NTNU
Kunnskap for en bedre verden

Institutt for datateknikk og informasjonsvitenskap

Eksamensoppgave i TDT4102 – Prosedyre- og objektorientert programmering

Faglig kontakt under eksamen: Lasse Natvig

Tlf.: 906 44 580

Eksamensdato: 15 August 2019

Eksamenstid (fra-til): kl. 0900 – 1300

Hjelpemiddelkode/Tillatte hjelpemidler:
C: Spesifiserte trykte og håndskrevne hjelpemidler tillatt:
Bjarne Stroustrup, PROGRAMMING, Principles and Practice Using C++.
Bestemt, enkel kalkulator tillatt.

Målform/språk: Bokmål

Antall sider: 11 inkludert vedlegg

Vedlegg 1: Ark om lærebokas Graph_lib m.m. (1 side)

Kontrollert av:

Informasjon om trykking av eksamensoppgave
Originalen er:
1-sidig X **2-sidig ☐**
sort/hvit ☐ **farger X**

Dato Sign

Merk! Studenter finner sensur i Studentweb. Har du spørsmål om din sensur må du kontakte instituttet ditt. Eksamenskontoret vil ikke kunne svare på slike spørsmål.

Generell introduksjon

Les gjennom oppgavetekstene nøye. Noen av oppgavene har lengre tekst, men dette er for å gi kontekst, introduksjon og eksempler til oppgavene.

Når det står "*implementer*", "*definer*" eller "*lag*" skal du skrive en fungerende implementasjon: hvis det handler om en funksjon skal du skrive deklarasjonen med returtype og parametertype(r) og hele funksjons-kroppen.

Når det står "*deklarer*" er vi kun interessert i funksjons- eller klassedeklarasjonen. Typisk vil dette være deklarasjoner du vanligvis finner i header-filer.

Hvis det står "*forklar*" står du fritt i hvordan du svarer, men bruk enkle kodelinjer og/eller korte tekst-forklaringer og vær kort og presis.

Dersom du mener at opplysninger mangler i en oppgaveformulering, gjør kort rede for de antagelser og forutsetninger du finner nødvendig.

Hver enkelt oppgave er ikke ment å være mer omfattende enn det som er beskrevet. Noen oppgaver fokuserer bare på enkeltfunksjoner og da er det utelukkende denne funksjonen som er tema. Andre oppgaver er "oppskriftsbasert" og vi spør etter funksjoner som utgjør deler i et program, eller forskjellige deler av en eller flere klasser. Du kan velge selv om du vil løse dette trinnvis, eller om du vil lage en samlet implementasjon, men sørg for at det går tydelig frem hvilke spørsmål du har svart på hvor i koden din.

Husk at funksjonene du lager i en deloppgave ofte er ment å skulle brukes i andre deloppgaver. Selv om du står fast på en deloppgave bør du likevel prøve å løse alle eller noen av de etterfølgende oppgavene ved å anta at funksjoner fra tidligere deloppgave er implementert.

All kode skal være i C++. Det er ikke viktig å huske helt korrekt syntaks for bibliotekfunksjoner. Oppgaven krever ikke kjennskap til andre klasser og funksjoner enn de du har blitt godt kjent med i øvingsopplegget, eller som står beskrevet i læreboka.

Det er ikke nødvendig å ha med include-statement eller vise hvordan koden skal lagres i filer.

Hele oppgavesettet er arbeidskrevende og det er ikke forventet at alle skal klare alt. Tenk strategisk i forhold til ditt nivå og dine ambisjoner!

Deloppgavene i de "tematiske" oppgavene er organisert i en logisk rekkefølge, men det betyr ikke at det er direkte sammenheng mellom vanskelighetsgrad og nummereringen av deloppgavene.

Hoveddelene av eksamensoppgaven teller i utgangspunktet med den andelen som er angitt i prosent. Den prosentvise uttellingen for hver oppgave kan likevel bli justert ved sensur basert på hvordan oppgavene har fungert. De enkelte deloppgaver kan også bli tillagt forskjellig vekt.

