Tentamen i TDP004
Objektorienterad Programmering
Lösningsförlag

Datum:	2010-08-27
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: 44 med 22 poäng vardera på teori och praktikdel.

Poäng	**Betyg**
38-44	**5**
31-37	**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. Vad medför nyckelordet `friend` och vad har detta med inkapsling att göra? (3p)

2. Ge exempel på olika tillfällen då det är lämpligast att använda `std::list`, `std::vector` respektive `std::map`. (6p)

3. Vad är skillnaden på dynamisk och statisk binding? Ge exempel på när de olika bindningarna inträffar. (4p)

4. a) Vad behöver du ändra i nedanstående kod om du vill byta container från `std::vector` till `std::list`? (6p)

```
double find_least_distance (const std::vector<Unit*>& units,
                            Point* point) const
{
      double closest = 10000000.0;
      double distance;
      unsigned int i = 0;
      bool firstTime = true;
      for(; i < units.size(); i++)
      {
             distance = units[i]->GetDistance(point);
             if ( (true == firstTime) || (distance < closest) )
             {
                  firstTime = false;
                  closest = distance;
             }
      }
      return closest;
}
```

   b) Titta på den första inparametern. Vad är fördelarna med att deklarera inparametern på det sättet? (2p)

Praktisk del

1. Ett lösenord i ett visst system måste uppfylla samtliga följande krav:
   - Det måste vara minst 5 och max 15 tecken långt.
   - Det måste innehålla både siffror och bokstäver.

   Gör en funktion med nedanstående signatur
   `bool check_password(const std::string password)`
   som testar huruvida lösenordet uppfyller kraven. (5p)

2. a) Den seriöse programmeraren har naturligtvis med sig en dator var hon/han än befinner sig, även så här i slutet av grillsäsongen. En användning för datorer i det här fallet är en beräkning av hur lång tid en viss mängd mat tar att tillaga. Olika sorters mat att grilla tar olika lång tid beroende på mängden, och i det här fallet finns tre olika sorters mat: korv, fisk och det vegetariska alternativet quorn. Grilltiden beräknas enligt nedan. De olika maträtterna ska även separat kunna svara på frågan om lämplig tillagningstemperatur. På grund av att grillen har den svårt att hålla en exakt temperatur, används benämningarna låg, mellan och hög temperatur enligt nedan.

   Korv: 5 min per 0,2 kg, på mellan-temperatur.

Fisk: 10 min per kg fisk, på låg temperatur.
Quorn: 2 min per 0,1 kg, dock max 5 min, på hög temperatur.

Implementera en arvshierarki för olika sorters mat där de olika sorternas mat kan beräkna grilltiden enligt ovan. (10p)

b) Kontrollera att din implementation från uppgift 2 a) fungerar som avsett genom att skriva ett program som efterfrågar tillagningstemperatur och tid för de olika sorternas mat. Din kontroll ska använda polymorfism. (6p)

Lösningsförslag teoretisk del

1. Det medför att en klass som har deklarerar en annan klass eller funktion som `friend` låter klassen/funktionen få tillgång även till klassens icke-publika data. `Friend` kan inte sägas bryta mot inkapsling eftersom det är den tillåtande klassen som deklarerar att den ger någon annan tillgång.

2. `std::list` är lämpligt att använda då man vill kunna lägga till och ta bort element snabbt varsomhelst i datastrukturen. Den är mindre lämplig att använda då man vill sortera elementen ofta eftersom detta är långsammare än för t ex `vector`. Det går också långsamt att iterera igenom elementen, igen jämfört med `vector`.

   `std::vector` är lämpligt att använda då man gör många iterationer igenom elementen. Det går snabbt att sortera elementen och det finns en random access-operator[], men det går inte att lägga till element längst fram m.h.a. standardinterfacet, utan man måste lägga till element längst bak.

   `Std::map` är lämpligt att använda då man vill knyta ihop ett element och en nyckel, eftersom `map` lagrar elementen i `std::pair`. Internt används en trädstruktur, vilket gör sortering och access av elementen snabbt.

