ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ
Одсек за РТИ Београд 27.11.2016.

Колоквијум из
Објектно оријентисаног програмирања I

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

а) Шта и у ком случају може бити резултат опеарције `delete p`;

б) Који је редослед конструкције и деструкције елемената низа који су објекти неке класе?

в) Навести могуће, стилски оправдане (тако да имитирају својства оператора за уграђене типове у језик), декларације операторске функције `operator-` за операнд(е) типа `X` када је у питању (а) унарни оператор, (б) бинарни оператор.

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

- *(25 поена)* ***Тачка*** у *n*-димензионалном координатном систему има целобројну вредност *n* и низ реалних координата (k_1, k_2, ..., k_n). Подразумевано, тачка је у дводимензионалном координатном систему и налази се у координатном почетку (обе координате имају вредност 0.0). Може да се створи тачка са свих *n* координата исте задате вредности и да се створи тачка на основу низа од *n* реалних бројева. Може да се дохвати димензија, као и појединачна координата тачке на основу индекса (`tacka[ind]`, у случају индекса ван опсега завршити програм). Може да се одреди растојање између две тачке једнаких димензија (квадратни корен суме квадрата растојања по координатама, `t1-t2`) и да се тачка упише у излазни ток (`it<<t`) у облику $(k_1, k_2, \ldots, k_n)$.
- *(25 поена)* ***Облак*** тачака има димензију координатног система, као и произвољан број тачака одговарајућих димензија. Ствара се празан, после чега се тачке додају појединачно (`oblak+=tacka`). Уништавањем облака уништавају се и његове тачке. Може да се дохвати број тачака у облаку, да се одреди центроид облака (тачка чије су координате аритметичке средине одговарајућих координата свих тачака облака) и да се облак упише у излазни ток (`it<<oblak`), тако што се у првом реду испише центроид у угластим заградма, а затим у посебним редовима испишу све тачке у облаку. Облак се не може копирати ни премештати, ни иницијализацијом, ни доделом.
- *(10 поена)* ***Компаратору*** облака се задаје дозвољено реално одступање. Могу се упоредити два облака, при чему се они сматрају једнаким ако им је растојање центроида мање или једнако дозвољеном одступању.

(10 поена) Написати на језику *C++* **програм** који направи два облака тродимензионалних тачака и у њих дода неколико тачака, испише их на главном излазу, упореди и испише резултат поређења. Користити фиксне параметре – није потребно ништа учитати с главног улаза.

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 <cmath>
#include <cstdlib>
using namespace std;
class Tacka {
public:
  Tacka(int ddim=2, double kkomp=0);
  Tacka(const Tacka &t) { kopiraj(t); }
  Tacka(Tacka &&t) { premesti(t); }
  Tacka(double *niz, int duz);
  ~Tacka() { brisi(); }
  Tacka& operator = (const Tacka &t) {
    if (this != &t) { brisi(); kopiraj(t); }
    return *this;
  }
  Tacka& operator = (Tacka &&t) {
    if (this != &t) { brisi(); premesti(t); }
    return *this;
  }
  friend double operator - (const Tacka &t1,
                            const Tacka &t2);
  friend ostream& operator << (ostream &it,
                               const Tacka &t);
  int dohvDim() const { return dim; }
  const double& operator [] (int i) const {
    if (i < 0 || i >= dim) exit(1);
    return koord[i];
  }
  double& operator [] (int i) {
    if (i < 0 || i >= dim) exit(1);
    return koord[i];
  }
private:
  int dim; double *koord;
  void kopiraj(const Tacka &t) {
    koord = new double[dim = t.dim];
    for (int i = 0; i < dim; i++)
      koord[i] = t.koord[i];
  }
  void premesti(Tacka &t) {
    dim = t.dim; koord = t.koord;
    t.koord = nullptr;
  }
  void brisi() {
    dim = 0; delete[] koord; koord = nullptr;
  }
};
Tacka::Tacka(int ddim, double kkoord) {
  koord = new double[dim = ddim];
  for (int i = 0; i < ddim; i++)
    koord[i] = kkoord;
}
Tacka::Tacka(double *niz, int duz) {
  koord = new double[dim = duz];
  for (int i = 0; i < duz; i++)
    koord[i] = niz[i];
}
double operator - (const Tacka &t1,
                   const Tacka &t2) {
  if (t1.dim == t2.dim) {
    double suma = 0;
    for (int i = 0; i < t1.dim; i++)
      suma += pow((t1.koord[i] -
                   t2.koord[i]), 2);
    return sqrt(suma);
  } else return 0;
}
ostream& operator << (ostream &it,
                      const Tacka &t) {
  it << "(" << t.koord[0];
  for (int i = 1; i < t.dim; i++)
    it << ", " << t.koord[i];
  return it << ")";
}
```

