

## Del 3: Kabal

Del 3 av eksamen er programmeringsoppgåver. Denne delen inneheld **11 oppgåver** som til saman gir ein maksimal poengsum på 180 poeng og tel **ca. 60 %** av eksamen. Kvar oppgåve gir maksimal poengsum på **5 - 30 poeng** avhengig av vanskegrad og arbeidsmengd.

Den utdelte koden inneheld kompillerbare (og kjørbare) .cpp- og .h-filer med forhåndskoda delar og ei full beskrivelse av oppgåvene i del 3 som ei PDF. Etter å ha lasta ned koden står du fritt til å bruke eit utveklingsmiljø (VS Code) for å jobbe med oppgåvene.

Du kan lasta ned .zip-fila med den utdelte koden frå Inspira. I neste seksjon finn du ei beskriving av korleis du gjer dette. Før du leverer eksamen, må du hugse å lasta opp ei .zip-fil med den utdelte koden og endringene du har gjort når du har løyst oppgåvene.

## Nedlasting og oppsett av den utdelte koden

Dei seks stega under beskriv korleis du kan lasta ned og setje opp den utdelte koden. Ei bildebeskriving av stega 1 til 5 kan du se i Figur 10.

1. Last ned .zip-fila frå Inspira
2. Lokaliser fila i File Explorer (Downloads-mappa)
3. Pakk ut .zip-fila ved å høgreklikke på fila og vel:  
*7-Zip → Extract here*
4. Kopier mappa som dukkar opp til C:/Temp
5. Opne mappa som ligg i C:/Temp i VS Code ved å velje:  
*File → Open Folder ...*

6. Køyr følgjande kommando i VS Code:

`Ctrl+Shift+P → TDT4102: Create project from TDT4102 template → Configuration only`

## Sjekkliste ved tekniske problem

Viss prosjektet du opprettar med den utdelte koden ikkje kompilerer (før du har lagt til eigne endringer) bør du rekkje opp hânda og be om hjelp. Medan du ventar kan du prøva følgjande:

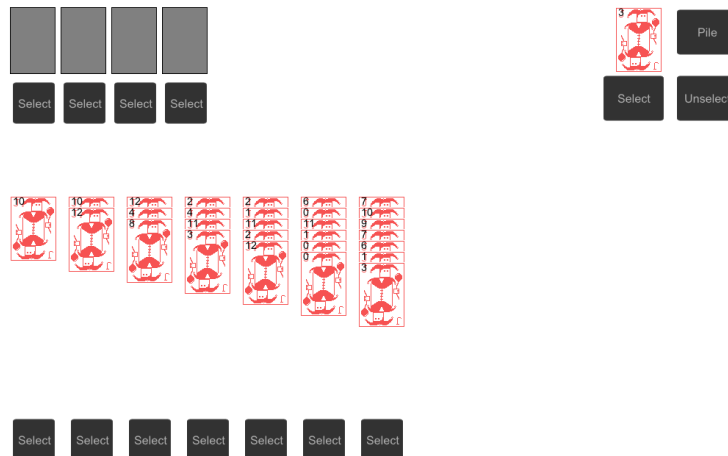
1. Sjekk at prosjektmappa er lagret i mappa C:\temp.
2. Sjekk at namnet på mappa **IKKJE** inneheld norske bokstavar (*Æ, Ø, Å*) eller mellomrom. Dette kan skapa problem når du prøver å køyre koden i VS Code.
3. Sørg for at du er inne i rett mappe i VS Code.
4. Køyr følgjande TDT4102-kommando:  
(a) `Ctrl+Shift+P → TDT4102: Create project from TDT4102 template → Configuration only`
5. Prøv å køyra koden igjen (`Ctrl+F5` eller `Fn+F5`).

6. Viss det framleis ikkje fungerer, lukk VS Code vindauget og opne det igjen. Gjenta deretter steg 5-6.

## Introduksjon

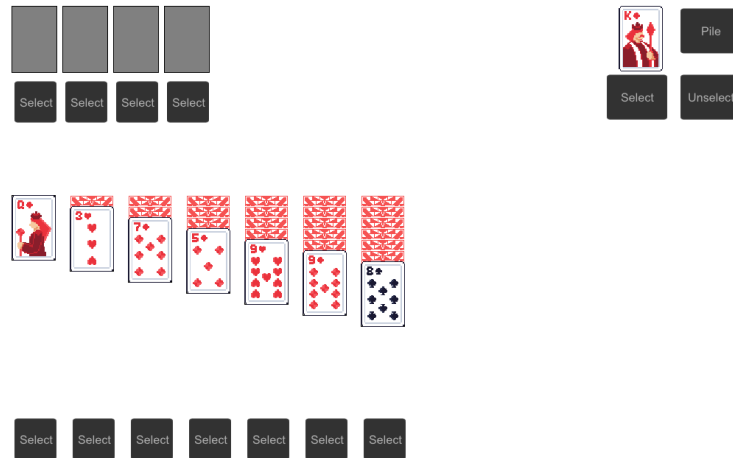
Kabal er eit korstpill som vanlegvis kun har éin spelar. Målet er å sortere ein tilfeldig kortstokk etter ein bestemt orden og mønster. Her er ei beskriving av korleis ein spiller kabal:

- Spelet begynner med ein standard kortstokk på 52 kort som blir stokka tilfeldig.
- Sju bunkar med kort blir lagt ut på bordet. Den første bunken har eitt kort, den andre bunken har to kort, og så vidare, opp til den sjuande bunken som har sju kort. Berre det øvste kortet i kvar bunke er synleg, resten ligg med bilesida ned.
- Målet er å flytte alle korta til fire fundament-bunkar som er tomme i begynninga. Kvar fundament-bunke skal byggast opp i stigande rekkefølgd, fra ess til konge, og må vere av samme sort (hjerter, ruter, kløver eller spar).
- På bordet kan kort flyttast frå ein av dei sju bunkane til ein annan bunke, men korta må bli lagt i synkande rekkefølgd og men vekslende farge (raud, svart, raud, osv.). Viss ein av dei sju bunkane blir tomme, kan ein konge (eller ein sekvens som startar med en konge) flyttast dit.
- Kort som ikkje er i nokon av dei sju bunkane er i ein eigen stokk. Spelaren kan trekke kort frå denne stokken, vanlegvis eitt av gongen, og prøve å bruke dei i bunkane eller fundament-bunkene.
- Spelet er vunne viss alle korta er plassert i fundament-bunkane i riktig rekkefølgd. Viss ingen fleire trekk er moglege og korta ikkje er fullstendig sortert, er spelet tapt.



