

i Forside

✓ Regler og samtykke

Dette er en **individuell** eksamen. Du har **ikke** lov til å kommunisere i noen form eller samarbeide med noen andre under eksamen.

Før du kan fortsette til selve øvingen må du forstå og **SAMTYKKE** til følgende:

Under insperaøving 2:

Jeg skal IKKE motta hjelp fra andre.

☐ Aksepter

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

☐ Aksepter

Jeg skal IKKE kopiere kode fra noen eksisterende online/offline kilder. (Du kan se, og deretter skrive din EGEN versjon av koden).

☐ Aksepter

Jeg er klar over at eksamenen kan bli underkjent 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 utestengt fra universitet og få annullert eksamensresultat.

☐ Aksepter

i 1 - Informasjon

Du kommer nå til **del 1** av insperaøvingen som består av flervalgsoppgaver. Denne delen inneholder **12 oppgaver** som totalt kan gi **60 poeng** og teller ca. **20 %** av eksamen.

Informasjon om poenggiving: For hver oppgave på denne delen får du en maksimal poengsum på 5 poeng. Det er minst et riktig svaralternativ per oppgave, men antall riktige svaralternativ per oppgave kan være mer enn en. For hvert riktige svaralternativ du velger får du $+\frac{\text{maksimal poengsum}}{\text{antall riktige svaralternativ}}$ poeng. For hvert uriktige svaralternativ du velger får du $-\frac{\text{maksimal poengsum}}{\text{antall riktige svaralternativ}}$ poeng. Uansett er det slik at det ikke er mulig å få en negativ poengsum for en oppgave; du får minimum 0 poeng per oppgave.

1 1.1 - Variabeldeklarasjoner og datatyper

Hva må minimum presiseres ved variabeldeklarasjon?

Velg ett eller flere alternativer

- ☐ Navn
- ☐ Verdi
- ☐ Datatype
- ☐ Navnerom (Namespace)

Maks poeng: 2

2 1.2 - Funksjoner og operatorer

Se på koden under.

```
1  constexpr int increment = 1;
2
3  constexpr int incrementValue(int value) { return value + increment; };
4
5  constexpr void func() {
6      int v1 = 10;
7      int v2 = incrementValue(v1);
8      constexpr int v3 = incrementValue(1);
9      constexpr int v4 = v3 + 3;
10 }
```

Hvilke (en eller flere) av følgende utsagn er korrekt?

Velg ett eller flere alternativer

- ☐ Variabelen v4 vil evalueres ved kjøretid.
- ☐ Kallet til funksjonen incrementValue() i linje 8 vil evalueres ved kompileringstid.
- ☐ Kallet til funksjonen incrementValue() i linje 7 vil evalueres ved kompileringstid.
- ☐ Funksjonen incrementValue() har ingen bivirkninger (side effects).

Maks poeng: 2

3 1.3 - Inn/ut datahåndtering

Hvilke (en eller flere) av de følgende deklarasjonene og definisjonene av operator << i klassen Date vil tillate oss å printe feltene i klassen?

Velg ett eller flere alternativer

- ☐ `#include <iostream>`
- ```
class Date {
public:
 int day = 0;
 int month = 0;
 int year = 0;
 Date() {};
 static std::ostream& operator<< (std::ostream& stream, Date date);
};

std::ostream& operator<< (std::ostream& stream, Date date) {
 stream << date.day << "/" << date.month << "/" << date.year;
 return stream
}
```
- ☐ `#include <iostream>`
- ```
class Date {
    int day = 0;
    int month = 0;
    int year = 0;
    Date() {};
    std::ostream& operator<< (std::ostream& stream, Date date);
};

std::ostream& operator<< (std::ostream& stream, Date date) {
    stream << date.day << "/" << date.month << "/" << date.year;
    return stream
}
```
- ☐ `#include <iostream>`
- ```
class Date {
 int day = 0;
 int month = 0;
 int year = 0;
public:
 Date() {};
 std::ostream& operator<< (std::ostream& stream, Date date);
};

std::ostream& operator<< (std::ostream& stream, Date date) {
 stream << date.day << "/" << date.month << "/" << date.year;
 return stream
}
```
- ☐ `#include <iostream>`
- ```
class Date {
public:
    int day = 0;
    int month = 0;
    int year = 0;
    Date() {};
    friend std::ostream& operator<< (std::ostream& stream, Date date);
};

std::ostream& operator<< (std::ostream& stream, Date date) {
    stream << date.day << "/" << date.month << "/" << date.year;
    return stream;
}
```

Maks poeng: 1

4 1.4 - Egendefinerte typer

Hvilke (en eller flere) alternativ er en egendefinert type?

Velg ett eller flere alternativer

- ☐ En struct
- ☐ En enum-klasse
- ☐ En abstrakt klasse
- ☐ En enum

Maks poeng: 4

5 1.5 - Egendefinerte typer

Se på koden under.

```

1  #include <iostream>
2
3  enum class Colour {RED, YELLOW, BLUE, GREEN};
4  enum class DressColour {RED, PINK, BLUE, GREEN};
5
6  void func(DressColour colour) {
7      if (...) {
8          std::cout << static_cast<int>(colour) << std::endl;
9      }
10 }
```

Hvilke (en eller flere) av sammenligningene vil IKKE gi feil?

Velg ett eller flere alternativer

- ☐

```
if (colour == Colour::BLUE) {
    std::cout << static_cast<int>(colour) << std::endl;
}
```
- ☐

```
if (colour == DressColour::PINK) {
    std::cout << static_cast<int>(colour) << std::endl;
}
```
- ☐

```
if (colour == BLUE) {
    std::cout << static_cast<int>(colour) << std::endl;
}
```
- ☐

```
if (colour == 1) {
    std::cout << static_cast<int>(colour) << std::endl;
}
```

Maks poeng: 1

6 1.6 - Minne

Koden under er et eksempel på ...

```
int* values = new int[3] {1, 2, 3};  
for (int i = 0; i < 3; i++) {  
    delete[] values;  
}
```

Velg ett eller flere alternativer

- ☐ Dobbel tom (double free)
- ☐ Minnelekkasje
- ☐ Dangling referanse (dangling reference)
- ☐ Dereferering av nullpeker

Maks poeng: 1

7 1.7 - Minne

Se på koden under.

```
#include <memory>

std::unique_ptr<Book> createBook() {
    std::unique_ptr<Book> book = new Book();
    return book;
}

int func() {
    {
        auto cppBook = createBook();
    }
    return 0;
}
```

Hvilke (en eller flere) av følgende utsagn er korrekt?

Velg ett eller flere alternativer

- ☐ book er allokert i heap, mens cppBook er allokert på stack.
- ☐ Når func() returnerer er minnet som ble allokert til cppBook deallokert.
- ☐ Både book og cppBook er allokert på stack.
- ☐ Koden inneholder en minnelekkasje.

Maks poeng: 1

8 1.8 - Minne

Se på koden under.

```
Book* createBook() {  
    Book* book = new Book();  
    return book;  
}  
  
int func() {  
    {  
        auto cppBook = createBook();  
    }  
    return 0;  
}
```

Hvilke (en eller flere) av følgende utsagn er korrekt?