Oppgave 1: Kodeforståelse (20 %)

Svar på denne måten på et vanlig svararк:
1a) … ditt svar her, 1b) … ditt svar her … osv.

1a) Hva skrives ut ?

```
int a = 20;
float b = a * 2 + 1;
int c = b++;
b += b / 100;
cout << a << " " << b << " " << c << endl;
```

1b) Hva skrives ut ?

```
int i = 1;
int j = 1;
while (j < 10) {
        j += i;
        i = j - i;
        cout << j << " ";
}
cout << endl;
```

1c)

```
int f(char h, char *e, char &i) {
        h++;
        *e = h;
        i += 2;
        return h;
}
```

```
char h = 'h';
char e = 'e';
char i = 'i';
h = f(h, &e, i);
cout << h << e << i << endl;
```

Gitt funksjonen `f` til venstre, hva skrives ut av programmet til høyre?

1d)

```
int safe(int n, int d) {
        if (d == 0) {
                throw d;
        }
        return n / d;
}
```

```
try {
        for (int n = 4; n > 0; n--) {
                cout << n;
                cout << safe(n, n - 2) << " ";
        }
}
catch (int e) {
        cout << "! ";
}
cout << endl;
```

Gitt funksjonen `safe` til venstre, hva skrives ut av programmet til høyre?

1e)

```
string apply(string s,
             map<char, string> &rules) {
   string s2 = "";
   for (int i = 0; i < s.length(); i++) {
       s2 += rules[s[i]];
   }
   return s2;
}
```

```
map<char, string> rules;
rules['A'] = "AB";
rules['B'] = "A";
string s = "A";
for (int i = 0; i < 4; i++) {
        cout << s << " ";
        s = apply(s, rules);
}
cout << endl;
```

Gitt funksjonen `apply` til venstre, hva skrives ut av koden til høyre?

1f) Hva skrives ut ?

```
string a = "trol";
string b = "rolo";

char c = 't';
for (int i = 0; i < 7; i++) {
       int k = a.find(c);
       cout << a[k];
       c = b[k];
}
cout << endl;
```

1g)

```
char rot13(char c) {
  if (c >= 'a' && c <= 'z') {
       c -= 'a';
       c = 'a' + ((c + 13) % 26);
  }
  return c;
}
string rot13(string s) {
   for (int i = 0; i < s.size(); i++) {
              s[i] = rot13(s[i]);
       }
   return s;
}
```

```
string msg = "catz";
cout << rot13('a') << rot13('n')
     << rot13('z') <<  " ";
cout << rot13(msg) << " ";
cout << rot13(rot13(msg)) <<
endl;
```

Gitt funksjonen `rot13` til venstre, hva skrives ut av koden til høyre? (Her trenger du vite at de 26 små bokstavene i det engelske alfabetet kommer etter hverandre i ASCII-tabellen, som «abcdefghijklmnopqrstuvwxyz».)

1h)

```
class S {
public:
       S() {}
       char f() { return 'S'; }
};

class U : public S {
public:
       char f() { return 'U'; }
};
```

```
S s;
U u;
S& r = u;
S* p = &r;
cout << s.f() << u.f() << r.f()
     << p->f() << endl;
```

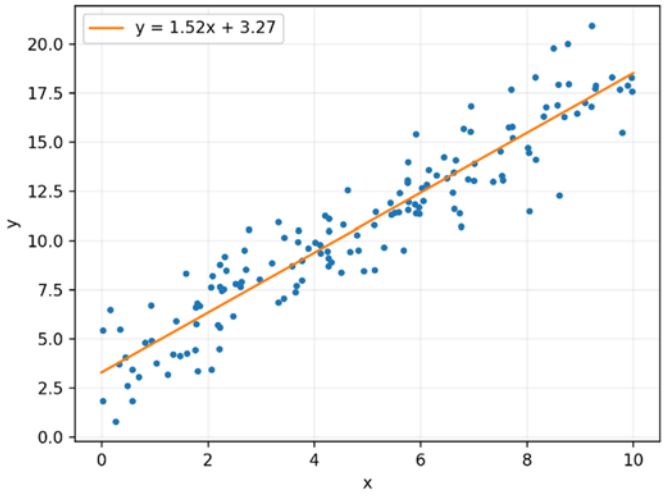
Hva skrives ut av koden til høyre når klassene er deklarert som til venstre?

