Datastructuren
Bachelor Informatica
najaar 2012

Dr. D.P. Huijsmans

Liacs, Universiteit Leiden

Opzet college Datastructuren

- Hoorcollege: woe 13:45-15:30 uur 412 (12/9) en 174 (>=19/9)
  - Docent:
    - Dr. D.P. (Nies) Huijsmans kamer 152 huijsman@liacs.nl tel:7052
- Werkcollege: woe 13:45-15:30 402 intro (5/9), niet 12/9,
  woe 11:15-13 practicumzaal 302-304 (>=19/9)
- Sommen en 5 progr. opdrachten
  - Assistenten:
    - Jan van Rijn, kamer 106 , janvanrijn@gmail.com
    - Duncan Rozemond, d.rozemond@gmail.com
    - Jan-Paul van Osta, jpvo91@hotmail.com
    - Website: http://www.liacs.nl/~jvrijn/ds2012
- Beoordeling:
  - Schriftelijk tentamen vr 4 jan 10-13 uur voldoende als cijfer >=5.5
  - Alle 5 Opdrachten pract. cijfer voldoende als >=5.5 (duo's)
  - Eindcijfer: cijfers tentamen en practicum wegen even zwaar, moeten elk apart voldoende zijn.

Boek en Sheets

- Aanbevolen boek bij college:
- 3e of 4e druk
- Data Structures and Algorithms in C++
- Adam Drozdek
- Uitgever: Thomson
- College gebruikt ook: Data Structures with Abstract Data Types and Pascal van Stubbs en Webre (helaas geen recente edities);
- Data Structures and Other Objects using C++ van Main en Savitch 4e druk
- Sheets: komen op website en deel (ADT specificaties en implementaties) wordt uitgedeeld.

GEGEVENS VERWERKEN EN DATASTRUCTUREN

- Gegevensverwerking:
  - Invoerdata -> bewerking -> uitvoerdata
  - Hoe structureren we data met het oog op bewerkingen zodanig dat:
  - er efficient met de geheugenruimte wordt omgegaan (minimaal bij voorkeur)
  - er zo weinig mogelijke tijd verstrijkt (complexiteit)
  - terwijl bewerkingen slim zijn georganiseerd (algoritme)
  - zo weinig mogelijk kans op fouten
  - zo overzichtelijk mogelijk gewerkt wordt
  - zo herbruikbaar mogelijk gewerkt wordt
    - modulair, hiërarchisch

Wat kan er in essentie in een computer worden opgeslagen?

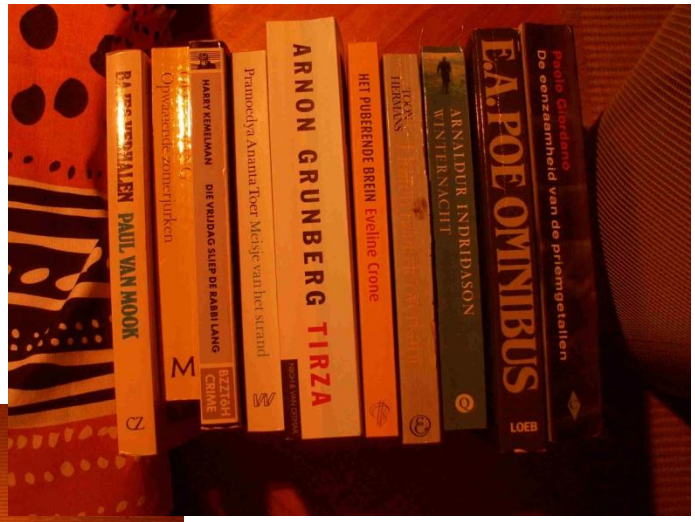
- In essentie is een computer een geordende rij **byte**s, elk apart adresseerbaar (adres), elk met een bepaald bitpatroon (waarde)
- Daarbovenop bouwt informatica laag voor laag meer bruikbare elementen op:
  - Een stel opeenvolgende bytes vormen een **elementair datatype** b.v. 1 byte/karakter, 8 bytes/floating point
  - Het elementair datatype kan een waarde (**value**) bevatten die bij een machinecode instructie als invoer gebruikt wordt
  - Het elementair datatype kan een (door)verwijzing (**pointer**) bevatten naar een beginadres van een of ander datatype

Bakjes met waardes en verwijzingen wat kunnen we daarmee?

