TENTAMEN: Objektorienterad programutveckling

Läs detta!

- *Uppgifterna är inte ordnade efter svårighetsgrad.*
- Börja varje uppgift på ett nytt blad.
- Skriv ditt namn och personnummer på varje blad (så att vi inte slarvar bort dem).
- **Skriv rent dina svar. Oläsliga svar r ä t t a s e j!**
- Programkod som finns i tentamenstesen behöver ej upprepas.
- Programkod skall skrivas i C++ och vara indenterad och kommenterad.
- Onödigt komplicerade lösningar ger poängavdrag.
- Givna deklarationer, parameterlistor etc. får ej ändras.
- Läs igenom tentamenstesen och förbered ev. frågor.

I en uppgift som består av flera delar får du använda dig av funktioner klasser etc. från tidigare deluppgifter, även om du inte löst dessa.

Lycka till!

Uppgift 1

Välj **ett** alternativ för varje fråga! Garderingar ger noll poäng. Inga motiveringar krävs. Varje korrekt svar ger två poäng.

1. Vilket beskriver interaktion mellan en aktör och ett system?
   a. användningsfall
   b. aktivitetsdiagram
   c. agent
   d. inget av ovanstående

2. Vad kan i allmänhet påvisas med testning?
   a. närvaro av fel
   b. frånvaro av fel
   c. a och b
   d. inget av ovanstående är korrekt

3. Vilket är ej ett designmönster?
   a. Operator
   b. Iterator
   c. Composite
   d. Functor
   e. Wrapper
   f. Abstract factory
   g. Adaptor
   h. inget av ovanstående

4. Vilket begrepp är mindre centralt än övriga i samband med Standard Template Library?
   a. Algorithm
   b. Polymorfism
   c. Iterator
   d. Container
   e. Generic programming
   f. inget av ovanstående

5. Tvingande protokoll innebär att alla subklasser måste implementera
   a. beteende enligt protokollet
   b. gränssnitt enligt protokollet
   c. kopieringskonstruktor och tilldelningsoperator
   d. två av a-c
   e. inget av ovanstående

6. Vilken/vilka konstruktion(er) i C++ används vanligen för att implementera delegering?
   a. komponentobjekt
   b. publikt arv
   c. privat arv
   d. två av a-c
   e. tre av a-c
   f. inget av ovanstående

(12 p)

Uppgift 2

Klassen `Die` modellerar en enkel sexsidig tärning. Man kan kasta ett tärningsobjekt med `Toss` och läsa av dess värde, d.v.s. antalet ögon som kom upp, med `Value`.

```
class Die {
public:
    Die();
    void Toss();
    int Value() const;
private:
    // data
};
```

Som ett hjälpmedel för synskadade vore det praktiskt om en tärning kunde "tala om" vilken sida som kommer upp när den kastas, t.ex. genom att avge en kort ljudsignal för varje öga. En sexa skulle alltså ge sex signaler i följd. Definiera en subklass `NoisyDie` till `Die` som implementerar detta beteende. Exempel på användning:

```
NoisyDie D1, D2(2000,50,500);
D1.Toss();                      // 3 pling hörs
cout << D1.Value();             // 3
D2.Toss();                      // 5 plong hörs
cout << D2.Value();             // 5
```

Man skall kunna välja värden på tre olika parametrar för ett objekt av den nya klassen: Ljudets tonhöjd i Hertz, längden i ms hos varje ljudstöt, samt pausen i ms mellan ljuden. I exemplet ovan kommer D2 att avge 50ms ljud av 2000Hz, med pauser på 500ms mellan ljuden. För D1 angavs inga speciella önskemål och den får därför defaultvärdena 1000Hz, 100ms längd, samt 1000ms paus. Detta skall hanteras med defaultparametrar i konstruktorn. En sinuston kan genereras i DOS med funktionen `sound`. Ljudet stängs av med `nosound`. Exempel:

```
sound( 1000 );  // börja avge en ton på 1000Hz
delay( 100 );   // låt pågå i 100ms
nosound();      // avbryt ljudet
```

(12 p)

Uppgift 3

I b och c skall en generisk listklass användas, t.ex. den i bilagan eller motsvarande.
Predikatet IsLeapYear(Y) avgör om Y är ett skottår. Ett år är skottår om det är delbart med 4 men ej med 100, eller om det är delbart med 400. År 1900 var t.ex. inte skottår medan 1996 och 2000 var skottår.

a) Definiera IsLeapYear som en funktionsobjektklass.

(3 p)

b) Definiera den generiska funktionsmallen `Exists` som tar en lista och ett predikat som parametrar och returnerar true om något element i listan uppfyller predikatet, och false annars. Funktionsmallen skall vara generisk med avseende på listans elementtyp samt predikatet. Listan får ej ändras.

(6 p)

c) Skriv ett kodavsnitt som tar reda på och skriver ut om något årtal i en heltalslista är ett skottår. Använd resultaten från a och b.

(3 p)

Uppgift 4

a) Vad skriver följande program ut? Motivera svaret!

```
class Base {
public:
    Base( int Y ) : X(Y) {}
    virtual void H() { cout << "Base::H " << X << endl; }
protected:
    int X;
};

class Sub : public Base {
public:
    Sub( int Y ): Base(Y), Z(2*Y) {}
    void H() { cout << "Sub::H " << X + Z << endl; }
private:
    int Z;
};

void F( Base X ) { X.H(); }
void G( Base & X ) { X.H(); }

void main() {
    Base B(2);
    Sub S(3);
    F( B );
    G( B );
    F( S );
    G( S );
}
```

(7 p)

forts.