```
class Oblak {
public:
  Oblak(int ddim=2) : brojTacaka(0),
                      prvi(nullptr), dim(ddim) {}
  Oblak(const Oblak&) = delete;
  Oblak& operator = (const Oblak&) = delete;
  ~Oblak();
  Oblak& operator += (const Tacka &t);
  int dohvBrojTacaka() const {
    return brojTacaka;
  }
  Tacka centroid() const;
  friend ostream& operator << (ostream &it,
                               const Oblak &o);
private:
  int dim; int brojTacaka;
  struct Elem {
    Tacka tacka; Elem *sled;
    Elem(const Tacka &t, Elem *s = nullptr) {
      tacka = t; sled = s;
    }
  };
  Elem *prvi, *posl;
};
Oblak::~Oblak() {
  while (prvi != nullptr) {
    Elem *stari = prvi;
    prvi = prvi->sled; delete stari;
  }
  brojTacaka = 0; posl = nullptr;
}
Oblak& Oblak::operator += (const Tacka &t) {
  posl=(!prvi?prvi:posl->sled)=new Elem(t);
  ++brojTacaka; return *this;
}
Tacka Oblak::centroid() const {
  double *centroid = new double[dim];
  for (int i = 0; i < dim; i++)
    centroid[i] = 0;
  Elem *tren = prvi;
  while (tren != nullptr) {
    for (int i = 0; i < dim; i++)
      centroid[i] += tren->tacka[i];
    tren = tren->sled;
  }
  for (int i = 0; i < dim; i++)
     centroid[i] /= brojTacaka;
  return Tacka(centroid, dim);
}
ostream& operator << (ostream &it,
                      const Oblak &o) {
  it<<"["<<o.centroid()<<"]"<<endl;
  Oblak::Elem *tren = o.prvi;
  while (tren != nullptr) {
    it << tren->tacka << endl;
    tren = tren->sled;
  }
  return it;
}
```

```
class Komparator {
public:
  Komparator(double oodst) { odst=oodst; }
  friend bool uporedi(const Komparator &k,
           const Oblak &o1, const Oblak &o2) {
    return (o1.centroid() - o2.centroid())
                                  <= k.odst; }
private:
  double odst;
};
```

```
int main() {
  Oblak oblak1(3), oblak2(3);
  Tacka a1(3, 1), b1(3, 2), c1(3, 3),
       a2(3, 1.5), b2(3, 2.5), c2(3, 3.5);
  Komparator komparator(0.87);
  ((oblak1+=a1)+=b1)+=c1; cout<<oblak1;
  ((oblak2+=a2)+=b2)+=c2; cout<<oblak2;
  cout << uporedi(komparator, oblak1, oblak2);
  return 0;
}
```

```
[(2, 2, 2)]
(1, 1, 1)
(2, 2, 2)
(3, 3, 3)
[(2.5, 2.5, 2.5)]
(1.5, 1.5, 1.5)
(2.5, 2.5, 2.5)
(3.5, 3.5, 3.5)
1
```