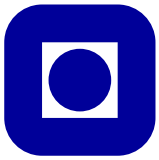


Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

TDT4102 Prosedyre- og objektorientert programmering Vår 2020

Øving 4

Frist: 2020-02-07

Mål for denne øvingen:

- Jobbe med `string`, funksjoner og `struct`
- Lage et fullstendig program (et enkelt spill)
- Kode grafikk med FLTK

Generelle krav:

- Alle funksjoner som defineres i `test.cpp` skal kalles fra main().
- 70% av øvingen må godkjennes for at den skal vurderes som bestått.
- Øvingen skal godkjennes av stud.ass. på sal.
- Bruk Visual Studio Code, med mindre du kjenner et annet utviklermiljø bedre.

Anbefalt lesestoff:

- Appendix B.8.1 og B.8.2

NB: Hele øvingen skal kompileres til en enkelt kjørbar fil (tilsvarende et prosjekt dersom du bruker Visual Studio Code), men legg merke til oppdelingen i separate filer.

1 Pass-by-value vs. pass-by-reference (10%)

Nyttig å vite: Pass-by-reference

De fleste funksjonene vi har sett på så langt i øvingsopplegget har tatt inn argumenter på såkalt «pass-by-value»-form. «Pass-by-value» vil si at verdien vi får inn som argument er en kopi av originalen. Hvis vi endrer på den, vil ikke endringen gjenspeiles i originalverdien som ble sendt inn. «Pass-by-reference» derimot, oppretter et alias til objektet som befinner seg utenfor funksjonen. Når det utføres en operasjon på referansen utføres det en operasjon på det opprinnelige objektet.
For å ta inn et argument som referanse må du bruke &. F.eks

```
int incrementByValueNumTimes(int& startValue, int increment, int numTimes);
```

Her blir `startValue` tatt inn som en refereanse.
Se §8.5.4 til 8.5.6 i læreboka (PPP) for illustrasjoner og argumentasjon for når verdi og (const) referanse bør benyttes

a) Kodeforståelse:
Når programmet under har kjørt ferdig, hvilken verdi er skrevet ut for `v0`?

```
int incrementByValueNumTimes(int startValue, int increment, int numTimes) {
    for (int i = 0; i < numTimes; i++) {
        startValue += increment;
    }
    return startValue;
}

void testCallByValue() {
    int v0 = 5;
    int increment = 2;
    int iterations = 10;
    int result = incrementByValueNumTimes(v0, increment, iterations);
    cout << "v0: " << v0
         << " increment: " << increment
         << " iterations: " << iterations
         << " result: " << result << endl;
}

int main() {
    testCallByValue();
}
```

b) utilities
Opprett en ny fil `utilities.cpp` med tilhørende headerfil. Legg til funksjonen `incrementByValueNumTimes()` fra forrige deloppgave i filen.

c) Tester
Opprett en ny fil `tests.cpp` med tilhørende headerfil, og legg til funksjonen `testCallByValue()` i `tests.cpp`. Denne funksjonen skal teste at `incrementByValueNumTimes()` virker. Kall `testCallByValue()` fra `main()`, gjerne med testmeny som i øving 2. `main()` skal være i `main.cpp`.

d) Funksjoner med referanseparameter
Lag en ny funksjon `incrementByValueNumTimesRef()`, som gjør akkurat det samme som `incrementByValueNumTimes()`, men bruker referanse. Funksjonen skal ikke returnere noe. Lag en ny funksjon `testCallByReference()` som tester

`incrementByValueNumTimesRef()` på tilsvarende måte som `testCallByValue()` tester `incrementByValueNumTimes()`. *Husk å oppdatere headerfilene.*

e) Swap

Skriv en funksjon `swapNumbers()` som bytter om på to heltallvariablers verdi. Funksjonen skal ligge i `utilities.cpp`. Bør denne funksjonen bruke referanser? Begrunn svaret ditt.

