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. Vilken/vilka av följande är en objektorienterad metodik (OOM)?
   a. CBT
   b. Coad/Yourdon
   c. SSA
   d. JSP
   e. samtliga är OOM
   f. ingen är OOM

2. Vilken/vilka av följande aktiviteter har ej en framträdande roll vid objektorienterad design?
   a. indelning av klasser i moduler
   b. klassificering av objekt
   c. konstruktion av algoritmer
   d. beskrivning av objektrelationer
   e. fler än en av a-d
   f. samtliga a-d ingår i designfasen
   g. inget av ovanstående är korrekt

3. Vad menas med ”tvingande protokoll”?
   a. mottagaren måste svara inom viss tid
   b. ett objekt styr ett annat att utföra operationer i en viss ordning
   c. någon måste föra anteckningar på projektmötet
   d. subklasser måste implementera vissa operationer i basklassen
   e. de överenskommelser som gjorts på projektmöten måste följas
   f. fler än två av ovanstående
   g. inget av ovanstående är korrekt

4. Vilken testaktivitet är speciellt viktig efter förändringar i ett system
   a. systemtest
   b. regressionstest
   c. integrationstest
   d. enhetstest
   e. ingen av ovanstående

5. Vilket/vilka påstående karakteriserar vattenfallsmodellen
   a. varje utvecklingsfas upprepas innan nästa fas påbörjas
   b. systemet byggs ut stegvis
   c. varje utvecklingsfas avslutas innan nästa fas påbörjas
   d. de tidiga utvecklingsfaserna genomförs parallellt
   e. alla utvecklingsfaserna upprepas
   f. fler än ett av a-e
   g. inget av ovanstående är korrekt

(10 p)

Uppgift 2

En vektorklass kan användas som grund för en enkel strängklass. Uppgiften går ut på att definiera klassen `String` genom utvidgning med arv från klassen `Vector` i bilagan. `String` skall ha lämplig konstruktor samt operationerna:

```
String operator+ ( const String & rhs ) const;
friend istream & operator>> ( istream & in, String & rhs );
friend ostream & operator<< ( ostream & out, const String & rhs );
```

Vektorklassens samtliga operationer skall gå att använda även för strängobjekt. Exempel:

```
String s, t;
cin >> s >> t;
s.push_back( '-' );
t[ 2 ] = '+';
String u = s + t, v;
v = u + u;
cout << v << "," << v.size();
```

Indata `"abc 12345"` ger utskriften `"abc-12+45abc-12+45, 18".`

(12 p)

Uppgift 3

I b och c skall en generisk vektorklass samt klassen Cref användas, se bilagan.

Funktionen

```
int intMax( const Vector<int> & v ) {
    int m = INT_MIN; // "minus infinity"
    for ( int i = 0; i < v.size(); i++ )
        if ( v[ i ] > m )
            m = v[ i ];
    return m;
}
```

kan användas för att beräkna det största talet i en heltalsvektor. Om vektorn är tom returneras INT_MIN, det minsta heltalet av typ **`int`**. Funktionen har vissa nackdelar. En är att den är begränsad till heltalsvektorer. Man kan göra den generisk men då uppstår frågan om operatorn < är definierad för vektorelementen. Dessutom är det oklart vad som skall returneras om vektorn är tom, alla typer har inte självklart ett minsta värde. Funktionen ovan returnerar en kopia av det största elementet. För generella typer kan kopiering vara olämpligt, bättre är att returnera en referens direkt till vektorelementet som innehåller det största värdet.

a) Definiera funktionsobjektklassen `Greater` för jämförelser av objekt av typen `String`. Du får anta att strängklassen har `operator>` definierad.

(1 p)

b) Definiera den generiska funktionen `max` som tar en vektor och ett jämförelsepredikat som parametrar. Funktionen skall returnera en konstantreferens till något av vektorelementen som innehåller det maximala värdet (flera kan vara maximala). Lämpligt värde skall returneras om vektorn är tom. Funktionen skall vara generisk med avseende på vektorns elementtyp samt jämförelsepredikatet. Funktionens returvärden skall paketeras med hjälp av klassen `Cref`.

(6 p)

c) Skriv ett kodavsnitt som skriver ut det största elementet i ett objekt av typen

   `Vector<String>`

   Lämplig utskrift skall göras om vektorn är tom. Använd resultaten från a och b.