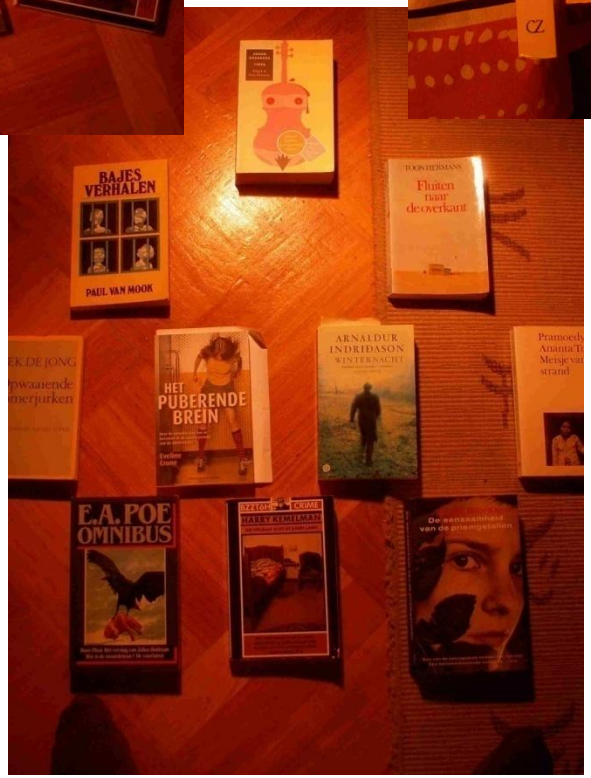
- Values en pointers plus de machinecode bewerkingen die we daarmee kunnen doen vormen de basis van ons denken over te automatiseren informatieverwerkingssystemen
- De bewerkingen en de volgorde waarin die plaatsvinden worden vooral bekeken vanuit **algoritmiek** en **complexiteit**
- De in- en uitvoerwaardes en pointers en de wijze waarop die gestructureerd zijn worden vooral bekeken vanuit **datastructuren**
- In feite zijn ze nauwelijks los van elkaar te zien

RELATIES

verzameling



Lineaire Lijst



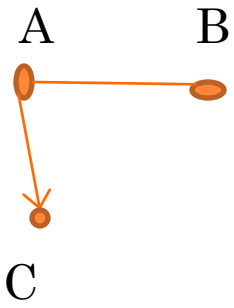
Binaire Boom

Voorbeelden van structuren (samenhang: relatie)

- Verzameling (set): relaties als identiek/verschillend; groter/kleiner; eerder/later; dichterbij/verderaf; links/rechts van; boven/onder
- Lijst (lineair): eerste/laatste; ordening (voor/na)
- Boom: knopen en bogen (nodes and edges) n knopen verbonden met [0..n-1] edges
- Graaf: n knopen en [0..nxn] bogen

Meest algemeen denkmodel voor bakjes met waardes en pointers

- Stel er zijn N bakjes die elk een bepaalde relatie tot de andere bakjes kunnen hebben; een structuur die alle mogelijke relaties tussen N items onderling altijd kan weergeven is de graaf die met de N items als knopen en NxN verbindingen als relaties de samenhang weergeeft in b.v. een 2D NxN verbindingsmatrix



	A	B	C
A	0	1	1
B	1	0	0
C	0	0	0

Grafen en bomen: N^2 of logN

- Grafen zijn de meest algemene manier om over algoritmische datastructuren na te denken, maar brengen een N^2 complexiteit met zich mee
- Omdat een voorziening voor elk van de mogelijke verbanden tussen N items meestal overdreven veel is, kan in bepaalde gevallen bij schaars gevulde verbindingsmatrices volstaan worden met een boomstructuur die meer een log N complexiteit met zich mee brengt

