

i Front

Institutt for datateknologi og informatikk

Eksamen TDT4102 - Prosedyre- og objektorientert programmering

Eksamensdato : Tirsdag 10. august 2021.

Eksamenstid (fra-til) : 0900-1300 + 30 min til filopplasting

Hjelpemiddelkode/Tillatte hjelpemiddel: A / Alle skriftlige/trykte hjelpemiddel tillatt

Faglig kontakt under eksamen : Rune Sætre, Truls Asheim og Lasse A. Eggen

Tlf : 452 18 103 (Rune), +45 2282 5830 (Truls), 913 69 975 (Lasse)

Email: tdt4102-fagans@idi.ntnu.no

Teknisk hjelp under eksamen: NTNU Orakel. Tlf : 73 59 16 00

ANNEN INFORMASJON

Gjør dine egne antagelser og presiser i besvarelsen hvilke forutsetninger du har lagt til grunn i tolking/avgrensning av oppgaven. Faglig kontaktperson skal kun kontaktes dersom det er direkte feil eller mangler i oppgavesettet.

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 alle svarene dine på det spørsmålet er feil.

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. Alle besvarelser blir kontrollert for plagiat. Du kan lese mer om juks og plagiering på eksamen her: <https://innsida.ntnu.no/wiki/-/wiki/Norsk/Juks+påeksamen>

Kildehenvisninger:

Selv om "Alle hjelpemiddel er tillatt", 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. Det vil i tillegg bli sendt SMS til alle kandidater for å sikre at ingen går glipp av viktig informasjon. Ha mobiltelefonen din tilgjengelig.

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 eksamen. Du vil få F på eksamen hvis du svarer blankt på en av de tre delene.

Slik svarer du på oppgavene:

Alle oppgaver som ikke 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, da dette kan medføre at formatering og elementer (bilder, tabeller etc.) vil kunne gå tapt.

Filoplasting:

Når du jobber i andre programmer fordi hele eller 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 eksamenstida går ut.

Det framgår av filopplastingsoppgaven hvilket filformat som er tillatt (**zip**).

Det er lagt til **30 minutter** til ordinær eksamenstid for eventuell digitalisering av håndtegnninger og opplasting av filer. Tilleggstida er forbeholdt innlevering og inngår i gjenstående eksamenstid som vises øverst til venstre på skjermen.

NB! Det er ditt eget ansvar å påse at du laster opp riktige og intakte 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.

Pass på at det ikke finnes noe forfatterinformasjon i filen(e) du skal levere.

I siste del av eksamen skal du bruke samme program som du satte opp i øving 0. Du må også vite hvordan du laster ned, pakker ut, og setter opp mapper fra en zip-fil som VS Code (eller tilsvarende) C++-prosjekter på maskinen din. Til slutt må du være i stand til pakke alle slike mapper sammen i en zip-fil igjen for å levere det du har kodet, innen tidsfristen, for å kunne bestå denne eksamen.

Eksamen - step by step

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

1. Les nøye igjennom disse introduksjons-sidene
2. **Last ned** medfølgende zip-fil med en gang eksamen starter. I zip-filen finner du .cpp- og .h-filer samt oppgaveteksten til del 3.
3. Les **forklarlingen** på problemet gitt i begynnelsen av hver seksjon.
4. Du kan bruke VS Code (eller et hvilket som helst annet utviklingsmiljø du foretrekker) til å åpne og jobbe med den oppgitte koden (som i Øving 0). All kode skal være C++.
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 huske å lagre.
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.
 - Last opp all den komplette koden som en .zip-fil. Ikke endre den opprinnelige mappestrukturen. For å få bestått på denne eksamen er det **HELT AVGJØRENDE AT DU LASTER OPP ZIP-FILEN**. Etter eksamensslutt (13:00) har du 30 minutter til rådighet til dette. Vi anbefaler likevel at du laster opp zip-filen minst en gang underveis i eksamenstiden.
 - Det er mulig å oppdatere både enkelt-svarene og filopplastningen flere ganger, i tilfelle du retter på noe etter første innlevering.
 - Prøv å last opp filen en gang midt i eksamenstiden, for å se at du klarer det, og hvor lang tid du bruker på det.
8. Husk at funksjonene du lager i en deloppgave ofte er ment å skulle brukes i andre deloppgaver. Selv om du står helt 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 riktig implementert.
9. Før manuell sensureringen av eksamen din, vil vi foreta automatisk testing og plagiatkontroll av all koden du har levert. Basert på resultatene kan det hende vi ber deg om en forklaring, og dette kan påvirke eksamensresultatet ditt. Du må huske å legge til kommentarlinjer i koden din, siden det også kan hjelpe oss å forstå koden din bedre.