**Figur 1:** Skjerm bilde av køyring av den utdelte koden.

Den utdelte koden inneheld mange filer, men du skal berre forhalde deg til `Tasks.cpp`, `ImageAtlas.cpp`, og `Card.cpp` og tilhøyrande header-filer. Dei andre filene treng du ikkje sjå på for å kunne svare på oppgåvene. All informasjon som trengst for å svare på oppgåvene står oppgeve i oppgåveteksten. Før du startar må du sjekka at den (umodifiserte) utdelte koden køyrer utan problem. Du skal sjå det same vindauget som i figur 1.



Figur 2: Skjerm bilde av den ferdige applikasjonen.

## Slik fungerer applikasjonen

Frå start ser spelet ganske uinteressant ut. Det liknar kanskje litt på kabal, men det ser ikkje heilt riktig ut. Når applikasjonen er ferdig implementert er spelet fungerande med varierte kort og fungerande spillologg. Du kan sjå korleis den ferdige applikasjonen skal sjå ut i Figur 2.

Tilstanden i spelet blir handtert gjennom `main.cpp`-fila. Denne fila skal **IKKJE** rørast. Ein del av kjernefunksjonaliteten i applikasjonen er allereie implementert. Din oppgåve er å implementere logikk og grafikk som gjer applikasjonen til eit fullverdig spel.

## Korleis svare på oppgåvene

Kvar oppgåve i del 3 har ein unik kode for å gjere det lettare for deg å vite kor du skal fylle inn svara dine. Koden er på formatet <T><siffer> (TS), der siffera er mellom 1 og 11 (T1 - T11). For kvar oppgåve vil ein finne to kommentarar som skal definere høvesvis begynninga og slutten av svaret ditt. Kommentrane er på formatet: // BEGIN: TS og // END: TS.

For eksempel kan ei oppgåve sjå slik ut:

```
// TASK T1
bool foo(int x, int y) {
// BEGIN: T1
// Write your answer to assignment T1 here, between the //BEGIN: T1
// and // END: T1 comments. You should remove any code that is
// already there and replace it with your own.

return false;

// END: T1
}
```

Etter at du har implementert løysinga di bør du enda opp med følgjande:

```
// TASK T1
bool foo(int x, int y) {
// BEGIN: T1
// Write your answer to assignment T1 here, between the //BEGIN: T1
// and // END: T1 comments. You should remove any code that is
// already there and replace it with your own.

    return (x + y) % 2 == 0;

// END: T1
}
```

**Det er veldig viktig at alle svara dine blir ført mellom slike par av kommentarar.** Dette er for å støtte sensurmekanikken vår. Viss det allereie er skriven kode *mellom* BEGIN- og END-kommentarane til ei oppgåve i kodeskjelettet som er gitt ut, så kan du, og ofte bør du, erstatte denne koden med din eigen implementasjon. All kode som er skriven *utanfor* BEGIN- og END-kommentarane **SKAL** du la stå som den er. I oppgåve T11 er det ikkje nokon funksjonsdeklarasjon eller funksjonsdefinisjon mellom BEGIN- og END-kommentarane, så her må du skrive all kode sjølv, og det er viktig at du skriv all koden din mellom BEGIN- og END-kommentarane for å få uttelling for oppgåven.

**Merk:** Du skal **IKKJE** fjerne BEGIN- og END-kommentarane.

Dersom du synast nokon av oppgåvene er uklare kan du oppgje korleis du tolkar dei og det du antar for å løyse oppgåva som kommentarar i koden du leverer.

**Tips:** trykker ein CTRL+SHIFT+F og søker på BEGIN: får ein snarvegar til starten av alle oppgåvene lista opp i utforskervindauget så ein enkelt kan hoppe mellom oppgåvene. For å kome tilbake til det vanlege utforskervindauget kan ein trykke CTRL+SHIFT+E.

# Oppgåvene

## Legg fram korta (40 poeng)

```
enum class CardColor {
    BLACK,
    RED
};

enum class Suit {
    SPADES,
    CLUBS,
    HEARTS,
    DIAMONDS
};

struct Card : public Entity {

    // Constructors...

    int get_rank() const;
    Suit get_suit() const;
    bool is_flipped() const;
    void set_flipped(bool flipped_);
    void flip();
    CardColor get_color() const;
    std::string get_identifier() const;

private:
    int rank;
    Suit suit;
    bool flipped = false;
};
```

**Figur 3:** Klassedeklarasjonar for CardColor, Suit og Card

### 1. (5 points) T1: Opp eller ned?

Card-klassen (Figur 3) har ein medlemsvariabel `flipped` som seier om eit kort ligg med bilesida opp (`flipped = false`) eller ned (`flipped = true`). Medlemsfunksjonen `is_flipped` blir brukt i koden til å henta ut verdien til `flipped`, men for augeblinken returnerer `is_flipped` alltid `false`, så alle korta ligg med bilesida opp på skjermen. Endre funksjonen `is_flipped` så den returnerer verdien til `flipped`.

Implementer funksjonen `Card::is_flipped` i `Card.cpp`.

Når oppgåve T1 er gjort vil du kunne sjå at nokon av korta ligg med bilesida ned, medan andre kort

ligg med bildesida opp.

## 2. (10 points) T2: Rødt eller svart?

Card-klassen (Figur 3) har ein medlemsfunksjon `get_color` som blir brukt i koden til å finne fargen til kortet. Vi vil at `get_color` skal returnere fargen til kortet basert på sorten til kortet. Sorten til eit kort blir beskrive av medlemsvariabelen `Card::suit`, som er ein instans av enumklassen `Suit`. Fargen til eit kort er enten `CardColor::BLACK` eller `CardColor::RED`. For augoblinken returnerer `get_color` alltid `CardColor::RED`, så alle korta er raude på skjermen.

Endre funksjonen `Card::get_color` i `Card.cpp` sånn at den returnerer den riktige fargen til kortet.

- Funksjonen skal returnere `CardColor::BLACK` for kort av sortane `Suit::SPADES` og `Suit::CLUBS`.
- Funksjonen skal returnere `CardColor::RED` for kort av sortane `Suit::HEARTS` og `Suit::DIAMONDS`.

Når oppgåve T2 er gjort vil du kunne sjå både raude og svarte jokerar.

## 3. (15 points) T3: Flytt fleire kort samtidig

Knappane merka med `Select` vel berre det fremste kortet i bunken, men i kabal er det mogleg å flytte fleire kort samtidig frå ein bunke til ein annan. Funksjonaliteten for å plukke opp fleire kort samtidig er for det meste allereie implementert, men éin av funksjonane må endrast. Funksjonen `inside_box` i `Tasks.cpp` skal returnere `true` dersom eit gitt punkt ligg på innsida av eit gitt rektangel, men for augoblinken returnerer den alltid `false`.

