Tentamen i TDP004
Objektorienterad Programmering
Lösningsförslag

Datum:	2009-04-15
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: 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 teoretisk uppgift på ett separat blad.
Uppgifterna är inte ordnade efter svårighetsgrad.

Lycka till!

TDP004 Objektorienterad Programmering
Teoretisk del

1. **Du har fått följande kod:**

```
class A
{
public:
        A();
        ~A();
        virtual int f() = 0;
        virtual int g();
protected:
private:
};

class B: public A
{
public:
        B();
        ~B();
        int g();
protected:
private:
        int f();
};

------------------
I klassen C:
A* a = new B();
B* b = new B();

int result1 = a->f();
int result2 = a->g();
int result3 = b->f();
int result4 = b->g();
```

Användandet av klasserna ovan kompilerar inte. Hitta felaktigheten och förklara varför det är en felaktighet! (4p)

Koden kompilerar inte p.g.a. raden ”int result3 = b->f();”. Klassen C kan inte komma åt funktionen f, eftersom den är privat i klassen B, men public i klassen A (klassen C är inte friend till någon av klasserna A eller B).

2. **Det vanligaste sättet när man använder objektorientering är att använda `class`. Vilken är skillnaden mellan att använda `struct` och `class` och vad måste du som programmerare tänka på om du använder `struct`? (2p)**

Skillnaden är att åtkomsten är public som default i struct, medan den är private som default i class. Detta gör att andra klasser kan komma åt variabler och funktioner på ett annat sätt än i class.

3. **Beskriv skillnader mellan `std::vector` och `std::list`. (4p)**

Båda kan hantera olika sorters element eftersom de instantieras mha template, tex
```
std::vector<double> doubleVector;
std::vector<ExampleClass*> classVector;
```
och motsvarande för `std::list`.
Båda hör till standardbiblioteket Standard Template Library (STL).

List: elementen utspridda i minnet. Snabbt att lägga till element först och sist. Långsammare iteration än vector. Minnesutrymme allokeras allt eftersom element läggs till. Egen implementation av en del algoritmer eftersom det inte finns random access-iteratorer till list.

Vector: Elementen sekventiellt efter varandra i minnnet. Snabbt att lägga till längst bak, långsamt på alla andra platser. Snabb iteration. Minnesutrymme allokeras för ett visst antal element. När detta tar slut allokeras minne för dubbla antalet element.

4. **a) Vilka tre olika sorters minneshantering finns det i C++? (3p)**
   b) Vilka variabler hamnar var? (3p)
   c) Vilka av dessa variabler måste programmeraren själv vara särskilt uppmärksam på och varför? (2p)

a) Statiskt minne, stackminne samt heap.
b) `static` finns i statiskt minne. Lokala variabler och funktionsargument finns på stacken/automatiskt minne, i s.k. stack frames. Dynamiskt allokerat minne finns på heapen (även kallad free store).
c) De variabler som allokeras med new eftersom de hamnar på heapen, eftersom programmeraren själv måste ta bort dessa.

5. **Återanvändning är ett av motiven bakom objektorientering. Ge exempel på fördelar och nackdelar med återanvändning. Skriv max 5 skilda saker. (5p)**

Fördelar: Antagligen mindre buggar pga att koden är väl testad. Man kan använda algoritmer som man inte skulle kunna skriva själv. Billigare och snabbare än att skriva koden från början (detta kan dock ifrågasättas i praktiken).
Nackdelar: Det tar längre tid att skriva kod som ska återanvändas eftersom man måste göra den mera generell för att den överhuvudtaget ska gå att återanvända.

TDP004 Objektorienterad Programmering
Praktisk del

Den här delen av tentamen bygger på uppgift 1 och 2, vilka bör lösas i ordning. Uppgift 3 och 4 kan göras i valfri ordning. Om du inte klarar av uppgift 2 innan du gör uppgift 3 eller 4, skapa själv lämplig indata att testa din implementation med.

1. **I mappen given_files finns filen medaljer.txt. På varje rad i filen finns en landsförkortning på tre bokstäver samt tre stycken tal. Det första talet är antalet guldmedaljer, det andra talet är antalet silvermedaljer och det tredje är antalet bronsmedaljer i ett stort mästerskap. Uppgiften går ut på att välja ett lämpligt sätt att modellera informationen. Ditt svar ska innehålla koden samt en kort motivering/diskussion i filen (max ca en sida) som behandlar t.ex. följande:**
   - **Hur ska nya instanser skapas?**
   - **Hur ska medlemsvariabler modifieras?**
   - **Hur hanteras åtkomst av medlemsvariabler? Finns det felkontroll av medlemsdatan?**
   - **Är ditt val förberett för att återanvändas genom arv?**

