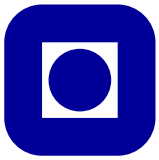

Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

TDT4102 Prosedyre og Objektorientert programmering Vår 2015

Øving 3

Frist: 2014-02-07

Mål for denne øvinga:

- Lage og bruke tabeller (arrays) med forskjellige datatyper
- Jobbe med tabeller og funksjoner
- Lage et fullstendig program (et enkelt spill)

Generelle krav:

- Bruk de eksakte navn og spesifikasjoner som er gitt i oppgava
- Det er valgfritt om du vil bruke en IDE (Visual Studio, Xcode), men koden må være enkel å lese, kompilere og kjøre

Anbefalt lesestoff:

- Kapittel 5, Absolute C++ (Walter Savitch)

NB: Hele øvinga skal gjøres i ett Visual C++/XCode-prosjekt, men legg merke til oppdelingen i separate filer.

1 Call-by-Value vs Pekere (10%)

De fleste funksjonene vi har sett på så langt i øvingsopplegget har tatt inn argumenter på såkalt «Call-by-value»-form, det 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.

a) Hva vil verdien til v0 være når følgende program har kjørt? (Skriv svaret som en kommentar i koden din).

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

int main() {
    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;
}
```

b) Opprett en ny fil `tests.cpp` med tilhørende header, legg til funksjonen `testPart1` i `tests.cpp`, og kall denne fra `main` (gjerne fra en meny slik vi gjorde i Øving 2). Funksjonen skal ikke ta inn noe, og heller ikke returnere noe. Kopier koden som står i `main()` over inn i denne `testPart1`.

c) Skriv om funksjonen `incrementByValueNumTimes`, slik at den ved å bruke pekere, endrer på v0. Funksjonen skal nå ikke returnere noe som helst.

Oppdater `testPart1()` slik at denne fortsatt fungerer.

d) Opprett en ny fil `utilities.cpp` med tilhørende header

Du kan kopiere over innholdet fra `utilities.cpp` fra oppgave 6 i øving 2 hvis du har den tilgjengelig.

e) Skriv en funksjon `swapNumbers()` som tar inn to heltall, og bytter om på dem. Funksjonen skal ligge i `utilities.cpp`. Bør denne funksjonen bruke pekere? Begrunn svaret ditt.

Test `swapNumbers` fra `testPart1()`

2 Tabeller (arrays) (20%)

Tabeller er nyttige for å lagre forskjellige typer data, og kan brukes for å lagre lignende data i et sammenhengende minneområde. I denne oppgava skal vi se på håndtering av en slik tabell

a) Lag en ny funksjon `testPart2()` i `tests.cpp`, og kall denne fra `main`

Denne skal ikke ta inn noe, og ikke returnere noe.

b) Lag en tabell `percentages` i `testPart2()`-funksjonen din, med plass til 20 heltall

c) Lag en funksjon `printArray` i `utilities.cpp` som skriver ut tabellen til skjerm, med alle elementene på en linje.

Funksjonen din skal være i stand til å skrive ut tabeller av enhver lengde.

Hint: Denne funksjonen skal ta to argumenter.

d) Lag en funksjon `randomizeArray` i `utilities.cpp` som tar inn tabellen, og størrelsen på tabellen, for så å fylle tabellen med tilfeldige prosentverdier (0-100, inklusive).
Bruk gjerne funksjoner du har skrevet tidligere i øving 2.
Legg også til et kall til denne funksjonen i `testPart2()`, slik at tabellen nå har innsatte verdier, skriv den endrede tabellen ut fra `testPart2()`.

e) Bruk `swapNumbers` i `testPart2()` til å bytte om de to første elementene i tabellen.
Tabeller og pekere er egentlig to sider av samme sak, vi kan derfor gjenbruke swapNumbers-funksjonen vi lagde tidligere på tabellen.

f) (10%) Skriv en funksjon `sortArray` i `utilities.cpp` som tar inn en tabell og størrelsen til denne, og sorterer den.
Du får ikke lov til å bruke den innebygde sorteringsfunksjonen C++ har.
