TDIU01 - Programmering i C++, grundkurs

Datalagring - poster och vektorer

Eric Elfving
Institutionen för datavetenskap

7 oktober 2015

LINKÖPINGS UNIVERSITET

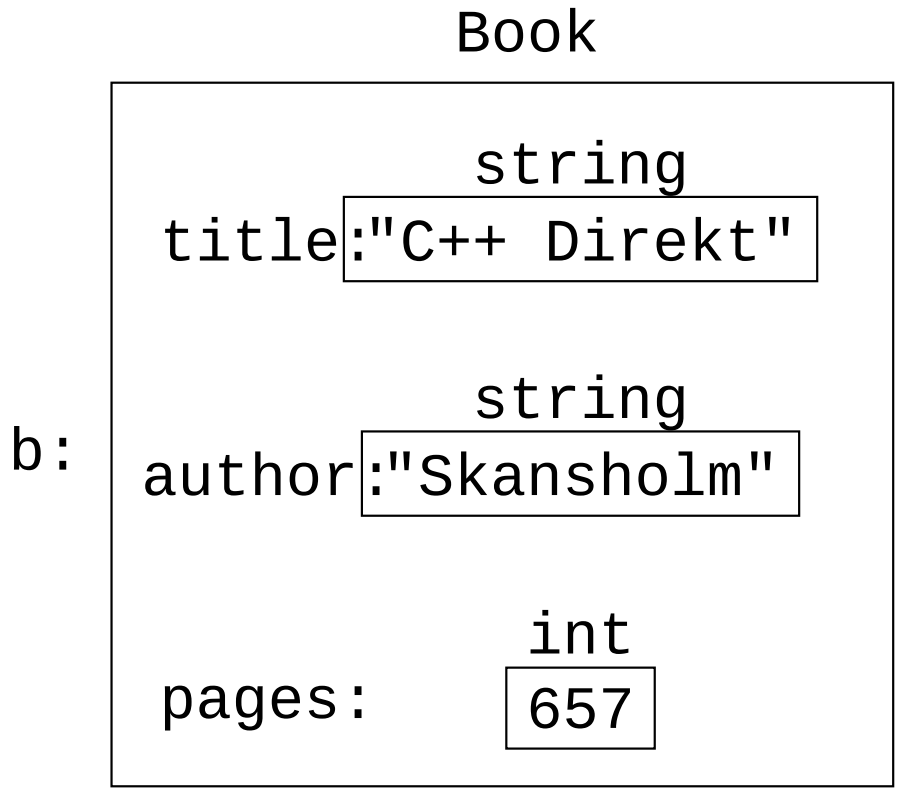
Poster


```
struct Book
{
  string title;
  string author;
  int pages;
};

int main()
{
  Book b;
  b.title = "C++ Direkt";
  b.author = "Skansholm";
  b.pages = 657;
}
```

- Man kan med hjälp av poster gruppera data som hör bra ihop till en datatyp.
- I C++ använder man sig av det reserverade ordet `struct` för att skapa en post.
- Posten kan sedan användas som en vanlig datatyp.
- En post är ett exempel på en sammansatt datatyp.
- Posten har namngivna fält som man kommer åt med punktoperatorn.

Poster


```
struct Book
{
  string title;
  string author;
  int pages;
};

int main()
{
  Book b;
  b.title = "C++ Direkt";
  b.author = "Skansholm";
  b.pages = 657;
}
```



Poster

Initiering och tilldelning

```
// Initiering, fälten fylls från vänster...
Book primer {"C++ Primer", "Lippman", 885};

Book b2 {"Professional C++"};
// författare och antal sidor får ett
// "nollvärde" (tom sträng resp. 0)

b2 = primer;      // Tilldelning, alla delar kopieras
```

Poster

Poster och funktioner

- Precis som andra datatyper kan man skicka poster till funktioner.
- Vi har en ¨komplicerad¨ datatyp $\Rightarrow$ (const-)referens bör användas.

