

Del 3: Kabal

Del 3 av eksamen er programmeringsoppgaver. Denne delen inneholder **11 oppgaver** som til sammen gir en maksimal poengsum på 180 poeng og teller **ca. 60 %** av eksamen. Hver oppgave gir *maksimal poengsum* på **5 - 30 poeng** avhengig av vanskelighetsgrad og arbeidsmengde.

Den utdelte koden inneholder kompillerbare (og kjørbare) .cpp- og .h-filer med forhåndskodede deler 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.

Du kan laste ned .zip-filen med den utdelte koden fra Inspera. I neste seksjon finner du en beskrivelse av hvordan du gjør dette. Før du leverer eksamen, må du huske å laste opp en .zip-fil med den utdelte koden og endringene du har gjort når du har løst oppgavene.

Nedlasting og oppsett av den utdelte koden

De seks stegene under beskriver hvordan du kan laste ned og sette opp den utdelte koden. En bildebeskrivelse av stegene 1 til 5 kan du se i Figur 10.

1. Last ned .zip-filen fra Inspera
2. Lokaliser filen i File Explorer (Downloads-mappen)
3. Pakk ut .zip-filen ved å høyreklikke på filen og velg:

7-Zip → Extract here

4. Kopier mappen som dukker opp til C:/Temp
5. Åpne mappen som ligger i C:/Temp i VS Code ved å velge:

File → Open Folder ...

6. Kjør følgende kommando i VS Code:

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

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. Sørg for at du er inne i riktig mappe i VS Code.
4. Kjør følgende TDT4102-kommando:

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

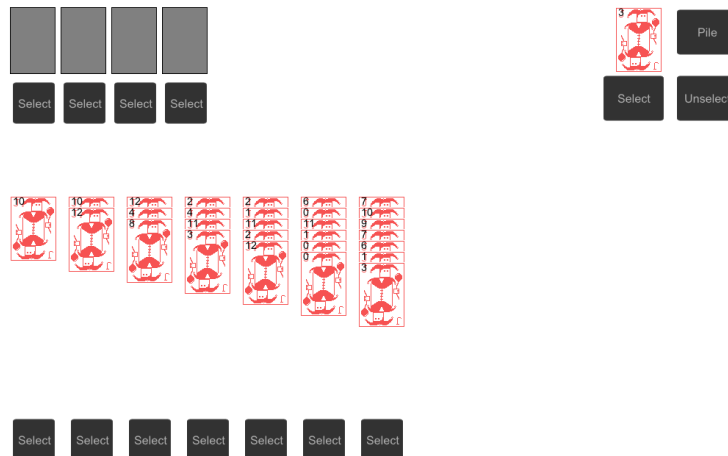
5. Prøv å kjøre koden igjen (`Ctrl+F5` eller `Fn+F5`).

6. Hvis det fortsatt ikke fungerer, lukk VS Code vinduet og åpne det igjen. Gjenta deretter steg 5-6.

Introduksjon

Kabal er et korstpill som vanligvis kun har én spiller. Målet er å sortere en tilfeldig kortstokk etter en bestemt orden og mønster. Her er en beskrivelse av hvordan man spiller kabal:

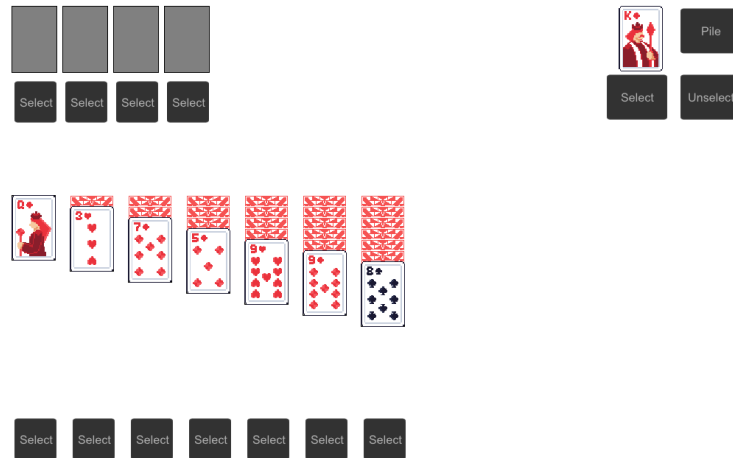
- Spillet begynner med en standard kortstokk på 52 kort som stokkes tilfeldig.
- Sju bunker med kort legges ut på bordet. Den første bunken har ett kort, den andre bunken har to kort, og så videre, opp til den sjuende bunken som har sju kort. Bare det øverste kortet i hver bunke er synlig, resten ligger med bildesiden ned.
- Målet er å flytte alle kortene til fire fundament-bunker som er tomme i begynnelsen. Hver fundament-bunke skal bygges opp i stigende rekkefølge, fra ess til konge, og må være av samme sort (hjerter, ruter, kløver eller spar).
- På bordet kan kort flyttes fra en av de sju bunkene til en annen bunke, men kortene må legges i synkende rekkefølge og med vekslende farge (rød, svart, rød, osv.). Hvis en av de sju bunkene blir tomme, kan en konge (eller en sekvens som starter med en konge) flyttes dit.
- Kort som ikke er i noen av de sju bunkene er i en egen stokk. Spilleren kan trekke kort fra denne stokken, vanligvis ett av gangen, og prøve å bruke dem i spillbrettet eller fundament-bunkene.
- Spillet er vunnet hvis alle kortene er plassert i fundament-bunkene i riktig rekkefølge. Hvis ingen flere trekk er mulige og kortene ikke er fullstendig sortert, er spillet tapt.



Figur 1: Skjermbilde av kjøring av den utdelte koden.

Den utdelte koden inneholder mange filer, men du skal kun forholde deg til `Tasks.cpp`, `ImageAtlas.cpp`, og `Card.cpp` og tilhørende header-filer. De andre filene trenger du ikke se på for å kunne besvare oppgavene. All informasjon som trengs for å besvare oppgavene står oppgitt i oppgaveteksten.

Før du starter må du sjekke at den umodifiserte utdelte koden kjører uten problemer. Du skal se det samme vinduet som i Figur 1.



Figur 2: Skjerm bilde av den ferdige applikasjonen.

Slik fungerer applikasjonen

Fra start ser spillet ganske uinteressant ut. Det ligner kanskje litt på kabal, men det ser ikke helt riktig ut. Når applikasjonen er ferdig implementert er spillet fungerende med varierte kort og fungerende spillogikk. Du kan se hvordan den ferdige applikasjonen skal se ut i Figur 2.

