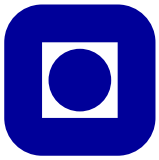

Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

TDT4102 Prosedyre- og objektorientert programmering Vår 2016

Øving 4

Frist: 2016-02-12

Mål for denne øvingen:

- 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 oppgaven
- 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 øvingen skal gjøres i ett Visual C++/XCode-prosjekt, men legg merke til oppdelingen i separate filer.

1 Call-by-Value versus pekere (15%)

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

a) Hva vil verdien til variabelen `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);
    std::cout << "v0: " << v0
              << " increment: " << increment
              << " iterations: " << iterations
              << " result: " << result << std::endl;
}
```

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

Hvis du ønsker kan du kopiere inn innholdet fra `utilities.cpp` fra Øving 3, om du har den tilgjengelig.

c) Opprett en ny fil `tests.cpp` med tilhørende headerfil, og legg til funksjonen `testCallByValue()` i `tests.cpp`. Denne funksjonen skal teste at `incrementByValueNumTimes()` virker, gjerne ved å kjøre den samme koden som ligger i `main()` i deloppgave a). Kall `testCallByValue()` fra `main()` (gjerne fra en meny slik vi gjorde i Øving 2). Funksjonen skal ikke ta inn noe, og heller ikke returnere noe.

d) Skriv om (endre) funksjonen `incrementByValueNumTimes`, slik at den ved å bruke pekere, endrer på `v0`. Funksjonen skal nå ikke returnere noe som helst. Husk å endre på `testCallByValue()` slik at den tar hensyn til at `incrementByValueNumTimes` er blitt endret.

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.

2 Tabeller (arrays) (10%)

Tabeller brukes for å lagre data av samme type i et sammenhengende minneområde. I denne oppgaven skal vi se på håndtering av tabeller.

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

b) Lag en tabell `percentages` i `testTablesSorting()`-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 må ta to argumenter.

d) Lag en funksjon `randomizeArray` i `utilities.cpp` som tar inn en tabell og størrelsen på tabellen, for så å fylle tabellen med tilfeldige prosentverdier (0-100, inklusive grensene).

Bruk gjerne funksjoner du har skrevet i tidligere øvinger.

Legg også til et kall til denne funksjonen i `testTablesSorting()`, slik at tabellen nå har innsatte verdier, skriv den endrede tabellen ut fra `testTablesSorting()`.

e) Bruk `swapNumbers` i `testTablesSorting()` til å bytte om de to første elementene i tabellen.

Siden tabeller og pekere i stor grad fungerer på samme måte, kan vi gjenbruke `swapNumbers`-funksjonen vi lagde tidligere på tabellen.

3 Sortering (10%)

I denne oppgaven skal vi se på sortering av tabeller.

a) (10%) Skriv en funksjon `sortArray` i `utilities.cpp` som tar inn en tabell og størrelsen til denne, og sorterer den.

Du skal her implementere din egen sorteringsalgoritme, og får ikke lov til å bruke sorteringsfunksjonen fra standardbiblioteket til C++. Det finnes et stort antall ulike algoritmer for sortering. I denne oppgaven kan du for eksempel implementere insertion sort eller selection sort.

Test funksjonen på tabellen `percentages` i `testTablesSorting()` fra Oppgave 2.

b) Skriv en funksjon `medianOfArray` i `utilities.cpp` som returnerer medianen til tabellen.

Funksjonen skal anta at tabellen den kjøres på *allerede er sortert*. Funksjonen skal fungere for vilkårlige tabellstørrelser (du må altså ta høyde for både odde antall elementer, og partall antall elementer).

Test funksjonen på tabellen `percentages` i `testTablesSorting()` før og etter den sorteres. Får du forskjellige svar?

4 Char-tabeller (C-strenger) (20%)

En C-streng er en tabell av verdier av typen `char` der siste element er 0 (`'\0'`). Dette gjør det enklere å bruke tabellen som en faktisk tekststreng, siden vi kan vite hvor teksten slutter.

Det er viktig å huske på å sette av plass til den avsluttende 0-en i tillegg til de tegnene vi vil ha i tabellen, hvis vi ønsker at tabellen skal bli behandlet som en C-streng.

a) Lag en ny funksjon `testCStrings()` 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 `testCStrings()`. 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 i løpet av de 2 semestrene året varer).

c) **Skriv funksjonen `randomizeCString` i `utilities.cpp`. Denne skal ta inn en char-tabell, et antall tegn, samt en øvre og nedre grense for tegn tabellen skal fylles med (f.eks. `'A'` og `'G'`), og skal fylle tabellen med tilfeldige tegn mellom og inklusive grensene.** Pass på at siste element i tabellen fortsatt er 0 (slik at vi har en C-streng).

Hint: Gjenbruk `randomWithLimits` fra øving 3.

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 `testCStrings()` uten å bruke noen form for løkke.**

f) **Skriv en funksjon `readInputToCString` i `utilities.cpp`. Denne skal la brukeren skrive 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. lagrer `'G'` i tabellen dersom øvre grense er `'F'`).

Bruk den innebygde funksjonen `toupper` når du sjekker om et tegn er innenfor grensene, slik at brukeren både kan skrive store og små tegn.

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

h) **Lag en ny heltallstabell `gradeCount` i `testCStrings`, 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 tilsvare 1, B tilsvare 2 etc, slik at snittet kan beskrives med tall).

i) **Utvid koden i `testCStrings()` 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 bruke 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.

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

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

Dette fungerer fordi en `char` i bunn og grunn bare er en tallverdi. Samsvaret mellom tegn og tall finner du i en ASCII-tabell.

5 Mastermind (45%)

I denne deloppgaven 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 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 det benyttes tabeller og fornuftig valgte funksjoner.

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

Hint: Definer 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 det er konvensjon at konstanter skrives med store bokstaver (mens vanlige variabler stort sett skrives med 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. Med andre ord skal de ha plass til `SIZE` antall `char`-elementer i tillegg til 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` - 1).

Siden vi i Mastermind jobber med tegn, kan vi bruke C-strenger til å forenkle utskrift til skjerm:

d) Prøv å bruke `std::cout` til å skrive ut tabellene i `playMasterMind`.

e) Bruk `readInputToCString` til å spørre spilleren etter `SIZE` antall 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 på. 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 øvingen som kan gjenbrukes her?

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) 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 dem etter tapet), og spørre brukeren om han/hun vil spille en runde til.