Tentamen i TDP004
Objektorienterad Programmering
Lösningsförslag

Datum:	2011-08-22
Tid:	14-18
Plats:	SU-salar i B-huset.
Jour:	Per-Magnus Olsson, tel 281456 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: 42 med 21 poäng vardera på teori och praktikdel.

Poäng	**Betyg**
36-42	**5**
29-35	**4**
22-28	**3**
0-21	**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) I vissa sorters korsord associeras varje bokstav med en siffra. I alla rutor där siffran finns ska den associerade bokstaven stå. Exempel: bokstaven B associeras med siffran 23, vilket betyder att i alla rutor i korsordet där siffran 23 står, ska bokstaven B skrivas. Naturligtvis får varje bokstav endast associeras med en siffra, och varje siffra får enbart associeras med en bokstav.

a) Vilken av de STL-containrar vi har gått igenom i kursen är lämplig för att spara ovanstående association? Ta endast hänsyn till följande fakta i den här deluppgiften samt informationen ovan: man lägger in ett begränsat antal associationer (= antalet bokstäver), antalet ”look up”/sökningar kan däremot bli mycket stort (beroende på hur duktig man är på att lösa korsord). Man vill alltid enbart ha reda på siffran som är associerad med en bokstav. Motivera ditt val av container (3p).

b) Antag nu att man även vill kunna hitta vilken bokstav som är associerad med en viss bokstav, alltså en omvänd ”look up” jämfört med a-delen (Exempel: vilken bokstav är associerad med siffran 23? Bokstaven B.). Vilken STL-container är lämplig nu? Den här containern ska naturligtvis även kunna hantera de relevanta kraven i a-delen. Motivera ditt val (3p).

2 a) När ett funktionsanrop sker och minne allokeras för lokala variabler, var allokeras det minnet? (2p)
b) När du skapar en `static`-variabel, var allokeras minnet för den variabeln? (2p)

3 En av orsakerna till att objektorientering utvecklades var återanvändning (eng. reuse) av kod. Beskriv hur detta underlättas inom objektorientering jämfört med icke objektorienterade språk. (2p)

4) a) Vad medför nyckelordet `friend`? (1p)
b) Det finns argument både för och emot varför `friend` bryter mot inkapsling. Ge ett argument för och ett emot (2p).

5) Du har följande kod, tagen från ett påhittat spel. Kodsnutten hanterar projektiler och kontrollerar huruvida de skapas inuti något föremål i omgivningen. Du kan anta att allt utom hanteringen av projektiler fungerar som avsett.

```
std::vector<Projectile*> projectiles;

projectile* p1 = new Projectile ();
```

```
   projectiles.push_back(p1);
            if(true == Environment.Intersection(p1))
            {
                        /* Error! Projectile created inside some object.
                        Delete projectile. */
                        delete p1;
            }
…
//Lots more things happen here.
…
//Program shutdown, remove all projectiles.
std::vector<Projectile*>::iterator i;
//Program crashes inside loop. WHY?
for(i = projectiles.begin(); i != projectiles.end(); i++)
{
            delete (*i);
}
```

Förklara varför programmet kraschar inuti loopen vid kommentaren och skriv kod för två möjliga olika sätt hur man kan lösa problemet. Förklara även varför dina ändringar löser problemet (6p).

Praktisk del

1. Du ska implementera ett enkelt substitutionschiffer som tar ett meddelande som inparameter och chiffrerar sedan meddelandet. Det ska även vara möjligt att dechiffrera ett meddelande, så din klass `Substitution_cipher` ska ha en `Cipher-` och en `Decipher-`funktion. Just det här substitutionschiffret tar ingen nyckel, det är förutbestämt hur en bokstav ska (de)chiffreras, se exempel nedan.

