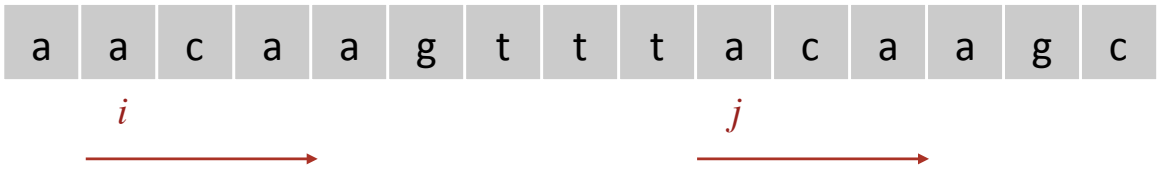
Röðun og leit

Finnum lengsta endurtekna hlutstreng, þ.e. kemur fyrir a.m.k. tvisvar

Einföld lausn, prófum alla mögulega staði sem strengirnir byrja, finnum lengsta sameiginlega forskeyti, $O(N^2)$ keyrslutími



Hagnýtingar: gagnaþjöppun, lífupplýsingafræði

LRS

input string

```
0 1 2 3 4 5 6 7 8 91011121314
a a c a a g t t t a c a a g c
```

myndum viðskeyti

suffixes

```
 0  a a c a a g t t t a c a a g c
 1  a c a a g t t t a c a a g c
 2  c a a g t t t a c a a g c
 3  a a g t t t a c a a g c
 4  a g t t t a c a a g c
 5  g t t t a c a a g c
 6  t t t a c a a g c
 7  t t a c a a g c
 8  t a c a a g c
 9  a c a a g c
10  c a a g c
11  a a g c
12  a g c
13  g c
14  c
```

Röðum viðskeytum, endurteknir strengir liggja saman

sorted suffixes

```
 0  a a c a a g t t t a c a a g c
11  a a g c
 3  a a g t t t a c a a g c
 9  a c a a g c
 1  a c a a g t t t a c a a g c
12  a g c
 4  a g t t t a c a a g c
14  c
10  c a a g c
 2  c a a g t t t a c a a g c
13  g c
 5  g t t t a c a a g c
 8  t a c a a g c
 7  t t a c a a g c
 6  t t t a c a a g c
```

Finnum lengsta sameiginlega forskeyti á milli samliggjandi viðskeyta

longest repeated substring

```
   1           9
a a c a a g t t t a c a a g c
```

LRS

Einföld útfærsla

```
int N = s.length();
String[] suffixes = new String[N];
for (int i = 0; i < N; i++)
   suffixes[i] = s.substring(i, N);
Arrays.sort(suffixes);
```

Skrifum fall lcp(s,t) sem finnur lengsta forskeyti

- lcp(“agacctta”,”agatt”) = “aga”

Berum saman samliggjandi viðskeyti

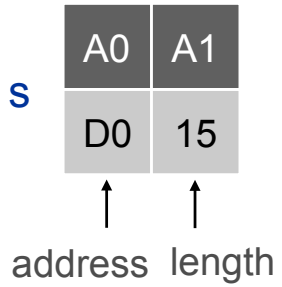
```
String lrs = "";
for (int i = 0; i < N-1; i++) {
   String x = lcp(suffixes[i], suffixes[i+1]);
   if (x.length() > lrs.length()) lrs = x;
}
```

Strengir í minni

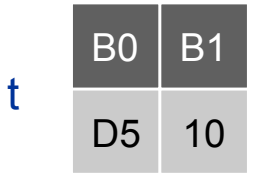
String geymir tilvísun á fylki af stöfum og lengd

- String er óbreytanlegur, strengir geta deilt sama fylki
- substring() reiknar út nýja lengd, skoðar ekki streng

D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE
a	a	c	a	a	g	t	t	t	a	c	a	a	g	c




```
s = "aacaagtttacaagc";

t = s.substring(5, 15);
```



Afleiðingar

- substring() tekur O(1) tíma
- suffix[] fylkið tekur aðeins O(N) minni
- Röðunin í LRS tekur lengstan tíma!

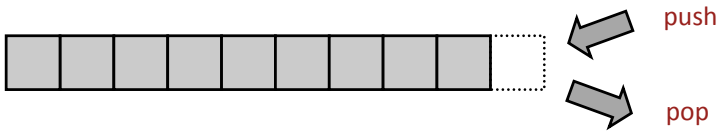
LRS

Inntak	Stafir	Einföld lausn	Suffix Sort	Lengd
LRS.java	2,162	0.6 sec	0.14 sec	73
Amendments	18,369	37 sec	0.25 sec	216
Aesop's Fables	191,945	3958 sec	1.0 sec	58
Moby Dick	1.2 million	43 hours †	7.6 sec	79
Bible	4.0 million	20 days †	34 sec	11
Chromosome 11	7.1 million	2 months †	61 sec	12,567
Pi	10 million	4 months †	84 sec	14

† áætlaður tími

API fyrir stafla

public class *StackOfStrings	
*StackOfStrings()	*create an empty stack*
boolean isEmpty()	*is the stack empty?*
void push(String item)	*push a string onto the stack*
String pop()	*pop the stack*



