TDIU01 - Programmering i C++, grundkurs

Grundläggande satser och uttryck

Eric Elfving
Institutionen för datavetenskap

7 oktober 2015

LINKÖPINGS UNIVERSITET

Översikt

- Uttryck
  - Litteraler
  - Operatorer
- Satser
  - Villkor
  - Upprepning
- Teckenhantering

Creative Commons 4.0

Litteraler

- Litteraler är värden som skrivs direkt i källkoden.
- Litteraler konverteras till inbyggda typer med regler givna i språket.
- Exempel:

Litteral	Typ	Värde
`1`	`int`	`1`
`0x1b`	`int`	`27`
`1.2`	`double`	`1.2`
`1.2e2`	`double`	`120.0`
`'c'`	`char`	`'c'`
`"c"`	`string`	`"c"`
`"kalle"`	`string`	`"kalle"`

- Sista exemplen är inte helt rätt, egentligen får man en ¨`char *`¨. Återkommer till detta senare i kursen.

Creative Commons 4.0

Operatorer

Aritmetiska operatorer

- C++ har de vanligaste matematiska operatorerna inbyggt: +, -, *, / utför normalt sett det man väntar sig...
- Beräkningsordning precis som i matematiken, kan ändras med parenteser.
- Det finns ett undantag från första punkten, heltalsdivision!
  - Matematiskt: $\frac{a}{b} = c + \frac{d}{b}$
  - C++: $\frac{a}{b} = c \quad a\%b = d$
  - Två heltal (int) dividerade ger ett heltal (hur många gånger kan jag dela a med b?)
  - För att få resten används modulus (vanligtvis kallat mod), i C++ används operatorn %.

Creative Commons 4.0

Operatorer

Sammansatt tilldelning

- Man kan modifiera en variabel med sammansatt tilldelning
- `x += 3` ökar `x` med tre. Kan såklart även skrivas `x = x + 3`
- Fungerar för alla aritmetiska operatorer (och några fler)

Creative Commons 4.0

Operatorer

Stegning

- En vanlig operation är att öka eller minska en variabel med värdet 1
- Därför finns det operatorer för just detta: ++ och - -
- Finns i prefix- (++x) och postfix-versioner (x++)
- Båda ökar värdet med 1, varför finns det två?

Creative Commons 4.0

Operatorer

Prefix stegning

- Prefix stegning är den som används oftast
- Ökar värdet och resultatet är det nya värdet

```
int x {1};
cout << x << endl;
cout << ++x << endl;
cout << x << endl;
```

```
1
2
2
```

LINKÖPINGS UNIVERSITET

Creative Commons 4.0

Operatorer

Postfix stegning

- Ökar värdet och resultatet är det gamla värdet.
- `x++` motsvarar följande beräkning, resultatet är `tmp`:

```
int tmp {x};
x = x + 1;
```

```
int x {1};
cout << x << endl;
cout << x++ << endl;
cout << x << endl;
```

```
1
1
2
```


Creative Commons 4.0

Operatorer

Jämförelseoperatorer

- Man vill ofta jämföra värden, då använder man jämförelseoperatorer

Operator	Betydelse
==	Likhet
!=	Olikhet
<	Mindre än
>	Större än
>=	Större eller lika
<=	Mindre eller lika

- Alla ger logiska värden som svar (**`bool`**, `true` eller `false`)

Creative Commons 4.0

Operatorer

Logiska operatorer

- Med hjälp av logiska operatorer kan man slå ihop logiska värden (eller uttryck)

Operator	Betydelse
&&	Logiskt OCH
||	Logiskt ELLER
!	Logiskt ICKE

- Beräkningsordning: ! > && > || (> tolkas som ¨beräknas innan¨)

Creative Commons 4.0

Strömoperatorer

- Det finns två strömoperatorer: << och >>.
- Utskriftsoperatorn << används för att skriva ut ett värde på en ström. Värdet kan vara konstant, till exempel en litteral.
- Inläsningoperatorn >> används för att hämta ett värde från en ström till en variabel.
- Vi har tagit upp två grundläggande strömmar, `cin` för inläsning och `cout` för utskrift.
- Vi kommer prata mer om strömmar senare i kursen.

```
int x;
cout << "Mata in ett tal: ";
cin >> x;
```


Creative Commons 4.0

Satser

Creative Commons 4.0

Satser

- Det vi pratat om hittills kallas uttryck
- Ett C++-program består av satser som körs i en sekvens
- Ett programs satser körs alltid i den ordning de står i källkoden
- Satser består av uttryck
- Satser avslutas alltid med ett semikolon

Creative Commons 4.0

Satser

Deklarationer

- En variabel deklareras innan den kan användas.
- Ett namn får endast användas en gång i ett program.
- Vi får ett kompileringsfel om vi omdeklarerar en variabel.

```
int x {3};
int x {4};      // Ger ett fel!
x = 5;          // korrekt
double x {4.3}; // Fel!
```


