Eksamensopgave i RD2-SWC

1. Juni 2018

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

Aflevering

Der skal afleveres én PDF-fil med opgavebesvarelsen i C++-delen og en (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.

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

Opgave 1 (½ time, 12,5 point)

I denne opgave skal vi arbejde videre på template klassen i nedenstående kode

```
template <class T>
class Box {
public:
    Box(int size = 10) {
        _l = 0; _b = 0; _h = 0; _size = size;
        if (size > 0)
            _array = new T[_size];
    }

    int size() { return _size; }

private:
    T* _array;
    int _size;
    int _l, _b, _h;
};
```

Member variablen _size angiver størrelsen på det array, som pointeren _array peger på. Bemærk, at _size kan også være nul.

Denne opgave går ud på at skrive en copy constructor, copy assignment operator samt en destructor til denne template klasse. Copy constructor samt copy assignment skal kopiere alle member variable samt oprette et nyt array og kopiere alle data. Husk at tage hensyn til antallet af elementerne i _array.

Koden skal testes med eksempelkoden nedenfor og der skal laves et ***screendump*** af output fra program, som medtages i eksamensbesvarelsen.

```
    {
        Box<double> b1(5);
        Box<double> b2(10);
        Box<double> b3 = b1;
        Box<double> b4(b1);
        b1 = b2;

        std::cout << "b1.size() = " << b1.size() << std::endl;
        std::cout << "b2.size() = " << b2.size() << std::endl;
        std::cout << "b3.size() = " << b3.size() << std::endl;
        std::cout << "b4.size() = " << b4.size() << std::endl;
    }
```

Opgave 2 (½ time, 12,5 point)

Polynomier er beskrevet vha. flere led, lad os i dette tilfælde kigge på eksemplet:

2x^5+3x^2+4x^0

Det sidste led er naturligvis en konstant, men for at simplificere opgaven, vil konstanter altid skrives som en koefficient gange x^0.

I dette eksempel er leddene af polynomiet 2x^5, 3x^2 og 4x^0. Tilsvarende er koefficienterne til hvert led 2, 3 samt 4. Potensen (power) i hvert led er ligeledes 5, 2 samt 0.

Denne opgave går ud på at indlæse et polynomie fra en tekst-streng, hvor der er præcis en linje og et polynomie skrevet på ovenstående form, f.eks.

22x^3+2x^2-1x^0

Det kan antages, at der altid står et tal foran hver variabel. Variablens navn er altid "x" og potensen er altid positiv. Koefficienten foran x kan være både positiv og negativ. Det kan antages, at der ingen mellemrum er i denne streng.

Skriv en funktion som tager en streng som input (der indeholder en streng med polynomiet) og returnerer en vector af koefficienter samt potenser.

Funktionerne defineres som

- `std::vector<std::pair<double, int> > readPolynomial(std::string& p)`

Funktionerne returnerer altså en std::vector med resultaterne. (Husk mellemrum mellem > og >).

HINT: Indlæs en tal-værdi, indlæs en karakter (x), indlæs en karakter (^), indlæs en tal-værdi (fortegnet +/- kan indgå som en del af tallet), fortsæt samme strategi ... Brug evt. en stringstream til at konvertere string til stream.

SIMPLIFICERING: ***ANTAG AT POLYNOMIET BESTÅR AF 3 LED***. Det er ikke nødvendigt at kontrollere om der er yderligere led, der er altid 3.

Includes: sstream, vector, utility

Følgende test-kode skal anvendes, og der skal laves et ***screendump*** af output fra dit program, som medtages i eksamensbesvarelsen:

```
std::string pstr = "22x^3+2x^2-1x^0";
std::vector<std::pair<double, int> > poly = readPolynomial(pstr);

for (unsigned int i = 0; i < poly.size(); ++i) {
    if (i>0) {
        if (poly[i].first > 0)
            std::cout << "+";
    }
    std::cout << poly[i].first << "x^" << poly[i].second;
}
std::cout << std::endl;
```

Opgave 3 (1 time, 25 point)

I denne opgave vil vi arbejde videre med polynomier. Opgaven er uafhængig af foregående opgave og kræver ikke, at denne er løst.

Skriv en klasse PolynomialTerm, som repræsenterer ét led af et polynomie. Følgende interface for klassen skal overholdes (se kommentarer for krav og forklaringer)

```
class PolynomialTerm {
public:
    // Default constructor skal initialize
    // member variable til default værdier
    PolynomialTerm();

    // Constructor til initialisering af member variable
    // Der skal anvendes initializer lists
    // (constructor initialization section)
    PolynomialTerm(double coefficient, int power);

    // Get-funktioner for member variable
    int getPower() const;
    double getCoefficient() const;

    // Set-funktioner for member variable
    void setPower(int power);
    void setCoefficient(double coeff);

    // Beregn ledets værdi når værdien for x kendes (parameteren)
    double evaluate(double x);

private:
    double _coeff;  // Member variabel for koefficienten
    int _power;     // Member variable for potensen (power)
};
```

Derudover skal der skrives en klasse Polynomial, som indeholder flg. constructors samt member funktioner:

- Default constructor
- void **addTerm**(const PolynomialTerm& term); // Tilføj led til polynomie (e.g. 2x^3)
- double **evaluate**(double x); // Beregn polynomiets værdi ved given x-værdi
- friend funktion for operatoren << (insertion operator)

HINT: Brug evt. en vector som member variabel til at gemme alle led i polynomiet.

