Egzamin z Programowania Obiektowego (5 II 1998)

Zad 1 (12 pkt.)

Język maszynowy pewnego procesora zawiera następujące instrukcje:

- **`MOV l,r`** `; l:=r`
- **`ADD l,r`** `; l:=l+r`
- **`SUB l,r`** `; l:=l-r`
- **`JMP instr`** ; skok bezwarunkowy do podanej instrukcji
- **`IFZERO r,instr`** ; skok do podanej instrukcji jeśli **`r=0`**, w p.p. przejście do następnej
- **`IFMINUS r,instr`** ; j.w. ale skok gdy **`r<0`**

W powyższym opisie "**`instr`**" wyznacza instrukcję, "**`r`**" oznacza wartość i może być w czterech postaciach:

- **`stała`**
- **`rejestr`** wartość zapisana w rejestrze
- **`[rejestr+stała]`** wartość w pamięci pod adresem będącym sumą wartości w rejestrze oraz stałej
- **`adres`** wartość w pamięci pod podanym adresem

W przypadku "**`l`**", które oznacza miejsce na wynik, nie jest dopuszczalna pierwsza postać, druga oznacza, że wynik znajdzie się w rejestrze a trzecia i czwarta - w odpowiedniej komórce pamięci.

Należy zaprojektować i zapisać (w **Smalltalku**) zestaw klas będący częścią systemu wspomagającego uruchamianie i testowanie programów na tej maszynie. Ma on umożliwiać przechowywanie **instrukcji** oraz **pułapek** (ang. *breakpoint*).

Program podczas działania może pobierać wartości z pamięci obliczając "**`Pamięć at: adres`**" (gdzie **`Pamięć`** to zmienna globalna a **`adres`** jest liczba całkowitą) oraz wpisywać coś do niej wykonując "**`Pamięć at: adres put: wartość`**". Podobnie odczytanie wartości z rejestru umożliwia "**`Rejestry at: rej`**" a wpisanie "**`Rejestry at: rej put: wartość`**" (**`Rejestry`** to zmienna globalna a "**`rej`**" jest obiektem klasy **`Rejestr`**). Zakładamy, że zmienne **`Pamięć`** i **`Rejestry`** oraz klasa **`Rejestr`** są dane.

Obiekty reprezentujące instrukcje powinny rozumieć komunikaty:

- **`breakpoint`**
  ustawiający na danej instrukcji pułapkę powodującą przerwanie wykonania programu po jej osiągnięciu
- **`breakpointCond: war`**
  ustawiający pułapkę przerywającą program jeśli spełniony jest podany warunek. "**`war`**" jest obiektem klasy **`Warunek`** (której **nie** trzeba pisać) i rozumie komunikat "**`oblicz`**" zwracając jako wynik wartość logiczną.
- **`clearBreakpoint`**
  usuwający pułapkę ustawiona na danej instrukcji
- **`run`**
  rozpoczynający wykonanie programu począwszy od tej instrukcji aż do jego końca lub do momentu przerwania przez pułapkę
- **`unRun`**
  rozpoczynający "cofanie wykonania programu", czyli przywracający stan sprzed wykonania tej instrukcji, następnie powtarzający to dla instrukcji, która była wykonana przed nią itd. aż do osiągnięcia instrukcji, od której rozpoczęło się wykonanie lub do zatrzymania przez pułapkę. Komunikat ten będzie wysyłany tylko do instrukcji, na której ostatnio zakończyło się wykonanie programu rozpoczęte komunikatem "**`run`**" lub "**`unRun`**".

Na każdej instrukcji może być ustawiona co najwyżej jedna pułapka.

Oprócz klas pozwalających reprezentować instrukcje programu oraz pułapki należy zaimplementować wszystkie klasy pomocnicze (oprócz standardowych), należy też napisać metody potrzebne do wykonania tych, które wymieniono powyżej. Nie trzeba jednak pisać metod potrzebnych do tworzenia programów, wystarczy określić, jakimi wartościami mają być inicjowane zmienne instancyjne obiektów. Zakładamy, że (poprawnie zbudowany) program będzie dany a jedyne zmiany będą polegały na dostawianiu (i usuwaniu) pułapek (dopuszczalne także pomiędzy kolejnymi wywołaniami **`run`** lub **`unRun`**).

Zad 2 (8 pkt.)

Zaprojektować i napisać w C++ następujące wzorce klas:

1. Wzorzec abstrakcyjnej klasy **`Macierz`** parametryzowany typem elementów macierzy kwadratowej z operacjami:

- **`"+"`** - dodawanie macierzy po współrzędnych (macierze muszą mieć takie same wymiary);
- **`"="`** - operator przypisania.
- **`element(i,j)`** - funkcja zwracająca element o indeksach (i, j)

2. Wzorzec klasy **`MacierzRzadka`** implementującej abstrakcyjna klasę **`Macierz`** z p. 1.

Obiekty tej klasy będą macierzami rzadkimi, tzn. takimi, których większość elementów jest zerem. Zakładamy, że dla obiektów typu, będącego argumentem wzorca będzie istniał konstruktor przekształcający liczbę **`0`** (zero) na element zerowy tego typu.
Implementacja powinna polegać na tym, że pamiętamy tylko elementy niezerowe macierzy (operacja **`"+"`** powinna zachowywać tę własność).
Należy przewidzieć konstruktor tworzący macierz rzadką z normalnej tablicy kwadratowej.
Można założyć, że operatory **`"+"`** i **`"="`** będą działały poprawnie tylko dla parametrów będących obiektami klasy **`MacierzRzadka`** (lub jej podklas).
Do implementacji klasy **`MacierzRzadka`** należy użyć wzorca klasy **`Słownik`** (nie trzeba tego implementować):

```
template <class K, class V>
class Słownik {
protected:
  ......
public:
  Słownik () {... };
  V At (K key);                       // zwraca wartość identyfikowana kluczem key
  void AtPut (K key, V value);        // wstawia parę (key, value) do słownika
  int Includes (K key);               // sprawdza, czy para o kluczu key istnieje w słowniku
};
```