Oppgave 2: Lineær regresjon (25 %)

Lineær regresjon er en analyse-teknikk som ble benyttet så tidlig som på starten av 1800-tallet. Teknikken brukes for å forsøke å finne en lineær sammenheng mellom to variabler x og y på formen

$$y = ax + b$$

Utgangspunktet er en samling med data for x og y, og i enkel lineær regresjon regner vi ut hvilken rette linje som best beskriver sammenhengen mellom x og y. Et eksempel er vist i Figur 1. Dette er en statistisk metode og *oppgaveteksten vil beskrive den matematikken du trenger for å kunne programmere løsninger på deloppgavene*. (Lineær regresjon har svært mange anvendelser og brukes sammen med andre mer avanserte metoder bl.a. i «nye» fagområder som «data science» og kunstig intelligens (Eng. Artificial Intelligence, AI).)



Figur 1. Eksempel på lineær regresjon. De blå punktene er (x, y)-par plottet i et vanlig x-y-koordinatsystem, og ved å bruke lineær regresjon har vi beregnet den oransje linjen som er det linjestykket som best passer til disse punktene (vårt datasett). Formelen for linjestykket er vist oppe til venstre.

2a) Implementer funksjonen `double sum(vector<double>& x)` som returnerer summen av alle tallene i en vektor `x`.

2b) Implementer funksjonen `double mean(vector<double>& x)` som returnerer gjennomsnittsverdien av tallene i vektoren `x`.

2c) Comma-separated values (CSV) er et vanlig filformat brukt til lagring av data på tabellform (rader og kolonner). En CSV-fil har én rad per linje, og hver linje er delt opp i kolonner separert med komma *eller mellomrom*. Et utsnitt av en CSV-fil benyttet for denne oppgaven er vist i Figur 2. Implementer funksjonen `void read_csv(string filename, vector<double>& x, vector<double>& y)` som leser data fra en CSV-fil med to kolonner x og y. Her er `filename` navnet på CSV-filen og `x` og `y` referanser til to vektorer av `double` som skal fylles med data fra filen, der første kolonne er `x` og andre er `y`. Du kan anta at kolonnene i CSV-filen er separert (adskilt) med et mellomrom. Om filen ikke kan åpnes skal funksjonen kaste et unntak som en tekststreng bestående av «`Couldn't read file`» og filnavnet `filename`.

70	152
85	214
83	203
77	219
56	152
90	215

Figur 2.

2d) Vi skal nå implementere kode for lineær regresjon. Gitt to vektorer med data x og y, ønsker vi å finne en rett linje som best mulig beskriver sammenhengen mellom x og y. Fra matematikken vet vi at en rett linje er gitt av formelen $y = ax + b$ hvor a er stigningstallet mens b er punktet linjen krysser y-aksen (dvs. når x = 0). (For lineær regresjon skal vi her benytte «minste kvadraters metode» oppfunnet av Gauss: linjen skal trekkes gjennom de gitte punktene slik at summen av kvadratene av avstandene fra disse punktene til linjen minimeres, hvor avstanden måles i y-retningen). Nedenfor er formlene du skal bruke for dette tilfellet:

$var = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - \bar{x})^2$	$cov = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - \bar{x})(y_i - \bar{y})$
$a = \frac{cov}{var}$	$b = \bar{y} - a\bar{x}$

Her betyr Σ summasjon for alle indeks-verdier $i = 0 \,..\, n-1$, der x_i angir et element i vektoren x. Videre er $\bar{x}$ vanlig notasjon for gjennomsnittsverdien til x. Tilsvarende for y. Merk at du ikke trenger forstå matematikken som ligger bak dette, og at du kan løse senere deloppgaver uten å løse denne deloppgaven.