Funksjonen `inside_box` tar inn to parameterar: eit punkt `point` og ein instans av `Rectangle` kalla `rectangle`. `Rectangle`-strukturen ser sånn ut:

```
struct Rectangle {  
    TDT4102::Point top_left;  
    int width;  
    int height;  
};
```

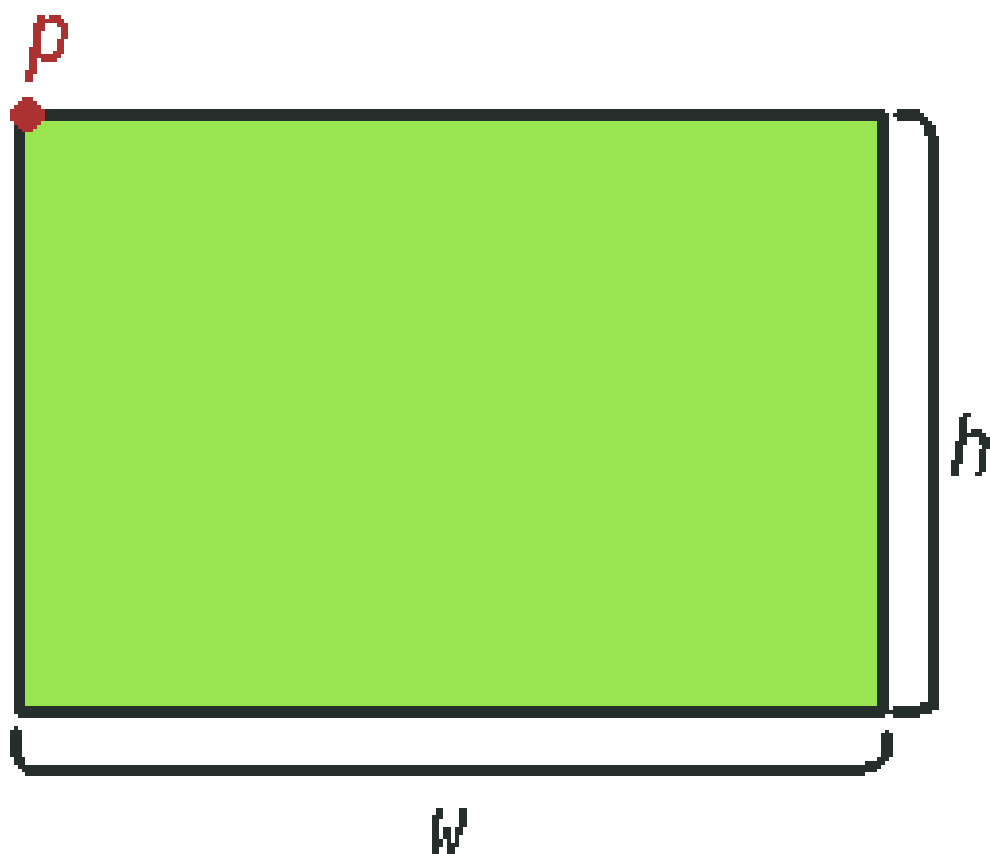
Funksjonen skal avgjere om `point` ligg på innsida av `rectangle`. Ein illustrasjon av problemstillinga finn du i Figur 4.

Endre funksjonen `inside_box` i `Tasks.cpp`.  
Gitt `point` og `rectangle`, returner `true` dersom:

- `point.x` ligg mellom  $x_r$  og  $x_r + w$ , og
- `point.y` ligg mellom  $y_r$  og  $y_r + h$ ,

der  $x_r$  og  $y_r$  er komponentane av `rectangle.top_left` og  $w$  og  $h$  er høvesvis felte `width` og `height` i `rectangle`.

Når oppgåve T3 er gjort vil det vere mogleg å velje fleire kort frå ein bunke og flytte dei samtidig frå ein bunke til ein annan. Korta ein vel vil markerast i gult.



**Figur 4:** Illustrasjon av gyldig område inne i eit rektangel.

#### 4. (10 points) T4: Snu øverste kort

Når kortet som ligg øvst i ein bunke med bildesida opp blir flytta til ein annan bunke vil dei resterande korta i bunken liggje med bildesida ned, eller så vil bunken vere tom. Vi vil endre på dette sånn at det øvste kortet i bunken alltid ligg med bildesida opp. Når det øvste kortet i bunken blir flytta til ein annan bunke skal derfor kortet som no blir liggjande øvst i bunken bli snudd med bildesida opp gitt at bunken ikkje er tom.

Funksjonen som skal handtere dette er `stack_set_flip` i `Tasks.cpp`. Funksjonen tar ein vektor av `Card`-objekt og skal snu det siste kortet i vektoren med bildesida opp, altså skal `flipped` bli satt til `false` for det siste kortet i vektoren.

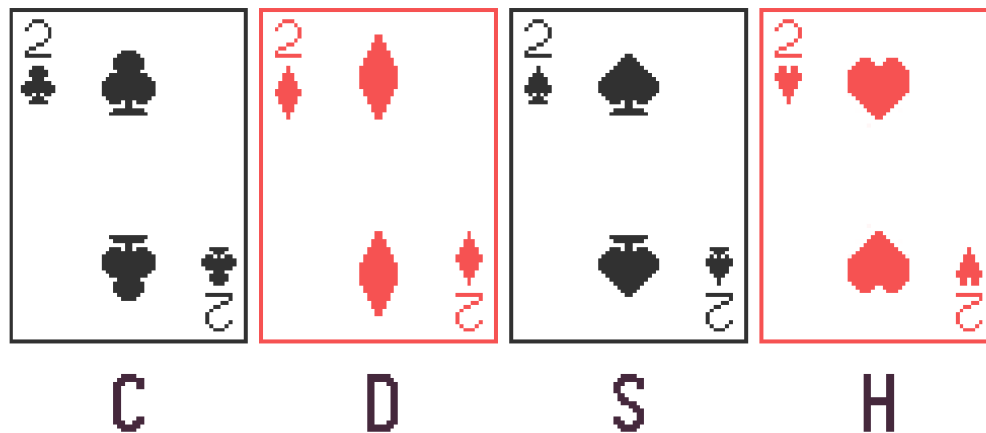
Implementer funksjonen `stack_set_flip` i `Tasks.cpp`.

- Sett `flipped` til `false` for det siste kortet i vektoren.
- Hugs å ta høgd for at bunken kan vere tom.