Du kan anta att meddelandet består av enbart stora bokstäver A-Z och du behöver inte bry dig om Å, Ä, Ö eller mellanslag. Chiffrering och dechiffrering görs på samma sätt: A byts ut mot Z, B byts ut mot Y och så vidare, se nedan:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Z Y X W V U T S Q R P O N M L K J I H G F E D C B A
```

När meddelandet ”TESTTEXT” chiffreras blir resultatet "GVHGGVCG" och när detta dechiffreras får man tillbaka det ursprungliga meddelandet.

Du ska läsa in meddelandet från ett kommandofönster, och den chiffrerade texten och den dechiffrerade texten ska skrivas ut i detsamma, tillsammans med en kort förklarande text.

Tips. Man kan få fram ASCII-koden för ett tecken genom:

```
char A = 'A';
int ascii_for_A = static_cast<int>(a); //ascii_for_A = 65
```

Om du behöver en ASCII-tabell så finns en sådan på sista bladet på tentamen (9p).

2. Ett visst företag har gett dig i uppdrag att implementera ett nytt system för att hantera löneberäkningar. Alla anställda hör till en av följande tre kategorier: fast anställd, timanställd eller konsult.

En fast anställd har samma månadslön varje månad oavsett hur många timmar han/hon har arbetat (ingen övertidsersättning här inte!), en timanställds lön baseras på antalet arbetade timmar medan en konsults lön i princip baseras på antalet **påbörjade** timmar, med vissa undantag. Undantaget är om konsulten jobbar mindre än 10 h en viss månad, eftersom avtalet då säger att hon/han fortfarande ska få lön för 10 h då.

Exempel: en timanställd har jobbat i 23,5 h den här månaden och får lön för alla dessa timmar. En konsult som har jobbat 23,5 h en viss månad får lön för 24 h. En konsult som jobbat 8 h får fortfarande betalt för 10 h.

En fast anställd tjänar 25000 kr/månad, en timanställd tjänar 120 kr/h och en konsult tjänar 385 kr/h.

a) Din uppgift är att implementera detta löneberäkningssystem. Tänk på objektorienterade principer i din implementation. Funktionerna som beräknar lönerna ska enbart utföra beräkningarna och returnera resultaten, inget ska skrivas ut i dem (8p).

b) Skriv ett eller flera testfall som testar din implementation och visar att den fungerar som uppgiften specificerar (4p).

Lösningsförslag teoretisk del

1.

a) Man kan t.ex. använda en `std::map` eftersom antalet inlagda element är litet, antalet sökningar kan vara stort och man vill associera en siffra och en bokstav med varandra. I en `std::map` är elementen av typen `std::pair` med nyckel och värde och detta lämpar sig väl för den här uppgiften. Uppslagning är snabbt (komplexitet $O(\log_2 n)$, där n är antalet element) pga att elementen lagras i en trädstruktur internt.

b) `std::vector` är nog det bästa valet, eftersom man vill göra en omvänd uppslagning, vilket inte stöds av `std::map`. Dock måste man skriva ett eget unärpredikat till find-funktionen för att hitta rätt element. En `std::vector` kommer dock att ge komplexitet O(n), dvs långsammare än `std::map` i uppgift a).

Det är möjligt att använda en `std::map` om man skriver en motsvarande funktion som för `vector`, men det blir lite mera jobb i det hör fallet eftersom det inte finns någon `find`-funktion för `std::map` (det sker ju normalt med indexering på nyckelvärdet). Komplexiteten blir detsamma som för `vector`

2. a) De allokeras på stacken i en stack-frame.

b) De allokeras i det statiska minnet. Man vet inte exakt när dessa variabler skapas, men är garanterade att finnas när man behöver dem.

3. Återanvändning underlättas till exempel genom inkapsling, vilket gör att man kan kapsla in implementationen av något, och detta något kan sedan användas som en byggsten i andra program.

4. a) Med `friend` kan en klass låta en funktion eller klass få tillgång till den deklarerande klassens medlemsvariabler och medlemsfunktioner, inklusive det som är deklarerat som `private`.

b) friend kan sägas bryta mot inkapsling eftersom som det ger någon annan klass tillgång till privata variabler och funktioner. Argumentet mot detta är att det är den deklarerande klassen som ger någon annan klass tillgång, och att det därför inte bryter mot inkapsling.

5. Koden kraschar därför att man gör `delete` på p1 två gånger i de fall som p1 skapas inuti ett objekt; först inuti kollisionskontrollen i `if`-satsen, och sedan längst ner då man tar bort alla projektiler. Det kan åtgärdas t.ex. genom ändra så att koden får följande utseende:

```
if(true == Environment.Intersection(p1))
{
            /* Error! Projectile created inside some object.
            Delete projectile. */
            delete p1;
}
else
{
            projectiles.push_back(p1);
}
```

d.v.s. projektilen läggs enbart till i projectiles om den inte har kolliderat.

Ett annat alternativ är:

```
if(true == Environment.Intersection(p1))
{
            /* Error! Projectile created inside some object.
            Delete projectile. */
            delete p1;
            p1 = NULL;
}
```

Eftersom det enligt standarden är OK att göra `delete` på något som är `NULL`. Det första sättet är att föredra eftersom man i det andra sättet får en del onödiga NULL-pekare i `projectiles`.

Lösningsförslag praktisk del

1. Enbart implementationen av chiffret visas här.

