TDIU01 - Programmering i C++, grundkurs

Underprogram - Funktioner

Eric Elfving
Institutionen för datavetenskap

7 oktober 2015

LINKÖPINGS UNIVERSITET

Översikt

- Återblick till satsblocken
- Funktioner - Namngivna satsblock
- Parametrar
  - Värdeöverföring
  - Referenser
  - const-referenser
- Returvärden
- Inför labben - slumptal med `<random>`

Satsblock

- En grupp av satser som hör bra ihop
- Omges av klammerparenteser
- Ses som en sats i språket
- Har ett eget ¨universum¨ (scope) med egna lokala deklarationer
- Kan dela värden med utomstående mha globala variabler

Satsblock

- Vad ska jag göra om jag vill utföra samma satser flera gånger?
  - Upprepningssatser går så klart - men bara om de ska upprepas direkt efter varandra
  - Om man vill ha koden på olika platser i programmet är funktioner bättre!

Funktioner

Ett kort exempel

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

void hello();  // deklaration
int main()
{
  hello();     // Anrop
  cout << "------" << endl;
  hello();
}
void hello()   // definition
{
  cout << "Hejsan" << endl;
}
```

```
Hejsan
------
Hejsan
```

Parametrar

- Med hjälp av parametrar kan man skicka data till sin funktion

```
void hello(string name) // deklaration + definition
{
  cout << "Hello " << name << '!' << endl;
}
int main()
{
  hello("Anna");
}
```

```
Hello Anna!
```

Parametrar

Värdeöverföring

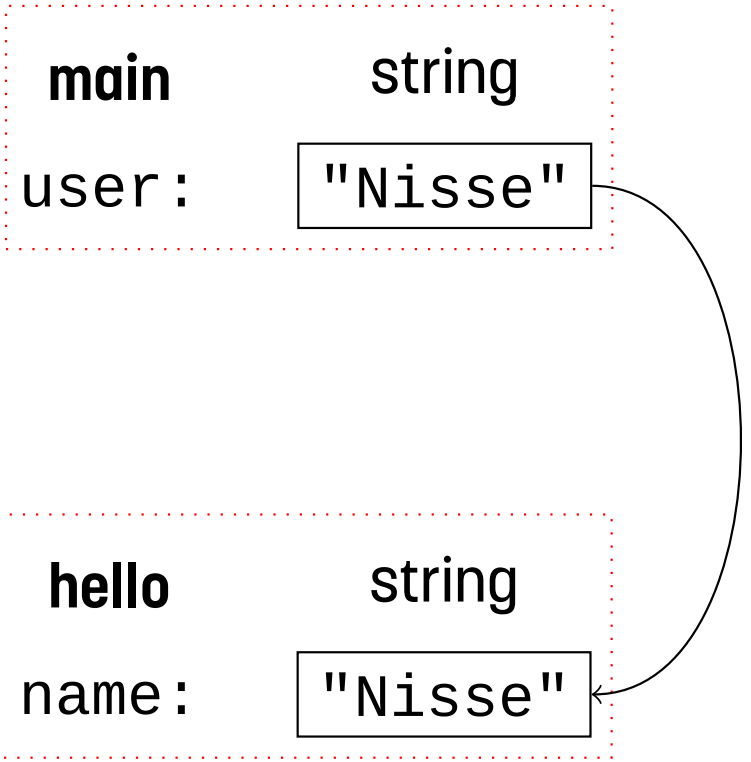
- Detta var exempel på värdeöverföring
- Värden kopieras vid överföring - en lokal kopia skapas i funktionen och kan användas som en lokal variabel
- Eventuella ändringar vi gör på parametern syns inte utifrån
- Kan också deklareras const för att förbjuda förändringar

Parametrar

Värdeöverföring

```
void hello(string name);

int main()
{
  string user {"Nisse"};
  hello(user);
}
```



Parametrar

Referensparametrar

- Om vi vill kunna ändra på vår anropares variabler får vi ta emot parametrarna som referenser
- En parameter deklareras som en referens med hjälp av &

```
void swap(int & a, int & b)
{
  int c {a};
  a = b;
  b = c;
}
int main()
{
  int v1 {2};
  int v2 {5};
  swap(v1, v2);
  cout << v1 << '\n' << v2 << endl;
}
```

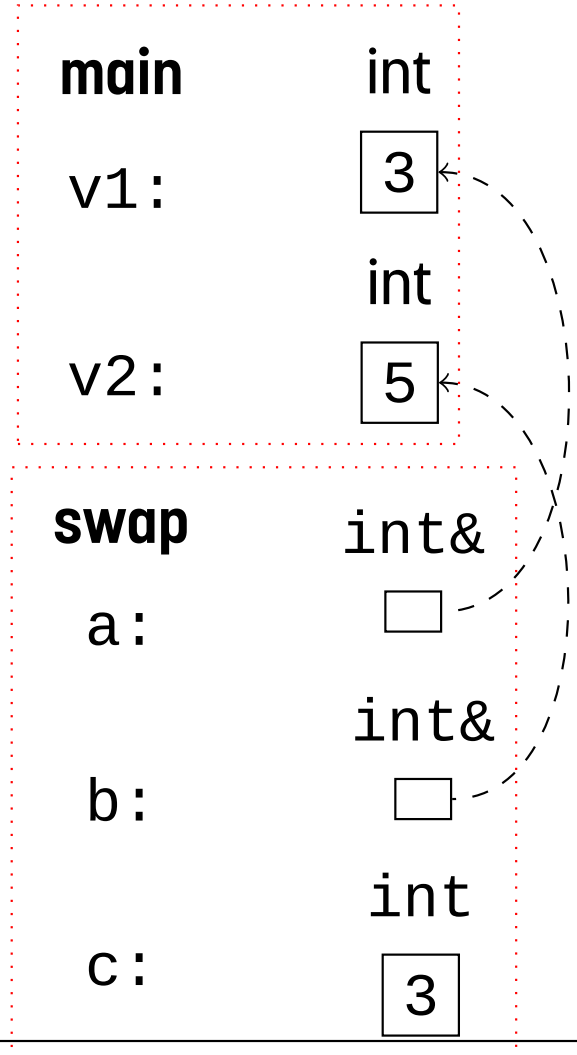
```
5
2
```

Parametrar

Referensparametrar

