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: 22 Mai 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: 10 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 = 5;
int b = 10 / a--;
int c = 10 % (++b);
cout << a << " " << b << " " << (c += 3);
```

1b)

```
int func(int a, int& b, long* c) {
        a = 300;
        b = -2;
        *c = 1;
        return (a + b + *c);
}
```

```
int i = 0;
long *k = new long(200000);
cout << func(i, i, k) << " ";
cout << i << " " << *k;
```

Gitt funksjonen func() til venstre, hva skrives ut av koden til høyre?

1c) Hva skrives ut ?

```
string s1 = "aaa";
string s2{ "bbb" };
string s3 = s1 + s2;
cout << s3 << " " << s3.size() << " ";
cout << s3[2] << s3[3] << " ";
s1[1] = 'A' + 2;
cout << s1 <<  " " << 'c' - 'a';
```

1d) Hva skrives ut?

```
int a = 1;
{
        int a = 5;
        int* p1 = new int(6);
        int* p2 = p1;
        p1 = &a;
        *p2 = 3;
        cout << a << " " << *p1 << " " << *p2;
}
```

1e)

```
int grey(const char* cStr) {
        int c = 0;
        if (*cStr != '\0') {
                if (*cStr == '1') c++;
                c = c + grey(cStr + 1);
        }
        return c;
}
```

```
cout << grey("0110101100110");
cout << " " << grey("111001");
```

Gitt funksjonen grey() til venstre, hva skrives ut av koden til høyre?

1f) Hva skrives ut ?

```
vector<char> charVec {'d', 'c', 'b', 'a'};
set<char> charSet {'d', 'c', 'b', 'a'};
auto it = charVec.begin();
while (it != charVec.end()) cout << *it++;
cout << " ";
auto it2 = charSet.begin();
while (++it2 != charSet.end()) cout << *it2;
```

1g)

```
struct dlNode {
       double data = 0;
       dlNode* next = nullptr;
       dlNode* prev = nullptr;
       dlNode(double num) :
              data{ num } {}
};
```

```
dlNode* n1 = new dlNode(1.1);
dlNode* n2 = new dlNode(2.2);
dlNode* n3 = new dlNode(3.3);
n1->next = n2; n2->prev = n1;
n2->next = n3; n3->prev = n2;
while (n3->prev != nullptr) {
       n3 = n3->prev;
}
cout << n1->next->data << " ";
cout << n3->data;
```

Gitt deklarasjonen av `dlNode()` til venstre, hva skrives ut av koden til høyre?

1h)

```
class Animal {
public:
  string name;
  Animal(string name):name{name} {}
  virtual string toStr() {
    return "Animal: " + name;
  }
};
class Dog : public Animal {
public:
  Dog(string name):Animal{name} {}
  string toStr() { return "Dog: " + name; }
};
```

```
Animal cat("Jesperpus");
Dog dog("Festus");
Dog dog2("Bob");
Animal *a = &cat;
Dog *d = &dog;
Animal *a2 = &dog2;
cout << a->toStr() << ", ";
cout << d->toStr() << ", ";
cout << a2->toStr() << ", ";
cout << (*a2).name;
```

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

1i)

```
template<typename T>
T mySum(const vector<T>& v) {
   T value = 0;
   for (int i=0; i<v.size(); i++) {
       if ((i % 2) != 0) {
          value += v[i];
       }
   }
   return value;
}
```