Includes: cmath, vector, ostream

Koden skal testes med eksempelkoden på næste side og der skal laves et ***screendump*** af output fra program, som medtages i eksamensbesvarelsen.

```
PolynomialTerm t0(3.0, 0);
PolynomialTerm t1(1.0, 1);
PolynomialTerm t2(1.0, 2);
PolynomialTerm t3;
t3.setCoefficient(2.1);
t3.setPower(3);

Polynomial p1;
p1.addTerm(t1);
std::cout << "p1(0) = " << p1.evaluate(0) << std::endl;
std::cout << "p1(1) = " << p1.evaluate(1) << std::endl;
std::cout << "p1(2) = " << p1.evaluate(2) << std::endl;

Polynomial p2;
p2.addTerm(t0);
p2.addTerm(t2);
std::cout << "p2(0) = " << p2.evaluate(0) << std::endl;
std::cout << "p2(1) = " << p2.evaluate(1) << std::endl;
std::cout << "p2(2) = " << p2.evaluate(2) << std::endl;

Polynomial p3;
p3.addTerm(t0);
p3.addTerm(t1);
p3.addTerm(t2);
p3.addTerm(t3);
std::cout << "p3(0) = " << p3.evaluate(0) << std::endl;
std::cout << "p3(1) = " << p3.evaluate(1) << std::endl;
std::cout << "p3(2) = " << p3.evaluate(2) << std::endl;

std::cout << p1 << std::endl;
std::cout << p2 << std::endl;
std::cout << p3 << std::endl;
```

Opgave 4 (2 timer, 50 point)

Denne opgave går ud på at lave det indledende analyse- og designarbejde til en 1. iteration af et it-system til en enkeltmandsvirksomhed, der beskæftiger sig med handel med ejendomme (ejendomsmægler) på Fyn.

Systemet skal kunne håndtere følgende typer af ejendomme: parcelhuse, ejerlejligheder og sommerhuse, og kunderne (købere og sælgere) er privatpersoner.

Den første iteration af systemet skal kunne klare følgende funktionalitet:

- Oprettelse af ejendomme, som udbydes til salg med de vigtigste informationer (adresse, pris, opførelsesår, beboelsesareal). Liggetid – altså hvor længe ejendommen har været til salg er også en vigtig information.
- Oprettelse af kunder med de nødvendige identifikationsoplysninger. Både sælgere og potentielle købere regnes som kunder.
- Fremvisninger: en potentiel køber skal kunne bestille en fremvisning med mægleren til stede (husk, at der kun er én mægler).
- Tilbud: en potentiel køber skal have mulighed for at give et tilbud på en ejendom.
- Nedslag: systemet skal kunne vise, hvor meget nedslag der er blevet givet i prisen både før ejendommen er blevet solgt og i forbindelse med handelen.[1]
- Mægleren vil have behov for at kunne trække aggregerede data ud af systemet især baseret på ejendommenes beliggenhed (land, by, forstad, og geografiske områder, fx Nordfyn, Østfyn etc.). Et eksempel kunne være gennemsnitlig kvadratmeterpris på Vestfyn.

De to centrale artefakter, som du skal udarbejde, er:

- Domænemodel

- Designklassediagram (DKD)

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

Det er imidlertid meget vigtigt, at du argumenterer for de trufne valg. Den gode besvarelse indeholder således fyldige kommentarer til de indgående elementer (klasser, attributter, relationer mellem klasser og metoder). Omkring DKD bør der også være overvejelser vedrørende anvendelse af designmønstre, inklusive *GRASP*.

Ikke alle kommentarer er lige relevante. Relevante kommentarer skal, i lighed med 'hjælpeartefakterne', skabe værdi enten i form af bedre forståelse for rationalet bag de trufne valg eller ved at forøge kendskabet til domænet. F.eks. behøver tilvalget af attributten *emailAdresse* for en kunde næppe nogen uddybende forklaring.

I bedømmelsen vægtes domænemodellen med ca. 40 %, designklassediagrammet med ca. 30 % og kommentarerne med ca. 30 %.

[1] Det er ofte således, at prisen justeres nedad undervejs, hvis ejendommen viser sig vanskelig at sælge, og i forbindelse med selve handelen vil der ofte blive givet et yderligere nedslag efter forhandling mellem køber og sælger.