**Hint:** `Card`-klassen har ein medlemsfunksjon `set_flipped` som kan brukast til å setje verdien til medlemsvariabelen `flipped`.

Når oppgåde T4 er gjort skal du kunne sjå at det øvste kortet i en bunke alltid ligg med bildesida opp (gitt at bunken ikkje er tom).

#### Fiks kortene! (55 poeng)



Figur 5: Kvar sort med tilhørande bokstavar.

#### 5. (15 points) T5: Korta får nytt utsjåande - Kort til streng

Eit kort kan identifiserast av ein streng som inneheld informasjon om verdien til kortet og sorten til kortet. Strengen er på formatet `VS`. `V` er rangen til kortet (`Card::rank`) pluss 1 og vil vere mellom 1 og 13. `S` er sorten til kortet og vil vere en av bokstavene 'S', 'C', 'H' eller 'D' for å indikere høvesvis `Suit::SPADES` (spar), `Suit::CLUBS` (kløver), `Suit::HEARTS` (hjerter) og `Suit::DIAMONDS` (ruter). Dette kan du sjå i Figur 5. I tabellen under kan du sjå strengrepresentasjonen for nokon utvalde korttypar.



| Verdi    | Sort                   | Strengrepresentasjon |
|----------|------------------------|----------------------|
| ess      | Suit::CLUBS (kløver)   | 1C                   |
| 2        | Suit::HEARTS (hjerter) | 2H                   |
| konge    | Suit::HEARTS (hjerter) | 13H                  |
| dronning | Suit::DIAMONDS (ruter) | 12D                  |
| knekt    | Suit::SPADES (spar)    | 11S                  |

For å kunne skrive ut informasjon om eit kort ynkjar vi å ha ein funksjon som kan ta eit kort og returnere strengrepresentasjonen beskrive ovenfor.

Implementer `Card::get_identifier` i `Card.cpp`.

Funksjonen tek ein referanse til eit `Card`-objekt og skal returnere ein streng på formatet `VS`, der `V` er verdien til kortet og `S` er ein bokstav som beskriv sorten til kortet.

- Verdien til kortet finn ein ved å ta medlemsvariabelen `rank` og leggje til 1.
- Sorten til kortet finn ein i medlemsvariabelen `suit`.
- Sorten til kortet blir beskrive slik:

Suit::SPADES → 'S'

Suit::CLUBS → 'C'

Suit::HEARTS → 'H'

Suit::DIAMONDS → 'D'

#### 6. (15 points) T6: Korta får nytt utsjåande – Kort til bildesti

No som vi har ein streng som kan identifisere eit kort, kan vi lage ein sti til eit bilde som representerer kortet. Ein titt inni `images`-mappa vil vise ei rekke bilde med namn på formatet `VS.png`, der `VS` er stringrepresentasjonar av ulike kort slik det blei beskrive i oppgåve T5. Den fullstendige stien til eit kort er dermed `images/VS.png`. Stien til sjølve bildemappa `images` kan du finne i ein global variabel med namn `IMAGE_DIR` som innehld verdien `"images/"` og er definert i `common.h`.

Vi ynkjar no å lage ein funksjon som tek inn eit kort og returnerer stien til det tilhøyrande bildet.

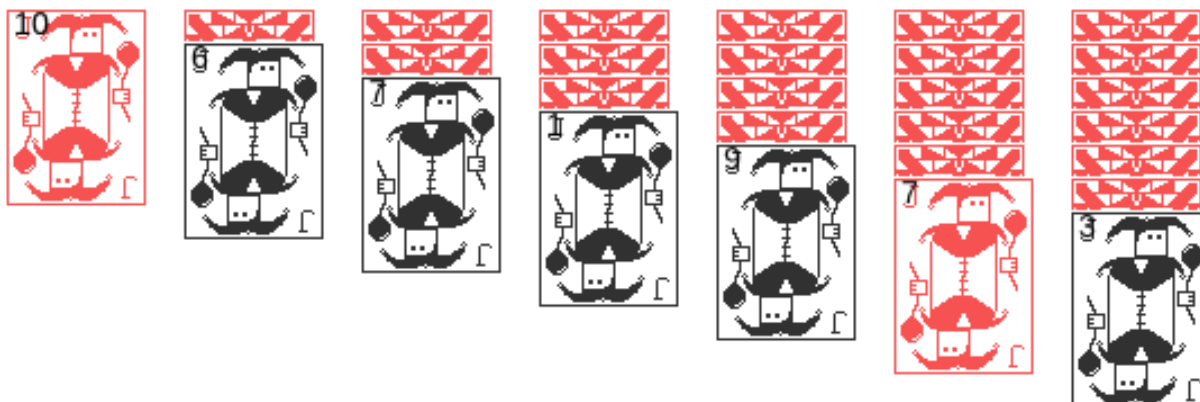
**Eksempel:** Dersom funksjonen tek inn eit kort av typen kløver dronning skal funksjonen returnere strengen `"images/12C.png"`.

Implementer funksjonen `get_card_image_path` i `Tasks.cpp`.

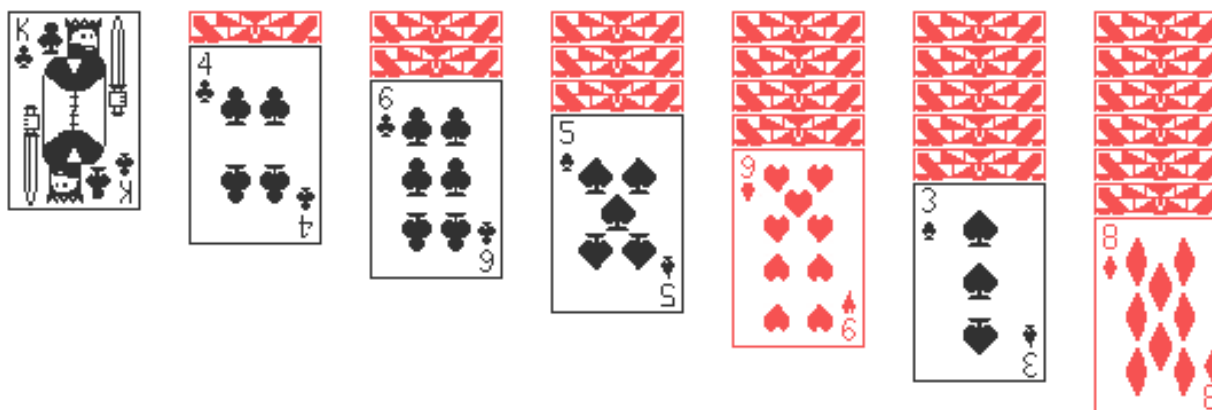
Funksjonen tek ein referanse til eit `Card`-objekt og skal returnere ein streng som er stien til bildet som skal visast for denne korttypen. Strengen skal vere samansett av

- den globale strengen `IMAGE_DIR`,
- resultatet av `Card::get_identifier` som du implementerte i oppgåve T5, og
- suffiks-strengen `".png"`.