Implementer funksjonen `pair<double, double> linreg(vector<double>& x, vector<double>& y)` som beregner verdier for *a* og *b* som beskrevet ovenfor. Funksjonen skal returnere begge verdiene som et `std::pair p` hvor `p.first` er *a* og `p.second` er *b*.

2e) Nå som vi har funnet en linje som best mulig passer vårt datasett kan vi bruke linjen til å prediktere (dvs. beregne en antatt) verdi for y gitt en vilkårlig x-verdi. Implementer funksjonen `vector<double> linpred(vector<double>& x, double a, double b)` som returnerer *en vector med alle y-verdiene* gitt ved å bruke $y = ax + b$ for *hver av* x-verdiene i x-vektoren.

2f) Selv om vi har funnet en linje betyr det ikke at linjen passer bra. Et mål på hvor bra linjen passer datasettet er R^2 som er et tall mellom 0 og 1 der 0 betyr dårligst og 1 betyr perfekt. R^2 beregnes fra formelen:

$$R^2 = 1 - \frac{\sum_{i=0}^{n-1}\left(y_i - \text{y\_pred}_i\right)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2}$$

Der y_pred_i er element nr. *i* fra vektoren returnert av funksjonen `linpred()` i forrige deloppgave. Implementer funksjonen `double r2(vector<double>& y, vector<double>& y_pred)` som sammenlikner `y` (de observerte verdiene) og `y_pred` (de predikterte verdiene, fra `linpred`) ved å beregne R^2 som gitt av formelen ovenfor.

2g) Vi er nå klar for å bruke lineær regresjon til å finne en sammenheng mellom to vektorer av data x og y. Skriv en main-funksjon som:

1. Leser inn x- og y-data fra filen "data.csv"
2. Beregner a og b ved hjelp av lineær regresjon
3. Regner ut R^2 for linja $y = ax + b$
4. Skriver ut til skjermen verdiene til a, b, og R^2

Oppgave 3: Bysykler (25%)

NTNU satser både på miljø og studenters helse, og har sammen med sykkel-general Ricardo plassert bysykler på ulike stasjoner i Trondheim. Du er ansatt av rektor for å programmere applikasjon *GunnarBikes* som hjelper Ricardo med å vite hvor syklene er og å organisere transport av sykler slik at alle stasjoner har et passelig antall sykler på plass hver morgen.

Figur 3. Utklipp av skjermbilde fra oppgaven Bysykler.

3a) Programmet benytter en datatype med navnet `Location` som er vist i Figur 4, der `name` er et *unikt* navn på plassen, og `p` er plassering på et kart. Datatypen `Point` er fra lærebokas grafikk-bibliotek (Se vedlegg). **Implementer konstruktøren** for `Location` (deklarert i Figur 4) *utenfor* klassedeklarasjonen og bruk initialiseringsliste.

```
struct Location {
    string name;
    Point p;
    Location(string str, Point pt);
};
```

Figur 4. Datatypen `Location`

3b) **Definer en klasse** `BikeStation` som inneholder følgende *private* medlemsvariable: `loc` av typen `Location`, `capacity` og `bikes` som begge skal være `unsigned int`, samt `display` som skal være en `Vector_ref<Shape>` og som benyttes for grafikken i oppgaven. Videre skal klassen inneholde *deklarasjoner* av en set- og en get-funksjon for medlemsvariabelen `bikes`, og en *inline-implementasjon* av en funksjon `getName()` som skal returnere `loc.name`. (Den endelige versjonen av denne klassen vil inneholde mer, men du skal begrense deg til det som er nevnt ovenfor i denne deloppgaven)

3c) **Implementer set- og get-funksjonen** for `bikes` som du deklarerte i forrige deloppgave.

3d) Skriv en medlemsfunksjon `string BikeStation::status()` som skal returnere en tekststreng som inneholder en kort tekstlig rapport om status på en stasjon med sykler. Formatet på tekststrengen er vist i Figur 3 ovenfor som "`10 out of 30`" der det første tallet er verdien på `bikes` (hvor mange sykler som er på stasjonen) og det andre er verdien til `capacity` (antall plasser for parkering av bysykler).

