TDDE71

Pekare och dynamiskt minne

Eric Ekström & Klas Arvidsson

Institutionen för datavetenskap

LINKÖPING UNIVERSITY

1 Repetition - ADT
2 Speciella medlemsfunktioner
3 Livekod
4 Git
5 GDB
6 Make

LINKÖPING UNIVERSITY

ADT

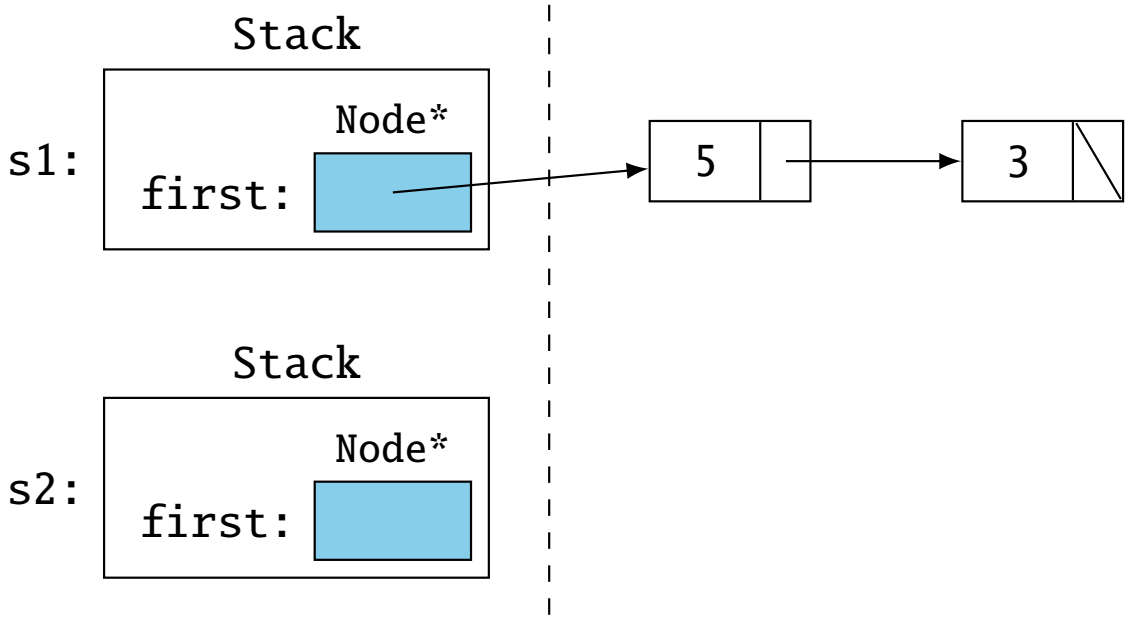
En klassdeklaration från lektionen.

```
class Stack
{
public:
  void push(int i);
  int  top() const;
  void pop();
  bool is_empty() const;

private:

  struct Node
  {
    int value;
    Node* next;
  };

  Node* first;
};
```

ADT

Vad händer om vi försöker kopiera stacken?

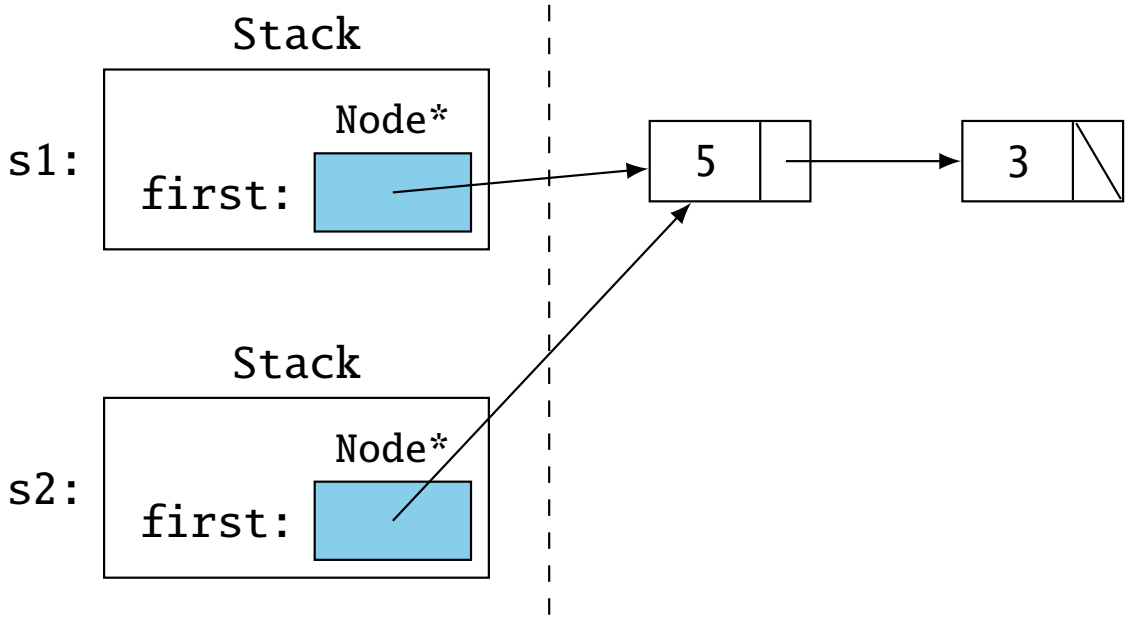
```
int main()
{
  Stack s1 {};
  s1.push(3);
  s1.push(4);

  Stack s2 {s1};
}
```



ADT

Vad händer om vi försöker kopiera stacken?

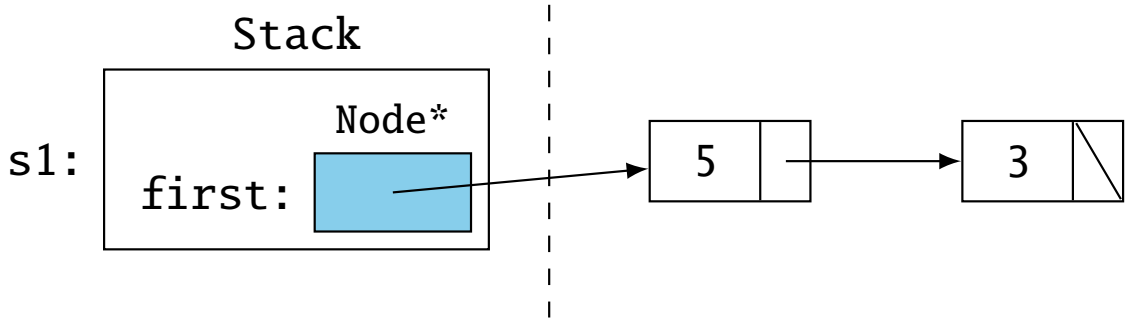
```
int main()
{
  Stack s1 {};
  s1.push(3);
  s1.push(4);

  Stack s2 {s1};
}
```



ADT

Vad händer när stacken går ur scope?

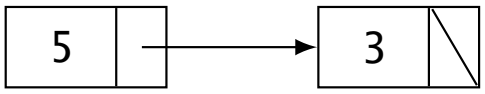
```
int main()
{
  Stack s1 {};
  s1.push(3);
  s1.push(4);

} //???
```



ADT

Vad händer när stacken går ur scope?

```
int main()
{
  Stack s1 {};
  s1.push(3);
  s1.push(4);

} //???
```



Minnesläcka!