**Eksempel:** Viss kortet er av typen hjerter-ess skal funksjonen returnere strengen `"images/1H.png"`.



Figur 6: Eksempel på korleis korta ser ut før oppgåve T5, T6 og T7 er gjort



Figur 7: Eksempel på korleis korta ser ut etter oppgåve T5, T6 og T7 er gjort

7. (15 points) **T7: Korta får nytt utsjåande – sjekk at bildet finst**

Før vi kan laste inn bilda til korta må vi dobbelsjekke at det eksisterer eit bilde for dei ulike korttypane. Vi treng derfor ein funksjon som sjekker at stien til bildet er gyldig.

Endre funksjonen `ImageAtlas::has_image` i `ImageAtlas.cpp` sånn at den returnerer `true` dersom nøkkelen `key` eksisterer i `container`.

Når oppgåve T5, T6 og T7 er gjort skal du kunne sjå at utsjåande til korta har endra seg. Eksempel på korleis korta ser ut før og etter oppgåvene er gjort kan du sjå i høvesvis Figur 6 og Figur 7.

8. (20 points) **T8: Gyldig trekk?**

Så langt har alle kort kunna blitt flytt fra bunke til bunke utan nokon avgrensingar, men vi må ikkje gløyma at det er reglar i kabal.

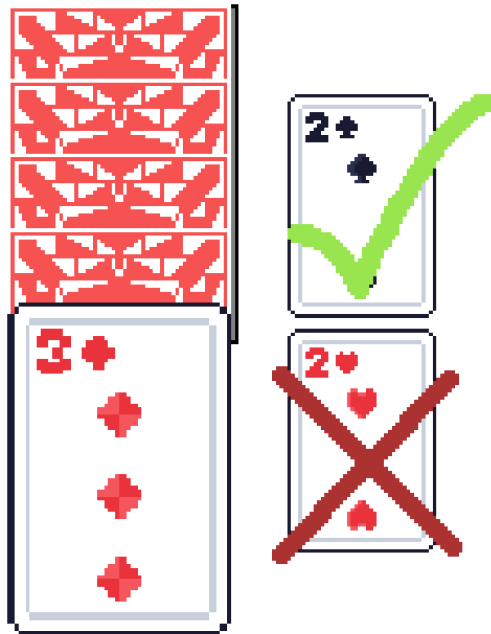
Vi skal no syrgje for at ein må følgje desse reglane for å leggje eit kort på toppen av ein bunke:

- Viss bunken er tom kan det **kun** leggst på ein konge.

- Viss bunken ikkje er tom, kan du leggje på eit kort som (1) har motsatt farge av den som ligg på toppen og (2) har ein verdi som er éin mindre enn kortet på toppen. Dette er illustrert i Figur 8.

Implementer funksjonen `can_push_to_tableau` i `Tasks.cpp`. Funksjonen tek eit kort (`card`) og ein bunke med kort (`cards`) og skal returnere `true` dersom kortet kan leggst på bunken og `false` elles.

Når oppgåve T8 er gjort skal det ikkje vere mogleg å flytte korta med mindre ein følger reglane.



Figur 8: . Gyldige og ugyldige kort.

## Inn/ut datahandtering og operatoroverlasting (75 poeng)

### 9. (20 points) T9: Gyldige filutvidingar

For augeblinken er bilda som blir brukt for korta hardkoda og bestemt på førehand. No ynkjar vi å utvide funksjonaliteten til spelet så ein kan velje mellom fleire ulike bilde for eit kort. Vi ynkjar derimot ikkje å støtta alle bilde, så vi treng ein funksjon som sjekker at bildet ein prøver å bruke har ein filutviding som er støtta.

Implementer funksjonen `is_valid_image_path` i `Tasks.cpp`.

- Funksjonen tek inn ein parameter `path` som er ein fullstendig sti til ei bildefil.
- Funksjonen skal returnere `true` dersom filutvidinga til filstien `path` er `".png"`, `".jpg"` eller `".bmp"`, og `false` elles.

**Hint:** Fordi nokon filer kan ha filutvidingar med store bokstavar kan det løne seg å omgi filutvidinga sånn at den har små bokstavar. Dette kan gjerast med funksjonen `extension_to_lower(extension)`.

10. (30 points) **T10: Innlasting av bildeatlas**

Det siste vi må gjere for å kunne velje mellom ulike bilde for eit kort, er å lese inn fila som beskriv kva for bilde som skal brukast for kvart kort. Vi har laga ei fil som beskriv kort og tilhøyrande bilde (sjå `images/alternative/manifesto.txt`). Denne fila følgjer oppsettet i Figur 9.

Dersom fila let seg opne skal vi behandle fila linje for linje ved å lese dei tre felte i kvar linje (se Figur 9), tilarbeide kvar linje, og sette inn bildet som linja beskriv i bildeatlas.

Implementer `load_image_atlas` i `ImageAtlas.cpp`.

- Opne fila som er oppgitt i parameteren `path`.
- Viss fila ikkje let seg opne, utløys eit unntak av typen `std::runtime_error`.
- Behandle fila linje for linje (se eksempelet nedafor):
  - Les inn felte `RANK`, `SUIT` og `PATH`.
  - Dann kortet sin nøkkel (`key`) ved å slå saman ein streng som består av `RANK + 1` og `SUIT`.
  - Dann bildet sin fulle sti (`full_path`) ved å danne strengen `IMAGE_DIR + PATH`.
  - Legg til bildet i atlaset ved å kalle `atlas.addImage(key, fullPath)`.
- Lukk fila når alle linjene har blitt lest.

**Eksempel:**

```
0 S alternative/card000.png
```

Skal leggje til eit bilde for spar-ess ved å gjere eit kall tilsvarende  
`atlas.addImage("1S", "images/alternative/card000.png");`

11. (25 points) **T11: Innsettingsoperator for Card**

I oppgåve T5 implementerte du ein funksjon som returnerer ein strengrepresentasjon av eit Card-objekt. Vi vil no overlaste innsettingsoperatoren (`operator<<`) til ein `std::ostream` for Card-klassen sånn at strengrepresentasjonen til kortet blir skrive når ein skriv eit Card-objekt til ein `std::ostream`.

