AARHUS UNIVERSITET
INGENIØRHØJSKOLEN

EKSAMEN

Kursus:	I1OPRG/ E1OPRG – Objektbaseret programmering
Eksamensdato:	xx. august 2019, kl. 09.30-11.30 (FP 09.30 – 12.15)
Eksamenstermin:	Reeksamen F2019
Ingeniørhøjskolen udleverer: 4 stk. kladdepapir	
Praktiske informationer: **Digital eksamen** Opgaven tilgås og afleveres gennem den digitale eksamensportal. Eksamensbesvarelsen kan kun afleveres digitalt. Det er ikke muligt, at digitalisere håndskrevne bilag. Eksamensbesvarelsen skal afleveres i PDF-format. Husk angivelse af navn og studienummer på alle sider (første side er nok, se nedenfor), samt i dokumenttitel/filnavn. Husk at uploade og aflevere i Digital eksamen. Du vil modtage en elektronisk afleveringskvittering, straks du har afleveret. Husk at aflevere til tiden, da der ellers skal indsendes dispensationsansøgning.	
Hjælpemidler: Alle hjælpemidler må benyttes, herunder internettet som opslagsværktøj, men det er IKKE tilladt at kommunikere med andre digitalt.	
Særlige bemærkninger: Der er en eller flere filer som bilag, der kan fines på Digital Eksamen, sammen med eksamenssættet.	
Ansvarlig underviser: Frank Bodholdt Jakobsen	

Instruktion

Dette eksamenssæt indeholder 3 opgaver, der enten skal løses i C eller C++, som angivet for hver enkelt opgave.

Hver opgave vægter i den samlede karakter med de %-satser, der står ved hver af dem.

Der er i opgaverne angivet en række header- og implementeringsfiler (.c, .h, eller .cpp), der skal skrives for at løse opgaverne ved at skrive den relevante kode.

Alle disse header- og implementeringsfiler skal ved afleveringen **samles i én fil**, der afleveres **i PDF-format** på Digital Eksamen. Det skal fremgå i denne PDF-fil, hvilken opgave og fil de enkelte kode-dele kommer fra. Dvs. at kode-filer **ikke** skal afleveres enkeltvis eller som zip fil, ligesom Visual Studio solutions og projects **ikke** skal afleveres.

Husk angivelse af navn og studienummer på første side.

Der er en eller flere filer med som bilag, som kan findes på Digital Eksamen, samme sted som eksamenssættet.

Opgave 1 (35 %)

Denne opgave skal løses i C.

I denne opgave skal der arbejdes med brændstoføkonomi (km/l) for biler, udtrykt som antallet af kørte kilometer per liter forbrugt brændstof. Økonomien kan beregnes per tankning eller over en længere periode.

Økonomien kan beregnes efter denne formel:

$$økonomi = \frac{afstand}{forbrugt\ brændstof}$$

Denne beregning skal vi bruge flere gange, og derfor giver det mening at vi laver en funktion, der udfører denne beregning.

Funktionen skal have navnet `beregnOekonomi`.

Den skal have to parametre, der er kommatal, der overføres ved call-by-value, og står for henholdsvis *afstand* og *forbrugt brændstof*.

Den skal returnere en kommatalsværdi, der er den beregnede *økonomi* efter ovenstående formel.

a) Skriv prototypen til funktionen `beregnOekonomi()`, som den skal stå i header-filen Oekonomi.h.

Du skal nu implementere denne funktion. Hvis parameteren *forbrugt brændstof* er 0 (nul), skal funktionen returnere 0, da man ikke kan dividere med 0.

b) Skriv implementationen af `beregnOekonomi()`, som den skal stå i implementationsfilen Oekonomi.c.

Du skal nu teste denne funktion, ved at skrive et program (`main()`), der kalder `beregnOekonomi`.

Du skal anvende følgende testdatasæt:

Afstand	Forbrugt Brændstof	Omtrentligt forventet resultat
800,00	50,00	16,00
200,00	0,00	0,00
1000,00	60,00	16,67

c) Skriv hele filen Opgave1C.c, med en `main()` funktion til et program, der tester `beregnOekonomi()` ved at kalde den med de angivne testdatasæt og udskrive returværdien

Opgaven fortsættes på næste side!

Du skal nu skrive noget kode, der kan hjælpe brugeren med at opgøre økonomien ud fra sin kørebog, ved at afstand og brændstofmængde for hver tankfuld indtastes, og økonomi per tankfuld og totalt udskrives af programmet. Indtastningen afsluttes ved at afstand eller brændstofmængde er mindre end eller lig med 0. Du **skal** bruge funktionen `beregnOekonomi()` hvor det er relevant.