Velg ett eller flere alternativer

- ☐ book er allokert i heap, mens cppBook er allokert på stack.
- ☐ Når func() returnerer er minnet som ble allokert til cppBook deallokert.
- ☐ Koden inneholder en minnelekkasje.
- ☐ Både book og cppBook er allokert på stack.

Maks poeng: 1

9 1.9 - Minne

Hvilke (en eller flere) av følgende utsagn er korrekt?

Velg ett eller flere alternativer

- ☐ Man kan ikke endre på adressen som er lagret i en peker.
- ☐ En peker kan ikke referere til en verdi som ikke eksisterer.
- ☐ Man kan endre på verdien en peker refererer til, men ikke uten å dereferere pekeren først.
- ☐ &-operatoren kan brukes til å finne adressen til en variabel.

Maks poeng: 1

10 1.10 - Klasser og arv

Se på klassedeklarasjonen under.

```
class Class {
    OtherClass* otherClass;
public:
    Class() {
        otherClass = new OtherClass();
    }
    ~Class() {
        delete otherClass;
    }
};
```

Hvilke (en eller flere) av følgende utsagn er korrekt?

Velg ett eller flere alternativer

- ☐ Å lage en instans av klassen vil allokere minne.
- ☐ Klassen har ikke en kopikonstruktør.
- ☐ Klassen har ikke en standard (default) konstruktør.
- ☐ Klassen mangler en destruktør.

Maks poeng: 1

11 1.11 - Klasser og arv

Hvilke (en eller flere) av følgende utsagn er korrekt?

Velg ett eller flere alternativer

- ☐ Et predikat (boolsk objekt) kan ikke deklarerer som en friend.
- ☐ Terminalen kan ikke skrive ut egendefinerte datatyper som er deklart med friend.
- ☐ friend-deklarasjoner kan brukes til å overlaste output-operatoren.
- ☐ friend-deklarasjoner gir tilgang til private klassemedlemmer.

Maks poeng: 2

12 1.12 - Klasser og arv

Se på deklarasjonene til klassene Parent og Child under.

```

1  class Parent {
2  private:
3      string firstName;
4  protected:
5      string surname;
6      string address;
7  public:
8      string getFirstName() { return firstName; }
9      string getSurname() { return surname; }
10 };
11
12 class Child: protected Parent {
13 private:
14     string secret;
15 };

```

Hva er tilgangsnivået til medlemsfunksjonene og medlemsvariablene i Child-klassen? (Kryss av i riktig boks for hver variabel eller funksjon.)

Finn de som passer sammen:

	protected	private	Ikke i skop	public
firstname	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
getFirstName()	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
getSurname()	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
address	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
surname	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Maks poeng: 5

i 2 - Informasjon

Del 2 av eksamen er kortsvarsoppgaver. Du skal svare på oppgavene i konteksten av programmeringsspråket C++ ut i fra det du har lært i emnet gjennom forelesningene, øvingene og pensumboken *Programming: Principles and Practice Using C++*, 2nd edition av Bjarne Stroustrup. Denne delen av eksamen inneholder **12 oppgaver** som til sammen gir en maksimal poengsum på 60 poeng og teller ca. **20 %** av eksamen.

Informasjon om poenggiving: Hver oppgave gir maksimal poengsum på **5 eller 10 poeng** avhengig av vanskelighetsgrad og arbeidsmengde. Du får poeng etter hvor presist svaret ditt er. Det vil si at det er mer uttelling for et kort og presist svar enn et langt og upresist svar.



13 2.1 - Grunnleggende kunnskap




Forklar kort hva en kompilator er.




Skriv ditt svar her



Format


B *I* U \times_2 \times^2 \mathcal{I}_x

Σ 

Words: 0

Maks poeng: 5

14 2.2 - Grunnleggende kunnskap

Forklar kort hvorfor teknikken fremoverdeklarasjon brukes på linje 1 i koden under.

```
1  class A;  
2  
3  class B {  
4      int b1;  
5      string b2;  
6      A * ba;  
7  };  
8  
9  class A {  
10     int a1;  
11     string a2;  
12     B * ab;  
13  };
```

Skriv ditt svar her

Format ▾

B

I

U

x_2

x^2

I_x

Σ

Words: 0

Maks poeng: 5

¹⁵ 2.3 - Variabeldeklarasjoner og datatyper

Forklar kort hvorfor man må sjekke om resultatet ligger innenfor et intervall $[a, c]$ når vi skal verifisere at en funksjon som utfører flyttalloperasjoner gir ønsket resultat b i stedet for å sjekke at resultatet er lik b .

Skriv ditt svar her

Maks poeng: 5

16 2.4 - Variabeldeklarasjoner og datatyper

Forklar kort hvorfor det er god praksis å deklarere en variabel konstant (const) når variabelen ikke skal endres.

Skriv ditt svar her

Format ▾ | **B** *I* U \times_2 \times^2 | $\frac{\square}{\square}$ | | | | |

Σ |

Words: 0

Maks poeng: 5

17 2.5 - Funksjoner og operasjoner

Forklar kort hvorfor man bruker *funksjoner*.

Skriv ditt svar her

Format ▾ | **B** *I* U \times_2 \times^2 | $\frac{\square}{\square}$ | | | | Ω | |

Σ |

Words: 0

Maks poeng: 5

18 2.6 - Kontrollstrukturer

Konverter følgende for-løkke til en while-løkke.

```
1  #include <iostream>
2
3  int b = 100;
4  for(int a = 1; a <= b; a *= 3){
5      std::cout << a << " < " << b << std::endl;
6      b++;
7  }
```

Skriv ditt svar her

Format ▾ | **B** *I* U \times_2 \times^2 | \mathcal{I}_x | | | | Ω | |

Σ |

Words: 0

Maks poeng: 5










19 2.7 - Templates


Forklar kort bruksområdene til templates.

Skriv ditt svar her

Format

B *I* U \times_2 \times^2 \mathcal{I}_x

Σ 

Words: 0

Maks poeng: 5

20 2.8 - Minne

Forklar kort *de fire ulike risikoene* man har med pekere.

Skriv ditt svar her

Format ▾ | **B** *I* U \times_2 \times^2 | $\frac{\square}{\square}$ | | | | Ω | |

Σ |

Words: 0

Maks poeng: 5

21 2.9 - Minne

Forklar kort forskjellen mellom grunn kopi (shallow copy) og dyp kopi (deep copy).

Skriv ditt svar her

Format

B


I


U


x_2


x^2


I_x

















Ω





Σ



Words: 0

Maks poeng: 5

22 2.10 - Klasser og arv

Forklar kort de to ulike måtene man kan lage en *abstrakt* klasse.

Skriv ditt svar her

Format

B

I

U

x_2

x^2

I_x

Σ

Words: 0

Maks poeng: 5

23 2.11 - Klasser og arv

Forklar kort hvorfor man bør markere redefinisjonen av virtuelle superklasse-funksjoner med nøkkelordet override i subclassen.

Skriv ditt svar her

Format

B

I

U

x_2

x^2

I_x

Σ

Words: 0

Maks poeng: 5

24 2.12 - Klasser og arv

Forklar kort hvorfor man må bruke funksjonen `emplace_back()` i stedet for funksjonen `push_back()` på en `unique_ptr` for å legge til en ny verdi.