2 `vector<int>` (10%)

Enkel håndtering av heltall i en beholder, `vector<int>`.

a) Definer funksjonen `testVectorSorting()`

Plasser funksjonen i `tests.cpp` og kall den fra `main()`. Funksjonen skal ikke ta inn noe, og ikke returnere noe.

b) Vector med heltall

Definer en variabel du navngir `percentages` av typen `vector<int>` i funksjonen `testVectorSorting()`.

c) Definer funksjonen `randomizeVector()`

Defineres i `utilities.cpp`. Funksjonen tar inn en referanse til en `vector<int>`, heltallet `n` og fyller vectoren med `n` tilfeldige prosentverdier, [0, 100]. Bruk det du allerede har lært i øving 3 for å generere tilfeldige tall. Legg også til et kall til denne funksjonen i `testVectorSorting()`, slik at `percentages` har innsatte verdier. Deretter skal `testVectorSorting()` skrive ut de tilfeldige tallene som nettopp ble lagret i `vectoren`.

d) Enkel ombytting (swapping)

Bruk `swapNumbers()` i `testVectorSorting()` til å bytte om på de to første elementene i `vectoren`.

3 Sortering (10%)

I denne oppgaven skal vi se på sortering av en `vector`.

a) Sortering av heltallsvector.

Definer funksjonen `sortVector()` i `utilities.cpp` som tar inn (dvs. har som parameter) en referanse til en `vector<int>` og sorterer denne. Du skal her implementere sorteringsalgoritmen *insertion sort*. Pseudokoden for denne algoritmen kan finnes på f.eks. wikipedia:

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    end while
    i ← i + 1
end while
```

Test funksjonen på `percentages` i `testVectorSorting()` fra oppgave 2.

b) Definer funksjonen `medianOfVector()`.

Funksjonen skal defineres i `utilities.cpp` og returnere medianen til `vectoren`. Medianen til en sortert rekke av tall er definert som det midterste elementet i rekken dersom antall tall

er odde. Dersom rekken har et like antall tall er medianen definert som gjennomsnittet av de to midterste tallene. Funksjonen tar inn en `vector` av heltall som antas å være sortert, og skal fungere for `vector`er av vilkårlig størrelse.

Test funksjonen på `vector`en `percentages` i `testVectorSorting()` før og etter den sorteres. Får du forskjellige svar?

I denne oppgaven kan du se at argumenter som sendes til funksjonen for `vector`-parameteren ikke skal oppdateres. Velg pass-by-value, pass-by-reference eller pass-by-const-reference, alt etter hva som passer best. Se kapittel 8.5 og særlig "rule of thumb" nederst på s. 282 i læreboken.

4 Struct (10%)

a) Lag structen `Student` i `utilities.h`. `Student` skal holde variablene:

- name, av typen `string`
- studyProgram, av typen `string`
- age, av typen `int`

b) **Definer funksjonen `printStudent()`.** Funksjonen skal defineres i `utilities.cpp`. Funksjonen skal ta inn en `Student` som parameter, og skrive en *fin* av utskrift av `name`, `studyProgram` og `age` til skjerm.

c) **Test structen og funksjonen fra `main()`** Det forventes at du alltid tester koden din, selv om det ikke er oppgitt i oppgaveteksten. Hvis du er usikker på hvordan du kan teste koden din kan du spørre en læringsassistent.

5 `string` og behandling av tegn (char) (15%)

Nyttig å vite: "char-aritmetikk"

En bokstav, `char`, er i bunn og grunn en tallverdi (`'\0'` = 0), og dermed kan vi bruke regneoperasjoner:

