Eksamen i Introduktion til C++
RB2-ICP

Kursusansvarlig: Thorbjørn Mosekjær Iversen

Eksamensdato: 1. juni 2023

Vigtig information vedrørende eksamen:

Til hver C++ opgave delopgave er der en test-kode, som skal eksekveres og et screenshot af outputet skal afleveres sammen med jeres kode. Hvis en delopgave ikke kan compile, så indsættes i stedet fejlmeddelelsen fra compileren. Som udgangspunkt vurderes opgaverne ud fra hvorvidt koden benytter korrekt c++ syntaks og løser opgaven.

Tilladte hjælpemidler

Offline hjælpemidler er tilladt, inklusiv offline versioner af hjemmesider, (f.eks. `www.cppreference.com`). Det er dog ikke tilladt at benytte kunstig intellegens / AI værktøjer (e.g. GitHub Copilot).

Aflevering

Besvarelsen skal bestå af én zip fil indeholdende følgende:

- Sceenshot af terminaloutputtet efter kørsel af testkoden for hver delopgave
  - dvs. i alt 6 screenshots: opg 1 + opg 2.1 - 2.5
- Kode for opgave 1 (eksklusiv main-fil)
- Kode for opgave 2 (eksklusiv main-fil)

Obs: Enhver form for kopier/indsæt (copy/paste) fra tidligere opgaver eller andre kilder anses som eksamenssnyd. Kode fra denne eksamensopgave (opgave teksten og de udleverede cpp og h filer) må dog gerne kopieres ind.

Opgavesæt

Opgavesættet består af 5 sider:

- 1 forside
- 4 sider med opgaver (opgave 1 + 2.1 - 2.5)

Derudover får i udleveret 3 filer med kode:

- main1.cpp - Testkode til opgave 1
- main2.cpp - Testkode til opgave 2.1-2.5
- cargo.h - Kode som skal benyttes i 2.1.1 (identisk med det der står i opgavesættet)

Opgave 1 (35 point)

I denne opgave skal du skrive en klasse til at repræsentere en 3-dimensionel vektor. Du skal implementere tre konstruktorer, to metoder til at regne et krydsprodukt og én metode til at normalisere vektoren. I forbindelse med normaliseringen skal du implementere en metode til at udregne kvadratroden af et vilkårligt positivt tal.

1. Opret klassen Vector3D. Den skal have 3 **public** variable: `double x`, `double y` og `double z`
2. Opret følgende tre constructorere
   (a) `Vector3D(double x=0, double y=0, double z=0)`
   (b) `Vector3D(std::vector<double> v)`
   (c) `Vector3D(std::string s)` hvor strengen s er på formatet "x y z", f.eks. "21.3 3.33 2"
3. Skriv metoden `Vector3D crossProduct(Vector3D v)`. Metoden skal udregne prikproduktet mellem vektoren selv og vektoren v. Resultatet skal returneres som en ny Vector3D.
   - Hint: krydsproduktet af vektorerne $\begin{pmatrix} a1 \\ a2 \\ a3 \end{pmatrix}$ og $\begin{pmatrix} b1 \\ b2 \\ b3 \end{pmatrix}$ er $\begin{pmatrix} a2 * b3 - a3 * b2 \\ a3 * b1 - a1 * b3 \\ a1 * b2 - a2 * b1 \end{pmatrix}$
4. Skriv metoden `void crossProduct(Vector3D v, Vector3D* output)`. Metoden skal udregne krydsproduktet mellem vektoren selv og vektoren v. Resultatet skal gemmes i den Vector3D som output er en pointer til.
5. Skriv metoden `double squareRoot(double n)` som udregner kvadratroden af et tal. Benyt følgende algoritme:
   (a) Definér startgættet x_0 til at være halvdelen af n.
   (b) Udregn iterativt bedre og bedre estimat af $\sqrt{n}$ vha. formlen $x_{i+1} = (x_i + n/x_i)/2$
   (c) Metoden skal returne resultatet når ændringen fra et gæt til det næste bliver mindre end 0.001 (dvs. x_k, hvor x_k er det første estimat hvor $-0.001 < x_k - x_{k-1} < 0.001$.)
6. Skriv metoden `void normalize()`. Metoden skal normalisere vektoren selv således at den efter metodekaldet har længden én.
   - Beny squareroot metoden som du lige har programmeret til at normalisere vektoren
   - Hint: Længden af en vektor er $\sqrt{a^2 + b^2 + c^2}$

Opgave 2 (65 point)

Du skrive et program der simulerer et varelager (`LoadingArea`) hvor lastbiler (`Truck`) kan komme og afhente normale pakker (`Cargo`) og vigtige pakker (`PriorityCargo`). Hver delopgave svarer til én klasse.

Følgende beskrivelse giver et overblik over opgaven:

- Alle pakker har en vægt og et antal dage indtil levering.
- Vigtige pakker er en særlig type pakker som skal afsted så hurtigt som muligt, uanset antal dage til levering.
- Lastbiler kan lades med en eller flere pakker. De er kun begrænset af at pakkernes samlede vægt ikke må overstige lastbilens kapacitet.
- Varelageret indholder en mængde pakker der er klar til afhentning. Det er muligt at tilføje og fjerne pakker på varelageret. Det er desuden muligt at fylde en lastbil med pakker efter en simpel strategi (der ikke tager højde for hvor hurtigt pakkerne skal afsted), og en avanceret strategi (der læsser pakkerne efter prioritet).