Sortering kan gjøres med f.eks. Insertion-sort, som beskrevet her: `http://en.wikipedia.org/wiki/Insertion_sort` .
Test funksjonen på tabellen `percentages` i `testPart2`.

g) Skriv en funksjon `medianOfArray` i `utilities.cpp` som returnerer medianen til tabellen.
Funksjonen skal anta at tabellen den kjøres på allerede ER sortert.
Test funksjonen på tabellen `percentages` i `testPart2()` før og etter den sorteres, får du forskjellige svar?
Funksjonen skal fungere for vilkårlige tabellstørrelser (du må altså ta høyde for både odde antall elementer, og partall antall elementer).

3 Char-tabeller (C-Strenger) (20%)

En C-Streng er en tabell med chars der siste element er 0. Det er derfor viktig å huske på å sette av plass til denne 0-en i tillegg til de tegnene vi vil ha i en char-tabell, hvis vi vil behandle char-tabellen som en C-streng.

a) Lag en ny funksjon `testPart3()` i `tests.cpp`, og kall denne fra `main` Denne skal ikke ta inn noe, og ikke returnere noe.

b) Lag en char-tabell `grades` i `testPart3()`, denne skal romme en C-streng med karakterene til en student for et skoleår, og må ha lengde deretter (vi antar at studenten tar 8 fag iløpet av de 2 semestrene året varer)

c) Skriv funksjonen `randomizeCString` i `utilities.cpp` som tar en char-tabell, et antall tegn, samt en øvre og nedre grense for tegn tabellen skal fylles med, og fyller tabellen med tilfeldige tegn mellom disse grensene. Pass på at siste element i tabellen fortsatt er 0 (slik at vi har en C-Streng.)
Hint: Gjenbruk `randomWithLimits` fra øving 2.

d) Bruk `randomizeCString` til å fylle inn 8 tilfeldige karakterer (A-F) i `grades`

C-Strenger har et stort fortrinn når det gjelder utskrift til skjerm, da de kan skrives ut mye enklere enn andre tabeller.

e) Skriv `grades`-tabellen til skjerm i `testPart3()` uten å bruke noen form for løkke

f) Skriv en funksjon `readInputToCString` i `utilities.cpp` som lar brukeren skrive inn et bestemt antall tegn til tabellen

Funksjonen skal ta inn en tabell, samt en øvre og nedre grense for hvilke tegn som er tillatt, og be brukeren om ny input dersom brukeren skriver tegn som ikke er innenfor (slik at vi ikke f.eks. tar inn 'G' som karakter).

Bruk den innebygde funksjonen `toupper` etter lesing, slik at brukeren kan skrive både store og små tegn.

g) Skriv en funksjon `countOccurencesOfCharacter` i `utilities.cpp` som tar inn en tabell med chars, lengden til denne, og et tegn. Funksjonen skal telle antall forekomster av dette tegnet

h) Lag en ny heltallstabell `gradeCount` i `testPart3`, som skal romme antallet forekomster av hver karakter (A-F)

Bruk `countOccurencesOfCharacter`, til å fylle tabellen med antallet forekomster av hver karakter i `grades`-tabellen. Regn så ut snittkarakteren ved hjelp av denne tabellen, og skriv denne ut til skjerm. (La A være 1, B være 2 etc, slik at snittet kan beskrives med tall).

i) Utvid koden i `testPart3()` slik at det genereres karakterer for 5 år, i verdirommet (A-E), og beregn snittet.

Test først med tilfeldig generert karakterer, skriv så om koden til å la brukeren angi karakterer for hvert år.

Regnereglene for karaktersnitt bruker 1 desimal, og rundes av etter vanlige regneregler. Hvis du vil finne en mer korrekt utregning, kan du bruke funksjonen `round` fra `cmath`-biblioteket til å runde av snittet. Du kan også bruke `setprecision` fra `iomanip`-biblioteket, til å sette utskriftspresisjonen. (For mer eksakte detaljer angående regneregler, se: `http://www.ntnu.no/studier/opptak/master/rangering`)

Hvis du vil skrive ut det avrundede snittet som bokstavkarakter, kan du bruke noe ala:

```
cout << (char) ('A' + round(average) - 1);
```

4 Mastermind (45%)

I denne deloppgava skal du implementere spillet Mastermind. Programmet ditt skal lage en tilfeldig kode på 4 bokstaver (A-F). Brukeren av programmet skal deretter gjette hvilke bokstaver det er i koden og hvilken rekkefølge de er i. Etter hver gang brukeren har gjettet, skal programmet fortelle brukeren hvor mange bokstaver brukere gjetter riktig, og hvor mange bokstaver som er riktig plassert. Det er ikke nødvendig å gjøre nøyaktig som beskrevet under, så fremt funksjonaliteten blir den samme og det benyttes arrays og fornuftig valgte funksjoner.

a) Lag en ny fil som inneholder funksjonen som starter spillet. La funksjonen hete `playMastermind` og kall den fra `main`-funksjonen i deloppgave 1.

Hint: Definere følgende heltallskonstanter i `playMastermind`:

- *SIZE (antall tegn i koden, 4).*
- *LETTERS (antall forskjellige bokstaver, 6).*

Dette gjør det enklere å forandre programmet senere, og lettere å lese koden din. Merk deg at alle konstanter bør skrives med store bokstaver (mens variabler består av flest små)

b) Lag følgende to tabeller i `playMasterMind`:

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

Siden brukeren skal prøve å gjette det som står i `code`, er det naturlig at begge tabellene har samme størrelse, dvs plass til SIZE char-elementer, samt en 0, slik at de kan skrives ut som C-strenger.

La den definerte størrelsen på `code` og `guess` være SIZE + 1, og sørg for at det siste elementet i tabellene er 0

c) Bruk funksjonen `randomizeCString` til å fylle `code`-tabellen med tilfeldige bokstaver mellom 'A' og 'A' + LETTERS.

Siden vi i Mastermind jobber med tegn, kan vi utnytte det at en tabell med tegn kan være en C-streng for å forenkle printing:

d) Prøv å skrive ut tabellene med `cout` i `playMasterMind`.

e) Bruk `readInputToCString` til å spørre spilleren etter SIZE bokstaver, og lagre dem i guess

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

g) Skriv funksjonen `checkCharacters` som skal returnere hvor mange riktige bokstaver spilleren har gjettet. (Uavhengig av posisjon).
Svaret skal returneres som et heltall.

Hint: Det er flere måter du kan implementere checkCharacters. En måte er å telle antall 'A'er i både `code` og `guess` tabellene. Antall riktig gjettede 'A'er er da det laveste antallet 'A'er i `code` eller `guess`. Ved å gjøre det samme for 'B', 'C' og så videre, kan man få det totale antall riktig gjettede bokstaver.

Har vi skrevet en funksjon før i øvinga som kan gjenbrukes?

h) Utvid koden din fra e) slik at programmet spør spilleren etter en ny kode så lenge `checkCharactersAndPosition` returnerer et tall mindre enn SIZE.

i) (5%) Sett sammen alle funksjonene i funksjonen fra a).
Når du er ferdig skal programmet lage og lagre en tilfeldig kode som spilleren kan gjette på inntil han finner den rette koden. For hver kode spilleren gjetter skal programmet skrive ut hvor mange rette bokstaver spilleren gjettet, og hvor mange av dem som var på rett plass.

Hint: Ved å skrive ut `code` i starten av programmet ditt vil det bli lettere å teste og feilsøke i koden din.

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

k) Utvid koden din til å gratulere brukeren med seieren (eller trøste med tapet), og spørre brukeren om han/hun vil spille en runde til