```
vector<double> dv{1.1,2.2,3.3,4.4};
vector<int> iv{ 10, 20, 30, 40, 50 };
cout << mySum(dv) << " " << mySum(iv);
```

Gitt template-funksjonen mySum() til venstre, hva skrives ut av koden til høyre?

1j)

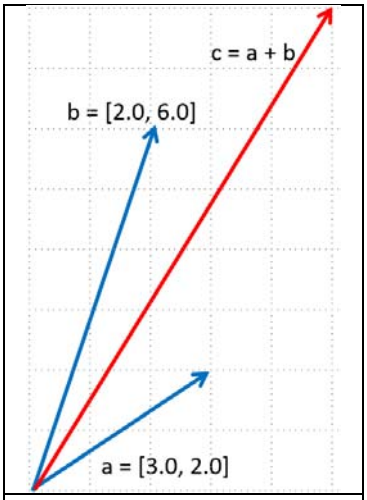
```
struct Pers { string name;
   Pers(string s) : name{ s } {}
   Pers(const Pers& other)
      { name = other.name; cout << "Pers-copy\n"; }
   Pers& operator=(const Pers& rhs) {
     name = rhs.name; cout << "Pers-assign\n";
     return *this;
   }
};
struct Stud : public Pers { int id;
   Stud(string s, int id) : Pers{ s }, id{ id } { }
   Stud(const Stud& other) : Pers{ other }
      { id = other.id; cout << "Stud-copy\n"; }
   Stud& operator=(const Stud& rhs) {
      Pers::operator=(rhs); id = rhs.id;
      cout << "Stud-assign\n";
      return *this;
   }
};
```

```
Stud a{ "a", 100 };
Stud c{ "c", 300 };
Stud b{ a };
Stud d{ "d", 400 };
d = c;
```

Gitt deklarasjonene til venstre, hva skrives ut av koden til høyre?

Oppgave 2: Vektor2d (30%)

I denne oppgaven skal du lage en klasse for matematiske vektorer i planet (2d, dvs. 2 dimensjoner). En vektor består av to komponenter (tall): x og y. Vi bruker notasjonen a = [x, y] for å angi en vektor a med verdier x og y. I Figur 1 er det vist to eksempel-vektorer a og b med blått. Vi bruker et vanlig koordinatsystem med origo i nedre venstre hjørne, positiv x-verdi er mot høyre og positiv y-verdi oppover.



Figur 1.

2a) Deklarer klassen `Vector2d` med to medlemsvariabler x og y av type double. Medlemsvariablene skal være public. Deklarer og implementer konstruktøren inline.

2b) Lengden til en vektor er definert som sqrt($x^2 + y^2$) der sqrt() er en funksjon i standardbiblioteket for å beregne kvadratrot. Implementer en medlemsfunksjon `Vector2d::length()` som returnerer lengden.

2c) Addisjon av vektorer gjøres komponentvis og resultatet er en ny vektor. Hvis vektorene [x0, y0] og [x1, y1] adderes så blir resultatet vektoren [x0 + x1, y0 + y1]. Dette er vist i Figur 1 der den røde pilen er vektoren c = a + b. Implementer **`operator+`** for addisjon av objekter fra klassen `Vector2d`. Operatoren skal returnere summen av to vektorer, og skal implementeres utenfor klassen (altså ikke være medlemsoperator).

2d) En vektor kan multipliseres med en konstant (skalar), dvs. [x, y] * k . Resultatet blir en ny vektor [x * k, y * k]. Implementer **`operator*`** for å kunne utføre slik multiplikasjon der [x,y] er et `Vector2D` objekt og k er av type double. Operatoren skal implementeres som medlemsoperator.

2e) Følgende kode gir kompileringsfeil

