UMEÅ UNIVERSITET
Institutionen för datavetenskap

2000-01-26

Kalle Prorok,
Thomas Johansson

# Tentamen i Objektorienterad programmering för ingenjörer (TDBB09) 2000-01-29, kl 09.00 - 15.00

**Lokal**: Skrivsal 1, Östra Paviljongerna, Ålidbacken 23

**Hjälpmedel**: inga (förutom penna, suddgummi och matsäck)

**Kursutvärdering**: Sist bland papperen finns en kursutvärderingsblankett. Ta med den hem och fyll i den. Lämna in den till Thomas när du hämtar ut den rättade tentan.

Börja varje uppgift på ett nytt blad.
Använd bara en sida av papperet.
Skriv namn på varje blad.
Lägg uppgifterna i rätt ordning.
Observera att uppgifterna inte är numrerade efter svårighetsgrad och kom ihåg att även delvis lösta uppgifter kan ge poäng.
Kommentera noga all källkod så att vi kan se hur du har tänkt.

**Maxpoäng** är 40, fördelade på 7 frågor.

**Betygsgränser:**

| | |
|---|---|
| 5 | 32p |
| 4 | 26p |
| 3 | 20p |

Fråga om det är något som är oklart ! Vi kommer till skrivsalen ca **10.30** och **12.00**.

**Lycka till !**

Thomas Johansson
tel 12 71 16

Kalle Prorok
tel 13 19 29

### *UPPGIFT 1 – OO (11p)*

Thomas Programmerare har just köpt ett fint hus. Hjälp honom med att konstruera ett klassdiagram rörande delar av huset och lite allmänt om hustyper! Minst 6 klasser, varav en abstrakt, och Arv, Association, Aggregat liksom några attribut och metoder ska ingå. Beskriv din notation! (7p)

*En astrakt basklass Hus, med attribut antalVåningar och en metod flyttaIn().*
*Två klasser Villa och Hyreshus, som ärver från Hus.*
*En klass Våning, som är aggregat till Hus.*
*En abstrakt basklass Person, som har en association till Hus (kan bo där, t.ex.).*
*En klass Brevlåda, aggregat i Villa.*

Nämn 4 fördelar med att använda objektorientering! (4p)

1) *Modellering nära verkligheten -> lättförstått*
2) *Modularisering -> stabilitet*
3) *Lätt att förändra*
4) *Lätt att bygga ut*
5) *Återanvändning*
6) *Klassbibliotek -> ökad produktivitet*

### *UPPGIFT 2 – Struktur (5p)*

Operator new genererar ett undantag, bad_alloc, när det inte finns mer minne att tillgå. Skriv en funktion som kollar hur många block om 1kByte som går att allokera genom att prova och som sedan återlämnar minnesblocken. Du kan anta att det är högst 128 Mbyte minne i maskinen, och att det finns minst 1 Mbyte ledigt.

```
#include <iostream.h>
#include <except.h>


long int getMemory(void)
{
  long int antal = 0;
  int** blocks;   // pekare till 128k block a' 1024 byte
  int blockSize = 1024 / sizeof(int);  // antal int på 1024
byte

  blocks = new int*[131072L];

  try
  {
    for (long int i = 0; i < 131072L; i++)
    {
      blocks[i] = new int[blockSize];// allokera 1024 byte
```

```
      antal++;
    }
  }
  catch (...)
  { }

  for (long int i = 0; i < antal; i++)
    delete [] blocks[i];

  return antal;
}
```

*Obs ! Om du vill provköra detta program så var medveten om att bl.a. Windows NT inte tycker om att få minnet undersökt på detta sätt alls. Försiktighet ! Dessutom behövs en minnesmodell som klarar av fält större än 32 k.*