Nedlastning av start-fil (zip)

- Last ned og lagre .zip-filen (lenken er på Inspira). Gjør dette med en gang eksamen starter... i tilfelle noe er galt, eller internett detter ut av og til underveis.
- Husk å lagre den på et sted på datamaskinen du husker og kan finne igjen.
- Pakk ut (unzip) filen.
- 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 øving 0 på begynnelsen av semesteret.
- Filene vi leverer ut kompilerer til et kjørende program, men du må selv skrive resten av koden for alle oppgavene og prøve å få hver programmet til å kjøre som et eget prosjekt. 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 alle del-spørsmål, må du laste opp alle kodefilene dine, etter at de på nytt er pakket sammen til en lignende .zip-fil (**ikke 7z, rar eller andre**) som den du begynte med. Ikke endre noe på de opprinnelige mappe/fil-navnene før du zipper filen og laster den opp til Inspira igjen. Last gjerne opp på nytt hver time!

Automatisk innlevering

Besvarelsen din leveres automatisk når eksamenstida er ute og prøven stenger, forutsatt at minst én oppgave er besvart. 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.

Trekk/avbrutt eksamen

Blir du syk under eksamen, eller av andre grunner ønsker å levere blankt/avbryte eksamen, gå til "hamburgermenyen" i øvre høyre hjørne og velg «Lever blankt». Dette kan ikke angres selv om prøven fremdeles er åpen.

Tilgang til besvarelse

Du finner besvarelsen din i Arkiv etter at sluttida for eksamen er passert.

☑ Rules and consent

REGLER OG SAMTYKKER

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

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

Under øvingen:

Jeg skal IKKE motta hjelp fra andre.

☐ Aksepter

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

☐ Aksepter

Jeg skal 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 øvingen kan bli underkjent uavhengig av hvor korrekt svarene mine er, hvis jeg ikke følger reglene og/eller IKKE aksepterer disse utsagnene.

☐ Aksepter

1.1 C-arrays vs vectors

Hvilke av de følgende utsagnene om array og vektor er sanne?

Velg ett eller flere alternativer

- ☐ Å få tilgang til elementene i en array tar alltid kortere tid enn i en tilsvarende vector.
- ☐ Array bruker aldri mer minne enn en identisk vector.
- ☐ Array kan ikke deklarereres dynamisk (på heap) mens vector kan det.
- ☐ Man kan få tilgang til elementene både i array og vector gjennom iteratorer.
- ☐ En vector holder styr på størrelsen mens en array gjør det ikke.

Maks poeng: 10

1.2 Function

Hva blir returverdien til f(p,p) hvis "int p=3;" i følgende to kodesegmenter:

a)

```
int f(int x, int c){  
    c=c-1;  
    if (c==0){  
        return 1;  
    }  
    x++;  
    return f(x,c)*x;  
}
```

Velg ett eller flere alternativer

- ☐ 20
- ☐ Kompileringsfeil
- ☐ 25
- ☐ 60

b)

```
int f(int &x, int c){  
    c=c-1;  
    if (c==0){  
        return 1;  
    }  
    x++;  
    return f(x,c)*x;  
}
```

Velg ett eller flere alternativer

- ☐ 25
- ☐ 20
- ☐ 60
- ☐ Kompileringsfeil

1.3 element-wise square

Følgende kodesegment prøver å skrive ut elementvise kvadrattall i en matrise, men mislykkes. Hvilken av de følgende kodelinjer inneholder feil? Mulige typer feil er logiske feil, typefeil, syntaksfeil og kjøretidsfeil.

Element wise square of $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 9 & 16 \end{bmatrix}$

```

1  #include <iostream>
2  using namespace std;
3  int main(void){
4      const int width = 3;
5      const int height = 3;
6      int A[width][height] = {
7          {1,2,3},
8          {1,1,1},
9          {1,0,0}}
10     for(int i = 0; i < width; i++){
11         for(int j = 0; j < height; j++){
12             A[i][j] = A[i][j] + A[i][j];
13         }
14     }
15     for(int i = 0; i < width; i++){
16         for(int j = 0; j < height; ){
17             cout << A[i][j] <<" ";
18         }
19         cout << endl;
20     }
21 }
```


Velg ett eller flere alternativer☐ 7☐ 8☐ 9☐ 10☐ 11☐ 12☐ 13☐ 14☐ 15☐ 16☐ 17

Maks poeng: 10

1.4 Virtual Function properties

Velg alle de sanne utsagnene om virtuelle funksjoner:

Velg ett eller flere alternativer

- ☐ En vanlig virtuell funksjon trenger ikke å defineres i sub-klassen.
- ☐ Kompilatoren bestemmer hvilken virtuell funksjon som skal kalles basert på objektet som pekes på av baseklassepekeren.
- ☐ Virtuelle funksjoner må tilhøre en klasse.
- ☐ Virtuelle funksjoner må ha 'virtual' før deklarasjonen av funksjonen.
- ☐ En virtuell funksjon kan være 'friend'-funksjon til en annen klasse.
- ☐ 'friend function' og 'virtual function' har samme funksjonalitet.
- ☐ Virtuelle funksjoner bør merkes med 'override' i subklassene.

