TDDC76 Programmering och datastrukturer

Introduktionslaboration

Denna introduktionslaboration introducerar programmeringsspråket C++, hur du skriver enkla C++-program samt grundläggande begrepp. Det är också en introduktion till C++-miljön för GNU GCC-kompilatorn g++.

De olika övningarna omfattar bl.a. följande:

- hur du skriver enkla program i C++
- hur du översätter källkod till objektkod och körbar kod med kompilatorn g++
- hur du kör program
- hur du använder standardbiblioteket för C++, bl.a. in- och utmatning med strömbiblioteket
- exempel på uttryck och operatorer
- exempel på satser (**if**, **for**)
- exempel på funktioner och funktionsparametrar
- hur du hittar information

Ett mycket viktigt syfte med denna laboration är att ta upp grundläggande begrepp

I förklaringarna till exemplen tas en del begrepp upp och du förväntas lära dig dessa i samband med denna laboration. Du kan behöva läsa på innan de kommande föreläsningarna, om de korta förklaringar som ges här inte gör att du förstår helt.

Målet är att hinna med dessa uppgifter vid det första laborationstillfället. Skulle du inte göra det ska du ändå börja med de ordinarie laborationena (Laboration 1) vid nästa schemalagda laborationstillfälle och i stället göra resterande uppgifter på egen hand dessförinnan.

Redovisning

Det är ingen redovisning av denna introduktionslaboration.

Ordinarie laborationer

Vid kursens andra schemalagda laborationstillfälle ska du börja med de ordinarie laborationsuppgifterna. Dessa hittar du på kursens webbsidor, via **Laborationer** i vänstermenyn. Du bör förbereda Laboration 1.1–1.3 till detta laborationstillfälle.

1. Ett enkelt program

Starta Emacs, ange även filnamnet program1.cc på kommandoraden. Emacs hamnar då automatiskt i C++-mod, vilket underlättar mycket. Skriv följande program på filen och spara.

```
/*
 * Program som skriver ett meddelande på standard utmatningsfil.
 */
#include <iostream>
using namespace std;

int main()
{
  cout << "C++ is so cool!" << endl;
  return 0;
}
```

Kompilera filen med följande kommando i ett terminalfönster:

```
g++ -std=c++11 program1.cc
```

Flaggan `-std=c++11` behövs inte alltid men lika bra att göra det till en vana (default för g++ är fortfarande standarden C++98). Kör programmet med kommandot (när det kompilerar utan fel och du erhåller filen a.out):

```
a.out
```

Vad kan man lära sig av detta, förutom hur man kompilerar och kör ett vanligt C++-program?

- ett program består alltid av en eller flera **funktioner**, i detta fall en enda funktion som heter `main`.
- ett program ska alltid ha en funktion som heter main och det är i `main` som **exekveringen**, körmimgen av programmet, startar.
- `main` ska alltid returnera **int**, ett heltalsvärde, 0 betyder att allt har gått väl.
- Att inkludera `<iostream>` innebär att ”importera” en del av **standardbiblioteket** som kallas **strömbiblioteket**; detta innehåller funktionalitet för att läsa och skriva s.k. **strömmar**.
- Standard utström för utskrifter heter `cout`. När man skriver på `cout` med utskriftsoperatorn <<, skrivs resultatet ut i det terminalfönster där programmet körs.
- `endl` tillhör också strömbiblioteket och är en **manipulator** som har den ”manipulativa effekten” att generera ny rad i utskriften.
- /*…*/ är en **kommentar**, där /* inleder och */ avslutar. All text däremellan utgör kommentar och är endast till för mänskliga läsare (det finns en del hjälpprogram som också läser kommentarer).

Alla namn i standardbiblioteket är kapslade i **namnrymden** `std`. En namnrymd är en slags modul som kapslar in de deklarationer som ingår i namnrymden så att de inte är direkt synliga. Ett sätt att göra namnen i en namnrymd synliga i ett program är att öppna namnrymden med direktivet **using namespace**.

Städa filkatalogen och kompilera sedan om programmet med kommandona:

```
clean
g++ -std=c++11 -o program1 program1.cc
```

Det körbara programmet erhålls nu på en fil med namnet `program1`. Kör programmet!

Frågor och uppgifter

- Ändra i programmet så det skrivs ut en tom rad både före och efter meddelandet som skrivs ut.

2. Variabler och deklarationer

Kopiera filen program1.cc till en ny fil med namnet program2.cc. Ändra enligt följande:

```
// Program som skriver ett meddelande på standard utmatningsfil.
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string message{"C++ is so cool!"};
  cout << '\n' << message << "\n\n";
  return 0;
}
```

Kompilera programmet så att det körbara programmet får namnet `message` och provkör (i detta fall krävs flaggan -std=c++11).

Vad kan man få ut av detta program?