Overlast innsettingsoperatoren (`operator<<`) for Card-klassen i `Tasks.cpp`.

Når oppgåve T11 er gjort skal det vere modleg å skrive eit kort til ein standard output-strøm:

```
std::cout << card << "\n"
```

**Merk:** Oppgåva ber deg om å skrive heile overlasteringa. Hugd å skriv svaret ditt innafor BEGIN- og END-kommentarane til oppgåve T11 for å få utteljing for oppgåven.

## Image Atlas filstruktur

Eit atlas blir lagra med følgjande filformat:

- Kvar linje har tre felt: RANK, SUIT, og PATH.
- Dei tre linjene er separert av ein tabulator (\t)

**Eksempel:** 0 S card000.png representerer eit spar-ess

```
0 S card000.png
1 S card001.png
2 S card002.png
3 S card003.png
4 S card004.png
5 S card005.png
6 S card006.png
```

**Figur 9:** Image Atlas filstruktur

**i Oppgave/utdelt kode del 3**

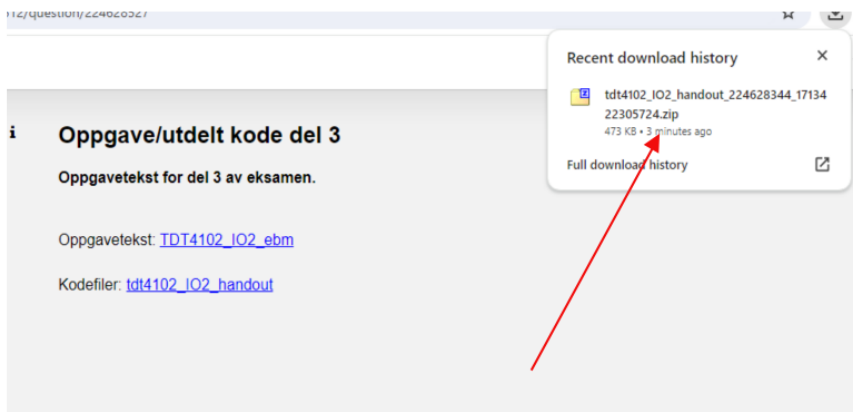
Oppgavetekst for del 3 av eksamen.

Oppgavetekst: [TDT4102\\_IO2\\_ebm](#)

Kodefiler: [tdt4102\\_IO2\\_handout](#)



(a) Steg 1




**i Oppgave/utdelt kode del 3**

Oppgavetekst for del 3 av eksamen.

Oppgavetekst: [TDT4102\\_IO2\\_ebm](#)

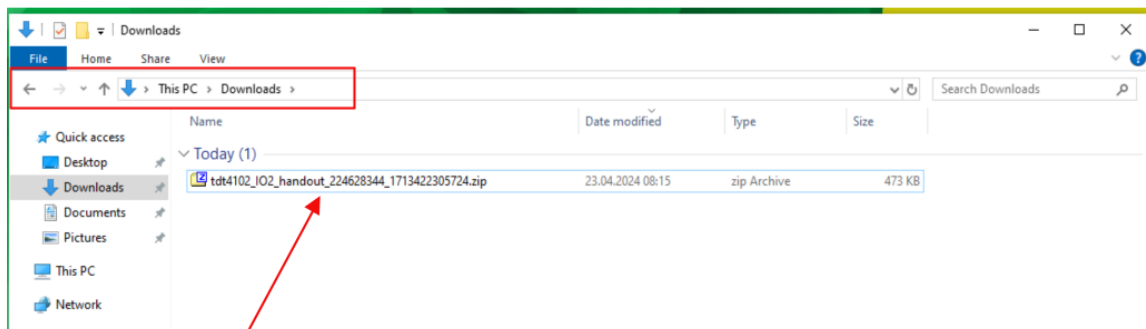
Kodefiler: [tdt4102\\_IO2\\_handout](#)

Recent download history

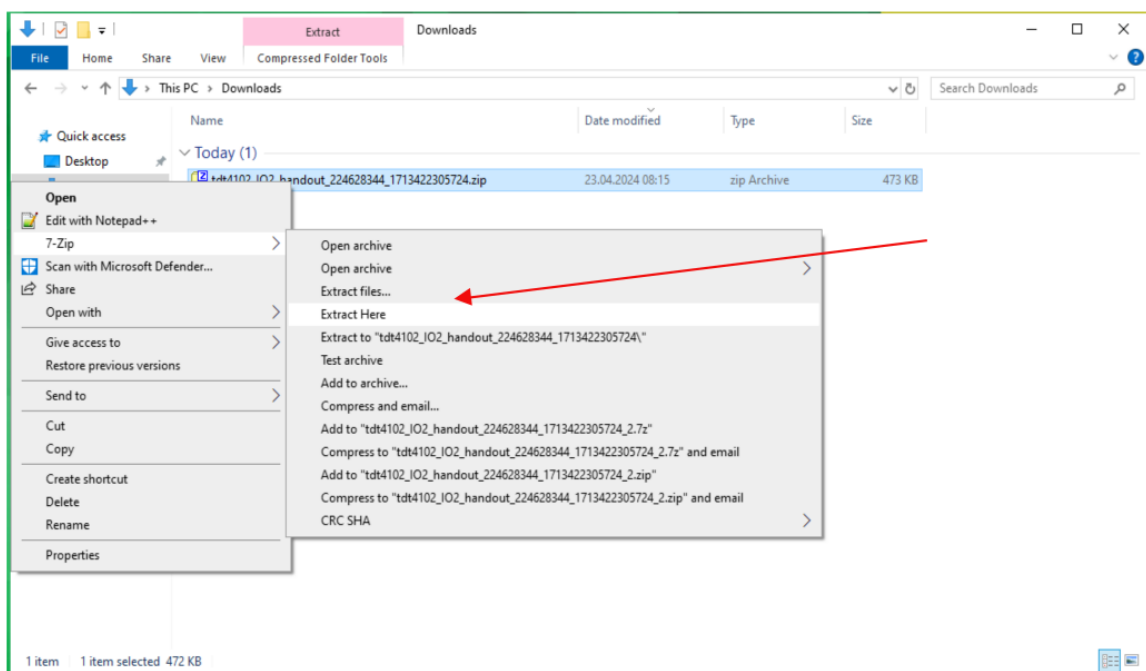
 tdt4102\_IO2\_handout\_224628344\_17134\_22305724.zip