Staflar

```
public class Reverse {
   public static void main(String[] args) {
      StackOfStrings stack = new StackOfStrings();
      while (!StdIn.isEmpty()) {
         String s = StdIn.readString();
         stack.push(s);
      }
      while (!stack.isEmpty()) {
         String s = stack.pop();
         StdOut.println(s);
      }
   }
}
```

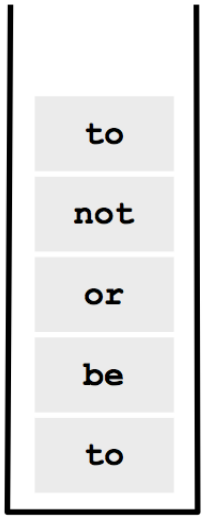
```
% more tiny.txt
it was the best of times

% java Reverse < tiny.txt
times of best the was it
```

staflinn þegar að StdIn var tómt

Staflar

```
public static void main(String[] args) {
   StackOfStrings stack = new StackOfStrings();
   while (!StdIn.isEmpty()) {
      String s = StdIn.readString();
      if (s.equals("-"))
         StdOut.println(stack.pop());
      else
         stack.push(s);
   }
}
```

```
% more test.txt
to be or not to - be - - that - - - is

% java StackOfStrings < test.txt
to be not that or be
```

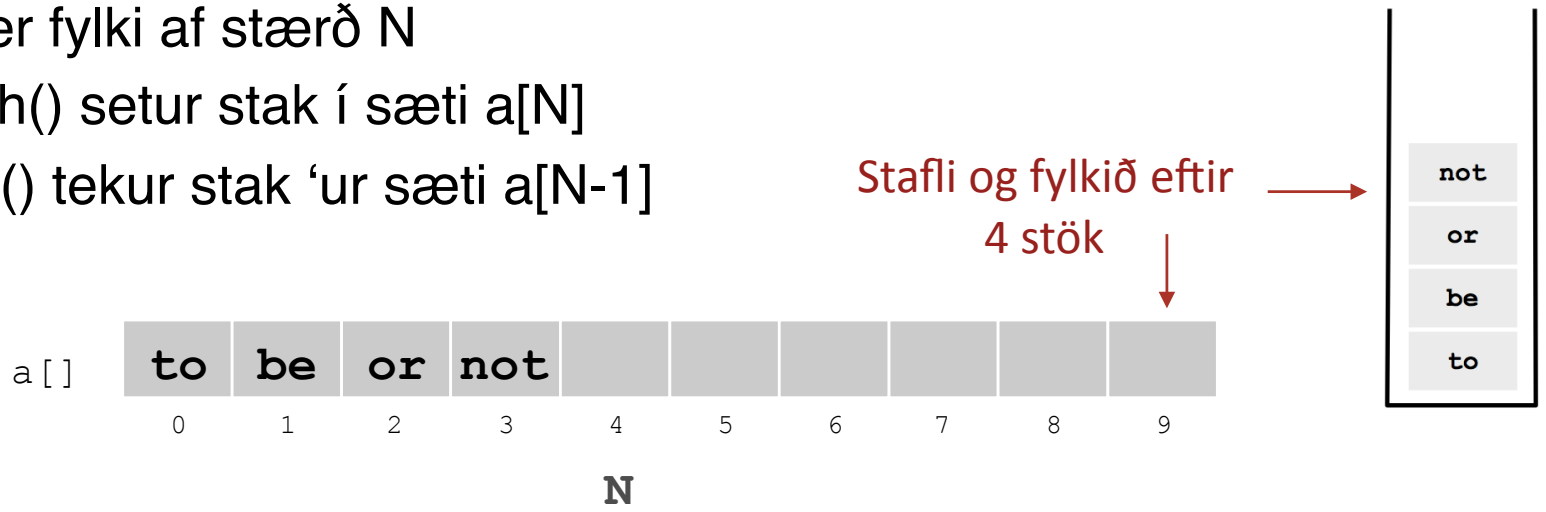


Staflinn eftir fyrstu pop() aðgerð

Staflar - útfærsla

Notum fylki til að geyma stökin

- a[] er fylki af stærð N
- push() setur stak í sæti a[N]
- pop() tekur stak ‘ur sæti a[N-1]




```
public class ArrayStackOfStrings {
   private String[] a;
   private int N = 0;

   public ArrayStackOfStrings(int max) { a = new String[max]; }
   public boolean isEmpty()            { return (N == 0);  }
   public void push(String item)       { a[N++] = item;    }
   public String pop()                 { return a[--N];    }
}
```

