Eksamensopgave i RD2-SWC

3. juni 2020

Alle hjælpemidler er tilladt. Husk at de generelle regler om snyd stadig gælder.

Aflevering

Der skal afleveres én samlet PDF med opgavebesvarelsen af både C++-delen og softwareudviklingsdelen, som alene er det dokument, der bliver bedømt. Desuden skal der afleveres et bilag (zip-fil) med al jeres kildekode fra C++. I kan kun aflevere et dokument og en fil med bilag.

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 eller andre kilder anses som eksamenssnyd. Kode fra denne eksamensopgave må dog gerne kopieres ind.

Opgavesættet består af 12 sider, 1 forside, 9 sider med C++ opgaver og 2 sider med softwareudviklingsopgaven.

Opgave 1 (45 minutter, 20 point)

I denne opgave skal klassen Lot implementeres. Lot repræsenterer en byggegrund som sælges for en minimumspris (mMinPrice) med en størrelse målt i m^2 (mSize) beliggende på en adresse (mAddress).

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

private:
    std::string mAddress;
    double mMinPrice;
    int mSize;
};
```

Opgaven vurderes ud fra en korrekt anvendelse af de tilegnede C++ færdigheder.

Det forventes, at der anvendes referencer, når der er tale om ikke-simple datatyper. Desuden forventes det, at const-keyword anvendes, hvor det er muligt.

a) Implementer constructor, copy og move constructor

Der skal implementeres følgende ting på klassen:

- Constructoren som tager parametrene adresse, minimumspris og størrelse
- getSize og setSize
- getMinimumPrice og setMinimumPrice
- getAddress og setAddress

b) Tilføj operatorer

Tilføj følgende operatorer til klassen

- Extraction operator (>>)
- Insertion operator (<<)

Insertion operatoren skal give et output på dette format:

20 tegn til adresse venstrejusteret, 9 tegn til mindsteprisen højrejusteret, 6 tegn til størrelsen højrejusteret.

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

Test-kode

a)

```
std::cout << "--------------------" << std::endl;
std::cout << "-- Test Opgave 1a --" << std::endl;
std::cout << "--------------------" << std::endl;
Lot lt;
lt.setAddress("Stendyssen 31");
lt.setMinimumPrice(979200);
lt.setSize(720);
const Lot& l = lt;
std::cout << std::left << std::setw(20) << l.getAddress() <<
             std::right << std::setw(9) << l.getMinimumPrice() <<
             std::setw(6) << l.getSize() << std::endl;
std::cout << std::endl;
```

b)

```
std::cout << std::endl;
std::cout << "--------------------" << std::endl;
std::cout << "-- Test Opgave 1b --" << std::endl;
std::cout << "--------------------" << std::endl;

std::string lots = "";
lots += "Stendyssen 1;965430;703\n";
lots += "Stendyssen 3;992160;736\n";
lots += "Stendyssen 5;990540;734\n";
lots += "Stendyssen 7;961380;698\n";
lots += "Stendyssen 9;909540;634\n";
lots += "Stendyssen 11;964620;702\n";
lots += "Stendyssen 13;903060;626\n";
lots += "Stendyssen 15;978390;719\n";
lots += "Stendyssen 17;976770;717\n";
lots += "Stendyssen 19;963000;700\n";
lots += "Stendyssen 21;981630;723\n";
lots += "Stendyssen 23;958950;695\n";
lots += "Stendyssen 25;953280;688\n";
lots += "Stendyssen 27;933840;664\n";

std::stringstream ss(lots);
std::vector<Lot> lotList;
while (!ss.eof()) {
    std::string line;
    std::getline(ss, line);
    if (line.size() > 1) {
        std::stringstream ssLine(line);
        Lot l;
        ssLine >> l;
        lotList.push_back(l);
    }

}
lotList.emplace_back("Stendyssen 29", 907110, 631);
lt.setAddress("Stendyssen 31");
lt.setMinimumPrice(979200);
lt.setSize(720);
```

