Tentamen i TDP004
Objektorienterad Programmering
Teoretisk del

Datum:	2010-04-07
Tid:	8-12
Plats:	SU-salar i B-huset.
Jour:	Per-Magnus Olsson, tel 285607 Jourhavande kommer att besöka skrivsalarna ungefär varje timme under skrivtiden.
Hjälpmedel:	Teoretisk del: Inga. Praktisk del: Den C++ information som finns i systemet.
Betygsättning:	Max antal poäng: 44 med 22 poäng vardera på teori och praktikdel.

Poäng	**Betyg**
38-44	**5**
31-37	**4**
24-30	**3**
0-23	**U**

Anvisningar: Börja med den teoretiska delen. När du är klar med den lämnar du in den och får den praktiska delen. När du har lämnat in den teoretiska delen kan du inte återvända till den.

Skriv svaret på varje teoretisk uppgift på ett separat blad.

Uppgifterna är inte ordnade efter svårighetsgrad.

Lycka till!

TDP004 Objektorienterad Programmering
Teoretisk del

1. a) I kursen har vi gått igenom << respektive >>. Vad användes dessa till? (2p)
   b) Ge exempel på användning av ovanstående. (2p)
   c) Det finns även andra användningsområden för ovanstående operatorer. Vilka är dessa, och ge ett exempel på ett sådant. (2p)

2. Betrakta nedanstående fullt giltiga C++kod.

```
void f(unsigned char& a)
{
     (a > 0) ? a++: a--;
}

unsigned char i = 0;
unsigned char j = 1;

f(i);
f(j);
```

   Villka värden har `i` och `j` nu? (2p)

3. a) Referenser som är deklarerade `const` är ett sätt att både få bra prestanda och skapa säkrare kod. Förklara varför detta uppnås genom `const`-deklarerade referenser ? (2p)
   b) Man kan tänka sig att använda sig pekare deklarerade `const` för att åstadkomma samma sak. Ge ett exempel på en pekare där `const`-deklarationen gör så att man inte kan ändra det pekaren pekar på. (2p)

4. Nyckelordet `static` kan användas på olika sätt. Vad åstadkommer den static-deklarerade variabeln i koden nedan? (2p)

```
int test::add_data(int a, int b, int c)
{
     static int x = 0;
     x++;
     return (a + b +c);
}
```

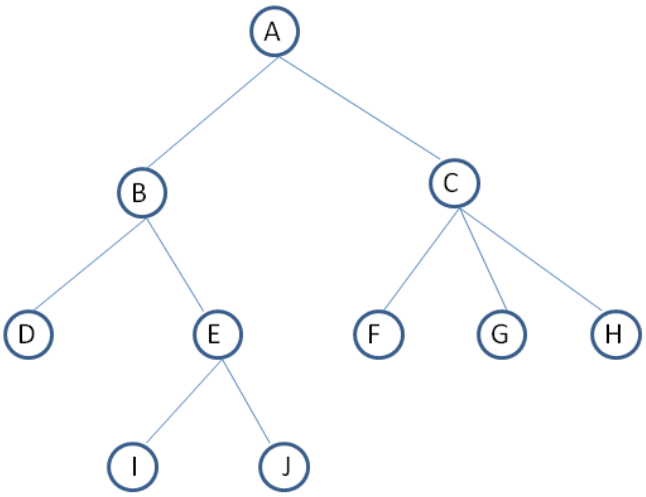
5. I större projekt är det vanligt att en kodstandard används. Vilka fyra saker tycker du är viktigast i en kodstandard? Du får ett poäng per sak samt en poäng per sak för bra motivering. (8p)

TDP004 Objektorienterad Programmering
Praktisk del

1. a) Skriv en basklass `Shape`, samt tre subklasser till denna. De tre subklasserna ska ha namnen `Pyramid`, `Rectangular_box` och `Ellipsoid`. Basklassen har tre olika medlemsvariabler av typen `double`: `width`, `depth` och `height`. Värden på dessa tas som inparametrar till subklassernas konstruktorer. Implementera även funktionen `virtual double calculate_volume() const = 0`, vilken beräknar och returnerar formens volym med hjälp av nedanstående formler och medlemsvariablerna. För `pyramid` är volymen V = (width* depth*height) / 3, för `rectangular_box` är volymen V = (width* depth*height) och för `ellipsoid` är $V = \frac{4\pi\ (width*depth*height)}{3}$. Konstanten π kan i det här fallet uppskattas till 3,14. (9p)

   b) Gör ett testfall där du visar att polymorfismen fungerar. (3p)

2. Träd är viktiga datastrukturer inom programmering och används för en mängd olika ändamål. Ett träd består av noder och bågar och ett träd måste ha precis en rotnod, vilken finns ”överst i trädet”. De flesta noder har flera barn, vilket är benämningen på de noder som finns ”under” föräldranoden i trädet. Noder som inte har några barn kallas löv. I bilden nedan är nod `A` rotnod, och noderna `D, F, G, H, I` och `J` är löv. Övriga noder är interna noder, d.v.s. varken rotnod eller löv.



