NEPTUN kód:	NÉV: Aláírás:

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

Programozás 2. NZH, 2018. május 18. – 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 `Sportolo` osztály. Kívülről el nem érhető (a leszármazott számára sem) módon, megfelelő típusú tagváltozókban tárolja a sportoló nevét és életkorát.

```
Sportolo(std::string nev,
      unsigned kor);
virtual void print()const;
virtual ~Sportolo();
```

Az alább felsorolt, kívülről is elérhető tagfüggvényekkel rendelkező `Cplx` osztály. Az osztály egy komplex számot reprezentál. Kívülről el nem érhető módon, `double` típusban tárolja a *re* és *im* komponenst.

```
Cplx(double re=0, double im=0);
double getRe()const;
double getIm()const;
void setRe(double re);
void setIm(double im);
```

1. feladat: Öröklés, sablonok (2×5 p)

a) **Származtass** *Sakkozo* osztályt a `Sportolo` osztályból! A sakkozó többlettulajdonsága a sportolóhoz képest a fejből ismert játszmák száma. Készíts konstruktort, mely beállítja az összes tagváltozót, valamint `print` függvényt, amely kiírja a sakkozó összes jellemzőjét. **A Sportolo osztályt nem módosíthatod.**
Írd meg a tesztelő **kétsoros kódrészletét**, melyben létrehozol egy sakkozót, és kiírod a jellemzőit.

```
class Sakkozo : public Sportolo{
   unsigned jatszma;
public:
   Sakkozo(std::string nev, unsigned kor, unsigned jatszma)
                          :Sportolo(nev,kor), jatszma(jatszma) {}
   void print() const{
      Sportolo::print();
      std::cout << jatszma << " jatszma";
   }
};

int main(){
   Sakkozo s("Leko Peter", 38, 1871);
   s.print();
}
```

b) Készítsd el a megadott Cplx osztály sablon változatát, ahol a sablonparaméter a tagváltozók típusát adja! Egysoros példával (float-okkal reprezentált komplex szám változó) mutasd meg a sablon használatát.

```
template <class T>
class Cplx{
   T re, im;
public:
```

```
   Cplx(T re=T(), T im=T()) :re(re), im(im){}
   T getRe()const{ return re; }
   T getIm()const{ return im; }
   void setRe(T re){ this->re = re; }
   void setIm(T im){ this->im = im; }
};

   Cplx<float> c1;
```

2. feladat: Operátorok

Egészítsd ki a feladatok felett megadott Cplx osztályt a következő operátorokkal. Amit lehetséges, tagfüggvényként valósítsd meg. Mindegyik operátor a komplex számoktól, valamint az adott operátortól megszokott/elvárható módon viselkedjen, ígérjen konstans viselkedést, ahol lehetséges:

Cplx + *Cplx*,
Cplx -= *Cplx*,
~ *Cplx*: komplex konjugáltat képző,
Cplx != *Cplx*,
Cplx / *double*: zérus osztónál dobjon szabályos kivételt,
double * *Cplx*,
`std::ostream`-re kiíró << (két, szóközzel elválasztott valós számot írjon ki).

A Cplx osztálynak a feladatok felett felsorolt tagfüggvényeit **nem kell megírnod**, elegendő kipontoznod deklarációjuk helyét az osztályban. (...)

Készíts rövid main függvényt, melyben bemutatod valamennyi operátor használatát! **+1 bónuszpontért:** kapd el az esetleg dobott kivételt. Elkapott kivétel esetén írj ki hibaüzenetet!

```
#include <iostream>
#include <stdexcept>
class Cplx{
   double re,im;
 public:
   ...
   Cplx operator+(const Cplx& rhs)const{return Cplx(re+rhs.re,im+rhs.im);}
   const Cplx& operator-=(const Cplx& rhs){re-=rhs.re;im-=rhs.im;return *this;}
   Cplx operator~()const{return Cplx(re,-im);}
   bool operator!=(const Cplx& rhs)const{return re!=rhs.re||im!=rhs.im;}
   Cplx operator/(double sc)const{if(sc==0) throw std::domain_error("0 az oszto");
                           return Cplx(re/sc, im/sc);}
};

Cplx operator* (double sc, const Cplx& v){
   return Cplx(sc*v.getRe(),sc*v.getIm());
}

std::ostream& operator<<(std::ostream& os, const Cplx& v){
   os<<v.getRe()<<' '<<v.getIm();
   return os;
}

int main()
{
   try{
      Cplx v=Cplx(3,2)+Cplx(2,4);
      std::cout<<2.5*v<<std::endl;
```

```
        std::cout<<~v<<std::endl;
        Cplx w;
        w=v/1.01;
        if(w!=v)
            v-=w;
        std::cout<<v<<std::endl;
    }
    catch(std::exception &e){
        std::cerr<<"Hiba: "<<e.what();
    }
}
```

3. feladat: Tároló

Készíts hosszú egész (long) értékeket tárolni képes *Halmaz* osztályt! A halmaz egy olyan tároló, mely tetszőlegesen sok *különböző* értéket tud tárolni, a sorrend irreleváns. A halmazba új elemet hozzáadni az `Add`, elemet kivenni a `Remove` tagfüggvénnyel lehet. Az Add függvény csak akkor bővíti a tárolót, ha a kérdéses érték még nincs benne a halmazban, ilyenkor az új elem az eddigiek mögé kerül. Hasonlóképp Remove csak akkor szűkíti, ha a kérdéses érték eleme a halmaznak. A logikai értéket visszaadó `isElement` függvény megvizsgálja, hogy a kérdéses érték benne van-e a halmazban. A paraméter nélküli `isEmpty` a halmaz ürességét adja vissza logikai értékben.

