

## Konstruktorer, destruktorer

- Hur skapar man objekt, och vad händer då
- Allt som skapas måste tas bort
- Överlagrade metoder
- Initiering med konstruktorer
- Destruktorer
- Initieringskonstruktor
- This-pekaren

2000-11-12

Thomas Johansson Datavetenskap

1

## Allokering/deallokering av objekt

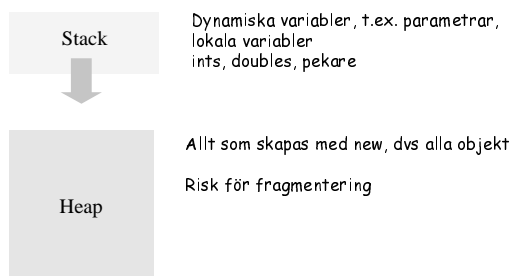
- Var allokeras objekten? Tre sätt:
  - Statiskt av kompilatorn
  - Dynamiska objekt på stacken
  - Eller på heapen med new
- Om new det enda sättet -> uniform metod
  - Pekare måste derefereras i C++
- C++ har alla sätten - och alla problemen
- Java har bara new (allt på heapen)

2000-11-12

Thomas Johansson Datavetenskap

2

## Stack och heap



2000-11-12

Thomas Johansson Datavetenskap

3

## Konstruktorer

- Samma namn som klassen
- Ingen returtyp
- Ofta överlagrade
- Anropas 'av new', och vid allokering av automatiska variabler
- Kan misslyckas

2000-11-12

Thomas Johansson Datavetenskap

4

## Exempel

```
class Bil
{
private:
    int hastighet;
public:
    Bil ()
    { hastighet = 0; }
};
```

2000-11-12

Thomas Johansson Datavetenskap

5

## En metod som alla andra

- Konstruktorn anropas när objektet skapas
  - Kan ta argument
- ```
Bil b; // Default
Bil b(10);

Bil *b = new Bil(50);
Bil *b = new Bil[50]; // Obs skillnaden !
Bil *b = new Bil[50](10); // Går ej !
```
- Finns ingen konstruktor anropas en 'default constructor', som inte gör någonting

2000-11-12

Thomas Johansson Datavetenskap

6

## Överlagrad konstruktor

```
class Bil
{
private:
    int hastighet;
public:
    Bil ()           // default
    { hastighet = 0; }
    Bil (int hast)  // med argument
    { hastighet = hast; }
};
```

## Initiering på 'kortform'

```
class Bil
{
private:
    int hastighet;
public:
    Bil () : hastighet(0) {}
    Bil (int hast) : hastighet(hast)
    { ... }
};
```

## Statiska objekt

- Dvs Globala objekt
  - Garanterat initierade före main startar
  - Ordningen ej förutsägbar -> problem !
  - Ett objekt kanske behövs av konstruktorn i ett annat, men är inte skapat ännu

```
#include "Bil.h"
#include "Passagerare.h"

Bil b;
Passagerare p;

void main ...
```

## Defaultvärden

- Kan användas av alla metoder, vanligt vid konstruktörer

```
class Bil
{
private:
    int hastighet;
public:
    Bil (int hast = 0)
    { hastighet = hast; }
};

Bil b;           -> ger hastighet = 0
Bil b(12);      -> ger hastighet = 12
```

## Återanvändning ?

- En konstruktor kan i princip anropa en annan, men det kan leda till problem
- Bättre är att låta defaultkonstruktorns arbete utföras av en annan metod
- På så vis kan alla konstruktörer dela på metoden

## Konstruktörer och arv

- En konstruktor i en subclass kan anropa basklassens konstruktor

```
class Fordon
{ private: int hastighet;
public:
    Fordon(int hast) : hastighet(hast){}
};

class Bil : public Fordon
{
public:
    Bil () : Fordon (0) {}
    Bil (int hast) : Fordon (hast)
    {cout << "hastighet " << hast;}
};
```

## Vi använder **new**, sen då ?

- Hur deallokera objekten ?
- -> **Explicit**
  - Problem med 'dangling pointers'
  - C++ har **delete**
- -> **Implicit**
  - Vi behöver *Garbage Collection* i någon form
  - Java har *GC*

2000-11-12

Thomas Johansson Datavetenskap

13

## Destruktorer

- Anropas när ett objekt tas bort
  - Antingen med **delete**
  - eller vid automatisk deallokering av lokala variabler
- Samma namn som konstruktorn men med ett Tilde (~) framför
- Inte lika mycket använda som konstruktorer, men en god regel är att alltid ha med en !

2000-11-12

Thomas Johansson Datavetenskap

14

## Exempel

```
class Bil
{
private:
    int hastighet;
public:
    Bil (int hast)
    ~Bil ()
};

Bil::Bil (int hast)
{ hastighet = hast; }
Bil::~Bil ()
{ }
```

2000-11-12

Thomas Johansson Datavetenskap

15

## Destruktorer forts

- Alla metoder som deklarerats i klassen måste ha en implementation, även om den är tom
- Vill man inte skriva en tom konstruktor så utlämna deklarationen -> default
- **Delete** används för att ta bort ett objekt som allokerats med **new**
- Använd utskriften i konstr/destr under utvecklingen !

2000-11-12

Thomas Johansson Datavetenskap

16

## Exempel

```
#include "Fordon.h"
#include "Bil.h"

int main ()
{
    Bil b;
    Bil *p = new Bil(10); // konstr *p
    ...
    delete p; // destr *p
} // destr b
```

2000-11-12

Thomas Johansson Datavetenskap

17

## Virtuell destruktör

- Har att göra med dynamisk bindning
- Gör att 'rätt' destruktör anropas

```
class Fordon
{ private: int hastighet;
public:
    Fordon(int hast) : hastighet(hast){}
    virtual ~Fordon()
};

class Bil : public Fordon
{ public:
    Bil () : Fordon (0) {}
    Bil (int hast) : Fordon (hast) {}
    virtual ~Bil ();
};
```

2000-11-12

Thomas Johansson Datavetenskap

18

## En sorts konstruktor till...

- Initieringskonstruktor (Copy konstruktor)
- Anropas när en kopia av ett objekt skall göras
- Kan anropas utan att du vet om det, t.ex. vid metodanrop
- Om 'djup kopiering' krävs är en initieringskonstruktor ett måste

## Exempel

```
class Bil
{ private:
  char* namn;
public:
  Bil () { namn = 0; }
  Bil (char* n)
  { namn = new char[sizeof(n) + 1];
    strcpy(namn, n); }
  Bil (Bil &b)
  { namn = new char[sizeof(b.namn) + 1];
    strcpy(namn, b.namn); }
  virtual ~Bil () { delete [] namn; }
};
```

## Forts.

- Obs initiering <> kopiering !

```
int main()
{
  Bil b ("bmw");
  Bil kopia(b); // initiering
  Bil c = b;    // initiering

  funk( b );   // kan bli init

  c = b;      // tilldelning
}
```

## This-pekaren

- Används om man behöver en pekare till det objekt som metoden 'körs i'
- Alla klassmetoder har en this-pekare som 'osynligt' argument
- Statiska metoder har ingen this-pekare

```
private:
  int hastighet;
public:
  Bil (int hastighet)
  { this->hastighet = hastighet; }
```