Skriv ditt svar her

Format ▾ | **B** *I* U \times_2 \times^2 | $\frac{\square}{\square}$ | | | | |

Σ |

Words: 0

Maks poeng: 5

i 3 - Informasjon

Del 3 av eksamen er programmeringsoppgaver. Denne delen inneholder **11 oppgaver** som til sammen gir en maksimal poengsum på ca. **180 poeng** og teller ca. **60 %** av eksamen. Hver deloppgave kan gi en poengsum fra **5 - 25 poeng** avhengig av vanskelighetsgrad og arbeidsmengde.

De utdelte filene inneholder kompilerbare (og kjørbare) .cpp- og .h-filer med kode og en full beskrivelse av oppgavene i del 3 som en PDF. Etter å ha lastet ned koden står du fritt til å bruke et utviklingsmiljø (VS Code) for å jobbe med oppgavene. Vi anbefaler på det sterkeste å kontrollere at koden kompilerer og kjører før du setter i gang med oppgavene.

Sjekkliste ved tekniske problemer

Hvis prosjektet du oppretter med den utdelte koden ikke kompilerer (før du har lagt til egne endringer) bør du rekke opp hånden og be om hjelp. Mens du venter kan du prøve følgende:

1. Sjekk at prosjektmappen er lagret i mappen C:\temp.
2. Sjekk at navnet på mappen **IKKE** inneholder norske bokstaver (Æ, Ø, Å) eller mellomrom. Dette kan skape problemer når du prøver å kjøre koden i VS Code.
3. Dobbelsjekke fremgangsmåte
 - Last ned .zip-filen fra Inspera og pakk den ut (unzip). Lagre filen i C:\temp som IO2TDT4102_kandidatnummer. Du skal altså skrive inn ditt eget kandidatnummer bak understreken.
 - Åpne mappen i VS Code. Bruk deretter følgende TDT4102-kommandoer for å opprette et fungerende kodeprosjekt:
 1. **Ctrl + Shift + P → TDT4102 : Force refresh of the course content**
 2. **Ctrl + Shift + P → TDT4012 : Create project from TDT4102 template → Configuration only**
4. Sørg for at du er inne i riktig fil i VS Code.
5. Prøv å kjøre koden igjen (**Ctrl + F5**)
6. Hvis det fortsatt ikke fungerer, lukk VS Code vinduet og åpne det igjen. Gjenta deretter steg 3-5.

i Oppgave/utdelt kode del 3

Oppgavetekst for del 3 av eksamen.

Oppgavetekst: [TDT4102_IO2_ebm](#)

Kodefiler: [tdt4102_IO2_handout](#)

25 Nedlasting/opplasting av del 3


LAST OPP

Last opp all den komplette koden som en .zip-fil. Alt i én zip-fil. Merk at filen må være av typen .zip (andre format som 7z, rar etc. godtas ikke). Ikke endre den opprinnelige mappestrukturen.



Last opp filen her. Maks én fil.

Alle filtyper er tillatt. Maksimal filstørrelse er **50 GB**.

 Velg fil for opplasting

Maks poeng: 10

Document 1

Attached



Inspiraøving 2 i TDT4102 - Prosedyre- og objektorientert programmering

Dato: Mandag 22.04 og Tirsdag 23.04

Eksamenstid (fra-til): 09:00-13:00

Hjelpemiddelkode/Tillatte hjelpemiddel: C / Bestemt, enkel kalkulator og spesifiserte trykte og håndskrevne hjelpemidler: [CPP reference](#), [dokumentasjon for AnimationWindow](#) og stemplet, håndskrevet A5-ark.

Faglig kontakt under IØ2: Bart van der Blokland (faglærer), Dragana Laketic (faglærer)

Tlf: 402 90 438 (Bart), 482 86 663 (Dragana)

ANNEN INFORMASJON

Denne øvingen har likt oppsett som årets eksamen, og vil gi dere et inntrykk av hvordan dere vil løse eksamen. Prøv å svare på alle spørsmål etter beste evne, selv oppgaver som ikke er fullstendig besvart kan gi uttelling i en eksamenssituasjon.

Ikke ha Inspira åpen i flere faner, eller vær pålogget på flere enheter, samtidig, da dette kan medføre feil med lagring/levering av besvarelsen din.

Sjekk deg overblikk over oppgavesettet før du begynner å løse oppgavene. Disponer tiden godt! I noen oppgaver i del 3 vil det være naturlig å bruke funksjoner fra tidligere oppgaver, disse kan brukes som om de fungerer selv om du ikke fikk til den tidligere oppgaven.

Les oppgavene nøye, gjør dine egne antagelser og presiser i besvarelsen hvilke forutsetninger du har lagt til grunn i tolkning/avgrensing av oppgaven. Husk å legge til kommentarlinjer i koden din der det kan hjelpe sensor å forstå koden din bedre. Faglig kontaktperson skal kun kontaktes dersom det er direkte feil eller mangler i oppgavesettet. Henvend deg til en eksamensvakt hvis du ønsker å kontakte faglærer. Noter gjerne spørsmålet ditt på forhånd. På flervalgsspørsmål får du positive poeng for riktige svar og negative poeng for feil svar. Summen vil aldri være mindre enn null poeng for et spørsmål, selv om du svarer mer feil enn riktig på det spørsmålet.

Juks/plagiat: Eksamen skal være et individuelt, selvstendig arbeid. Det er tillatt å bruke hjelpemidler, men vær obs på at du må følge eventuelle anvisninger om kildehenvisninger under. Under eksamen er det ikke tillatt å kommunisere med andre personer om oppgaven eller å distribuere utkast til svar. Slik kommunikasjon er å anse som juks, og det gjelder både den som hjelper, og den som blir hjulpet. Alle besvarelser blir kontrollert for plagiat.

Kildehenvisninger: Selv om enkelte hjelpemidler er tillatte, er det ikke tillatt å kopiere andres kode og levere den som din egen. Du kan se på andre åpent tilgjengelige ressurser, og deretter skrive din egen versjon av det du så, i henhold til copyright-forskrifter.

Varslinger: Hvis det oppstår behov for å gi beskjeder til kandidatene underveis i eksamen (for eksempel ved feil i oppgavesettet), vil dette bli gjort via varslinger i Inspira. Et varsel vil dukke opp som en dialogboks på skjermen i Inspira. Du kan finne igjen varselet ved å klikke på bjella øverst i høyre hjørne på skjermen.

Vekting av oppgavene: Del 1 teller ca. 20% av totalen, del 2 teller ca. 20% av totalen, og

del 3 teller ca. 60% av totalen på denne øvingen. Du vil ikke få godkjent øvingen om du svarer blankt på en av de tre delene. Dette innebærer at du må få til å laste opp en zip-fil med kode i del 3.

Slik svarer du på oppgavene: Alle oppgaver (unntatt del 3 som er av typen filopplasting), skal besvares direkte i Inspira. I Inspira lagres svarene dine automatisk hvert 15. sekund. NB! Klipp og lim fra andre programmer frarådes også da dette kan medføre at formatering og elementer (bilder, tabeller etc.) vil kunne gå tapt.