Creative Commons 4.0

Satser

Uttryckssatsen

- Uttryckssatsen är den enklaste formen av sats.
- Uttryck följt av ett semikolon.

```
x++;
cout << x;
4;
```


Creative Commons 4.0

Satser

Satsblock

- Man kan när man vill skapa ett nytt block.
- Block används för att gruppera satser som hör bra ihop.
- Görs med klammerparenteser (`{ ... }`)
- Deklarationer som görs i ett block gäller endast där.
- Blocket skapar ett eget område (scope) där variabler lever och dör.
- Ett block ses som en sats.

```
int x {3}, y {5};
{
  int x {4};  // EJ fel, x skapas lokalt!
  int z;
  cout << x << ' ' << y; // skriver ut "4 5"
}
z = 6;  // Fel, z finns inte här!
```


Creative Commons 4.0

Satser

Villkor

```
if ( uttryck )
  sats
else
  sats
```

- `if`-satsen: ¨om sant, gör följande¨
- `uttryck` ska ge ett logiskt värde (bool) eller kunna konverteras enligt de vanliga reglerna.
- `else`: ¨annars, gör detta¨
- else kan utelämnas

Creative Commons 4.0

Satser

Villkor

```
if ( uttryck )
{
  sats1;
  sats2;
  ...
}
```

- Ibland räcker inte en sats i villkoret, då kan man använda block!
- Detta är det vanliga skrivsättet, använd alltid block för att slippa problem.

Creative Commons 4.0

Satser

Villkor

```
if ( uttryck1 )
  sats1
else if ( uttryck2 )
  sats2
else
  sats3
```

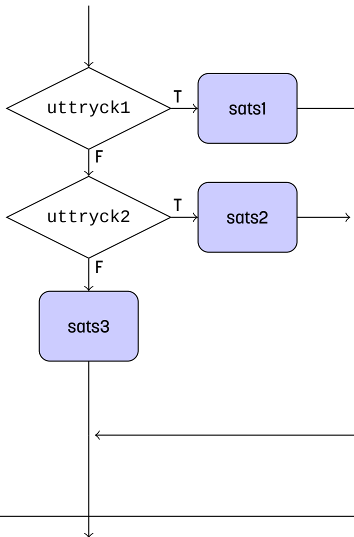
- Ibland vill man kontrollera flera villkor
- Byt ut else-grenens sats mot en ny if-sats!
- Väldigt vanlig operation - många språk har egna namn såsom elsif och elif
- Kan byggas på med valfritt antal ¨else if¨

Creative Commons 4.0

Satser

Villkor

```
if ( uttryck1 )
  sats1
else if ( uttryck2 )
  sats2
else
  sats3
```



BY NC
Creative Commons 4.0

Satser

Uppräkning

```
switch ( uttryck )
{
  case v1:
    sats1
  case v2:
    sats2
  default:
    sats3
}
```

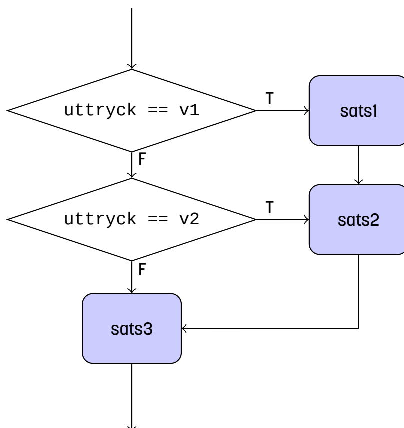
- Ibland vill man kontrollera värdet på ett uttryck (ofta en variabel)
- Det går bra med if-satsen men switch-satsen är specialgjord för detta
- Om ett värde hittas utförs alla satser nedanför detta. Det kallas att man ¨faller igenom¨ övriga villkor.

Creative Commons 4.0

Satser

Uppräkning

```
switch ( uttryck )
{
  case v1:
    sats1
  case v2:
    sats2
  default:
    sats3
}
```



Creative Commons 4.0

Satser

Uppräkning

```
switch ( uttryck )
{
  case v1:
  case v2:
    sats1
    break;
  case v3:
    sats2
    break;
  default:
    sats3
}
```

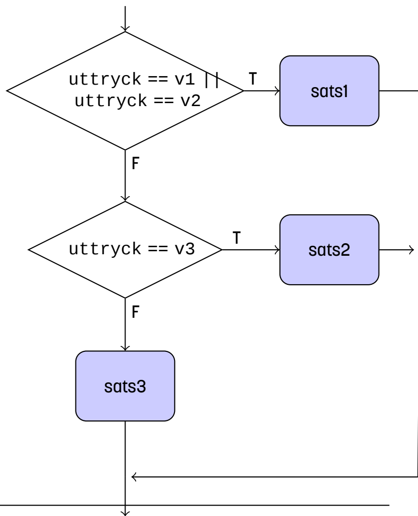
- För att utföra samma satser för flera värden har man flera case.
- Om man vill få switch-satsen att fungera mer som if, kan man använda sig av break.