Tilstanden i spillet håndteres gjennom `main.cpp`-filen. Denne filen skal **IKKE** røres. En del av kjernefunksjonaliteten i applikasjonen er allerede implementert. Din oppgave er å implementere logikk og grafikk som gjør applikasjonen til et fullverdig spill.

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 kan en oppgave se 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 løsningen din bør du ende opp med følgende:

```
// 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 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 kodeskjelettet, 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 T11 er det ikke noen funksjonsdeklarasjon eller funksjonsdefinisjon mellom BEGIN- og END-kommentarene, så her må du skrive all kode selv, og det er viktig at du skriver all koden din mellom BEGIN- og END-kommentarene for å få uttelling for oppgaven.

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.

Oppgavene

Legg frem kortene (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: Klassedeklarasjoner for CardColor, Suit og Card

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

Card-klassen (Figur 3) har en medlemsvariabel `flipped` som sier hvorvidt et kort ligger med bilesiden opp (`flipped = false`) eller ned (`flipped = true`). Medlemsfunksjonen `is_flipped` blir brukt i koden til å hente ut verdien til `flipped`, men for øyeblikket returnerer `is_flipped` alltid `false`, så alle kortene ligger med bilesiden opp på skjermen. Endre funksjonen `is_flipped` så den returnerer verdien til `flipped`.

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

Når oppgave T1 er gjort vil du kunne se at noen av kortene ligger med bildesiden ned, mens andre kort ligger med bildesiden opp.

2. (10 points) T2: Rødt eller svart?