Ljót lausn: notum fylki sem stækkar

Staflar - útfærsla

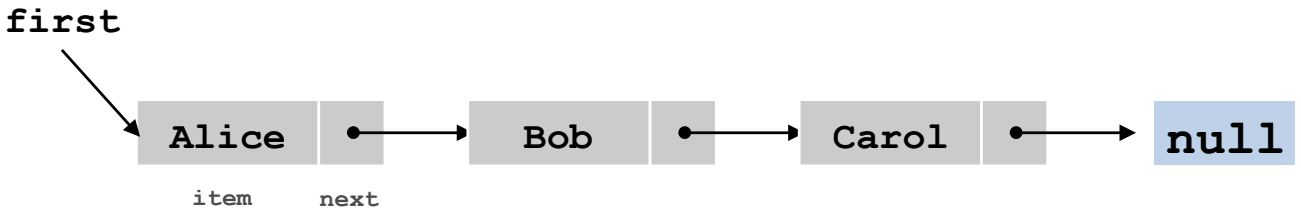
	StdIn	StdOut	N	a[]				
				0	1	2	3	4
			0					
push	to		1	to				
	be		2	to	be			
	or		3	to	be	or		
	not		4	to	be	or	not	
	to		5	to	be	or	not	to
pop	-	to	4	to	be	or	not	to
	be		5	to	be	or	not	be
	-	be	4	to	be	or	not	be
	-	not	3	to	be	or	not	be
	that		4	to	be	or	that	be
	-	that	3	to	be	or	that	be
	-	or	2	to	be	or	that	be
	-	be	1	to	be	or	that	be
	is		2	to	is	or	not	to

Tengdir listar

Tengdur listi er endurkvæm gagnagrind

- Listi af hnútum
- Hver hnútur geymir eitt stak
- og tilvísun á næsta hnút í listanum

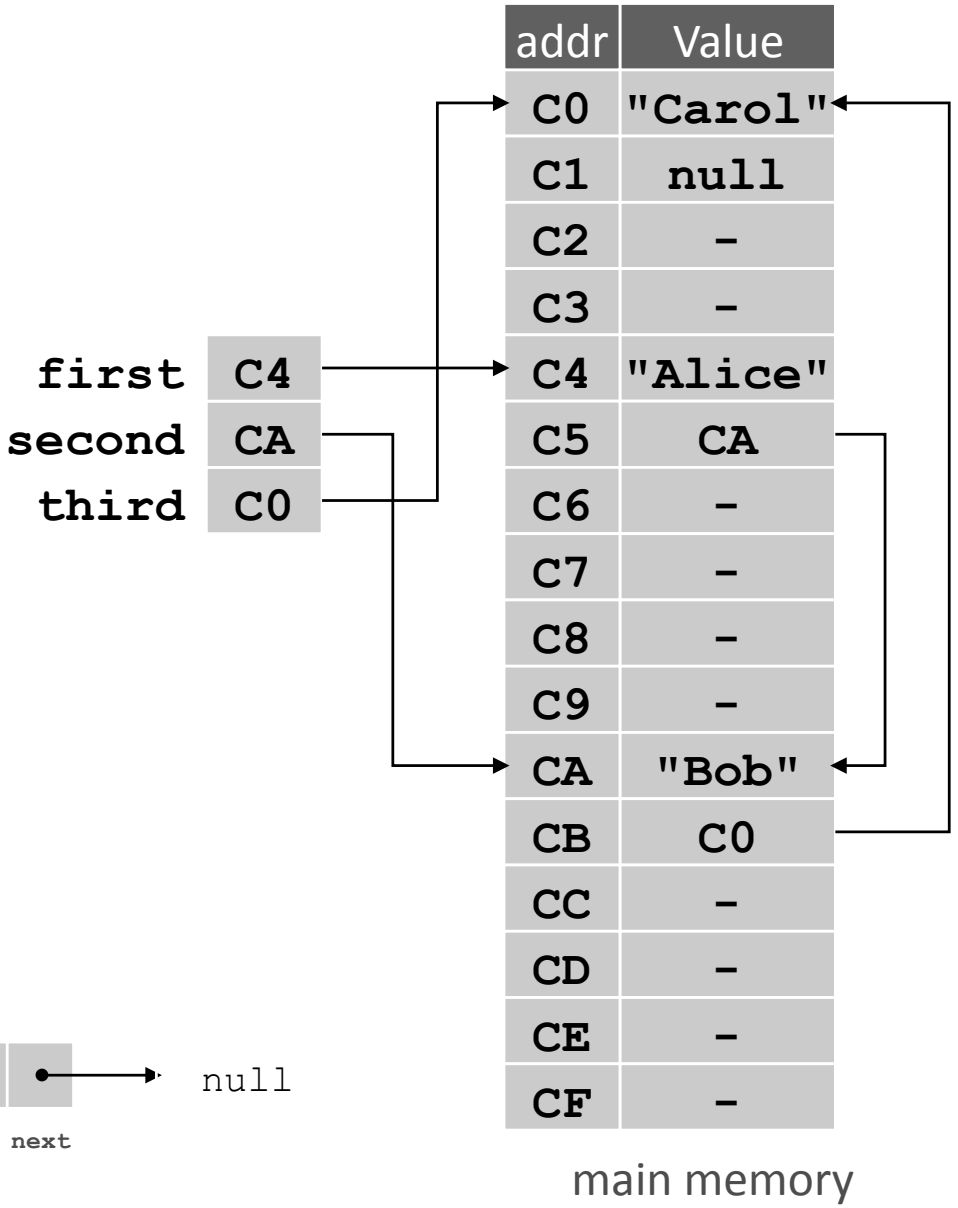
```
public class Node {
  private String item;
  private Node next;
}
```