473 KB • 3 minutes ago

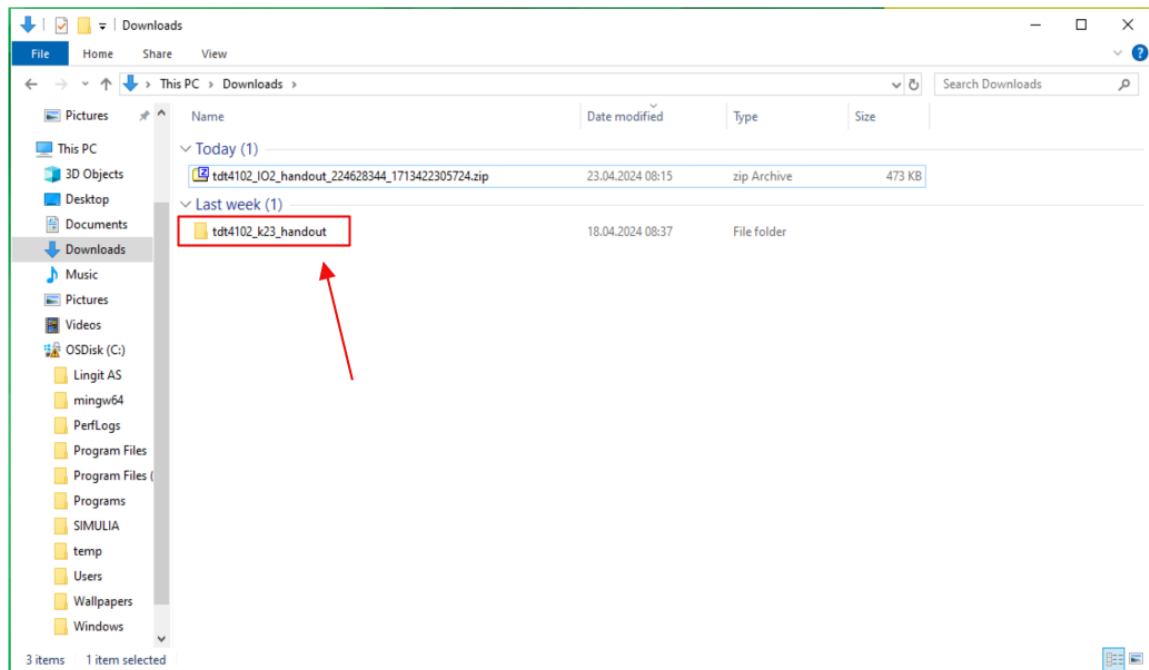
Full download history



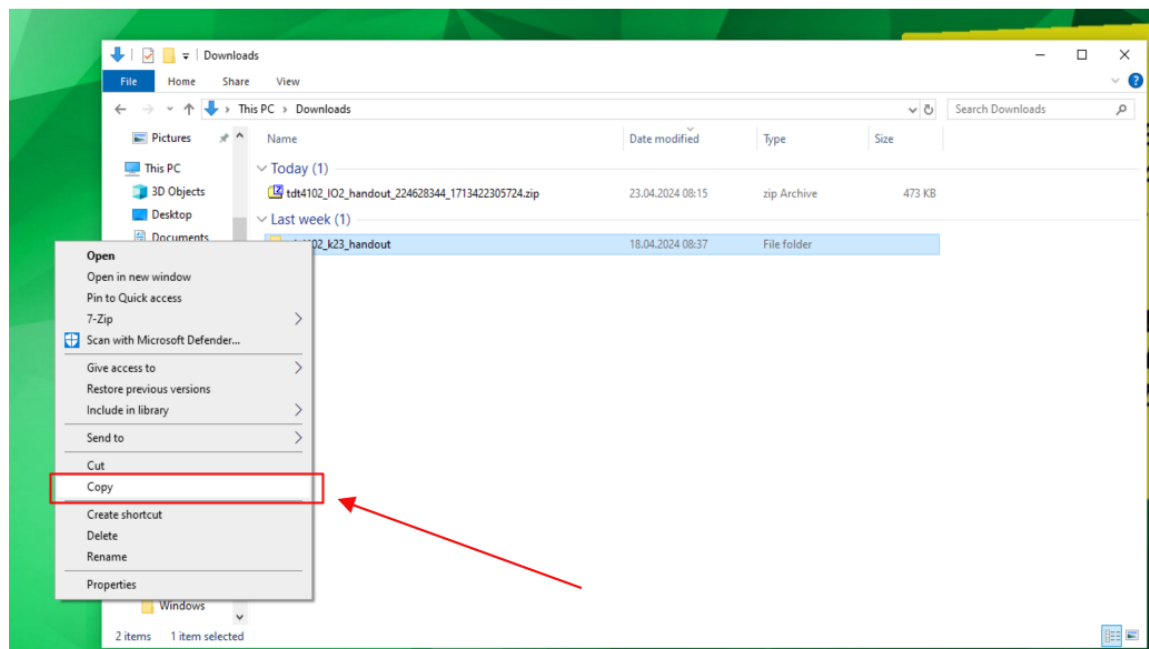
(c) Steg 2



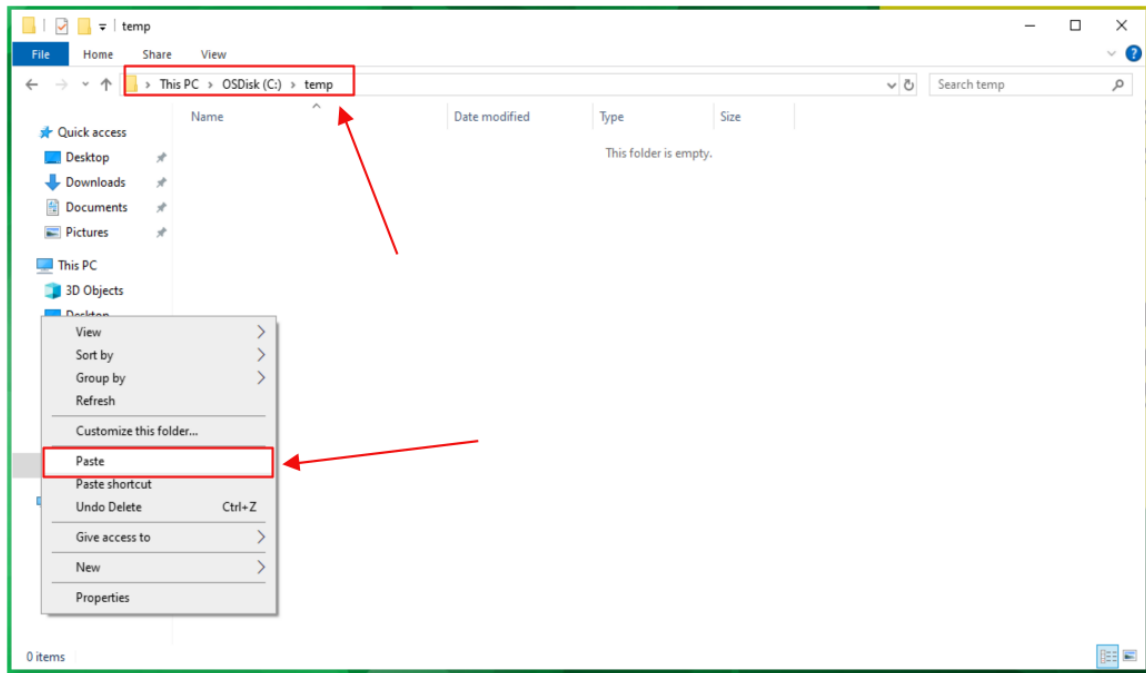
(d) Steg 3