Card-klassen (Figur 3) har en 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 et kort blir beskrevet av medlemsvariabelen `Card::suit`, som er en instans av enumklassen `Suit`. Fargen til et kort er enten `CardColor::BLACK` eller `CardColor::RED`. For øyeblikket returnerer `get_color` alltid `CardColor::RED`, så alle kortene er røde på skjermen.

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

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

Når oppgave T2 er gjort vil du kunne se både røde og svarte jokere.

3. (15 points) T3: Flytt flere kort samtidig

Knappene merket med `Select` velger bare det fremste kortet i bunken, men i kabal er det mulig å flytte flere kort samtidig fra en bunke til en annen. Funksjonaliteten for å plukke opp flere kort samtidig er stort sett allerede implementert, men én av funksjonene må endres. Funksjonen `inside_box` i `Tasks.cpp` skal returnere `true` dersom et gitt punkt ligger på innsiden av et gitt rektangel, men for øyeblikket returnerer den alltid `false`.

Funksjonen `inside_box` tar inn to parametere: et punkt `point` og en instans av `Rectangle` kalt `rectangle`. `Rectangle`-strukturen ser slik ut:

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

Funksjonen skal avgjøre hvorvidt `point` ligger på innsiden av `rectangle`. En illustrasjon av problemstillingen finner du i Figur 4.

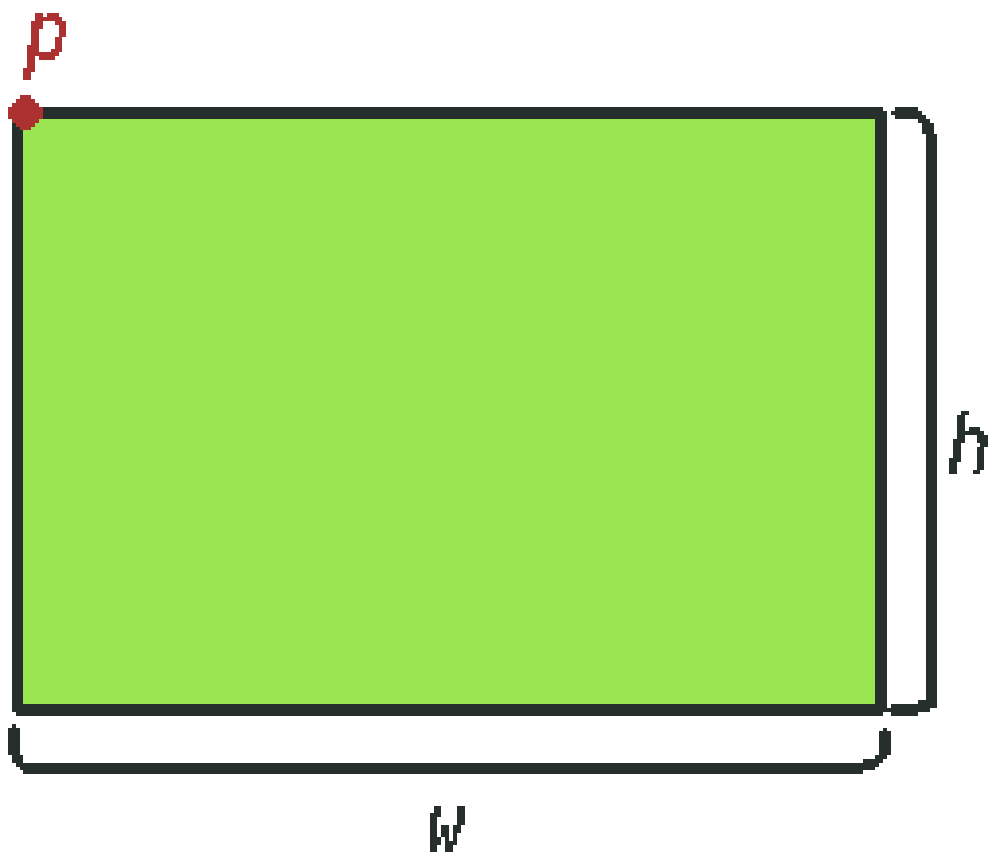
Endre funksjonen `inside_box` i `Tasks.cpp`.

Gitt `point` og `rectangle`, returner `true` dersom:

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

der x_r og y_r er komponentene av `rectangle.top_left` og w og h er henholdsvis feltene `width` og `height` i `rectangle`.

Når oppgave T3 er gjort vil det være mulig å velge flere kort fra en bunke og flytte dem samtidig fra en bunke til en annen. Kortene man velger vil markeres i gult.



Figur 4: Illustrasjon av gyldig område inne i et rektangel.