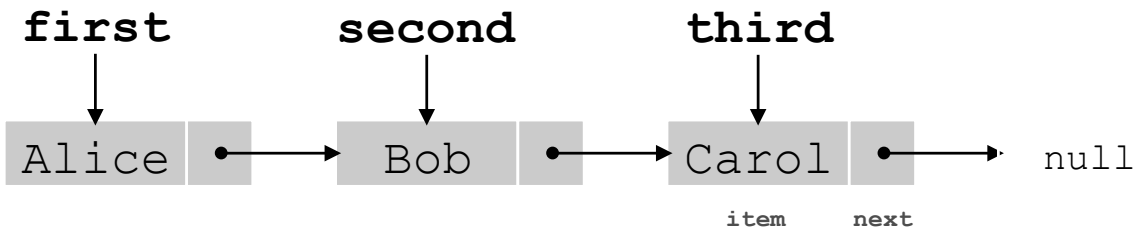


tilvísunin null vísar ekki
á neinn hlut, endar listann

Tengdir listar

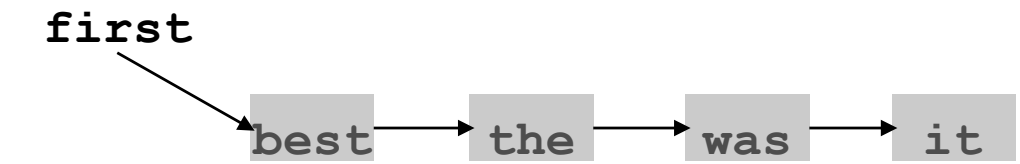
```
Node third  = new Node();
third.item  = "Carol";
third.next  = null;

Node second = new Node();
second.item = "Bob";
second.next = third;

Node first  = new Node();
first.item  = "Alice";
first.next  = second;
```

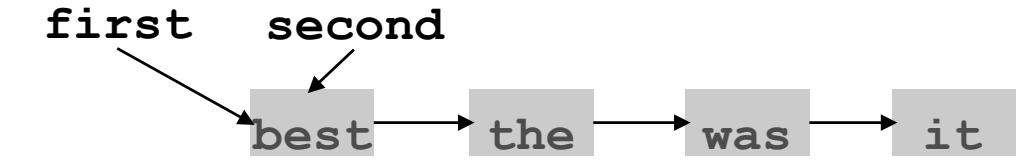




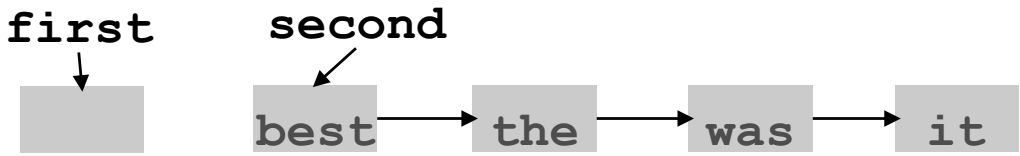
Staflar útfærsla

Stafli með tengdum lista, útfærsla á push()