4 b)
Studera följande klassdefinitioner:

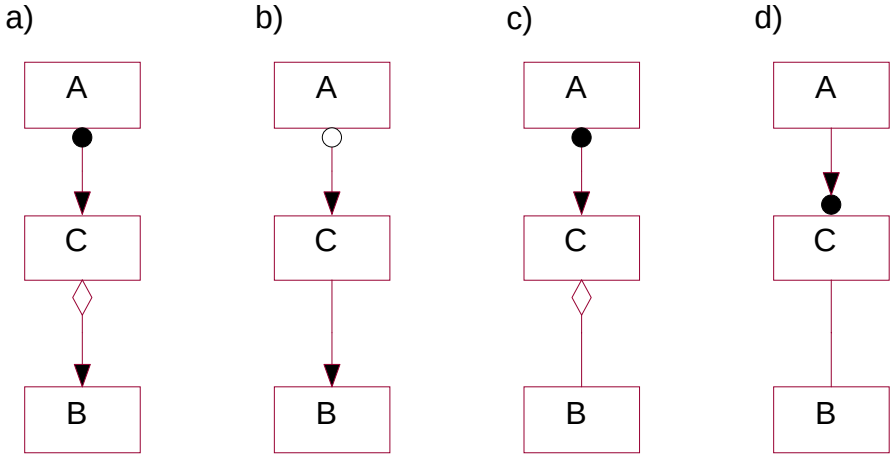
```
class C;

class B {
public:
    B( C &aC ) : theC(aC) {}
private:
    C &theC;
};

class C {
public:
    C() : Count(0), theB(new B(*this)) { }
    ~C() { if ( Count == 0 ) delete theB; }
    void Incr() { Count++; }
    void Decr() { Count--; }
private:
    B *theB;
    int Count;
};

class A {
public:
    A( C &aC ): theC(aC) { theC.Incr(); }
    ~A() { theC.Decr(); }
private:
    C &theC;
};
```

Vilket av klassdiagrammen beskriver bäst möjliga inbördes relationer mellan A, B och C? Motivera svaret!



(5 p)

Uppgift 5

Klasserna `Square` och `Circle` nedan beskriver figurobjekt som kan ritas och raderas från skärmen (hur ritning går till bortses från här). Man kan även ändra storlek på figurobjekt. De ritas ej om automatiskt vid storleksförändring. Figurerna ritas centrerade kring sin mittpunkt. Funktionerna `ZoomOutSquare` och `ZoomOutCircle` ritar och raderar resp. objekt upprepade gånger i allt mindre skala så att man får intrycket att det försvinner. Programmet är tyvärr ganska ostrukturerat. En ny `ZoomOut`-funktion måste t.ex. skrivas för varje ny figurklass.

```
struct Point {
    Point( int X, int Y ) : myX(X), myY(Y) {}
    int myX, myY;
};

class Square {
public:
    Square( int Size, Point Middle )
        : mySize(Size), myMiddle(Middle) {}
    void Resize( int NewSize ) { mySize = NewSize; }
    int GetSize() const { return mySize; }
    void Paint() { /* paint the square*/ }
    void Erase() { /* erase the square*/ }
private:
    int mySize;
    Point myMiddle;
};

class Circle {
public:
    Circle( int Radius, Point Pos )
        : myRadius(Radius), myCentre(Pos) {}
    void NewRadius( int NewRadius ) { myRadius = NewRadius; }
    int GetRadius() const { return myRadius; }
    void Draw() { /* draw the circle */ }
    void Undo() { /* undo the circle */ }
private:
    Point myCentre;
    int myRadius;
};

void ZoomOutSquare( Square Sq ) {
    while ( Sq.GetSize() > 0 ) {
        Sq.Paint();
        // delay( 10 ); // ms
        Sq.Erase();
        Sq.Resize( Sq.GetSize() - 1 );
    }
}

void ZoomOutCircle( Circle C ) {
    int R;
    for ( R = C.GetRadius(); R > 0; ) {
        C.Draw();
        // delay( 10 ); // ms
        C.Undo();
        C.NewRadius( --R );
    }
}
```

forts

```
void main() {
    Square S( 100, Point( 100, 200 ) );
    Circle C( 200, Point( 300, 400 ) );
    ZoomOutSquare( S );
    ZoomOutCircle( C );
}
```

Skriv om hela programmet! Inför en basklass för figurklasserna med tvingande protokoll för lämpliga operationer. Definiera om figurklasserna. Det räcker med en `ZoomOut`-funktion, definiera den! Namn på attribut och funktioner i de nya klasserna får väljas fritt bland namnen i de befintliga klasserna, men så att namngivningen blir så enhetlig som möjligt.

(12 p)

Bilaga

```
// List class interface (á la Weiss)
//
// Access is via ListItr class
//
// ******************PUBLIC OPERATIONS*********************
// bool IsEmpty( )        --> Return true if empty; else return false
// bool IsFull( )         --> Return true if full; else return false
// void MakeEmpty( )      --> Remove all items
// ******************ERRORS********************************


// ListItr class interface; maintains "current position"
//
// CONSTRUCTION: with (a) List to which ListItr is permanently
//     bound or
//
// ******************PUBLIC OPERATIONS*********************
// void Insert( Etype X ) --> Insert X after current position
// bool Remove( Etype X ) --> Remove X
// bool RemoveNext()      --> Remove next cell
// bool Find( Etype X )   --> Set current position to view X
// bool IsFound( Etype X )--> Return true if X would be found
// void Zeroth( )         --> Set current position to prior to first
// void First( )          --> Set current position to first
// void operator++        --> Advance (both prefix and postfix)
// bool operator+( )      --> True if at valid position in list
// Etype operator( )      --> Return item in current position
```