4. (10 points) T4: Snu øverste kort

Når kortet som ligger øverst i en bunke med bildesiden opp flyttes til en annen bunke vil de resterende kortene i bunken ligge med bildesiden ned, eller så vil bunken være tom. Vi vil endre på dette slik at det øverste kortet i bunken alltid ligger med bildesiden opp. Når det øverste kortet i bunken flyttes til en annen bunke skal derfor kortet som nå blir liggende øverst i bunken snus med bildesiden opp gitt at bunken ikke er tom.

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

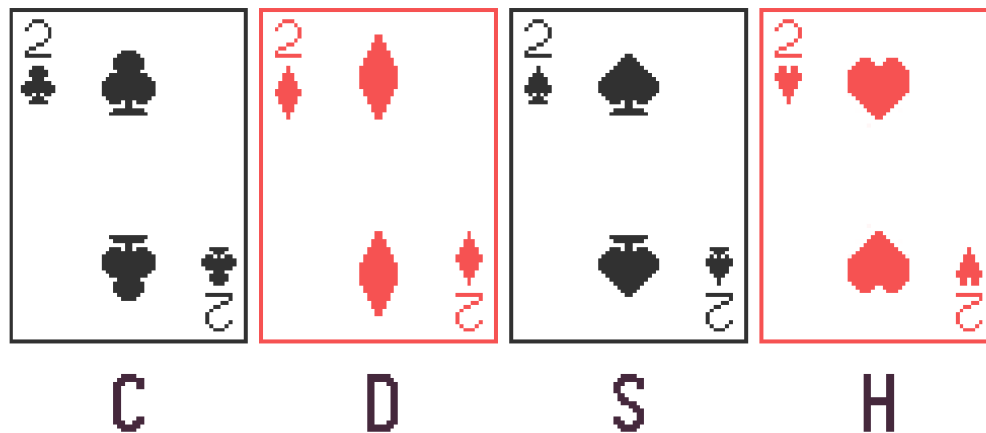
Implementer funksjonen `stack_set_flip` i `Tasks.cpp`.

- Sett `flipped` til `false` for det siste kortet i vektoren.
- Husk å ta høyde for at bunken kan være tom.

Hint: `Card`-klassen har en medlemsfunksjon `set_flipped` som kan brukes til å sette verdien til medlemsvariabelen `flipped`.

Når oppgave T4 er gjort skal du kunne se at det øverste kortet i en bunke alltid ligger med bildesiden opp (gitt at bunken ikke er tom).

Fiks kortene! (55 poeng)



Figur 5: Hver sort med sin tilhørende bokstav.

5. (15 points) T5: Kortene får nytt utseende - Kort til streng

Et kort kan identifiseres av en streng som inneholder 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 være mellom 1 og 13. `S` er sorten til kortet og vil være en av bokstavene 'S', 'C', 'H' eller 'D' for å indikere henholdsvis `Suit::SPADES` (spar), `Suit::CLUBS` (kløver), `Suit::HEARTS` (hjerter) og `Suit::DIAMONDS` (ruter). Dette kan du se i Figur 5. I tabellen under kan du se strengrepresentasjonen for noen utvalgte korttyper.

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 et kort ønsker vi å ha en funksjon som kan ta et kort og returnere strengrepresentasjonen beskrevet ovenfor.

Implementer `Card::get_identifiser` i `Card.cpp`.