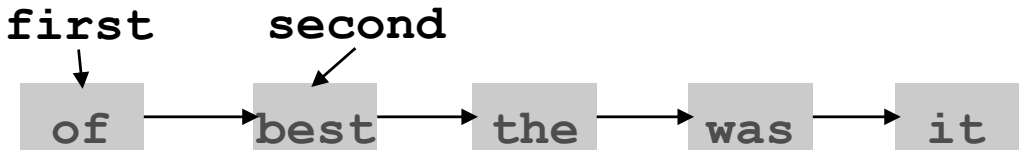





```
Node second = first;
```



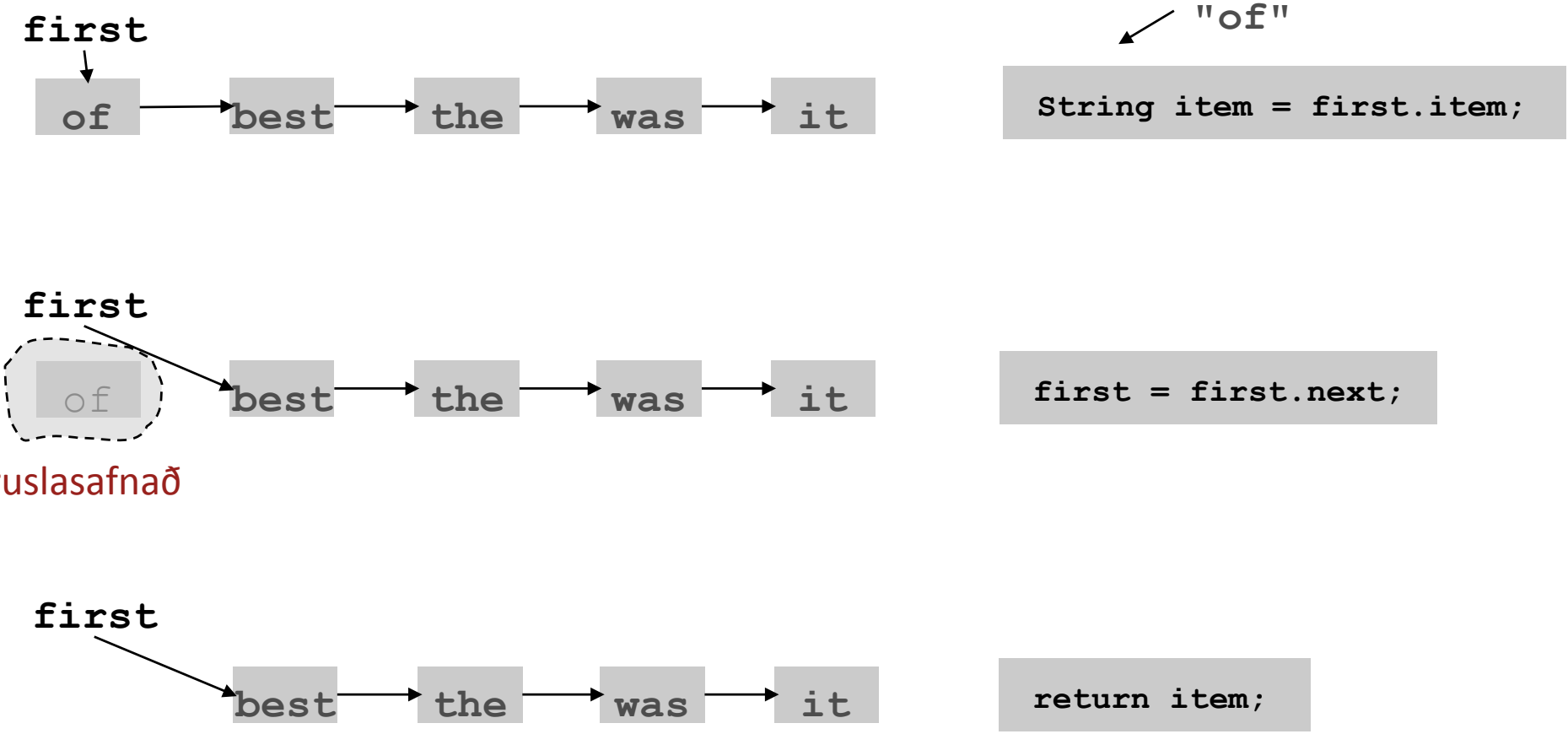

```
first = new Node();
```




```
first.item = "of";
first.next = second;
```

Staflar útfærsla

Stafli með tengdum lista, útfærsla á pop()

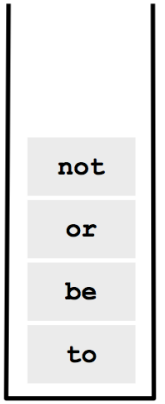


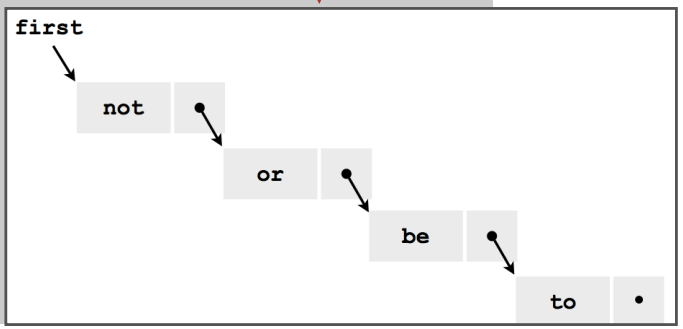
Staflar útfærsla

```
public class LinkedStackOfStrings {
   private Node first = null;

   private class Node {
      private String item;
      private Node next;
   }

   public boolean isEmpty() { return first == null; }

   public void push(String item) {
      Node second = first;
      first = new Node();
      first.item = item;
      first.next = second;
   }

   public String pop() {
      String item = first.item;
      first = first.next;
      return item;
   }
}
```