Maks poeng: 10

1.5 fibonacci recursive equivalent

```
int fib(int n){ //n>=0
    if (n<=1){
        return n;
    }
    return fib(n-1)+fib(n-2);
}
```

Hvilke av fire følgende alternativer er semantisk ekvivalent med fib over?

Which of the four following options are semantically equivalent to fib above?

Velg ett eller flere alternativer

☐

```
int fib(int n){
    int f[n+1],i;
    f[0]=0,f[1]=1;
    for(int i=2;i<=n;i++){
        f[i]=f[i-1]+f[i-2];
    }
    return f[n];
}
```

☐

```
int fib(int n){
    int a=0,b=1,c,i;
    if (n==0){
        return a;
    }
    for(int i=2;i<=n;i++){
        c=a+b;
        a=b;
        b=c;
    }
    return b;
}
```

☐

```
int fib(int n){
    int a=0,b=1,i;
    if (n==0){
        return a;
    }
    for(int i=2;i<=n;i++){
        n=a+b;
        a=b;
        b=n;
    }
    return b;
}
```

☐

```
int fib(int n){
    if (n<=1){
        return n;
    }
    return fib(n+1)+fib(n-1);
}
```

Maks poeng: 10

2.1 Element wise sum

Se på følgende kodesegment. Det brukes rekursjon for å skrive ut den elementvise summen av de to vektorene. Skriv den ikke-rekursive versjonen av koden under:

```
void print_elementwise_sum(vector<int> obj1, vector<int> obj2){  
    if (obj1.size() != obj2.size()){  
        cout<<"Dimention mismatch, can't add";  
    }  
    else if(obj1.size()>0){  
        cout << obj1.back() + obj2.back() << endl;  
        obj1.pop_back();  
        obj2.pop_back();  
        return print_elementwise_sum(obj1,obj2);  
    }  
    return;  
}
```

Skriv ditt svar her

1	
---	--

Maks poeng: 10

2.2 Inheritance outputs

Hva blir skrevet ut av følgende to kodesegmenter? Bruk a) og b) svar-boksene under:

a)

```
#include<iostream>
using namespace std;
class alpha {
public:
    alpha(){
        cout<<"alpha's constructor called\n";
    }
    ~alpha(){
        cout<<"alpha's destructor called\n";
    }
};
class beta{
public:
    beta(){
        cout<<"beta's constructor called\n";
    }
    ~beta(){
        cout<<"beta's destructor called\n";
    }
};
class Derived: public alpha, beta{
public:
    Derived(){
        cout<<"Derived's constructor called\n";
    }
    ~Derived(){
        cout<<"Derived's destructor called\n";
    }
};

int main()
{
    Derived d;
    return 0;
}
```

Skriv ditt svar her

b)

```
1  #include<iostream>
2  using namespace std;
3
4  class Charlie {
5  public:
6      virtual string print() const{
7          return "printed from Charlie\n";
8      }
9  };
10
11 class Delta: public Charlie{
12 public:
13     virtual string print() const override{
14         return "printed from Delta\n";
15     }
16 };
17
18 void print1( const Charlie obj ){
19     cout << obj.print();
20 }
21 void print2( const Charlie& objr ){
22     cout << objr.print();
23 }
24
25 int main(){ //2_2_2
26     Charlie C;
27     Delta D;
28     print1(C);
29     print1(D);
30     print2(C);
31     print2(D);
32     return 0;
33 }
```

Skriv ditt svar her

Maks poeng: 10

2.3 file handling

Se på følgende kodesegment:

```
1  #include<iostream>
2  #include<fstream>
3  #include<string.h>
4  int main(){
5      ifstream fin;
6      fin.open("FIRST.txt");
7      ofstream fout;
8      fout.open("Second.txt");
9      char ch;
10     while(!fin.eof()){
11         fin.getch(ch);
12         if(ch>='a'&&ch<='z')
13             {ch=toupper(ch);}
14         fout<<ch;
15     }
16     return 0;
17 }
```

a) La oss anta at FIRST.txt bare inneholder engelske bokstaver (a-z, A-Z). Kan du forklare hva koden gjør da?

