Linköpings Universitet
Institutionen för datavetenskap (IDA)
UPP-gruppen 2023-08-28

Uttryck, satser och strömhantering

Mål

I denna laboration ska du lära dig:

- Uppgift 0.1:
  - Arbeta med variabler
  - Formaterad inläsning och utskrift
- Uppgift 0.2:
  - Vanliga repetitions- och villkorssatser
  - Syftet med funktioner, parameteröverföring och funktionsanrop
  - Skapa egna datatyper
  - Hantering av listor och sortering
  - Läsa data från en fil

Läsanvisningar

- Uppgift 0.1:
  - Skriva, kompilera och köra huvudprogrammet `main`
  - Inkludering av standardbibliotek (`#include`)
  - Användning av standard namnrymd (`using namespace std`)
  - Datatyper (`int, char, double, std::string`)
  - Variabeldeklaration, initiering (`{}`), och tilldelning (=)
  - Formaterad inmatning (`cin, >>, get, getline, ignore`)
  - Formaterad utmatning (`cout, <<, endl`)
- Uppgift 0.2:
  - Styrsatser - villkor och loopar (`if, for, while, do-while`)
  - Aritmetiska operatorer (`+,-,*, /, \%`)
  - Formaterad utmatning (`setprecision, fixed, setw, setfill`)
  - Grundläggande användning av funktioner (Deklaration, definition, anrop, parameteröverföring och returvärden)
  - Aggregat (`struct`)
  - Filströmmar och strängströmmar (`fstream, stringstream`)
  - Grundläggande databehållare (`vector`)

Frågor

Frågorna är indelade i två kategorier: förberedelse och reflektion. Förberedelsefrågorna är bra att söka svaren till innan ni påbörjar laborationen, då den kunskapen kommer att hjälpa er i ert arbete.

Reflektionsfrågorna är menade att besvaras efter laborationen är genomförd. När vi har skrivit kod, och speciellt när vi avslutat en labb och bara vill komma vidare, är det svårt att lyfta blicken för att betrakta vad vi gjort och varför. Dessa frågor (ett segment som återkommer i varje lab) är ett tillfälle att just fundera kring vad vi gjort och försöka förstå varför. Ifall ni inser att dessa frågor inte reflekterar den kod ni skrivit så är detta också ett bra tillfälle att fundera över varför ni inte använt koncepten eller verktygen som nämns i frågorna och vad dessa kan tillföra. (Detta är också ett ypperligt tillfälle att gå tillbaka till sin kod och genomföra förbättringar, så kallad refaktorering.) Frågorna ger förhoppningsvis insikt i hur och varför vi skriver vår kod och gör att vi kan arbeta ännu bättre i framtiden.

Förberedelsefrågor:

- Vad är skillnaden mellan cin » och cin.get()?
- Vad innebär det att utföra en ”flush” på utmaningsströmmen och varför behöver vi göra det?
- Vilka typer av loopar har vi och när är de bäst att användas?
- Hur ser ett funktionsanrop ut? Kan de se olika ut, förutom till namn, och vad beror det på?

Reflektionsfrågor:

- Vad händer med inmatningsströmmen när användaren matar in en datatyp som inte överensstämmer med variabeln vi läser in till?
- Vissa formatteringar ”hänger kvar” tills nästa utskrift. Vilka gör det?
- Vad händer när vi dividerar två heltal med varandra? Vad händer när vi delar ett heltal med en reellt tal?
- Varför använder vi funktioner?
- Finns det en oändlig loop?
- Hur returnerar vi flera värden från en funktion?
- Varför vill vi gruppera ihop data till ett aggregat?

Kompilering med kursens kompileringsflaggor

I kursen förväntar vi oss att ni inte har några fel eller varningar som kompilatorn kan upptäcka åt oss. De flaggor vi använder är `std=c++17 -Wall -Wextra -pedantic -Weffc++ -Wold-style-cast`. För att t.ex. kompilera filen `uppgift01.cc` skriver du i terminalen:

```
g++ -std=c++17 -Wall -Wextra -pedantic -Weffc++ -Wold-style-cast uppgift01.cc
```

Uppgift 0.1

Skriv ett program som hämtar data från tangentbordet (inmatning från användaren av ditt program) och skriver ut det enligt körexemplen på kommande sidor. Det som syns på skärmen när man kör programmet ska överensstämma till fullo med givna exempel. Det gäller även antal blanktecken mellan ord och tal.

