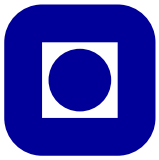


Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

TDT4102 Prosedyre- og objektorientert programmering Vår 2019

Øving 4

Frist: 2019-02-08

Mål for denne øvingen:

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

Generelle krav:

- Bruk Visual Studio eller Xcode, med mindre du kjenner et annet utviklermiljø bedre.
- 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.

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/XCode), men legg merke til oppdelingen i separate filer.

1 Pass-by-value vs. pass-by-reference (15%)

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.

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 `vectorer` av vilkårlig størrelse.
Test funksjonen på `vectoren 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 `string` og behandling av tegn (char) (20%)

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 `stringen 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`. En øvre og nedre grense for hvilke tegn som er tillatt og `n` er parametre. Strengen som returneres skal fylles vha. konsollinput, `cin`. Funksjonen skal også sjekke om input består av alfanumeriske tegn og er innenfor øvre og nedre grense, 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).

5 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). Svaret skal returneres som et heltall.

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).

6 Mastermind med grafikk (20%)

For å løse denne oppgaven må du opprette et grafikkprosjekt.

Utdelt kode, `masterVisual.cpp` og `masterVisual.h` kan brukes til å gi en grafisk framstilling av spillet du har programmert. Vi forklarer her hvordan du kan kople spill-koden til den utdelte grafikk-koden. De aller fleste av begrepene som benyttes her er allerede forelest, og forståelsen av `Vector_ref` er sentralt i oppgaven som skal gjøres. Vi vil hjelpe deg å bruke den utdelte koden nedenfor.

Grafikkvindu

Opprett et vindu som gjør det mulig å tegne figurer slik som rektangler og sirkler på skjermen. 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()`:

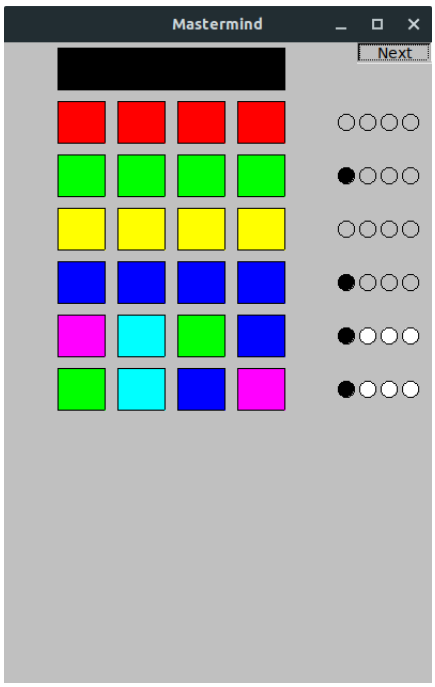
```
MastermindWindow mwin{Point{900, 20}, win_w, win_h, "Mastermind"};
```

Punktet (900, 20) angir posisjonen til øvre venstre hjørne av vinduet. `win_w` og `win_h` er dine egne variable du benytter for å angi henholdsvis bredde og høyde for vinduet, og vi angir "Mastermind" som vinduets tittel. Vi foreslår en bredde på 400 og høyde på 600 piksler.

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.

a) Visning av kode og gjetning

Den utdelte koden har allerede definert to funksjoner, en for å vise et forsøk på å gjette kode og en funksjon for å skjule koden som skal gjettes. `addGuess()` tegner opp resultatet av en gjetning i vinduet som en rad med fargede rektangler. 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.



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. `addGuess()` plasserer hver ny gjetning som en ny rad litt lenger ned i vinduet, ved å la runde-nr inngå i beregning av y-koordinatet. 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 rett og slett tegner en sort rektangel oppå den øverste raden som tidligere viste den hemmelige koden. Parametere er referanse til vinduet og antall tegn i koden. Hovedutfordringen i denne deloppgaven er å plassere kall på `addGuess()` og `hideCode()` på egnet sted inne i funksjonen `playMastermind()`. Merk at `MastermindWindow` også har en medlemsfunksjon `wait_for_button()`[1], som gjør det mulig å fryse vinduet og vente til en knapp er trykket før vinduet går til neste visning. Den er meget nyttig ved debugging av grafikk og du kaller den slik: `mwin.wait_for_button();`

Programmet du nå lager krever at brukeren både skriver i konsoll-vinduet og trykker på next-knappen i det grafisk vinduet. Det kan virke noe tungvindt, men er en konsekvens av at vi har koplet grafikk til en tekst-basert løsning. En mer naturlig løsning vil være å bare bruke det grafiske vinduet, og det vil du være i stand til om noen få øvinger.

b) 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. Du har også fått utdelt *deler av* en funksjon `addFeedback()` som for en gitt gjetning kan vise hvor mange bokstaver som er korrekt plassert, og hvor mange bokstaver som er riktige men har feil plassering. Du skal fullføre koden i `addFeedback()` slik at den tegner sirkler som gir riktig tilbakemelding på en gjetning. Parameterne er vinduet, antall korrekte plasseringer, antall korrekte bokstaver med feil posisjon, størrelsen på koden og hvilken runde tilbakemeldingen tilhører. Bruk sorte fylte sirkler (skiver) for å markere korrekt bokstav med korrekt posisjon og hvite skiver for korrekt bokstav uavhengig av posisjon.

Vinduet `mwin` er en instans av (et objekt) av typen `MastermindWindow` som er definert[2] i `masterVisual.h`. Fra koden ser vi at ethvert slikt objekt har to såkalte medlemsvariable, `vr` og `vc` som begge er av typen `Vector_ref`. Den første, `vr` benyttes av `addGuess()` for å lagre referanser til rektangler, mens den andre brukes i denne deloppgaven for referanser til sirkler. Vi har tilgang til disse medlemsvariablene via såkalt dot-notasjon. `mwin.vr` kan leses som "mwin sin vr".

Som sagt skal tilbakemeldingene vises som sirkler og referanser til dem lagres i `vc`. Funksjonen som er utdelt oppretter allerede sirklene, *din oppgave er å plassere sirklene korrekt, gi de korrekt farge og kople dem til vinduet.* For å plassere sirklene må du beregne korrekt posisjon for sirklenes senter i y- og x-retning. Funksjonen inneholder kommentarer der det skal være nødvendig å legge til kode.

Du får mye hjelp til denne oppgave ved å studere `addGuess()`. Som du ser er det relativt omfattende beregninger for å regne ut hvordan vinduets størrelse kan fordeles på de ulike grafiske elementene. Mye av det finner du øverst i `masterVisual.cpp` som `constexpr`-uttrykk. Prøv gjerne ut programmet ditt før du er sikker på at alle beregningene av x og y-koordinater er riktige. Disse beregningene er jo ekstra-arbeid som er en konsekvens av grafikken, men grafikken kan også gi deg utvidet hjelp under debuggingen fordi layouten visualiseres!

[1] Hvis du slår opp på definisjonen av `MastermindWindow` i `masterVisual.h` så vil du ikke finne denne medlemsfunksjonen. Årsaken er at den er definert i `Simple_window` som er en del av lærebokas grafikk-bibliotek. `MastermindWindow` er en utvidelse av `Simple_window`, og arver funksjonen derfra. Arv er et sentralt begrep i objektorientert programmering som du vil lære mer om senere.

[2] I definisjonen av typen utnyttes begrepene klasser (`struct` og `class`), konstruktør og arv. Disse vil bli forklart nærmere senere i kurset, og du trenger ikke forstå dem enda for å fullføre denne øvingen.