Tentamen i TDP004
Objektorienterad Programmering
Exempellösningar

Datum:	2008-12-16
Tid:	14-18
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: 46 med 23 poäng vardera på teori och praktikdel.

Poäng	**Betyg**
39-46	**5**
31-38	**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 uppgift på ett separat blad. Uppgifterna är inte ordnade i svårighetsgrad.

Lycka till!

TDP004 Objektorienterad Programmering
Teoretisk del

1. Du har fått två versioner av destruktorn i en klass, enligt nedan. Vilken av dessa två versioner bör vara snabbast? Varför? (2p)

objectvector är en medlemsvariabel av typen std::vector som innehåller en mycket stor mängd element av typen LargeObject*.

Version 1.
```
ExampleClass::~ExampleClass()
{
    std::vector<LargeObject*>::iterator i;
    for(i = objectvector.begin(); i != objectvector.end(); i++)
    {
        delete (*i);
    }

    // I övrigt samma som den andra versionen.
}
```

Version 2.
```
ExampleClass::~ExampleClass()
{
    std::vector<LargeObject*>::reverse_iterator i;
    for(i = objectvector.rbegin(); i != objectvector.rend(); i++)
    {
        delete (*i);
    }
    // I övrigt samma som den första versionen.
}
```

2. I en header-fil finns ett interface för en viss datastruktur som du är intresserad av, enligt nedan. Du har inte tillgång till vare sig cpp-filen för klassen Stack eller någon mera information om klassen Stack_Implementation. Vilken eller vilka objektorienterad(e) principer kan detta sätt att implementera en klass sägas vara exempel på? Motivera ditt svar? (4p)

```
class Stack_Implementation;

class Stack
{
public:
    int pop();
    int peek();
    void push(int value);

private:
    Stack_Implementation* impl;
};
```

3. Funktionerna f och g enligt nedan. Vad kommer att skrivas ut på skärmen i funktionen g? Motivera ditt svar. (6p)

```
void f(int c, int& d, double* e)
{
    ++c;
    d /= c;
    e  = 0;
}

void g()
{
    double x     = 12.5;
    int y        = 2;
    int z        = 3;

    f(z, y, &x);

    std::cout << ”x = ”<< x << std::endl;
    std::cout << ”y = ”<<  y << std::endl;
    std::cout << ”z = ”<<  z << std::endl;
}
```

4a) Vad behöver du ändra i nedanstående kod om du vill byta container från std::vector till std::list? (7p)

```
double ExampleClass::find_greatest(const std::vector<double>& data) const
{
        double greatest = 0.0;
        unsigned int i = 0;
        bool firstTime = true;
        for(; i < data.size(); i++)
        {
                if ( (true == firstTime) || (data[i] > greatest) )
                {
                        firstTime = false;
                        greatest = data[i];
                }
        }
        return greatest;
}
```

4b) Vad är poängen med att argumentet till funktionen är av typen const std::vector<double>&? Vad kallas det sättet att överföra data? Vad kallas det alternativa sättet? Hur skulle funktionens argument skrivas i det fallet? (4p)

TDP004 Objektorienterad Programmering
Praktisk del

1. I filen shoesize.cpp finns funktionen Calculate_shoe_size() som begär personers skostorlek som indata och beräknar sedan den genomsnittliga skostorleken som sedan skrivs ut på skärmen. Men det finns ett **allvarligt** fel i programmet. Du har två uppgifter.

   1a) Hitta felet och beskriv ett testfall som visar när felet uppkommer. Förklara varför felet uppkommer. Testfallet behöver inte vara automatiskt och kan t.ex. bestå av den indata som orsakar felet, i den ordning som gör att felet uppkommer.
   1b) Rätta felet i koden och kontrollera att ditt testfall från A nu ger rätt resultat.

