# Poligon

Mirko ima 4 psa koje trenira za nadolazeće veliko natjecanje. Postavio je poligon dimenzija 8x8 na kojeg može postaviti prepreke kroz koje psi ne mogu prolaziti. Mirkovi psi su označeni brojevima 1, 2, 3, 4. Psi trče naokolo po poligonu, a više pasa (pa čak i svi) se mogu nalaziti u istom polju poligona u istom trenutku. Napravite klasu Poligon na kojoj definirane sljedeće funkcije i operatori (vidi primjer):

- Konstruktor, koji ne prima ništa, te inicijalizira poligon tako da se na njemu ne nalazi niti jedna prepreka, a da su psi smješteni ovako:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | | | | | | |
| 3 | 4 | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

- Operator[], koji djeluje ovako: ako je P poligon, a x=(r, s) uređeni par cijelih brojeva, onda operacija P[x] postavlja prepreku u r-ti redak i s-ti stupac ako se tamo već ne nalazi niti prepreka niti pas. Ako se tamo već nalazi prepreka, ovaj operator ga uklanja. Ako se tamo nalazi pas, operator ne radi ništa. Možete pretpostaviti da su r, s iz skupa {1, ..., 8}. Operator vraća true ako je dodao ili uklonio prepreku, a false ako nije promijenio poligon.
- Operator (), koji djeluje ovako: neka je P poligon, a s string. Ako je u s zapisan neki broj iz skupa {1, 2, 3, 4}, onda P(s) vraća uređeni par (r, s) koordinata na kojima se nalazi odgovarajući pas. Ako je s neki drugi string, P(n) vraća uređeni par (0, 0).
- Operator >, koji se poziva ovako: P > x, gdje je P poligon, a x=(p, n) uređeni par int-ova. Ako je n>0, onda pas p treba trčati udesno za n polja, sve dok ne naleti na prepreku ili rub poligona, kada staje. Ako je n < 0, pas treba trčati lijevo. Ako je n = 0, pas se ne miče. Psi mogu u biti u isto vrijeme na istom polju poligona. Možete pretpostaviti da je n iz skupa {-8, -7, ..., 7, 8}. Omogućite ulančavanje ovog operatora. **Ovaj operator obavezno implementirajte kao virtualnog člana klase Poligon.**
- Operator ^, koji djeluje kao i operator >, ali pomiče psa prema gore ako je n>0, a prema dolje ako je n<0. **Ovaj operator obavezno implementirajte kao virtualnog člana klase Poligon.**
- Operator ==, koji vraća djeluje ovako: ako su P i Q dva poligona, onda P==Q vraća true ako se psi nalaze na istim mjestima u P i u Q. Pri tome je bitno samo da postoji neki pas na istom mjestu, ne moraju imati istu oznaku (tj. na istim koordinatama u Q se može nalaziti bilo koja permutacija pasa iz P).
- Prefix operator ++, koji radi sljedeće: ako pas s brojem 1 može trčati udesno (naravno, ne kroz prepreke) tako da dođe na poziciju na kojoj je neki drugi pas, onda ++P pomiče psa s brojem 1 na najdesniju takvu poziciju. Inače, promatra se pas broj 2, pa pas 3, pa pas 4. Slično, u prefix operatoru -- psi trebaju trčati ulijevo do najljevije takve pozicije, u postfix operatoru ++ prema gore do najgornje takve pozicije, u postfix operatoru -- prema dolje do najdonje takve pozicije.
- Operator !, koji djeluje ovako: ako je P poligon, onda !P nizom poziva operatora > i ^ dovodi pse u poziciju u kojoj su svi smješteni na različitim poljima unutar nekog 2x2 podkvadrata na

poligonu (na primjer, kao u inicijalnoj poziciji). Operator vraća bool, i to true ako je pse moguće dovesti u takav položaj, a false inače. Konačni položaj mora biti odabran tako da zahtijeva ukupno najmanji mogući broj poziva gore navedenih operatora. (Uočite da unutar funkcije operator! smijete deklarirati druge varijable tipa Poligon i na njima pozivati operatore; ti se pozivi ne broje.) **Ovaj operator mora biti član klase Poligon.**

Napomene:

- Operatore (osim !, ^, >) smijete definirati bilo kao članove, bilo kao friend funkcije klase Poligon.
- Funkcije i operatore označite sa const svugdje gdje je to primjereno.
- Operator ! će se pozivati samo u 2 testna primjera. Pri tome, u jednom od njih će pse biti moguće dovesti u traženi položaj pomoću **samo 4 ili manje** poziva operatora ^ i >. U oba testna primjera, operator ! će biti pozvan za 3 različita poligona.
- Operatore morate implementirati tako da bude omogućeno brojanje broja poziva, kao u primjeru dolje (ako ih implementirate „kao na vježbama“, to će biti ispunjeno).
- Vaš program smije koristiti maksimalno 512MB memorije.
- Smijete definirati i druge pomoćne funkcije i operatore.

Primjer datoteke main.cpp:

```
#include <iostream>
#include "poligon.h"

using namespace std;

class TestPoligon : public Poligon
{
public:
    int brojPoziva;
    TestPoligon &operator>( const pair<int, int> &x )
    {
        ++brojPoziva;
        Poligon::operator>( x );
        return *this;
    }
};


int main( void )
{
    TestPoligon P;

    cout << P[ make_pair( 1, 5 ) ] << endl; // ispise 1, stavi prepreku na (1, 5)
    cout << P[ make_pair( 1, 1 ) ] << endl; // ispise 0
    // 12..x...
    // 34......
    // ........
    // itd

    pair<int, int> x = P( "3" );
    cout << x.first << ", " << x.second << endl; // 2, 1

    P > make_pair( 1, 7 ); // pas 1 trci udesno za 7 polja, ali nailazi na prepreku
    ( P ^ make_pair( 2, -2 ) ) ^ make_pair( 1, -2 );
    // ....x...
    // 34......
    // .2.1....
```

```
    // ........
    // itd

    P.brojPoziva = 0;
    !P; // poziva P>>(1,n), za neki n<=-3
    // ispisuje 1 (testni main ce brojati sve operatore):
    cout << P.brojPoziva << endl;
    // ....x...
    // 34......
    // 12......
    // ........
    // itd

    Poligon Q;
    ( ( ( Q ^ make_pair( 1, -1 ) ) ^ make_pair( 2, -1 ) ) ^ make_pair( 3, -1 ) ) ^ make_pair(
4, -1 );
    // ........
    // 12......
    // 34......
    // ........
    // itd

    cout << ( P == Q ) << endl; // ispise 1

    ( P > ( make_pair( 1, 1 ) ) ) ^ make_pair( 1, -1 );
    // ....x...
    // 34......
    // .2......
    // .1......
    // ........
    // itd

    P--;
    // ....x...
    // 34......
    // ........
    // .*......
    // ........
    // itd Na poziciji * su psi 1 i 2.


    return 0;
}
```