Eksamensopgave i RD2-SWC

15. august 2022

Alle offline hjælpemidler er tilladt.

Aflevering

Der skal afleveres én samlet PDF med opgavebesvarelsen af både C++-delen og softwareudviklingsdelen, som alene er det dokument, der bliver bedømt. I kan kun aflevere et dokument som altså både skal indeholde C++ afleveringen med kildekode (formateret som farvet tekst) og test samt softwareudviklingsdelen med SQL-kode (formateret som tekst).

Til hver C++ opgave er der en ***test-kode***, som skal ***eksekveres*** og et ***output/screendump*** skal indsættes i afleveringen. Hvis programmet ikke kan compile, så indsættes i stedet ***fejlmeddelelsen*** fra compileren. Generelt vurderes opgaverne ud fra en korrekt anvendelse af de tilegnede C++ færdigheder, som er opnået gennem gennemførelse af faget. Det forventes, at de tillærte færdigheder anvendes når det kan, herunder anvendelsen af referencer ved ikke simple datatyper samt brug af const, hvor det er muligt.

Enhver form for kopier/indsæt (copy/paste) fra tidligere opgaver eller andre kilder anses som eksamenssnyd. Kode fra denne eksamensopgave må dog gerne kopieres ind.

Opgavesættet består af 7 sider, 1 forside, 4 sider med C++ opgaver og 2 sider med SQL og softwareudviklingsopgaver.

Opgave 1 (1 time, 25 point)

I denne opgave skal vi kigge på en klasse Expense, som markerer en udgift for et produkt.

```
class Expense
{
public:
    Expense();

private:
    std::string mName;
    double mCost;
};
```

Includes: string

a) Implementer klassen Expense

I første del af opgaven skal klassen Expense implementeres færdig. Alle funktioner SKAL implementeres i en cpp-fil og opfylde følgende krav:

- Standard constructor implementeres
- Constructor med parametrene name (*mName*) og cost (*mCost*).
- Get/setter for name
- Get/setter for cost

b) Implementer klassen ProductCost

I anden del af opgaven skal vi kigge på klassen ProductCost:

```
class ProductCost
{
public:
    ProductCost() {}

    void addExpense(double amount, const Expense& expense);

    double totalCost() const;

private:
    std::string mName;
    std::vector<std::pair<double, Expense> > mExpenses;
};
```

Includes: vector, ostream

Her skal der implementeres følgende (alle implementationer laves i CPP-filen)

- Get/set for product name (dvs. *getProductName*, *setProductName* på member *mName*)
- Funktionen *addExpense* som angivet i headeren, hvor der skal tilføjes et *pair* til vectoren *mExpenses* hvor *amount* og *expense* begge indgår.
- Funktionen *totalCost*, som udregner den totale omkostning som det flydende kommatal multipliceret med prisen (*cost*) i Expense objektet. Resultet er summen af alle beregninger for hvert element i vectoren *mExpenses*.
- Insertion operatoren (<<) implementeres som friend på klassen

Se testkode på næste side

Test-kode (HUSK Screendump af test)

```
    // Opgave 1a test
    std::cout << "---------------" << std::endl;
    std::cout << "-- Opgave 1a --" << std::endl;
    std::cout << "---------------" << std::endl;
    Expense transport("Transport (km)", 2.17);
    Expense packaging("Packaging", 1.01);
    Expense leather("Leather", 1130.05);
    Expense buckle("Buckle", 5.30);
    Expense workhours("Workhours", 180);

    std::cout << std::setw(20) << std::left << transport.getName()
              << transport.getCost() << std::endl;
    std::cout << std::setw(20) << std::left << packaging.getName()
              << packaging.getCost() << std::endl;


    // Opgave 1b test
    std::cout << std::endl;
    std::cout << "---------------" << std::endl;
    std::cout << "-- Opgave 1b --" << std::endl;
    std::cout << "---------------" << std::endl;
    ProductCost p1;
    p1.addExpense(1200, transport);
    p1.addExpense(1, packaging);
    p1.addExpense(0.2, leather);
    p1.addExpense(2, buckle);
    p1.addExpense(1, workhours);

    std::cout << p1;
    std::cout << "Total cost for product " << p1.getProductName() << ": "
              << p1.totalCost() << std::endl << std::endl;
```

Includes: iomanip, iostream, vector

Opgave 2 (1 time, 25 point)

