ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ
Одсек СИ Београд, 13.01.2019.

Завршни колоквијум из Објектно оријентисаног програмирања 1

1) *(укупно 100 поена)* Написати на језику C++ следећи систем класа. Класе опремити оним конструкторима, деструктором и операторима доделе који су потребни за безбедно и ефикасно коришћење класа. Грешке пријављивати изузецима типа једноставних класа које су опремљене писањем текста поруке. За генеричке збирке није дозвољено коришћење класа из стандардне библиотеке шаблона (STL).

- *(30 поена)* ***Секвенца*** садржи произвољан број података неког типа. Елементи се могу додавати на крај секвенце. Може се узети елемент са почетка секвенце. Може се дохватити елемент са произвољне позиције у секвенци, као и дужина секвенце. Грешка је уколико се покуша додавање новог елемента након узимања првог елемента из секвенце.
- (*10 поена*) ***Разломак*** се ствара задавањем целобројног имениоца и бројиоца. Подразумевана вредност имениоца је 1. Два разломка је могуће сабрати (`r1 + r2`) и упоредити (`r1 > r2`). Разломак је могуће уписати у излазни ток (`it << razlomak`) у облику *бројилац/именилац*.
- *(10 поена) Музички* ***знак*** се ствара задавањем трајања у виду разломка. Трајање је могуће дохватити. Сабирањем два музичка знака (`znak1 + znak2`) добија се збир њихових трајања. Музички знак је могуће имплицитно конвертовати у разломак који представља његову дужину. Могуће је саставити текстуални опис музичког знака. Знак се у излазни ток уписује (`it<<znak`) у облику *текстуални_опис****(****трајање****)***.
- *(10 поена)* ***Нота*** је музички знак који се ствара задавањем октаве (цео број у опсегу 1-5), висине (*DO, RE, MI, FA, SOL, LA, SI*) и трајања. Ноту је могуће померити за произвољан број октава наниже (`nota<<=pomeraj`) или навише (`nota>>=pomeraj`), што се извршава смањивањем целобројне вредности октаве приликом померања наниже, односно увећавањем исте приликом померања навише. Текстуални опис ноте је облика *висина*. ***Пауза*** је музички знак који се ствара задавањем трајања. Текстуални опис паузе је _.
- *(20 поена)* ***Такт*** се састоји од секвенце музичких знакова. Ствара се празан са задатим разломком који одређује максимално трајање свих музичких знакова који се налазе у њему. Могуће је дохватити максимално трајање такта. Могуће је додати музички знак у такт. Грешка је уколико се додавањем музичког знака прелази максимално трајање такта. Могуће је завршити такт након чега се у њега више не могу додавати музички знакови. Такт је непотпун уколико је збир трајања музичких знакова у њему мањи од максималног трајања такта. Може се проверити да ли је такт завршен, као и да ли је непотпун. Приликом уписа у излазни ток (`it<<takt`) уписују се сви музички знакови одвојени знаком размака, а затим знак |. Такт није могуће копирати ни на који начин.
- *(20 поена)* ***Композиција*** се састоји од секвенце тактова. Ствара се празна након чега се тактови додају један по један. Грешка је уколико такт који се додају у композицију није завршен. Сви тактови у оквиру композиције морају бити једнаког максималног трајања. Непотпун такт се може појавити само на почетку и крају композиције. Композиција се у излазни ток уписује (`it<<kompozicija`) тако што се сваки такт упише у по једном реду. Композицију није могуће копирати ни на који начин.

(0 поена) Приложена је главна функција која направи једну композицију са неколико тактова и испише је на главни излаз.