```
Vector2d a{1,2};
Vector2d b = 2.0 * a;
error: invalid operands to binary expression ('double' and'Vector2d')
       Vector2d b = 2.0 * a;
```

Hvorfor gir denne koden feilmelding, selv om vi antar at `operator*` fra forrige deloppgave er implementert?

2f) Definer (implementer) overlasting (Eng. overloading) av utskriftsoperatoren **`operator<<`** for et objekt fra klassen `Vector2d`. Formatet skal være [x, y]. Vi bryr oss ikke om presisjon/antall desimaler. Deklarasjonen ser slik ut: `ostream & operator<<(ostream& out, const Vector2d& v)`

2g) Skriv en funksjon **`vectorSum`**`(const vector<Vector2d>& vectors)`
som returnerer et `Vector2d` objekt som er summen av alle vektorene i `vectors`.

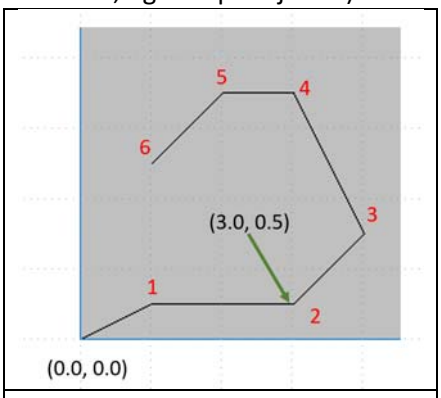
2h) Vi skal nå bruke en STL-vector av `Vector2d` -objekter for å representere et spor i planet. Sporet kan f.eks. være fra en robot, fra et forsøksdyr eller en jogger, og må være innenfor området (0.0, 0.0) til (10.0, 10.0) (sporingsområdet). I Figur 2 viser den sorte streken et spor som starter i koordinat-posisjon (0.0, 0.0) og er representert med `vector<Vector2d> track = {{1.0, 0.5}, {2.0, 0.0}, {1.0, 1.0}, {-1.0, 2.0}, {-1.0, 0.0}, {-1.0, -1.0}};` Sporet består av 6 *linjestykker*, hver representert ved et `Vector2d`-objekt. Posisjonene etter hver forflytning er angitt med røde tall nr. 1, 2, 3 … til og med 6 som er sluttposisjon i dette tilfellet. Vi kan tenke oss

at måleenheten er kilometer (km) og at joggerens klokke lagrer posisjonen ved hvert 10 minutt. Som et eksempel har vi vist med den grønne pilen at joggeren er ved *posisjon* (3.0, 0.5) etter 20 minutter. (Husk at hvert element i `track` bare gir *forflyttningen* de siste 10 minutter, og *ikke* posisjonen). Skriv en funksjon

```
void trackStats(const vector<Vector2d>& track)
```

som rapporterer lengden på joggeturen i km med 2 desimaler etter punktum, maksimal-hastigheten i hele meter pr minutt for turen, og sluttposisjon relativt til startpunktet. For sporet i figur 2 vil utskriften bli slik:

```
Length: 9.18 km, max-speed: 224 m/min, ended at [1, 2.5]
```



Figur 2.

2i) Vi ønsker å *erstatte* de elementer i vektoren som ville føre til en posisjon utenfor sporingsområdet med en null-vektor, dvs. et `Vector2d`-objekt med verdi {0.0, 0.0}. Skriv en funksjon **`cleanTrack`**() som tar inn en referanse til et spor og gjør denne operasjonen element for element. For å få full score på denne oppgaven skal du bruke `transform` fra <algorithm> med et *funksjons-objekt*. Deklarasjonen for `transform` er vist i vedlegget. Andre løsninger kan også gi poeng.

Oppgave 3: Matvarepriser (10 %)

Du ønsker å hjelpe dine medstudenter med å finne billige matvarer, og har laget et lite program som lagrer matvarepriser i en liten database. Databasen er deklarert som et map av set: `map<string, set<Food>>` der datatypen `Food` er deklarert som i Figur 3. Nøkkelen i map-et er navnet på matvaren. For hver matvare en kjenner minst en pris på så er prisen(e) lagret i et set av ett eller flere `Food` -objekter.

```
struct Food {
        string name;
        double price;
        string where;
};
```

Figur 3.

3a) For at databasen beskrevet i innledningen skal fungere må du implementere `operator<` for datatypen `Food` slik at `set<Food>` er sortert med de laveste prisene først i set'et. Implementer **`operator<`** slik.

3b) Skriv en funksjon `void` **`addPrice`**`(map<string, set<Food>>& db, Food fp)` som legger en kopi av `Food` objektet `fp` inn i databasen `db`.

3c) Skriv en funksjon `void` **`printAllPrices`**`(const map<string, set<Food>>& db)` som skriver ut til konsollet alle priser som er registrert i databasen `db`. (Vi bryr oss ikke om antall desimaler i prisen). Formatet er gitt ved eksempel i Figur 4.