- Vi har ett exempel på den andra formen av kommentar, **radslutskommentar**. En sådan inleds med `//` och avslutas då raden tar slut.
- Inkluderingen <`string`> görs därför att **datatypen** `string` används i programmet för att deklarera en variabel som heter `message`. `string` tillhör standardbiblioteket.
- En **variabel** är en namngiven behållare för värden; en variabel är ett slags **objekt**.
- En **datatyp** anger en **värdemängd** och vilka **operationer** som är tillåtna på entiteter av typen ifråga.
- string är en datatyp för objekt som kan innehålla strängar, dvs en följd av tecken, i detta fall av typen **char**.
- "C++ is so cool!" är ett uttryckligt strängvärde. Sådana uttryckliga värden kallas **litteraler**.
- Alla variabler i ett program måste **deklareras**. Variabeln message deklareras i detta fall som en **lokal variabel** i funktionen main. messages **räckvidd** är inom main; den är inte är åtkomlig utanför main.
- message **initieras**, dvs ges ett värde direkt i samband med deklarationen, vilket man ofta bör göra. Det, i detta fall, sämre alternativet är att först deklarera variabeln och sedan **tilldela** den ett värde i en efterföljande **tilldelning**, vilket görs med **operatorn** =.

```
string message;
message = "C++ is so cool!";
```

- Det finns några olika syntaktiska former för initiering. Nedan visas tre varianter för initieringen av message, vilka ger samma resultat (den tredje har lite annan semantik men ger samma resultat).

```
string message{"C++ is so cool!"};    // nytt i C++11
string message("C++ is so cool!");    // kallas direktinitiering
string message = "C++ is so cool!";   // kallas kopieringsinitiering
```

- Variabler kan ibland initieras automatiskt. Det finns regler för i vilka fall variabler initieras automatiskt och med vilket värde, i princip ett ”nollvärde” för datatypen i fråga. Variabler av typen `string` initieras alltid automatiskt till tomma strängen, som motsvarar **stränglitteralen `""`**.
- Observera, att initiering och tilldelning är två helt skilja saker, även om = förekommer även i syntaxen för koperingsinitiering!

Frågor och uppgifter

- Vilken effekt har teckenlitteralen ’\n’ (s.k. *escape-sekvens*) då den skrivs ut?
- Ändra i programmet så att utskriften blir ”Message of the day: C++ is so cool!”, utan att ändra värdet för variabeln message.

3. Inmatning via tangentbordet

Städa filkatalogen, kopiera föregående program till en fil med namnet program3.cc och ändra till följande:

```
#include <iostream>
#include <string>
using namespace std;

const string message{"C++ is so cool!"};

int main()
{
  string first_name;

  cout << "Skriv ditt förnamn: ";
  cin >> first_name;

  cout << '\n' << first_name << " tycker \"" << message << "\"\n\n";

  return 0;
}
```

Kompilera och provkör programmet. Prova olika sätt att mata in namnet, t.ex. att skriva mellanrumstecken före namnet och även att slå ENTER-tangenten innan du skriver namnet.

Vad är nytt i detta program?

- Vi har gjort `message` till en **konstant** genom att ange **const** före datatypen `string`.
- En konstant måste alltid initieras och kan sedan inte ändras.
- Konstanten `message` har deklarerats utanför funktionen `main`, **deklareras på filnivå**. Genom detta är dess räckvidd potentiellt hela programmet. Skulle det finnas fler funktioner och även på andra filer som ingår, skulle `message` kunna kommas åt från samtliga funktioner; `message` är **global**.
- Vi har deklarerat en ny variabel `first_name`, för att lagra ett förnamn som ska läsas in.
- `cin` är standardströmmen för inmatning och är kopplad till tangentbordet.
- för att läsa data från `cin` använder vi **inläsningsoperatorn** `>>` i för att läser in namnet till variabeln `first_name`. Detta är ytterligare ett sätt att ge en variabel ett (nytt) värde.
- Operatorn >>, liksom <<, gör så kallad **formaterad utskrift**, respektive **formaterad inläsning**. I detta fall innebär det vid läsningen: 1) först läses eventuella inledande **vita tecken** (mellanrum, tabulertecken, radslut) i inströmmen förbi, 2) sedan läses tecken för tecken och lagras i variabeln till dess ett vitt tecken dyker upp eller vi nått slutet av inströmmen (filslut).
- Vi har delat upp koden i `main` i fyra avsnitt med hjälp av tomma rader: variabeldeklaration, inläsning av förnamnet, utskrift av meddelandet, samt avslutning (**return**). Detta ökar läsbarheten, genom att vi enkelt kan urskilja de olika delmomenten.

Frågor och uppgifter

- Vilken effekt har escape-sekvensen `\"` i en stränglitteral?
- Ändra i programmet så att du kan mata in både ditt för- och efternamn i var sin variabel. Låt programmet skriva ut dessa snyggt i meddelandet.
- Prova lite olika sätt att skriva in ditt för- och efternamn, t.ex. på samma rad med ett eller flera mellanrumstecken före, mellan och efter namnen; på olika rader, etc., för att testa den formaterade inmatningen.

