EDAF30 – Programmering i C++

1. Introduktion

Sven Gestegård Robertz
Datavetenskap, LTH

2015

EDAf30: Programmering i C++, 7.5 hp

Kursens syfte är att ge kunskaper i objektorienterad programmering i C++.

Mål:

- Kunskap och förståelse
  - kunna förklara grundläggande begrepp inom objektorienterad C++-programmering
  - kunna tolka, analysera och förklara befintlig C++-kod.
- Färdighet och förmåga
  - kunna utveckla ett fungerande C++-program från en given specifikation
  - kunna felsöka metodiskt i C++-kod.

EDAf30: Programmering i C++, 7.5 hp
Obligatoriska moment

Kursen examineras genom

- laborationer
- inlämningsuppgifter
- skriftlig tentamen

Slutbetyget baseras på den skriftliga tentamen.

EDAf30: Programmering i C++, 7.5 hp
Administration

- Kursprogram
- Kurregistrering
- Anmälan till laborationer

Historik
C++ härstammar från Simula och C.

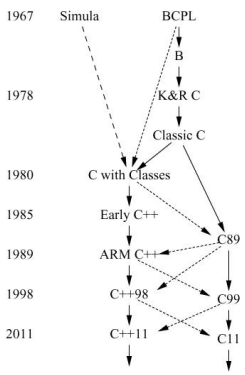
1967: Simula (Dahl & Nygaard)
1972: C (Dennis Ritchie)
1978: K&R C (Kernighan & Ritchie)
1980: C with Classes (Bjarne Stroustrup)
1985: C++ (Bjarne Stroustrup)

- ISO-standard 1998

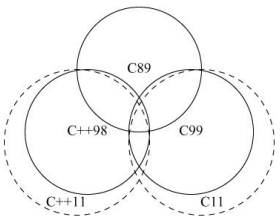
Andra släktingar:

1995: Java (James Gosling et al.)
2000: C# (Anders Hejlsberg)

- virtuell maskin
- automatisk minneshantering
- *säkra* språk



C++ är inte en ren utökning av C



- både ISO C och ISO C++ härstammar från K&R C, och är "syskon"
- Vissa detaljer är inkompatibla mellan C och C++
- Ytorna är inte skalenliga

Generellt: skriv inte C++ som om det vore C

Ett första program i C++
Hello, World!

hello.cpp

```
#include <iostream>
int main( )
{
   std::cout << "Hello, World!" << std::endl;
   return 0;
}
```

hello.cpp

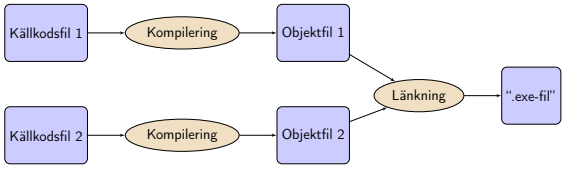
```
#include <iostream>
using namespace std;

int main( )
{
   cout << "Hello, World!" << endl;
   return 0;
}
```


Vad är ett program?

C++ är ett kompilerat språk

- Källkod
- Objektfil
- Exekverbar fil



Vad är C++?

ISO-standarden för C++ definierar två saker

- *Språket* C++ (*Core language features*), t ex
  - datatyper (t ex **char**, **int**)
  - kontrollflödesmekanismer (t ex **if**- och **while**satser).
- *Standardbiblioteket* (*Standard-library components*), t ex
  - Datastrukturer (t ex `string`, `vector` och `map`)
  - Operationer för in- och utmatning (t ex `<<` och `getline()`)
  - Algoritmer (tex `find()` och `sort()`)
  - Deklarationerna *inkluderas* med t ex **#include** `<vector>`

Datatyper
Primitiva typer

- Heltalstyper: **char**, **short**, **int**, **long**, **long long**
  - **signed** (som i Java)
  - **unsigned** (icke-negativa tal, finns inte i Java)
- Flyttalstyper: **float**, **double**, **long double**
- **bool** (`boolean` i Java)
- Pekare (heltalstyp, aritmetik, till skillnad från i Java)
- Referenser (konstanta och kan inte vara "`null`")[1]

[1] Egentligen inte en typ, utan ett *alias* till ett objekt

Datatyper
Sammansatta typer

- Arrayer ("fält", "vektorer". Som i Java, fast
  - osäkra
  - C-strängar är **char**`[]`
  - kan ha godtycklig typ som element
- **struct**, **class** (som klass i Java. Medlemsvariabler kallas ibland "fält")
- `std::string` (liknar `java.lang.String`)
- `std::vector`, `std::list` ... (liknar motsvarande i `java.util`)

Variabler

- Några begrepp:
  - en *deklaration* introducerar ett *namn*
  - en *typ* definierar mängden möjliga värden och operationer (för ett *objekt*)
  - ett *objekt* är ett stycke minne som innehåller ett *värde*
  - ett *värde* är en följd bitar som ska tolkas enligt en viss *typ*.
  - en *variabel* är ett namngivet *objekt*
  - En variabel deklarerad **const** får inte ändras (`final` i Java)
  - En variabel deklarerad **constexpr** måste ha ett värde som kan beräknas vid kompilering.
- Regler för variabler:
  - En variabel måste *deklareras*
  - En variabel bör *initieras* (En konstant måste *initieras*)

Variabler
Deklaration och initiering

Deklaration utan initiering (undvik)

```
int x;
```

Deklaration och initiering

```
int x{7};     // Rekommenderad stil i C++
int y = {7};  // C++ med extra =
int z = 7;    // C-stil
```

C-stil: Se upp med implicit typkonvertering

```
int x = 7.8;  // x == 7. Ger ingen varning
int y {7.8};  // ger varning (eller fel med -pedantic-errors)
```


Variabler
Automatisk typinferens

`auto`: Man behöver inte ange typen på en variabel om den kan härledas från initieringen.