Nadenken over de rol van data bij geautomatiseerde gegevensverwerking

- Probeer zo veel mogelijk te abstraheren van de computer implementatie
  - Hoe zou je het b.v. uitvoeren met pen en papier
  - Hoe zou je er b.v. naar een medestudent toe een opdracht voor formuleren
- Afschermen van details op lager niveau
  - Alleen die informatie zichtbaar maken die op hoger niveau nodig is voor gebruik (invoer eisen,uitvoer verwachtingen)
- Hierarchische opdeling van bewerkingsniveaus
  - Modules
  - functies

Abstract Data Type (ADT)

- Capsule van gestructureerde data samen met de functies die het voor een bepaald bewerkingsdoel geschikt maken

  Wat is het doel van een logisch onderdeel; wat is er nodig om het in werking te zetten en wat kan men als uitvoer verwachten

  - Object Georienteerd programmeren (OOP)
    - Classe in C++ (dit college)

Sorteer voorbeeld

- Functie die, gegeven een ongesorteerde rij invoerwaarden, als uitvoer dezelfde rij maar dan gesorteerd aflevert
- Nodig voor buitenstaander:
  - Naam van deze sorteer functie
  - Een te sorteren rij invoer gegevens
  - Welk elementair datatype
  - Als uitvoer dezelfde rij maar nu gesorteerd
  - (mogelijk ook: welke sorteervolgorde)

Manuele implementatie

- Persoon1 die kan sorteren
- Persoon2 die stapel briefjes met elk een naam heeft
- Persoon2 die persoon1 opdraagt om de stapel naambriefjes gesorteerd terug te geven

  - Persoon2 hoeft niets te weten van de manier waarop persoon1 het sorteren uitvoert; hij kan achteraf wel kontroleren of de afgeleverde stapel briefjes goed gesorteerd is.
  - Persoon1 kan de briefjes sorteren m.b.v. enkele stapels of op rij gelegde briefjes; deze details zijn alleen intern (voor hem) relevant

Computer implementatie sorteerfunctie

- Naar buiten toe zelfde specificatie als bij manuele implementatie:
  - A=Sort(A) of A=Sort(A,CS) CS=collating sequence

Intern moeten alle details van de gebruikte datastructuur , het afchecken van randvoorwaarden en het algoritme om m.b.v. machine instructies de sorteerstappen uit te voeren uitgeschreven zijn; hiervoor wordt weer zoveel mogelijk met onderling afgeschermde (deel)datastructuren en functies gewerkt.

Complexiteit O(N), O(N^2), O(logN), O(1) etc.

- Een van de hoofddoelen van een informatica toepassing is het in de hand houden van de complexiteit van een gegevensverwerking
- Complexiteit: afhankelijkheid aantal gebruikte bewerkingen als functie van het aantal invoer gegevens (N)
- Zo is voor ongesorteerde items de complexiteit vaak minstens O(N) of O(N^2)
- Voor gesorteerde items kan de complexiteit vaak beperkt worden tot O(logN) of O(NlogN)
- De volgende tabel laat duidelijk zien hoe complexiteit doorwerkt op de benodigde rekentijd (die we minimaal willen houden)

N in min	N msecs	N^2	N^2 in dagen	logN msec
	1	1		0
	2	4		1
	4	16		2
	8	64		3
	16	256		4
	32	1K		5
	64	4K		6
	128	16K		7
	256	64K	~1 min	8
	512	256K		9
~ 1 sec	1K	1M	~17 min	10
	2K	4M		11
	4K	16M		12
	8K	64M		13
	16K	256M		14
	32K	1G	11.5 dagen	15
~ 1 min	64K	4G		16
	128K	16G		17
	256K	64G		18
	512K	256G		19
~ 17 min	1M	1T	31.7 jaar	20

Sortering en complexiteit

- In het vak Complexiteit wordt nader ingegaan op hoe het aantal bewerkingen afhangt van het algoritme , de datastructuur en het aantal items N
- In dit college zullen we vooral kijken naar die datastructuren die qua **adresseer complexiteit** een O(logN) of O(1) gebruik mogelijk maken
- Bij gesorteerde data items is vaak een O(logN) complexiteit mogelijk vandaar onze speciale aandacht bij datastructuren voor **sorteren, gesorteerd houden** en bewerkingen op **gesorteerd gehouden data items**
- Met **hash** technieken kan O(1) bereikt worden