1 Repetition - ADT
2 Speciella medlemsfunktioner
3 Livekod
4 Git
5 GDB
6 Make

LINKÖPING UNIVERSITY

Speciella medlemsfunktioner

Kompilatorn genererar ett antal medlemsfunktioner åt oss.

Kopieringskonstruktor	`X(X const& other)`
Kopieringstilldelningsoperator	`X& operator=(X const& rhs)`
Destruktur	`~X()`
Flyttkonstruktor	`X(X && other)`
Flytttilldelningsoperator	`X& operator=(X && rhs)`

Dessa är de speciella medlemsfunktionerna.

Speciella medlemsfunktioner

När behöver vi själva implementera dessa?

- Om objektet hanterar en unik resurs
  Tex en fil eller ström
- Om objektet inte får kopieras
- Om vi behöver göra något speciellt när objektet tas bort
- Generellt: om objektet ansvarar för dynamiskt minne

Speciella medlemsfunktioner

Kopieringskonstruktor

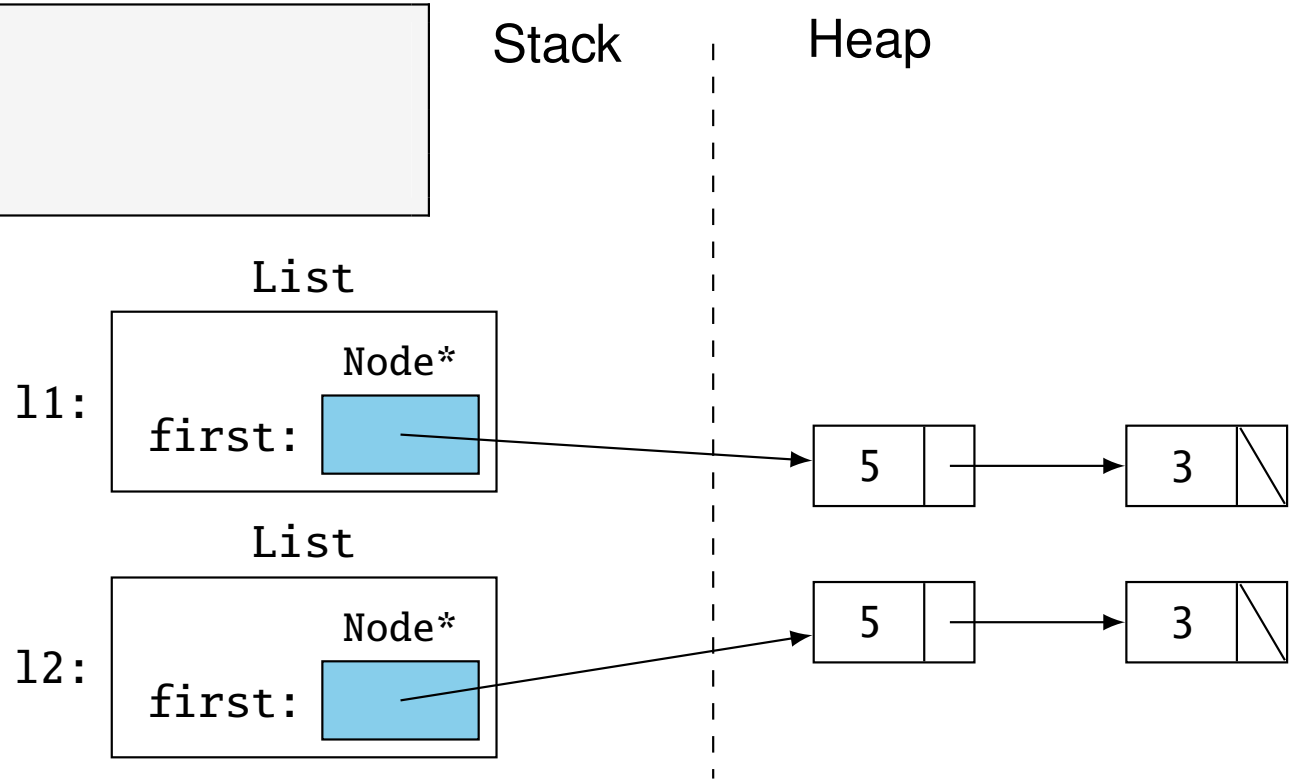
```
Foo(Foo const& other)
```

- Skapa ett nytt objekt utifrån ett annat objekt av samma typ.
- I fallet av länkade strukturer bör denna göra en djup kopia av det andra objektet.
- Anropas automatiskt i många lägen
  - Vid definition av nya variabler
  - När objekt returneras
  - Vid parameteröverföring

Speciella medlemsfunktioner

Kopieringskonstruktor

```
List l1 {};
l1.insert(3);
l1.insert(5);
List l2 {l1};
```



Speciella medlemsfunktioner

Kopieringstilldelningsoperator

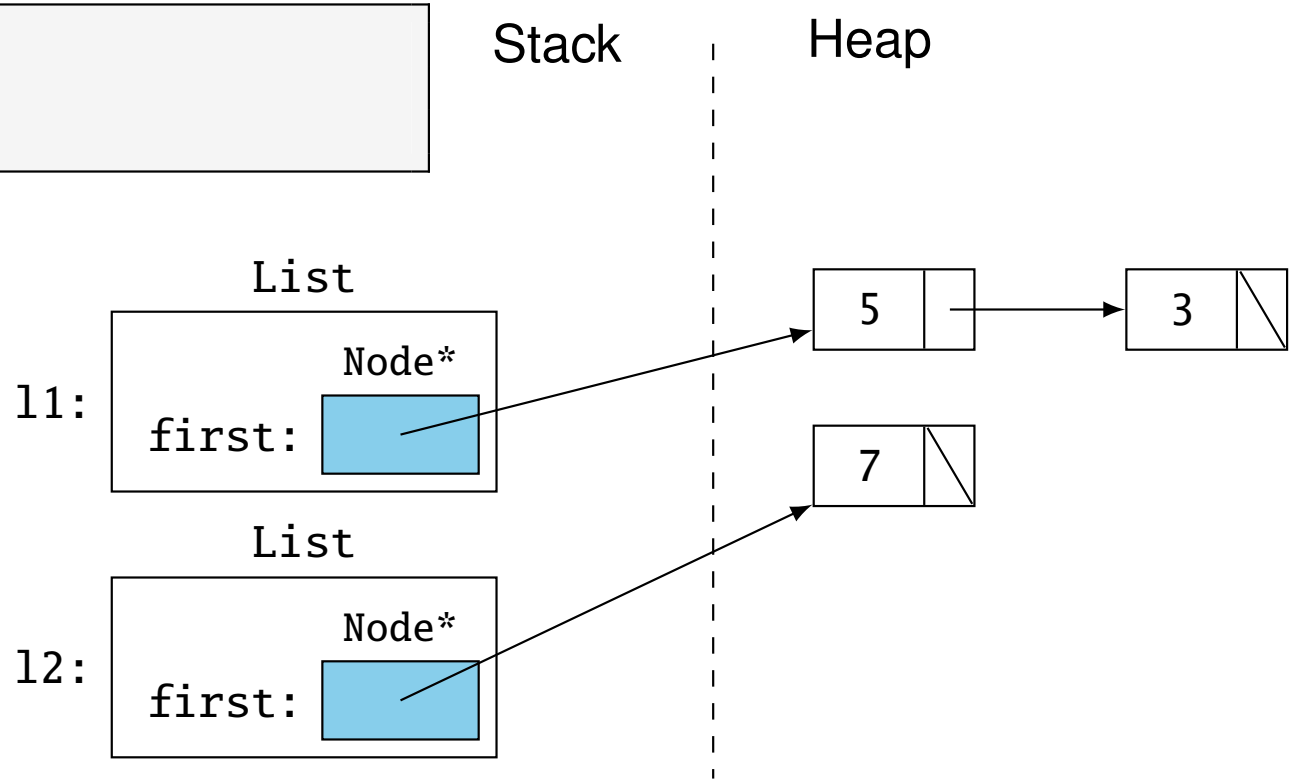
```
Foo& operator=(Foo const& other)
```