Ett tips är att du gör en liten del i taget och ser att den fungerar för det första körexemplet och sedan går vidare med nästa problem. Detta för att slippa hantera många möjliga fel vid kompilering. När ni känner er klara med ett problem kan ni kommentera bort koden för den delen, för att slippa göra samma inmatning igen. (Kom ihåg att sedan köra flera delproblem i följd för att se ifall de påverkan varandra.) Kommentarer i C++ kan skrivas på två olika sätt:

```
// Två stycken "/" kommenterar en rad till radens slut.
// Precis som denna raden och raden ovan.

/* Ett stycke som inleds med "/*" och avslutas med "*" följt av "/" kommenterar
   all kod som är däremellan. Precis som detta stycket */
```

OBS! Ditt program ska fungera för båda körexemplen, dock behöver du inte klara av alla möjliga varianter av indata då detta är i princip omöjligt i detta läge. Med det menar vi att ni inte behöver hantera att användaren matar in en sträng när det är en integer som programmet förväntar sig att ta emot.

KRAV: Du får endast använda en variabel av respektive datatyp och totalt fyra datatyper. När vi skriver kod har vi också krav på hur vi skriver kod, detta gäller genom hela kursen. Kravet finns för att det är viktigt att göra kod lättläslig, lätt att förstå och för att undvika fel. Vid bedömning utgår vi från detta bedömningsprotokoll. Innan inlämning, gå igenom protokollet och er kod. Ni behöver inte beakta rekommendationer kring konstruktioner och koncept ni inte använt ännu (klasser, standardbibliotek, iteratorer, minnesläcka, undantag, typkonvertering).

Programkörningar uppgift 0.1

Programmet ska ge resultat enligt följande exempel vid olika körningar. Användarens indata markeras här med ***fet kursiverad*** stil. Observera att eventuella inmatningar som kommer efter begärd inmatning skall ignoreras (vilket är tydligt i Körexempel 2).

Körexempel 1

```
Skriv in ett heltal: 10
Du skrev in talet: 10

Skriv in fem decimaltal: 12.1 30.2 27.3 13.4 11.5
Du skrev in talen: 12.1 30.2 27.3 13.4 11.5

Skriv in ett heltal och ett flyttal: 67 3.141592
Du skrev in heltalet:         67
Du skrev in flyttalet:    3.1416

Skriv in ett flyttal och ett heltal: 3.141592 67
Du skrev in heltalet:---------67
Du skrev in flyttalet:----3.1416

Skriv in ett tecken: T
Du skrev in tecknet: T

Skriv in ett ord: Kalle
Du skrev in ordet: Kalle

Skriv in ett heltal och ett ord: -5 Nisse
Du skrev in talet |-5| och ordet |Nisse|.

Skriv in ett tecken och ett ord: A Ria
Du skrev in "Ria" och 'A'.

Skriv in en rad text: Hej världen
Du skrev in: "Hej världen"

Skriv in en till rad text: Hej igen världen
Du skrev in: "Hej igen världen"

Skriv in tre ord: Testar ett program
Du skrev in: "Testar", "ett" och "program"
```

Körexempel 2

```
Skriv in ett heltal: 10.3
Du skrev in talet: 10

Skriv in fem decimaltal: -12 10330    27    13   11 fem
Du skrev in talen: -12 10330 27 13 11

Skriv in ett heltal och ett flyttal: 3267   3.141592 d
Du skrev in heltalet:       3267
Du skrev in flyttalet:    3.1416

Skriv in ett flyttal och ett heltal: 67 3.141592
Du skrev in heltalet:----------3
Du skrev in flyttalet:---67.0000

Skriv in ett tecken: Tomas
Du skrev in tecknet: T

Skriv in ett ord: 123Kalle
Du skrev in ordet: 123Kalle

Skriv in ett heltal och ett ord: 12är det lunch
Du skrev in talet |12| och ordet |är|.

Skriv in ett tecken och ett ord: oklart
Du skrev in "klart" och 'o'.

Skriv in en rad text:   här     skriver vi lite mer
Du skrev in: "  här     skriver vi lite mer"

Skriv in en till rad text: lite     mera text    här
Du skrev in: "lite     mera text    här"

Skriv in tre ord: Testar       ett   program
Du skrev in: "Testar", "ett" och "program"
```

Uppgift 0.2 - Löparlistan

Du ska skriva ett program som låter användaren skriva in namnet på en fil vars innehåll är resultatet av en löpartävling. Resultaten av tävlingen ska läsas från filen, sorteras så att den snabbaste löparen hamnar först, och slutligen skrivas ut i en tabell där de N antal snabbaste löparna visas.