```
Bread:
25.45 COOP-Moholt
Milk:
19.99 COOP-Lohove
20.5 KIWI-Moholt
21.5 Rema-Eberg
```

Figur 4.

3d) Skriv en funksjon

```
void bestPrice(const map<string, set<Food>>& db, string name)
```

som skriver ut den laveste prisen for en matvare, angitt som siste parameter. For databasen i eksemplet med bare 4 priser skal funksjonen for `name` = "Milk" skrive ut:

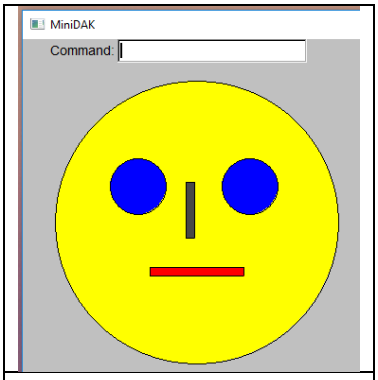
```
Best price for Milk is 19.99 at COOP-Lohove
```

og for `name` = "Juice" skrive ut:

```
No price for Juice
```

Oppgave 4: Mini-DAK (40 %)

Datamaskin-Assistert Konstruksjon (DAK), eller Computer Aided Design (CAD), brukes innen en rekke fagområder for å lage tekniske tegninger på en datamaskin. I denne oppgaven skal du skrive kode for et enkelt DAK-program kalt "Mini-DAK". Vi skal benytte oss av grafikkrammeverket fra læreboka som du har blitt kjent med i øvingsopplegget. Visuelt ser Mini-DAK ut som i Figur 5: et vindu med en tekstboks der brukeren kan skrive inn kommandoer. Gyldige kommandoer er:



Figur 5.

- **`rect color x y w h`** betyr tegn et rektangel med bredde **`w`** og høyde **`h`** på angitt posisjon (**`x`**, **`y`**). Posisjonen (**`x`**, **`y`**) angir øverste venstre hjørne av rektangelet. Rektangelet får fargen **`color`**.
- **`circle color x y r`** betyr tegn en sirkel med radius **`r`** på angitt posisjon (**`x`**, **`y`**). Posisjonen (**`x`**, **`y`**) angir senter av sirkelen. Sirkelen får fargen **`color`**.
- **`save filename`** betyr lagre tegningen til filen **`filename`**
- **`load filename`** betyr last inn tegning fra filen **`filename`**

I tillegg til kommandoene skal brukeren kunne flytte på elementer i tegningen med musa. Når brukeren trykker et sted i vinduet vil programmet få inn koordinater (x, y) som angir posisjonen til musepekeren. (0,0) er her øvre venstre hjørne i vinduet. Det blir da opp til programmet du skriver å finne ut om noen av tegneelementene (rektangler eller sirkler) er under musepekeren.

4a) Implementer funksjonen

```
bool is_inside_rectangle(int x, int y, int r_x, int r_y, int r_width, int r_height)
```

som returnerer true hvis punktet (x, y) er inni et rektangel angitt av posisjonen (r_x, r_y), bredde r_width og høyde r_height, ellers false. Vi regner kanten av rektangelet som utenfor.

4b) Implementer funksjonen

```
bool is_inside_circle(int x, int y, int c_x, int c_y, int c_rad)
```

som returnerer true bare hvis punktet (x, y) er inni en sirkel med senter (c_x, c_y) og radius c_rad. Fra matematikken vet vi at et punkt (x,y) er innenfor en sirkel hvis $((x - c\_x)^2 + (y - c\_y)^2) < c\_rad^2$, ellers returneres false.

4c) I tegnekommandoene forklart ovenfor er fargen (**`color`**) en tekststreng, f.eks. `"blue"`, mens tegnefunksjonene i rammeverket tar som argument et Color-objekt. Vi trenger dermed å oversette fra tekststreng til Color-objekt. Anta det finnes en global variabel `colors`, definert som:

```
map<string, Color> colors = {
        {"red", Color::red},
        {"blue", Color::blue},
        ... // and more colors
};
```

Implementer funksjonen **`string_to_color()`**, som tar som argument en farge som tekststreng og returnerer et `Color` -objekt etter oppslag i `colors`. Om ikke tekststrengen finnes i `colors` skal det kastes et unntak.

4d) Det vil også være behov for å oversette fra et Color-objekt til en tekststreng. Implementer funksjonen `string` **`color_to_string`**`(Color color)` som returnerer navnet til fargen color basert på colors-mappet nevnt i forrige deloppgave. Om fargen *ikke* eksisterer skal funksjonen returnere "unknown color". Du kan anta at likhetsoperatoren (==) er implementert på Color objekter.

4e) Vi deklarerer en felles *abstrakt* klasse `DAKShape` for alle tegneelementer i Mini-DAK, se Figur 6.

```
class DAKShape {
protected:
	Shape &shape; // The underlying Shape to draw
	DAKShape(Shape &s) : shape{ s } { }
public:
	virtual bool is_inside(const Point xy) const = 0;
	virtual string to_string() const = 0;
	virtual ~DAKShape() { }
	void attach_to(Graph_lib::Window & win) { win.attach(shape); }
	void move(int dx, int dy) { shape.move(dx, dy); }
	void set_color(Color c) { shape.set_fill_color(c); }
	Color get_color() const { return shape.fill_color(); }
};
```

Figur 6.

Deklarer klassen `DAKRectangle` som er en spesialisering av `DAKShape`. Klassen skal ha et Rectangle-objekt som medlemsvariabel. Sørg for at `DAKShape` vet om dette objektet. Konstruktøren til `DAKRectangle` skal ta samme argumenter som konstruktøren til Rectangle (se vedlegg). Skriv klasse-deklarasjonen til `DAKRectangle` og implementer dens konstruktør inline.

4f) Implementer medlemsfunksjonen `DAKRectangle::to_string()` som returnerer en `string` som beskriver et rektangel-objekt som en tekststreng med samme format som beskrevet innledningsvis i denne oppgaven. Med andre ord på formen:

```
rect color x y w h
```

der color, x, y , w og h skal være tekstlig beskrivelse av verdiene til disse 5 medlemsvariablene.
Hint: Husk at du kan kalle Rectangle::point(0) for å få returnert et objekt av type `Point` som angir posisjonen til øverste venstre hjørne til rektangelet.

4g) Implementer medlemsfunksjonen `DAKRectangle::is_inside(Point p)` som returnerer true hvis punktet `p` er inni rektangelet, og ellers false. Du har bruk for samme hint som i forrige deloppgave.

4h) I resten av denne oppgaven antar vi at du også har skrevet en klasse `DAKCircle` som har akkurat samme oppbygging og funksjonalitet som `DAKRectangle`. Videre har vi en klasse `MiniDAK` som er en