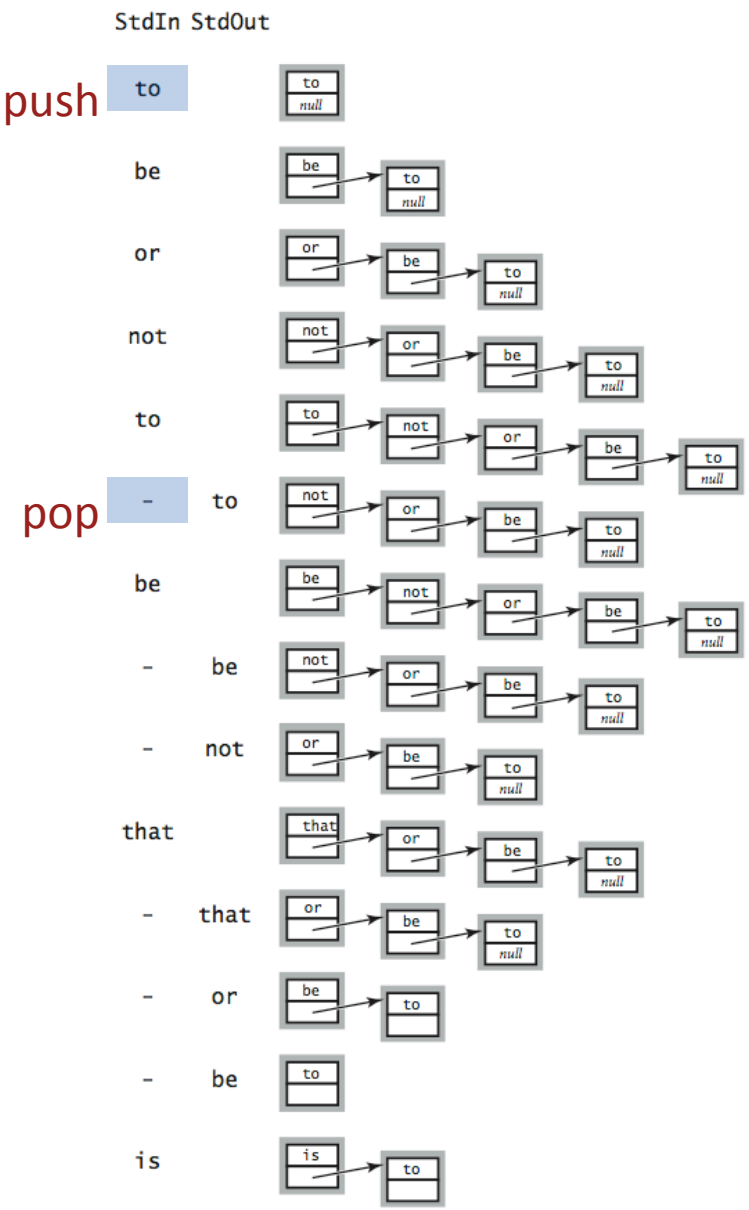
Innri private klasi, ósýnilegur öðrum klösum

stafli og tengdur listi eftir 4 stök





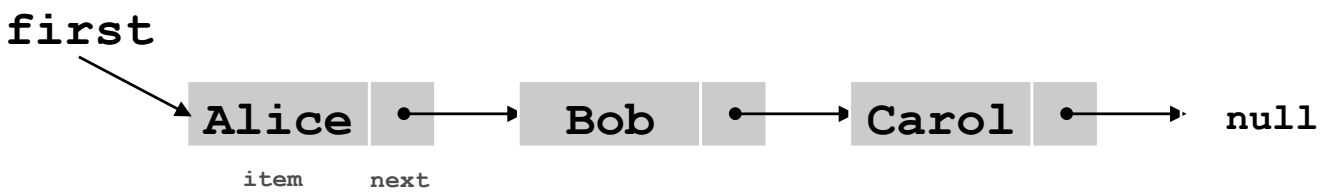
Staflar


StdIn StdOut
push
to
be
or
not
to
pop
-
to
be
-
be
-
not
that
-
that
-
or
-
be
is
to
null

Tengdir listar

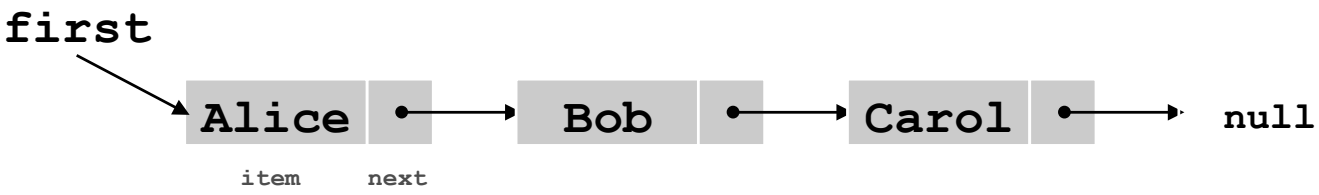
Hvað gerir þessi kóði?

```
for (Node x = first; x != null; x = x.next) {
   StdOut.println(x.item);
}
```



Hvað gerist hérna?

```
Node last = new Node();
last.item = StdIn.readString();
last.next = null;
Node first = last;
while (!StdIn.isEmpty()) {
   last.next = new Node();
   last = last.next;
   last.item = StdIn.readString();
   last.next = null;
}
```



Stafli með sniðmáti

```
public class Stack<Item> {
   private Node first = null;

   private class Node {
      private Item item;
      private Node next;
   }

   public boolean isEmpty() { return first == null; }

   public void push(Item item) {
      Node second = first;
      first = new Node();
      first.item = item;
      first.next = second;
   }

   public Item pop() {
      Item item = first.item;
      first = first.next;
      return item;
   }
}
```

tagið í sniðmáti, valið seinna

Autoboxing

Sniðmát leyfa bara klasa, ekki frumstæð gagnatög

- Hvert gagnatag hefur hliðstæðan klasa
- T.d. Integer fyrir int, Double fyrir double …