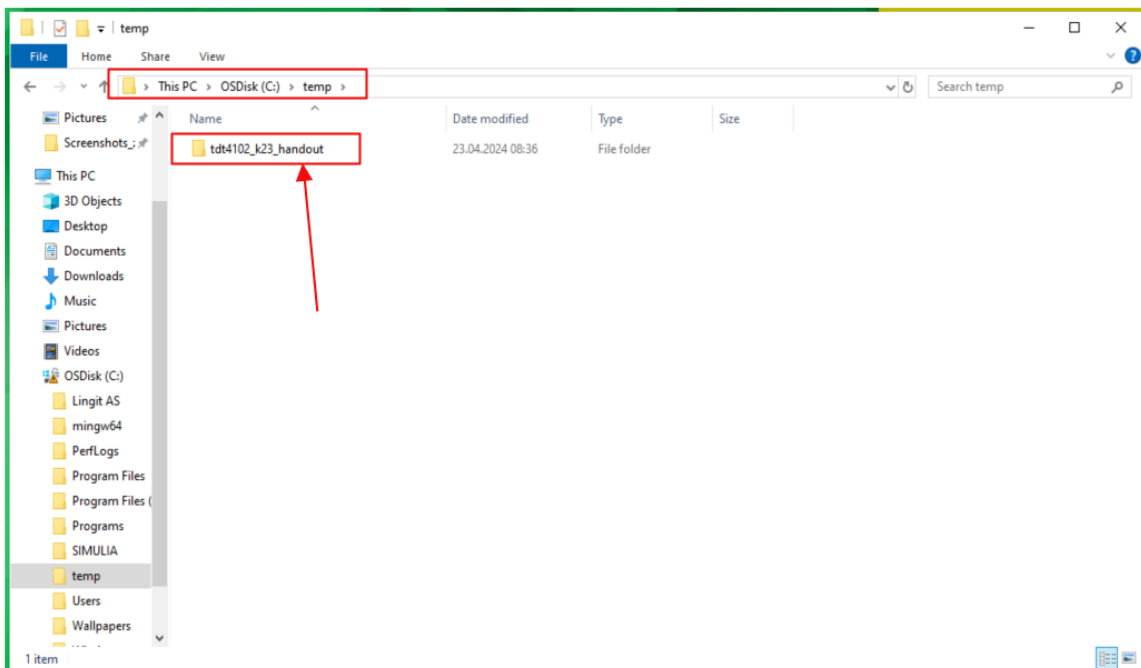
(e) Steg 3



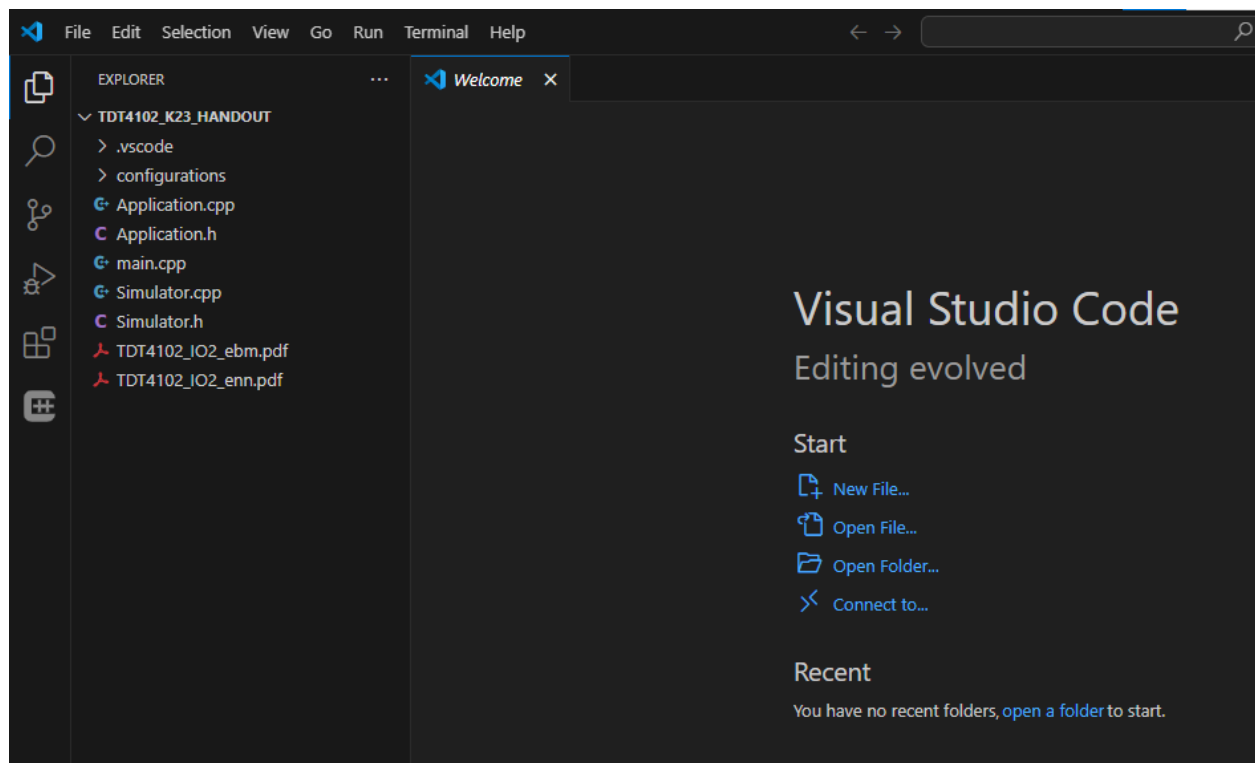




(g) Steg 4







(k) Steg 5

**Figur 10:** Nedlasting og oppsett av den utdelte koden.