Skriv ditt svar her

b) Selv om koden vil kjøre (de fleste gangene), er den ikke perfekt. Hva kan du legge til for å gjøre filhåndteringen mer robust (kort svar)?

Skriv ditt svar her

c) Hvis du fjerner linje 12 (bare), hva blir da skrevet i SECOND.txt hvis FIRST.txt inneholder "Prosedyre- og objektorientert programmering!" (uten doble anførselstegn).

Skriv ditt svar her

Maks poeng: 10

2.4 constructor implementation

Skriv inn definisjonene for begge (de to ikke-tomme) konstruktørene for klassen Result, gitt dette kodesegmentet. Skriv svarene som om de står utenfor scopet til klassen.

```
class DayTime{
    int mins, hours, secs;
public:
    DayTime(int mins = 0, int hours = 0, int secs = 0):mins(mins), hours(hours), secs(secs) {}
    void print_time(){
        cout<< hours << "H" << mins << "mins" << secs << "secs" << "\n";
    }
};

class Result{
    int result_id;
    DayTime time;
public:
    Result();
    Result(int id, const DayTime& z);
    Result(int id, int mins, int hours, int secs);
    void print_result(){
        cout << result_id << " ";
        time.print_time();
    }
};
```

Skriv ditt svar her

Maks poeng: 10

2.5 string

Se på følgende kodesegment:

```
2  ✓ void f(char* str){
3      int c[256] = {0};
4  ✓  for(unsigned int i = 0; i < strlen(str); i++){
5      |      c[(int)str[i]]++;
6      |  }
7  ✓  for(unsigned int i = 0; i < strlen(str); i++){
8  ✓      if (c[(int)str[i]] != 0){
9      |          cout << str[i] << " " << c[(int)str[i]] << endl;
10     |          c[(int)str[i]] = 0;
11     |      }
12     |  }
13 }
```

a) 3 poeng: Navnene på variablene er endret slik at koden er vanskelig å forstå. Forklar kort hva funksjonen f gjør?

b) 7 poeng: Endre funksjonen f slik at den bare skriver ut bokstaven/en bokstav med flest forekomster.

Skriv ditt svar her, både a) og b)

1	
---	--

i Informasjon Zip

VIKTIG:

Zip-filen du skal laste ned inneholder kompilerbare (og kjørbare) .cpp- og .h-filer med forhåndskodede deler og en full beskrivelse av oppgavene i del 3 som en PDF-fil. Etter å ha lastet ned zip-filen står du fritt til å bruke et utviklingsmiljø etter eget valg (for eksempel VS Code) for å jobbe med oppgavene.

For å få bestått på denne eksamen er det **HELT AVGJØRENDE AT DU LASTER OPP ZIP-FILEN**. Etter eksamensslutt (13:00) har du 30 minutter til rådighet til dette

De korte svarene i Inspera (del 1 og 2) lagres automatisk hvert 15. sekund, helt til eksamenstiden er slutt. Og zip-filen (i del 3) kan også endres / lastes opp flere ganger. Så hvis du vil gjøre noen endringer etter at du har lastet opp filen, kan du bare endre koden, zippe den sammen på nytt, og laste opp filen igjen. Når prøvetiden er over, vil siste versjon av alt du har skrevet eller lastet opp i Inspera automatisk bli sendt inn som ditt gjeldende svar, så **sørg for at du laster opp minst en gang halvveis, og en gang før tiden går ut.**

På neste side kan du laste ned .zip-filen, og senere laste opp den nye .zip-filen med din egen kode inkludert. Husk at PDF-dokumentet med alle oppgavene er inkludert i zip-filen du laster ned.

3.1 Nedlasting/opplasting av kode

LAST NED

[Trykk her for å laste ned utdelt kode](#)

LAST OPP

Last opp all den komplette koden som en .zip-fil. Ikke endre den opprinnelige mappestrukturen. For å få bestå prøven er det **HELT AVGJØRENDE AT DU LASTER OPP DEN NYE ZIP-FILEN DIN KORREKT**, minst en gang i løpet av de 4 timene til rådighet.

Det er mulig å oppdatere både enkeltsvarene og filopplastningen **flere ganger**, i tilfelle du retter på noe etter første innlevering.


Prøv å last opp en oppdatert zip-fil minst en gang ekstra, midt i prøvetiden, for å se at du klarer det, og for å finne ut hvor lang tid du bruker på det.

