Lösningsförslag

Tentamen i
Objektorienterad programmering för ingenjörer
1998-01-12, kl 16.00 - 22.00

Maxpoäng är 40 p.

Betygsgränser:

5	32p
4	26p
3	20p

Uppgift 1. (3p)

Förklara innebörden av objektorienteringens tre huvudprinciper:

a) Inkapsling
b) Arv
c) Polymorfism

a) Inkapsling: s21; Data är inkapslat i objektet så att inget annat objekt kan komma åt det förutom via de tillhandahållna operationerna. Objektet tillhandahåller ett beteende och tillstånd.
b) Arv: s22; Vi kan skapa nya klasser som ärver egenskaper och på så sätt återanvända kod och modelleringsegenskaper.
c) Polymorfism: s451; Vi vill kunna utföra samma operation (fast med olika impl) på objekt av olika typer. Dynamisk bindning löser detta i run-time.

Uppgift 2. (3p)

Det objektorienterade programspråket Java skiljer sig på flera viktiga punkter från C++. Ge minst 3 exempel.

Exempel på svar:
Java har 'Garbage Collector', dvs automatisk återlämning av minne till systemet
Java saknar pekare
Java har inget multipelt arv
Java har 16-bitars tecken
Java:s strängar kan konkateneras med '+' och avslutas inte med '\0' som i C++
Alla funktioner i Java måste vara medlemsfunktioner

Java saknar preprocessor (och därmed include-filer, #define osv)

Uppgift 3. (2p)

Förklara innebörden av dessa två deklarationssatser och speciellt vad som skiljer dem åt:

```
const int * v = (int *)0;
int * const w = (int *)0;
```

Den första satsen betyder 'en pekare till en const int på address 0', dvs det pekaren pekar på får inte ändras. Däremot kan pekaren själv ändras så att den pekar på något annat.

Den andra satsen betyder 'en konstant pekare till en int på address 0', dvs värdet på address 0 kan ändras, däremot kan pekaren inte fås att peka någon annanstans.

Följande exempel belyser detta:

```
void main(void)
{
  const int * v = (int *)0;
  int * const w = (int *)0;

  int b;
  v = &b;      // pekare ändras, går bra
  *v = 45;     // värdet på adress 0 ändras, går ej

  w = &b;      // pekaren ändras, går ej
  *w = 45;     // värdet på adress 0 ändras, går bra
}
```

Uppgift 4. (3p)

Förklara vad som menas med en abstrakt klass, vad syftet är och nämn ett sätt att göra en klass abstrakt.

Abstrakt klass: s449; En klass för vilken det inte går att instansiera objekt. Beskriver gemensamma egenskaper som skall ärvas, "definierar gränssnittet". Impl med rent virtuella funktioner eller med protected konstruktor.

Uppgift 5. (7p)

Redogör kortfattat för de 7 stegen i OMT ("aktiviteterna" enligt boken).

De 7 stegen: s 410-434
1.Finna objekt – brainstorm,vad inte hur, tillstånd och beteende, roller etc

2.Klassificera objekt – se vilka klasser objekten är förekomster av. Beskriv klassen. Knantifiering. Dyn/Stat klass?
3.Definiera relationer mellan klasser och objekt – arv,aggregat,association,användning. Stat/Dyn.
4.Gruppera klasser – i moduler. Skapa en öppen arkitektur. Skilj applikation från infrastruktur
5.Generera scenarion – Identifiera händelser och provkör. Rita händelse-/tidsdiagram
6.Definiera protokoll och attribut – definiera operationsnamn och parametrar, interna attribut
7.Verifiera systemet – dokumentera design. Testa mer detaljerad, speciellt felfall.

Uppgift 6. (4p)

Skriv en 'säker' divisionsrutin för heltal som genererar ett undantag istället för att försöka dividera med noll. Bifoga ett testprogram !

```
#include <iostream.h>

int dividera(int taljare, int namnare)
{
  if( namnare == 0 ) throw "Division med noll !";
  return taljare/namnare;
}

void main(void)
{
  int a, b;

  try
  {
      cout << "Ge täljare : ";
      cin >> a;
      cout << "Ge nämnare : ";
      cin >> b;

      cout << "a/b blir : " << dividera(a, b) << endl;
  }
  catch (char* text)
  {
      cerr << "Undantag : " << text;
  }
}
```

Uppgift 7. (3p)

Följande funktion för sortering av heltalsvektorer hittar du i din föregångares arkiv när du börjar på ditt nya jobb. Full av övertygelse om objektorienteringens förnämlighet ger du dig raskt i kast med att göra funktionen generell, så att den kan

sortera vektorer av godtyckliga objekt, dvs både andra inbyggda typer som long och float, men även egendefinierade typer. Du kommer ihåg den utmärkta föreläsningen om **templates** och inser att de är en förutsättning för att få poäng på denna uppgift.