Voorbeeld toepassingen

- Tijdens dit college zullen we o.a. kijken naar geschikte datastructuren voor:
  - Spellen: dam, schaak, triomino, catan
  - Beeld en videobewerking
  - Historische data: temporele data
  - Geografische data
  - Spatio-temporele data

Ruimte besparing Expliciet/impliciet

- Een belangrijk aspect van de organisatie van het beperkte computergeheugen is:
  - Welke datastructuur gebruikt zo weinig mogelijk geheugenruimte?
  - Vaak kan op veel informatieopslag bespaard worden door data waarden niet **expliciet** op te slaan, maar **impliciet** beschikbaar te maken
  - Impliciet wil zeggen dat in het algoritme door variatie in b.v. tellers de benodigde waarden gegenereerd worden wanneer nodig
  - Voorbeeld: de pixels van een digitale foto hebben zowel een 2D plaatscoordinaat als helderheidswaarde(s); van alle plaatscoordinaten wordt echter alleen het aantal rijen en kolommen opgeslagen; alleen de helderheden worden per pixel expliciet in bytes opgeborgen

Impliciete/Expliciete data opslag

- De verwevenheid van datastructuren en algoritmes is heel sterk als sprake is van impliciete data:
  - Tijdens de werking van het algoritme, b.v. in do loops via tellers, worden benodigde data waarden lokaal gegenereerd
  - Zonder het algoritme is niet eens duidelijk dat andere data dan die welke expliciet opgeslagen is, een rol spelen
  - Reden te meer om een datastructuur en het erop werkende algoritme bij elkaar te houden in een ADT

Veiligheid
Afscherming

- Extra reden voor inkapselen is veiligheid
- Verbergen implementatie details toegangsregeling b.v.
- Verwerken wachtwoorden etc.

Inzichtelijkheid eenvoud

- Loont minimaliseren ruimte/tijdgebruik?
- Complexe datastructuur
- Complex algoritme
- Correctheid?

- Eenvoud
- KIS
- Sneller geprogrammeerd
- Sneller correct
- Ruimte en/of tijd vaak geen struikelblok

Een eerste ADT: de STAPEL

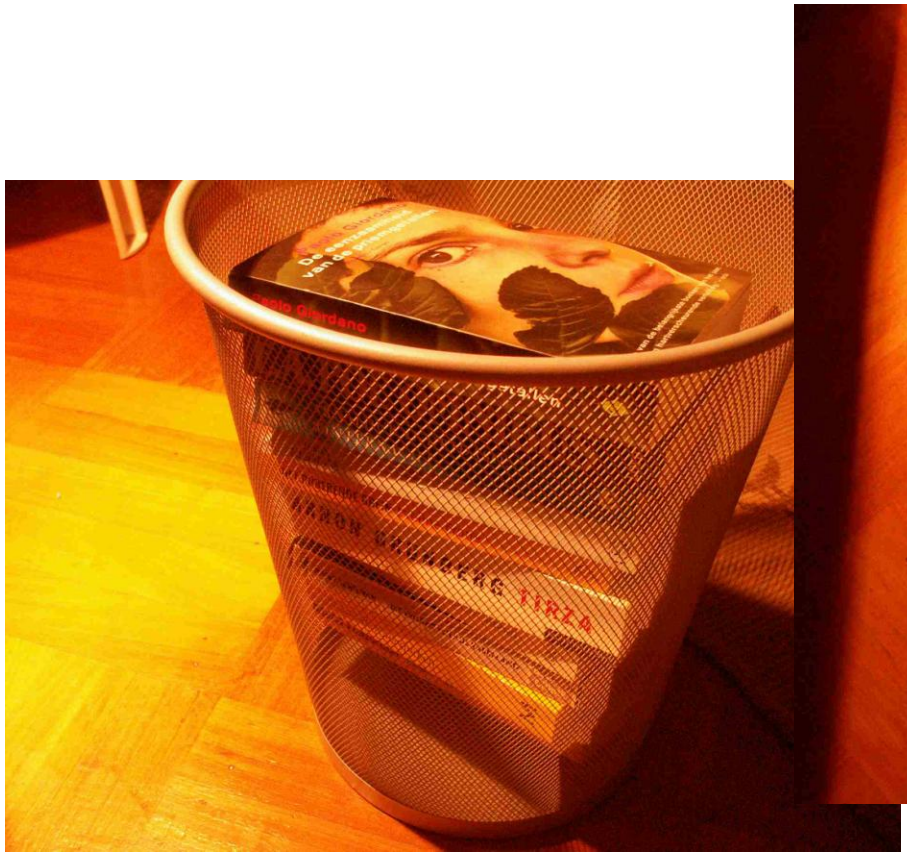
- Kenmerken van een Stapel (stack):
  - Alleen bovenste zichtbaar (top of stack)
  - Bovenste kan eraf genomen worden (pop stack)
  - Nieuw element kan bovenaan toegevoegd worden (push stack)