```
// finner den n-te bokstaven i alfabetet (A-Z)
char c = 'A' + n - 1;
```

Samsvaret mellom tegn og tall finner du i en ASCII-tabell.

a) Definer funksjonen `testString()`

Defineres i `tests.cpp`. Funksjonen har ingen parameter og skal ikke returnere noe.

b) Definer funksjonen `randomizeString()`

Defineres i `utilities.cpp`. Funksjonen skal returnere en `string` av en gitt lengde som inneholder tegn i et gitt intervall. Parametere er antall tegn strengen skal bestå av, samt en øvre og nedre grense for tegn strengen kan fylles med (f.eks. `'A'` og `'G'`).

c) Opprett en `string grades` i `testString()`.

d) Tilfeldige karakterer

Benytt `randomizeString()` til å fylle inn 8 tilfeldige tegn (her karakterer) (A-F) i `grades`.

e) Utskrift

Skriv `grades`-stringen til skjerm i `testString()`.

f) Definer funksjonen `readInputToString()`

Defineres i `utilities.cpp`. Funksjonen skal returnere en `string` av lengde `n`. Der tegnene i strengen skal ligge innenfor en øvre og nedre grense. Parametere er en øvre og nedre grense for hvilke tegn som er tillatt og `n`. Strengen som returneres skal fylles vha. konsollinput, `cin`. Hvis input ikke ligger innenfor nedre og øvre grense skal funksjonen be brukeren om ny input, her bryr vi oss ikke om input er store eller små bokstaver, slik at begge deler skal godkjennes. ("aA" == "Aa").

Se Appendix B.8.1 for funksjoner som kan hjelpe deg med å konvertere mellom store og små bokstaver, finne ut om det du leser er en bokstav, siffer, e.l.

g) Definer funksjonen `countChar()`

Defineres i `utilities.cpp`. Funksjonen tar inn en `string` og en `char`. Funksjonen skal returnere antall forekomster av tegnet i strengen som er sendt til funksjonen.

h) Telle karakterer og gjennomsnitt.

Opprett en `vector` med heltall, `gradeCount`, i `testString()`. Den skal romme antall forekomster av hver karakter (A-F).

Bruk `countChar()` til å fylle `gradeCount` med antallet forekomster av hver karakter i `grades`. Regn ut og skriv snittkarakteren til konsollen. (La A tilsvare 5, B tilsvare 4, osv, slik at snittet kan beskrives med tall).

6 Mastermind (25%)

I denne deloppgaven skal du implementere spillet Mastermind. Programmet ditt skal lage en tilfeldig kode på 4 bokstaver bestående av tegnene i intervallet [A, F]. Brukeren av programmet skal deretter gjette hvilke bokstaver koden inneholder og hvilken rekkefølge de er i. Etter hver gang brukeren har gjettet, skal programmet fortelle brukeren hvor mange bokstaver brukeren gjettet riktig og hvor mange bokstaver som ble riktig plassert. Det er ikke nødvendig å implementere programmet ditt nøyaktig som beskrevet under, så lenge funksjonaliteten blir den samme og du gjenbruker kode fra tidligere i øvingen.

a) Grunnleggende oppsett.

Lag en ny fil som inneholder funksjonen som starter spillet. La funksjonen hete `playMastermind()`. Kall den fra `main`. Definer følgende heltallskonstanter (`constexpr`) i `playMastermind()`:

- `size`, antall tegn i koden, 4.
- `letters`, antall forskjellige bokstaver, 6.

Hvorfor bruker vi `constexpr` og ikke `const` her?
Når kunne det være hensiktsmessig å benytte `const` framfor `constexpr` for dette spillet?

b) Datastruktur. Lag to tekststrenger (`string`) i `playMasterMind()`

- `code` - Skal inneholde koden spilleren skal prøve å gjette.
- `guess` - Skal inneholde bokstavene spilleren gjetter.

c) Generer kode.

Bruk funksjonen `randomizeString()` til å fylle `code` med tilfeldige bokstaver mellom `'A'` og `'A'` + (`letters` - 1).

d) Spillerinput.

Bruk `readInputToString()` til å spørre spilleren etter `size` antall bokstaver, og lagre dem i `guess`.

e) Korrekt plassering.

Skriv funksjonen `checkCharactersAndPosition()`, som skal returnere hvor mange riktige bokstaver spilleren har på riktig posisjon. Svaret skal returneres som et heltall.

f) Korrekt bokstav, uavhengig av posisjon.

Skriv funksjonen `checkCharacters()`, som skal returnere hvor mange riktige bokstaver spilleren har gjettet, uavhengig av posisjon. Resultatet skal være hvor mange bokstaver som befinner seg i både kode og gjett, om de er på samme posisjon i kode og gjett er likegyldig. Svaret skal returneres som et heltall.