- Ersätt ett existerande objekt med innehållet från ett annat objekt av samma typ.
- I fallet av länkade strukturer bör denna göra en djup kopia av det andra objektet.
- Samma funktionallitet som kopieringskonstruktorn men anropas i andra situationer.
- returvärdet är en referens till sig själv

Speciella medlemsfunktioner

Kopieringstilldelningsoperator

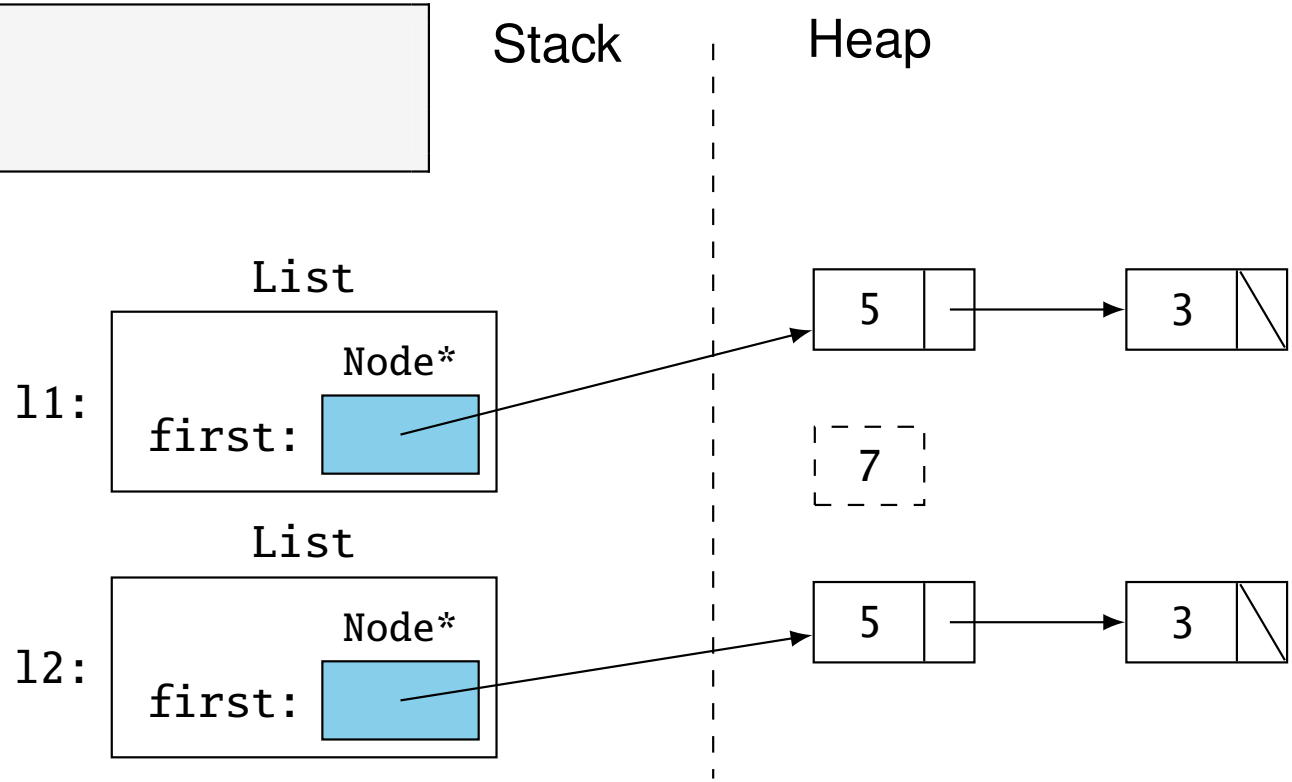
```
List l1 {3, 5};
List l2 {7};
l2 = l1;
```



Speciella medlemsfunktioner

Kopieringstilldelningsoperator

```
List l1 {3, 5};
List l2 {7};
l2 = l1;
```



Speciella medlemsfunktioner

Destruktor

```
~Foo()
```

- Körs när ett objekt ska tas bort.
  Om delete anropas eller objektet går ur scope.
- Bör ansvara för att ta bort allt minne som objektet har allokerat.
- Ska aldrig anropas explicit.