Creative Commons 4.0

Satser

Uppräkning

```
switch ( uttryck )
{
  case v1:
  case v2:
    sats1
    break;
  case v3:
    sats2
    break;
  default:
    sats3
}
```



Creative Commons 4.0

Satser

Upprepning

- Ofta vill man upprepa kod
- C++ har tre upprepningssatser;
  - `while`
  - `do ... while`
  - `for`

Creative Commons 4.0

Satser

Upprepning - while

```
while ( uttryck )
{
  satser
}
```

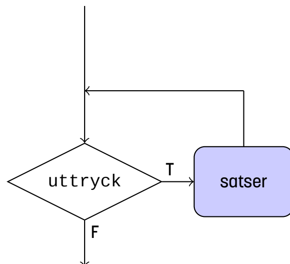
- `while` - den enklaste.
- Utför `satser` så länge `uttryck` är sant
- En förtestad upprepningssats

Creative Commons 4.0

Satser

Upprepning - while

```
while ( uttryck )
{
  satser
}
```



Creative Commons 4.0

Satser

Upprepning - do-while

```
do
{
  satser
}
while ( uttryck );
```

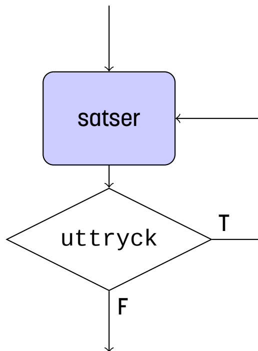
- `do-while` - en eftertestad while-loop.
- Utför `satser` minst en gång.

Creative Commons 4.0

Satser

Upprepning - do-while

```
do
{
  satser
}
while ( uttryck );
```



Creative Commons 4.0

Satser

Upprepning - for

```
for ( init; cond; end )
{
  satser
}
```

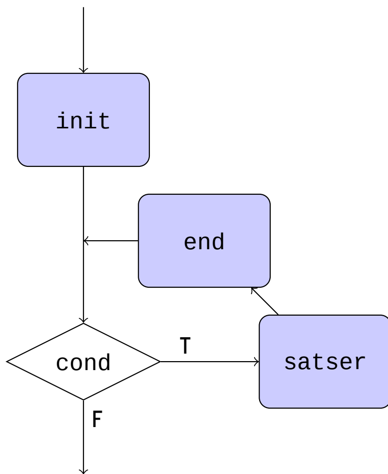
- `for` - En loop med ¨inbyggd¨ räknare.
- Utför `init` en gång, kör `satser` följt av `end` så länge `cond` är sant.

Creative Commons 4.0

Satser

Upprepning - for

```
for ( init; cond; end )
{
  satser
}
```



Creative Commons 4.0

Upprepning

Vilken ska jag välja

1. Ska jag upprepa oändligt många gånger ⇒ `for (;;)`
2. Vet jag hur många gånger det ska upprepas ⇒ `for`
3. Ska jag utföra det minst en gång ⇒ `do-while`
4. Annars ⇒ Det som känns bäst... (ofta `while`)

Creative Commons 4.0

Exempel

Creative Commons 4.0

Skriv ut talen 1..5


```
int i {1};
while ( i <= 5 )
{
  cout << i << endl;
  ++i;
}
```

```
int i {1};
do
{
  cout << i << endl;
  ++i;
}
while ( i <= 5 );
```

```
for ( int i {1};
      i <= 5; ++i )
{
  cout << i << endl;
}
```


Creative Commons 4.0

Räkna antal ¨vita¨ tecken

- Alla inläsningsoperationer ger ett resultat som kan tolkas som om det gick bra att läsa, detta kan man utnyttja i en while-loop:

```
#include <iostream>
#include <cctype>
using namespace std;
int main()
{
  char c;
  int space {};
  while ( cin.get(c) )
  {
    if ( isspace(c) )
    {
      ++space;
    }
  }
  cout << "Du matade in " space << " vita tecken."
}
```

LINKÖPINGS UNIVERSITET

Creative Commons 4.0

cctype

- I inkluderingsfilen `<cctype>` finns teckenhanteringsfunktioner
- Samtliga tar emot ett tecken och ger 0 som svar vid falskt, nollskiljt vid sant.
- Endast garanterat att de fungerar som väntat för engelska tecken (ASCII)

Namn	Funktion
`isaplpha(c)`	c är ett alfabetiskt tecken
`isblank(c)`	c är ett vitt tecken (tab och mellanslag)
`isspace(c)`	`isblank` eller nyrad
`isdigit(c)`	c är ett numeriskt tecken
`isupper(c)`	c är en versal
`islower(c)`	c är en gemen

Creative Commons 4.0 BY NC

www.liu.se