Følgende pseudokode kan bruges til at strukturere programmet:

Nulstil totalAfstand
Nulstil totalBraendstof
Udfør

> *Bed brugeren om at indtaste en afstand for en tank*
> *Indlæs afstanden*
> *Bed brugeren om at indtaste påfyldt brændstof for en tank*
> *Indlæs brændstofmængden*
>
> ***Hvis** afstanden er større end 0 **og** brændstofmængden er større end 0*
> *Beregn økonomi for denne tank og udskriv den*
> *Opdater totalAfstand med afstand*
> *Opdater totalBraendstof med brændstofmængden*

***Så længe** afstanden er større end 0 **og** brændstofmængden er større end 0*

Beregn økonomi for alle tankninger ved hjælp af totalAfstand og totalBraendstof og udskriv den

Du er velkommen til at bruge en anden struktur, fx en struktur der afbryder indtastningen med det samme, når den indtastede afstand ikke er større end 0.

d) Skriv hele filen Opgave1D.c, med en `main()` funktion til et program, der gennemfører beregningerne som beskrevet.

Opgavesættet fortsættes på næste side!

Opgave 2 (25 %)

Denne opgave skal løses i C.

Denne opgave skal arbejde med en funktion, der på en simpel måde skal checke om en C-string er et gyldigt, dansk CPR-nummer.

Den har prototypen:

```
int checkCPR(const char * CPR);
```

Parameteren `CPR` er en C-string, der indeholder CPR nummeret, der skal checkes.

Et gyldigt CPR nummer skal være nøjagtig 11 tegn langt. Det har formatet: `"123456-1234"`, hvor alle tegnene er et ciffer, undtagen det 7. tegn (index 6), som skal være tegnet `'-'`.

Funktionen skal returnere 1 (et) hvis `CPR` opfylder disse betingelser, ellers 0 (nul).

At de første 6 tegn skal være en gyldig dato, skal du **ikke** checke i denne opgave, ligesom andre regler for ægte CPR-numre heller **ikke** skal checkes.

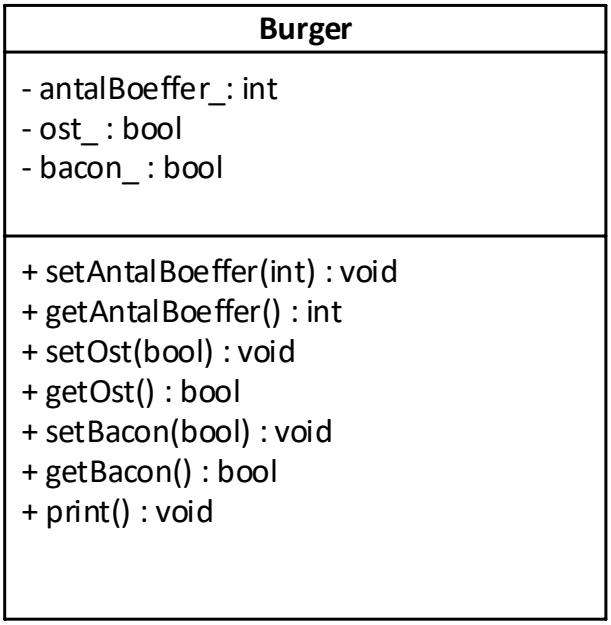
a) **Skriv checkCPR() funktionen. Du skal skrive generel kode med passende løkker og passende betingede sætninger (switch eller if) der checker parameteren CPR.**

Opgavesættet fortsættes på næste side!

Opgave 3 (40 %)

Denne opgave skal løses i C++.

Denne opgave drejer sig om en klasse `Burger`, der kan antages at være en del af et fastfoodsystem. UML klassediagrammet for klassen er angivet nedenfor:



Om klassen `Burger` gælder følgende:

Attributterne skal opfylde følgende betingelser:

Navn	Beskrivelse	Gyldige værdier	Default værdi
`antalBoeffer_`	Antallet af bøffer i burgeren	Antallet af bøffer skal være større end 0 og mindre end eller lig med 3	1
`ost_`	Angiver der er ost i burgeren	Alle gyldige `bool` værdier	`false`
`bacon_`	Angiver der er bacon i burgeren	Alle gyldige `bool` værdier	`false`

Opgaven fortsættes på næste side!

Klassen `Burger` har **kun** følgende parametriserede constructor.

`Burger(int boeffer,`
`bool ost,`
`bool bacon);`

Parametre:

`boeffer`: antal bøffer i burgeren
`ost`: skal der ost i burgeren
`bacon`: skal der bacon i burgeren

Beskrivelse:

Constructoren validerer parametrene, hvis det er relevant, og sætter de relevante attributter. Hvis en parameter ikke er gyldig, indsættes defaultværdien.

Om metoderne for klassen gælder:

`void print() const;`

Parametre:

Ingen

Beskrivelse:

Metoden udskriver burgerens informationer fra attributterne i dette pæne format, hvor "`boef`" eller "`boeffer`" skrives ved 1 henholdsvis flere bøffer, og "`med`" eller "`uden`" skrives ved `true` henholdsvis `false` for den pågældende attribut. For eksempel:

```
Burger med 1 boef uden ost med bacon
```

Eller

```
Burger med 2 boeffer uden ost uden bacon
```

`Set`-metoderne er almindelige metoder, der sætter de tilsvarende attributter. Hvis det er relevant, skal værdier fra parameteren valideres ud fra ovenstående beskrivelse. Der indsættes default værdien, hvis parameteren ikke indeholder en gyldig værdi.

`Get`-metoderne er almindelige metoder, der bare returnerer den tilsvarende attribut.

