NTNU

Kunnskap for en bedre verden

Institutt for datateknologi og informatikk

Prøve-eksamen (Auditorieøving 2)

TDT4102 - Prosedyre- og objektorientert programmering

Faglig kontakt under eksamen: Rune Sætre, Özlem Özgöbek og und.ass.er

Tlf: 452 18 103 (Rune), 457 90 081 (Özlem),

Epost: satre@ntnu.no, ozlem.ozgobek@ntnu.no, tdt4102-undass@idi.ntnu.no

"Eksamen" Dato: 23. April. 2020

Eksamenstid (fra-til): 0800 - 1200

Hjelpemiddelkode/Tillatte hjelpemidler: F - Alle trykte og håndskrevne hjelpemidler (og digitale opptak). Alle kalkulatorer er tillatt, inkludert VS Code.

Annen informasjon: Samarbeid er ikke tillatt.

Merk! Studenter vil IKKE finne sensur for denne prøve-eksamen i Studentweb, siden dette bare er en test. Har du spørsmål om din sensur må du IKKE kontakte instituttet ditt. Eksamenskontoret vil IKKE kunne svare på slike spørsmål heller.

Denne PDF-filen er bare et tilleggs-dokument og inneholder nøyaktig det samme som den egentlige prøven tilgjengelig i Inspera.

REGLER OG SAMTYKKER

Dette er en individuell eksamen. Du har ikke lov til å kommunisere (gjennom web-forum, chat, telefon, hverken i skriftlig, muntlig e.l. form), ei heller samarbeide med noen andre under eksamen.

For å kunne fortsette til eksamen må du forstå og samtykke i følgende:

NTNU-policy for juks ved eksamen:

(**Norsk**) https://innsida.ntnu.no/wiki/-/wiki/Norsk/Juks+på eksamen

Under Eksamen:

Jeg vil IKKE motta hjelp fra andre.

- Aksepter

Jeg vil IKKE hjelpe andre eller dele løsningen min med noen.

- Aksepter

Jeg vil IKKE copy-paste noe kode fra noen eksisterende online/offline kilder. (Du kan se, og deretter skrive din EGEN versjon av koden).

- Aksepter

Jeg er klar over at jeg kan stryke uavhengig av hvor korrekt svarene mine er, hvis jeg ikke følger reglene og/eller IKKE aksepterer disse utsagnene.

- Aksepter

Jeg er også klar over at juks kan ha alvorlige konsekvenser, som å bli ute-stengt fra universiteter og få annullert eksamensresultater.

- Aksepter

I tillegg: Jeg gir samtykke til at mine eksamensdetaljer kan brukes til forskningsformål av forskeren ved NTNU for å forbedre undervisningen i fa-get.

- Ikke bruk resultatene mine til forskning.
- Aksepter

Generell Intro

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.All kode skal være i C++. Du kan bruke VS Code eller et hvilket som helst annet utviklingsmiljø du foretrekker. 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. Du står ellers fritt til å bruke de deler av C++ du finner nødvendig.

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.

På slutten av eksamen ber vi deg laste opp en .zip-fil som inneholder de komplette kodefilene. Koden din trenger IKKE å være feilfri eller å kjøre uten problemer for deg å passere, men det er en fordel å sende inn en løpende kode. Vi vil evaluere hvert spørsmål i detalj for å se hvor mye du har lært på dette kurset.

Før den detaljerte evalueringen av eksamenene dine, vil vi foreta en plagieringskontroll av all koden du har oppgitt. Basert på resultatene av denne sjekken, kan det hende vi ber deg om en forklaring, og dette kan påvirke resultatene av eksamenen din. Å legge til kommentarer til koden din hjelper oss å forstå koden din bedre, derfor må du legge til kommentarlinjer i koden.

Hvis du oppdager at informasjon mangler i en oppgave, forklare eventuelle forutsetninger og forutsetninger du synes det er nødvendig å gjøre, legg dem til som kommentarlinjer i koden din.

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 eren økende vanskelighetsgrad utover i deloppgavene.

Oppgaver er merket med prosenter som indikerer hvor mye de teller til sluttkarakteren.. 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.

Eksamen - Step by Step

Dette dokumentet vil lede deg trinn for trinn igjennom hva du skal gjøre nå på denne eksamen.

0. Les nøye igjennom **general introduction (**introduksjons) -sidene.

1.**Last ned** medfølgende .zip-fil. I .zip-filen finner du mapper med .cpp-filer og datafiler.