Speciella medlemsfunktioner

Flyttkonstruktor

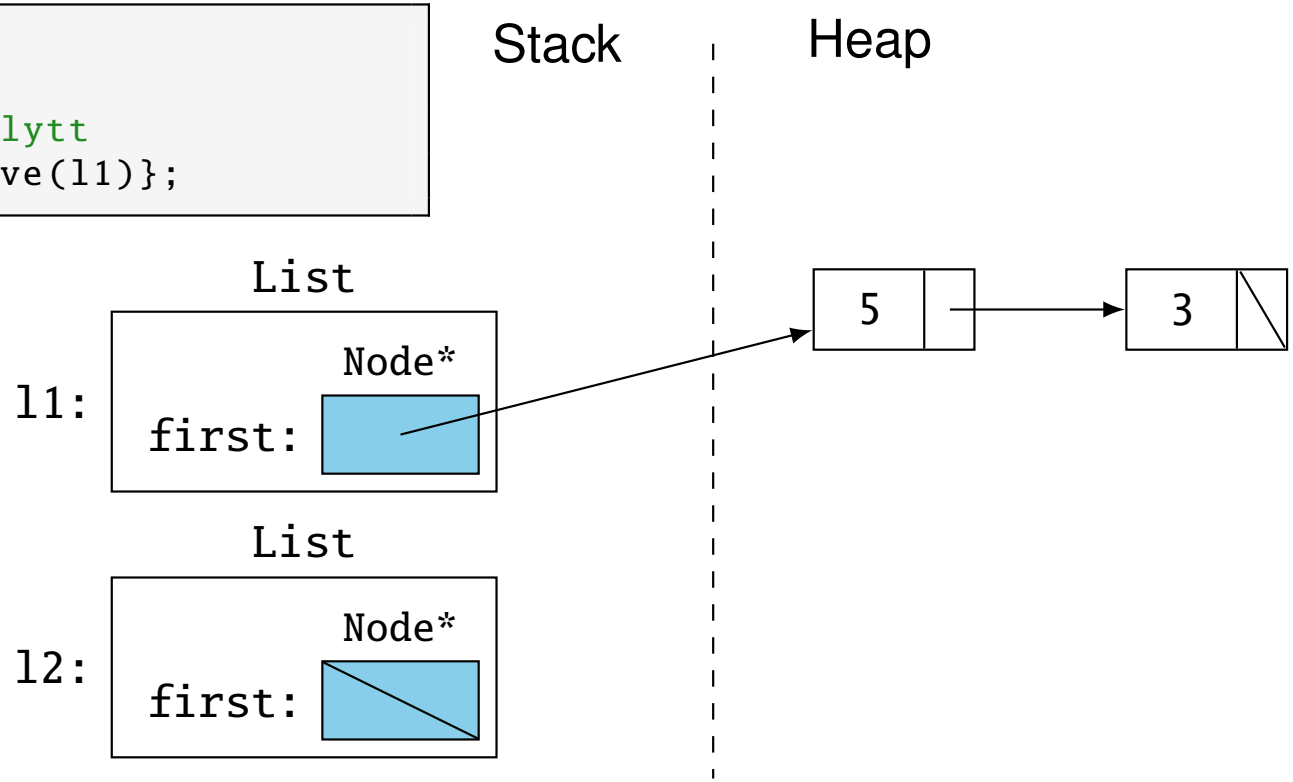
```
Foo(Foo && other)
```

- Skapa ett nytt objekt genom att “sno” innehållet från ett annat objekt av samma typ.
- Används när man vill ha en kopia av ett döende objekt.
- Anropas av kompilatorn för att optimera.
  Bör sällan anropas explicit.

Speciella medlemsfunktioner

Flyttkonstruktor

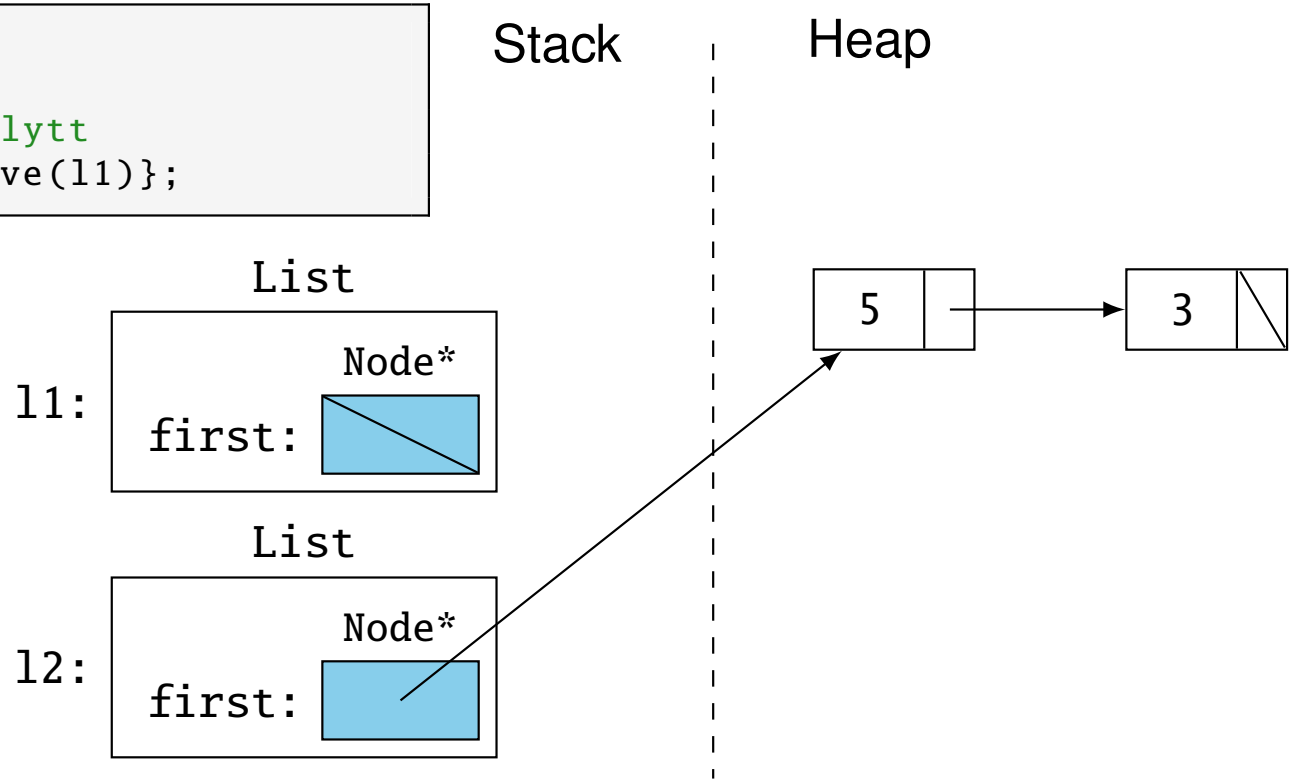
```
List l1 {3, 5};

// tvinga fram flytt
List l2 {std::move(l1)};
```



Speciella medlemsfunktioner

Flyttkonstruktor

```
List l1 {3, 5};

// tvinga fram flytt
List l2 {std::move(l1)};
```