Filopplasting: Når du jobber i et annet program (VS Code) fordi deler av besvarelsen din skal leveres som filvedlegg – husk å lagre besvarelsen din med jevne mellomrom. Merk at alle filer må være lastet opp i besvarelsen før eksamenstiden går ut. Det framgår av filopplastingsoppgaven hvilket filformat som er tillatt (zip). Det er lagt til 15 minutter til ordinær tid for opplasting av filer. Tilleggstiden er forbeholdt innlevering og inngår i gjenstående tid som vises øverst til venstre på skjermen. NB! Det er ditt eget ansvar å påse at du laster opp riktige og fungerende filer. Kontroller zip-filen du har lastet opp ved å klikke Last ned når du står i filopplastingsoppgaven. Alle filer kan fjernes og byttes ut så lenge prøven er åpen.

INSPERAØVING - TRINN FOR TRINN

Denne listen vil lede deg trinn for trinn igjennom hva du skal gjøre på denne øvingene.

1. Les nøye igjennom introduksjonssiden.
2. Start VS Code. En god måte å gjøre dette på er å trykke på søk-menyen nede i venstre hjørne og skrive "code".
3. Gå til mappen C:\temp.
 - Last ned .zip-filen fra Inspira og pakk den ut (unzip). Lagre filen i C:\temp som IO2TDT4102_kandidatnummer. Du skal altså skrive ditt eget kandidatnummeretter understreken.
4. Gå inn i VS Code og åpne mappen du nettopp opprettet. Dette gjøres ved å trykke på "File" øverst til venstre, og deretter trykke "Open Folder". Velg mappen du lagde.
5. Opprette et gyldig VS Code-prosjekt basert på utdelte filer i del 3 med følgende TDT4102-kommandoer:
 - (a) `Ctrl + Shift + P` → TDT4102: Force refresh of the course content
 - (b) `Ctrl + Shift + P` → TDT4102: Create project from TDT4102 template → Configuration only.
6. Når utvidelsen gjenkjenner at prosjektet er korrekt oppsatt vil det stå "TDT4102 Project ✓" nederst i venstre hjørne av VS Code.
7. Kjør programmet ved å trykke `Ctrl + F5` eller `Fn + F5`, eller ved å gå inn i menyen **Run and Debug** og velge **Build and Run Debug**.
8. Les forklaringen på problemet gitt i begynnelsen av hvert seksjon.
9. Etter å ha fullført hver enkelt kode-oppgave må du huske å lagre (`Ctrl + S`).
10. Send inn koden din selv om den ikke kan kompileres og/eller ikke fungerer riktig. Fungerende kode er **IKKE** kravet for at du skal stå, men det er en fordel.

- (a) Også last opp alle øvrige oppgavefiler som du skal zippe ved bruk av TDT4102-kommandoen **Ctrl + Shift + P** → **TDT4102: Prepare a zip file for delivery**. Ikke endre på de forhåndsinnstilte mappe/fil-navnene for å zippe filen og last deretter opp til Inspira igjen. For å få beskjed på denne eksamen er det **HELT AVGJØRENDE AT DU LASTER OPP ZIP-FILEN**. Etter prøveslutt har du 30 minutter til rådighet til dette. Vi anbefaler likevel at du prøver å laste opp filen en gang underveis i eksamenstiden også, for å sjekke at du klarer det.
11. Husk at funksjonene du lagrer i en filoppbevaring etter at du har kikket på andre deltagers løsninger og deretter deler dem på en offentlig platform er **ikke** tillatt og vil føre til tiltak for juks.
12. Vi vil føre automatisk testing og plagiatkontroll av all koden du har levert, og vil også føre kontroll av alt øvrig materiell.

Automatisk innlevering: Besvarelsen din leveres automatisk når eksamenstida er ute og prøven stenger, forutsatt at minst en oppgave er besvart (og at zip-fila er riktig lastet opp). Dette skjer selv om du ikke har klikket Lever og gå tilbake til Dashboard på siste side i oppgavesettet. Du kan gjenåpne og redigere besvarelsen din så lenge prøven er åpen. Dersom ingen oppgaver er besvart ved prøveslutt, blir ikke besvarelsen din levert. Dette vil anses som "ikke møtt" til eksamen.

Tilgang til besvarelse: Du finner besvarelsen din i "Arkiv" i Inspira etter at sluttida for hele eksamen er passert.

Document 5

Attached



Del III:

Fuglesimulering (180p)

Del 3 av eksamen er programmeringsoppgaver. Denne delen inneholder **11 oppgaver** som til sammen gir en maksimal poengsum på ca. **180 poeng** og teller ca. **60 %** av eksamen. Hver deloppgave kan gi en poengsum fra **5 - 25 poeng** avhengig av vanskelighetsgrad og arbeidsmengde.

De utdelte filene inneholder kompilerbare (og kjørbare) .cpp- og .h-filer med kode og en full beskrivelse av oppgavene i del 3 som en PDF. Etter å ha lastet ned koden står du fritt til å bruke et utviklingsmiljø (VS Code) for å jobbe med oppgavene. Vi anbefaler på det sterkeste å kontrollere at koden kompilerer og kjører før du setter i gang med oppgavene.

Sjekkliste ved tekniske problem

Hvis prosjektet du oppretter med den utdelte koden ikke kompilerer (før du har gjort egne endringer) bør du rekke opp hånden og be om hjelp. Mens du venter kan du prøve følgende:

1. Sjekk at prosjektmappen er lagret i mappen C:\temp.
2. Sjekk at navnet på mappen **IKKE** inneholder norske bokstaver (*Æ, Ø, Å*) eller mellomrom. Dette kan skape problemer når du prøver å kjøre koden i VS Code.
3. Sørg for at du har:
 - Last ned .zip-filen fra Inspira og pakk den ut (unzip). Lagre filen i C:\temp som **IO2TDT4102_kandidatnummer**. Du skal altså skrive *ditt eget kandidatnummer* etter understreken.
 - Åpne mappen i VS Code. Bruk deretter følgende TDT4102-kommandoar for å opprette et fungerende kodeprosjekt:
 - (a) Ctrl+Shift+P → TDT4102: Force refresh of the course content
 - (b) Ctrl+Shift+P → TDT4102: Create project from TDT4102 template → Configuration only
4. Sørg for at du er inne i riktig fil i VS Code.
5. Kjør følgende TDT4102-kommandoer:
 - (a) Ctrl+Shift+P → TDT4102: Force refresh of the course content
 - (b) Ctrl+Shift+P → TDT4102: Create project from TDT4102 template → Configuration only
6. Prøv å kjør koden igjen (Ctrl+F5 eller Fn+F5 eller F5).
7. Hvis det fortsatt ikke fungerer, lukk VS Code vinduet og åpne det igjen. Gjenta deretter steg 5-6.

Introduksjon

Fuglesimuleringen er en enkel simulering av hvordan fugler flyr i flokker. Koden for fuglesimuleringen er inndelt i tre hoveddeler:

- Et *Application*-objekt som styrer funksjonsbaren på toppen av animasjonsvinduet, inn-/ut-datahåndtering og hovedløkken som tegner hver frame til skjermen og ber et *Simulator*-objekt om å oppdatere fuglenes tilstand.
- Et *Simulator*-objekt som har oversikt over alle fuglene i simuleringen.