Last opp zip-filen med besvarelsen din her. Alt i én zip-fil.



Last opp filen her. Maks én fil.

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

 Velg fil for opplasting

Maks poeng: 200

Question 11

Attached



Part III: Spillet 1024

Maksimal score for del 3 er 200 poeng.

Introduksjon til spillet 1024

Spillet 1024 ble veldig populært i flere utgaver og kloner i år 2014. Det går ut på at man har et spillbrett med 4x4 ruter, der to av rutene til å begynne med har verdien 2. Oppgaven til spilleren er å få en av rutene i brettet til å inneholde verdien 1024. En spiller kan utføre fire forskjellige trekk, skyve alle brikker oppover, til høyre, nedover eller til venstre. Når brikkene skyves til en av sidene vil to like brikker som ligger langs aksen det flyttes legges sammen og bli en ny brikke med summen av de opprinnelige. Hvis du ikke er kjent med spillet kan du prøve en utgave av det på <https://poweroftwo.nemoidstudio.com/1024>.

Vi har allerede implementert en grafisk representasjon, men spillets logikk mangler. I denne eksamensoppgaven skal du implementere stort sett all logikk for spillet over flere deloppgaver.

Hvordan besvare del 3?

Alle oppgavene i del 3 er satt opp slik at de skal besvares i filen `Game.cpp`. Hver oppgave har en tilhørende unik kode for å gjøre det lettere å finne frem til hvor i filen du skal skrive svaret. Koden er på formatet `<tegn><siffer> (TS)`, eksempelvis G1 og G2. I `Game.cpp` vil du for hver oppgave finne to kommentarer som definerer henholdsvis begynnelsen og slutten av koden du skal føre inn. Kommentarene er på formatet:

```
// BEGIN: TS og //END: TS.
```

Det er veldig viktig at alle svarene dine er skrevet mellom slike kommentar-par, for å støtte sensurmekanismen vår. Hvis det allerede er skrevet noen kode mellom BEGIN- og END-kommentarene i filene du har fått utdelt, så kan, og ofte bør, du erstatte den koden med din egen implementasjon.

For eksempel, for oppgave G1 ser du følgende kode i utdelte `Game.cpp`

```
1 void Game::new_game() {
2     // BEGIN: G1
3     // END: G1
4 }
```

Etter at du har implementert din løsning, bør du ende opp med følgende istedenfor

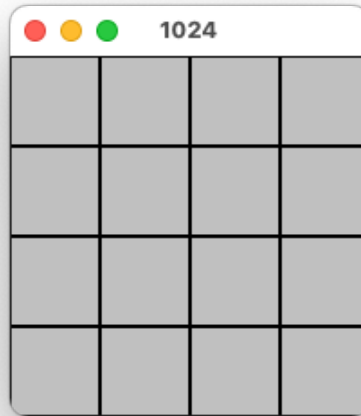
```
1 void Game::new_game() {
2     // BEGIN: G1
3
4     /* Your answer here. Code and // possibly explaining comments */
5
6     // END: G1
7 }
```

Merk at BEGIN- og END-kommentarene **IKKE skal fjernes**.

Til slutt, hvis du synes noen av oppgavene er uklare, oppgi hvordan du tolker dem og de antagelsene du må gjøre som kommentarer i den koden du sender inn.

I zip-filen finner du bl.a. `Game.h` og `Game.cpp`. Det er kun `Game.cpp` som skal redigeres for å komme i mål med oppgavene. Du kan se i headerfilene for å få en oversikt over deler av koden før du starter å løse oppgavene. `Game.h` inneholder klassedefinisjonen som tar for seg spillets logikk og er sammen med `Game.cpp` filene som er viktige i denne eksamensoppgaven. `GameWindow.h`, `Tile.h`, `settings.h` og `utilities.h` inneholder diverse deklarasjoner og definisjoner som brukes til den grafiske representasjonen - det er ikke nødvendig å sette seg inn i eller forstå koden som er knyttet til grafikk i denne eksamenens del 3.

Før du starter må du sjekke at den (umodifiserte) utdelte koden kjører uten problemer. Du skal se det samme vinduet som i figur 1. Når du har sjekket at alt fungerer som det skal er du klar til å starte programmering av svarene dine.



Figur 1: Utlevert kode uten endringer.

Game - Spillogikk (200 poeng)

Spillets logikk skal implementeres i klassen Game. Den skal kun inneholde spillets logikk og har ansvar for å korrekt flytte brikkene på brettet når spilleren velger å bevege dem opp, ned, til høyre eller venstre, samt starte spillet på nytt.