Speciella medlemsfunktioner

Flyttilldelningsoperator

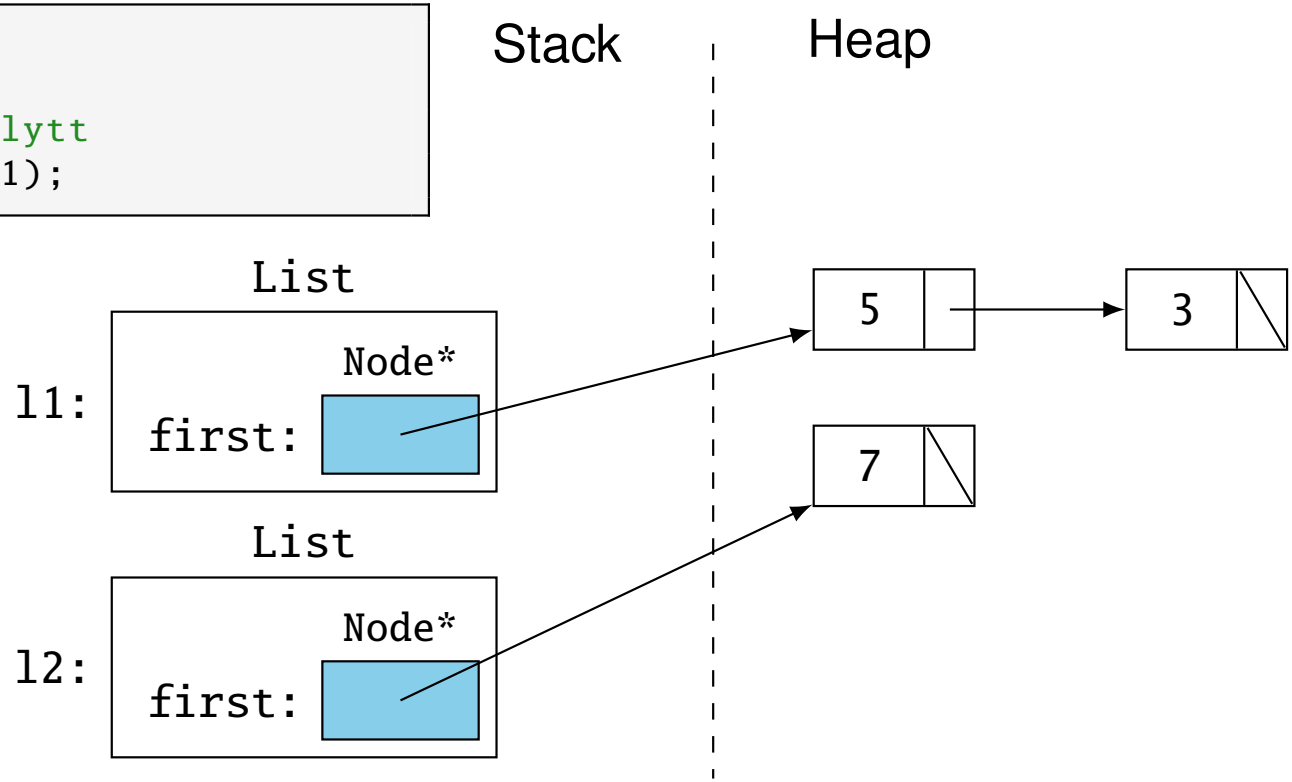
```
Foo& operator=(Foo && other)
```

- Ersätt ett existerande objekt genom att “sno” innehållet från ett annat objekt av samma typ.
- Används när man vill ha en kopia av ett döende objekt.
- Anropas av kompilatorn för att optimera.
  Bör sällan anropas explicit.

Speciella medlemsfunktioner

Flyttilldelningsoperator

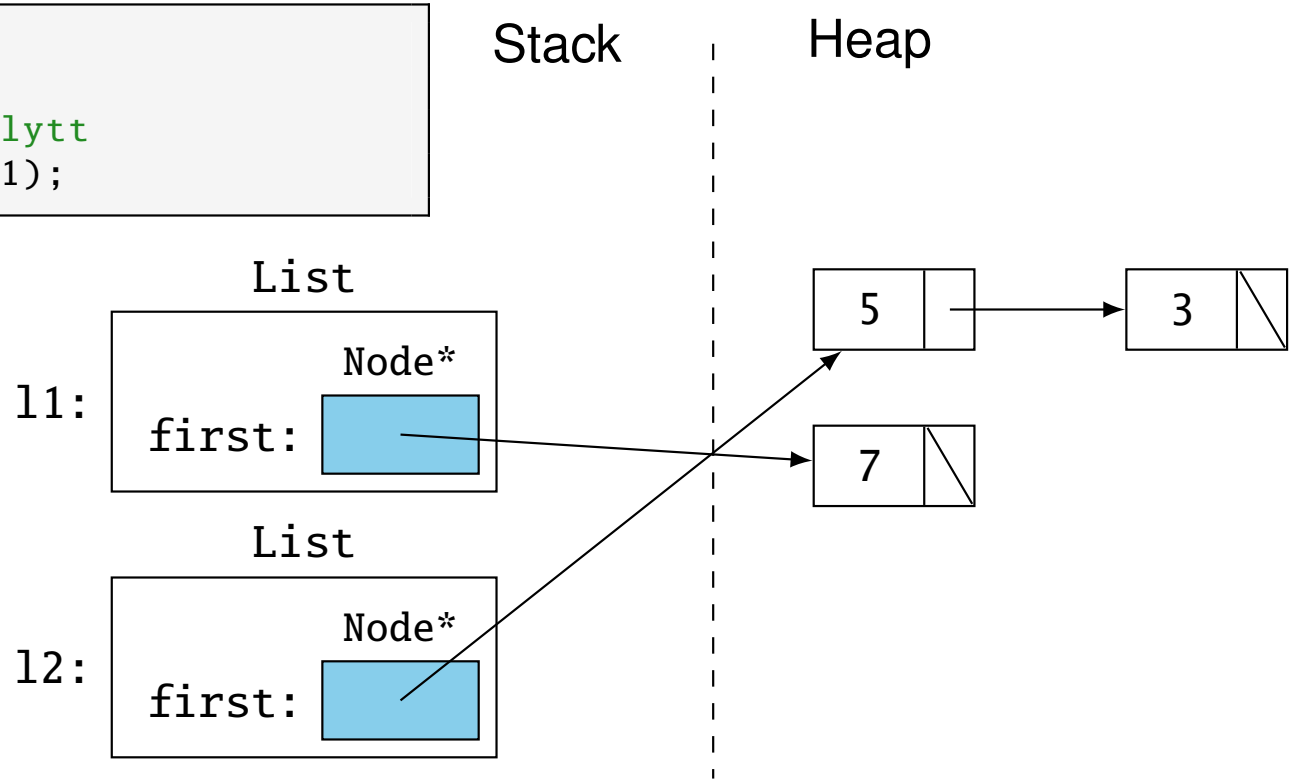
```
List l1 {3, 5};
List l2 {7};
// tvinga fram flytt
l2 = std::move(l1);
```