I `given_files` finns klasserna `Node` och `Tree`. `Node` implementerar noderna och `Tree` har en medlemsfunktion `Node* build_tree()` som skapar ett träd enligt bilden och returnerar en pekare till rotnoden.

Den uppgift är att skriva en funktion som hittar en väg från rotnoden till en given målnod, och returnerar denna i en `std::vector<std::string>`, där innehållet i varje `string` är nodens namn. Vägen ska lagras i ordning, med rotnoden först och målnoden sist. Exempel: om funktionen anropas med målnod `J` så ska svaret vara en `vector` med fyra strängar, varav den första innehåller `A`, den andra strängen innehåller `B`, den tredje `E` och den fjärde `J` (hela vägen är `A, B, E, J`). Om den givna noden inte finns i trädet ska en tom `vector` returneras.

Funktionen ska vara generell så att den klarar av andra träd än det ovanstående. Du väljer själv hur och var du implementerar din funktion: i `Node`, `Tree` eller i någon annan klass som du skapar själv. Plocka själv ut någon nod från klassen `Tree` som målnod för testning av din algoritm. Tips: med rekursion blir uppgiften lättare. (4p)

3. Katalogen given_files finns två filer med namnen `person_info.cpp/h`. Dessa är tänkta att spara information om en person, dennes adress m.m. Tyvärr har programmeraren gjort vissa fel, både modelleringsfel och missar i implementationen. Din uppgift är att hitta och rätta felen. När du ändrar något, skriv en kommentar vad du har ändrat och varför. Om du hittar saker som du skulle vilja göra om helt men inte hinner med, förklara vad det är, varför din ändring behövs och hur du tycker att det bör fungera. Uppgiften erbjuder många olika möjligheter, poäng utdelas för vettig förändring med vettig motivering. (6p)

Lösningsförslag teoretisk del

1. a) Utmatning (<<) repektive inmatning (>>).
   b) `std:.cout <<”Mata in värde” <<std::endl;`
   `std::cint >> x;`
   c) Operatorerna kan även användas till skift där << betyder skift till vänster, vilket är detsamma som en multiplikation med två. >> ger division med två.

2. Funktionen `f` ökar värdet på inparametern med ett om det är större än noll, annars minskar det med ett. Eftersom `i` och `j` är av typen `unsigned char` så kan den lagra värden i intervallet 0-255. När i minskas med ett blir värdet 255 eftersom värdet -1 hamnar utanför det giltiga intervallet och värdet går runt till det största giltiga värdet inom intervallet. Variabeln `j` får värdet 2.

3. a) Referensen skickas som inparameter/returvärde istället för hela datastrukturen, vilket minskar mängden data som måste kopieras. `const`-deklarationen gör så att mottagaren inte kan ändra på datastrukturen.
   b) Man kan t.ex. använda `const int* a;` Detta ska utläsas som att `a` är en pekare till en `const int.`
4. Den räknar antalet gånger funktionen `add_data` har blivit anropad. Eftersom `static`-variabeln `x` delas mellan alla instanser av funktionen `test::add_data` så kommer `x` att skapas när `add_data` blir anropad för första gången, men den kommer att räknas upp vid varje anrop till funktionen `add_data`, inklusive det första.

Lösnings förslag praktisk del

1. Shape inklusive underklasser i samma .h/cpp-fil för att spara utrymme.

```
class Shape
{
public:
      Shape(double width, double height, double depth);
      virtual ~Shape(void);

      virtual double calculate_volume() const = 0;
protected:
      double width;
      double height;
      double depth;
};

class Pyramid : public Shape
{
      Pyramid(double width, double height, double depth);
      ~Pyramid();
      double calculate_volume() const;
```

```
};

class Ellipsoid : public Shape
{
     Ellipsoid(double width, double height, double depth);
     ~Ellipsoid();
     double calculate_volume() const;
};

class Rectangular_box : public Shape
{
     Rectangular_box(double width, double height, double
depth);
     ~Rectangular_box();
     double calculate_volume() const;
};

//cpp-filen.
#include "Shape.h"

Shape::Shape(double width, double height, double depth)
{
     this->width = width;
     this->height = height;
     this->depth = depth;
}

Shape::~Shape(void)
{
}
```