3e) Skriv konstruktøren til `BikeStation`
`BikeStation::BikeStation(Location where, unsigned int cap, unsigned int numBikes)`
Denne skal initialisere `loc`, `capacity` og `bikes` med verdier fra parameterlisten. Videre skal den legge pekere til følgende *grafiske elementer* inn i medlemsvariabelen `display`:

1) en peker til et `Rectangle`-objekt som har øvre venstre hjørne i samme punkt som gitt av medlemsvariabelen `loc` (se del-oppgave 3b). Bredde og høyde skal være som gitt av to konstanter `dispWidth` og `dispHeight` som du har tilgjengelig i koden din. Rektangelet skal fylles med fargen `Color::white`.
2) en peker til et `Text`-objekt med navnet på stasjonen, plassert ut fra samme punkt som rektangelet, med fargen `Color::blue`. Teksten skal ha font-størrelse 20.
3) en peker til et `Text`-objekt som viser `status` til stasjonen ved å vise den tekststrengen som returneres av `status()` fra forrige deloppgave i det hvite rektangelet, plassert 2 piksel (bildepunkter) inn fra rektangelets venstre kant og 15 piksel nedenfor rektangelets øvre kant. Denne teksten skal ha fargen `Color::black`, og du trenger ikke bry deg om font-størrelsen (standard/default passer fint).

Se for øvrig Figur 3 og bruk informasjon fra vedlegget.

3f) For å teste applikasjonen *GunnarBikes* ønsker vi å simulere en dag med sykling. Du skal skrive en funksjon **`simulateOneDay()`** som tar inn en vektor av pekere til alle bysykkelstasjonene (`vector<BikeStation*>`). Funksjonen skal gjøre følgende:

- *Prøve* å utføre `ridesPerDay` mulige sykkelturer, der `ridesPerDay` er en gitt konstant.
- For hver *potensielle* sykkeltur skal det velges en *tilfeldig* stasjon hvor en ønsker å ta ut en sykkel. Bruk `rand()` fra standardbiblioteket (se vedlegget) til å velge en tilfeldig stasjon for å ta ut en sykkel og en tilfeldig stasjon for å parkere den. Dette representerer ett ønske om en sykkeltur.
- For at en syklist skal være sikker på at man kan sette fra seg sykkelen på den stasjonen en sykler til, så er det slik at man må kunne reservere en ledig plass på ende-stasjonen ved uttak av en sykkel. På denne måten er en garantert at sykkelturen kan gjennomføres. En tilstrekkelig og nødvendig betingelse for at en sykkeltur er vellykket er derfor at det finnes minst en ledig sykkel på avreise-stasjonen og at det er minst en ledig sykkel-plass på destinasjons-stasjonen. For hver sykkeltur (fra forrige punkt) som ønskes utført så må en sjekke denne betingelsen, og en ønsker statistikk over antall avslåtte sykkelturer. Det antallet skal telles opp i et `map<string, int>` som returneres av funksjonen. Her er nøkkel-feltet det entydige (unike) navnet på stasjonen en ønsker å sykle i fra, og heltallet er antallet sykkelturer derfra som må avslås.
- Du kan anta `BikeStation` også har en get-funksjon `unsigned int getCapacity()` for å lese ut variabelen `capacity`.

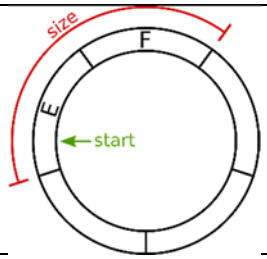
3g) Skriv en funksjon **`printStats`**() som skriver ut til konsollet en oversikt over avslåtte sykkelturer slik som vist i Figur 5. Funksjonen tar som input et map av samme type som returneres av `simulateOneDay`() i forrige deloppgave, og linjene skal være alfabetisk sortert etter navnet på stasjonen (Festningen kommer før Marinen osv.)

```
Unsuccessful rides:
70 bike trips refused at Festningen
101 bike trips refused at Marinen
180 bike trips refused at Pirbadet
62 bike trips refused at Samfundet
```

Figur 5.