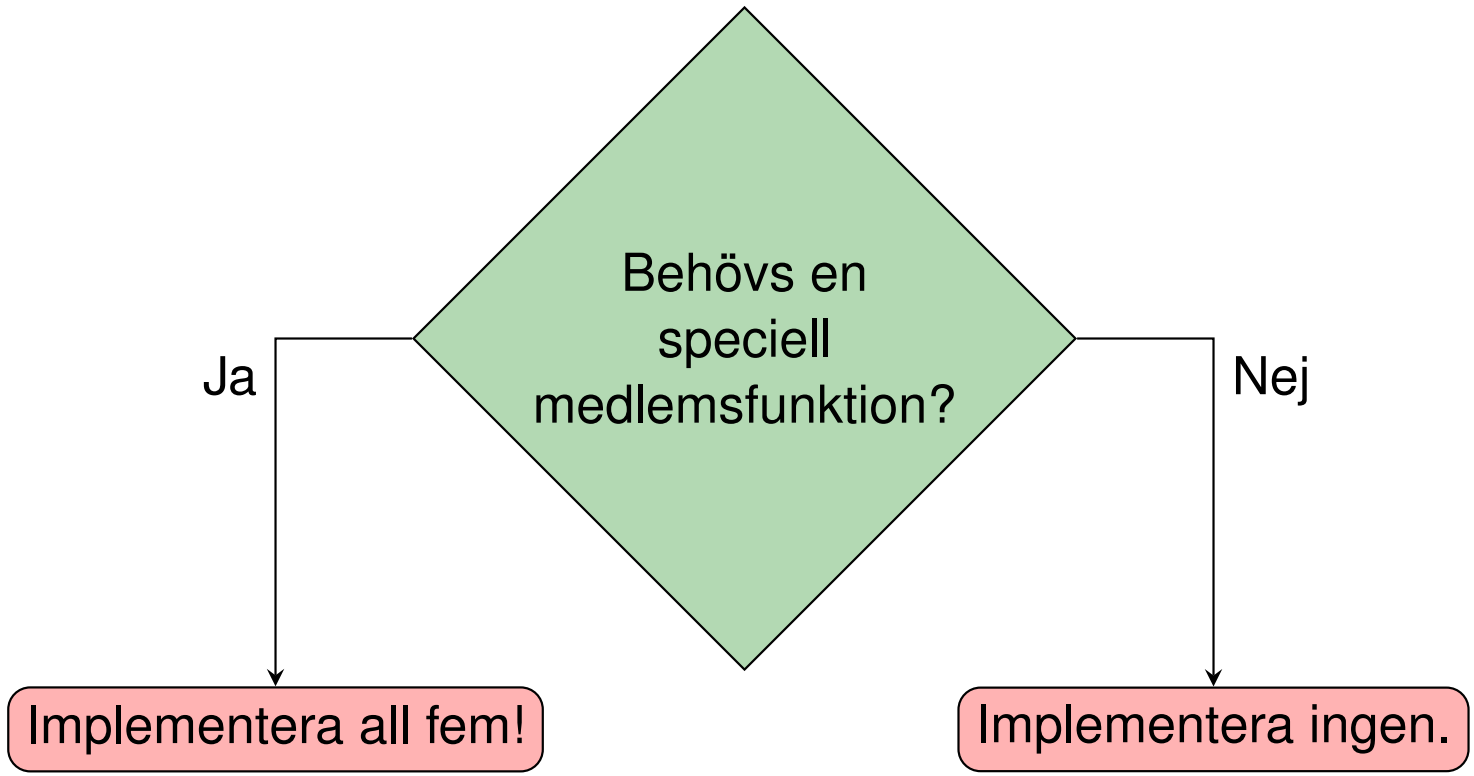
Speciella medlemsfunktioner

Flyttilldelningsoperator

```
List l1 {3, 5};
List l2 {7};
// tvinga fram flytt
l2 = std::move(l1);
```



Speciella medlemsfunktioner - Rule of five



Speciella medlemsfunktioner

- Om vi vill säga åt kompilatorn att använda dess standardimplementation kan vi använda nyckelordet `default`.

```
X(X const& other) = default;
~X() = default;
```

- Om vi vill säga åt kompilatorn att inte generera någon standardimplementation kan vi använda nyckelordet `delete`.

```
X(X const& other) = delete;
X& operator=(X const& other) = delete;
```

1 Repetition - ADT
2 Speciella medlemsfunktioner
3 Livekod
4 Git
5 GDB
6 Make

LINKÖPING UNIVERSITY

1 Repetition - ADT
2 Speciella medlemsfunktioner
3 Livekod
4 Git
5 GDB
6 Make

LINKÖPING UNIVERSITY

Git

- Ett så kallat versionshanteringssystem
  - Sparar vår kod i ögonblicksbilder
  - Vi kan spåra ändringar
  - Vi kan gå tillbaka i tiden
  - Vi kan återskapa arbete om olyckan är framme
- Git != Github/Gitlab
  - Git är i första hand ett sätt att versionshantera
  - Går att koppla till en server så att flera kan samarbeta, men ej nödvändigt
- https://learngitbranching.js.org/

Git

Grundanvändning av git

```
$ git init
$ git status
On branch main

No commits yet

nothing to commit
```

Git

Grundanvändning av git

```
$ echo "En sträng" > fil.txt
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include what will be committed)
        fil.txt
```

Git

Grundanvändning av git

```
$ git add fil.txt
$ git status
On branch main

No commits yet

Changes to be committed:
        new file: fil.txt
```

Git

Grundanvändning av git

```
$ git add fil.txt
$ git status
On branch main

No commits yet

Changes to be committed:
        new file: fil.txt

$ git commit -m "Min första commit!"
$ git status
On branch main
nothing to commit, working tree clean
```

Git

Grundanvändning av git

```
$ echo "en ny sträng" > fil.txt
$ echo "till en annan fil" > annan.cc
$ git diff
fil.txt
@@ -1 +1 @@
-en sträng
+en ny sträng
```

Git

Grundanvändning av git

```
$ echo "en ny sträng" > fil.txt
$ echo "till en annan fil" > annan.cc
$ git diff
fil.txt
@@ -1 +1 @@
-en sträng
+en ny sträng

$ git add .
$ git status
On branch main

No commits yet

Changes to be committed:
        new file: annan.cc
        modified: fil.txt
$ git commit -m "Min andra commit"
```

Git

Grundanvändning av git

```
$ git log
commit 50e6a8 (HEAD -> main)
Author: Eric Ekström
    Min andra commit

commit bd07e4
Author: Eric Ekström
    Min första commit!
```

Git

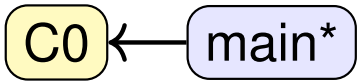
Remote repos

- Vi kan koppla vårt lokala repo till en remote, och på så sätt dela kod med varandra.
  `git remote add origin <address>`
- Git sköter (oftast) att slå ihop kod från flera användare så att vi kan jobba parallellt.
- `git push` för att ladda upp commits till en remote.
- `git pull` för att ladda ner commits från en remote.

Git

Branches

```
git commit
```



Git

Branches

```
git commit
git branch bar
```

Git

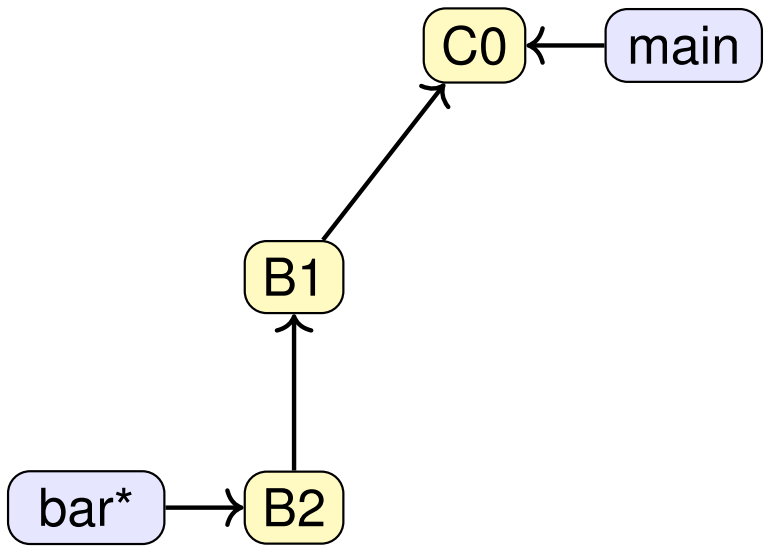
Branches

```
git commit
git branch bar
git checkout bar
```