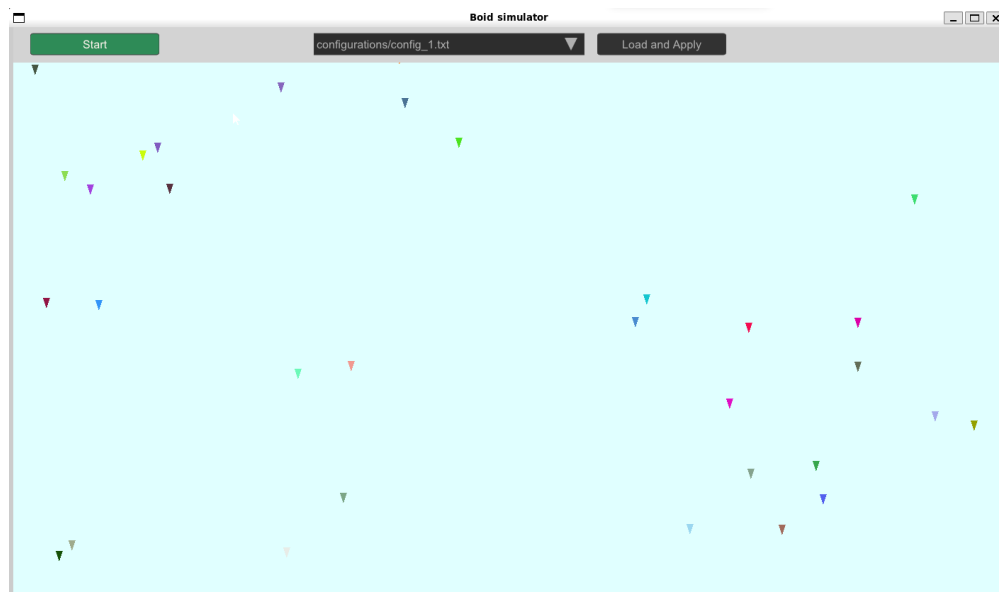
- En abstrakt baseklasse, Bird, som er ansvarlig for å oppdatere en fugls posisjon og hastighet, og å tegne fuglen til skjermen for hver frame.

Den abstrakte baseklassen Bird har to subklasser; Doves og Hawks. Duene følger reglene beskrevet under. Haukene flyr litt tilfeldig over skjermen og ønsker å unngå andre hauker. Du trenger ikke ta stilling til haukene før i siste oppgave.

Reglene som duene skal følge er basert på fire flokkegenskaper:

- **Separasjon (separation):** Fuglen styrer for å unngå kollisjon med andre fugler i flokken.
- **Sammenstilling (alignment):** Fuglen styrer i samme retning som resten av flokken.
- **Samhold (cohesion):** Fuglen styrer mot sentrum av flokken.
- **Unngåelse (avoidance):** Fuglen styrer for å unngå jegerfugler.

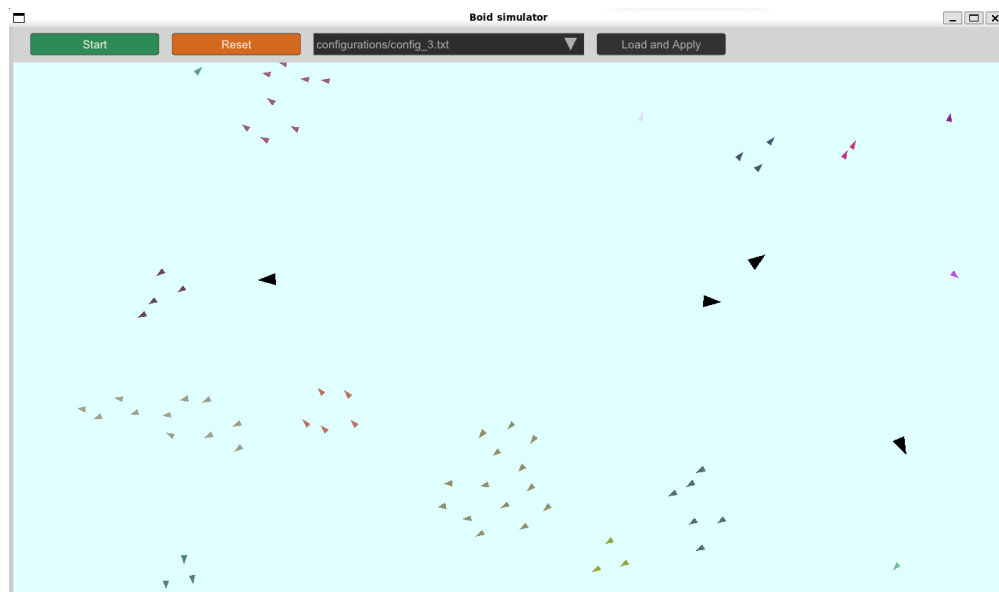
I .zip-filen finner du et kodeskjelett med klart markerte oppgaver.



Figur 1: Skjerm bilde av kjøring av den utdelte koden. I animasjonsvinduet ser man en funksjonsbar bestående av en Start/Stopp-knapp, en nedtrekksmeny for å laste inn ulike konfigurasjoner, og en knapp for å bruke den valgte konfigurasjonen (Load and Apply). I tillegg ser man 32 fugler i vinduet.

Før du begynner på oppgavene bør du sjekke at den utdelte koden kjører uten problemer. Du skal se omtrent det samme vinduet som i Figur 1. Fuglene plasseres tilfeldig i vinduet, så startposisjonen til de åtte fuglene vil variere og er ikke viktig.

Fuglesimuleringen i den utdelte koden inneholder minimalt med funksjonalitet (Figur 1). Når alle oppgavene er gjort vil man derimot kunne se at fuglene flyr i flokker mens de prøver å unngå jegerfuglene som flyr rundt i tilfeldige baner (Figur 2).



Figur 2: Skjerm bilde av den kjørende simuleringen når den er ferdigstilt.

Hvordan besvare oppgavene

Hver oppgave i del 3 har en unik kode for å gjøre det lettere for deg å vite hvor du skal fylle inn svarene dine. Koden er på formatet <T><siffer> (TS), der sifrene er mellom 1 og 11 (T1 - T11). For hver oppgave vil man finne to kommentarer som skal definere henholdsvis begynnelsen og slutten av svaret ditt. Kommentarene er på formatet: // BEGIN: TS og // END: TS.

For eksempel ser oppgave T2 i den utdelte koden slik ut:

```
// Task T2: Update the position of the bird using
// on its current position and velocity.
void Bird::updatePosition()
{
    // BEGIN: T2
    ;
    // END: T2
}
```

Det er veldig viktig at alle svarene dine føres mellom slike par av kommentarer. Dette er for å støtte sensurmekanikken vår. Hvis det allerede er skrevet kode mellom BEGIN- og END-kommentarene til en oppgave i det utdelte kodeskjettet, så kan du, og ofte bør du, erstatte denne koden med din egen implementasjon. All kode som står *utenfor* BEGIN- og END-kommentarene **SKAL** du la stå som den er. I oppgave T6 og T9 er BEGIN- og END-kommentarene plassert utenfor funksjonsdeklarasjonen, noe som åpner for å bruke egne hjelpefunksjoner og globale variabler. Du skal ikke endre navn eller parameter til disse funksjonene.