Oppgave 4: Ringbuffer og testing (30%)

Et buffer er et område i minne som brukes til midlertidig lagring av data før det kan bli behandlet videre. I et sirkulært buffer, eller *ringbuffer*, vil man begynne forfra igjen når man skriver (eller leser) forbi det siste elementet. Figur 6 viser en mulig tilstand i et slikt buffer med 5 posisjoner, se også tabellen nedenfor. Et ringbuffer anvendes ofte når data skal overføres mellom to ulike enheter og prosesseres hos mottakeren i samme rekkefølge som det kommer inn (som en kø). Minnet i en datamaskin er i utgangspunktet lineært, så for at det skal "se ut" som det er sirkulært trenger vi litt ekstra programlogikk.



Figur 6. Ringbufferet i tilstand 4. Starten på bufferet (posisjon 0) er øverst i figuren og bufferet er «med urviseren». "start" peker på første gyldige element i bufferet (her er start = 4), og "size" angir antall gyldige elementer som er lagret (her er size = 2).

I et ringbuffer må vi holde styr på starten på gyldige data i bufferet (`start`), samt hvor mye gyldige data som er lagret (`size`). I tillegg må vi vite maks størrelse på bufferet (`capacity`). I tabellen nedenfor har vi vist 5 tilstander for et buffer med plass til 5 elementer (`capacity = 5`).

Tilstand nr.	Bufferinnhold	Verdi for `start` og `size`	Neste operasjon på ringbuffer `rb`
1	`[ _ _ _ _ _ ]`	`start=0, size=0`	`rb.write("ABC");`
2	`[ A B C _ _ ]`	`start=0, size=3`	`rb.write("DEF");`
3	`[ F B C D E ]`	`start=1, size=5`	`s = rb.read(3);`
4	`[ F _ _ _ E ]`	`start=4, size=2`	`s = rb.read(-1);`
5	`[ _ _ _ _ _ ]`	`start=1, size=0`	

I starten (tilstand nr. 1) er bufferet tomt, og `start = 0` og `size = 0`. Hvis vi *skriver* tre elementer `A B C` til bufferet, blir resultatet som vist i tilstand 2. Operasjonen (koden) for å gjøre dette er vist i siste kolonne. Merk at posisjon i bufferet er nummerert fra 0 og oppover til `size` - 1. Om vi nå skriver tre nye elementer `D E F` til bufferet, vil vi skrive forbi slutten av bufferet og begynne på starten igjen, som vist i tilstand 3. Vi vil dermed overskrive det første elementet. Siden vi overskriver starten på gyldige data, inkrementeres `start` til å peke på det neste elementet `B` (som er lagret i posisjon 1). Legg merke til at nå blir `size = capacity` siden bufferet er fullt.

Lesing fra ringbufferet vil frigjøre plass tilsvarende antall elementer lest. Hvis vi leser 3 elementer, returnerer bufferet `B C D`. Etter lesingen får bufferet tilstand nr. 4, og det er denne tilstanden som er vist i Figur 6. For read()-funksjonen er det slik at argumentet -1 benyttes for å angi lesing av resten av innholdet i bufferet. Om vi leser de siste 2 elementene (`E` og `F`) blir bufferet tomt og vi får tilstand nr. 5. Merk at `start = 1` selv om bufferet er tomt, siden starten på gyldige data kan være hvor som helst i det underliggende bufferet.