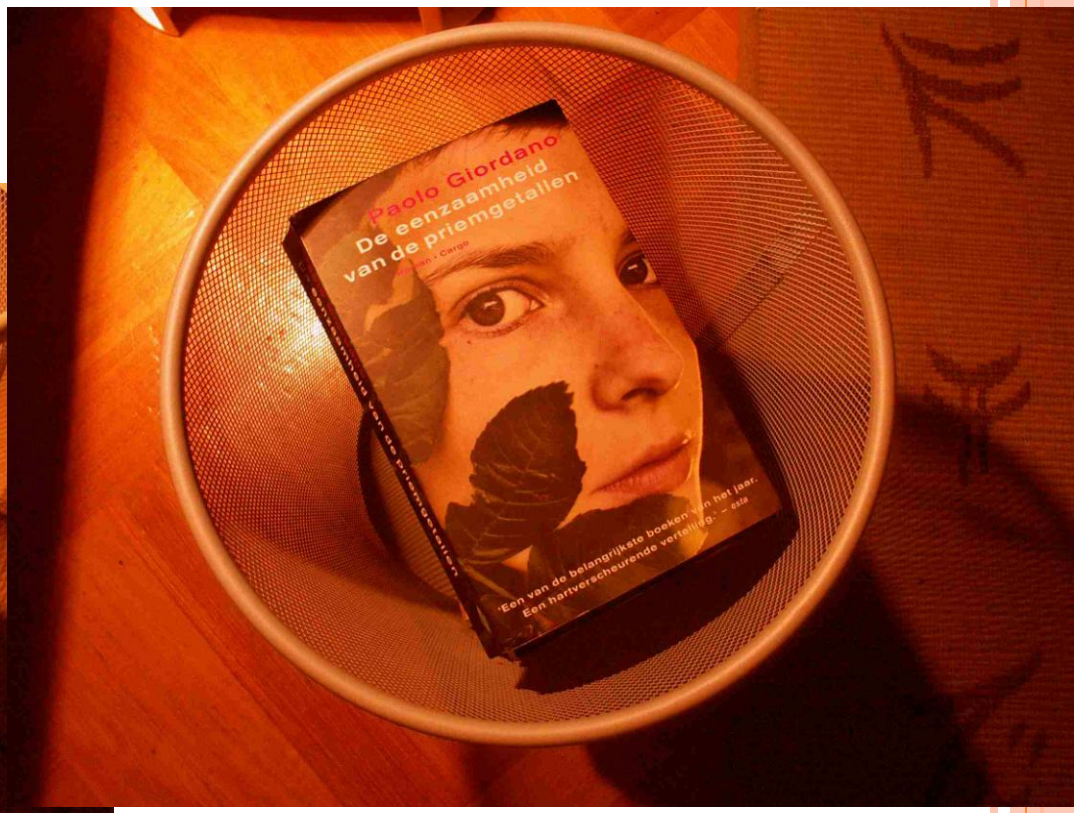
  Randvoorwaarden:

  kan leeg zijn (IsEmpty Y/N)

  (kan ook vol zijn, maar Drozdek houdt daar even geen rekening mee)

Stapel (Stack)





Stapel (stack)

