Zadatak: "macka"

U ovom zadatku promatramo mačku koja se kreće po kartonskim kutijama i u njih dodaje, ili iz njih uzima, svoje igračke. Imamo dva moguća tipa ponašanja: pomak mačke do neke kutije, te promjena broja igračaka u kutiji pored koje se mačka nalazi. Cilj nam je definirati strukture pomoću kojih možemo zabilježiti njena takva ponašanja.

Slijed kutija koje su dostupne mački zvat ćemo `igraonica`. Klijentski program zadaje broj kutija po igraonici, te početni broj igračaka u svakoj kutiji. Uvijek će postojati barem jedna kutija (možda bez igračaka) i mačka se na početku nalazi uz prvu kutiju. Mačka se u svakom trenutku nalazi uz točno jednu kutiju, i nju ćemo u tom trenutku zvati *bliska* kutija.

Primjerice, ako je lista `brojevi` neka vezana lista cijelih brojeva $\{1, 2, 3, 4, 5\}$, tada sljedeći dio programa ispisuje `6 2 89 4 0`.

```
igraonica ig(brojevi);
macka ma;

ma || new vrijednost(6) || new skok(4)
   || new vrijednost(0) || new skok(2)
   || new vrijednost(89);

int broj = ma(ig);

// operatorom dijeljenja dohvacamo broj igracaka u kutiji s danim indeksom
cout << ig / 0 << '␣' << ig / 1 << '␣' << ig / 2 << '␣' << ig / 3 << '␣' << ig / 4 << endl;
```

Opširniji testni klijentski program nalazi se na dnu zadatka.

Potrebno je implementirati strukture `igraonica`, `ponasanje`, `skok` (podtip od `ponasanje`), `vrijednost` (podtip od `ponasanje`) i `macka`.

Struktura `igraonica` pamti kolekciju cijelih brojeva (`int`) te informaciju o trenutno *bliskoj* kutiji. Postoji konstruktor koji prima `list<int>`, i tako primljenom kolekcijom elemenata inicijalizira igraonicu. Osim toga, `igraonica` definira i operator dijeljenja (`operator/`) koji prima indeks te vraća referencu na broj spremljen u kolekciji na tom indeksu.

Struktura `ponasanje` pamti jedan cijeli broj (`int`). Taj broj zovemo argumentom ponašanja. Struktura `ponasanje` definira čisti virtualni `operator()` koji uzima jedan parametar: referencu na objekt tipa `igraonica`, i ništa ne vraća. Definira i operator pretvorbe u `int`. Taj operator vraća argument ponašanja.

Struktura `skok` je podtip strukture `ponasanje`. Nadograđuje virtualni `operator()` tako da on mijenja mačkinu poziciju na način da joj je sad *bliska* kutija čiji je indeks argument ponašanja (varijabla članica strukture `ponasanje`). Mora postojati konstruktor koji uzima cijeli broj (`int`) i njime inicijalizira argument ponašanja.

Struktura `vrijednost` je također podtip strukture `ponasanje`. Ova struktura nadograđuje virtualni `operator()` tako da on postavlja broj igrača u *bliskoj* kutiji na vrijednost koja je argument ponašanja (varijabla članica strukture `ponasanje`). Mora postojati konstruktor koji uzima cijeli broj (`int`) i njime inicijalizira argument ponašanja. Osim toga, `vrijednost` definira i operator pridruživanja koji primljeni niz znakova (`const char*`) sastavljen od nepraznog niza znamenaka (iz skupa $\{0, \ldots, 9, a, b, c, d, e, f\}$) interpretira kao zapis nekog (nenegativnog cijelog) broja u sustavu s bazom 16 te tako pročitan broj sprema kao argument ponašanja. Taj operator vraća referencu na objekt koji je odredište pridruživanja.

Struktura `macka` mora moći pamtiti pokazivače na ponašanja. Definira i `operator()` koji prima referencu na objekt tipa `igraonica` i slijedno simulira sva zapisana ponašanja. Pozivatelju vraća finalan broj igračaka u posljednjoj kutiji. Osim toga, definira i `operator||` kojim se dodaje novo ponasanje. Taj operator prima objekt tipa `ponasanje*`.

Sučelje struktura koje definirate spremite u datoteku `macka.h`. Implementaciju spremite također u `macka.h`, ili u novu datoteku `macka.cpp`. Implementacijsku datoteku predajte čak i ako je prazna, tj. ako ste implementaciju napisali u datoteci `macka.h`.

Testni primjer:

```
#include <iostream>
#include <list>
#include "macka.h"

using namespace std;

int main() {
  list <int> b = {7, 8, 9}; /* ekvivalentno: l.push_back(7); l.push_back(8); l.push_back(9); */
  igraonica ig(b);
  cout << ig / 0 << '␣' << ig / 1 << '␣' << ig / 2 << endl; /* izlaz: 7 8 9 */

  ig / 1 = ig / 0 = 2;
  cout << ig / 0 << '␣' << ig / 1 << '␣' << ig / 2 << endl; /* izlaz: 2 2 9 */

  macka ma;

  ma || new skok(2) || new vrijednost(40)
     || new skok(0) || new vrijednost(1)
     || new skok(2) || new vrijednost(20);

  vrijednost nova_vrijednost(5);
  nova_vrijednost = "1d";

  ma || new skok(0) || new skok(0) || &nova_vrijednost;

  int z = ma(ig) * int(nova_vrijednost);
  cout << z << '␣' << ig / 0 << '␣' << ig / 1 << '␣' << ig / 2 << endl; /* izlaz: 580 29 2 20 */

  return 0;
}
```

Napomene:

- Na pojedinom objektu tipa `macka` najviše će se jednom pozvati operator pozivanja (`operator()`). Pojedini objekt tipa `igraonica` bit će argument u najviše jednom takvom pozivu.
- Nigdje ne morate provjeravati smislenost parametara koji dolaze iz klijentskog dijela programa, npr. ne morate provjeravati jesu li indeksi u granicama.
- Pozivatelj nikad neće koristiti konstantne varijable bilo kojeg među spomenutim tipovima.
- Sve operatore možete definirati unutar definicija odgovarajućih struktura (ne morate koristiti globalne funkcije).
- U svojem rješenju nigdje ne morate dealocirati memoriju.

Ako imate dodatnih pitanja, javite na lmikec@math.hr