Eksempel: koden er ABCD. Om man gjetter CDEE vil C og D være bokstaver som er en del av koden og denne funksjonene skal returnere 2. En annen gjetning kan være BACD, alle bokstavene er med i koden, men kun to er på korrekt plass. For BACD skal funksjonen likevel returnere 4, for den bryr seg kun om hvor mange av bokstavene som er med i både koden og gjettet.

g) Spill-løkke.

Utvid koden fra **d)** slik at programmet spør spilleren etter en ny kode så lenge `checkCharactersAndPosition()` returnerer et tall mindre enn `size`.

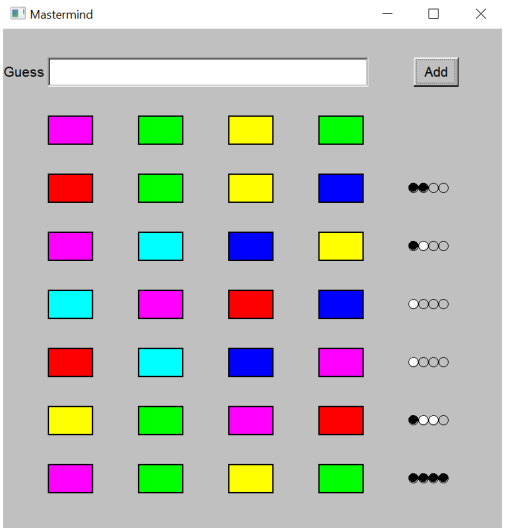
h) Fullfør spillet.

Flett sammen spillogikken i `playMastermind()`. Når du er ferdig skal programmet lage og lagre en tilfeldig kode som spilleren kan gjette på inntil den rette koden er funnet. For hver kode spilleren gjetter skal programmet skrive ut hvor mange riktige bokstaver spilleren gjettet, og hvor mange av dem som var på rett plass.

i) Begrenset antall forsøk.
Utvid koden din slik at spilleren har et begrenset antall forsøk på å gjette koden.

j) Seier eller tap.
Utvid spillet til å gratulere spilleren med seieren (eller trøste den etter tapet).

7 Mastermind med grafikk (20%)



Figuren over viser en grafisk framstilling av spillet du har programmert. Den øverste raden med rektangler i vinduet representerer den hemmelige koden som skal gjettes. Hver rektangel representer en bokstav. De resterende radene med rektangler er gjetninger brukeren har utført. Sirklene til høyre for rektangelene gir brukeren tilbakemeldig om hvor mange bokstaver de har gjettet riktig, og hvor mange bokstaver som er gjettet i korrekt posisjon.

Utdelt kode, `masterVisual.cpp` og `masterVisual.h` skal brukes til å gi en grafisk framstilling av spillet. Det er ikke nødvendig å forstå den utdelte koden for å løse oppgaven. Det du skal bruke av den utdelte koden blir forklart. På slutten av kurset vil du være i stand til å forstå den utdelte koden.

Skalering

Når man jobber med grafikk er det lurt å bruke konstanter for størrelsene på elementene i vinduet. I denne oppgaven skal du bruke en veldig enkel skalering der størrelsen og plasseringen av rektanglene og sirklene er avhengig av hvor stort vinduet er og hvor mange elementer det er i vinduet.

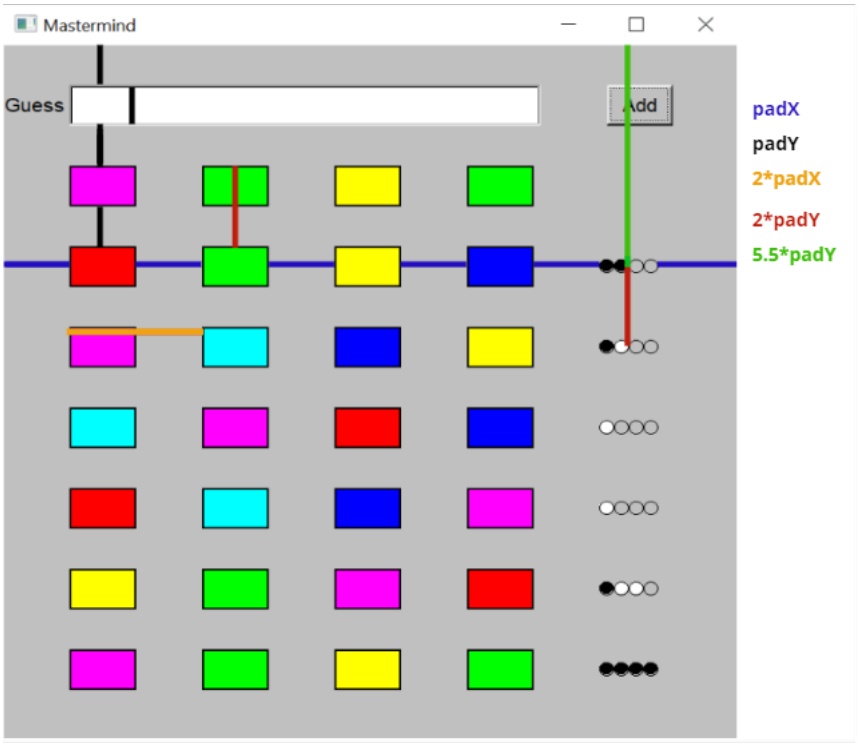
I `masterVisual.h` er det definert en del konstanter. Konstantene du skal endre er `winW, winH, padX, padY` og `radCircle`.

`winW` og `winH` er vinduets bredde og høyde.

`padX` er avstanden mellom hvert element i x-retning og bredden til hver rektangel og bredden til alle sirklene i x-retning. Med element menes det her en rektangel eller alle sirklene i x-retning. `padX` er vinduets bredde delt på to ganger antall elementer i x-retning pluss en. F.eks hvis du skal gjette en kode på fire bokstaver: `padX=winW/(2*5+1)`. Siden du har fem elementer i x-retning, fire rektangler og ett element med sirklene.

`padY` er avstanden mellom hvert element i y-retning og høyden til hver rektangel. `padY` er vinduets bredde delt på antall elementer i y-retning ganger to pluss en. F.eks hvis du skal spille seks runder er `padY=winH/(2*8+1)`. Siden du har seks elementer til de seks rundene i tillegg har du to elementer i tekstboksen og koden som skal gjettes.

`radCircle` er radiusen til hver av sirklene. Siden alle sirklene til sammen skal ha bredde `padX` så er radiusen til hver sirkel `padX` delt på antall sirkler ganger to. F.eks hvis du skal gjette fire bokstaver er `radCircle=padX/8`. Resten av konstantene bestemmer plasseringen til add-knappen og tekstboksen. Figuren nedenfor viser Mastermindvinduet med mål.



a) Grafikkvindu

Bestem hvor mange bokstaver som skal gjettes, og hvor mange runder som skal spilles. Vi anbefaler fire bokstaver og seks runder, og bestem hvor stort vinduet skal være. Vi foreslår en bredde på 500 og høyde på 500 piksler. Definer `padX, padY` og `radCircle`.

Opprett et grafikkvindu i `playMastermind()`. I denne øvingen skal du benytte datatypen `MastermindWindow` som er definert i `masterVisual.h`. Opprett vinduet (som du kaller `mwin`) ved å plassere følgende kodelinje i `playMastermind()`:

```
MastermindWindow mwin{Point{900, 20}, winW, winH, "Mastermind"};
```

Punktet (900, 20) angir posisjonen til øvre venstre hjørne av vinduet. `winW` og `winH` er bredden og høyden, og det siste argumentet er tittelen til vinduet.

I denne øvingen bør du opprette *ett nytt vindu for hvert spill*, da vil programmet rydde opp etter seg selv og hvert nye spill starter med et tomt vindu.

Du må i tillegg bytte ut funksjonen `readInputToString()` med `mwin.getInput()` i `playMastermind()`. `getInput()` leser inn tekst fra tekstboksen i Mastermindvinduet. Den sjekker at gjetting er på riktig format, og bare inneholder lovlige tegn. Funksjonen returnerer en `string` med store bokstaver.

Nederst i `main()` må du skrive `return gui_main`, og øverst i `main`-filen må du inkludere `Graph.h` og `Simple_window.h`. I tillegg må du bruke navneromet `Graph_lib`. Det gjør du ved å skrive `using namespace Graph_lib;` øverst i `main`-funksjonen.

b) Visning av kode og gjetning

Din oppgave er nå å definere `addGuess()` som skal tegne en rad med fargede rektangler i vinduet basert på en gjetning. Vi benytter tallverdiene for farger som ble presentert i forelesning (se slide "FLTK colors and colorvalues"). Tallverdien 1 = rød, 2 = grønn osv. Parameterne til funksjonen er en referanse til vinduet du har opprettet, bokstavkoden som er gjettet, lengden av denne koden målt i antall tegn, første bokstav som kan gjettes og hvilken runde (eller omgang) gjetningen tilhører. Som du ser i eksempel-figuren får vi en rad med rektangler for hver runde.

`MastermindWindow` inneholder en `Vector_ref vr` som skal lagre referansene til rektanglene i vinduet. Du kan tegne en rektangel i vinduet ved å skrive :