Funksjonen tar en referanse til et Card-objekt og skal returnere en streng på formatet `VS`, der `V` er verdien til kortet og `S` er en bokstav som beskriver sorten til kortet.

- Verdien til kortet finner man ved å ta medlemsvariabelen `rank` og legge til 1.
- Sorten til kortet finner man i medlemsvariabelen `suit`.
- Sorten til kortet blir beskrevet slik:

Suit::SPADES → 'S'

Suit::CLUBS → 'C'

Suit::HEARTS → 'H'

Suit::DIAMONDS → 'D'

6. (15 points) T6: Kortene får nytt utseende – Kort til bildesti

Nå som vi har en streng som kan identifisere et kort, kan vi lage en sti til et bilde som representerer kortet. En titt inni `images`-mappen vil vise en rekke bilder med navn på formatet `VS.png`, der `VS` er stringrepresentasjoner av ulike kort slik det ble beskrevet i oppgave T5. Den fullstendige stien til et kort er dermed `images/VS.png`. Stien til selve bildemappen `images` kan du finne i en global variabel med navn `IMAGE_DIR` som inneholder verdien `"images/"` og er definert i `common.h`.

Vi ønsker nå å lage en funksjon som tar inn et kort og returnerer stien til det tilhørende bildet.

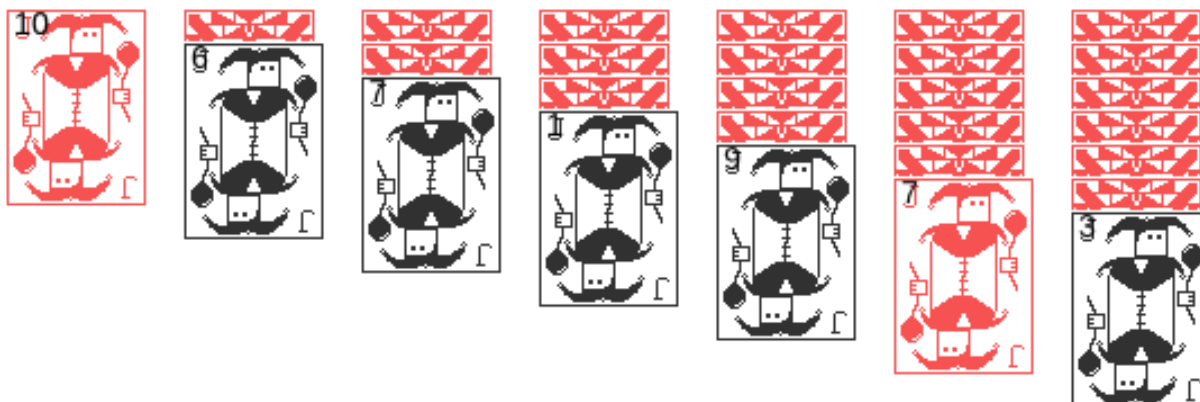
Eksempel: Dersom funksjonen tar inn et kort av typen kløver dronning skal funksjonen returnere strengen `"images/12C.png"`.

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

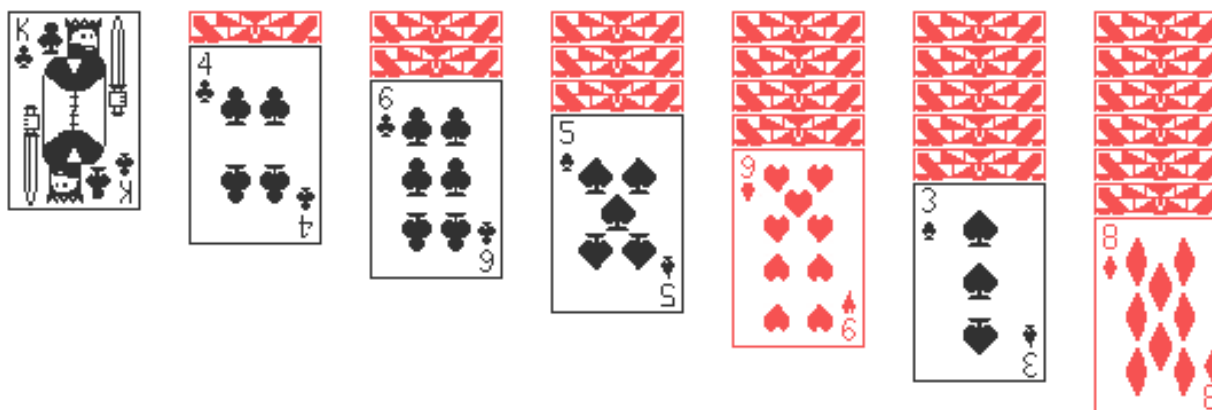
Funksjonen tar en referanse til et Card-objekt og skal returnere en streng som er stien til bildet som skal vises for denne korttypen. Strengen skal være sammensatt av

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

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



Figur 6: Eksempel på hvordan kortene ser ut før oppgave T5, T6 og T7 er gjort



Figur 7: Eksempel på hvordan kortene ser ut etter oppgave T5, T6 og T7 er gjort

