Tentamen i TDP004
Objektorienterad Programmering
Lösningsförslag

Datum:	2008-08-14
Tid:	08-12
Plats:	PC6-PC7 i E-huset.
Jour:	Per-Magnus Olsson, tel 285607 Jourhavande kommer att besöka skrivsalarna varje timme under skrivtiden.
Hjälpmedel:	Teoretisk del: Inga. Praktisk del: En bok om C++.
Betygsättning:	Max antal poäng: 42 med 21 poäng vardera på teori och praktikdel.

Poäng	**Betyg**
36-44	**5**
30-37	**4**
23-29	**3**
0-22	**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 efter svårighetsgrad.

Lycka till!

TDP004 Objektorienterad Programmering

Teoretisk del

1. En viss funktion finns i tre versioner: A, B och C enligt nedan. Det enda som skiljer dem åt är inparametern.
A. double Calculate(LargeObject object)
B. double Calculate(LargeObject* object)
C. double Calculate(LargeObject& object)

Om LargeObject tar upp mycket minne, hur påverkas exekveringstiden för version B jämfört med version A? Snabbare, långsammare eller ingen skillnad? Varför?

Om LargeObject tar upp mycket minne, hur påverkas exekveringstiden för version C jämfört med version A? Snabbare, långsammare eller ingen skillnad? Varför? (6p)

Båda: snabbare pga mindre minneskopiering.

2. Jämför funktionerna
static int DoSomething(LargeObject& object)
int DoSomething(LargeObject& object) const
Vad är skillnaden mellan en const-funktion och en static-funktion (funktionerna är identiska i övrigt)? Kan en funktion vara både static och const? Förklara varför eller varför inte? (3 + 3p)

En const-funktion får använda men inte ändra på medlemsvariabler. Den får bara anropa andra funktioner som är const. En static-funktion existerar oberoende av alla instanser av klassen och får därför enbart använda variabler och andra funktioner som är static. Det behöver inte ens finnas någon instans av klassen för att en static-funktion ska existera. En funktion kan inte vara både const och static, detta eftersom definitionerna på en static och en const-funktion motsäger varandra.

3. Du ska använda koden för klassen Vehicle som basklass för subklassen Motorcycle. När du designar Motorcycle ser du att du kommer att använda och ändra medlemsvariablerna på många ställen, så du vill kunna göra detta utan att använda Get-funktionerna. Med nedanstående kod fungerar dock inte detta på det sätt som du skulle vilja. Varför? Hur vill du lösa det och vad får din lösning för några konsekvenser? (4p)

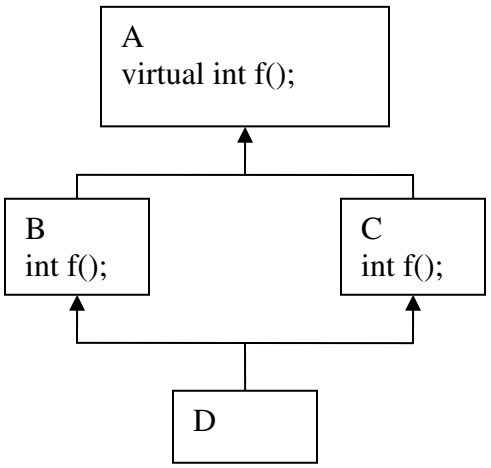
```
class Vehicle
{
    public:
        Vehicle(int vehicleId, std::string vehicleColor);
        ~Vehicle();

        void SetOwner(Person* Owner);
```

```
        Person* GetOwner() const;
        int GetVehicleID() const;
        std::string GetVehicleColor() const;
    private:
        int m_ID;
        Person* m_Owner;
        std::string m_Color;
};
```

Medlems-variabler/funktioner som är private ärvs inte till basklasser, dvs är omöjliga att komma åt direkt i basklasserna.
Ett förslag är att flytta alla medlemsvariabler som är private till att vara protected istället. Detta påverkar inte existerande interface utåt, eftersom andra, icke-ärvande klasser inte kommer åt protected data. Om andra klasser ärver från basklassen påverkas dessa på så vis att de nu får tillgång till de medlemsvariabler som tidigare var private. De kommer dock att fungera på samma sätt som tidigare, men koden kanske känns lite omständig nu när det är möjligt att direkt komma åt variablerna.

4. Vad är multipelt arv? I kursen tog vi upp ett välkänt problem med multipelt arv. Beskriv detta. (4p)
Det är när en klass ärver från mer än en klass. Problem inträffar när man har en arvshierarki enligt nedan. Båda klasserna B och C implementerar den virtuella funktionen f(), medan D inte gör det. Vad som händer D->f() anropas är odefinierat.



