Effektiv Kode med C og C++

Forelesning 4, vår 2013

Alfred Bratterud

Agenda:

* Oblig 1!
* Litt repetisjon
  * Minne
  * Objekters levetid
  * Det "rare" fra lab'en
* Klasser
* Initialisering og "default constructor"
* Destructor og memory leaks

Oblig 1

* Oblig 1 kommer på github i dag. Den har frist:
  14.feb kl. 23:59
* Levering via github:
  * Mappen 'oblig1' - kan være tom, men må ligge i ditt PRIVATE github-repo, senest om 1 uke
  * Link på være levert på fronter også da
* 2 deloppgaver: wordcloud og string som tall.
* Konkurranse! De 3 beste kåres

Merkelig oppførsel fra peker til "const"

* Det at jeg ikke fikk "bus error" da jeg forsøkte å fjerne const med cast, skyldtes .rodata, som jeg foreslo

* At man ikke får endret en const ved å sette en peker til den, skyldes et subtilt triks i gcc. Den JUKSER!
* Assembly-kode, og gjennomgang ligger på github, for interesserte.

Litt repetisjon

Tre VIKTIGE typer minne

* Static Memory
  * Settes av "link-time"
  * Kan ikke forandre størrelse
  * Gjelder globalt- og namespace skop
* Dynamic Memory
  * Settes "runtime" med "new eller "malloc"
  * Frigjøres "runtime" med "free" eller "delete"
* Automatic Memory
  * Legges på stack
  * Gjelder alt i lokalt skop, uten "new"/"malloc"

Objekters levetid

Hentet fra Stroustrups "Programming", s.1048

* "Local (automatic) objects" konstrueres når de blir "støtt på" av en tråd, dør ved enden av skopet
* "Temporary objects" opprettes inni et sub-uttrykk og dør ved slutten av uttrykket
* "Namespace objects" lever så lenge programmet kjører
* "Local static objects" konstrueres når de blir støtt på, og dør når programmet dør.
* "Free store objects" opprettes med new og ødelegges med "delete"

Initialisering

* Initialisering er å gi en startverdi til en variabel
* Initialiserer man ikke, er innholdet "udefinert"
* Initialisering skal (må) alltid skje så tidlig som overhodet mulig.
* Initialisering er rett frem for "vanlige typer", litt mer komplekst for medlemmer, ille for arrayer.

OOP 1

Objektorientering i C++

Klasser

* Klasser er structer med medlemsfunksjoner, der medlemsvariabler er "private" som standard. Dette heter "enkapsulering"
* Medlemmer kan være av alle typer
* Syntaksen er rett slik:

```
class student{
  int nr;
  string name;
  public:
     int get_nr();
     string get_name();
}
```

* Nå er studentklassen "read-only"
* Men hvor er "kroppene" til funksjonene?

Interface v.s. Implementation

* "Interfacet" til en klasse (eller et bibliotek) består av deklarasjoner av alle medlemmene, men kun med "signaturene" til funksjonene
    * Denne ligger gjerne i en egen "header-fil" (Student.h)
* "Implementasjonen" ligger gjerne i en annen fil (Student.cpp), som "inkluderer" header-fila
* Hvorfor?
* Fordi da kan vi bruke ferdig-kompilerte versjoner
* ...og skifte ut ferdig-kompilerte "biter" uten å rekompilere alt.

Standard konstruktorer

* En konstruktor er en medlemsfunksjon med samme navn som klassen
* Med C++ har også primitive typer en standard konstruktor, så `int()` er gyldig, også `int(5)`.
* En standard konstruktor blir opprettet for deg, og tar ingen argumenter
* I egne konstruktorer er det egen notasjon for å initialisere medlemmer. Litt lettere i C++11.
* Man kan alltid ha flere konstruktorer, men bare i C++11 kan en konstruktor kalle en annen.

konstruktorer i klasser

* Konstruktoren har som jobb å initialisere alle medlemmer - ved å kalle deres konstruktorer.
* OBS: Lager du en konstruktor selv, uten argumenter, mister du standard konstruktoren.
* I alle egne konstruktorer må alle medlemmer initialiseres manuelt - evt. ved å kalle standard-konstruktor.
* Man må da initialisere slik:

```
class student{
  int nr;
  string name;
  public:
     student(string name): nr(1 0),name(name) { ... }
     int get_nr();
     string get_name();
};
```

Instansiering av klasser

```
class student{
  int nr;
  string name;
  public:
    student(string name): nr(10),name(name) { ... }
    student(): nr(10),name("N/A") { ... }
};
```

- Hvilken er riktig?

```
student s("Alfred");
new student s("Alfred");
student s;
student* s=(student*)malloc (sizeof(student));
```

Instansiering av klasser

*

```
class student{
  int nr;
  string name;
  public:
    student(string name): nr(10),name(name) { ... }
    student(): nr(10),name("N/A") { ... }
};
```

* Hvilken er riktig?

```
student s("Alfred");
new student("Alfred");
student s;
student* s=(student*)malloc (sizeof(student));
```

Destruktorer

* Destruktor har ansvar for å rydde opp, dvs. frigjøre det minnet klassen har satt av.
* Destruktoren til en klasse "myClass" heter "~myClass()
* Alle objekter har en "default destructor"
* ....og den fjerner sizeof(object)
* Hva vil vi typisk gjøre i en "destructor"?
* Hvorfor har vi ikke destructorer i java/php?

Nå: Demo!

"classes.cpp"
"default_constructors.cpp"
"destructors.cpp"