```
#include <string>
#include <iostream>

class Substitution_cipher
{
public:

     Substitution_cipher(void) : first_uppercase_character(Get_index('A'))

     {
         alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
         alphabet_length = static_cast<int>(alphabet.size());
     }

     ~Substitution_cipher(void)
     {
     }

     //Cipher the string in.
     std::string Cipher(const std::string& in) const
     {
         std::string outString;
         unsigned int i;

         for(i = 0; i < in.size(); i++)
         {
            if(in[i] != ' ')
            {
                outString.push_back( Get_letter(in[i]));
            }
         }
         return outString;
     }

     //Since deciphering is done in the same way as ciphering, we can use the same
function.
     std::string Decipher(const std::string& in) const
     {
            return Cipher(in);
     }

private:
     //Returns the ASCII-number for char inchar.
     int Get_index(char inchar) const
     {
            return static_cast<int>(inchar);
     }

     /*     Get the (de)ciphered letter corresponding to char inchar.
            Example: inchar = 'A';
            gives inpos = 65
            gives position = 26 - (65 - 65) - 1 = 25, i.e. the last position in
alphabet.
            alphabet[1] = 'A'.
      */
     char Get_letter(char inchar) const
```

```
	{
		int inpos = Get_index(inchar);
		int position = alphabet_length - (inpos - first_uppercase_character) - 1;
		return alphabet[position];
	}

	std::string alphabet;
	int alphabet_length; // Length of alphabet
	const int first_uppercase_character; // ASCII code for first upper case
character
};
```

2. a)

```
#include <math.h> //for ceilf
class Employee
{
public:
	Employee(void)
	{
	}

	virtual ~Employee(void)
	{
	}

	virtual float Calculate_pay(float worked_hours) = 0;
};

class Salaried_employee: public Employee
{
public:
	Salaried_employee(void)
	{
		monthly_pay = 25000;
	}

	virtual ~Salaried_employee(void)
	{
	}

	float Calculate_pay(float worked_hours)
	{
		return monthly_pay;
	}
private:
	float monthly_pay;
};

class Hourly_employee: public Employee
{
public:
	Hourly_employee(void)
	{
		pay_per_hour = 120.0f;
	}
	virtual ~Hourly_employee(void)
	{
	}

	float Calculate_pay(float worked_hours)
```

```
	{
		return pay_per_hour * worked_hours;
	}
private:
	float pay_per_hour;
};

class Counsultant: public Employee
{
public:
	Counsultant(void)
	{
	    min_hours_per_month = 10;
	    pay_per_hour = 385;
	}

	virtual ~Counsultant(void)
	{
	}

	float Calculate_pay(float worked_hours)
	{
	    if(worked_hours < min_hours_per_month)
	    {
	       return min_hours_per_month * pay_per_hour;
	    }

	    return pay_per_hour * ceilf(worked_hours);
	}
private:
	float pay_per_hour;
	float min_hours_per_month;
};
```

b) Man bör framförallt testa att implementationen av `Calculate_pay`/polymorfism fungerar. Se exempel på detta nedan.

```
Employee* con = new Counsultant();
Employee* sal = new Salaried_employee();
Employee* he = new Hourly_employee();

//Test Calculate_pay for the different classes.
float he_sal = he->Calculate_pay(100);
float con_sal = con->Calculate_pay(9);
float sal_sal = sal->Calculate_pay(200);

std::vector<Employee*> empl;
empl.push_back(con);
empl.push_back(sal);
empl.push_back(he);

//Test polymorfism
std::vector<Employee*>::const_iterator i;
for(i = empl.begin(); i != empl.end(); i++)
{
	std::cout <<”Pay is ”<<(*i)->Calculate_pay(20.5f) << std::endl;
}
```