Poster

Poster och funktioner

```
int main()
{
  Book my_book;
  cout << "Mata in information om en bok" << endl;
  input(my_book);  // Låt användaren mata in information
  cout << "Inmatat data:" << endl;
  print(my_book);  // Skriv ut bokens information
}
```

Poster

Poster och funktioner

```
void print(const Book & b)
{
  cout << "Titel:       " << b.title  << '\n'
       << "Författare:  " << b.author << '\n'
       << "Antal sidor: " << b.pages  << endl;
}

void input(Book & b)
{
  cout << "Titel: ";
  cin >> b.title;
  cout << "Författare: ";
  cin >> b.author;
  cout << "Antal sidor: ";
  cin >> b.pages;
}
```

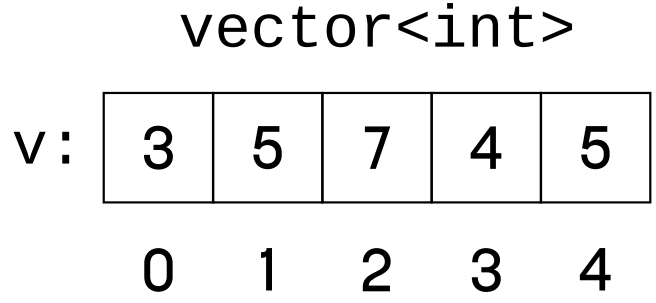
Vektorer

- Poster är bra om man har olika data som på något sätt hör bra ihop.
- Vektorer används om man har flera värden av samma datatyp som hör bra ihop.
- En vektor deklareras på formatet `vector<datatyp> namn`, där `datatyp` är den typ av värden man vill lagra i vektorn och `namn` är det namn man vill ha på sin vektor.

Vektorer

- En vektor sägs ha element som man kan komma åt genom indexering (varje element har en given position, ett index)
- Från början är vektorn tom, man kan lägga till värden med hjälp av `.push_back(värde)`

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

int main()
{
  vector<int> v;
  v.push_back(3);
  v.push_back(5);
  v.push_back(7);
  v.push_back(4);
  v.push_back(5);
}
```



Vektorer

- Föregående var jobbigt, vi kan initiera vår vector med värden:

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

int main()
{
  vector<int> v {3,5,7,4,5};
}
```

Vektorer

- Vi kommer åt element med hakparenteser eller `at`-funktionen

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

int main()
{
  vector<int> v {2,3,5,7};
  cout << v[0] << endl
       << v.at(1) << endl
       << v.at(2) << endl
       << v.at(3) << endl;

}
```

```
2
3
5
7
```

- `at` rekommenderas då den kontrollerar att elementet verkligen finns!

Vektorer

- Man vill ofta iterera över sin vektor (gå igenom alla element)
- Då är funktionen `size` tillsammans med en `for`-loop bra att ha

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

int main()
{
  vector<int> v {1, 5, 6, 9, 10};
  for ( int i {}; i < v.size(); ++i )
  {
    cout << v[i] << endl;  // eller v.at(i)
  }
}
```

Vektorer

- I C++11 introducerades ett nytt sätt att iterera över en vektor - den intervallbaserade for-loopen

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

int main()
{
  vector<int> vec {1, 5, 6, 9, 10};
  for ( int val : vec )
  {
    cout << val << endl;
  }
}
```

- `val` blir i detta fall en kopia av varje värde i `vec`

Vektorer

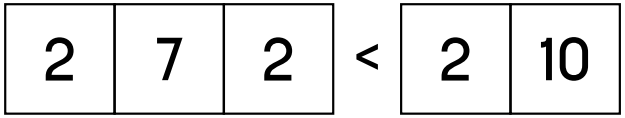
- Om man vill kunna ändra på värdet i vektorn kan man skapa referenser (som i funktioner)

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

int main()
{
  vector<int> vec {1, 5, 6, 9, 10};
  for ( int & val : vec )
  {
    ++val;
  }
}
```

Vektorer

Jämförelse

- Både vektorer och strängar jämförs elementvis.
- Strängjämförelse fungerar alfabetiskt (tänk ordlista) enligt teckentabellen ('A'<'a')
- Vektorjämförelse fungerar så länge typen som lagras kan jämföras.



Vektorer och strängar

- Vektorer och strängar har många likheter
- Följande funktioner fungerar för både vektorer och strängar (`x` kan bytas ut mot en variabel av typen `string` eller `vector<datatyp>`)

`x[i]`	ta fram element `i`
`x.at(i)`	ta fram element `i` (kontrollerad)
`x.size()`	Antal element i x
`x.clear()`	Töm x (ta bort alla element)
`x.empty()`	Ger true om x är tom

www.liu.se

LINKÖPINGS UNIVERSITET