Git

Branches

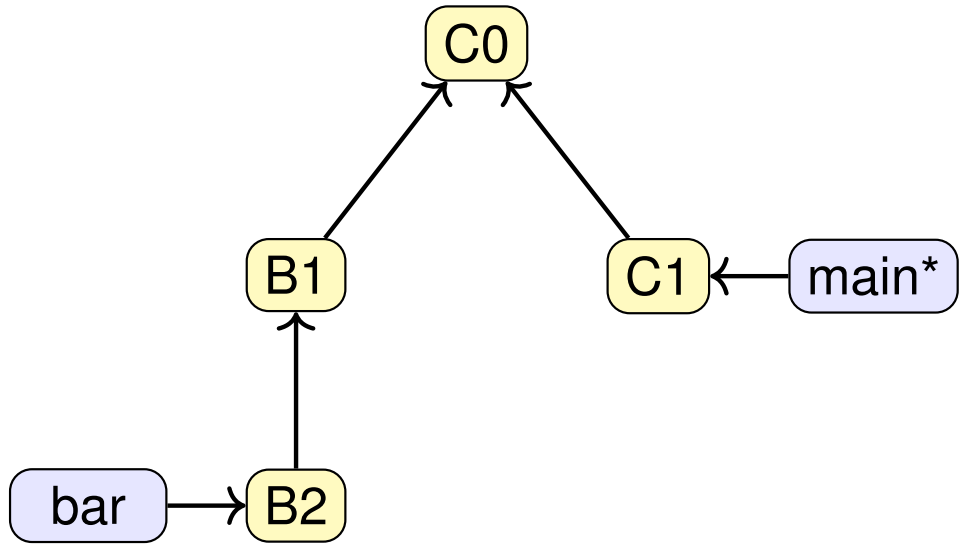
```
git commit
git branch bar
git checkout bar
git commit; git commit
```



Git

Branches

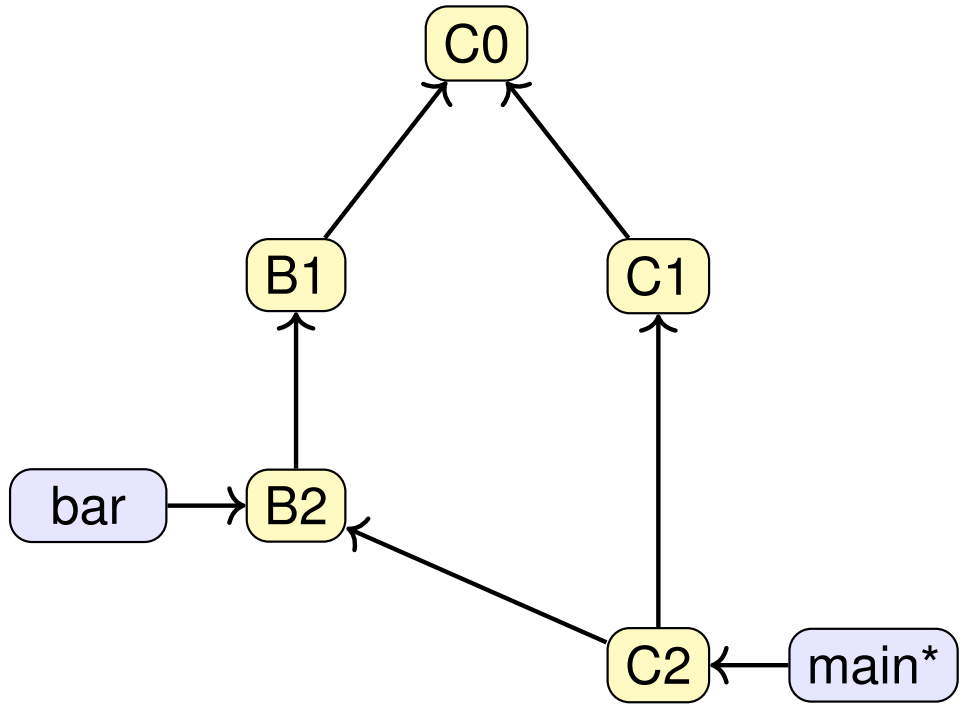
```
git commit
git branch bar
git checkout bar
git commit; git commit
git checkout main; git commit
```



Git

Branches

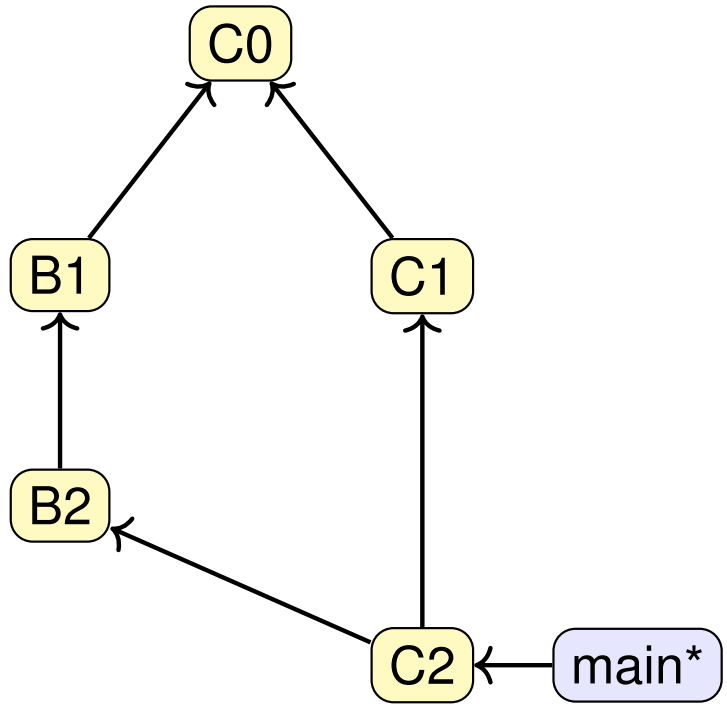
```
git commit
git branch bar
git checkout bar
git commit; git commit
git checkout main; git commit
git merge bar
```



Git

Branches

```
git commit
git branch bar
git checkout bar
git commit; git commit
git checkout main; git commit
git merge bar
git branch -d bar
```



Git

Viktiga commandon

Skapa ett git-repo:

- `init` - skapa ett nytt tomt repo i en befintlig katalog
- `clone <address>` - kopiera ett repo från en remote

Hålla koll på sitt git-repo:

- `status` - visar relevant information
  (Tips: Kör alltid status mellan varje operation!)
- `diff` - visar nuvarande ändringar jämfört med senast commit
- `log` - visar listan med tidigare commits

Git

Viktiga commandon

Göra nya commits:

- `add <file>` - registrerar ändringar till nästa commit.
- `commit -m "message"` - skapa en ny commit

Prata med en remote:

- `push` - synkronisera server med de commit du har lokalt
- `pull` - syknronisera ditt lokala repo med det som finns på servern

Git

Viktiga commandon

Hantera branches:

- `branch <namn>` - skapa en branch
- `branch -d <namn>` - ta bort en branch
- `checkout <namn>` - byt branch
- `merge <namn>` - tillämpa en branch till nuvarande

Git

Tips: Lyssna på vad git säger!

```
$ git push
fatal: No configured push destination.
Either specify the URL from the command-line or configure a
remote repository using

    git remote add <name> <url>

and then push using the remote name

    git push <name>
```

Git

Tips: Git config

Vi kan anpassa det mesta i git.

```
# Sätt användarnamn och mailadress
git config --global user.name eriek23
git config --global user.email eric.ekstrom@liu.se

# Välj vilken editor som ska användas
git config --global core.editor emacs

# Skapa ett alias 'git ci' för 'git commit'
git config --global alias.ci commit
```

Se till att sätta upp namn och mailadress!
Annars kan assistenten inte se vem det är som skrivit kod.