Autoboxing: þegar hlutir eru búnir til sjálfvirkt úr gildum

Autounboxing: þegar hlutum er breytt í gildi

```
Stack<Integer> stack = new Stack<Integer>();
stack.push(17);          // autobox     (int -> Integer)
int a = stack.pop();     // auto-unbox  (Integer -> int)
```

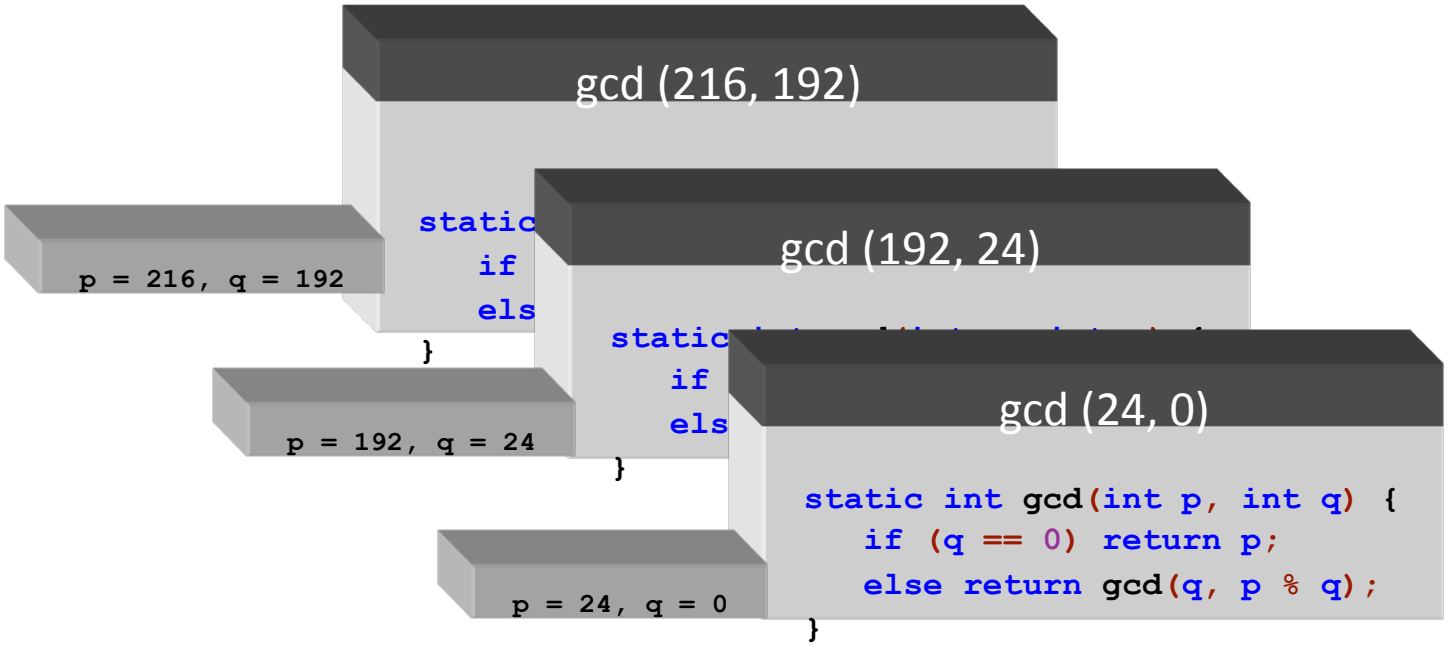
Meiri staflar

Staflar eru notaðir við

- Þýðingu forritstexta í þýðanda
- Java sýndarvélinni
- “Undo” hegðun í ritvinnslu o.fl.
- PostScript forritunarmálinu [sic!] fyrir prentara
- Fyrir fallaköll í þýðanda

Útfærsla á fallaköllum í þýðanda

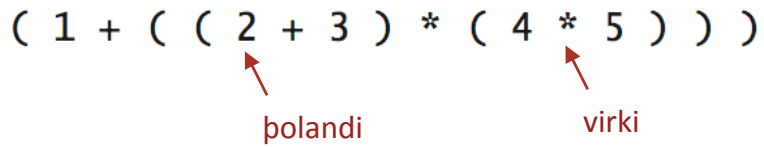
- Fallakall: framkvæmum push á umhverfi (viðfangsbreytur) og skila-adressu
- Return: pop til að skila gildinu



- Endurkvæmt fall: fall sem kallar á sjálft sig
- Getum alltaf hermt eftir endurkvæmni með stafla