```
    lotList.push_back(lt);

    for (unsigned int i = 0; i < lotList.size(); ++i) {
        const Lot& l = lotList[i];
        std::cout << l << std::endl;
    }
```

Opgave 2 (1 time 15 minutter, 30 point)

Når nye byggegrunde udstykkes, sælges de ved en hemmelig budgivning. Inden en bestemt dag skal alle bud være afleveret. Hver budgiver har mulighed for at byde på alle grunde med forskellige prioriteter. Hvis man, som budgiver, ikke er højestbydende på sin førsteprioritet, har man altså stadig muligheden for at få sin anden-prioritet osv. Hver budgiver "vinder" altså kun retten til at købe én byggegrund.

a) Implementation af klasser

Vi skal her implementere de klasser som skal bruges til at finde den korrekte "vinder" af budrunden. Vi vil her anvende klassen "Plot" som repræsenterer en grund, klassen "Buyer" som repræsenterer en budgiver samt klassen "Bid" som repræsenterer et bud på en bestemt grund med en given prioritet.

Der er lavet skeletter til klasserne Plot, Buyer og Bid som kan findes som bilag til denne opgave fra næste side.

Del 1: Du skal implementere operator< på klassen Bid, som foretager en sammenligning af to bud (Bid) på baggrund af prioriteten af buddet.

Del 2: Implementer funktionen addBid på klassen Buyer. Funktionen skal oprette et nyt bud med parametrene fra funktionskaldet addBid. Dette nye bud skal lagres i member variablen mBids (bemærk pointere). Herefter skal listen over bud sorteres afhængig af prioriteten, så højeste prioritet (1.) er først i listen. HINT: Det kan være en fordel at benytte std::sort med brug af en lambda-funktion.

BEMÆRK: Du skal IKKE implementere destructors i denne opgave, da vi helt ekstraordinært accepterer alle memory-leaks for at undgå unødig kompleksitet i eksamensopgaven.

b) Udvælgelse af "vinder"

I denne opgave skal vi lave en algoritme, som automatisk udvælger den korrekte "vinder" af hver byggegrund. Til dette skal vi bruge tre køer (brug std::deque eller std::vector) dBuyers, dWinners og dNone. dBuyers repræsenterer budgivere, der endnu ikke har vundet en grund. dWinners repræsenterer budgivere, som har vundet en grund. dNone repræsenterer budgivere, som ingen grund fik. Ideen er skitseret herunder i pseudokode. Det er din opgave at implementere dette i C++. Dette skal implementeres i en funktion (her er parameteren *buyers* en liste af alle budgivere)

```
void findWinningBids(std::vector<Buyer*> buyers).

så længe der er budgivere i dBuyers
    sæt buyer = næste budgiver i dBuyers
    for hvert bud b fra buyer i prioriteret rækkefølge gøres
        hvis bud b er bedre end tidligere bud på grunden g
            hvis grunden g var vundet af en anden køber oldBuyer
                flyt oldBuyer fra køen dWinners til dBuyers
            sæt bedste bud b på grunden g
            flyt budgiver buyer fra dBuyers til dWinners
            afbryd løkken
    hvis ingen bud fra buyer kan vinde en grund
        flyt buyer fra dBuyers til dNone
udskriv vindere
udskriv tabere
```

Plot.h

```
#ifndef PLOT_H
#define PLOT_H

#include <string>

class Bid;

class Plot
{
public:
    Plot() : mBid(nullptr) {}
    Plot(const std::string& address) : mAddress(address), mBid(nullptr) {}

    void setBid(Bid* bid) { mBid = bid; }
    Bid* getBid() { return mBid; }
    const Bid* getBid() const { return mBid; }

    void setAddress(const std::string& address) { mAddress = address; }
    const std::string& getAddress() const { return mAddress; }

private:
    std::string mAddress;
    Bid* mBid;
};

#endif // PLOT_H
```

Buyer.h

```
#ifndef BUYER_H
#define BUYER_H

#include <string>
#include <vector>

#include "Bid.h"
#include "Plot.h"

class Buyer
{
public:
    Buyer() {}
    Buyer(const std::string& name) : mName(name) {}

    void setName(const std::string& name) { mName = name; }
    const std::string& getName() const { return mName; }

    // Laver et bud (Bid) på grunden plot med prisen price
    // Gem bud i mBids vector (bemærk pointer)
    // Skal sortere alle bud afhængig af prioritet, så
    // højeste prioritet er (1.) er forrest i listen.
    // HINT: std::sort med brug af lambda-funktion
    void addBid(Plot* plot, double price, int priority);

    // Returner alle bud lavet af denne køber/budgiver
    std::vector<Bid*> getBids() { return mBids; }

private:
    std::string mName;
    std::vector<Bid*> mBids;
};