För att lösa uppgiften ska du definiera ett aggragat som representerar en löpare. Varje rad i filen innehåller information om en löpare på följande format:

```
[namn] [timmar] [minuter] [sekunder]
```

Läs in den informationen och skapa en instans av aggregatet för varje rad i filen. Alla löparna sparas i en vektor som sedan ska sorteras och skrivas ut.

KRAV: Inläsning från fil och sortering ska ske i funktioner som anroppas från huvudprogrammet.

Körexempel 3

```
Ange filnamn: UPPloppet_resultat.txt
Ange antal rader: 5

   Namn   |   Tid
==========================
     Joan | 0:06:38
     Mika | 0:10:29
     Tory | 0:14:19
    Ziggy | 0:14:33
    Sandy | 0:31:20
```

TIPS: För att läsa in informationen från filen, läs in en rad i taget till en sträng. Skapa sedan en strängström med den raden i, och läs ut varje bit data från strängströmmen.

TIPS: Sortering får utföras med valfri algoritm. Nedan finns kod för sortering av heltal genom bubblesort, vilket kan användas som inspiration.

```
vector<int> v{9,8,7,6,5,4,3,2,1};
for (unsigned int i{}; i < v.size() - 1; ++i)
{
  for (unsigned int j{}; j < v.size() - i - 1; ++j)
  {
    if (v.at(j) > v.at(j + 1))
    {
      swap(v.at(j), v.at(j + 1));
    }
  }
}
```

Bonusuppgift: Felhanterad inläsning (6 poäng)

I uppgift 0.2 har vi inte implementerat någon felhantering. Nu ska du förbättra din lösning så den hanterar två typer av fel. Dels ska programmet kunna detektera och återhämta sig när användaren matar in något oväntat. Ett exempel är när användaren matar in bokstäver där programmet förväntar sig siffror. Dessutom ska programmet kontrollera att varje lyckad inläsning har ett rimligt värde. Om programmet inte kan slutföra sin uppgift på ett väldefinierat sätt ska ett nytt värde efterfrågas.

Körexempel 4

```
Ange filnamn: 123inteenfil.txt
FEL: Filen gick inte att öppna!
Ange filnamn: UPPloppet_resultat.txt
Ange antal rader: abc
FEL: Inmatningen måste vara ett heltal!
Ange antal rader: -1
FEL: Det finns inte -1 rader i filen.
Ange antal rader: 123
FEL: Det finns inte 123 rader i filen.
Ange antal rader: 5

   Namn   |   Tid
==========================
     Joan | 0:06:38
     Mika | 0:10:29
     Tory | 0:14:19
    Ziggy | 0:14:33
    Sandy | 0:31:20
```

TIPS: Målet är att programmet **alltid** (oavsett inmatning) ska lyckas läsa in alla värden som behövs för att skriva ut en korrekt tabell. Det är okej att tolka inmatningar av typen ”12skräp” som heltalet 12. [1]

KRAV: Kontroll ska göras att varje individuell inläsning lyckas genom att testa om strömmen `cin` är “sann”, om inmatningsförsöket (t.ex. `cin >> x`) är “sant” eller genom att läsa av strömmens felflaggor. Du måste både nollställa felflaggorna och “läsa bort” den felaktiga inmatningen innan nytt inläsningsförsök kan lyckas. Försök till ny inmatning ska fortgå tills användaren lyckas mata in godkänd data.

KRAV: Felsäker inläsning av ett tal ska skrivas som en funktion som används på lämpligt sätt.

[1] Det finns några kontrollsekvenser (Ctrl-c avbryter programmet, Ctrl-z pausar programmet i bakgrunden, Ctrl-d avbryter inmatningen) som ni inte behöver fundera på, det är meningen att de ska fungera som vanligt. Om användaren matar in någon obskyr kombination med ”Ctrl” så får programmet krascha eller bete sig hur ni vill.