2. Les **forklaringen** på problemet gitt i begynnelsen av hver seksjon.

3. I .cpp-filene finner du både **grunnleggende kode** og **oppgavene** du må fullføre.

4. Du kan bruke VS Code (eller et hvilket som helst annet utviklingsmiljø du foretrekker) til å åpne og jobbe med den oppgitte koden.

5. Du kan bruke boken eller andre online / offline ressurser, men du kan **IKKE** samarbeide med andre på noen måte, eller direkte kopiere og lime inn online kode som om det er din egen.

6. Etter å ha fullført hver enkelt kodeoppgave, bør du sende inn svaret ditt gjennom Inspera.

- **Kopier og lim inn** svarene dine i de oppgitte svarfeltene i Inspera
  OG (til slutt)
- **Last opp** all den komplette koden som en .zip-fil. Ikke endre den opprinnelige mappestrukturen.
- Det er mulig å oppdatere både enkelt-svarene og filopplastningen flere ganger, i tilfelle du retter på noe etter første innlevering.

7. Send inn koden din selv om den ikke kan kompileres og / eller ikke fungerer riktig. Fungerende kode er **IKKE** et krav for at du skal stå, men det er en fordel.

8. Det er en avsluttende sjekkside for deg for å forsikre deg om at du har sendt inn alt. Husk å fylle den ut før tiden går ut.

Nedlasting av start-fil
Vi gir dere en .zip-fil med noen forhånds-programmerte deler av programmene og en datafil som du trenger å bruke i programmet ditt.

Last ned og lagre .zip-filen (lenken er nedenfor).

Husk å lagre den på et sted på datamaskinen du husker og kan finne igjen.

Pakk ut (unzip) filen. Du finner de forskjellige oppgavene i forskjellige mapper, inkludert oppgaveforklaringer som kommentarer. Du kan også finne disse forklaringene i Inspera, som forskjellige del-spørsmål.

Begynn å jobbe med oppgavene i VS Code (eller et hvilket som helst annet utviklingsmiljø du foretrekker). Vi forventer at du vet hvordan du sammenstiller og kjører kode, slik det ble forklart i oppgave 0 på begynnelsen av semesteret.

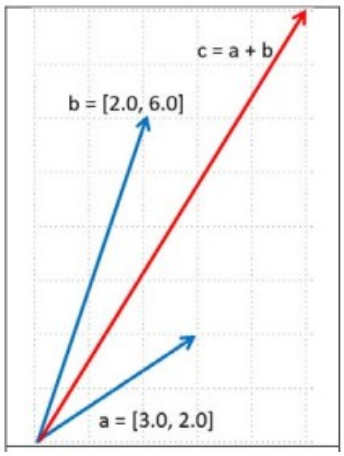
Basefilene vi leverer kompilerer ikke til kjørende programmer, så du må selv skrive resten av koden for alle oppgavene og prøve å få hver av programmene til å kjøre som egne prosjekter. Det er ikke et krav at all den innleverte koden kan kjøres, men det er en fordel.

Etter at du er ferdig med kodingen av hvert del-spørsmål, må du **kopiere og lime inn svarene** dine som egne del-svar på de følgende sidene i Inspera, og på den siste spørsmålsiden **laste opp alle kodefilene dine**, etter at de på nytt er pakket sammen til en lignende .zip-fil som den du begynte med. **Ikke endre noe på den opprinnelige mappestrukturen før du zipper filen og laster den opp til Inspera igjen**.

Last ned .zip-filen på Inspera.

1: Vector 2D Problem (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

1a) Deklarer klassen Vector2d med to medlemsvariabler x og y av type double. Medlemsvariablene skal være public. Deklarer og implementer konstruktøren inline.

1b) 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.

1c) 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).

1d) 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.

1e) 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)
```

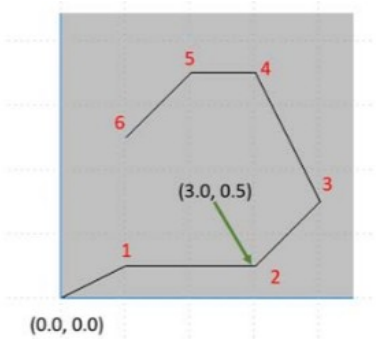
1f) Skriv en funksjon:

```
vectorSum(const vector<Vector2d>& vectors)
```

om returnerer et Vector2d objekt som er summen av alle vektorene i vectors .

1g) 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

1h) Vi ønsker å erstatte de elementer i vektoren som ville føre til en posisjon utenfor sporingsområdet med en null-vektor, Figur 2. 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.

2: Mini-DAK Problem (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.

2a) 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.

2b) 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.

2c) 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.

2d) Det vil også være behov for å oversette fra et Color-objekt til en tekststreng. Implementer funksjonen