Az elkészített Halmaz osztály dinamikus tömbben kell tárolja az értékeket (nem használhatsz STL tárolót). Írd meg az alapértelmezett konstruktort, destruktort, másoló konstruktort és az értékadó operátort! Írd meg az isEmpty, isElement és a Remove függvényt is. Az Add függvényt **nem** kell elkészítened, de feltételezheted, hogy létezik, és ha kell, használhatod.

Írj main függvényt, melyben kb. 8–10 sorban bemutatod a halmaz használatát.

```
#include <iostream>

class Halmaz
{
    int elemSzam;
    long* adat;
public:
    Halmaz():elemSzam(0),adat(NULL){}

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

    const Halmaz& operator=(const Halmaz &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;
    }

    ~Halmaz(){delete[] adat;}

    void Add(long elem);
```

```
    bool isEmpty()const{return elemSzam==0;}

    bool isElement(long elem)const{
        for(int i=0; i<elemSzam; ++i)
            if (adat[i]==elem) return true;
        return false;
    }

    void Remove(long elem){
        if (!isElement(elem)) return;
        long *t=new long[elemSzam-1];
        for(int i=0,j=0;i!=elemSzam-1;++i,++j){
            if(adat[j]==elem) ++j;
            t[i]=adat[j];
        }
        delete[] adat;
        adat=t;
        --elemSzam;
    }

};

int main()
{
    Halmaz h,g;
    h.Add(3L);
    h.Add(3L);
    std::cout<<h.isElement(5);
    g=h;
    g.Remove(3);
    std::cout<<g.isEmpty()<<h.isEmpty();
}
```

4. feladat: STL

Ejtőernyős ugrást hajtunk végre. A mozgásunk egydimenziósnak tekinthető. Természetesen van nálunk egy barometrikus magasságmérő berendezés, amely másodpercenként rögzíti aktuális magasságunk mérőszámát méterben. A programunk szabványos bemenetére küldi majd a műszer az általa mért pozíció értékeket. Előre nem tudjuk, hogy hány érték fog érkezni. A mérés végét 112m alatti érték jelzi (földetérés). Ezt már nem szabad benne hagyni az adatsorban.

Készíts programot, amely egy `std::vector`-ban tárolja a pozíció értékeket, majd elvégzi az előírt számításokat.

a) Olvasd be a mért értékeket, és tárold el (x) az ugróponttól mért távolságokat ($x_i := x_0 - x_i$), így már az ugrás kezdete lesz az origó!
b) Hozz létre egy újabb vector-t a pillanatnyi sebességek számára! Számítsd ki és tárold el a vectorban a sebességeket ($v = dx/dt$, a kiadódó m/s egységben)! Ha a deriváláshoz STL algoritmust használsz, és ezt helyesen teszed, +1 bónuszpontot kapsz.
c) Hasonló módon számítsd ki a gyorsulás értékeit is ($a = dv/dt$, m/s^2).
d) Határozd meg, és írd ki, hogy mely időpontban nyitottuk ki az ernyőt (legnegatívabb gyorsulás). Ha STL algoritmust használsz, és ezt helyesen teszed, +1 bónuszpontot kapsz.
e) Váltsd át a v tömb m/s egységben értendő értékeit km/h-ra (szorzás 3.6-del). (Lehet új céltömbbe, de lehet a régi értékek helyén is.) Ha STL algoritmust használsz, és ezt helyesen teszed, +2 bónuszpontot kapsz.
f) Mekkora volt a legnagyobb sebességünk az esés során?

```
#include <iostream>
```

```cpp
#include <vector>
#include <algorithm>
#include <numeric>

class ms2kmh{public: double operator()(double x){return 3.6*x;}}; // +1p e)-hez

int main()
{
   double xx,x0;
   std::vector<double> x;
   std::cin>>xx; x0=xx;
   while(xx>=112)
   {
      x.push_back(x0-xx);
      std::cin>>xx;
   }

   std::vector<double> v(x.size());
   std::adjacent_difference(&x[0],&x[x.size()],&v[0]); // +1p vagy
   std::adjacent_difference(x.begin(),x.end(),v.begin()); // +1p vagy
   v[0]=x[0];
   for(unsigned j=1u;j<x.size();++j)
      v[j]=x[j]-x[j-1];

   std::vector<double> a(v.size());
   std::adjacent_difference(v.begin(),v.end(),a.begin());

   double *p=std::min_element(&a[0],&a[a.size()]); // +1p vagy
   std::cout<<p-&a[0]<<'\n';

   std::vector<double>::iterator it=std::min_element(a.begin(),a.end()); // +1p vagy
   std::cout<<it-a.begin()<<'\n';

   unsigned minpos=0;
   for(unsigned i=1u;i!=a.size();++i)
      if(a[minpos]>a[i])minpos=i;
   std::cout<<minpos<<'\n';

   std::transform(v.begin(),v.end(),v.begin(),ms2kmh()); // +1p vagy
   for(unsigned j=0;j<v.size();++j)
      v[j]=3.6*v[j];

   it=std::max_element(v.begin(),v.end());
   std::cout<<*it<<'\n';

}
```