I denne opgave skal vi kigge på klassen Product:

```
class Product
{
public:
    Product();

    Product(const std::string& name, double price);

    // Getters
    const char* getName() const { return mName; }
    int getLength() const { return mLengthOfName; }
    double getPrice() const { return mPrice; }

    // Setter
    void setName(const std::string& name);


private:
    char* mName;
    int mLengthOfName;
    double mPrice;
};
```

Includes: string

Følgende skal implementeres på/til klassen Product (implementationerne SKAL være i en separat cpp-fil)

- Default constructor som på fornuftig vis initialiserer default variable.
- Constructor med navn (*name*) og pris (*price*), der gemmer strengen *name* som et char array, der peges på af pointeren *mName*, længden af strengen gemmes i *mLengthOfName* og *price* gemmes i *mPrice*.
- Setteren *setName*, som gemmer name i *mName* og længden af strengen i *mLengthOfName* på samme måde som beskrevet for constructoren.
- Copy constructor
- Move constructor
- Copy assignment operator
- Move assignment operator

Test-koden findes på næste side.

Test-kode (HUSK Screendump af test)

```
// Opgave 2
std::cout << "---------------" << std::endl;
std::cout << "-- Opgave 2  --" << std::endl;
std::cout << "---------------" << std::endl;
std::vector<Product> products;
products.emplace_back("Leather bag", 1500);
products.emplace_back("Lamp", 1250);
products.emplace_back(products[0]);  // Copy constructor
products.emplace_back(std::move(products[0]));  // Move constructor
products.emplace_back("Table", 1200);
products.emplace_back("Desk", 2000);
products[4] = products[1];
products[5] = std::move(products[1]);

for (Product p : products) {
    if (p.getLength() > 0)
        std::cout << std::left << std::setw(20) << p.getName()
                  << p.getPrice() << std::endl;
    else
        std::cout << "No description available" << std::endl;
}

std::cout << std::endl;
products[2].setName("LALA");
products[4].setName("Old Table");
for (const Product &p : products) {
    if (p.getLength() > 0)
        std::cout << std::left << std::setw(20) << p.getName()
                  << p.getPrice() << std::endl;
    else
        std::cout << "No description available" << std::endl;
}

std::cout << std::endl;
products[3].setName("LALA2");
products[5].setName("Old Desk");
for (const Product &p : products) {
    if (p.getLength() > 0)
        std::cout << std::left << std::setw(20) << p.getName()
                  << p.getPrice() << std::endl;
    else
        std::cout << "No description available" << std::endl;
}
```

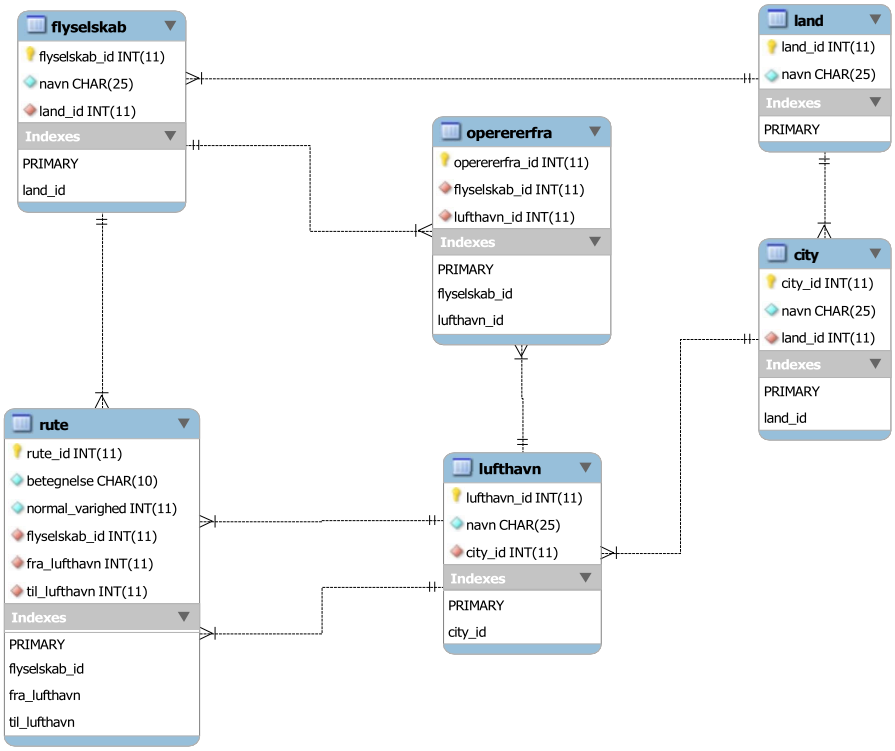
Includes: iomanip, iostream, vector

Opgave 3 (SQL, 1 time, 25 point)

Denne opgave drejer sig om en database til håndtering af flyselskabers data.