Cpp-filen

```
Pyramid::Pyramid(double width, double height, double depth)
: Shape(width, height, depth)
{
}

Pyramid::~Pyramid()
{
}

double Pyramid::calculate_volume() const
{
     return ((width * height * depth) / 3.0);
}
```

```
//-------------------------------------

Ellipsoid::Ellipsoid(double width, double height, double
depth) : Shape(width, height, depth)
{
}

Ellipsoid::~Ellipsoid()
{
}

double Ellipsoid::calculate_volume() const
{
     return ((3.0*3.14)/4.0) *(width * height * depth);
}

//-------------------------------------

Rectangular_box::Rectangular_box(double width, double
height, double depth) : Shape(width, height, depth)
{
}

Rectangular_box::~Rectangular_box()
{
}

double Rectangular_box::calculate_volume() const
{
     return (width * height * depth);
}
```

2.
Implementation är i det här fallet i Node-klassen.

```
#include <vector>
#include <string>

class Node
{
public:
     Node(std::string name);
     ~Node(void);
     std::string m_name;
     std::vector<Node*> m_children;

     void print_path(Node* start, Node* goal);
     std::vector<std::string>
```

```
     find_path(Node* start, Node* goal);
};

cpp-filen
#include "node.h"
#include <iostream>
#include <algorithm>

Node::Node(std::string name)
{
     m_name = name;
}

Node::~Node(void)
{
}

//Skriver ut en returnerad väg
void Node::print_path(Node* start, Node* goal)
{
     std::vector<std::string> path = start->find_path
(start, goal);
     for(unsigned int i = 0; i < path.size(); i++)
     {
          std::cerr << path[i]<< std::endl;
     }
}

//Söker vägen
std::vector<std::string>
Node::find_path(Node* start, Node* goal)
{
     std::vector<std::string> path;
     if(goal == this)
     {
          path.push_back(m_name);
     }

     for(unsigned int i = 0; i < m_children.size(); i++)
     {
          path = m_children[i]->find_path(start, goal);
          if( !path.empty())
          {
               path.push_back(m_name);
               if(start == this)
               {
                    reverse(path.begin(), path.end());
```

```
                }
                return path;
            }
        }
        return path;
}
```

3. Koden redovisas nedan. Diverse ändringsförslag ges här.
   De mest uppenbara programmeringsfelen är:
   - minnesläckan i `adress`, som allokeras med `new` men inte tas bort, vilket borde ske senast i destruktorn.
   - `get_address`, vilket returnerar en pekare till `std::string`, och ger därigenom mottagaren möjlighet att ändra på privat data.

Modelleringsfel är bland annat hanteringen personnummer, vilket ju är födelsedatumet (sex siffror: ÅÅMMDD) samt en kontrollstruktur, normalt med fyra siffror. Dock har en del personer ett P istället för den först siffran (t.ex. P123). Detta går naturligtvis inte att hantera i nedanstående kod. För att hantera detta kan man tänka sig att göra en klass som hanterar och lagrar personnummer. Vidare är hanteringen av adressen i allmänhet dålig, se t.ex. ”felhanteringen” av postnummer. Ingen hänsyn tas till att ett postnummer (i Sverige) består av fem siffror, vilket gör att man kan lagra t.ex. 123465789 som postnummer. Är det nödvändigt att ha en separat struct för adress på det sätt som den används just nu? Vidare, bör man dela upp förnamnet och efternamnet i två `std::string`? Alla get-funktioner bör vara `const` och mottagaren ska inte kunna ändra returvärdet från någon funktion.

```
#include <string>

//To allow several people to share the same address.
struct address
{
      address(std::string the_street)
      {
            value = the_street;
      }
      std::string* get_address() {return &value;}

private:
      std::string value;
};

/* Stores the information about a person.
*/
class person_info
{
public:
      person_info(std::string complete_name);
      ~person_info(void);
```

```
	void set_address(const std::string the_street);

	void set_person_number(int number);

	//Only return name if the person_numbers match,
otherwise return empty string
	std::string get_name(int number);

	std::string* get_street();

	void set_city(const std::string the_city);

	void set_zip_code(int code);

	address* m_address;

private:

	int zip_code;

	std::string m_city;

	std::string name;

	int person_number;
};
```

cpp-filen.

```
#include "stdafx.h"
#include "person_info.h"
#include <iostream>

person_info::person_info(std::string complete_name)
{
	name = complete_name;
	zip_code = -1;
	person_number = -1;
}

person_info::~person_info(void)
{
}

void person_info::set_person_number(int number)
{
```

```
     if(number <= 0)
     {
          std::cout << "Invalid person number! Must be >
0!" << std::endl;
          return;
     }

     person_number = number;
}

std::string person_info::get_name(int number)
{
     if( person_number != number )
     {
          return name;
     }

     std::string empty;
     return empty;
}


void person_info::set_address(const std::string the_street)
{
     m_address = new address(the_street);
}

void person_info::set_city(const std::string the_city)
{
     m_city = the_city;
}

void person_info::set_zip_code(int code)
{
     if(code <= 0)
     {
          std::cout << "Invalid zip code! Must be > 0!" <<
std::endl;
          return;
     }

     zip_code = code;
}

std::string* person_info::get_street()
{
     return m_address->get_address();
```

```
}
```