НАПОМЕНЕ: **а)** Колоквијум траје **150** минута.
б) Сваку класу стављати у засебне датотеке (обавезно .h, по потреби и .cpp)
в) Рад се предаје на предвиђеном мрежном диску (`Rad L:`).
г) На располагању је документација на *Web*-у **на адресама: http://www.cplusplus.com/ и http://en.cppreference.com**
д) Није дозвољено уз себе имати електронске уређаје, без обзира да ли су укључени или искључени.
ђ) Резултати колоквијума биће објављени на *Web*-у на адреси: `http://rti.etf.rs/rti/ir2oo1/index.html/`

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

class GDodaj : public exception {
   string r = "Dodavanje nije moguce";
public:GDodaj(string raz) :r(raz){}
   GDodaj() = default;
   const char* what()const override
   {return r.c_str(); }};
class GIndeks : public exception {
public:const char* what()const override {
   return "Nedozvoljen indeks"; }};
class GTrajanje : public exception {
public:const char* what()const override {
   return "Prekoracenje trajanja takta"; }
};
```

```
template<typename T>
class Sekvenca {
   struct Elem {
     T t;Elem *sl;
     Elem(T e) :t(e), sl(nullptr) {}
   };
   Elem *pr = nullptr, *posl = nullptr;
   int d = 0; bool moze = true;
   void kopiraj(const Sekvenca &s);
   void premesti(Sekvenca &s);
   void brisi();
public:
   Sekvenca() = default;
   Sekvenca(const Sekvenca
   &s){kopiraj(s);}
   Sekvenca(Sekvenca &&s) { premesti(s); }
   Sekvenca& operator=(const Sekvenca &s){
     if (this != &s){brisi(); kopiraj(s);}
      return *this;
   }
   Sekvenca& operator=(Sekvenca &&s) {
     if (this != &s){brisi();premesti(s);}
     return *this;
   }
   ~Sekvenca() { brisi(); }
   Sekvenca& dodaj(T elem);
   T uzmi();
   int duz()const { return d; }
   T& operator[](int i);
   const T& operator[](int i)const {
   return const_cast<Sekvenca&>(*this)[i];
   }
};
template<typename T>
Sekvenca<T>& Sekvenca<T>::dodaj(T t) {
   if (!moze) throw GDodaj();
   posl=(pr?posl->sl:pr) = new Elem(t);
   d++;return *this;
}
template<typename T>
T Sekvenca<T>::uzmi() {
   T t = pr->t;
   Elem *pom = pr;
   pr = pr->sl;
   if (!pr) posl = pr;
   delete pom;
   d--; moze = false; return t;
}
template<typename T>
void Sekvenca<T>::kopiraj(const
   Sekvenca<T> &s) {
   d = 0; moze = true;
   for (Elem *e = s.pr; e; e = e->sl)
   dodaj(e->t);
}
template<typename T>
void Sekvenca<T>::premesti(Sekvenca<T> &s)
   {
   d = s.d; moze = s.moze;
   pr = s.pr; posl = s.posl;
   s.pr = nullptr;
}
template<typename T>
void Sekvenca<T>::brisi() {
   while (pr) {
    Elem *pom = pr;
       pr = pr->sl;
        delete pom;
   }
}
template<typename T>
T& Sekvenca<T>::operator[](int i) {
   if (i < 0 || i >= d) throw GIndeks();
   Elem *e = pr;
   for (; i > 0; i--, e = e->sl);
   return e->t;
}
```

```
class Razlomak{
   int i, b;
public:Razlomak(int br, int im = 1)
    :i(im), b(br){}
   friend Razlomak operator+(const
   Razlomak &r1, const Razlomak &r2){
     return Razlomak(r1.b*r2.i +
       r2.b*r1.i, r1.i*r2.i);
   }
   friend bool operator>(const Razlomak
   &r1, const Razlomak &r2){
     return r1.b*r2.i > r2.b*r1.i;
   }
   friend ostream& operator<<(ostream &it,
   const Razlomak &r){
     return it << r.b << "/" << r.i;
   }
};
```

```
class Znak{
   Razlomak t;
public:
   explicit Znak(Razlomak tr) :t(tr){}
   Razlomak tr()const{ return t; }
   friend Razlomak operator+(const Znak
   &z1, const Znak &z2){
     return z1.t + z2.t;
   }
   operator Razlomak()const{ return t; }
   virtual string opis()const = 0;
   friend ostream& operator<<(ostream &it,
   const Znak &z){
     return it << z.opis() << "(" << z.t
         << ")";
   }
   virtual ~Znak(){}
};
```

