena; }
  virtual Poklon* kopija() const =0;
  friend ostream& operator<<
    (ostream& it, const Poklon& p) {
      return it << p.dohVrs() << '.'
        << p.id << '(' << p.cena <<')';}
};

int Poklon::posId = 0;
```

```
class Autic: public Poklon {
public:
  Autic(float cena): Poklon(cena) {}
  char dohVrs() const { return 'A'; }
  char dohPol() const { return 'M'; }
  Autic* kopija() const
    { return new Autic(*this); }
};
```

```
class Lutka: public Poklon {
public:
  Lutka(float cena): Poklon(cena) {}
  char dohVrs() const { return 'L'; }
  char dohPol() const { return 'Z'; }
  Lutka* kopija() const
    { return new Lutka(*this); }
};
```

```
class Ukras: public Poklon {
public:
  Ukras(float cena): Poklon(cena) {}
  char dohVrs() const { return 'U'; }
  char dohPol() const { return '?'; }
  Ukras* kopija() const
    { return new Ukras(*this); }
};
```

```
class GPrazna {};
inline ostream& operator<<
  (ostream& it, const GPrazna&)
{ return it << "*** Zbirka je prazna!";}
```

```
template <typename E>
class Zbirka {
  struct Elem {
    E* e; Elem* sled;
    Elem(E *ee) { e = ee; sled = 0; }
  };
  Elem *prvi, *posl; int duz;
  void kopiraj(const Zbirka& z);
  void brisi();
public:
  Zbirka() { prvi = posl = 0; duz = 0; }
  Zbirka(const Zbirka& z) { kopiraj(z);}
  ~Zbirka() { brisi(); }
  Zbirka& operator=(const Zbirka& z) {
    if (this != &z){brisi();kopiraj(z);}
    return *this;
  }
  virtual Zbirka& stavi(E* e) {
    posl = (!prvi?prvi:posl->sled) =
      new Elem(e);
    duz++; return *this;
  }
  virtual E* uzmi() {
    if (!prvi) throw GPrazna();
    E* e = prvi->e; Elem* stari = prvi;
    prvi = prvi->sled; delete stari;
    if (!prvi) posl = 0;
    duz--; return e;
  }
  int brElem() const { return duz; }
  template <typename T>
  friend ostream& operator<<
    (ostream& it, const Zbirka<T>&z);
};
```

```
template <typename E>
void Zbirka<E>::kopiraj
    (const Zbirka<E>& z) {
  prvi = posl = 0; duz = z.duz;
  for (Elem* tek=z.prvi; tek;
       tek=tek->sled)
    posl = (!prvi ? prvi : posl->sled)
         = new Elem(tek->e->kopija());
}

template <typename E>
void Zbirka<E>::brisi() {
  while (prvi) {
    Elem* stari = prvi; prvi=prvi->sled;
    delete stari->e; delete stari; }
  posl = 0; duz = 0;
}

template <typename T>
ostream& operator<<
    (ostream& it, const Zbirka<T>& z) {
  it << '[';
  for (Zbirka<T>::Elem* tek=z.prvi; tek;
       tek=tek->sled) {
    it<<*tek->e; if (tek->sled) it<<',';
  }
  return it << ']';
}
```

```
class Magacin: public Zbirka<Poklon> {
  Magacin(const Magacin&) {};
  void operator=(const Magacin&) {};
public:
  Magacin(): Zbirka<Poklon>() {}
};
```

```
class GPol {};
inline ostream& operator<<
    (ostream& it, const GPol&)
  { return it << "*** Ne odgovara pol!";
}
```

```
class GCena {};
inline ostream& operator<<
    (ostream& it, const GCena&)
  { return it << "*** Previsoka cena!";
}
```

```
class Paketic: public Zbirka<Poklon> {
  char pol; float dozv, cen;
public:
  Paketic(char p, float d)
    { pol = p; dozv = d; cen = 0; }
  Paketic& stavi(Poklon* p) {
    char pl = p->dohPol();
    if (pl!=pol&&pl!='?') throw GPol();
    float cn = p->dohCen();
    if (cen+cn > dozv) throw GCena();
    Zbirka<Poklon>::stavi(p); cen += cn;
    return *this;
  }
  float dohCen() const { return cen; }
  float dohDozv() const { return dozv; }
};
```

```
class DedaMraz {
public:
  Paketic* napravi(float max,
      const Dete& det, Magacin& mag)  {
    char polD = det.dohPol();
    Paketic* pak=new Paketic(polD, max);
    int br = mag.brElem();
    for (int i=0; i<br; i++) {
      Poklon* pok = mag.uzmi();
      char polP = pok->dohPol();
      if ((polP=='?' || polP==polD) &&
           pak->dohCen()+pok->dohCen()<=
           pak->dohDozv())
        pak->stavi(pok);
      else
        mag.stavi(pok);
    }
    return pak;
  }
};
```

```
int main() {
  Magacin mag;
  mag.stavi(new Autic(500));
  mag.stavi(new Lutka(450));
  mag.stavi(new Autic(600));
  mag.stavi(new Ukras(150));
  cout << "M:" << mag << endl;
  Dete dete("Marko",'M');
  Paketic* pak(DedaMraz().napravi
    (1000,dete,mag));
  cout << "D:" << dete << endl
       << "P:" << *pak << endl
       << "M:" << mag  << endl;
  delete pak;
}
```

```
M:[A.1(500),L.2(450),A.3(600),U.4(150)]
D:Marko:M
P:[A.1(500),U.4(150)]
M:[L.2(450),A.3(600)]
```