Opgave 5 - Småopgaver i problemløsning + mere øvelse i klasser

Dette kursus forkuserer på to vigtige aspekter af programmering: 1. Basal problem løsning vha. simple programmerings koncepter, så som loops, if-statements, og vectorer, og 2. strukturering af kode vha. indkapsling af funktionalitet i klasser og disse kalssers interaktion. I denne uge er der øvelser i begge dele. Del 1 består af syv små individulle programmeringsøvelser som træner basal problemløsning. Del 2 er en opgave i at skrive en klasse der kan håndtere regning med brøker. Der er nok at tage fat på, så sørg for at prioritere de opgaver som træner det I har sværest ved.

Opgave 1.1

Der findes naturlige tal, hvor om det gælder, at summen af cifrene opløftet i deres positions potens er lig tallet. Fx er 135 et sådant tal, fordi summen af en i første potens, tre i anden potens og fem i tredje potens er lig tallet selv (1+9+125=135). Opgaven går ud på at skrive en metode, som kan udskrive samtlige disse tal, som er mindre end 1000, vha. std::cout.

Opgave 1.2

Denne opgave går ud på at skrive en metode med følgende signatur:

```
bool neighbors(std::string s)
```

Der returneres *true*, hvis parameteren s indeholder nabobogstaver i 'stigende' rækkefølge, fx opfylder *gh* kriteriet; men det gør *hg* ikke. Findes der ikke nabobogstaver returneres *false*. Opgaven skal løses for små bogstaver i det engelske alfabet (`abcdefghijklmnopqrstuvwxyz`).

Eksempler:

'hsgafkroduitjsla' returnerer *false* – ingen nabobogstaver

'krstuoinnaqwlp' returnerer *true* – s og t er nabobogstaver

Opgave 1.3

Denne opgave går ud på at skrive en metode med følgende signatur:

```
bool sumOfTwoEqualsThird(std::vector<int> vec)
```

Parameteren *vec* er en vector af heltal større end nul. Metoden returner *true*, hvis det forekommer, at summen af to elementer på forskellige pladser er lig med et tredje element. Hvis ikke returneres *false*.

Eksempler:

{17,29,512,58,43} returnerer *false*

{46,39,18,15,21} returnerer *true* (18+21=39)

{45,29,31,58,99} returnerer *false*; 29 må kun bruges én gang

Opgave 1.4

Et palindrom er et ord, som staves ens både forfra og bagfra. Eksempler: mellem, otto, kajak. Skriv en metode som kan afgøre om et ord er et palindrom. Signatur:

```
bool isPalindrom(std::string text)
```

Sætninger kan også være palindromer, fx *en af dem der tit red med fane.* Udvid metoden således at den også kan håndtere sætninger.

Opgave 1.5

Det klassiske danske indregistreringsnummer til personbiler og visse andre køretøjer består af 7 karakterer. De to første er store bogstaver, og de fem sidste er tal, hvoraf de første ikke må være nul. Eksempler på valide indregistreringsnumre: MA39604 og KD47612. Skriv en metode som kan validere den type numre.

Opgave 1.6

Opgaven går ud på at skrive en metode, som kan afgøre om eventuelle parenteser i en tekststreng er matchede. Det vil sige, at der skal være lige mange start- og slutparenteser, og på et vilkårligt sted i teksten må der ikke have forekommet flere slut- end startparenteser.

Eksempler: (())() er ok; ()()((())) er ok; ()()) er ikke ok;)(()() er ikke ok. Løsningen skal tage højde for, at der kan forekomme andre tegn end start- og slutparenteser i teksten.

Opgave 1.7

I denne opgave går ud på at skrive funktionen:

```
void forkortBroek(int input_taeller,
                  int input_naevner,
                  int& output_taeller,
                  int& output_naevner)
```

Funktionen tager en brøk som input og outputter den forkortede brøk. Hint: en brøk kan forkortes hvis der findes et heltal der går op i både tæller og nævner.

Opgave 2

Denne opgave går ud på at skrive en klasse, som kan håndtere brøker og hedder **`Broek`**. Klassen har to private member variable, som er af typen **`int`** og indeholder henholdsvis brøkens tæller (**`taeller`**) og nævner (**`naevner`**).

Nedenfor er angivet header-filen til klassen **`Broek`**.

Metoderne, som regner på brøker (**`adder, subtraher, multiplicer`** og **`divider`**) fungerer på den måde, at aktuelt objekt er første operand og parameteren er anden operand, fx hvis aktuelt objekt er 1/5 og parameteren 2/3, skal metoden **`divider`** returnere brøken 3/10. Ingen af operanderne må ændre værdi.

Alle brøker skal være forkortede, dvs. hvis konstruktor kaldes med parametrene 6 og 12, skal det oprettede objekt indeholde tallene 1 og 2. Det samme gælder de brøker, som beregningsmetoderne returnerer.

Løsningen skal ikke nødvendigvis, men må gerne forholde sig til, at en brøks nævner ikke kan være nul.

Det kan anbefales at lave hjælpemetoder, fx **`forkortBroek`** fra opgave 1.7. Nedenstående .h-fil er derfor ikke nødvendigvis komplet.

```
class Broek
{
public:
            Broek();
            Broek(int taeller, int naevner);

            int getTaeller();
            int getNaevner();
            Broek adder(Broek b);
            Broek subtraher(Broek b);
            Broek multiplicer(Broek b);
            Broek divider(Broek b);
            Broek operator+(Broek b);
            Broek operator-(Broek b);
            Broek operator*(Broek b);
            Broek operator/(Broek b);

private:
            int taeller;
            int naevner;
};
```

De sidste fire metoder inden destruktoren er *operator overloads*, således at man fx kan lægge brøker sammen ved at benytte + operatoren.