7. (15 points) **T7: Kortene får nytt utseende – sjekk at bildet finnes**

Før vi kan laste inn bildene til kortene må vi dobbeltsjekke at det eksisterer et bilde for de ulike korttypene. Vi trenger derfor en funksjon som sjekker at stien til bildet er gyldig.

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

Når oppgave T5, T6 og T7 er gjort skal du kunne se at utseende til kortene har endret seg. Eksempler på hvordan kortene ser ut før og etter oppgavene er gjort kan du se i henholdsvis Figur 6 og Figur 7.

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

Så langt har alle kort kunne blitt flyttet fra bunke til bunke uten noen begrensninger, men vi må ikke glemme at det er regler i kabal.

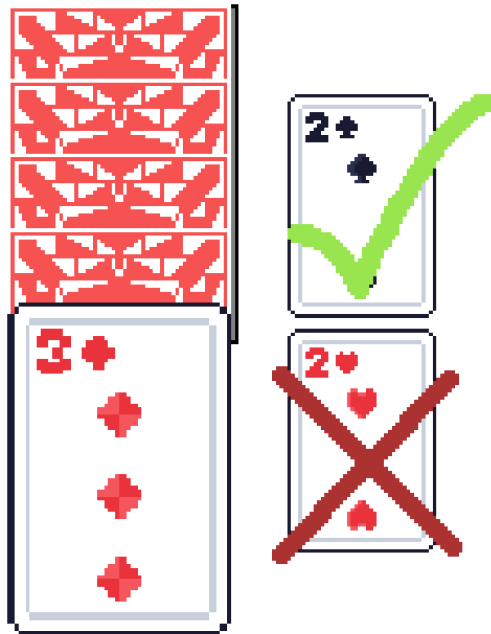
Vi skal nå sørge for at man må følge disse reglene for å legge et kort på toppen av en bunke:

- Hvis bunken er tom kan det **kun** legges på en konge.

- Hvis bunken ikke er tom, kan du legge på et kort som (1) har motsatt farge av den som ligger på toppen og (2) har en verdi som er én mindre enn kortet på toppen. Dette er illustrert i Figur 8.

Implementer funksjonen `can_push_to_tableau` i `Tasks.cpp`. Funksjonen tar et kort (`card`) og en bunke med kort (`cards`) og skal returnere `true` dersom kortet kan legges på bunken og `false` ellers.

Når oppgave T8 er gjort skal det ikke være mulig å flytte kortene med mindre man følger reglene.



Figur 8: Gyldige og ugyldige kort. .

Inn/ut datahåndtering og operatoroverlasting (75 poeng)

9. (20 points) T9: Gyldige filutvidelser

For øyeblikket er bildene som brukes for kortene hardkodet og bestemt på forhånd. Nå ønsker vi å utvide funksjonaliteten til spillet så man kan velge mellom flere ulike bilder for et kort. Vi ønsker derimot ikke å støtte alle bilder, så vi trenger en funksjon som sjekker at bildet man prøver å bruke har en filutvidelse som er støttet.