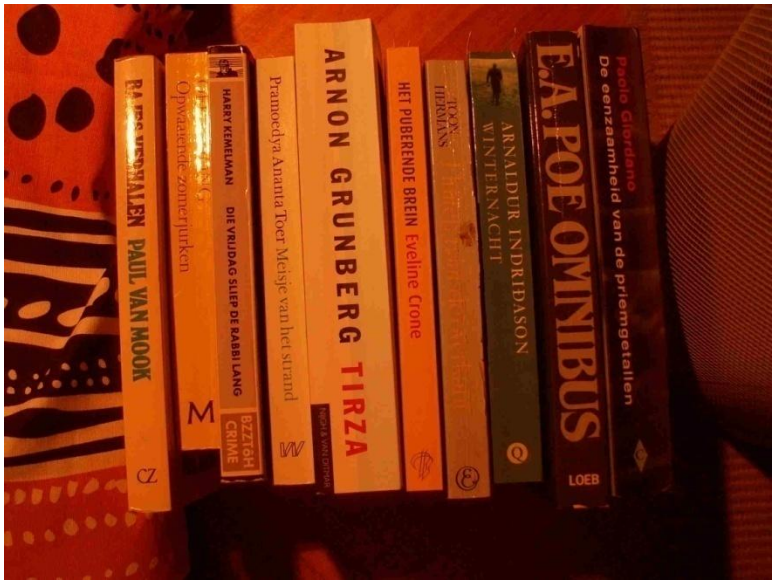
- Rij data elementen (STACK):
  - met bovenste waarde is top of stack (TOS) en
  - communicatie element (COMVAL):
- Waarvan alleen bovenste eraf gehaald kan worden (POP TOS -> COMVAL)
- Waaraan communicatie element bovenop toegevoegd kan worden (PUSH: COMVAL -> TOS)
- Kan leeg zijn (EMPTY: Y staat voor Stack leeg)
- Kan vol zijn (FULL: Y staat voor Stack vol)
- Inherente tijdsvolgorde (LIFO)
- Mogelijke extra voorziening:
- Waarvan alleen bovenste te zien is (TOS -> COMVAL zonder POP)

ADT Stack

- Datastructuur: Lineaire lijst
- Pointer naar laatst toegevoegde (en nog op stack)
- Interne Teller voor aantal op stack (NOS waarde)
- Functie die test op leeg -> geeft Y/N boolean terug
- Functie die test op vol -> geeft Y/N terug
- Functie die COMVAL toevoegt: PUSH(COMVAL)
- Functie die terugkeert met TOS: COMVAL=POP
- Functie voor creatie en annuhilatie stack structuur
- Eventueel nog:
  - Functienaam voor doorgave TOS waarde COMVAL=TOS

