NEPTUN kód:	NÉV: Aláírás:

feladat	pont	min	elért
1.	10	–	
2.	10	–	
3.	10	–	
4.	10	–	
∑	40	14	

Programozás 2. NZH, 2019. május 14. – BME-TTK, fizika BSc
Arcképes igazolvány hiányában nem kezdheted meg a ZH-t. A feladatok megoldására összesen 90 perc áll rendelkezésre. A feladatlapot névvel ellátva akkor is be kell adni, ha semmilyen megoldást sem készítettél. Minden feladatmegoldást külön lapra írj! Minden lapra írd fel a neved és a neptun kódod, valamint a feladat sorszámát! Ezek hiányában a feladatot nem értékeljük. Szabványos C++98/11 megoldásokat értékelünk csak.

..13 – elégtelen, 14..20 – elégséges, 21..27 – közepes, 28..34 – jó, 35.. – jeles

---

A feladatokban szükség lesz az alábbi két osztályra:

Az alább felsorolt, kívülről is elérhető tagfüggvényekkel rendelkező `Pet` osztály. Kívülről el nem érhető (a leszármazott számára sem) módon, megfelelő típusú tagváltozókban tárolja a kisállat fajnevét (pl. kutya, macska, homár...) és árát.

```
Pet(const std::string &faj,
      unsigned ar);
virtual void print()const;
     //kiírja a fajt és az árat
virtual ~Pet();
```

Az alább felsorolt, kívülről is elérhető tagfüggvényekkel rendelkező `Rac` osztály. Az osztály egy racionális számot reprezentál. Kívülről el nem érhető módon, `int` típusban tárolja a *sz* és *n* komponenst. Egyszerűsített alakot tárol, amihez privát `simplify()` függvénnyel bír.

```
Rac(int sz, int n);
int getsz()const;
int getn()const;
Rac sqr()const; //szam negyzete
```

1. feladat: Öröklés, sablonok (2×5 p)

a) **Származtass** *Kutya* osztályt a `Pet` osztályból! A kutya többlettulajdonsága a kisállathoz képest a fajta (pl. komondor, mudi, kuvasz...) és az egyed neve. Készíts konstruktort, mely beállítja az összes tagváltozót, valamint `print` függvényt, amely kiírja egy kutya összes jellemzőjét. **A Pet osztályt nem módosíthatod.** Írd meg a tesztelő **kétsoros kódrészletét**, melyben létrehozol egy kutyát, és kiírod a jellemzőit.

```
class Kutya : public Pet{
   std::string fajta, nev;
public:
   Kutya(unsigned ar, const std::string& fajta, const std::string &nev)
                          :Pet("Canis Lupus",ar), fajta(fajta), nev(nev) {}
   void print() const{
      Pet::print();
      std::cout << fajta << nev;
   }
};

int main(){
   Kutya k(12000, "foxterrier", "Lexi");
   k.print();
}
```

b) Készítsd el a megadott `Rac` osztály sablon változatát, ahol a sablonparaméter a tagváltozók típusát adja! (A tagfüggvények implementációját is írd meg.) Hibás adatra nem kell figyelned. Egysoros példával (`short`-okkal reprezentált racionális szám változó) mutasd meg a sablon használatát.

```
template <class T>
class Rac{
   T sz, n;
```

```
    void simplify(){/*nem kellett megvalositani*/}
public:
    Rac(T sz, T n) :sz(sz), n(n){}
    T getsz()const{ return sz; }
    T getn()const{ return n; }
    Rac sqr()const{
        return Rac(sz*sz,n*n);
    }
};

    Rac<short> r1(2,3);
```

2. feladat: Operátorok

Egészítsd ki a feladatok felett megadott Rac osztályt a következő operátorokkal. Amit lehetséges, tagfüggvényként valósítsd meg. Mindegyik operátor a racionális számoktól, valamint az adott operátortól megszokott/elvárható módon viselkedjen, ígérjen konstans viselkedést, ahol lehetséges. Hibás adatra nem kell figyelned. Figyelj oda, hogy mindig egyszerűsített alakot tárolj, valamint nincs default ctor.