Spillbrettet er representert av 1D-vectoren `vector<int> board`, som inneholder `board_size * board_size` antall heltall. Du vil implementere en 2D-indeksering av denne vektoren for å gjøre det lettere å implementere logikken, men også tillate brukere av klassen å indeksere et rutenett.

Spilleren kan velge å skyve alle brikker til høyre, venstre, opp eller ned. Siden alle operasjoner i utgangspunktet er like, men gjøres i forskjellige retninger skal vi først implementere skyving av alle brikker til høyre og gjenbruke den metoden for å skyve brikkene i de andre retningene. Som med alle andre programmer er det mulig å implementere dette spillet på mange forskjellige måter og i denne oppgaven skal vi bl.a. bruke matriseoperasjoner til å manipulere spillbrettet så vi kan gjenbruke høyreskyvingen. Mer om dette når det er relevant i hver enkelt oppgave.

Vi har overlastet operator `<< (ostream&, const Game&)`, som du kan bruke for å skrive ut spillbrettet til terminalen når du måtte ønske. Merk at funksjonen ikke vil fungere før etter oppgave G4 siden funksjonene som henter verdiene fra spillbrettet mangler implementasjonsdetaljer. Du får også utdelt diverse funksjoner som kan brukes til å debugge logikken underveis, se oppgave G6 og tabell 1 for mer informasjon.

Oppgavene må ikke gjøres i en bestemt rekkefølge. F.eks. kan funksjonen du implementerer i av deloppgave G5 gjenbrukes i tidligere oppgaver hvis du ønsker det. Du står fritt til å bruke metoder definert hvor som helst i del 3 andre steder i del 3.

1. (10 points) G1: `Game::index` - Beregn 1D-index ut fra 2D-koordinater

Funksjonen skal returnere en verdi som tilsvarer 1D-indeksen beregnet fra 2D-koordinatene til elementet med koordinatene (x, y) i spillbrettet. F.eks. skal $(0, 0)$ gi 0 og $(3, 2)$ gi 11 når spillbrettet er 4×4 stort. Bruk gjerne medlemsvariabelen `board_size`, som inneholder størrelsen på spillbrettet langs en akse, for eksamensoppgaven er det 4 (spillbrettet er 4×4).

2. (10 points) G2: `int Game::at(int x, int y) const` - Les en verdi fra 2D-koordinat

3. (10 points) G3: `int& Game::at(int x, int y)` - Les en verdi fra 2D-koordinat

Funksjonene skal returnere verdien som befinner seg i spillbrettets posisjonen (x, y) . Merk at det er to medlemsfunksjoner som skal implementeres (G2 og G3), `int Game::at(int x, int y) const` og `int& Game::at(int x, int y)`. Spillets brikker er lagret i medlemsvariabelen `board`.

Hvis posisjonen i spillbrettet ikke eksisterer skal funksjonen kaste et passende unntak av typen `std::out_of_range`.

Vi implementerer to funksjoner her for å ha mulighet til å hente både referanser til verdier i spillbrettet og kopier av verdiene. Det er hensiktsmessig siden vi ønsker begge funksjonalitetene, 1) for å gjøre det lettere å overskrive verdier i spillbrettet og 2) for å lese verdier uten mulighet til å redigere spillbrettet (f.eks. en spiller som ser spillbrettet skal ikke ha mulighet til å manipulere brikkene, det ville vært en enkel måte å jukse i spillet på).

4. (10 points) G4: `Game::new_game()` - Nytt spill

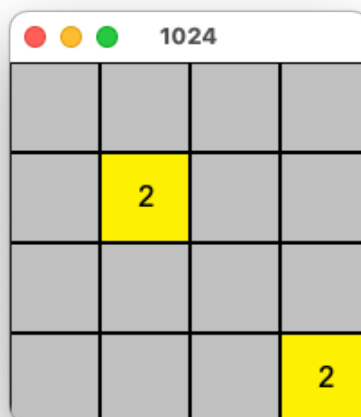
Medlemsfunksjonen skal gjenstarte spillet så en spiller kan begynne med blanke ark. Spillbrettet skal kun bestå av 0-verdier, med unntak av to fliser som er tilfeldig plassert på spillbrettet. Se figur 2 for et eksempel på hvordan spillbrettet kan se ut etter denne funksjonen.

Merk at du skal bruke medlemsfunksjonen `place_new_2` til å plassere 2-tallene - se neste deloppgave.

5. (10 points) G5: `Game::place_new_2` - Plasser ett tilfeldig 2-tall

Oppgaven din er å plassere ett 2-tall på en tilfeldig plass på spillbrettet.

