Exercise 5

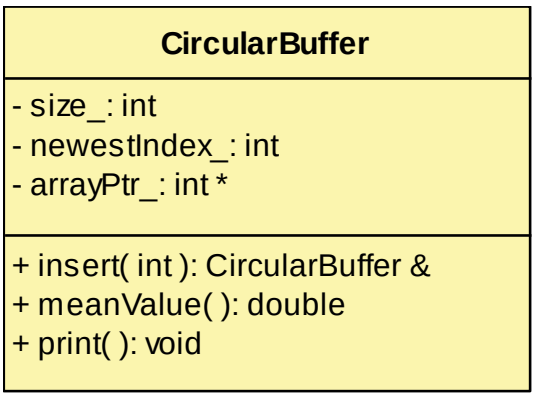
Indledning:

I denne øvelse skal du anvende ***dynamisk hukommelsesallokering***, dvs. hvor hukommelse først allokeres når programmet kører – altså dynamisk. Dynamisk hukommelsesallokering bruges i mange sammenhænge, og en af de typiske anvendelser er, når størrelsen på en buffer er afhængig af brugerens input. I det tilfælde vil størrelsen på bufferen medtages som parameter til en constructor, der så vil allokere hukommelse internt. I C++ skal du huske, at alt hvad du allokerer, skal du også deallokere – ellers får du en ***memory leak***, som med tiden kan føre til at programmet bryder ned. Du skal også holde styr på, hvad din pointer peger på efter `delete`, så du undgår at have en ***dangling pointer***.

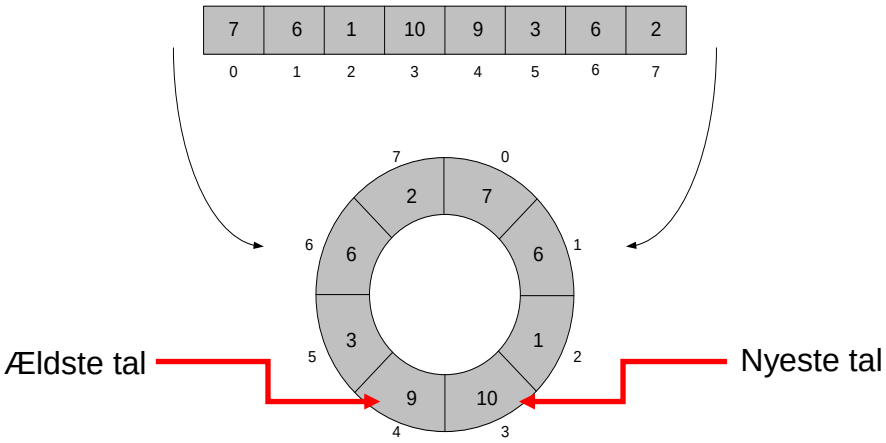
Gennem denne øvelse vil du opnå erfaring med allokering og deallokering af dynamisk hukommelse vha. C++ operatorerne `new` og `delete`, og med hvordan du tilgår det hukommelse du allokerer.

Exercise 5.1:

I denne øvelse skal du impementere klassen `CircularBuffer`, hvori tal kan indsættes og hvorfra de kan udskrives. UML klassediagrammet for klassen er givet herunder (bemærk at diagrammet ikke er fuldstændigt, der mangler bla. a. en constructor)



Internt i klassen anvendes et dynamisk allokeret array (via `arrayPtr_`) til at indeholde værdierne i den cirkulære buffer. Arrayet skal anvendes som om enderne på det var ”sat sammen”. Det betyder, at når arrayet er fyldt op, begynder man forfra med at indsætte på plads 0 – dvs. at *det ældste tal* overskrives når et nyt indsættes. Dette er illustreret nedenfor, hvor `size_= 8`, `newestindex_= 3` og `arrayPtr_[newestindex_] = 10`. Det næste tal vil så blive indsat på det ældste tals plads, altså på plads 4.



a) Diskutér med en medstuderende, hvordan den cirkulære buffer virker. Hvad indeholder den fra starten af, hvad sker der i detaljer når bufferen er fyldt op og nye værdier indsættes.

b) Diskutér med en medstuderende, hvor du vil allokere hukommelsen som `arrayPtr_` skal pege på, og hvor du vil deallokere den?

c) Diskutér med en medstuderende, hvad sammenhængen er mellem den plads der indeholder det nyeste tal og den plads hvorpå det næste element skal indsættes?

d) Returtypen til `insert()` er `CircularBuffer &`. Overvej hvorfor og hvad det kan bruges til.

Klassens constructor skal som nævnt anvende *dynamisk hukommelsesallokering* til at allokere et arrayet dynamisk med det antal pladser, der angives som parameter til constructoren, og initialisere samtlige pladser i arrayet til 0.

Beskrivelsen af klassens øvrige metoder kan du se herunder.

`CircularBuffer & insert( int )`

Parametre: integer med det tal, der skal indsættes i bufferen

Returværdi: reference til det aktuelle objekt

Beskrivelse: Funktionen skal indsætte tallet på første ledige plads eller på ***ældste*** plads hvis bufferen er fuld. Hvis bufferen er fuld overskrives det ældste tal.

`double meanValue( )`

Parametre: ingen

Returværdi: middelværdien af alle tallene i bufferen

Beskrivelse: Funktionen skal beregne og returnere middelværdien af samtlige tal (også nuller) i bufferen.

`void print( )`

Parametre: ingen

Returværdi: ingen

Beskrivelse: Funktionen udskrive tallene i bufferen – ***nyste først*** og ældste sidst.

e) Implementér klassen `CircularBuffer`.

f) Skriv et testprogram som tester klassens metoder. Husk at overveje dine testdata og dine *forventede* output, og sammenlign det med dit *faktiske* output.

g) Har du testet anvendelse af returværdien fra `insert()` (jvf. spørgsmål d) ?

Exercise 5.2:

Du skal nu ændre klassen `CircularBuffer` så den kan indeholde objekter af typen `Point` i stedet for heltal. Klassen `Point` kan du hente fra Exercise 1.
Bemærk, at *middelværdien* af et antal punkter i sig selv er et punkt.
Skriv derefter et ***nyt*** testprogram til klassen (bemærk, at du skal bruge ***begge*** testprogrammer i en senere øvelse).
Bemærk også, at du desværre bliver nødt til at ændre i implementeringen af metoden `meanValue()`. Det er *ikke* godt, for det betyder, at den ikke er generel, og at den f.eks. ikke længere kan bruge til heltal. Men det råder vi bod på i exercise 8 ☺

Exercise 5.3:

I forberedelsen til teorilektionen har du set på klassen `IntArray`. Arrayet i denne klasse er erklæret dynamisk. Arrayets størrelse kan ændres mens programmet kører vha. metoden `setSize()`. Implementeringen af `IntArray` findes sammen med denne øvelse.

Det data, som er i arrayet, mistes dog når størrelsen ændres !

Det er jo bare ikke godt nok ☹

Omskriv metoden `setSize()` så det data, der er i arrayet, bibeholdes. Hvis størrelsen sættes ***ned***, forsvinder data selvfølgelig fra de ”øvre pladser”, men resten skal bibeholdes. Hvis størrelsen sættes ***op***, skal alle data bevares, og de ”nye” pladser skal sættes til 0.