(3 p)

Uppgift 4

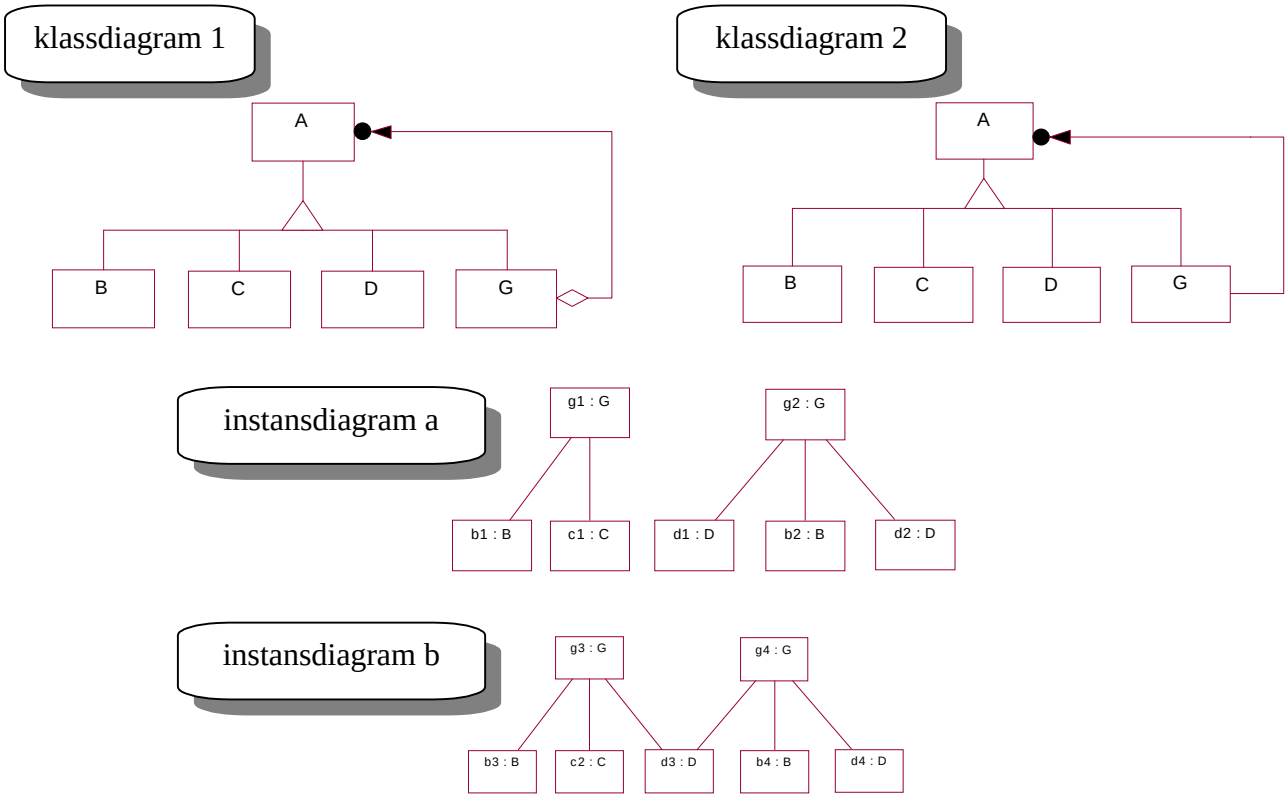
a) Ange för vart och ett av funktionsanropen numrerade 1-7 i main huruvida de är korrekta eller ej, och i förekommande fall vilken utskrift som fås? Motivera svaret!

```
class A {
public:
    void f() { cout << "A::f " << endl; }
    void g() { cout << "A::g " << endl; }
    virtual void h() { cout << "A::h " << endl; }
};

class B : public A {
public:
    void g() { cout << "B::g " << endl; }
    void h() { cout << "B::h " << endl; }
    void i() { cout << "B::i " << endl; }
};

void main() {
    B b;
    A a, *ap = &b;
    a.f();   // 1
    a.g();   // 2
    a.h();   // 3

    ap->f(); // 4
    ap->g(); // 5
    ap->h(); // 6
    ap->i(); // 7
}
```

(7 p)

forts.

4 b)
Diagrammen a och b längst ner beskriver tänkta objektinstansieringar av klassdiagrammen ovanför. I vilken mån är de objektstrukturer som beskrivs av instansdiagrammen förenliga med klassdiagrammen? Motivera svaret!
Anm. I ett instansdiagram tolkas en linje (länk) mellan två objekt som att objekten är hopkopplade på något sätt, oavsett om relationen modellerats som en association eller aggregat i klassdiagrammet.