```
class Nota :public Znak{
public:enum Visina{ DO, RE, MI, FA, SOL,
   LA, SI };
private:
   int o; Visina v;
public:
   Nota(int ok, Visina vi, Razlomak t)
   :Znak(t), o(ok), v(vi){}
   Nota& operator<<=(int p){
     o -= p; return *this; }
   Nota& operator>>=(int p){
     o += p; return *this; }
   string opis()const override{
   string s[]{"DO", "RE", "MI", "FA",
   "SOL", "LA", "SI"};
       return s[v];
   }
};
```

```
class Pauza :public Znak{
public:
   Pauza(Razlomak tr) :Znak(tr){}
   string opis()const override{ return
   "_"; }
};
```

```
class Takt{
   Sekvenca<Znak*> s;
   Razlomak max, tek;
   bool z = false;
public:
   Takt(Razlomak mt) :max(mt), tek(0){}
   Takt(const Takt &t) = delete;
   void operator=(const Takt &t) = delete;
   Takt& dodaj(Znak &z);
   Takt& zavrsi(){z=true; return *this; }
   bool zavrsen()const{ return z; }
   bool nepotpun()const{ return max>tek; }
   Razlomak tr()const{ return max; }
   friend ostream& operator<<(ostream &it,
   const Takt &t){
     for (int i = 0; i < t.s.duz(); i++)
      it << *t.s[i] << " ";
     return it << "|";
   }
};
Takt& Takt::dodaj(Znak &zn){
   if (z) throw GDodaj("Takt je zavrsen");
   if (tek + zn.tr() > max)
     throw GTrajanje();
   s.dodaj(&zn); tek = tek + zn;
   return *this;
}
```

```
class Kompozicija{
   Sekvenca<Takt*> s;
   bool prvi = true, moze = true;
   Razlomak tr = Razlomak(0);
public:
   Kompozicija() = default;
   Kompozicija(const Kompozicija &k) =
   delete;
   void operator=(const Kompozicija &k) =
   delete;
   Kompozicija& dodaj(Takt &t);
   friend ostream& operator<<(ostream &ot,
   const Kompozicija &k){
     for (int i = 0; i < k.s.duz(); i++)
       ot << *k.s[i] << endl;
     return ot;
   }
};
Kompozicija& Kompozicija::dodaj(Takt &t){
   if (!t.zavrsen())
    throw GDodaj("Takt nije zavrsen");
   if (prvi){prvi = false;tr = t.tr();}
   else{
     if (t.tr() > tr || tr > t.tr())
      throw GDodaj("Nejednaka duzina");
     if (t.nepotpun()){
       if (!moze) throw GDodaj("Nepotpun
   takt");else moze = false;}
   }
   s.dodaj(&t);
}
```

```
int main() {
   try {
   Razlomak osm(1,8),cet(1, 4),pol(1, 2);
   Nota mi_2(2, Nota::MI, pol),
     re_4(2, Nota::RE, cet),
     do_8(2, Nota::DO, osm),
     la_8(2, Nota::LA, osm),
     sol_4(2, Nota::SOL, cet),
     si_2(2, Nota::SI, pol);
   Pauza p_2(pol),p_4(cet),p_8(osm);

   Takt t1(pol), t2(pol), t3(pol);
   t1.dodaj(re_4).zavrsi();
   t2.dodaj(mi_2).zavrsi();
   t3.dodaj(la_8).dodaj(p_8).dodaj(sol_4).
   zavrsi();

   Kompozicija k;
   cout<<k.dodaj(t1).dodaj(t2).dodaj(t3);
   }
   catch (exception &e) {
   cout << endl << e.what();
   }
}

//Primer izvrsavanja:
RE(1/4) |
MI(1/2) |
LA(1/8) _(1/8) SOL(1/4) |
```