```
mwin.vr.push_back(new Rectangle{Point{xPos,yPos},padX,padY});
mwin.vr[mwin.vr.size()-1].set_fill_color(c);
mwin.attach(mwin.vr[mwin.vr.size()-1]);
```

Her er `Point{xPos,yPos}` posisjonen til hjørnet oppe til venstre av rektangelen, `padX` og `padY` er bredden og høyden til rektanglet. `c` er en farge.

`yPos` er avhengig av runde-nr. Mens `xPos` starter på samme sted hver runde. Du må inkrementere `xPos` for hver rektangel du tegner. Fargen `c` er avhengig av hvilken bokstav som er gjettet.

Ved utvikling og debugging av programmet kan det være nyttig å spille med synlig kode. Det kan du oppnå ved å kalle `addGuess()` med den hemmelig bokstavkoden som den andre parameteren, og verdien 0 for runde-nr. Deretter inkrementerer du runde-nr for hver gjetning.

Du må plassere kall på `addGuess()` på egnet sted inne i funksjonen `playMastermind()`. Du må også kalle `mwin.redraw()` etter spill-løkken din før du spørr om brukeren vil spille igjen. Du skal fortsatt spørre brukeren om de vil spille igjen i terminalen.

Når du mener logikken i programmet ditt er helt riktig kan du endre det til å spille med skjult kode. For å gjøre det enkelt har vi laget en funksjon `hideCode()` som tegner en sort rektangel oppå den øverste raden som tidligere viste den hemmelige koden.

c) Vis antall korrekte plasseringer

I denne deloppgaven skal du kode tilbakemeldingsdelen av mastermind-spillet, dvs. de sorte og hvite sirklene i høyre del av figuren. Dette skal du gjøre i funksjonen `addFeedback()`. Parameterne er vinduet, antall korrekte plasseringer, antall korrekte bokstaver med feil posisjon, størrelsen på koden og hvilken runde tilbakemeldingen tilhører.

`MastermindWindow` inneholder en `Vector_ref vc` som skal lagre referansene til sirklene i vinduet. Du kan lage en sirkel ved å skrive:

```
Circle{Point{xPos,yPos},radCircle};
```

`Point{xPos,yPos}` er her sentrum av sirkelen og `radCircle` er radiusen. Bruk sort sirkler for å markere korrekt bokstav med korrekt posisjon og hvit for korrekt bokstav uavhengig av posisjon.

Plasser kallet av `addFeedback()` på egnet sted i `playMastermind()`.