(5 p)

Uppgift 5

Ett system för hantering av larm i t.ex. fordon skall tas fram. Larm ges som reaktioner på fel och andra notabla händelser som kan inträffa vid olika tidpunkter och på olika ställen, t.ex. i kyl- eller elsystemet i en bilmotor. I larmhanteringssystemet modelleras varje larmtyp med en egen klass. Varje objekt av en larmklass innehåller information om orsak, tid och plats för larmet. Larmobjekt kan sparas i en larmlogg för att senare behandlas i en annan del i systemet som vidtar lämpliga åtgärder. Ett exempel på utskrift av en sådan logg:

```
18:17:05: the temperature in the engine was 25 degrees below the limit 95
19:06:33: time for service of the gear box
19:58:18: the temperature in the steam boiler was 5 degrees above the limit 100
20:45:00: coffee time! Cake is served in the lounge
...
```

Uppgiften består i att programmera komponenter i systemet. Följande två färdiga klasser är givna:

Klassen `Time`

När ett objekt av klassen skapas registreras aktuellt klockslag automatiskt i objektet. Det finns operationer för att fråga efter tiden, samt skriva ut den.

```
class Time {
public:
    Time() { setTime(); }
    int getHour() const;
    int getMinute() const;
    int getSecond() const;
    void print() const;   // print time in the format hh:mm:ss
private:
    void setTime();       // save current time
    // +data representation
};
```

Klassen `Alarm`

Klassen hanterar information om tid och plats för felet men ingenting om orsaken.

```
class Alarm {
public:
    Alarm( const char *where ) {
        strcpy( thePlaceOfOccurrence, where );
    }
    Time when() const { return theTimeOfOccurrence; }
    const char *where() const { return thePlaceOfOccurrence; }
    virtual void what() const = 0;
private:
    Time theTimeOfOccurrence;
    char thePlaceOfOccurrence[ 100 ];
};
```

Operationer:

`when`	returnerar klockslaget då alarmobjektet skapades
`where`	returnerar platsen för felet
`what`	skriver ut information om när, var och vad larmet gäller

Konkreta alarmklasser definieras som subklasser till `Alarm`.

Klassen `AlarmLog`

Klassen används för att spara larmobjekt av olika slag i tidsordning för senare utskrift.

```
class AlarmLog {
public:
    void log( <lämplig typ> anAlarm ); // add an alarm object
    void printLog() const;             // print out all alarms
private:
    // välj en lämplig datarepresentation!
};
```

Krav:

- Man skall kunna spara ett i princip obegränsat antal larmobjekt med operationen log.
- Med `printLog` skall man få en utskrift av alla sparade larmobjekt enligt exemplet ovan.

Uppgifter

a) Definiera en klass för hantering av påminnelse om service (eller kafferast) genom arv från `Alarm` på lämpligt sätt. Implementera medlemsfunktionerna.

(3 p)

b) Definiera en klass för hantering av temperaturlarm genom arv från `Alarm` på lämpligt sätt. Inför ev. ytterligare dataattribut för att hålla reda på temperaturer m.m.. Implementera medlemsfunktionerna.

(6 p)

c) Gör klart klassen `AlarmLog`. Använd lämplig metod för att lagra larmobjekten. Ett anrop av `printLog` skall ge en utskrift i samma format som exemplet ovan visar.

(4 p)

d) Skriv ett program som konstruerar och loggar några olika typer av alarmobjekt i ett `AlarmLog`-objekt och därefter skriver ut loggens innehåll.

(3 p)

Bilaga

```
// Vector class interface: support bounds-checked arrays
//
// Object: must have zero-parameter constructor and operator=
// CONSTRUCTION: with (a) an integer size only
//
// ******************PUBLIC OPERATIONS*********************
// =                          --> Copying assignment
// [ ]                        --> Indexing with bounds check
// bool empty( )              --> Return true if Vector empty, false ow
// int size( )                --> Return # elements in Vector
// int capacity( )            --> Return capacity of vector
// void resize( int newSize ) --> Change bounds of Vector
// void reserve( int newCapacity ) --> Change Vector capacity
// void push_back( Object x ) --> Add x at end of Vector

template <class Object>
class Vector {
public:
    explicit Vector( int initSize = 0 );
    Vector( const Vector & rhs );
    ~Vector( );
    bool empty( ) const;
    int size( ) const;
    int capacity( ) const;
    Object & operator[]( int index );
    const Object & operator[]( int index ) const;
    const Vector & operator= ( const Vector & rhs );
    void resize( int newSize );
    void reserve( int newCapacity );
    void push_back( const Object & x );
};
```

```
template <class Object>
class Cref {
public:
   Cref( ) : obj( NULL ) { }
   explicit Cref( const Object & x )
       : obj( &x ) { }

   const Object & get( ) const {
      if( isNull( ) )
         throw "Cref: NULL pointer exception";
      else
         return *obj;
   }
   bool isNull( ) const { return obj == NULL; }
private:
   const Object *obj;
};
```