Merk: Du skal **IKKE** fjerne BEGIN- og END-kommentarene.

Hvis du synes noen av oppgavene er uklare kan du oppgi hvordan du tolker dem og de antagelsene du gjør for å løse oppgaven som kommentarer i koden du leverer.

Tips: Trykker man CTRL+SHIFT+F og søker på BEGIN: får man snarveier til starten av alle oppgavene listet opp i utforskervinduet slik at man enkelt kan hoppe mellom oppgavene. For å komme tilbake til det vanlige utforskervinduet kan man trykke CTRL+SHIFT+E.

Hvordan levere del 3

Når du er ferdig med oppgavene og er klar til å levere skal du laste opp alle .h- og .cpp-filene i hoved-mappen som en .zip-fil i Inspira. Du står fritt til å bruke den innebygde funksjonen i VS Code til å lage .zip-filen. Disse filene inkluderer:

- Application.h
- Application.cpp
- Simulator.h
- Simulator.cpp
- main.cpp

Oppgavene

Oppgavene vil ha følgende struktur:

Første del inneholder motivasjon, bakgrunnsinformasjon og hvordan koden er satt sammen for oppgaven.

Neste del er en tekstboks som inneholder oppgaven og spesifikke krav.

Siste del forklarer resultatet du kan forvente når du har fullført oppgaven.

1. (10 points) T1: Overlast + operatoren

I den utdelte koden brukes en hastighetsvektor for å bestemme fuglens fart og retning, og en posisjon for å bestemme en fugls plassering. Begge er lagret som en struktur av typen `FloatingPoint` som inneholder to flyttall:

```
struct FloatingPoint {  
    double x;  
    double y;  
};
```

Vi ønsker å kunne legge sammen to `FloatingPoint`-strukturer uten å måtte aksessere feltene i strukturerne direkte hver gang. Addisjon av to vektorer utføres på følgende måte:

$$(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$$

Overlast + operatoren for `FloatingPoint` i `Simulator.cpp`.

Når oppgave T1 er ferdig skal det være mulig å legge sammen to strukturer av typen `FloatingPoint`.

2. (5 points) **T2: Få fuglene til å bevege seg**

Nå står alle fuglene helt i ro når man trykker på start. Det er fordi funksjonen `updatePosition` i `Bird` ikke er implementert og fuglene blir tegnet på samme posisjon i hver frame.

Implementer funksjonen `updatePosition` i `Simulator.cpp`.

- Oppdater posisjonen (`position`) til fuglen ved å legge sammen fuglens hastighet (`velocity`) og posisjon (`position`).

Når oppgave T2 er ferdig skal duene fly i rette linjer over skjermen hvis man klikker på Start og stå i ro hvis man deretter klikker på Stop.

3. (10 points) **T3: Legg til tilbakestillingsknapp**

Nå som fuglene flytter på seg mellom start og stopp av simuleringen ønsker vi å kunne tilbakestille simuleringen så man kan kjøre simuleringen på nytt uten å måtte restarte hele programmet. `Simulator` har en klasse `ResetButton` som er definert i `Simulator.h`:

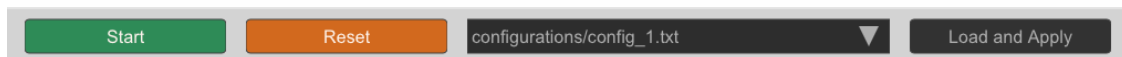
```
class ResetButton : public Button {
public:
    ResetButton ();
    ~ResetButton ();

    static void callback();
    static void setSimulator(Simulator* const _sim);

private:
    static Simulator *sim;
};
```

Legg til en knapp av typen `ResetButton` i `run`-funksjonen til `Application` i `Application.cpp`.

Når oppgave T3 er ferdig skal funksjonsbaren til simuleringen se ut som i Figur 3.



Figur 3: Skjerm bilde av funksjonsbaren etter at oppgave T3 er fullført.

4. (10 points) **T4: Tilbakestill simuleringen**

Tilbakestillingsknappen du la til i forrige oppgave gjør foreløpig ingenting når man klikker på den. Det er fordi `callback`-funksjonen ikke er implementert enda.

Implementer `callback`-funksjonen til tilbakestillingsknappen i `Simulator.cpp`.

- Tilbakestill simuleringen ved å utnytte simulatorobjektet sin `resetSimulation`-funksjon.

Når oppgave T4 er ferdig skal tilbakestillingsknappen sette fuglene i startposisjon når den blir klikket på. Noter deg at knappen kun fungerer når simuleringen er stoppet og at fuglene vil plasseres ulikt etter hver tilbakestilling.

5. (20 points) **T5: Last inn konfigurasjoner** **T5: Last inn konfigurasjonar**

Vi ønsker å kunne teste flere sammensetninger av fugler. For å gjøre dette har vi et sett med konfigurasjoner i form av .txt-filer. Disse filene finner du i mappen som heter `configurations`. Hver fil består av to tall som er separert med et linjeskift. Det første tallet representerer antall duer i simuleringen, mens det andre tallet representerer antall hauker i simuleringen.

Implementer funksjonen `loadAndApplyConfiguration` i `Application.cpp`.

- Utløs et passende unntak dersom filen ikke kan åpnes.
- Les inn tallene fra den gitte konfigurasjonsfilen.
- Sørg for at konfigurasjonen som leses inn blir brukt i simuleringen. Simulator-objekter har en funksjon som heter `applyConfiguration()`.

Når oppgave T5 er ferdig skal du kunne se en endring av antall fugler på skjermen når du trykker på knappen `Load and apply` etter at du har valgt en konfigurasjon fra nedtrekksmenyen.

6. (20 points) **T6: Venn eller fiende**

For at duene skal kunne vite hvordan de skal forflytte seg i neste frame ønsker de en oversikt over fuglene rundt seg. Det er to typer fugler i simuleringen; duer (doves) og hauker (hawks). Duene ser på de andre duene som potensielle venner og hauker som potensielle fiender. I tillegg er duene sitt synsfelt begrenset av de globale variablene `FRIEND_RADIUS` for venner, og `AVOID_RADIUS` for fiender. Alle duer som er utenfor `FRIEND_RADIUS` og alle hauker som er utenfor `AVOID_RADIUS` skal derfor ignoreres. Dette er illustrert i Figur 4. Venner og fiender lagres i hver sin tabell (vector) av delte pekere (`shared_ptr`) til fagleobjekt (`Bird`). Funksjonen du skal implementere skal kalles på hvert `Bird`-objekt for hver frame.