Du kan utgå från att jämförelseoperatorn < finns definierad för alla objekt som skall sorteras, samt även kopieringskonstruktor och tilldelningsoperator.

```
#include <iostream.h>

const int FALSK = 0;
const int SANN = ! FALSK;

// sortera vektor i stigande ordning
//(Bubble sort är inte någon effektiv sorteringsmetod för stora
// datamängder, däremot är den rätt effektiv i programmerartid
// eftersom det är lätt att komma ihåg den.)

template <class Typ>
void bubble(Typ* vektor, int storlek)
{
  int klar = FALSK;                          // vi är inte klara än

  while(! klar)                              // loopa tills klar
  {
    klar = SANN;                             // kanske …
    for(int i = 0; i < storlek - 1; i++)     // gå igenom vektorn
    {
      if(vektor[i] > vektor[i + 1])          // om i fel ordning
      {
        Typ temp = vektor[i];                // byt plats
        vektor[i] = vektor[i + 1];
        vektor[i + 1] = temp;
        klar = FALSK;                        // och signalera
      }
    }
  }
}

void main(void)
{
  float v[] = { 1.0, 3.0, 2.0, 45.0, 34.4 };
  int s = sizeof(v) / sizeof(float);

  cout << "Före : ";
  for(int i = 0; i < s; i++) cout << v[i] << " ";
  cout << endl;

  bubble(v, s);

  cout << "Efter : ";
  for(i = 0; i < s; i++) cout << v[i] << " ";
  cout << endl;
}
```

Uppgift 8. (7p)

För att kunna hantera heltal som är större än vad som ryms i C++:s int och long int skall du skriva en klass **BigNum** som kan hantera tal med många siffror. Objekt i klassen skapas med det maximala antalet siffror som argument till konstruktorn. Det skall också finnas en konstruktor som initierar talet till önskat värde från ett **int**-argument, i annat fall skall värdet vara 0. Observera att för att göra det hela lite enklare behöver klassen bara kunna hantera **positiva** heltal. Kom ihåg att kommentera källkoden noggrant !

Följande skall också finnas:

In- och utmatning med **<<** och **>>.**

En **additionsoperator**, så att det går att lägga ihop två BigNum:s. Det skall också gå att lägga ihop ett **BigNum** med en **int**. Obs att det skall gå att skriva både **b + i** och **i + b,** där b är ett **BigNum** och i är en **int**. Resultatet skall vara ett BigNum.

Tilldelningsoperator och **initieringskonstruktor**.

Det är lämpligt att lagra talen i en vektor eller i en sträng, med varje siffra för sig. ASCII eller ANSI teckenuppsättning kan förutsättas.

Följande program skall fungera:

```
#include <iostream.h>
#include "bignum.h"

void main(void)
{
  BigNum tal1(100, 5750);          // initierar tal1 till 5750
  BigNum tal2(100);                // tal2 är 0
  BigNum summa1(101);
  BigNum summa2(101);

  cout << "Skriv in ett tal: ";
  cin >> tal2;

  summa1 = tal1 + tal2 + 5;
  summa2 = 6 + tal2;

  cout << "Summorna är: " << summa1 << " " << summa2 << endl;
}
```

Du behöver inte tänka på att programmet skall vara effektivt skrivet. Det behöver inte heller gå att utöka ett BigNum:s storlek efter det att det skapats.

BigNum: Programförslag (som slösar med minne & cpu-tid...);

```
//-----------------------------------------------------------------
-----------
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>

class BigNum
{
public:
  BigNum(int size, int val=0);
  BigNum(BigNum& b);
  BigNum operator=(BigNum b); // No reference..
  ~BigNum();
  friend BigNum operator+ (BigNum b1, BigNum b2);// No refs
  friend BigNum operator+ (int i1, BigNum b2);
  friend BigNum operator+ (BigNum b1, int i2);
  friend ostream& operator<< (ostream& os, const BigNum& b);
  friend istream& operator>> (istream& is, BigNum& b);
private:
  int *n; // int* easier to manage than char* with <<, >>
  int sz;
};

BigNum::BigNum(int size, int val)
{
  n = new int[sz = size];
  for (int i = 0; i < sz; i++)
  {
     n[i] = val % 10;
     val /= 10;
  }
}

BigNum::BigNum(BigNum& b)
{
  n = new int[sz = b.sz];
  for (int i = 0; i < sz; i++)
  {
    n[i] = b.n[i];
  }
}

BigNum::~BigNum()
{
  delete [] n;
}

BigNum BigNum::operator=(BigNum b)
{
  delete [] n;
```

```
  n = new int[sz = b.sz];
  for (int i = 0; i < b.sz; i++)
      n[i] = b.n[i];
  return *this;
}

BigNum operator+(BigNum b1, BigNum b2)
{
  if (b1.sz >= b2.sz)
  {
    BigNum sum(b1.sz + 1, 0);// Set to zero
    for (int k = 0; k < b1.sz; k++) sum.n[k] = b1.n[k];
    int minne = 0;
    for (int i = 0; i < b2.sz; i++)
    {
      minne += b1.n[i] + b2.n[i];
      sum.n[i] = minne % 10;
      minne /= 10;
    }
    sum.n[b2.sz] += minne;
    return sum;
  } else
  {
    return b2 + b1; // Häpp!
  }
}

BigNum operator+(int i1, BigNum b2)
{
  BigNum t(20,i1);
  return t + b2;
}

BigNum operator+(BigNum b1, int i2)
{
  BigNum t(20,i2);
  return b1 + t;
}

ostream& operator<<(ostream& os, const BigNum& b)
{
  int j=b.sz - 1;
  while (j >= 0 && b.n[j] == 0) j--; // skip leading zeroes
  for (int i = j; i >= 0; i--)
    os << b.n[i];
  return os;
}

istream& operator>>(istream& is, BigNum& b)
{
  char *tmp = new char[b.sz + 1];
  for(int k = 0;k < b.sz; k++) b.n[k] = 0;
  is >> tmp;
```

```
  int i = 0,len = strlen(tmp)-1;

  while (i < b.sz && len >= 0)
  {
    b.n[i++] = tmp[len--] - '0'; // conv from ascii
  }
  delete [] tmp;
  return is;
}

int main(int argc, char **argv)
{
    BigNum tal1(100,5750);
    BigNum tal2(100);
    BigNum summa1(101);
    BigNum summa2(101);

    cout << "Skriv in ett tal: ";
    cin >> tal2;
    summa1 = tal1 + tal2 + 5;
    summa2 = 6 + tal2;
    cout << "Summorna är " << summa1 << " " << summa2 << endl;

    char end[10];
    cin >> end;
     return 0;
}
//-------------------------------------------------------------------
-----------
```