   Poäng ges för vettiga val och motiveringar till dessa. (10p)

```
/* Suggested design for problem 1. Motivations appearent in
code. */
class CountryInfo
{
public:
     CountryInfo(std::string a_Name,
                    int nr_gold,
                    int nr_silver,
                    int nr_bronze)
     {
          m_name = a_Name;
          SetNoGold(nr_gold);
          SetNoSilver(nr_silver);
          SetNoBronze(nr_bronze);
     }
     virtual ~CountryInfo() {};

     const std::string GetName() const { return m_name; }
     const int GetNoGold() const { return m_gold; }
     const int GetNoSilver() const { return m_silver; }
     const int GetNoBronze() const { return m_bronze; }

     void SetNoGold(int no_gold)
     {
          if (no_gold >= 0)
```

```
            {
                m_gold = no_gold;
            }
            else
            {
                std::cout << "Error!
CountryInfo::SetNoGold(). Can not set number of gold medals
to value < 0." << std::endl;
                m_gold = 0;
            }
      }
      void SetNoSilver(int no_silver)
      {
            if (no_silver >= 0)
            {
                m_silver = no_silver;
            }
            else
            {
                std::cout << "Error! CountryInfo::
SetNoSilver (). Can not set number of silver medals to
value < 0." << std::endl;
                no_silver = 0;
            }
      }

      void SetNoBronze(int no_bronze)
      {
            if (no_bronze >= 0)
            {
                m_bronze = no_bronze;
            }
            else
            {
                std::cout << "Error! CountryInfo::
SetNoBronze (). Can not set number of bronze medals to
value < 0." << std::endl;
                m_bronze = 0;
            }
      }

protected:

      std::string m_name;
      int m_gold;
      int m_silver;
      int m_bronze;
```

```
};
```

2. **Läs in information från filen medaljer.txt och lagra informationen i instanser av din datastruktur från uppgift 1. Om öppningen av filen misslyckas ska det meddelas till användaren, annars antas det att all information i filen är korrekt inläst. Lagra instanserna på lämpligt sätt. (5p)**

```
/* The information is stored as std::vector<CountryInfo*>
m_info for problems 2-4. */
bool Problem2(const std::string & file_name)
{
	std::ifstream infile( file_name.c_str() );

	if( !infile)
	{
		// Failed to open file.
		std::cout << "Failed to open file "
			<< file_name << std::endl;
		return false;
	}

	CountryInfo* info;
	std::string country_name;
	int gold;
	int silver;
	int bronze;

	//Read all data in file.
	while( infile >> country_name >> gold >> silver >>
bronze)
	{
		info = new CountryInfo(country_name, gold,
	silver, bronze);
		m_info.push_back(info);
	}

	//Close file after finished reading.
	infile.close();

	return true;
}
```

3. **Gör en funktion find_countries som tar inparametrar av lämpliga typer och returnerar en STL container (av valfri och passande typ) innehållande de länder som uppfyller kraven att antalet medaljer av olika valörer har de värdena som inparametrarna anger. Exempel: Om funktionen får inparametrarna 1,2,3 så ska returvärdet innehålla de instanser som**

representerar de länder som har tagit 1 guldmedalj, 2 silvermedaljer och 3 bronsmedaljer. Om det inte finns något land som uppfyller kraven ska en tom container returneras. Välj lämpliga inparametrar och returtyp till funktionen samt implementera funktionen. Motivera dina val kortfattat i filen. (3p)

```
	/* std::list or std::vector are the most suitable
	containers in this case. */
std::vector<CountryInfo*> find_countries(const int no_gold,
const int no_silver, const int no_bronze)
{
	std::vector<CountryInfo*> returnValue;
	std::vector<CountryInfo*>::const_iterator i;
	for(i = m_info.begin(); i != m_info.end(); i++)
	{
		if( ((*i)->GetNoGold() == no_gold) &&
			((*i)->GetNoSilver() == no_silver) &&
			((*i)->GetNoBronze() == no_bronze))
		{
			returnValue.push_back( (*i));
		}
	}
	return returnValue;
}
```

4. **Ta informationen från uppgift 2, sortera den så att ett land med fler medaljer av en viss typ rankas än ett land med färre medaljer. Om antalet medaljer av en viss typ är lika mellan två länder, använd medaljtypen närmast under för att avgöra ordning. Guld rankas högst, silver näst högst och brons lägst. Någon av de inbyggda sort-funktionerna i STL ska användas, d.v.s. du ska *inte* implementera någon egen sorteringsalgoritm. Efter att informationen är sorterad, skriv ut den på skärmen med först landsförkortningen, sedan antalet guldmedaljer, antalet silvermedaljer och sist på raden antalet bronsmedaljer. Det land med högst ranking ska stå först och ha siffran 1 framför sig, näst högst ranking skrivs ut därefter med siffran 2 före sig och så vidare. Se exempel nedan. (5p)**

   Exempel på indata:
   USA 0 1 0
   Nor 0 1 1
   Swe 2 1 0

   Utskrift efter sortering:
   1 Swe 2 1 0
   2 Nor 0 1 1
   3 USA 0 1 0

```
void Problem4()
{
	//Sort m_info using the STL sort function.
	sort(m_info.begin(), m_info.end(), SortPredicate);

	//Print to screen.
	for(unsigned int i = 0; i != m_info.size(); ++i)
	{
		std::cout << i+1 <<" " <<m_info[i]->GetName()
				<<" "<<m_info[i]->GetNoGold()
				<<" "<<m_info[i]->GetNoSilver()
				<<" "<<m_info[i]->GetNoBronze()
				<< std::endl;
	}
}

//Used as sorting predicate. If put in a class it must be
public because it is called by STL function sort.
bool Uppg1::SortPredicate(const CountryInfo* a,
				const CountryInfo* b)
{
	if(a->GetNoGold() > b->GetNoGold() )
	{
		return true;
	}

	if(a->GetNoGold() < b->GetNoGold() )
	{
		return false;
	}

	if(a->GetNoSilver() > b->GetNoSilver() )
	{
		return true;
	}

	if(a->GetNoSilver() < b->GetNoSilver() )
	{
		return false;
	}

	if(a->GetNoBronze() > b->GetNoBronze() )
	{
		return true;
	}

	return false; }
```