   Lämna in testfallet och den rättade koden. (3+3p)

2. Skapa en klass Zoo som sparar pekare till olika Animal i någon container från standardbiblioteket. Animal är basklassen i en hierarki av djur. Animal har subklasserna Cat och Dog. Klassen Animals konstruktor är Animal(const std::string & name). Animal har en pekare till en std::string som sparar djurets namn, och har även en rent virtuell funktion void Make_sound() som skriver ut det hundens/kattens namn ” says ” samt det ljud som den typen av djur gör (”meow!” respektive ”wow!”) på skärmen. Exempel: om en katt heter Catbert skriver Make_sound ut ”Catbert says meow!” Implementera dessa klasser och funktioner. För att visa att det du har gjort fungerar, skapa några instanser av olika sorters djur i Zoo och iterera sedan igenom djuren och anropa Make_sound() för alla instanserna. (8p)

3. En möbelfirma ska datorisera sin lagerhantering och du ska göra en del av systemet. Data om de olika möblerna sparas i objekt av typen Furniture. Din uppgift är att implementera spara-funktionen i Furniture-klassen. Denna använder medlemsvariabler vars data är parametrar till konstruktorn, dessa sparas i medlemsvariabler av samma typ som konstruktorns inparametrar. Signaturen för klassen Furnitures konstruktor ser ut enligt:

   ```
   Furniture(const std::string type,
             const std::string name,
             std::list<const std::string>& colors);
   ```

   Funktionen bool Save() const som du ska implementera ska skriva all information i en fil med namnet type_name.txt. Om öppnandet av filen misslyckas ska funktionen returnera false, annars ska den returnera true efter att informationen har skrivits. (9p)

   Exempel:

   ```
   std::string type = ”Sofa”;
   std::string name = ”Ryd”;
   ```

```
std::string color1 = ”white”;
std::string color2 = ”red”;
std::string color3 = ”black”;

std::list<const std::string> colors;
colors.push_back(color1);
colors.push_back(color2);
colors.push_back(color3);

Furniture f = Furniture(type, name, colors);
```
När funktionen Save anropas skrivs följande i filen Sofa_Ryd.txt
Type: Sofa
Name: Ryd
Colors: white, red, black,

Observera att det inte ska vara något kommatecken efter den sista färgen.

Exempellösningar teoretisk del

1. Version 2 bör vara snabbare eftersom inga element kopieras, iterationen och borttagningen sker ju från slutet av objectvector. I version 1 kopieras alla element efter det som tas bort, vilket är många enligt uppgiften. Detta beror på att i en std::vector är elementen garanterade att ligga sekventiellt efter varandra i minnet, och därför måste efterliggande element kopieras för att undvika ”hål” i det minne som är allokerat för objectvector. OBS. Det är inte hastigheten som är det viktiga här, utan att du vet hur minneshanteringen fungerar för en vector.

2. Det kan sägas vara exempel på framförallt inkapsling men även abstraktion. Inkapsling eftersom hela implementationen är gömd i Stack_Implementation och abstraktion eftersom vi inte vet hur Stack_Implementation implementerar medlemsfunktionerna. Denna implementation är ett s.k. pimpl (pointer to implementation) idiom.

3. Följande skrivs ut i funktionen g:
x = 12.5
y = 0
z = 3

I funktionen f sätts pekaren e till 0, men inte det underliggande värdet x. Enbart den lokala kopian det riktiga värdet på y ändras, pga int& d och värdet på y blir 0 pga heltalsdivisionen 2/4. Enbart den lokal kopian av z ändras (genom parametern c),

4a) Funktionen behöver ändras till:

```
double ExampleClass::find_greatest(const std::list<double>& data) const
{
	double greatest = 0.0;
	std::list<double>::const_iterator i = data.begin();
	bool firstTime = true;
	for(; i != data.end(); i++)
	{
		if ( (true == firstTime) || ((*i) > greatest) )
		{
			firstTime = false;
			greatest = (*i);
		}
	}
	return greatest;
}
```

4b) Call by reference, endast en referens till variabeln skickas, och man sparar minne genom att undvika en onödig kopiering av inparametern. Med const hindrar vi funktionen att ändra indatan. Dessutom går koden snabbare att exekvera just pga att man inte kopierar så mycket minne.

I motsatsen call by value skapas däremot en kopia av variabeln vid funktionsanropet. Syntaxen för det blir double ExampleClass::find_greatest(const std::list<double> data) const.

Exempellösning praktisk del

1.
Shoesize.cpp

```
void Calculate_shoe_size()
{
	float numberPeople = 0;
	float totalSize = 0.0;
	float size;
	int averageSize;

	do
	{
		numberPeople++;

		cout << "Enter the person's shoe size: ";
		cin >> size;

		totalSize += size;
	}
	while (size > 0);

	if (numberPeople > 0)
	{
		averageSize = static_cast<int>(totalSize / numberPeople);
	}
	else
	{
		averageSize = 0;
	}

	cout << "The number of people is " << numberPeople << endl;
	cout << "The average shoe size is " << averageSize << endl;
}
```

1a)
Input: 16 14 0
Förväntat resultat: 15
Resultat: 10

Detta beror på att antalet personer räknas upp även om storlek är 0, d.v.s. antalet personer blir alltid en för mycket.