```
class MiniDAK : public Graph_lib::Window {
private:
	In_box cmd_box; // input box for commands
	Vector_ref<DAKShape> shapes; // vector of shapes in the drawing
	int selected_shape; // index into shapes or -1 if none are selected
	Point mouse; // mouse position
public:
	MiniDAK(int w, int h);
	void add_shape(DAKShape *shape);//adds a shape to the window (stored in shapes)
	int handle(int event); // handle events (See text)
	void on_enter_pressed();
	void on_mouse_click(int x, int y);
	void on_mouse_drag(int x, int y);
	void save(string filename);
	void load(string filename);
	void do_command(string command);
};
```

Figur 7.

spesialisering av `Window` og er deklarert som vist i Figur 7.

Når brukeren er ferdig med å skrive inn en kommando i tekstboksen (Se Figur. 5) og trykker Enter-knappen på tastaturet, skal kommandoen tolkes og utføres av programmet. Vi har programmert medlemsfunksjonen `handle`() slik at medlemsfunksjonen `on_enter_pressed`() blir kalt når brukeren trykker Enter-knappen. Implementer **`on_enter_pressed`**() som skal hente ut innholdet i tekstboksen `cmd_box` og gi tekststrengen videre som parameter i et kall på funksjonen `do_command`(). Om kommandoen var vellykket (`do_command` kastet *ikke* et unntak) skal innholdet i `cmd_box` tømmes. Om `do_command` kastet et unntak, skal `on_enter_pressed` skrive ut en informativ feilmelding til konsollet. (Funksjonen do_command() skal ikke implementeres før i deloppgave 4m)

4i) Når brukeren trykker et sted i vinduet med musen, kalles funksjonen `on_mouse_click`(). Funksjonen får inn koordinatene til musepekeren gjennom parametrene. Koordinat-posisjonen til musen må lagres til senere bruk. Her ønsker vi å sjekke om brukeren trykket på noen av DAKShape-objektene lagret i `shapes`. Om et DAKShape-objekt befinner seg under musepekeren, skal indeksen til det objektet lagres i `selected_shape`. Om det er mer enn ett DAKShape under musepekeren, skal det objektet blant disse med høyest indeks i shapes-vektoren bli valgt. Om ingen DAKShape befinner seg under musepekeren, settes `selected_shape` til -1. Implementer **`on_mouse_click`**().

4j) Funksjonen `on_mouse_drag`() blir kalt gjentatte ganger når brukeren flytter på musa mens museknappen holdes inne, dvs. etter `on_mouse_click`(). Hvis brukeren har valgt et DAKShape, som angitt i forrige deloppgave, skal `on_mouse_drag`() flytte det til den nye posisjonen hver gang funksjonen er blitt kalt. Implementer **`on_mouse_drag`**(). Merk at Shape::move() kun tar inn *endringen* og ikke absolutt posisjon (se vedlegget). Husk å oppdatere koordinatene til musepekeren.

4k) Implementer funksjonen **`save`**() som lagrer tegningen til filen `filename` ved å skrive ut alle rektangel og sirkel-objekter som en liste med kommandoer som spesifisert i introduksjonen, en kommando per linje. Om filen ikke kan åpnes skal det kastes et unntak.

4l) Implementer funksjonen **`load`**() som laster inn en tegning fra fil. Om filen ikke kan åpnes skal det kastes et unntak. Funksjonen skal kalle **`do_command`**() for hver tegne-kommando i filen. Om **`do_command`**() kaster et unntak, skal en feilmelding skrives ut til konsollet med informasjon om feilen og hvilket linjenummer i filen feilen oppsto.

4m) Funksjonen **`do_command`** () tar inn en kommandostreng og forsøker å tolke denne i samsvar med formatet spesifisert i innledningen av oppgaven. Brukere gjør ofte feil, og kommandoer kan inneholde feil. Funksjonen må derfor sjekke at kommandoen er på korrekt format og kaste et unntak hvis det oppstår en feil. Du kan anta at angitt filnavn i kommandoen load ikke vil inneholde blanke tegn. For å forenkle oppgaven skal du bare **bry deg om de to kommandoene** `rect` og `save`.

…---oooOooo---…

The STL algorithm **transform** is specified in <algorithm> as shown below. It reads an input sequence given by iterators `first1` and `last1`, calls the unary operation `unary_op` to each element, and writes the result of that operation to `d_first`. The input is an «half-open sequence» [`first1`, `last1`) where the element given by `first1` is included but the element given by `last1` is not included. Note that the result can be delivered to the input-container and you do not need to use the return value.

```
template< typename InputIt, typename OutputIt, typename UnaryOp >
OutputIt transform(InputIt first1, InputIt last1, OutputIt d_first, UnaryOp unary_op);
```

Elements from **Graph_lib.h** and other include files used in the textbook (Note that this is only parts of the declarations, with focus on what you might need. `...` indicates lines have been deleted.)

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

struct Rectangle : Shape {
      Rectangle(Point xy, int ww, int hh);
      ...
      int height();
      int width();
};

struct In_box : Widget {
      In_box(Point xy, int w, int h, const string& s);
      ...
      string get_string();
      void clear_value(); // empties the in_box
};

class Shape {
      ...
public:
      virtual void move(int dx, int dy); // move the shape +=dx and +=dy
      Point point(int i);
      ...
};
```

…---oooOooo---…