Rac - *Rac*,
Rac /= *Rac*,
~ *Rac*, reciprokképző,
Rac == *Rac*,
int * *Rac*,
`std::ostream`-re kiíró << (két, szóközzel elválasztott egész számot írjon ki) és
`(float)` *Rac* (típuskonverzió operátor valósra).

A Rac osztálynak a feladatok felett felsorolt (létező) tagfüggvényeit **nem kell megírnod**, elegendő kipontoznod deklarációjuk helyét az osztályban.
Készíts rövid main függvényt, melyben bemutatod valamennyi operátor használatát!

```
#include <iostream>
#include <stdexcept>
class Rac{
    ...
    Rac operator-(const Rac& rhs)const{return Rac(sz*rhs.n-n*rhs.sz,n*rhs.n);}
    const Rac& operator/=(const Rac& rhs){sz*=rhs.n;n*=rhs.sz;return *this;}
    Rac operator~()const{return Rac(n,sz);}
    bool operator==(const Rac& rhs)const{return sz==rhs.sz && n==rhs.n;}
    operator float()const{return (float)sz/n;}
};

Rac operator* (int k, const Rac& rhs){
    return Rac(k*rhs.getsz(),rhs.getn());
}

std::istream& operator>>(std::ostream& os, const Rac& rhs){
    os<<rhs.getsz()<<' '<<rhs.getn();
    return os;
}

int main()
{
    Rac r1=Rac(3,2), r2(2,4);
    r1=r1-r2;
    r2/=r1;
```

```
    r1=~r1;
    std::cout<<r2;
    if(r1==r2)
        r2=5*r2;
    std::cout<<(float)(r1)<<std::endl;
}
```

3. feladat: Tároló

Készíts hosszú egész (long) értékeket tárolni képes várakozási *Sor* osztályt! A Sor egy tároló, mely tetszőlegesen sok értéket tud tárolni. A Sorhoz új elemet hozzáadni a push, elemet kivenni a pop tagfüggvénnyel lehet. A push függvény mindig a tömb végére helyezi az új elemet. A pop függvény a legrégebben betett értéket adja vissza, és egyúttal el is távolítja a Sorból.

Az elkészített Sor osztály a következőket kell tudja: dinamikus tömbben tárolja az értékeket (nem használhatsz STL tárolót). Írd meg az alapértelmezett konstruktort, destruktort, másoló konstruktort és az értékadó operátort! Írd meg a pop függvényt is, amely visszaadja a legrégebben betett értéket, és egyúttal el is távolítja azt a Sorból. Ha üres Sorra hívja a felhasználó a pop függvényt, dobjon std::underflow_error kivételt! A push függvényt **nem** kell elkészítened, de feltételezheted, hogy létezik, és ha kell, használhatod.

Írj main függvényt, melyben kb. 8 sorban bemutatod a Sor használatát.

+1 bónuszpontért: kapd el a Sor által esetleg dobott kivételt. Elkapott kivétel esetén írj ki hibaüzenetet!

```
#include <iostream>
#include <stdexcept>

class Sor
{
    int elemSzam;
    long* adat;
public:
    Sor(){elemSzam=0; adat=NULL;}

    Sor(const Sor &m){
        adat=NULL;
        *this=m;
    }

    const Sor& operator=(const Sor &m){
      if(this!=&m){
        elemSzam=m.elemSzam;
        delete[] adat;
        adat=new long[elemSzam];
        for(int i=0;i!=elemSzam;++i)
            adat[i]=m.adat[i];
      }
      return *this;
    }

    ~Sor(){delete[] adat;}

    void push(long elem);

    long pop(){
        if(elemSzam==0) throw std::underflow_error("Hiba: ures sor");
        long ret=adat[0],*t=new long[elemSzam-1];
        for(int i=1;i!=elemSzam;++i)
            t[i-1]=adat[i];
```

```
        delete[] adat;
        adat=t;
        --elemSzam;
        return ret;
    }

};

int main()
{
    try{
        Sor h,g;
        h.push(3L);
        h.push(2L);
        g=h;
        std::cout<<h.pop();
        std::cout<<g.pop()<<g.pop();
    }
    catch(std::exception &e){
        std::cerr<<e.what();
    }
}
```