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

- Funksjonen tar inn et parameter `path` som er en fullstendig sti til en bildefil.
- Funksjonen skal returnere `true` dersom filutvidelsen til filstien `path` er `".png"`, `".jpg"` eller `".bmp"`, og `false` ellers.

Hint: Siden noen filer kan ha filutvidelser med store bokstaver kan det lønne seg å omgjøre filutvidelsen slik at den har små bokstaver. Dette kan gjøres med funksjonen `extension_to_lower(extension)`.

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

Det siste vi må gjøre for å kunne velge mellom ulike bilder for et kort, er å lese inn filen som beskriver hvilket bilde som skal brukes for hvert kort. Vi har laget en fil som beskriver kort og tilhørende bilder (se `images/alternative/manifesto.txt`). Denne filen følger oppsettet i Figur 9.

Dersom fila lar seg åpne skal vi behandle fila linje for linje ved å lese de tre feltene i hver linje (se Figur 9), bearbeide hver linje, og sette inn bildet som linja beskriver i bildeatlas.

Implementer `load_image_atlas` i `ImageAtlas.cpp`.

- Åpne filen som oppgis i parameteren `path`.
- Hvis filen ikke lar seg åpne, utløs et unntak av typen `std::runtime_error`.
- Behandle filen linje for linje (se eksempelet nedenfor):
 - Les inn feltene `RANK`, `SUIT` og `PATH`.
 - Dann kortet sin nøkkel (`key`) ved å slå sammen en streng som består av `RANK + 1` og `SUIT`.
 - Dann bildet sin fulle sti (`fullPath`) ved å danne strengen `IMAGE_DIR + PATH`.
 - Legg til bildet i atlaset ved å kalle `atlas.addImage(key, fullPath)`.
- Lukk filen når alle linjene har blitt lest.

Eksempel:

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

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

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