Du kan anta at det finnes minst en ledig plass, det vil si minst en plass med verdien 0.



Figur 2: Nytt spill.

6. (10 points) **G6: Game::flip - Speil spillbrettet horisontalt**

Siden spillbrettet er et rektangel kan vi tenke på det som en matrise. I denne oppgaven skal du rokkere om på elementene i den interne representasjonen, altså rekkefølgen på elementene i board. Resultatet av en omrokking skal gjøre at `at()`-funksjonen henter ut elementer i en annen rekkefølge.

Som eksempel kan vi ta en rad fra en matrise, `[1, 2, 3, 4]`, som speilvendt blir `[4, 3, 2, 1]`. Hvis vi bestemmer at dette er første rad, vil `at(0, 0)`, `at(1, 0)` og `at(2, 0)` gi oss hhv. 1, 2, 3, men i det speilvendte tilfellet vil de samme funksjonskallene gi oss hhv. 4, 3, 2. Se også figur 4 for hvordan en speilvending av figur 3 ser ut i programmet.

Tips: fra og med denne oppgaven kan det være nyttig å enkelt fylle inn debug-verdier i spillbrettet og teste dine implementasjoner av `flip()`, `transpose()`, osv. Vi har opprettet funksjonalitet som bl.a. lar deg kalle dine implementasjoner fra spillvinduet. Spillvinduet tolker enkelte tastetrykk og utfører noen handlinger som endrer spillbrettet uten å sjekke gyldigheten av spillet. Merk at du kan sette spillet i en ugyldig tilstand, men da skal du kunne trykke 'r' for å resette spillet (dette kaller din `new_game()`-funksjon). Tabell 1 inneholder en oversikt over hvilke taster som gjør hvilken handling.

Tast	Handling
R/r	Start spillet på nytt
F/f	Speilvend spillbrettet
T/t	Transponer spillbrettet
I/i	Fyll spillbrettet med heltallsverdiene i intervallet <code>[1, 16]</code> (figur 3)
D/d	Fyll hele spillbrettet med 2-tal
P/p	Fyll spillbrettet med verdier for å teste push og merge (figur 6)
q	Lukk spillet/programmet

Tabell 1: Tastetrykk for debugging.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Figur 3: Spillbrettet fylt med tallene 1-16 (tastetrykk 'i').

4	3	2	1
8	7	6	5
12	11	10	9
16	15	14	13

Figur 4: Speilvending av figur 3 (tastetrykk 'f').

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Figur 5: Transponert av figur 3 (tastetrykk 't').

7. (10 points) **G7: Game::transpose - Transponer spillbrettet**

Vi fortsetter tankegangen med at spillbrettet er en matrise. Din oppgave er å transponere spillbrettet. Å transponere en matrise betyr å speile den langs diagonalen. Det er det samme som å bytte om på kolonnene og radene. F.eks. er spillbrettet i figur 5 den transponerte av matrisen i figur 3.

Merk at en matrise som transponeres to ganger vil være lik den opprinnelige matrisen. Det er verdt å dobbelsjekke at dette stemmer for din implementasjon.

8. (20 points) **G8: Game::push_right - Skyv alle elementer til høyre**

Denne funksjonen skal skyve alle elementer på spillbrettet så langt til høyre som mulig.

F.eks. vil en rad med elementene [2 2 0 4] etter denne operasjonen bli [0 2 2 4] og raden [2 0 2 0] bli [0 0 2 2]. Se figur 7 for resultatet av et kall til denne funksjonen med figur 6 som utgangspunkt.

Tips: avhengig av hvordan du ønsker å løse oppgaven kan bruke en beholder som både kan `push_back()` og `push_front()`, f.eks. `std::list`.

Tips 2: Vi har lagt inn en funksjonskall til denne funksjonen i `Game::move_right()` (G10). Det har vi gjort så du kan se resultatet av implementasjonen din når du trykker høyre piltast. Du kan også sette spillbrettet til figur 6 med tastetrykk 'p'.

9. (20 points) **G9: Game::merge_right - Slå sammen like fliser**

Denne funksjonen skal slå sammen to og to elementer med lik verdi som står inntil hverandre på samme rad. Hvis det er tre elementer på samme rad med lik verdi er det de to lengst til høyre som skal slås sammen.

Flisene som slås sammen skal danne en flise med den totale verdien av flisene som ble slått sammen. Du kan anta at alle fliser allerede er skjøvet helt til høyre før denne funksjonen kalles (`Game::push_right` kalles før denne funksjonen) så det er ingen åpenrom mellom to brikker som har høyere verdi enn 0.