Under har vi deklarert klassen `RingBuf` som er et ringbuffer der elementene er av type `char`. I denne oppgaven skal du implementere deler av denne klassen.

```
class RingBuf {
private:
	char *buf;     // The underlying buffer
	int capacity; // Capacity of underlying buffer (max size)
	int start;     // Start of valid data
	int size;      // Size of valid data (0 if empty)
public:
	RingBuf(int capacity);
	RingBuf(const RingBuf &other); // copy constructor
	RingBuf(RingBuf &&other); // move constructor
	~RingBuf();
	RingBuf& operator=(RingBuf rhs); // assignment operator, copy assignment
	void write(char c); // write a character to the buffer
	void write(string s); // write a string of characters to the buffer
	char read(); // read a char from the buffer
	string read(int count); // read a number of chars from the buffer
	string peek();
	friend void testRingBuf();
};
```

4a) Implementer konstruktøren til `RingBuf`. Konstruktøren skal allokere bufferet `buf` med plass til `capacity` antall elementer. Husk å initialisere `capacity`, `start` og `size` som beskrevet i innledningen.

4b) Implementer kopikonstruktøren.

4c) Implementer destruktøren.

4d) Implementer movekonstruktøren.

4e) Implementer tilordningsoperatoren (Eng. Copy assignment).

4f) Implementer funksjonen `void RingBuf::write(char c)` som skriver et element `c` til bufferet. Husk å oppdatere både `start` og `size` som beskrevet i starten av oppgaven.

4g) Implementer funksjonen `char RingBuf::read()` som leser et element fra bufferet. Om bufferet er tomt skal funksjonen gi en melding om dette ved å kaste et unntak (tekststreng). Husk å oppdatere `start` og `size` som beskrevet i starten av oppgaven. Du trenger ikke nulle ut elementet som ble lest.

4h) Implementer funksjonen `void RingBuf::write(string s)` som skriver en tekststreng til bufferet.

4i) Implementer funksjonen `string RingBuf::read(int count)` som leser `count` antall elementer fra bufferet. Hvis `count` er større enn `capacity` eller `count` er -1, skal funksjonen lese ut alle de gyldige elementene som er lagret i bufferet.

4j) Implementer funksjonen `string RingBuf::peek()` som returnerer innholdet i bufferet uten å fjerne elementer. Den er spesielt nyttig for å teste at `RingBuf` oppfører seg som forventet. `peek()` returnerer en `string` med alle de gyldige elementene i bufferet, dvs fra og med `start` til (men ikke med) `start + size`.

4k) Testing er en viktig del av programmering. Skriv en funksjon `testRingBuf()` som sjekker at `RingBuf` oppfører seg som forventet. Testen skal opprette et ringbuffer og utføre operasjonene som ble beskrevet og vist i tabellen i innledningen. Etter hver operasjon skal du sjekke at verdien til medlemsvariablene `start` og `size` er riktige og at `peek()` returnerer riktige verdier. Sjekk også returverdien til `read()` når du tester lesefunksjonaliteten. Bruk `assert()` som beskrevet i vedlegget.

...---oooOooo---...

This one-page appendix contains 3 sections: 1) rand() & srand(), 2) Graph_lib, 3) assert()
(Note that this is only brief descriptions or parts of the declarations, with focus on what you might need. `...` indicates lines have been deleted.)

1) rand() and srand()

```
int rand(); /* generates and returns a pseudo-random number between 0 and
               RAND_MAX (0 and RAND_MAX included) */
```

2) Elements from Graph_lib.h used in the textbook

```
struct Point {
        int x; int y;
};

template <class T>
class Vector_ref { // also covered in detail in the textbook
        ...
        void push_back(T* p);
        T& operator[](int i);
        int size() const;
        ...
}

class Shape {
        ...
public:
        ...
        void set_fill_color(Color col);
        void set_color(Color col);
        ...
};

struct Rectangle : Shape {
        Rectangle(Point xy, int ww, int hh); // Upper left corner given by xy
        ...
};

struct Text : Shape {
        // the point is the bottom left of the first letter
        Text(Point x, const string& s);
        void set_font_size(int s);
        ...
}
```

3) assert(condition)

- assert checks if condition is true. If condition is *false*, assert will abort the program and output an error message about the file and line number where the assertion failed.
- assert has no return value
- assert is useful for testing and debugging of a program. It is used to verify that the program behaves as expected, e.g., that a variable has the expected value, a function returns the correct result etc.

…---ooo0ooo---…