TENTAMEN
DVA222 - OBJEKTORIENTERAD PROGRAMMING

21 Januari 2014 kl. 8:10–13:30
Rikard Lindell, 021-151759

- Denna tentamen omfattar 7 uppgifter och totalt 42 poäng.
- Betygsgränser: 21p: 3, 30p: 4, 37p: 5
- ECTS betygsgränser: 21p: D, 26p: C, 31p: B, 37p: A
- Alla svar skall motiveras och om förutsättningar saknas skall rimliga antaganden göras.
- ***Inga hjälpmedel är tillåtna.***
- ***Påbörja varje uppgift på ett nytt papper!***

Lycka Till,
Rikard

1- ALLMÄNT (6P)

Förklara och exemplifiera följande reserverade ord (keywords) i C++:

- `static`
- `operator`
- `this`
- `delete`
- `friend`
- `protected`

2- KLASSER (6P)

Skapa en klass `Work` som representerar arbetstid (i minuter) och lönesatser (SEK/min). Denna klass skall stödja följande operationer:

```
Work* w = new Work(2, 60);          //2 SEK/min, 60 minuter
w-add(65);                          //addera 65 minuter arbetstid
w->printSalary();                   //in SEK
Work::reset(w);                     //återställ arbetstiden till noll
Work *v = new Work(3);              //3 SEK/min, 0 minuter
int r = w->compare(v);              //0:   w=v,   1:   w>v,   -1:   w<v
Work u(v);                          //kopiera v i u
```

3- ARV (6P)

Vissa objektorienterade språk stödjer multipelt arv (multiple inheritance):

a) Vad är multipelt arv? Vad är dess för- och nackdelar? Ge ett exempel.
b) Hur hanteras detta i språk som inte stödjer multipelt arv?

4- DYNAMISK BINDNING (6P)

Vad skrivs ut när nedanstående program körs? Motivera ditt svar!

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

class A {
public:
    A() {cout << "A ctor" << endl;}
    A(A& a) {cout << "A copy ctor" << endl;}
    virtual ~A() {cout << "A dtor" << endl;}
    virtual void f() {cout << "A f" << endl;}
};

class B : public A {
public:
    B() {cout << "B ctor" << endl;}
    ~B() {cout << "B dtor" << endl;}
    void f() {cout << "B f" << endl;}
};

class C : public B {
public:
    C() {cout << "C ctor" << endl;}
    ~C() {cout << "C dtor" << endl;}
    void f() {cout << "C f" << endl;}
};

class X {
public:
    X() {cout << "X ctor" << endl;}
    X(X& x) {cout << "X copy ctor" << endl;}
    ~X() {cout << "X dtor" << endl; }
    void f() {cout << "X f" << endl; }
};

X foo(A& a, X x)
{
    cout << "*** in foo" << endl;
    A aa;
    aa = a;
    X xx(x);
    cout << "*** leaving foo" << endl;
    return xx;
}
```

```
int main() {
    cout << "*** starting main" << endl;
    A * a_ptr_to_a = new A;
    B * b_ptr_to_b = new B;
    C * c_ptr_to_c = new C;
    X x1;
    X * x_ptr_to_x = &x1;
    A * a_ptr_to_b = b_ptr_to_b;
    A * a_ptr_to_c = c_ptr_to_c;
    B * b_ptr_to_c = c_ptr_to_c;

    cout << "*** calls of f #1:" << endl;
    a_ptr_to_a -> f();
    b_ptr_to_b -> f();
    c_ptr_to_c -> f();
    x_ptr_to_x -> f();

    cout << "*** calls of f #2:" << endl;
    a_ptr_to_b -> f();
    a_ptr_to_c -> f();
    b_ptr_to_c -> f();

    X x2;

    cout << "*** calling foo" << endl;
    x2 = foo( *a_ptr_to_a, x1 );

    cout << "*** back in main" << endl;
    cout << "*** delete a_ptr_to_a:" << endl;
    delete a_ptr_to_a;

    cout << "*** delete b_ptr_to_b:" << endl;
    delete b_ptr_to_b;

    cout << "*** delete b_ptr_to_c:" << endl;
    delete b_ptr_to_c;

    cout << "*** leaving main" << endl;
    return 0;

}
```

5- OPERATORÖVERLAGRING (6P)

I C++ finns två olika sätt att överlagra operatorer:

a) Beskriv dessa och ge ett exempel för var och en.

b) Vilken eller vilka sätt är möjliga att använda för följande operatorer: =, +, ++, >>, [], !

6- GENERISKA CONSTRUKTIONER (6P)

En stack är en speciell sorts lista av objekt. Det enda sättet att manipulera objekten är via metoderna `Push` och `Pop`. Metoden `Push` lägger till det nya objekt överst på stacken. `Pop` tar bort samt returnerar det översta objektet från stacken.

Skriv en generisk klass (class template) som representerar en stack. Skriv också ett program som visar hur din stack ska användas för att skapa och hantera värden på två olika typer av stackar: en med heltal som värden och en med tecken.

7- OBJEKTORIENTERAD ANALYS OCH DESIGN (6P)

Gör en objektorienterad design för spelet Tetris som går ut på att placera olika fallande block så att de passar bra ihop med varandra i realtid. Det finns sju olika slags block i spelet vilka bildas genom att fyra kvadrater eller rutor placeras intill varandra i olika mönster. Spelplanen är 10 rutor bred och 20 rutor hög. Ett block i taget av slumpvist vald typ börjar falla ned ovanifrån. Användaren kan styra detta block så länge som det faller genom att flytta blocket i sidled eller rotera blocket ett kvarts varv i taget. När ett fallande block når botten på spelplanen eller träffar på ett annat block så fastnar blocket, varpå ett nytt block böjar falla ned. På så viss staplas blocken på varandra och när de når toppen är spelet förlorat. För att förhindra detta ska man försöka bilda hela vågräta rader av rutor på spelplanen, eftersom dessa då försvinner och ovanstående raders innehåll flyttas ned.

Ta fram lämpliga klasser för detta spel. För varje klass skall metoder respektive instansvariabler beskrivas. Slutligen, ge ett exempel på vilka meddelanden, inklusive avsändar- och mottagarobjekt, som kommer att skickas då en användare trycker på tangenten för att flytta det fallande block ett steg åt vänster. Observera att du inte behöver göra någon implementation i C++. Det räcker att en lämplig objektorienterad design beskrivs och förklaras.