Avstanden mellom to punkt regnes ut på følgende måte:

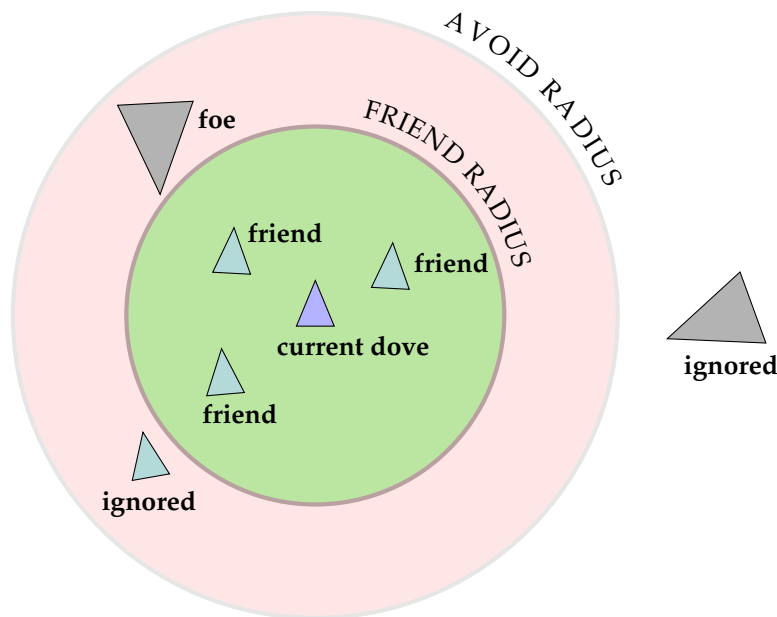
$$\text{distance}((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Implementer funksjonen `makeFriendsAndFoes` i `Simulator.cpp`.

- Fjern elementene som allerede ligger i vektorene `friends` og `foes` fra forrige frame.
- Filtrer de andre fuglene i simuleringen basert på type. Merk at listen funksjonen tar inn inneholder alle fugler. Du må altså hoppe over din egen `Bird`-instans.
- Pass på at venner er innenfor `FRIEND_RADIUS` og at fiender er innenfor `AVOID_RADIUS`.

Hint: Man kan finne informasjon om hvordan man bruker matematiske funksjoner som er implementert i standardbiblioteket ved å gå til C++ reference og siden som heter *Common math functions*.

Merk: BEGIN- og END-kommentarene er utenfor funksjonsdefinisjonen, noe som åpner for muligheten til å lage egne hjelpefunksjoner eller globale variabler for å løse oppgaven.



Figur 4: Illustrasjon av hvilke fugler som er venner og fiender for en spesifikk due (lilla trekant). De andre duene er illustrert med blå trekanter, mens haukene er illustrert med grå trekanter.

Resultatet av oppgave T6 kan man ikke verifisere visuelt.

7. (15 points) T7: Bevegelseslogikken

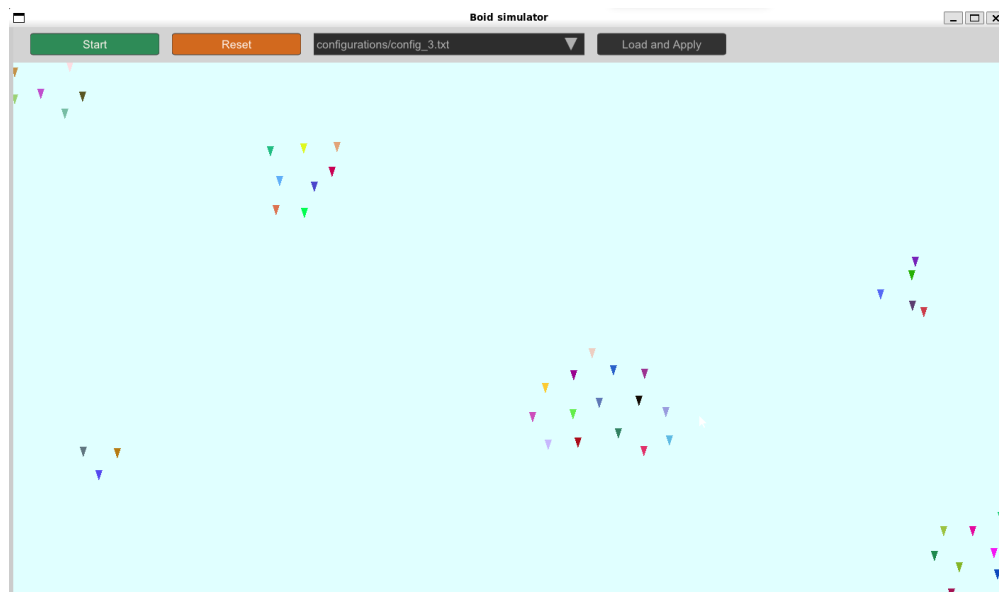
Vi ønsker nå å få duene til å følge reglene vi introduserte i introduksjonen til koden. Dette gjøres ved å oppdatere hastigheten til duene. Reglene er allerede implementert i funksjonene `calculateCoherence`, `calculateAlignment`, `calculateSeparation` og `calculateAvoidance`. Disse funksjonene regner ut hvordan hastigheten til en due må forandre seg for at den skal overholde reglene. Den nye hastigheten til en due kan man dermed finne ved å legge til bidraget fra hver regel til duens nåværende hastighet. Størrelsen til hastighetsvektoren skal ikke overstige `MAX.SPEED`.

Implementer funksjonen `updateVelocity` i `Simulator.cpp`.

- Regn ut den nye hastigheten til fuglen basert på separasjon, sammenstilling, samhold, og unngåelse.
- Dersom den utregnede hastigheten overstiger den maksimale hastigheten bestemt av `MAX.SPEED`, må den skaleres ned på en valgfri måte til å være under `MAX.SPEED`.

Hint: Koden inneholder en hjelpefunksjon `magnitude` som kan være nyttig til å regne ut størrelsen til en hastighetsvektor.

Når oppgave T7 er ferdig skal simuleringen se ut som i Figur 5 etter at den har kjørt litt.



Figur 5: Skjerm bilde av simuleringen etter at oppgave T7 er implementert.

8. (20 points) **T8: Se hvor du flyr**

Frem til nå har fuglene vært orientert med spissen nedover i animasjonsvinduet uansett hvilken retning de har flydd. Vi ønsker nå å tegne trekantene slik at de reflekterer fuglenes retning. Likning (1) gir retningen en fugl flyr gitt x - og y -komponentene fra fuglens hastighetsvektor.

$$\theta(x, y) = \begin{cases} \arctan\left(\frac{y}{x}\right), & \text{if } x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi, & \text{if } x < 0 \\ \frac{\pi}{2} \cdot \text{sign}(y), & \text{if } x = 0 \end{cases} \quad (1)$$

Oppdater funksjonen `draw` i `Simulator.cpp` slik at fuglene snur seg i den retningen de flyr.

