```
Dátum adattípus:

struct Date {
 int y;
 int m;
 int d;
 };

 Date x1;
 x1.y=2005;
 x1.m=3;
 x1.d=15;

 Date x2;
 x2.y=2000;
 x2.m=2;
 x2.d= 29; // Ki tudja van ilyen;
 x2.m=15; // Huh!!

 Ellenőrizzük az értékeket:
 void init_date( Date& dd, int y, int m, int d )
 {
    if( !is_date(y,m,d) ) throw "Hibas datum";
    dd.y=y; dd.m=m; dd.d=d;
}

Végezzünk műveleteket:

Date holnap( Date ma )
{
    Date h;
    h.y = ma.y; h.m = ma.m; // valóban ?
    h.d = ma.d+1; // Huh!!
    return h;
}

Írjunk egy helyes add_day() függvényt
a Date típushoz.

Date holnap( Date ma ) {
    Date h=ma;
    add_day( h, 1)
    return h;
}

Egységbe zárás. Tagfüggvények definiálása:

struct Date {
 int y, m, d;

  Date(int y, int d, int m){ ... }; // ellenőrzi az adatokat
  void add_day(int d ){ ... }
 };
```

```
 Használat:

Date ma(2013,02, 26 );
Date holnap = ma; // Van ilyen?
holnap.add_day(1);

ma.m=13; // huh!!

struct Date {
private:
   int y, m, d;
 public:
   Date(int y, int m, int d); // ellenőrzi az adatokat
   void add_day(int d );
   int year() {return y; }
   int month() {return m; }
   int day() {return d; }
 };

Mekkora a költsége a getter függvényeknek? 0!
Date dd(2013,2,26);
int ev = dd.y; // HIBA !! private
int ev = dd.year(); // Igy már jó

struct és class közötti különbségek:

struct A{  === class A{
                  public:

};             };

class A {  === struct A{
                  private:

};             };

Megállapodás: ha nincs viselkedése, azaz nincs tagfüggvénye
az osztálynak, akkor struct-ot használunk.
Hogyan vesszük észre ha valaki a napokat és a hónapokat felcseréli?
Date dd(2013,2,3); helyett Date dd(2013,3,2); Sehogyan!

class Date {
public:
   enum Month {
      jan=1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec };
   Date(int y, Month m, int d); // ellenőrzi az adatokat
   void add_day(int d );
   int year() {return y; }
   Month month() {return m; }
   int day() {return d; }
private:
   int y;
   Month m;
   int d;
 };
```

```
111     Date::Month m=Date::dec;
112     int x = m; // OK FONTOS
113     m = 10;    // Hiba!
114     m = Date::Month(10); // OK
115
116     Date ma( 2013, Date::feb, 26 ); // OK
117     Date ma( 2013, 26, Date::feb ); // Hiba rossz paraméter sorrend.
118
119     const holnap(2013, Date::feb, 27 );
120     int ev=holnap.year(); // Hiba mert nem konstans függvény
121     Date szabadnapok[30]; // Hiba nincs paraméter nélküli konstruktor
122
123
124   class Date {
125   public:
126       enum Month {
127           jan=1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec
128       };
129
130       class Invalid{}; // throw-hoz
131
132     Date(int y, Month m, int d); // ellenőrzi az adatokat
133     Date();
134
135     void add_day(int d );
136
137     int year() const {return y; }
138     Month month() const {return m; }
139     int day() const {return d; }
140
141   private:
142     int y;
143     Month m;
144     int d;
145     };
146
147
148   Tagfüggvény definiálása osztályon kívül:
149
150   Date::Date()
151   {
152       static const Date dd(2001,Date::jan,1); // start of 21st century
153       y=dd.y;
154       m = dd.m;
155       d = dd.d;
156   }
157
158   Date::Date(int y, Date::Month mm, int dd) {
159       if (!is_date(y,mm,dd)) throw Invalid();
160       this->y = y; // Paraméter és tagváltozó megkülönböztetése
161       m = mm;
162       d = dd;
163   }
164
165
```

```
166  void Date::add_day(int n) {
167      if (n<0) throw Invalid();
168      while (days_in_month(y,m)<n) {
169          n -= days_in_month(y,m);
170          if( m == dec ) { m=jan; y++; } else m = Month(m+1);
171      }
172      d += n;
173  }
174
175  Segédfüggvények vagy globális függvények.
176  Az osztály implementálása nincs kihatással rá.
177  Sok esetben Date értéket vesznek át a paraméterlistán.
178
179  bool leapyear(int y) {
180      if (y%4) return false;
181      if (y%100==0 && y%400) return false;
182      return true;
183  }
184
185  int days_in_month(int y, Date::Month m) {
186      switch (m) {
187      case Date::feb:                    // the length of February varies
188          return (leapyear(y))?29:28;
189      case Date::apr: case Date::jun: case Date::sep: case Date::nov:
190          return 30;
191      default:
192          return 31;
193      }
194  }
195
196  bool is_date(int y, Date::Month  m, int d) {
197      if (d<=0) return false; // d mindig pozitív
198      if (days_in_month(y,m)<d) return false;
199      return true;
200  }
201
202  Kiíratás és beolvasás is segéd operátor függvény.
203  Miért?
204
205  ostream& operator<<(ostream& os, const Date& d) {
206     return os << '(' << d.year() << ',' << d.month() << ',' << d.day() << ')';
207  }
208
209  istream& operator>>(istream& is, Date& dd) {
210      int y, m, d;
211      char ch1, ch2, ch3, ch4;
212      is >> ch1 >> y >> ch2 >> m >> ch3 >> d >> ch4;
213      if (!is) return is;
214      if (ch1!='(' || ch2!=',' || ch3!=',' || ch4!=')') { // oops: format error
215          is.clear(ios_base::failbit);                    // set the fail bit
216          return is;
217      }
218      dd = Date(y,Date::Month(m),d);     // update dd
219      return is;
220  }
```