Eksamensopgave i RD2-SWC

4. juni 2019

Alle hjælpemidler, som ikke kræver brug af internet, er tilladt.

Aflevering

Der skal afleveres én PDF-fil med opgavebesvarelsen i C++-delen og én PDF-fil med opgavebesvarelsen i Softwareudviklingsdelen. Desuden skal al kildekode afleveres i en zip-fil. En komplet a*flevering indeholder altså **3 filer***.

Til hver C++ opgave er der en ***test-kode***, som skal ***eksekveres*** og et ***screendump*** skal indsættes i afleveringen. Hvis programmet ikke kan compile, så indsættes i stedet ***fejlmeddelelsen*** fra compileren.

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

Opgavesættet består af 6 sider, 1 forside, 4 sider med C++ opgaver og 1 side med software-udviklingsopgaven.

Opgave 1 (1-1½ time, 35 point)

I denne opgave skal klassen Product implementeres. Klassen Product må ikke benytte klassen std::string og skal i stedet benytte et char-array.

```
class Product {
public:
    Product();
    Product(const char* name, double price);

    Product(const Product& p);
    Product(Product&& p);

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

Constructoren tager som input et char-array, der indeholder en streng med navnet på produktet. I denne opgave skal selve strengen gemmes som member variabel i et char-array (ikke som std::string).

HINT: Husk at C-style strings er termineret med et null-tegn.

a) Implementer constructor, copy og move constructor

I første del af opgaven skal constructor, copy og move constructor implementeres.

Implementer desuden get/set funktionerne til member-variablene mName og mPrice (getName, setName, getPrice og setPrice).

b) Tilføj operatorer

Tilføj følgende operatorer til klassen

- Copy assignment
- Move assignment
- Comparison equal (= =)
- Insertion operator (<<)

Se test-kode på næste side (HUSK screendump):

Test-kode

a)

```
std::cout << "--------------------" << std::endl;
std::cout << "-- Test Opgave 1a --" << std::endl;
std::cout << "--------------------" << std::endl;
char productname[] = "Product 1";
Product p1(productname, 15.1);
Product p2;
productname[8] = '2';
p2.setName(productname);
p2.setPrice(30.2);
Product p3;
p3.setName("Product 3");
p3.setPrice(10.3);
Product p4("Product 4", 14.4);
Product p5(p4);
Product p6(std::move(p3));

std::cout << "Product name   | Price" << std::endl;
std::cout << p1.getName() << "         " << p1.getPrice() << std::endl;
std::cout << p2.getName() << "         " << p2.getPrice() << std::endl;
std::cout << p3.getName() << "         " << p3.getPrice() << std::endl;
std::cout << p4.getName() << "         " << p4.getPrice() << std::endl;
std::cout << p5.getName() << "         " << p5.getPrice() << std::endl;
std::cout << p6.getName() << "         " << p6.getPrice() << std::endl;
std::cout << std::endl;
```

b)

```
std::cout << "--------------------" << std::endl;
std::cout << "-- Test Opgave 1b --" << std::endl;
std::cout << "--------------------" << std::endl;
Product p7("Product 7", 14.7);
Product p8;
p8 = p7;
p5 = std::move(p4);
bool p7ep8 = (p7==p8);
bool p6ep2 = (p6==p2);

std::cout << "Product name   | Price" << std::endl;
std::cout << p2.getName() << "         " << p2.getPrice() << std::endl;
std::cout << p5.getName() << "         " << p5.getPrice() << std::endl;
std::cout << p7.getName() << "         " << p7.getPrice() << std::endl;
std::cout << p8.getName() << "         " << p8.getPrice() << std::endl;
std::cout << std::boolalpha << p7ep8 << std::endl;
std::cout << std::boolalpha << p6ep2 << std::endl;
std::cout << p8 << std::endl;
std::cout << std::endl;
```

Opgave 2 (½-1 time, 15 point)

I denne opgave skal vi implementere en klasse med to funktioner, der kalder hinanden. Klassens struktur ses herunder:

```
class A {
public:
    A();
    void f(int a, int b);

private:
    void g(int a, int b);
};
```

Når funktionen *f* kaldes på et objekt af typen A, så skal *f* kalde funktionen *g*.

Krav til funktionen *g*

Funktionen *g* skal observere, hvis parameteren *a* eller *b* er fejlfyldt:

- Hvis parameteren *a* er negativ, skal funktionen ikke returnere men benytte en exception af typen *invalid_argument_negative*.
- Hvis parameteren *b* er nul eller positiv, skal funktionen heller ikke returnere, men benytte en exception af typen *invalid_argument_positive*.

Der skal ikke udskrives nogen fejlbesked i funktionen *g*.

Krav til exceptions

De to exceptions eksisterer ikke i standardbiblioteket. Disse skal derfor implementeres med relevante fejlbeskeder. Begge exceptions skal nedarve fra *invalid_argument* fra standardbiblioteket.

Krav til funktionen *f*

Funktionen *f* skal blot kalde funktionen *g*. Dog skal funktionen *f* fange de to forskellige exceptions og udskrive nedenstående fejl afhængig af den exception, som fanges:

- *invalid_argument_negative:* Error: parameter *a* is negative
- *invalid_argument_positive:* Error: parameter *b* is zero or positive

BEMÆRK: Efter at have udskrevet fejlen, skal den pågældende (altså præcis den samme) exception sendes videre til den kode, som kalder *f* (rethrow).

Test

Se test-kode på næste side (HUSK screendump):

```
A a;
try {
    std::cout << std::endl << "1: Calling a.f(-1,-1)" << std::endl;
    a.f(-1,-1);
} catch (std::exception& e) {
    std::cerr << "1: " << e.what() << std::endl;
}

try {
    std::cout << std::endl << "2: Calling a.f(1,1)" << std::endl;
    a.f(1,1);
} catch (std::exception& e) {
    std::cerr << "2: " << e.what() << std::endl;
}

try {
    std::cout << std::endl << "3: Calling a.f(1,0)" << std::endl;
    a.f(1,0);
} catch (std::exception& e) {
    std::cerr << "3: " << e.what() << std::endl;
}

try {
    std::cout << std::endl << "4: Calling a.f(0,-1)" << std::endl;
    a.f(0,-1);
} catch (std::exception& e) {
    std::cerr << "4: " << e.what() << std::endl;
}
```