- Finn retningen duene flyr (θ) ved å bruke Likning (1). Funksjonen `sign(y)` gir fortegnet til y . Du skal selv implementere funksjonaliteten til `sign()`.
- Finn frontpunktet til trekanten ved å legge til

$$\cos(\theta) \cdot \text{size}$$

til fuglens x -posisjon, og

$$\sin(\theta) \cdot \text{size}$$

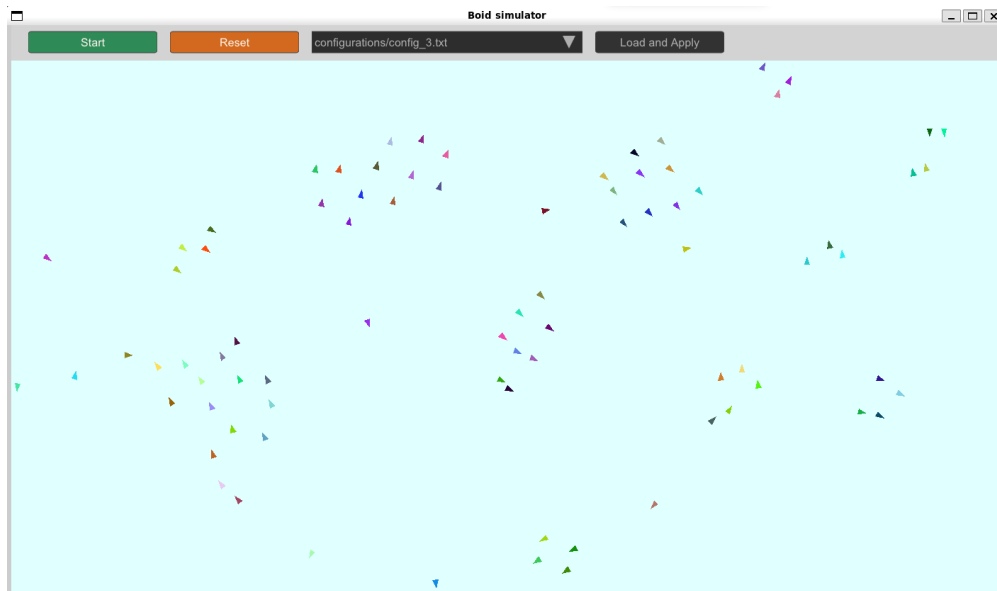
til fuglens y -posisjon. Husk at `size` er et attributt i `Bird`-klassen.

- Sidepunktene til trekanten finner man med samme fremgangsmåte som for frontpunktet, men i stedet for θ bruker man $\theta + \frac{2}{3}\pi$ for det ene sidepunktet og $\theta - \frac{2}{3}\pi$ for det andre sidepunktet. π er omtrent 3.1415926535.

Hint: Man kan finne informasjon om bruken av matematiske funksjoner som er implementert i standardbiblioteket ved å gå til C++ reference og siden som heter *Common math functions*.

Noter deg at de trigonometriske funksjonene i standardbiblioteket opererer i radianer.

Når oppgave T8 er ferdig skal simuleringen se ut som i Figur 6 etter at den har kjørt litt.



Figur 6: Skjerm bilde av simuleringen etter at oppgave T8 er implementert.

9. (25 points) **T9: FLOKKEN STILLER LIKT!**

For å synliggjøre hvilken flokk en due tilhører ønsker vi at fargen til alle duer i samme flokk skal være lik. En flokk består av alle duene som er lagret som venner i tillegg til alle duene dine venner har lagret som venner.

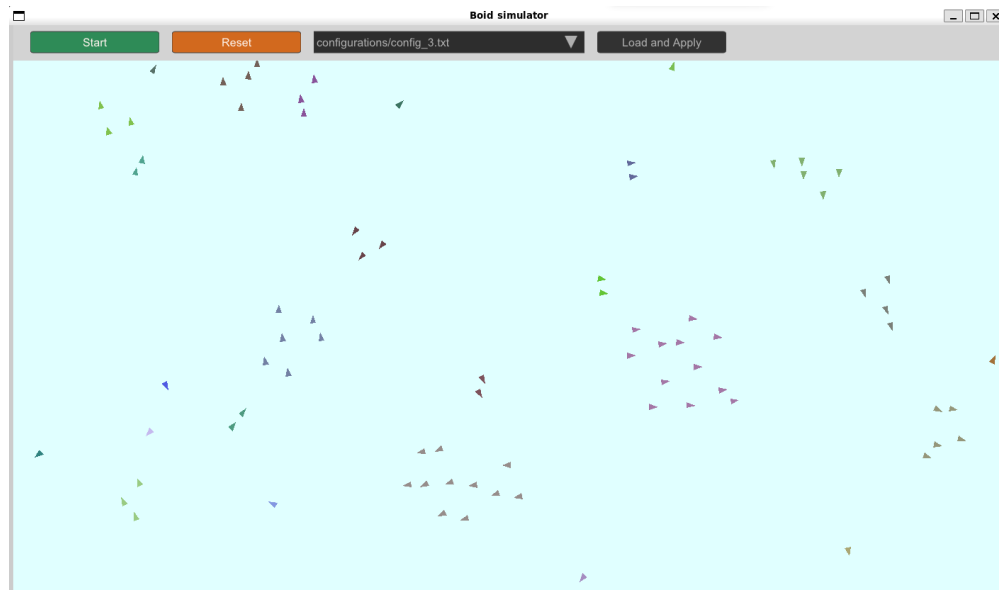
I koden har hver fugl to farger; `originalColor` og `displayColor`. `originalColor` er fargen fuglen ble tildelt da den ble opprettet. Denne fargen kan man ikke endre. `displayColor` er fargen som fuglen viser på skjermen. Denne fargen kan man endre med metoden `setColor` i `Dove`.

Implementer funksjonen `colorBirds` i `Simulator.cpp`.

- Alle duer som er i samme flokk skal ha samme farge.
- Ulike flokker skal helst ha forskjellig farge.

Merk: BEGIN- og END-kommentarene er utenfor funksjonsdefinisjonen, noe som åpner for muligheten til å lage egne hjelpefunksjoner eller globale variabler for å løse oppgaven.

Når oppgave T9 er implementert skal simuleringen se ut som i Figur 7 etter at den har kjørt litt.



Figur 7: Skjerm bilde av simuleringen etter at oppgave T9 er implementert.

10. (20 points) **T10: Throw more doves!**

Vi ønsker å kunne legge til flere duer mens simuleringen kjører.

Implementer funksjonen `addBird` i `Simulator.cpp`.

- Legg til en ny due hvis knappen D på tastaturet holdes inne samtidig som man klikker på venstre museknapp i simuleringsvinduet.
- Startposisjonen til den nye duen skal være gitt av koordinatene til museklikket.
- Starthastigheten og startfargen til den nye duen er valgfrie.

Når oppgave T10 er implementert skal man kunne lage nye duer med museklikk dersom korrekt tast også holdes inne.

11. (25 points) **T11: Hauken kommer!**

For å gjøre simuleringen enda mer spennende ønsker vi å introdusere hauker som duene skal prøve å unngå. I denne oppgaven skal du oppdatere `Hawk`-klassen i `Simulator.h`. Alle metodene og attributene som `Hawk`-klassen trenger er allerede implementert. Det eneste som gjenstår er å legge dem til i klassedeklarasjonen.

Oppdater klassen Hawk slik at den inneholder relevante deklarasjoner for metoder og attributter.

- Fjern eller kommenter ut kodelinjen `#define HAWK_IS_IMPLEMENTED`.
- Bruk tilbakemeldingene du nå får fra kompilatoren til å oppdatere klassesdeklarasjonen til Hawk.
- Husk å tenk gjennom hvilket synlighetsnivå de ulike attributtene og metodene bør ha.

Tips: Du kan legge til kodelinjen `#define HAWK_IS_IMPLEMENTED` igjen dersom du ikke har løst oppgaven enda og ønsker å fjerne feilene du får fra kompilatoren.

Når oppgave T11 er gjort er simuleringen ferdigstilt, og du skal kunne se simuleringen som i Figur 2.