3. Statisk (tidig) binding är för icke-virtuella funktioner, där bindningen sker vid kompileringstillfället. Dynamisk (sen) bindning används för virtuella funktioner, då det avgörs vid exekveringstillfället vilken kod som körs.

4. a) Förändringar i fet stil:

```
double find_least_distance (const std::list<Unit*>& units,
                            Point* point) const
{
      double closest = 10000000.0;
      double distance;
      std::list<Unit*>::const_iterator i;
      bool firstTime = true;
      for(; i != units.end(); i++)
      {
            distance = (*i)->GetDistance(point);
            if ( (true == firstTime) || (distance < closest) )
            {
                 firstTime = false;
                 closest = distance;
            }
      }
      return closest;
}
```

b) Poängen med att använda en referens `&` är att man genom att använda en referens snabbar vi upp exekveringen eftersom vi skickar en referens till objektet som inparameter istället för att skicka hela objektet. `Const` gör att funktionen inte kan ändra på inparametern, vilket ger säkrare kod.

Lösningsförslag praktisk del

1.

```
#include <string>
#include <cctype>

bool test_password(const std::string pass)
{
	if(pass.length() < 5)
	{
		return false;
	}

	if(pass.length() > 15)
	{
		return false;
	}

	bool letter_found = false;
	bool digit_found = false;

	for(unsigned int i = 0; i < pass.length(); i++)
	{
		/*	Obs. C++ primer är otydlig angående typen på
			funktionernas returvärde.
			Kompilatorn säger att de returnerar en int samt
			att 0 är false. */

		if (0 != isalpha( pass[i] ))
		{
			letter_found = true;
		}

		if (0 != isdigit( pass[i] ))
		{
			digit_found = true;
		}

		if ((true == letter_found) && (true == digit_found))
		{
			return (letter_found && digit_found);
		}
	}
	return false;
}
```

2 a) I filen Food.h

```
//Obs enum behöver inte användas, men är lämpligt.
enum temp
{
	Low,
```

```
	Medium,
	High
};

class Food
{
public:
	Food(void);
	virtual ~Food(void);
	void set_temp(temp value);
	virtual temp get_temp() {return m_temp;};

	virtual float get_cooking_time(float weight_in_kilo) = 0;

protected:
	temp m_temp;
};

// I filen Fish.h
#include "Food.h"

class Fish : public Food
{
public:
	Fish(void);
	~Fish(void);

	float get_cooking_time(float weight_in_kilo)
	{
		float min_per_kilo = 10;
		return (weight_in_kilo * min_per_kilo);
	}
};
//I filen HotDog.h
#include "Food.h"

class HotDog : public Food
{
public:
	HotDog(void);
	~HotDog(void);

	float get_cooking_time(float weight_in_kilo)
	{
		float min_per_kilo = 25; //5 min per 0.2 kg
		return (weight_in_kilo * min_per_kilo);
	}
};

//I gilen Quorn.h

#include "Food.h"

class Quorn : public Food
{
public:
	Quorn(void);
```

```
        ~Quorn(void);

        float get_cooking_time(float weight_in_kilo)
        {
                float min_per_kilo = 20;
                float max_cooking_time = 5;
                float cooking_time = min_per_kilo * weight_in_kilo;

                if(cooking_time > max_cooking_time)
                {
                        cooking_time = max_cooking_time;
                }

                return cooking_time;
        }
};
```

2 b) Polymorfism kan t.ex. användas för att testa a-delen enligt nedan.

```
Food* q = new Quorn();
Food* f = new Fish();
Food* h = new HotDog();
std::cout << "Cooking time for 2 kg. "<< std::endl;
std::cout << "Fish: " << f->get_cooking_time(2) << std::endl;
std::cout << "Hot dog: " << h->get_cooking_time(2) << std::endl;
std::cout << "Quorn: " << q->get_cooking_time(2) << std::endl;
```