Technische und Wirtschaftswissenschaftliche Universität Budapest
Fakultät für Elektrotechnik und Informatik
Lehrstuhl für Breitband Infokommunikation und Elektromagnetische Theorie

Grundlagen der Programmierung II.
(BMEVIIIA114)
Nachgeholter große Klausur
Studiengang Informatiker
24. Mai 2013

Name:

Neptun:

Unterschrift:

	Punktzahl
1. Aufgabe (3)	
2. Aufgabe (4)	
3. Aufgabe (4)	
4. Aufgabe (4)	
5. Aufgabe (5)	
Summe (20):	
Note:	

Bewertung:
0 - 8,5: 1
9 - 11,5: 2
12 - 14,5: 3
15 - 17,5: 4
18 - 20: 5

Die Lösungen müssen auf dem Aufgabenblatt nach den Aufgaben ausgearbeitet werden! Sie können immer voraussetzen, dass alle benötigten Eingabedaten in der vorgeschriebenen Form zur Verfügung stehen. Sie dürfen nur die C/C++-Zusammenfassungen benutzen. Rechner, Notebooks, Handys, Taschenrechner usw. sind nicht erlaubt.

Lesen Sie die folgenden Aufgaben aufmerksam durch! Für die Lösungen dürfen Sie keine STL-Container benutzen, nur wenn es explizit erlaubt ist. Geben Sie keine überflüssigen Code bzw. Funktionen an, fokussieren Sie auf die Aufgabe!

1. Aufgabe

Was gibt das folgende Programm auf die Standardausgabe aus? Geben Sie Ihre Antwort in den punktierte Bereich an!

```
#include <iostream>
using namespace std;

struct Alap {
    Alap(int i = 9)                 { cout << i << "k"; }
    Alap(const Alap&)               { cout << "c" << "z"; }
    virtual void f(int i)           { cout << "f"; }
    void operator=(const Alap& a)   { cout << "=" << "z"; }
    virtual ~Alap()                 { cout << "d"; }
};

struct Uj :public Alap {
    Uj(int a = 228, char *m = "z")  { cout << m << "K"; }
    Uj(const Uj& u) :Alap(u)        { cout << "C" << "z"; }
    void f(int i)                   { cout << "F" << i; }
    ~Uj()                           { cout << "D"; }
};

void f(Uj& u) {
    u.f(9);
}
```

```
int main() {
Uj u19(108, "v1");      cout << endl;
Alap *pa = new Uj;      cout << endl;
Uj u2 = u19;            cout << endl;
u19 = u2;               cout << endl;
pa->f(107);             cout << endl;
f(u19);                 cout << endl;
delete pa;              cout << endl;
}
```

```
.....................................
.....................................
.....................................
.....................................
.....................................
.....................................
.....................................
.....................................
```

2. Aufgabe

Gegeben sei die folgende C++-Klasse, die Bruchzahlen darstellen kann. Ergänzen Sie die Code mit zusätzlichen notwendigen Funktionen, damit die durch (*), (**) und (***) markierte Zuweisungen korrekt funktionieren! Zeigen sie für welche Anweisungen welche extra Funktionen sind gebraucht!

```
#include <iostream>
using namespace std;

class Bruch {
private:
    int num, dnum; // Bruch representiert: num / dnum
public:









};









int main(void){
    Bruch B1(3,2);              // (*) B1 = 3/2
    Bruch B2(B1);               // (*) B2 = 3/2
    B2 = B1;                    // (*) B2 = 3/2
    B1 = B2 ^ 3;                // (**) B1 = 27/8;
    cout << B1 << endl << B2;   // (***) Standardausgabe: "27/8"  "3/2"
    return 0;
}
```

3. Aufgabe

Ihres Team soll eine generische Stack-Klasse (einen Stapelspeicher) entwickeln. Die Elemente sollen dynamisch gespeichert werden (d. h. die Anzahl der Elemente ist unbegrenzt.) Die Vereinbarung wird in dem unten stehenden Deklaration berücksichtigt. Sie dürfen die Deklaration ergänzen. Die Klasse soll auch als Wertparameter an Funktionen übergeben werden können, sowie die Mehrfachzuweisung muss auch Funktionieren.

```
template <class T>
class Stack {
public:
    Stack(int n = 0);
    // Ein Stack erzeugen, mit n Stück Elemente aus der typ T;

    void push(int);
    // Neue element ins Stack platzieren
    // soll das Exception range_error() werfen wenn der Stack voll ist

    T  pop();
    // Das zuletzt platzierte Element entnehmen
    // soll das Exception range_error() werfen wenn der Stack leer ist

    bool empty() const;
    // True, falls der Stack leer ist

    void dup();
    // Dupliziert das letzte Element im Stack (platziert eine Kopie);
    // soll das Exception range_error() werfen wenn der Stack leer ist

    void swap();
    // Tauscht die zwei obersten Elemente des Stacks aus
    // soll das Exception range_error() werfen wenn der Stack leer ist

    int size() const;
    // Liefert die Anzahl der Elemente im Stack

};
```

Ergänzen Sie die obigen Deklaration so, dass die Klasse die benannten Anforderungen erfüllt!
Implementieren Sie die folgenden Member-Funktionen: `Konstruktor(en), dup(), push()`!