ERD-diagrammet er vist nedenfor. De anvendte navne er på dansk, bortset fra *city*. Ordet *by* er et keyword i SQL og kan derfor ikke anvendes som navnet på en tabel.

Ideen er, at vi har et antal lande, som har en eller flere byer. I byerne er der en eller flere lufthavne. Hvert land har et eller flere flyselskaber, og disse opererer ud fra en eller flere lufthavne. Endelig har vi ruter, hvortil der bl.a. er knyttet et flyselskab og en til- og fra-lufthavn. Alle ruter er tur/retur, dvs. hvis der er en rute fra A til B, er der også en rute fra B til A.



Databasen kan etableres i MySQL ved hjælp af tekstfilen ***FlyselskabDDL***.

I opgavebesvarelserne er det tilladt at spørge på indholdet af datafelter, men ikke på værdien af id'er.

FLYSELSKAB.NAVN = 'SAS' er således tilladt; men det er **FLYSELSKAB.FLYSELSKAB_ID = 1** *ikke*.

<u>Opgave 1 (5 %)</u>
Lav en liste over flyselskabernes navne samt det land, de er hjemmehørende i.

<u>Opgave 2 (5 %)</u>
Lav en liste over navnene på de flyselskaber, som opererer i lufthavne i USA. Hvert navn må kun forekomme én gang. Du kan se bort fra rute-tabellen i denne opgave.

<u>Opgave 3 (5 %)</u>
Lav en liste over navnene på de byer, som har mere end én lufthavn.

<u>Opgave 4 (5 %)</u>
Skriv en forespørgsel som kan lave en liste over samtlige ruter med angivelse af navnet på flyselskabet, rutebetegnelsen, normal varighed (timer) og navnene på fra-lufthavn og til-lufthavn.

<u>Opgave 5 (5 %)</u>
Skriv en forespørgsel som viser navnene på de lufthavne, som ikke indgår i en rute.

Opgave 4 (Domænemodellering, 1 time, 25 point)

Denne opgave går ud på at lave det indledende analyse- og designarbejde til en 1. iteration af et it-system til et museum, som alene vedrører én kendt person. Det kunne fx være H.C. Andersen, Storm P. eller J.F. Willumsen.

Denne opgavetekst er inspireret af Storm P. Museet på Frederiksberg i København. Robert Storm Petersen (1882-1949) var blandt meget andet tegner, maler, humorist, forfatter, skuespiller og samfundskritiker.

Systemet har blandt andet til formål at kunne holde styr hvilke genstande museet ejer, hvor de er fysisk placeret i bygningen og diverse forhold omkring museets gæster.

Museet kan beskrives på følgende måde:

- Der udstilles malerier, tegninger, bøger, plakater, informationstavler og personlige effekter (fx skrivebord, seng, piber, lamper).
- Museet har seks udstillingslokaler, og alle lokaler kan indeholde alle typer af genstande.
- Der er ansat tre medarbejdere, hvoraf mindst to er til stede i åbningstiden.
- Gæsterne betaler entré ved indgangen, men får *ikke* udleveret en billet.
- Der udstedes årskort, som blot skal forevises ved indgangen. Indehaveren af årskortet, som er personligt, kan medbringe én gæst.
- Personer under 18 år har gratis adgang.

Den første iteration af systemet skal som minimum kunne klare følgende funktionalitet:

- Holde styr på samtlige genstande, deres placering, betegnelse, kort beskrivelse og for værkernes vedkommende fremstillingsår.
- Kunne registrere museets åbningstider og medarbejdernes arbejdstid.
- Registrere antal solgte billetter og de derved skabte indtægter.
- Registrere antallet af samtlige gæster, dvs. udover solgte billetter, årskort med ledsager og børn.

Det centrale artefakt, som du skal udarbejde, er:

- Domænemodel

Og du bestemmer selv, hvilke 'hjælpeartefakter' (fx brugsmønstre, kontrakter) du vil lave. Husk at et artefakt skal kun udarbejdes, hvis den skaber værdi for systemet.

Det er imidlertid vigtigt, at du argumenterer for de trufne valg. Den gode besvarelse indeholder relevante kommentarer til de indgående elementer (klasser, attributter, relationer mellem klasser og metoder). Nogle af disse er selvforklarende, andre er ikke. Formålet med kommentarerne er især at bidrage til din og andres forståelse af domænet.