4. feladat: STL

Egy biológus egy hím kísérleti szarvasbogár mozgását szeretné elemezni. A bogár az origóból indul, pozícióját másodpercenként mérték a laboratórium asztalához rögzített descartes-i koordinátarendszerben, centiméter egységben, az érvényes tartományon valamelyik koordináta nemnegatív. Ezt a rögzített adatsorozatot kapja majd a programunk a szabványos bemenetén szóközzel elválasztott valós számpárok formájában. Az utolsó érvényes adatot két negatív érték követi.

Készíts programot, amely egy `std::vector`-ban tárolja a mért adatokat. A koordinátákat adott `Vekt2D` típusú objektumokban kell tárolni, értelemszerűen az origóból az adott pontba mutató vektor. A `Vekt2D` típust nem kell létrehoznunk; publikus valós tagváltozókkal bír (x, y), és rendelkezik aritmetikai, valamint kiíró és beolvasó operátorokkal. Az `std::vector` ilyen `Vekt2D`-ket tároljon!

Szükség szerint létrehozhatsz segítő függvényt vagy osztályt.

a) Olvasd be, és tárold el a nyomkövető által szolgáltatott koordinátákat!
b) Váltsd át méterbe az összes koordinátát, és az eredményt helyezd egy új tömbbe. Ha STL algoritmust használsz, és ezt helyesen teszed, +1 bónuszpontot kapsz.
c) Hozzuk létre a bogár sebességvektorának tömbjét, majd a skalár sebességértékek tömbjét. (Természetesen ez utóbbi már valósak tömbje, amibe a vektorok abszolút értéke kerül.) Ha STL algoritmust használsz, és ezt helyesen teszed, +1 bónuszpontot kapsz.
d) Az útja során egyszer állt csak meg a szarvasbogár (két egymást követő mért pont azonos), ahol igen kis koncentrációban nőstény feromont helyeztünk el. Hányadik másodpercnél történt ez? Mennyi volt a bogár csúcssebessége? Ha STL algoritmust használsz, és ezt helyesen teszed, +1 bónuszpontot kapsz.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
#include "vekt2d.h"

Vekt2D cm2m(const Vekt2D &v)//function
{
    return Vekt2D(v.x/100,v.y/100); //v/100 is jo
}
```

```
class hossz{//functor
  public:
    double operator()(const Vekt2D &u)
    {
        return std::sqrt(u.x*u.x+u.y*u.y);
    }
};

int main()
{
    Vekt2D a;
    std::vector<Vekt2D> s;
    std::cin>>a;
    while(a.x>=0 || a.y>=0)
    {
        s.push_back(a);
        std::cin>>a;
    }

    std::vector<Vekt2D> s_m(s.size());
    std::transform(s.begin(),s.end(),s_m.begin(),cm2m); //+1p

    std::vector<Vekt2D> vv(s_m.size());
    std::adjacent_difference(s_m.begin(),s_m.end(),vv.begin()); // +1p
    std::vector<double> v(vv.size());
    std::transform(vv.begin(),vv.end(),v.begin(),hossz());

    std::vector<double>::iterator megall=std::min_element(v.begin()+1,v.end()),
    max_seb=std::max_element(v.begin(),v.end()); // +1p

    std::cout<<"megallas "<<megall-v.begin()<<" masodperc\n";
    std::cout<<"leggyorsabb "<<*max_seb;
}
```