F.eks. vil en rad med elementene [0 2 2 4] etter denne operasjonen bli [0 0 4 4], raden [0 2 2 2] bli [0 2 0 4] og raden [2 2 2 2] bli [0 4 0 4].

Tips: du kan trykke 'd' for å fylle spillbrettet med kun 2-tall.

10. (10 points) **G10: Game::move_right - Spilleren flytter til høyre**

2		2	
	2	2	4
	2	2	2
8	8	8	8

Figur 6: Spillbrettet fylt med testverdier for push og merge (tastetrykk 'p').

		2	2
	2	2	4
	2	2	2
8	8	8	8

Figur 7: Et kall til funksjonen push() med figur 6 som utgangspunkt.

			4
		4	4
		2	4
		16	16

Figur 8: Forventet oppførsel av move_right() med figur 6 som utgangspunkt.

Denne funksjonen skal gjennomføre spillerens trekk til høyre - altså når spilleren trykker høyre piltast. La oss si at et spillbrett inneholder raden [8 4 4 2], etter at denne funksjonen har gjort jobben sin skal det nye innholdet i raden være [0 8 8 2]. Se figur 8 for eksempel på korrekt oppførsel etter at det er gjennomført *ett* trekk med figur 6 som utgangspunkt.

Algoritmen du kan bruke for å lykkes med dette er som følger:

1. Flytt alt så langt til høyre som mulig.
2. Slå sammen fliser.
3. Flytt alt så langt til høyre som mulig.

Når du har gjort denne oppgaven skal du kunne trykke høyre piltast på tastaturet for å flytte alle brikkene til høyre i spillbrettet. Ettersom du implementerer flere retninger kan du bruke de andre piltastene til å flytte de respektive retningene.

11. (10 points) **G11: Game::move_down - Spilleren flytter nedover**

Denne funksjonen skal gjennomføre spillerens trekk nedover. I innledningen nevnte vi at alle forflytninger er like, men i forskjellige retninger. Matriseoperasjonene, transponering og flip/speiling, du har implementert så langt skal være nok til at du kan implementere G11, G12 og G13.

For å gjennomføre forflytninger i andre retninger skal du gjenbruke move_right(), men først må spillbrettet transformeres slik at nedover blir høyre. Når en forflytning er gjennomført må spillbrettet transformeres tilbake igjen.

12. (10 points) **G12: Game::move_left - Spilleren flytter til venstre**

Som G11, men spilleren ønsker å flytte til venstre.

13. (10 points) **G13: Game::move_up - Spilleren flytter oppover**

Som G11 og G12, men spilleren ønsker å flytte oppover.

14. (10 points) **G14: Game::free_spots - Er det noen ledige plasser?**

Funksjonen skal returnere true hvis det er mulig å plassere en ny flis på brettet, med andre ord om det finnes en flis som har verdien 0. Hvis ikke skal funksjonen returnere false.

15. (10 points) **G15: Game::tick - Fullføring av trekk**

Denne funksjonen kalles etter at en spiller har forsøkt å gjennomføre et trekk. Din oppgave er å sjekke om trekket spilleren forsøkte seg på flyttet noen brikker og om det fortsatt er ledige plasser igjen på brettet etter trekket. Hvis det er tilfellet skal det plasseres en ny flis med verdien 2 på brettet.

Hvis et trekk flyttet på en brikke vil det gjenspeiles i medlemsvariabelen `bool moved`. Den er `true` hvis et trekk førte til at spillbrettet endret seg og `false` hvis spillbrettet er uendret etter at spilleren har trykket en av piltastene.

16. (10 points) **G16: Game::win - Har spilleren vunnet?**

Hvis spillet er vunnet skal funksjonen returnere `true`, eller `false` hvis ikke.

Spillet er vunnet hvis en flis på spillbrettet holder verdien vi har lagret i medlemsvariabelen `win_value`, 1024.

17. (20 points) **G17: Game::legal_moves - Er det noen gyldige trekk igjen?**

Denne funksjonen skal sjekke om det er gyldige trekk igjen i de tilfellene det kun er fliser på brettet som har verdi over 0.

Hvis det er mulig å gjennomføre et gyldig trekk skal funksjonen returnere `true`, ellers `false`.

Denne funksjonen kalles etter et trekk og kun hvis vi vet at det ikke finnes noen ledige plasser på brettet. Det betyr at spillbrettet alltid vil være fylt av verdier over 0 når denne funksjonen kalles. En mulig algoritme er å finne ut av om det gjenstår gyldige trekk er å sjekke alle kolonner og rader for om det er mulig å slå sammen tilstøtende fliser. Hvis det er mulig, så finnes det gyldige trekk.