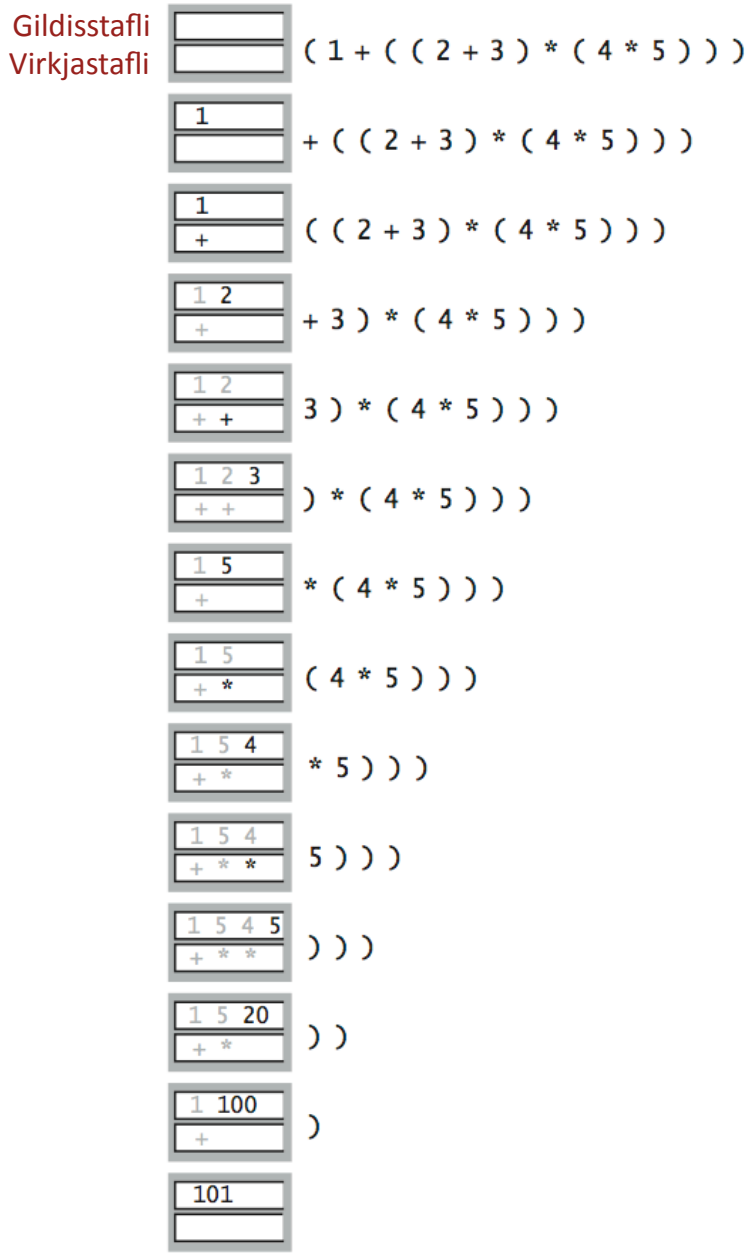
Reiknivél

Markmið: reikna segðir



Tveir staflar: Gildi og Virkjar

- Gildi: push á gildisstaflann
- Virki: push á virkjastaflann
- vinstri svigi: ekkert
- hægri svigi: pop á virkja og tvö pop og á gildi framkvæmum reikning og push niðurstöðu á gildisstaflann



Java útfærsla

```
public class Evaluate {
   public static void main(String[] args) {
      Stack<String> ops  = new Stack<String>();
      Stack<Double> vals = new Stack<Double>();
      while (!StdIn.isEmpty()) {
         String s = StdIn.readString();
         if      (s.equals("("))                ;
         else if (s.equals("+"))    ops.push(s);
         else if (s.equals("*"))    ops.push(s);
         else if (s.equals(")")) {
            String op = ops.pop();
            if      (op.equals("+")) vals.push(vals.pop() + vals.pop());
            else if (op.equals("*")) vals.push(vals.pop() * vals.pop());
         }
         else vals.push(Double.parseDouble(s));
      }
      StdOut.println(vals.pop());
   }
}
```

```
% java Evaluate
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
101.0
```

Af hverju virkar þetta? Þegar reiknirítið sér virkja með tvö gildi innan sviga þá skilur það útkomuna eftir á staflanum

```
( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )
```

Rétt eins og það hefði alltaf verið

```
( 1 + ( 5 * ( 4 * 5 ) ) )
```

Endurtökum

```
( 1 + ( 5 * 20 ) )
( 1 + 100 )
101
```

Betri útfærsla: fleiri aðgerðir, forgangsröð virkja, tengireglur, ekkert óþarfa bil

```
1 + (2 - 3 - 4) * 5 * sqrt(6*6 + 7*7)
```

Stafla forritunarmál

Athugasemd 1: þetta reiknirit skilar sömu niðurstöðu ef virkinn kemur á eftir 2 gildum

```
( 1 ( ( 2 3 + ) ( 4 5 * ) * ) + )
```

Athugasemd 2: þá verða svigarnir óþarfir!

```
1 2 3 + 4 5 * * +
```

Þessi framsetning er kölluð Postfix eða “reverse Polish notation”.

Notuð í PostScript, Forth, reiknivélum, JVM …

Tvær gagnagrindur sem geyma hluti í röð

- Staflar
  - Bætum stökum “efst” í staflann
  - Tökum stök ofan af staflanum
  - “Last in, first out” – LIFO

- Biðraðir
  - Bætum stökum “aftast” í biðröð
  - Tökum stök “fremst” úr biðröðinni
  - “First in, first out” – FIFO