4. Funktioner och funktionsparametrar

Skriv följande program på en fil med namnet program4.cc, kompilera och provkör.

```
#include <iostream>
#include <iomanip>
using namespace std;

void print_squares(int n)
{
  cout << '\n' << setw(6) << "i" << setw(12) << "i * i" << "\n\n";

  for (int i = 1; i <= n; ++i)
  {
    cout << setw(6) << i << setw(12) << i * i << '\n';
  }
}

int main()
{
  int x;

  cout << "Ge ett positivt heltal: ";
  cin >> x;

  if (x < 1)
    cout << "Det var inget positivt heltal!\n";
  else
    print_squares(x);

  return 0;
}
```

Vad är nytt?

- Vi har ett program som består av två funktioner, `main` och `print_squares` som **anropas** i `main`.
- Returtypen **void** innebär att funktionen `print_squares` inte returnerar något värde alls.
- **Parametern** `n` till `print_squares` har typen **int**, en **heltalstyp** som tillhör de **enkla datatyperna**.
- Manipulatorn `setw`(*fältvidd*) bestämmer **fältvidden** för den efterföljande utskriften.
- Vi måste inkludera `<iomanip>` då vi använder **parametriserade manipulatorer**, som `setw`, däremot inte för oparametriserade manipulatorer, som `endl`.
- **for**-satsen är ett exempel på en **iterationssats.** Den används ofta för att stega igenom en följd av värden, som i detta fall stega styrvariabeln `i` från 1 till värdet i parametern `n`.
- **if-else**-satsen är ett exempel på en **selektionssats**. Sådana används för att styra vilken väg program-exekveringen ska ta, i detta fall om en felutskrift ska göras eller om `print_squares` ska anropas.
- Det villkorsuttryck som står inom parentesen efter **if** styr exekveringen: om det blir *sant* (**true**) utförs satsen efter **if**, om det blir *falskt* (**false**) utförs satsen efter **else**.
- I **if**-satsens **else**-gren finns anropet av `print_squares`, där variabeln `n` ges som argument till parametern `n`. I detta fall kopieras `x`:s värde till parametern `n` i samband med anropet.
- {…} som förekommer som funktionskropp och i **for**-satsen är en **blocksats** som grupperar andra satser.

Frågor och uppgifter

- Provkör med ett stort heltalsvärde, t.ex. 999999, och kolla utskriften noga. Kan du förklara resultatet?
- Ändra i `print_squares`, så att utskrift endast görs om parametern `n` är mindre än eller lika med ett visst värde, till exempel 10000.

5. Kommandoradsargument

Skriv följande program och spara på filen program5.cc (<iostream> behövs också).

```
int main(int argc, char* argv[])
{
  if (argc == 1)
  {
    cout << "C++ is so cool!\n";
  }
  else
  {
    for (int i = 1; i < argc; ++i)
      cout << argv[i] << '\n';
  }
}
```

Kompilera programmet och provkör det på två sätt:

```
program5
program5 C++ är så koolt!
```

Vad är nytt?

- `main` har i detta fall försetts med två parametrar, `argc` och `argv`. Genom dessa överförs värden till `main` från exekveringssystemet. Det är definierat i C++ att `main` kan ha just dessa två parametrar.
- Via parametern `argc` (”*argument count*”) och `argv` (”*argument vector*”) kan programmet komma åt text som skrivits på kommandoraden, inklusive själva programnamnet.
- `argc` är ett heltal som anger hur många ”ord” som skrivits på kommandoraden.
- Varje ”ord” finns som en separat C-sträng (se nedan) i fältet `argv`.
- Ett **fält** (eng. ”array”) är en sammansatt datatyp där elementen är av samma typ, i detta fall **char***.
- Endimensionella fältvariabler definieras på formen: *elementdatatyp fältnamn*[*dimension*]
- När man deklarerar ett fält, som den formella parametern `argv`, behöver man inte ange någon **dimension** och det skulle dessutom inte kunna användas på något sätt.
- De enskilda elementen i ett fält åtkoms via **indexering** med **indexeringsoperatorn** `[]`.
- Första index i ett fält är alltid 0 (det förklarar uttrycket ”`i < argc`” i **for**-satsen; de `argc` stycken ”orden” på kommandoraden finns på index 0 till `argc`-1 i fältet `argv`).
- **char*** är en primitiv strängdatatyp från C. **char*** betyder egentligen typmässigt ”pekare till tecken” men normalt pekar en sådan pekare (en adress) på det första tecknet i en sträng av typen **char**[], ”fält av tecken”, där det på slutet finns strängavslutningstecknet ’\0’ (nulltecken); en s.k. C-sträng.
- **char***/**char**[] är lågnivåsträngtyper och en ökänd källa till fel i C- och C++-program. Dessa har i C++ kompletterats (läs gärna ersatts) med den fullfjädrade strängtypen `string`.

Frågor och uppgifter

- Vad innebär det att `argc` är lika med 1 när man kör ett program?
- Vad finns i `argv[0]`?
- Ändra i programmet, så att alla kommandoradsargument man ger skrivs ut på en enda rad, med ett mellanrumstecken mellan varje argument.
- Ändra så att alla kommandoradsargument skrivs ut på en enda rad men i omvänd ordning.