Tentamen i TDP004
Objektorienterad Programmering
Teoretisk del

Datum:	2011-04-28
Tid:	08-12
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) En del av följande uppgifter innehåller felaktigheter. Rätta till de felaktiga, om du tycker att någon eller några är korrekta skriver du det. (5p).
    a) int number = `new int`(3);
    b) `cin <<` new_int;
    c) `float` decimal_number == 123.4;
    d) `float` &float_ref;
    e) `double` double_number = 5678;

2) Eftersom arv är användbart, då måste väl multipelt arv vara mångdubbelt så användbart? I kursen tog vi upp ett potentiellt problem med multipelt arv. Beskriv detta och ge ett exempel på när det kan inträffa. (4p)

3) Vad används const till i C++sammanhang? (2p)

4) Nedanstående kod kompilerar men kraschar när den exekveras. Förklara varför. (2p)

```
int & calculate_average(const std::vector<int>& v)
{
     std::vector<int>::const_iterator j;
     int sum = 0;
     for(j = v.begin(); j != v.end(); j++)
     {
               sum += (*j);
     }
     return sum;
}
```

5) I C kan man bara explicit typkonvertera på ett sätt. I C++ kan man däremot typkonvertera på inte mindre än fyra olika sätt.
    a. Ge namnen på dessa fyra. (2p)
    b. Förklara två (valfria) av dessa med var sitt exempel. (2p)

6) Virtuella funktioner kan vara mycket användbart i en del sammanhang.
    a. Vad är det som skiljer en virtuell funktion och en rent virtuell funktion? Du kan förutsätta att det är vanliga medlemsfunktioner som avses. (2p)
    b. Vad har dessa för effekt på klassen där funktionerna är definierade? (2p)

Praktisk del

1) Inom matematiken finns något som kallas Maclaurinserier, döpt efter matematikern Colin Maclaurin. Med hjälp av en Maclaurinserie kan man approximera (uppskatta) olika funktioner runt x=0. För att uppskatta exponentialfunktionen e^x väldigt noggrant kan följande Maclaurinserie användas: $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$. Observera att antalet termer i serien är oändligt. Man kan också göra en lite mindre noggrann approximation genom att begränsa antalet termer, men samma formel för varje term används fortfarande. Om man vill approximera e^x med 3 termer så blir resultatet $1 + x + \frac{x^2}{2}$ och om man vill approximera med 6 termer så blir resultatet
$1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120}$.

   Skriv en funktion som gör approximationer av e^x med hjälp av Maclaurinserien enligt ovan. Din funktion ska ta antalet termer som inparameter, beräkna det generella uttrycket enligt ovan och skriva ut det totala uttrycket med alla termerna på skärmen. Det ställs inget krav att texten/termerna ska formatteras som ovan, men det måste klart framgå vad som hör till vilken term. För full poäng krävs en effektiv implementation. (6p)

   Tips! Nollfakultet (0!) är lika med 1. Givet formeln och exemplen ovan bör du kunna lista ut hur serien ska se ut med olika termer.

2) Du ska implementera ett system för att hålla ordning på böckerna i ett bibliotek. Följande information ska sparas för varje bok: titel, författarens namn, antal sidor i boken, utgivningsår, förlag samt ISBN-nummer. Ett ISBN-nummer är ett unikt nummer för varje bok (jämför med personnummer).
   a) Din uppgift är att skapa en lämplig datastruktur för att kunna spara ovanstående information om böckerna. Varje instans ska spara information om en bok. (3p)
   b) Antag att du har skapat din datastruktur enligt a-delen av uppgiften. Du ska nu välja en lämplig STL-container för att spara en stor mängd instanser som representerar böcker. Böckerna ska läggas in i din container när registret skapas och de tas i princip aldrig bort från systemet. Det som sker oftast är att en låntagare söker efter en bok med hjälp av ISBN-numret.