```
void swap(int & a, int & b);

int main()
{
  int v1{3}, v2{5};
  swap(v1, v2);
}
```



Parametrar

const-referenser

- Värdeöverföring leder till (ofta onödig) kopiering
- referensöverföring gör att vi kan ¨råka¨ ändra på anroparens variabler
- Dessutom kräver referenser att värdet går att ändra på (fungerar inte med konstanter eller litteraler)
- Därför kan man deklarera sin parameter som const-referens!
- Bra att ha vid överföring av stora variabler såsom strängar som man inte vill ändra på
- Kompilatorn kommer ge fel om vi råkar ändra på parametern

```
void hello(const string & name)
{
  cout << "Hello " << name << '!' << endl;
}
```

Parametrar

Vilken ska jag välja?

1. Vill jag ändra på parametern? $\Rightarrow$ referens
2. Är det en inbyggd datatyp (`int, double, char, bool`)?
   - 2.1 Vill jag kunna ändra på den lokalt? $\Rightarrow$ värdeöverföring
   - 2.2 Annars $\Rightarrow$ const
3. Annars $\Rightarrow$ const-referens (blir vanligaste valet)

Defaultargument

- Om man vill kan man ange defaultvärden (skönsvärden) för sina parametrar
- Då kan funktionen anropas utan den parametern

```
void print_stars(const int n = 10)
{
  for ( int i {}; i < n; ++i )
  {
    cout << '*';
  }
  cout << endl;
}

int main()
{
  print_stars(20);
  print_stars();
}
```

```
********************
**********
```

Defaultargument

- Parametrar med defaultargument måste anges i slutet av parameterlistan

```
void fun(int a, int b=5, int c);
```

```
g++ -c args.cc
args.cc:1:6: error: default argument missing for parameter 3
  of 'void fun(int, int, int)'
```

Defaultargument


```
void fun(int a=1, int b=5, int c=4);
```

Anrop	a	b	c
f()	1	5	4
f(2)	2	5	4
f(3,4)	3	4	4
f(3,4,6)	3	4	6

Funktioner

- En funktion är ett namngivet satsblock
- Funktioner deklareras innan de kan anropas
- Liknar vanliga variabeldeklarationer, skrivs på formatet

```
returtyp funktionsnamn([parametrar]);
```

- Funktioner används ofta för att beräkna värden, skickar tillbaka resultat av typen `returtyp`
- Om man inte vill ge tillbaka värden väljer man typen void
- `[parametrar]` betyder att det inte behöver anges

Returvärden

- Alla funktioner vi skapat hittills har haft returtypen `void` för att symbolisera avsaknad av returvärde.
- Alla funktioner kan returnera ett värde
- I C++ görs det med return:
  ```
  return expr;
  ```
  där `expr` är ett uttryck med samma datatyp som funktionens returtyp.
- Funktionen avslutas direkt när return nås.
- En funktion kan ha flera return-satser, all kod efter den första nås inte.

Returvärden


```
int max(const int a, const int b)
{
  if ( a > b )
  {
    return a;
  }
  return b;
}
int main()
{
  int a,b;
  cout << "Mata in två tal: ";
  cin >> a >> b;
  int m { max(a,b) };
  cout << "Det största är "
       << m << endl;
}
```

```
Mata in två tal: 2 6
Det största är 6
```

Slumptal

<random>

- Inkluderingsfilen `<random>` är ny i C++11
- Det finns massor man kan göra, t.ex finns det stöd för olika fördelningar
- Baseras på typen `random_device`
- Ger ett slumptal av typen `unsigned int`

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

int main()
{
  random_device rnd;
  cout << rnd() << endl;
}
```

Hur får jag ett tal i ett givet intervall?

Slumptal


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

int rand(const int low, const int high)
{
  random_device rnd;
  return rnd() % (high - low + 1) + low;
}

int main()
{
  for (int i {}; i < 10; ++i)
  {
    cout << rand(2, 10) << endl;
  }
}
```

```
7
2
2
2
4
6
6
9
10
5
```

www.liu.se

LINKÖPINGS UNIVERSITET