Opgaven fortsættes på næste side!

a) Skriv hele header-filen Burger.h for klassen `Burger`.

b) Implementer alle metoderne og constructoren i filen Burger.cpp.

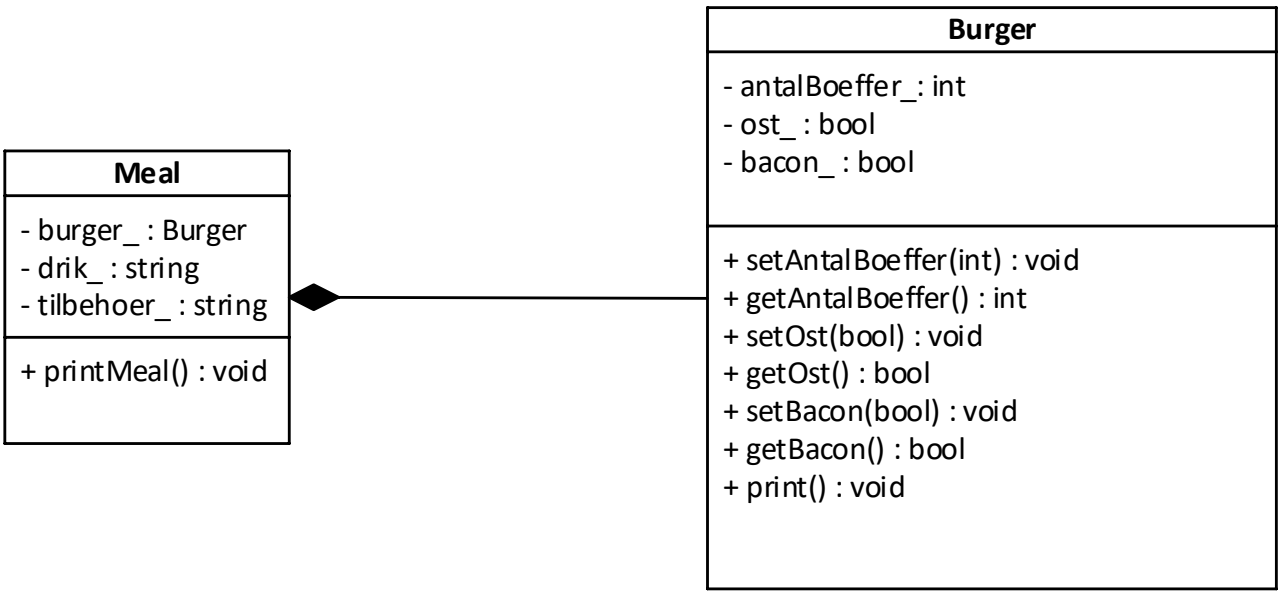
Nu skal klassen testes i et program, ved at skrive en `main()` funktion i en ny fil, Opgave3C.cpp.

Programmet skal oprette to `Burger`-objekter, et med 1 bøf uden bacon og ost, og et med det hele: 3 bøffer, bacon og ost. Derefter skal disse objekter udskrives.

c) Test klassen `Burger` ved at skrive hele filen Opgave3.cpp med `main()` funktionen til et program, der udfører ovennævnte test.

Opgaven fortsættes på næste side!

Burgere kan købes som en del af et "Meal", og dette er beskrevet i UML på følgende måde.



Om klassen `Meal` gælder følgende:

Attributterne skal opfylde følgende betingelser:

Navn	Beskrivelse	Gyldige værdier	Default værdi
`burger_`	Et Burgerobjekt	Alle gyldige burgerobjekter	Som for Burger
`drik_`	Hvilken drikkevare der købes	Alle gyldige `string` værdier	`"Coca Cola"`
`tilbehoer_`	Hvilket tilbehør, der købes	Alle gyldige `string` værdier	`"Pommes Frites"`

Opgaven fortsættes på næste side!

Klassen `Meal` har **kun** følgende parametriserede constructor.

```
Meal(int boeffer,
     bool ost,
     bool bacon,
     string drik,
     string tilbehoer);
```

Parametre:

`boeffer`: antal bøffer i burgeren
`ost`: skal der ost i burgeren
`bacon`: skal der bacon i burgeren
`drik`: hvilken drik, der købes
`tilbehoer`: hvilket tilbehør, der købes

Beskrivelse:

Constructoren bruger parametrene til at initialisere de relevante attributter.

Om metoderne for klassen gælder:

```
void printMeal() const;
```

Parametre:

Ingen

Beskrivelse:

Udskriver et `Meal` objekt på en pæn måde.

Definitionen af klassen `Meal` kan du finde i den medfølgende fil Meal.h, som du kan indsætte i dit Visual Studio projekt. `printMeal()` metoden er er implementeret i den delvise implementationsfil Meal.cpp, som du kan indsætte i dit Visual Studio projekt, og arbejde videre på for at løse opgave d)

Du kan bruge metoden `printMeal()` til at teste din implementation i en `main()` funktion, men det er **ikke** en del af opgaven.

d) Skriv constructoren for `Meal`, som den skal stå i implementationsfilen Meal.cpp