I oppgave T5 implementerte du en funksjon som returnerer en strengrepresentasjon av et Card-objekt. Vi vil nå overlaste innsettingsoperatoren (`operator<<`) til en `std::ostream` for Card-klassen slik at strengrepresentasjonen til kortet skrives når man skriver et Card-objekt til en `std::ostream`.

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

Når oppgave T11 er gjort skal det være mulig å skrive et kort til en standard output-strøm:

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

Merk: Oppgaven ber deg om å skrive hele overlastingen. Husk å skrive svaret ditt innenfor `BEGIN-` og `END-`kommentarene til oppgave T11 for å få uttelling for oppgaven.

Image Atlas filstruktur

Et atlas lagres med følgende filformat:

- Hver linje har tre felt: RANK, SUIT, og PATH
- De tre linjene er separert av en tabulator (\t)

Eksempel: 0 S card000.png representerer et 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

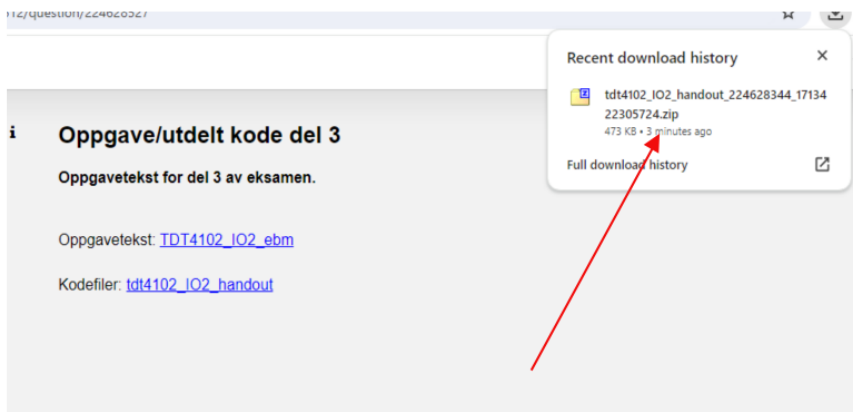
Oppgavetekst for del 3 av eksamen.

Oppgavetekst: [TDT4102_IO2_ebm](#)

Kodefiler: [tdt4102_IO2_handout](#)



(a) Steg 1




i Oppgave/utdelt kode del 3

Oppgavetekst for del 3 av eksamen.

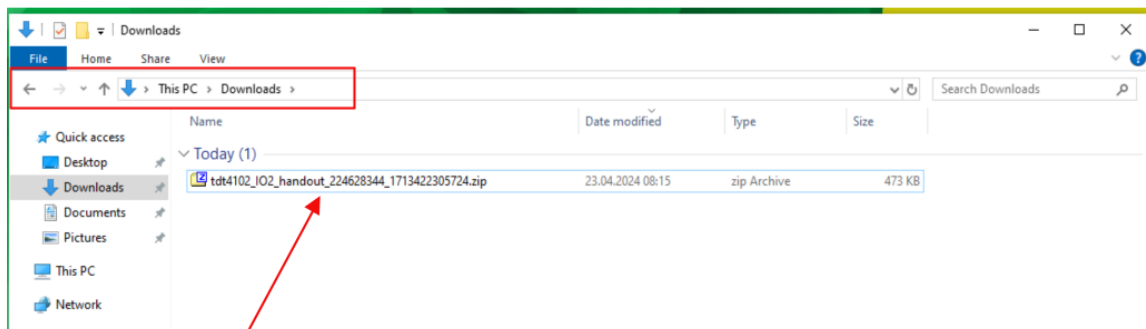
Oppgavetekst: [TDT4102_IO2_ebm](#)

Kodefiler: [tdt4102_IO2_handout](#)

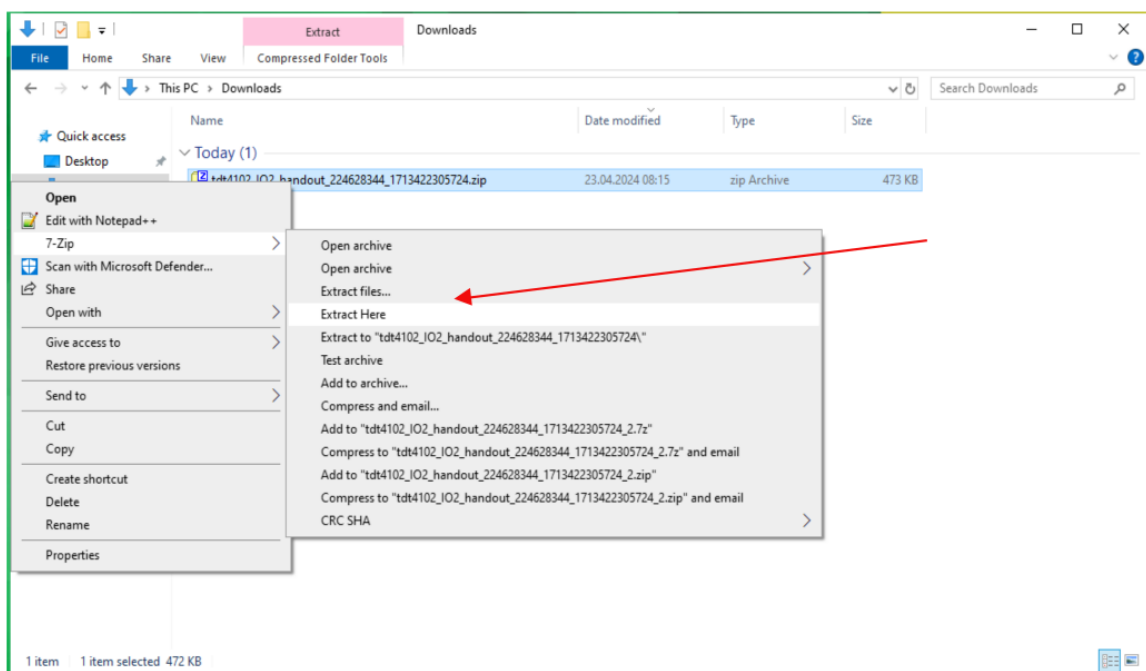
Recent download history

 tdt4102_IO2_handout_224628344_17134_22305724.zip
473 KB • 3 minutes ago

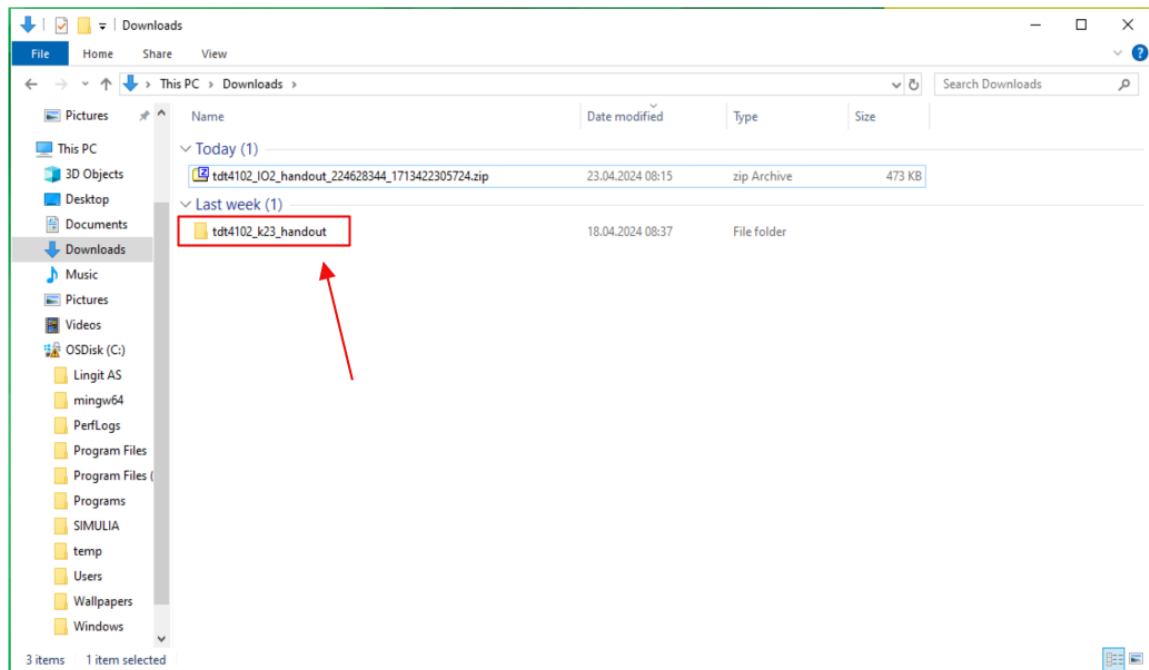
Full download history



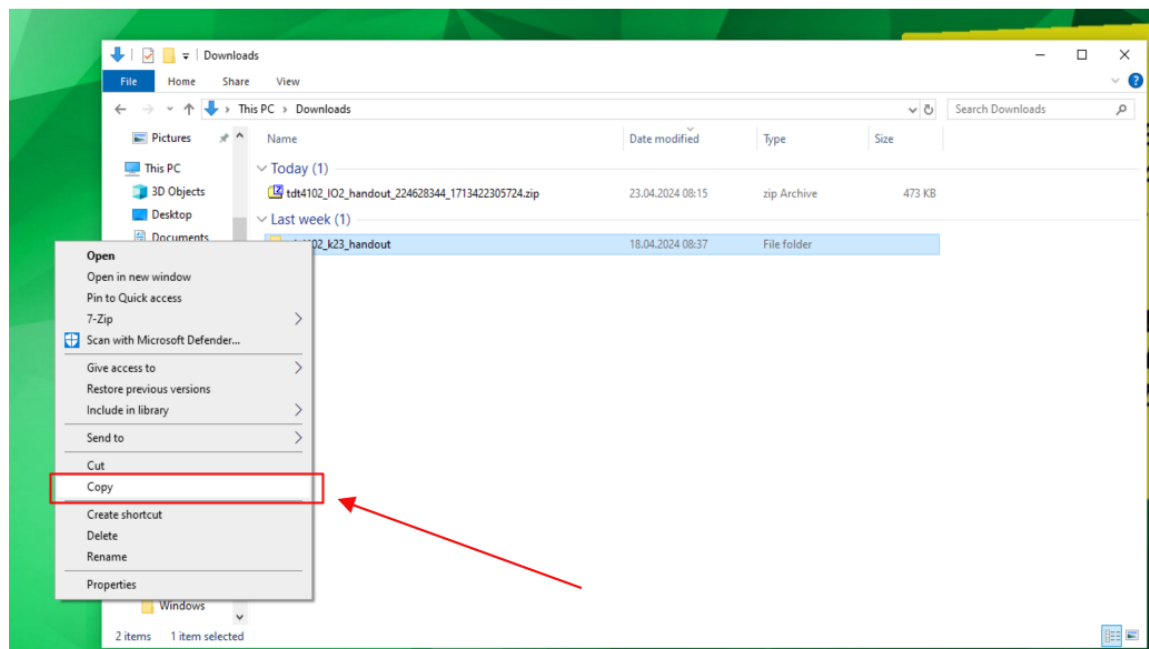
(c) Steg 2

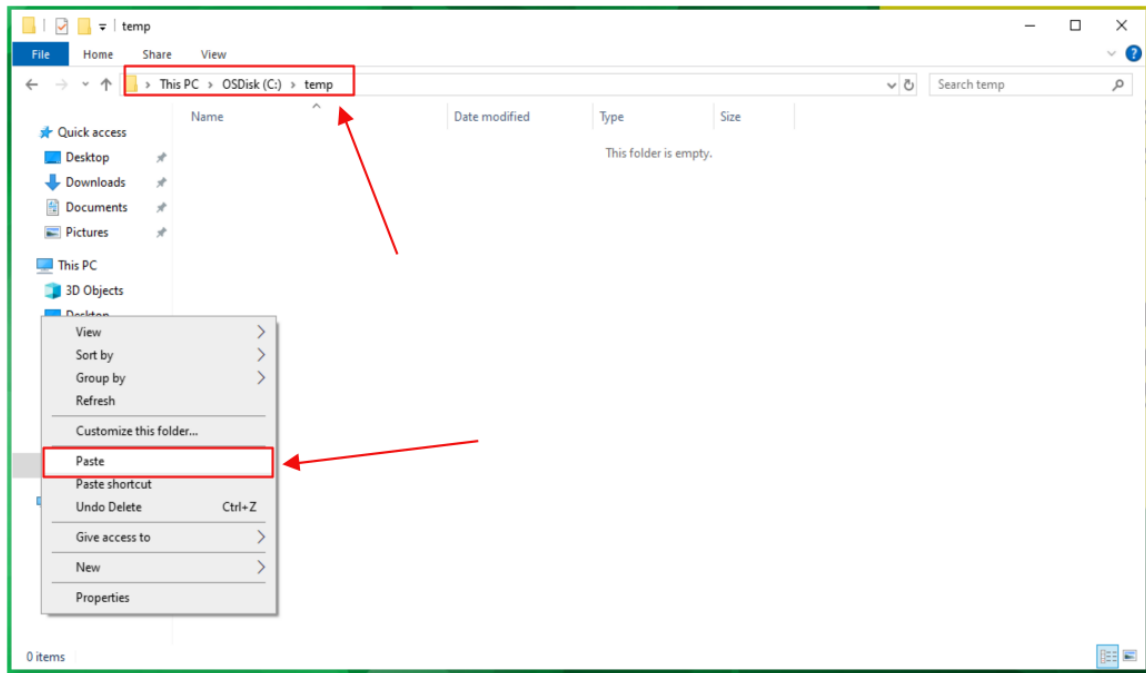


(d) Steg 3

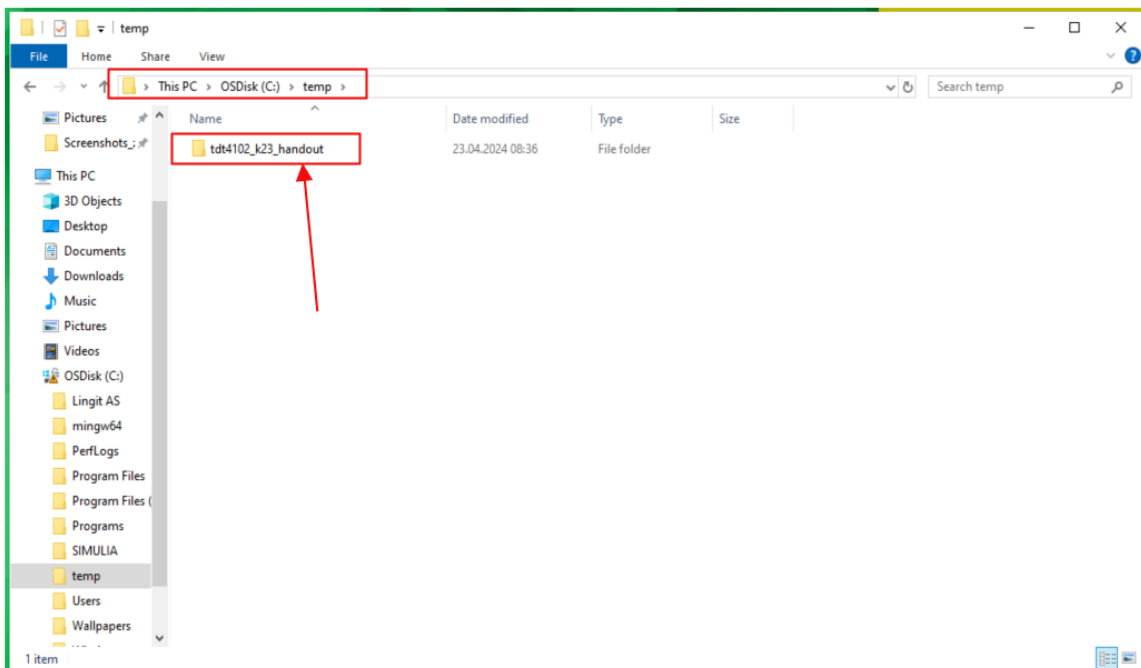


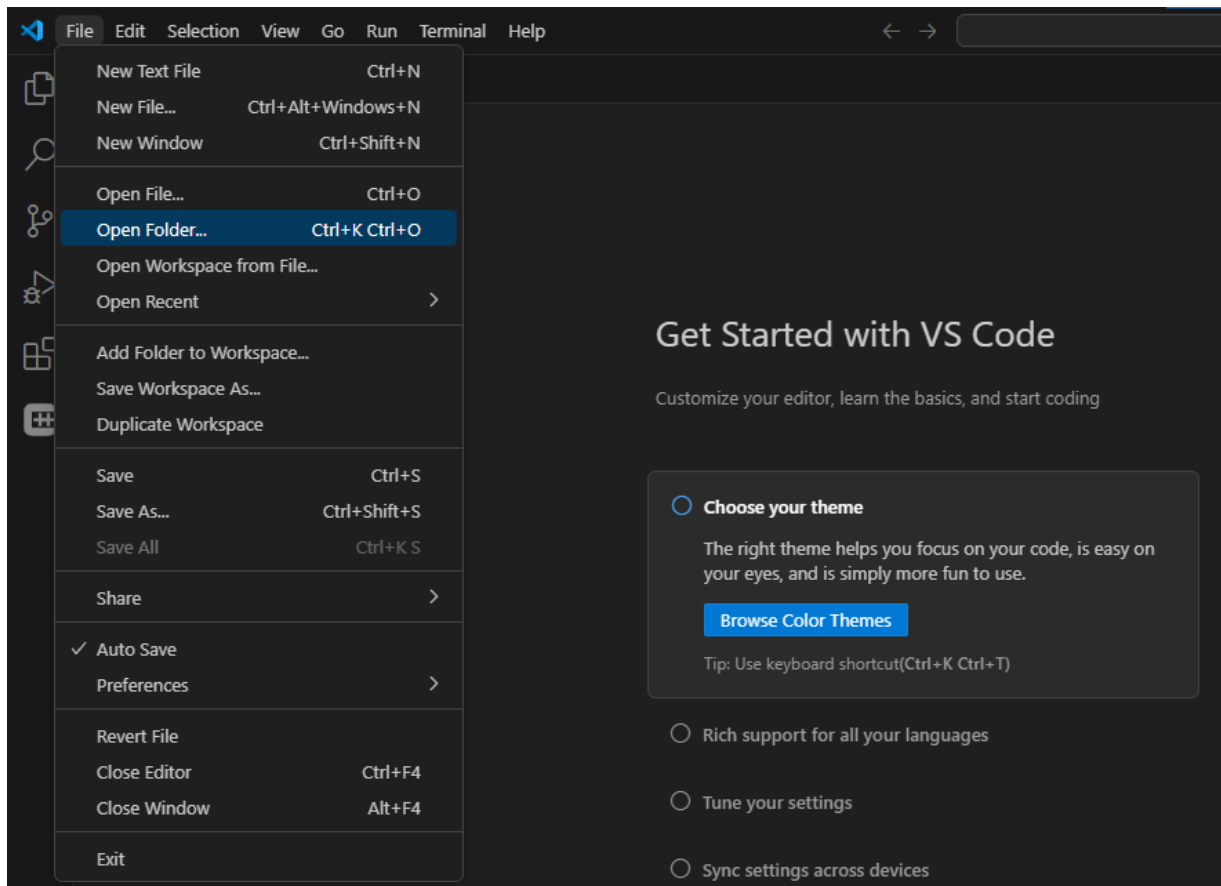
(e) Steg 3



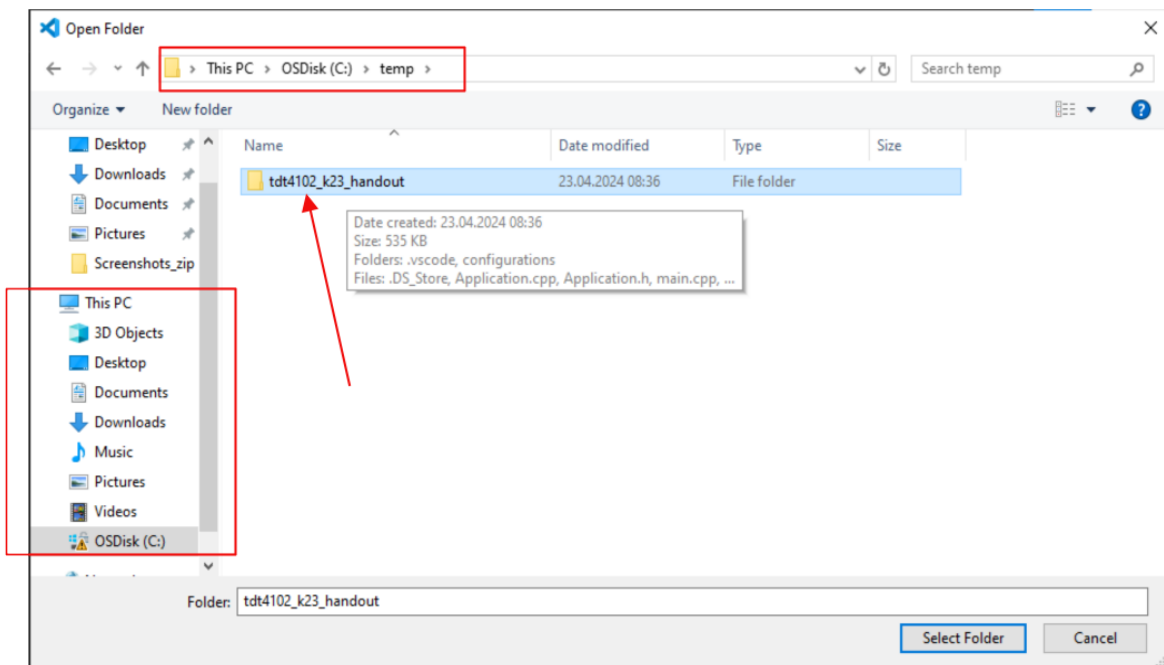


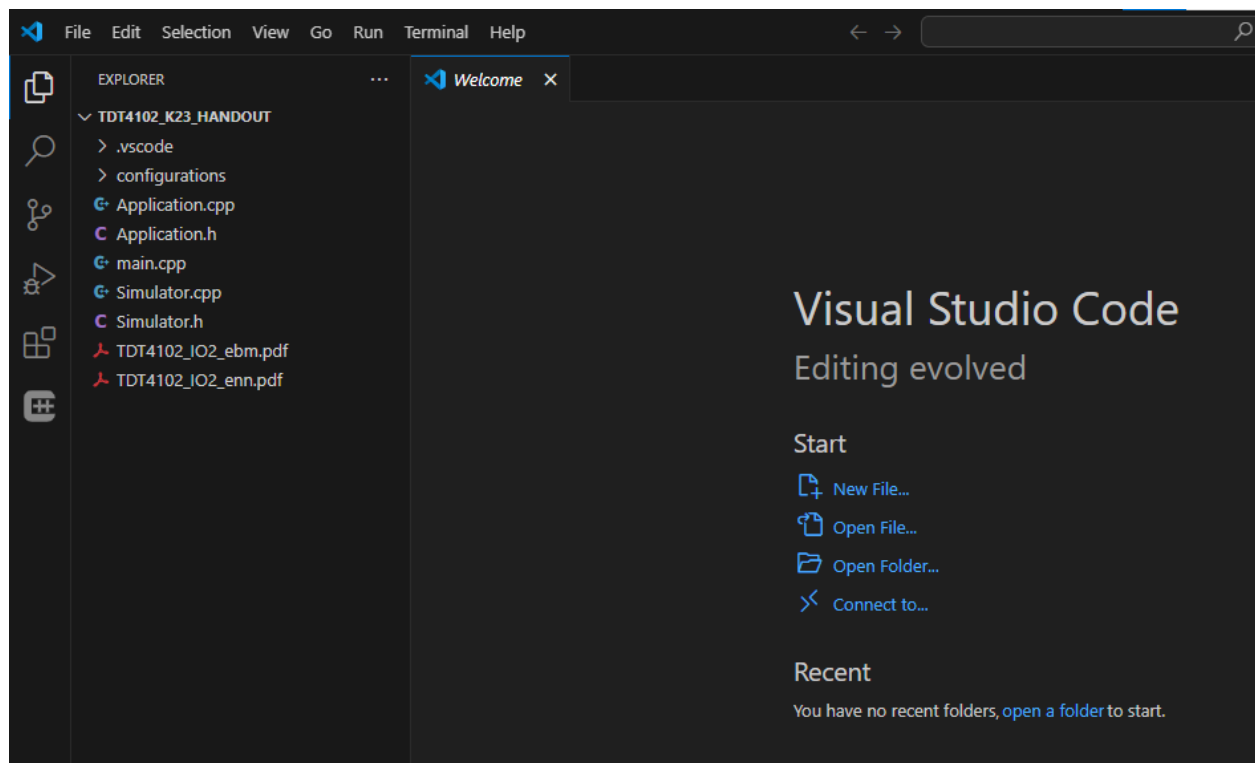
(g) Steg 4





(i) Steg 5





(k) Steg 5

Figur 10: Nedlasting og oppsett av den utdelte koden.