#endif // BUYER_H
```

Bid.h

```
#ifndef BID_H
#define BID_H

#include "Plot.h"

class Buyer;

class Bid
{
public:
    Bid() { mBuyer = nullptr; mPlot = nullptr; }
    Bid(Buyer* b, Plot* p, double price, int priority) { mBuyer = b; mPlot = p;
mPrice = price; mPriority = priority; }

    void setBuyer(Buyer* b) { mBuyer = b; }
    Buyer* getBuyer() { return mBuyer; }
    const Buyer* getBuyer() const { return mBuyer; }

    void setPlot(Plot* p) { mPlot = p; }
    Plot* getPlot() { return mPlot; }
    const Plot* getPlot() const { return mPlot; }

    void setPrice(double price) { mPrice = price; }
    double getPrice() { return mPrice; }

    void setPriority(int p) { mPriority = p; }
    int getPriority() { return mPriority; }

    // Implementer en sammenligningsoperator mindre end
    // som sammenligner på baggrund af prioritet
    bool operator<(const Bid& rhs);

private:
    Buyer* mBuyer;
    Plot* mPlot;
    double mPrice;
    int mPriority;
};

#endif // BID_H
```

Test

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

a)

```
std::cout << std::endl;
std::cout << "--------------------" << std::endl;
std::cout << "-- Test Opgave 2a --" << std::endl;
std::cout << "--------------------" << std::endl;

std::vector<Plot*> plots;
plots.push_back(new Plot("Stendyssen 1"));
plots.push_back(new Plot("Stendyssen 3"));
plots.push_back(new Plot("Stendyssen 5"));
plots.push_back(new Plot("Stendyssen 7"));
plots.push_back(new Plot("Stendyssen 9"));
plots.push_back(new Plot("Stendyssen 11"));
plots.push_back(new Plot("Stendyssen 13"));
plots.push_back(new Plot("Stendyssen 15"));
plots.push_back(new Plot("Stendyssen 17"));

std::vector<Buyer*> buyers;
buyers.push_back(new Buyer("Preben"));
buyers.push_back(new Buyer("Ole"));
buyers.push_back(new Buyer("Henrik"));
buyers.push_back(new Buyer("Christian"));
buyers.push_back(new Buyer("Jan"));

buyers[0]->addBid(plots[7], 996408, 1);
buyers[0]->addBid(plots[8], 985400, 2);
buyers[0]->addBid(plots[5], 990312, 3);
buyers[1]->addBid(plots[7], 997315, 2);
buyers[1]->addBid(plots[8], 985123, 1);
buyers[1]->addBid(plots[5], 1001002, 3);
buyers[2]->addBid(plots[7], 1002003, 1);
buyers[2]->addBid(plots[4], 1000000, 2);
buyers[3]->addBid(plots[5], 990123, 1);
buyers[3]->addBid(plots[4], 999000, 2);
buyers[4]->addBid(plots[5], 991322, 1);
buyers[4]->addBid(plots[8], 990000, 2);


for (Buyer* buyer : buyers) {
    for (Bid* bid : buyer->getBids()) {
        std::cout << std::left << std::setw(12) <<
                     bid->getBuyer()->getName() <<
                     std::setw(20) << bid->getPlot()->getAddress() <<
                     std::right << std::setw(9) <<
                     std::setprecision(10) << bid->getPrice() <<
                     std::setw(5) << bid->getPriority() << std::endl;
    }
}
```

b)

```
std::cout << std::endl << std::endl;
std::cout << "--------------------" << std::endl;
std::cout << "-- Test Opgave 2b --" << std::endl;
std::cout << "--------------------" << std::endl;
findWinningBids(buyers);

std::cout << std::endl;
std::cout << std::left << std::setw(12) << "Navn" <<
             std::setw(20) << "Grund" <<
             std::right << std::setw(9) << "Pris" <<
             std::setw(5) << "Pr." << std::endl;

for (Plot* g : plots) {
    Bid* bid = g->getBid();
    if (bid) {
        std::cout << std::left << std::setw(12) <<
                     bid->getBuyer()->getName() <<
                     std::setw(20) << bid->getPlot()->getAddress() <<
                     std::right << std::setw(9) << bid->getPrice() <<
                     std::setw(5) << bid->getPriority() << std::endl;
    } else {
        std::cout << std::left << std::setw(12) << "Ikke solgt" <<
                     std::setw(20) << g->getAddress() << std::endl;
    }

}
```