```
string color_to_string(Color color)
```

om 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.

2e) Vi deklarerer en felles abstrakt klasse DAKShape for alle tegneelementer i Mini-DAK. Deklarasjonen er i .cpp filen du har lasted ned.
Basert på DAKShape:
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.

2f) Implementer medlemsfunksjonen `DAKRectangle::to_string()`

om 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.

2g) 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.

2h) Implementer klassen DAKCircle som har akkurat samme oppbygging og funksjonalitet som DAKRectangle.

2i) Vi har en klasse MiniDAK spesialisering av Window og er deklarert i .cpp filen du har nedlasetet.

Når brukeren er ferdig med å skrive inn en kommando i tekstboksen 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 2n)

2j) 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 ().

2k) 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. Husk å oppdatere koordinatene til musepekeren.

2l) 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.

2m) 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.

2n) 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.

3: Åpen oppgave - Plot problem (30%)

I denne oppgaven ber vi deg om å designe og implementere et lite program. Programmet vil få en serie koordinater (heltall) fra brukeren (enten gjennom tastatur eller fil) og tegne linjer mellom punktene som disse koordinatene representerer.

Programmet skal:

- Vise en konsollmeny som lar brukeren legge inn enten
    - et filnavn å lese x- og y-verdier fra
    - en sekvens med heltall som representerer vekslende x- og y-koordinater
- Formatet til fil- eller tastatur-innputten er en sekvens med komma-separerte heltall
    - Whitespace er uten betydning.
    - Whitespace-tegn inkluderer ' ' (mellomrom). '\t' (tab), '\n' (ny linje), '\r' (retur)
- Konvertere de innleste heltallene til Points
- Tegne en linje fra punkt til punkt som disse koordinatene representerer.
- Håndtere unntak som manglende fil, feil inndata, odde antall inntastede tall, ...

I denne oppgaven står du fritt til å designe programmet ditt slik du vil og du kan bruke alt av det C++ tilbyr. Det er opp til deg hvor mye du vil inkludere i programmet ditt.

Eksempel:

Input: 112, 108, 112, 208, 121, 108, 312, 208, 312, 108

Heltallene i denne innputten representerer punkter i et 2d-koordinatområde, slik som (x1, y1), (x2, y2), (x3, y3), (x4, y4) hvor x1 = 112, y1 = 108, x2 = 112 , y2 = 208 osv.

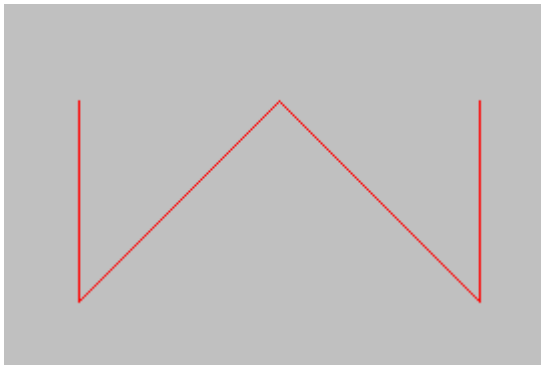
Som **output** skal brukeren se linjer tegnet mellom følgende punkter:

(x1, y1) og (x2, y2)

(x2, y2) og (x3, y3)

(x3, y3) og (x4, y4)

Output:



3a) Vær så snill å copy-paste svaret ditt på oppgaven her. Husk å last opp ny .zip-versjon av alle kodefilene i alle folderne du har (lastet ned).

File Upload

Last opp filen her. Maks én fil (.zip) som inneholder alle folderne. [Velg]

Final Check (Sluttkontroll)

Før du avslutter eksamen, sjekk en gang til at du har gjort alt dette riktig:

Jeg har gitt samtykker.	Ja
Jeg har lastet ned .zip-filen og jobbet med den for å fullføre de oppgitte oppgavene.	Ja
Jeg har kopiert og limt inn svarene mine i de riktige boksene i Inspera.	Ja
Jeg har lastet opp min endelige kode som en .zip-fil, og beholdt den opprinnelige mappestrukturen.	Ja

Feedback (Optional)

Dette er første gang vi kjører en digital hjemme-eksamen i TDT4102. Din tilbakemelding kan hjelpe oss med å forbedre kvaliteten på fremtidige eksamener.

Hvis du ikke har tid til å gi tilbakemelding nå, kan du gjøre det senere gjennom dette eksterne skjemaet:

https://forms.gle/4fUnWsD19WPe6nLw6

Appendix

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);
	...
};
```