1b) En lösning som kräver väldigt lite ändring är följande. Man kan flytta totalSize += size; inuti if-satsen också, men det ger ingen funktionell skillnad.

```
do
{
	cout << "Enter the person's shoe size: ";
	cin >> size;

	totalSize += size;
```

```
        if(size > 0)
        {
                numberPeople++;
        }

}
while (size > 0);
```

2.

```
//Zoo.h
#include <vector>

class Animal;

class Zoo
{
public:
        Zoo(void);
        ~Zoo(void);

        std::vector<Animal*> animals;
};

//Zoo.cpp
#include "StdAfx.h"
#include "Zoo.h"
#include "Cat.h"
#include "Dog.h"

Zoo::Zoo(void)
{
        Animal* animal1 = new Cat("Catbert");
        animals.push_back(animal1);

        Animal* animal2 = new Dog("Dogbert");
        animals.push_back(animal2);

        for(unsigned int i = 0; i < animals.size(); ++i)
        {
                animals[i]->Make_sound();
        }
}

Zoo::~Zoo(void)
{
        for(std::vector<Animal*>::iterator i = animals.begin(); i != animals.end(); ++i)
        {
                delete (*i);
        }
}

//Animal.h
#include <string>
```

```
class Animal
{
public:
	Animal(const std::string name);
	virtual ~Animal(void);

	virtual void Make_sound() = 0;

protected:
	std::string* m_name;

};

//Animal.cpp

#include "Animal.h"

Animal::Animal(const std::string name)
{
	m_name = new std::string(name);
}

Animal::~Animal(void)
{
	delete m_name;
}

//Dog.h

#include "Animal.h"

class Dog : public Animal
{
public:
	Dog(const std::string name);
	~Dog(void);

	void Make_sound();
};

//Dog.cpp
#include "Dog.h"

#include <iostream>

Dog::Dog(const std::string name): Animal(name)
{
}

Dog::~Dog(void)
{
}

void Dog::Make_sound()
{
	std::cout<<*m_name <<" says wow!"<< std::endl;
```

```
}
```

//Cat.h

```
#include "Animal.h"

class Cat : public Animal
{
public:
	Cat(const std::string name);
	~Cat(void);

	void Make_sound();
};
```

//Cat.cpp

```
#include "Cat.h"

#include <iostream>

Cat::Cat(const std::string name) : Animal(name)
{
}

Cat::~Cat(void)
{
}

void Cat::Make_sound()
{
	std::cout<< *m_name <<" says meow!"<< std::endl;
}
```

I någon test-funktion i Zoo.

```
Animal* animal1 = new Cat("Catbert");
animals.push_back(animal1);
Animal* animal2 = new Dog("Dogbert");
animals.push_back(animal2);

for(unsigned int i = 0; i < animals.size(); ++i)
{
	animals[i]->Make_sound();
}
```

Ger utskrften:
->
Catbert says meow!
Dogbert says wow!

3.
//Furniture.h

```
#include <vector>

class Furniture
```

```
{
public:
	Furniture(const std::string type,
		  const std::string name,
		  std::list<std::string>& colors);

private:
	std::string m_type;

	std::string m_name;

	std::list<std::string> m_colors;
};

//Furniture.cpp
Furniture::Furniture(const std::string type,
			const std::string name,
			st:list<std::string>& colors)
{
	m_name = name;

	m_type = type;

	m_colors = colors;
}

bool Furniture::Save() const
{
	std::stringstream ss;
	ss << m_type <<"_" << m_name<< ".txt";
	std::string fileName;
	fileName = ss.str();
	std::ofstream file(fileName.c_str());

	if(NULL == file)
	{
		return false;
	}

	file  << "Type: "<< m_type <<'\n'
	      << "Name: "<<m_name <<'\n'
	      << "Colors: ";

	for(std::list<std::string>::const_iterator i = m_colors.begin(); i != m_colors.end(); i++)
	{
		file << (*i);

		//Place , after the color if it is not the last color.
		/* If we have come here then it we know that at
		 least one element in m_colors exists. */
		if( (*i) != m_colors.back())
		{
			file <<", ";

		}
	}
```

```
        file<<'\n';
        file.close();
        return true;
}
```