### *UPPGIFT 3 – Klasser (6p)*

Skriv en klass Cmplex (som representerar komplexa tal med real- och imaginärdel) med konstruktor, initieringskonstruktor, additionsoperator och tilldelningsoperator (=). Utskrift skall också gå att göra. Följande program skall fungera:

```
#include <iostream.h>
#include ”cmplex.h”

void main(void)
{
  Cmplex a;       // skall ge ett tal med värdet (0, 0)
  Cmplex b(3.5, 6.8);

  Cmplex c(a);
  a = a + a;

  cout << ”Summan av ” << a << ” och ” << b <<
          ” är ” << a + b << endl;
}
```

```
#include <iostream.h>

class Cmplex
{
  private:
          double re, im;

  public:
          Cmplex(double re = 0.0, double im = 0.0)
          { this->re = re;
                  this->im = im;
```

```
        }
        Cmplex(Cmplex& arg)
        {
                re = arg.re;
                im = arg.im;
        }
        Cmplex& operator=(Cmplex& arg2)
        {
                if (this == &arg2) return *this;
                re = arg2.re;
                im = arg2.im;
                return *this;
        }
        Cmplex operator+(Cmplex& arg2)
        {
                Cmplex temp;
                temp.re = re + arg2.re;
                temp.im = im + arg2.im;
                return temp;
        }
        friend ostream& operator<<(ostream& os, Cmplex arg)
        {
                os << "(" << arg.re << ", " << arg.im <<
")";
                return os;
        }
};


void main(void)
{
  Cmplex a;       // skall ge ett tal med värdet (0, 0)
  Cmplex b(3.5, 6.8);

  Cmplex c(a);
  a = a + a;

  cout << "Summan av " << a << " och " << b <<
          " är " << a + b << endl;
}
```

### *UPPGIFT 4 – Java (4p)*

a) Varför går Java-program vanligtvis långsammare än motsvarande C++-program?
b) Nämn tre saker (egenheter, finesser) som finns i C++ men inte i Java.
c) Java har något som kallas *interface*, vad är det ?
d) Hur många företag har ansvar för språket?

*a) Java-program tolkas normalt av en Java virtual machine, dvs ett program som läser bytekoden och utför instruktionerna där.*
*b) Pekare, templates, include-filer*
*c) En slags lättviktstyp, något som måste implementeras av en klass men som inte är en riktig basklass. Används i stället för multipelt arv bl.a.*
*d) Ett, SUN Microsystems, har rättigheterna till namnet, och om något annat företag vill sälja Java-system måste de följa SUN:s design av Java.*

### *UPPGIFT 5 – C++ (6p)*

a) Vad är en manipulator?
b) Vad menas med en inline-funktion? Vilka två sätt finns att skapa en sådan?
c) Varför vill man (Bjarne S) få bort #define? Och vad ska man använda istället?
d) När behövs en destruktor?

*a) specialmeddelanden som man skickar till ioströmmarna, t.ex endl, setw etc. De kan ha parametrar.*
*b) funktionskoden läggs ut istället för att anrop sker. Detta ger snabbare men större kod. Görs genom att skriva inline före funktionsnamnet eller genom att definiera funktionen direkt i klassdeklarationen. Frivilligt för kompilatorn att göra den inline. (Går t.ex ej för rekursiva fkner).*
*c) #define tal 3 ger ingen information om typen och #define max(a,b) a>b?a:b är lurigt. Använd const resp template istället är Bjarnes (och vårt) råd.*
*d) när man gjort något i klassen som behöver göras ogjort när objektet upphör att existera; exvis lämna tillbaka allokerat minne, boka av en resurs etc.*

### *UPPGIFT 6 – S T L (3p)*

Vad är en iterator och ge ett programexempel som använder en!

*En iterator är en pekarabstraktion i STL. Det finns några varianter; const, random, bi-dir, forward, input/output. I viss mån kan en vanlig pekare användas som en iterator.*
*..*
*Del av ett programexempel:*

```
std::vector<int> nums(0);
// Fill nums
...
std::vector<int>::iterator cur = nums.begin();
while(cur != nums.end())
{
std::cout << *cur; // print i(n)t to screen
cur++; // "point" to next element in nums
}
```

***UPPGIFT 7 – Templates (5p)***

Mille Mått skulle vilja ha ett klassbibliotek utvecklat åt sig för beräkningar där talen är i olika enheter. Han har tänkt använda det för konverteringar. Implementera några få typparametriserade klasser för utvärdering av hans idéer.

Följande körexempel ska fungera:

```
main()
{
  miles<int> m = 37;
  km<double> k(m); // Automatisk omvandling
  cout << k; // Ska ge utskriften: 59.2 km
}
```

Not. Om du inte vet hur man implementerar med templates kan du göra en enklare lösning men få färre poäng. Anta att en mile är 1.6 km om du inte vet det exakta värdet.

```
#include <iostream.h>
template <class T> class miles
{
        T value;
    public:
        miles(T v): value(v) {}
        T get(){ return value;}
};

template <class T> class km
{
        T value;
    public:
        km(miles<int> v): value(v.get()*1.6) {}
         friend ostream& operator<<(ostream& os,km<T> k)
         { os << k.value << " km ";}
};


main()
{
    miles<int> m = 37;
     km<double> k(m);                    // Automatisk omvandling
    cout << k; // Ger utskriften: 59.2 km
}
```

< SLUT PÅ UPPGIFTER >