      Bestäm vilken STL-container är mest lämplig för registret. Skapa två instanser av din datastruktur som representerar böcker, från a)-delen av uppgiften, och lägg till dem i registret. I samma fil, skriv en kort text som motiverar ditt val av container. (3p)
   c) Skriv en funktion som låter användaren söka efter en bok med hjälp av ISBN-numret. Funktionen ska som inparameter ta ISBN-numret och vad övrigt du tycker är lämpligt.

Returvärdet är en pekare till boken. Kom ihåg att hantera fallet med att ingen bok med ISBN-numret hittades. (3p)

d) Även om det sällan händer att en bok tas bort så händer det till exempel när en slarvig låntagare tappar bort en bok. Skriv en funktion som tar bort en bok från den container du har valt. Inparameter till funktionen är ISBN-numret. (3p)

e) Biblioteket ska nu börja tillhandahålla CD-skivor för utlåning. För varje CD ska följande information sparas: artist, namnet på skivan, skivbolag som get ut skivan, utgivningsår, samt namn och längd (i minuter och sekunder) på alla låtarna på skivan. Vilka ändringar kräver detta i den datastruktur som du skapade i del a)? Finns det något som går att återanvända? (3p)

Lösningsförslag

Teoretisk del.

Uppgift 1

a) int number = `int`(3); // Ska inte vara något `new`
b) `cout` << new_int; // Ska vara <<, inte >>
c) `float` decimal_number = 123.4; //Ska vara =, inte ==
d) `float` &float_ref; // Måste initieras
e) `double` double_number = 5678; // Korrekt

Uppgift 2

Det problem dom vi har gått igenom är om man har två klasser som ärver från samma basklass, och överlagrar samma funktion men på olika sätt. Om man sedan har en subklass som ärver från båda dessa klasser så kan den överlagrade funktionen ställa till problem. Se föreläsning 6.

Uppgift 3

`const` används för att se till att ett visst minnesområde (variabel) inte kan ändras under tiden programmet körs. Kompilatorn ser till att ingen funktion skriver ett nytt värde till variabeln. Man kan naturligtvis även deklarera funktioner `const`, vilket gör att den `const`-deklarerade funktionen inte kan ändra någon klassvariabel.

Uppgift 4

Koden returnerar en referens till en lokal variabel, vars minnesutrymme finns i funktionens stack frame. När sedan funktionens scope är borta (funktionen har returnerat) så försvinner även funktionens stack frame och därmed är inte variabeln giltig eftersom det är troligt att annan information kommer att skrivas i samma minnesutrymme. Man kan ha ”tur” och kunna komma åt variabeln ändå, men det ska man absolut inte lite på.

Uppgift 5

a) `static_cast, dynamic_cast, reinterpret_cast` samt `const_cast`.
b) Ett exempel på `static_cast` är

```
float i = 4.65;
```

```
int k = static_cast<int>(i);
```

Ett exempel på `reinterpret_cast` är

```
int *a;

char *cp = reinterpret_cast<char*>(a);
```

Här tolkar vi om bitarna i a (pekare till int), till att vara en pekare till en `char`, och hoppas att det går bra. Observera att ingen felkontroll sker.

Uppgift 6.

a) En virtuell funktion deklarerar en funktion som kan överlagras. Den får implementeras i den klassen där den är deklarerad. En rent virtuell funktion deklarerar en funktion som **måste** överlagras. Den får **inte**implementeras i den klassen där den är deklarerad (gäller C++).
b) En virtuell funktion har ingen direkt effekt. Det är logiskt att virtuella funktioner används i samband med arv, men inget krav. En rent virtuell funktion gör klassen abstrakt, dvs det går inte att skapa en instans av klassen där funktionen är deklarerad.