Git

Merge Conflict

```
$ git merge bar
Auto-merging Time.cc
CONFLICT (content): Merge conflict in Time.cc
Automatic merge failed; fix conflicts and then
commit the result.
```

```
int main()
{
<<<<<<<<< HEAD
  cout << "Bar" << endl;
=========
  cout << "Foo" << endl;
>>>>>>>>> bar
}
```

Git

Merge Conflict

Vi fixar konflikten

```
int main()
{
  cout << "Bar" << endl;
}
```

och kör sedan

```
$ git add Time.cc
$ git commit -m "Fixed merge conflict"
```

Git

Övrigt om git

- `.gitignore` - för filer som inte ska versionshanteras
- `git stash` - Vad gör man om man har arbete som inte är redo att bli en commit?
- ssh-nycklar - för att slippa skriva in lösenord vid varje `push` och `pull`.

1 Repetition - ADT
2 Speciella medlemsfunktioner
3 Livekod
4 Git
5 GDB
6 Make

LINKÖPING UNIVERSITY

GDB

GDB (*“Gnu DeBugger”*) kan hjälpa oss hitta minnesfel.

```
$ g++ -g list.cc
$ gdb ./a.out
Reading symbols from ./a.out...
(gdb)
```

GDB

```
(gdb) run
Starting program: ./a.out

Program received signal SIGSEGV, Segmentation fault.
0x000055555555522b in foo () at list.cc:10
10          std::cout << *p2 << std::endl;
```

GDB

Fler knep med GDB:

- `break` - Stoppa programmet på en specifik rad.
- `print` - Skriv ut nuvarande värdet på en variabel.
- `backtrace` - Skriv ut all funktioner som anropats.

1 Repetition - ADT
2 Speciella medlemsfunktioner
3 Livekod
4 Git
5 GDB
6 Make

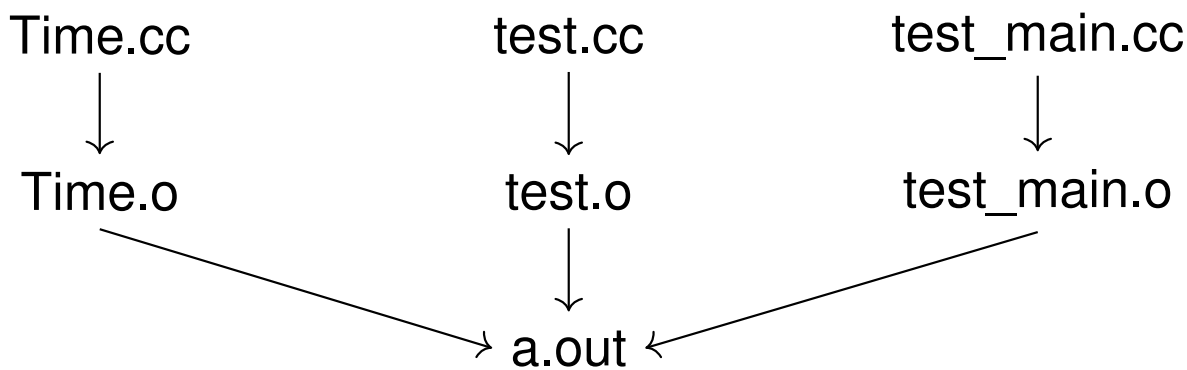
LINKÖPING UNIVERSITY

Kompilera stora projekt

- Mycket att hålla koll på
  - Vilka flaggor ska användas?
  - Vilka filer finns?
  - Hur ska varje fil kompileras?
  - Finns det olika exekverbara filer?
- Lång kompileringstid om all filer ska kompileras om

Kompilera *stora* projekt

```
$ g++ -c Time.cc
$ g++ -c test.cc
$ g++ -c test_main.cc
$ g++ Time.o test.o test_main.o
```



Make

- Ett verktyg där vi specifiera hur kompilering ska ske
- Sparas i en fil som heter `Makefile`
  - Innehåller regler för hur varje fil ska kompileras
  - Körs genom kommandot `make` i terminalen
- Make håller koll på vilka filer som har uppdaterats och behöver kompileras om

Make - Exempel

```
a.out: Time.o test.o test_main.o
    g++ Time.o test.o test_main.o


Time.o: Time.cc Time.h
    g++ -c Time.cc


test.o: test.cc
    g++ -c test.cc


test_main.o: test_main.cc
    g++ -c test_main.cc
```

Make - Exempel

```
a.out: Time.o test.o test_main.o
    g++ Time.o test.o test_main.o


Time.o: Time.cc Time.h
    g++ -c Time.cc


test.o: test.cc
    g++ -c test.cc


test_main.o: test_main.cc
    g++ -c test_main.cc
```

- Mål: Den filen som ska skapas `a.out`

Make - Exempel

```
a.out: Time.o test.o test_main.o
    g++ Time.o test.o test_main.o


Time.o: Time.cc Time.h
    g++ -c Time.cc


test.o: test.cc
    g++ -c test.cc


test_main.o: test_main.cc
    g++ -c test_main.cc
```

- Beroenden: “Om dessa filer uppdaterats behöver vi kompilera om.”
  `Time.cc Time.h`

Make - Exempel

```
a.out: Time.o test.o test_main.o
    g++ Time.o test.o test_main.o


Time.o: Time.cc Time.h
    g++ -c Time.cc


test.o: test.cc
    g++ -c test.cc


test_main.o: test_main.cc
    g++ -c test_main.cc
```

- Kommando: Hur skapas målet?
  `g++ -c test.cc`

Make - Exempel

```
a.out: Time.o test.o test_main.o
    g++ Time.o test.o test_main.o


Time.o: Time.cc Time.h
    g++ -c Time.cc


test.o: test.cc
    g++ -c test.cc


test_main.o: test_main.cc
    g++ -c test_main.cc
```

- Regel: Alla bitar tillsammans är en regel.

```
test_main.o: test_main.cc
    g++ -c test_main.cc
```

Make - Exempel

```
a.out: Time.o test.o test_main.o
    g++ Time.o test.o test_main.o


Time.o: Time.cc Time.h
    g++ -c Time.cc


test.o: test.cc
    g++ -c test.cc


test_main.o: test_main.cc
    g++ -c test_main.cc
```

- Mål: Den filen som ska skapas
  `a.out`
- Beroenden: “Om dessa filer uppdaterats behöver vi kompilera om.”
  `Time.cc Time.h`
- Kommando: Hur skapas målet?
  `g++ -c test.cc`
- Regel: Alla bitar tillsammans är en regel.

```
test_main.o: test_main.cc
    g++ -c test_main.cc
```

Make - Stort exempel

```
FLAGS = -std=c++17 -Wall -Wextra
LIBS  = -lsfml-window -lsfml-graphics -lsfml-system
OBJS  = player.o enemy.o main.o

# $(var) ger värdet av variabeln
# $@ refererar till målet och $^ till beroenden
game: $(OBJS)
    g++ $(FLAGS) -o $@ $^ $(LIBS)

# % är en wildcard, $< refererar till första beroendet
%.o: %.cc %.h
    g++ $(FLAGS) -c $<

# detta är en regel som inte skapar en fil
.PHONY: clean
clean:
    rm *.o game
```

www.ida.liu.se/~TDDE71/

LINKÖPING UNIVERSITY