5. I samband med arv är det viktigt att tänka på minneshantering. Vad är det som är särskilt med minneshanteringen vid arv och hur löser man det? (2p)

Programmeraren måste se till att allt minne som subklassen allokerat dynamiskt lämnas tillbaka. Detta löses genom att ha en virtuell destruktor i basklassen. Om man inte har detta anropas inte subklassens destruktor, och det minne som subklassen har allokerat dynamiskt lämnas inte tillbaka.

Praktisk del

1. Implementera klassen Lightbulb med två public-variabler: name av typen std::string och strength av typen int. Skriv sedan en medlemsfunktion i samma klass:
   void PrintSortedLightbulbs(std::vector<Lightbulb*> lightbulbs)
   vilken först sorterar i ordning enligt följande ordning: glödlamporna ska sorteras i alfabetisk ordning på sortens glödlampa, och om det finns flera glödlampor av samma sort så ska dessa sorteras så att en med lägre styrka kommer före en med högre styrka. Efter att vectorn har sorterats ska den skrivas ut, från första till sista elementet. Ovanför utskriften av typen och styrkan ska det stå ”Type strength”. Exempel på utskrift:

   Type strength

   Diode 5
   Diode 12
   LowEnergy 10
   LowEnergy 30
   Normal 25

   (7p)

   Det är viktigt att sorteringskriteriet är korrekt eftersom det kan behövas två jämförelser för att sortera korrekt. Det är möjligt att anropa sort med andra funktioner än en statisk medlemsfunktion, detta är ett av många möjliga sätt att lösa det.

```
#include <string>
#include <vector>
#include <algorithm>
#include <iostream>

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

     std::string type;
     int strength;

     static bool Sorting(const Lightbulb* a,
                         const Lightbulb* b)
     {
```

```
            std::string typeA = a->type;
            std::string typeB = b->type;

            if(typeA == typeB)
            {
                  return (a->strength < b->strength);
            }
            else
            {
                  return typeA < typeB;
            }
      }

      void PrintSortedLightbulbs(std::vector<Lightbulb*>
      lightbulbs)
      {
            std::vector<Lightbulb*>::const_iterator i;
            //Sort vector.
            sort(lightbulbs.begin(),
                  lightbulbs.end(), Sorting);

            //Print vector
            std::cout << "Type Strength"<<std::endl;
            std::cout << "-------------"<<std::endl;

            for(i = lightbulbs.begin();
                  i < lightbulbs.end(); ++i)
            {
                  std::cout<<(*i)->type <<" "
                              <<(*i)->strength <<std::endl;
            }
      }
};
```

2. Implementera en basklass Testcase med de båda subklasserna TestAddition, TestSubtraction. Klassen TestCase ska ha en funktion Run vilken ska ta två parametrar av typen const int, och utföra lämplig operation på dessa, och resultatet av operationen ska returneras. Subklasserna ska implementera funktionen Run, och vilken operation som ska utföras kan man se i klassnamnet. Låt kompilatorn verifiera att funktionen Run inte ändrar några medlemsvariabler.

   Efter att du har implementerat dessa klasser, skapa en instans av varje subklass och lägg dessa i en std::list. Iterera genom alla instanser i listan och anropa Run för vart och ett av elementen, med av dig valda inparametrar (10p).

Enbart koden för TestCase och dess subklasser ges här.

```
class Testcase
```

```
{
public:
     Testcase(void);
     virtual ~Testcase(void);
     virtual int Run(const int a, const int b) const = 0;
};

class TestAddition : public Testcase
{
public:
     TestAddition() {};
     ~TestAddition() {};
     int Run(const int a, const int b) const
     {
          return a + b;
     }
};

class TestSubtraction : public Testcase
{
public:
     TestSubtraction() {};
     ~TestSubtraction() {};
     int Run(const int a, const int b) const
     {
          return a - b;
     }
};

class TestMultiplication : public Testcase
{
public:
     TestMultiplication() {};
     ~TestMultiplication() {};
     int Run(const int a, const int b) const
     {
          return a * b;
     }
};
```

3. Skriv en funktion int ConvertToBinary(const std::string number) som konverterar en binär representation av ett tal till en vanligt heltal. Du får förutsätta att number enbart innehåller ettor och nollor. Exempel: om ConvertToBinary anropas med en sträng innehållande 110111 så bör den returnera 55, om den anropas med 101 så ska den returnera 5 etc (5p).

```
int ConvertToBinary(const std::string number)
{
	int exponent = 0;
	int sum = 0;

	//Iterate from the end, as the least significant bit is at the end.
	for(int i = number.size() - 1; i >= 0; i--)
	{
		//We only care if the number is 1.
		if(1 == number[i])
		{
			sum += static_cast<int>(pow(2.0, exponent));
		}
		exponent++;
	}
	return sum;
}
```

Det inns många olika sätt att lösa den här uppgiften detta är ett av dem.