Praktisk del

Uppgift 1.

```
void Maclaurin_series(int number_of_terms)
{
	int denominator = 1;
	for( int i = 0; i < number_of_terms; i++ )
	{
		//In the two first terms, the denominator is one.
		if(i == 0)
		{
			denominator*= (i+1);
		}
		else
		{
			denominator*= i;
		}

		//Print the result to the screen.
		cout << "x^" << i << "/" << denominator;
		if( i != number_of_terms - 1 )
		{
			cout << " + ";
		}
	}
}
```

Eftersom x^0 = 1 så är ovanstående korrekt, men det är naturligtvis också korrekt att skilja den första termen från de andra och skriva 1 istället för x^0/1.
Kravet på en effektiv implementation är för att undvika att beräkna n! för varje term, Det räcker att göra en ytterligare multiplikation per term.

Uppgift 2.

a)

```
class Book
{
public:
	Book(string title, string author,
		string publisher, string ISBN,
		int publishing_year, int number_of_pages)
	{
		m_author = author;
		m_title = title;
		m_publisher = publisher;
		m_ISBN = ISBN;
		m_publishing_year = publishing_year;
		m_number_of_pages = number_of_pages;
	};
	virtual ~Book() {};

	const string Get_title() const
	{
		return m_title;
	}

	const string Get_author() const
	{
		return m_author;
	}
```

```
	const string Get_publisher() const
	{
		return m_publisher;
	}

	const string Get_ISBN() const
	{
		return m_ISBN;
	}

	const int Get_number_of_pages() const
	{
		return m_number_of_pages;
	}

	const int Get_publishing_year() const
	{
		return m_publishing_year;
	}

protected:
	string m_title;
	string m_author;
	string m_publisher;
	int m_publishing_year;
	int m_number_of_pages;
	string m_ISBN;
};
```

b) Här är `std::map` ett lämpligt val. Exempelvis som `std::map<std::string, Book*>`. Detta eftersom uppslagningen/find är snabb eftersom en trädstruktur används internt för att lagra elementen.

c)
Om man vill ha en fristående funktion kan den se ut enligt nedan. I en klass bör naturligtvis `std::map` vara en medlemsvariabel och ej inoparameter.

```
Book* Find_book(const string& ISBN, const map<string, Book*>& c)
	{
		Book* book_pointer = NULL;
		map<string, Book*>::const_iterator i = c.find(ISBN);
		if(i != c.end())
		{
			book_pointer = i->second;
		}
		return book_pointer;
	}
```

d)

```
void Remove_book(string& ISBN, map<string, Book*>& c)
{
	int number_of_books_removed = c.erase(ISBN);
	cerr<<"Book with ISBN "<< ISBN << " was removed " <<number_of_books_removed <<
endl;
}
```

Funktionen kommer alltid att skriva ut antingen 0 eller 1 eftersom returvärdet från `erase` är antalet element som har tagits bort. Se i övrigt kommentaren för uppgift c) ovan.

e) Om man kan acceptera att en del variabelnamn i Book från a)-delen inte passar direkt på variabelnamn för CD-skivor så kan man låta Book bli basklass till CD. Det blir dock lite konstigt om man tänker sig att CD är en sorts bok. En annan möjlighet är att byta namn på Book till Article och låta Article bli en basklass för alla sorters föremål som biblioteket lånar ut. All gemensam information sparas i Article och det som är unikt för mediet sparas i olika subklasser. Detta gäller t.ex. information om låtarna för CD.

Båda alternativen kräver dock att varje CD får ett unikt ID-nummer motsvarande bokens ISBN-nummer.

Alternativet med Article nedan, med get/set-funktioner et.c. utelämnade och endast konstruktorer utan argument visade.

```
class Article
{
public:
	Article();
	virtual ~Article() {};
protected:
	string m_creator;
	string m_title;
	string m_publisher;
	string m_id;
	int m_publishing_year;
};

class Book: public Article
{
	Book();
	virtual ~Book();
protected:
	int m_number_of_pages;
};

class CD: public Article
{
	struct Song_Information
	{
		string m_name;
		int m_minutes;
		int m_seconds;
	};
public:
	CD();
	virtual ~CD();

protected:
	list<Song_Information*> songs;
};
```

Det är inte viktigt att man använder en lista för att spara information om låtarna, men det är viktigt att informationen finns med på något sätt.