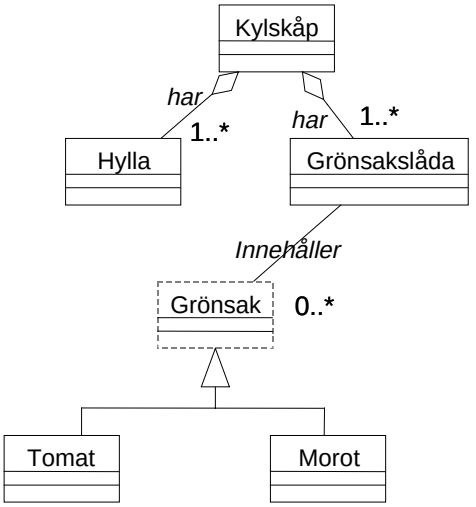
Uppgift 9. (4p)

När du kommer hem från dagens föreläsning i OOP upptäcker du att din tioåriga dotter byggt om kylskåpet till ett värmeskåp som hemuppgift från skolan. Eftersom den frysta middagen långsamt tinar i platskassarna beslutar du dig för att återställa allt till utgångsläget för att sedan köpa ditt barn ett eget kyskåp att experimentera med. Medan du sitter på golvet och funderar på de olika bitarna så dyker ide'n upp om att detta skulle vara ett utmärkt exempel att träna klassdiagram på.

Rita alltså ett klassdiagram över ditt kylskåp med innehåll. Kom ihåg att förklara vilken notation du använder för diagrammet (Booch, OMT eller UML) och förklara också vad symbolerna betyder. Diagrammet måste innehålla minst ett vardera av aggregat, association, arv och abstrakt basklass. Det måste också finnas minst fem olika klasser.

Förslagsvis nedanstående med den nya UML notationen:

Romb betyder aggregat, Triangel arv och linje association. Grönsak är en abstrakt basklass.



Uppgift 10. (4p)

Följande program i C++ är givet:

```
#include <iostream.h>
class K
{ public:
    K();                            // konstruktor
    K( const K& k );                // initieringskonstruktor
    ~K();                           // destruktor
    K& operator=( const K& k );     // tilldelning
};

K::K(){ cout << "Konstruktor" << endl; }
K::K( const K& k ){ cout << "Initiering" << endl; }
K::~K(){ cout << "Destruktor" << endl; }
K& K::operator=( const K& k )
{ cout << "Tilldelning" << endl;
  return *this; }

K func( K arg )
{ K lokalt(arg);
  return lokalt;
}

void main()
{ K x, y;
```

```
  cout << "Programmet startar" << endl;
  x = func( y );
  cout << "programmet avslutas" << endl;
}
```

a) Vad skrivs ut när programmet körs ? Förklara varje utskrift, dvs vilket objekt som skapas, tilldelas osv.

Konstruktor	Konstruerar x
Konstruktor	Konstruerar y
Programmet startar	Cout i main
Initiering	Initierar arg
Initiering	Initierar lokalt
Initiering	Initierar tempvariabel
Destruktor	Destruerar lokalt
Destruktor	Destruerar arg
Tilldelning	X = temp
Destruktor	Destruerar temp
programmet avslutas	Cout i main
Destruktor	Destruerar y
Destruktor	Destruerar x

b) Ge en tumregel för när man behöver egendefinierad tilldelning och initiering.

När man använder dynamiskt allokerat minne (med new) i sitt objekt.