Stack: Lineaire lijst implementatie

- Zie los uitgedeelde pseudocode

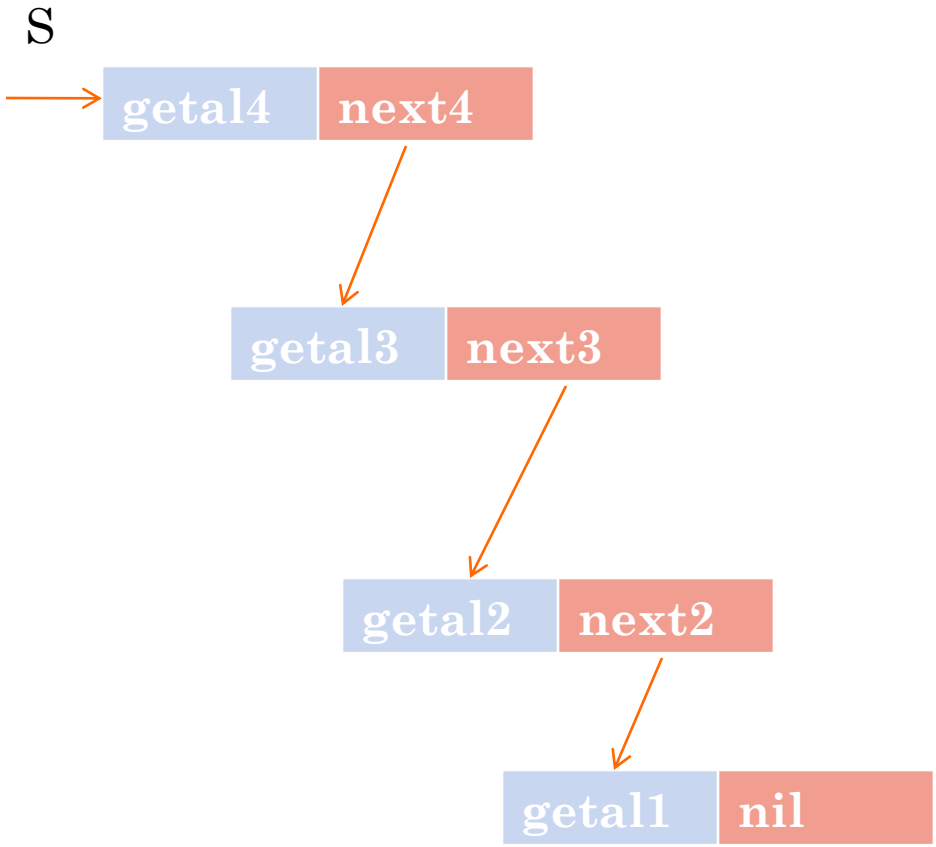


Gebruik van stack in programma

- Pop en Push gaan ervan uit dat er een element bij kan resp. af kan
- Wat gebeurt er als stack vol resp stack leeg?
- Oplossing1:
- Voorkom deze situaties door test vooraf :
  - If (Empty(S)=N) Pop(Stack,element) else?
  - If (Full(S)=N) Push(Stack,element) else?
- Reageer door test achteraf:
  - Aan pop en push procedure boolean return waarde toevoegen, testen na gebruik en goed reageren
  - OS kan programma dan al afgebroken hebben

Stack: Linked list implementatie

- Zie los uitgedeelde pseudocode



Lege stack

LIFO:
DATA ELEMENTEN MET TIJDSTEMPEL

- Losse verzameling dataobjecten met elk een tijdstempel
- Pop: steeds dataobject uit verzameling met recentste tijdstempel teruggeven
- Push: opslaan dataobject in verzameling met huidig tijdstempel

Toepassingen stack

- - omkeren volgorde
- - editten commandline
- - sorteren met 2 of 3 stacks
- - rekenen met polish notation
- - checken geneste haakjes
- - doorgave functie parameters (recursief)
- - oneerlijke rij LIFO (voordring rij)

Gebruik van de stapel om volgorde om te keren (check op palindroom)

- Met behulp van de stapel functies kunnen we een 1 niveau hogere ADT definieren die een stapel en bijbehorende functies gebruikt om een reeks apart ingevoerde karakters qua volgorde om te keren bij uitvoer:
- Gebruik de stapel om karakter voor karakter op de stapel te pushen tot laatste (eind-regel) karakter ingevoerd;
- Haal tot de stapel leeg is, steeds een karakter van de stapel (pop stack) en voer die uit.

Gebruik van een stapel om een commandline te editten

- Stel dat we voor het editten van een commandline instruktie de backspace (voorgesteld als *) hebben om een voorgaand karakter ongedaan te maken
- We maken m.b.v. een stapel een commandline-edit ADT werkend op losse karakters die:
  - Zolang er geen eind-regel karakter (einde invoer) komt:
    - als invoer karakter * is 1 karakter “poppen” van stack, anders invoer karakter “pushen” op stack

    Vervolgens: invoer 1 voor 1 karakter van stack “poppen” en commando-regel opbouwen tot stack leeg

    Command-line ter executie aan OS aanbieden

    b.v. in: grep feee*st tekk*st.lst -> tsl.tsket tseef perg

    uit: grep feest tekst.lst -> OS

Hoeveel stapels nodig om te sorteren?

- 5 minuten bedenktijd:
- Werk op papier een sorteer ADT uit die, met behulp van zo weinig mogelijk stapel ADTs, een invoer van losse integers gesorteerd uitvoert (elke stapel heeft 1 I/O-veld)