Geben Sie ein einfaches Beispielprogramm (was höchstens aus 10 Anweisungen besteht) an, welche demonstriert, wie die Klasse benutzt werden kann. Auf Fehlerbehandlung kann im Testprogramm (aber nicht in der Klasse) verzichtet werden.

4. Aufgabe

Nehmen Sie an, dass die Stack-Klasse der 3. Aufgabe ist fertig und funktioniert einwandfrei. Unter Verwendung dieser Klasse **deklarieren Sie eine Klasse** Rechner, die elementare Operationen (`addieren()`, `subtrachieren()`, `multiplizieren()`, `dividieren()`) mit den beiden obersten Elementen des Stacks durchführt (auf **`int`** Variablen). Die Klasse entnimmt die zwei obersten Elementen aus dem Stack, führt die jeweilige Operation durch, und speichert das Ergebnis auf dem Stack (d. h. die Operanden werden entfernt!) Sie soll auch über eine Member-Funktion `ausgabe()` verfügen, die das oberste Element auf die Standardausgabe ausgibt, läßt aber dem Wert im Stack erhalten! Es soll auch möglich sein, aus `istream`-Objekten Ganzzahlen einzulesen mit dem gewöhnlichen `>>` Operator!

Implementieren Sie die Member-Funktionen `addieren()` und `ausgabe()`, sowie den Operator `>>`. Der folgende Codeteil soll korrekt funktionieren:

```
#include <iostream>
using namespace std;

int main(){
    Rechner r;
    std::cin >> r >> r; // Eingabe: 1 2
    r.addieren();
    r.ausgabe();        // Ausgabe: 3
    return 0;
}
```

5. Aufgabe

A feladat megoldásához **használhatja az STL** elemeit! Három független nyilvántartási rendszer adatait szeretnénk integrálni úgy, hogy az bővíthető legyen, és ne tároljunk benne többszörösen (redundánsan) adatokat. Jelenleg a magánszemélyeket (`Person`) a lakossági adatbázisban (`Lako`), az olimpikonokat (`Olimpikon`) a Magyar Olimpiai Bizottság (`MOB`), az egyetemi hallgatókat (`Hallgato`) a Neptun (`Neptun`) adatbázisában tartják nyilván. Természetesen lehet olyan olimpikon, aki egyben hallgató is. A feladatanalízis során a szereplők attribútumait az alábbiakban határoztuk meg:

Person (`Person`)
- név (Text)
- születési év (egész)

Olimpikon (`Olimpikon`)
- név (Text)
- születési év (egész)
- legjobb olimpiai helyezés (egész)

MOB, Neptun, Lako
- tárolók

Hallgató (`Hallgato`)
- név (Text)
- születési év (egész)
- görgetett átlag (valós)

Olimpikon hallgató (`OlimpikonHallgato`)
- név (Text)
- születési év (egész)
- görgetett átlag (valós)
- legjobb olimpiai helyezés (egész)

Kezdeti modellünk csak minimális funkciókkal rendelkezik:

- objektumok létrehozása (minden attribútum legyen megadható a konstruktorban)
- objektumok megszüntetése
- új adat felvétele a megfelelő adatbázisba (`regisztral()`)
- az olimpikonok és a hallgatók névsorának kiírása a standard kimenetre (`MOB::listaz()`, `Neptun::listaz()`). A névsor soronként egy nevet tartalmazzon! Azon olimpikonok neve mellett, akik egyben hallgatók is, jelenjen meg az "olimpikon és hallgató" szöveg.

Tervezzen objektummodellt a feladat megoldására. Osztálydiagrammal mutassa be az egyes osztályokat és kapcsolataikat, melyen jelölje a tagváltozók és tagfüggvények láthatóságát is!
Deklarálja C++ nyelven a `MOB`, a `Person`, `Olimpikon`, a `Hallgato` és az `OlimpikonHallgato` osztályokat!
Valósítsa meg a következő tagfüggvényeket: `MOB::regisztral()`, `OlimpikonHallgato` konstruktora, és az osztályok azon metódusát, ami szükséges a MOB névsor elkészítéséhez (`MOB::listaz()`)! Működjön az alábbi kódrészlet!

```
Lako lakosok;     MOB mobadat;      Neptun neptunadat;

lakosok.add(new Szemely("Jancsi", 1950));
Olimpikon* op = new Olimpikon("juliska", 1954, 3.56);
lakosok.regisztral(op);
mobadat.regisztral(op);
OlimpikonHallgato* oph = new OlimpikonHallgato("Cseh Laci", 1987, 2008, 3.14);
lakosok.regisztral(oph);
mobadat.regisztral(oph);
neptunadat.regisztral(oph);
mobadat.listaz();
```