Deklaration och initiering

```
auto x = 7;     // int x
auto c = 'c';   // char c
auto b = true;  // bool b
auto d = 7.8;   // double d
```

NB! med `auto` finns ingen risk för felaktig typkonvertering, så det är säkert att använda tilldelning (=).

Använd inte `auto` om du behöver vara explicit med typdeklarationen, t ex om

- att ange typen gör koden mer lättläst
- man vill bestämma talområde eller precision (t ex `int`/ `long` eller `float`/ `double`)

Arrayer ("C-arrayer","fält")

Deklaration utan initiering

```
int x[7];   // innehåller odefinierade värden
```

Deklaration och initiering av `int`-array

```
int a[7] {0, 1, 2, 3, 4, 5, 6};
int b[20] {0, 1, 2, 3}; // resten av elementen initieras till 0
```

Utelämnande av längden

```
int b[] {1, 1, 0, 1, 0, 0, 1};
```

Initiering med = som i C

```
int a[7] = {0, 1, 2, 3, 4, 5, 6};
int b[]  = {1, 1, 0, 1, 0, 0, 1};
```


C-arrayer
tilldelning, djup kopiering

```
int a[7] = {0, 1, 2, 3, 4, 5, 6};
int b[7] = {1, 1, 0, 1, 0, 0, 1};
```

Ej tillåtna tilldelningar

```
b = a;                      //"invalid array assignment"
b = {1, 1, 0, 1, 0, 0, 1}; //"assigning from an initializer list"
```

Korrekt men omständligt

```
for (int i=0; i != 7; ++i)
  b[i]=a[i];

int i=0;
for(int d : {1, 1, 0, 1, 0, 0, 1})
  b[i++] = d;
```


Operatorer

Gemensamma med Java

```
+ - * / % ++ -- += -= *=  ...
```

Trinära villkorsoperatorn (som i Java)

```
z = (x>y) ? x : y;

if (x>y)
   z=x;
else
   z=y;
```

Många fler, t ex

Medlems- och pekaroperatorer: `[] * & . ->`
In- och utmatning: `<< >>`

Konsoll-I/O
Programexempel

In- och utmatning

```
#include <iostream>

using namespace std;

int main( ) {
   string namn;

   cout << "Vad heter du? ";
   cin >> namn;
   cout << "Goddag, " << namn << "!" << endl;

   return 0;
}
```

Manipulatorer

Fil att inkludera

```
#include <iomanip>
```

Antal positioner

```
... << setw(4) << antal << ...
// Gäller bara nästa utskrift (antal)
```

Antal decimaler

```
... << fixed << setprecision(2) << a << ... << x
// Gäller tills nästa setprecision
```


Inmatning

Läs och returnera nästa tecken

```
cin.get()
```

Läs ett `antal` tecken till C-strängen `str`

```
cin.getline(str, antal)
```

Returnera en kopia av nästa tecken (oläst)

```
cin.peek()
```

Läs nästa tecken och släng det

```
cin.ignore()
```


Filhantering

Fil att inkludera

```
#include <fstream>
```

Läsa från fil

```
ifstream f1("infil.txt");
f1 >> ord;
...
f1.close();
```

Skriva till fil

```
ofstream f2("utfil.txt");
f2 << namn << " " << telnr << endl;
...
f2.close();
```


Bibliotek

sin, cos, sqrt, log, exp, floor ...

```
#include <cmath>
```

rand, srand, abs, labs ...

```
#include <cstdlib>
```


Slumptal

`<cstdlib>`

Tärning

```
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int main( ) {
   int antal;
   cout << "Antal kast: ";
   cin >> antal;
   srand(time(0)); //Nytt startvärde varje gång
   for (int i=0; i<antal; i++) {
     cout << rand()%6+1 << " ";
   }
   cout << endl;
}
```


Slumptal

Bättre C++: kapsla slumpgenereringen i ett objekt

Anta att vi har en klass `Rand_int` som ger slumptal i ett intervall $[min, max]$.

med RandInt-objekt

```
int main()
{
  unsigned long seed = time(0);
  Rand_int dice{1,6, seed};
  int n{20};
  for(int i = 0; i != n; ++i) {
    cout << dice() << " ";
  }
  cout << endl;
}
```

C-varianten

```
int main( )
{
  unsigned int seed = time(0);
  srand(seed);
  int n{20};
  for (int i=0; i<n; i++) {
    cout << rand()%6+1 << " ";
  }
  cout << endl;
}
```

Slumptal
Exempel på slumptalsgenerator-klass

Slumptalsgeneratorn `Rand_int`

```
#include <iostream>
#include <random>

class Rand_int {
public:
    Rand_int(int low, int high) :dist{low,high} {}
    Rand_int(int low, int high, unsigned long seed)
            :re{seed}, dist{low,high} {}
    int operator()() {return dist(re);}
private:
    std::default_random_engine re;
    std::uniform_int_distribution<> dist;
};
```

Detaljerna i denna klass kommer att förklaras senare i kursen.

Vi har talat om

- Kursen
- C++
  - Historik
  - Datatyper och variabler
  - Jämförelser med Java och C
  - Något om I/O och standardbibliotek

Nästa föreläsning: Tecken och texter

Vi kommer att studera

- Tecken och teckenkodning
- Strängar, C-strängar och `std::string`
- Standardbibliotek och I/O