2.1 Cargo

1. Tag udgangspunkt i nedenstående klasse Cargo. Tilføj getter metoden `int getDaysToDelivery()` som returnerer daysToDelivery. Vær opmærksom på at metoden senere skal overrides i en subklasse til Cargo.

```
class Cargo{
private:
    std::string name;
    double weight;
    int daysToDelivery;
public:
    Cargo(std::string n, double w, int d):name(n), weight(w),daysToDelivery(d){
        /* Ensure daysToDelivery is in the interval [1 day, 99 days] */
        if(daysToDelivery<1){
            daysToDelivery = 1;
        }else if(daysToDelivery>100){
            daysToDelivery = 99;
        }

        /* Ensure weight is in the interval ]0kg, 50kg] */
        if(weight<=0){
            weight = 0.001;
        }else if(weight>50){
            weight = 50;
        }
    }

    std::string getName(){
        return name;
    }

    double getWeight(){
        return weight;
    }
};
```

2.2 PriorityCargo

1. Opret klassen `PriorityCargo`. Klassen skal være en subklasse til Cargo. Klassen skal indeholde præcis følgende to metoder (dvs. du må ikke oprette andre metoder eller member variable).

(a) en public constructor `PriorityCargo(std::string n, double w)` som sætter superklassens navn til n, weight til w, og daysToDelivery til 99..

(b) `int daysToDelivery()` som overskriver den tilsvarende metode i superklassen. Den nye metode skal blot returnere 0. (Det sikrer at PriorityCargo kommer på lastbilerne før normale pakker, så længe der er plads til dem).

2.3 Truck

1. Opret klassen `Truck`. Klassen skal have to private variable
   - `double capacity`
   - `std::vector` indeholdende Cargo objekter kaldet **allCargo**
2. Klassen skal have en public constructor `Truck(double c)` som initialiserer capacity til c
3. Skriv metoden `void addCargo(Cargo c)` som tilføjer objektet c til den private vektor af cargo objektor. Du skal være opmærksom på følgende:
   - Rækkefølgen på objekterne i vektoren skal svare til den rækkefølge de blev tilføjet i. Dvs. første objekt er det der først blev tilføjet, andet objekt er det næste objekt der blev tilføjet, osv.
   - Obs. i en senere delopgave vil PriorityCargo også blive tilføjet til en truck med denne metode, hvilket vil medføre slicing af PriorityCargo instansen. Dette er bevidst, da vi er ligeglade med om en pakke er en normal Package eller en vigtig pakke når først pakken er blevet smidt på en lastbil.
4. Skriv en **getter** metode som returnerer en kopi af allCargo
5. Skriv en metode `double remainingCapacity()` som returnerer hvor meget plads der er tilbage på lastbilen.

2.4 LoadingArea

1. Opret klassen `LoadingArea`. Klassen skal have én privat variabel: `availableCargo` som er en std::vector indeholdende pointere til cargo objekter
2. Skriv en **getter** metode for den private variabel availableCargo
3. Skriv en public metode `void addCargo(Cargo &c)`. Metoden skal tilføje en pointer der pejer på c til availableCargo.
   - Alle navnene i availableCargo skal være unikke, så pointeren til c skal ikke tilføjes hvis der i forvejen er en pointer til et Cargo objekt med samme navn som c.
4. Skriv en public metode `void removeCargo(std::string s)` som fjerner et Cargo objekt fra availableCargo hvis det har navnet s.
   - Hint: Husk at addCargo sikrer at alle navnene i availableCargo er unikke
5. Skriv en public metode `void loadTruckSimple(Truck &t)`. Metoden skal flytte én pakke ad gangen fra availableCargo til t. Følgende strategi skal benyttes:
   - Forsøg at tilføje én pakke en ad gangen i den rækkefølge de kommer i availableCargo. Hvis en pakke ikke kan være på lastbilen fordi den er for tung, springes pakken over og der fortsættes til næste pakke indtil alle pakker har været forsøgt tilføjet én gang.
     - Husk at lastbilens kapacitet ikke må overskrides.
     - Husk at pakker der sættes på lastbilen skal fjernes fra varelageret
6. Skriv en public metode `void loadTruckAdvanced(Truck &t)`. Metoden skal flytte én pakke ad gangen fra availableCargo til t. Følgende strategi skal benyttes:

- Pakkerne flyttes til lastbilen én efter én i prioriteret rækkefølge. Hvis en pakke ikke kan være på lastbilen fordi den er for tung, springes pakken over og der fortsættes til næste pakke i prioriteringsrækkefølgen. Dette fortsætter til der ikke længere er plads på lastbilen.
  - Prioriteringsrækkefølgen er som følger:
    * Pakker med lav daysToDelivery prioriteres før pakker med højere daysToDelivery
    * Hvis flere pakker har samme daysToDelivery, har tunge pakker højere prioritet end lettere pakker.
  - Husk at lastbilens kapacitet ikke må overskrides.
  - Hint: Da metoden fortsætter til der ikke længere er plads på lastbilen kan man blive nødt til at gennemgå listen af pakker flere gange, modsat loadTruckSimple metoden hvor alle pakke kun forsøges tilføjet en enkelt gang.