Informatica

1ste Bachelor EW-TEW-HI

Corneel Weyers

Hoofdstuk 1

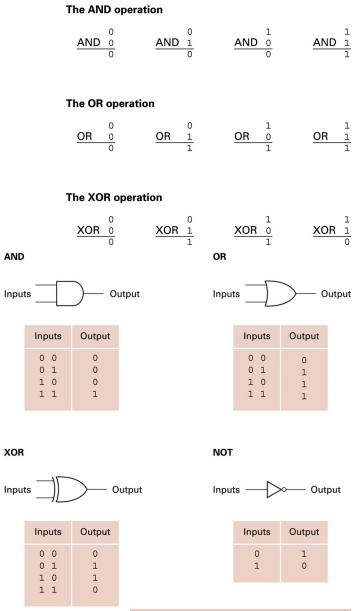
Gegevensopslag

1. Bits en hoe ze worden opgeslagen

Bit = binary digit → 0 en 1

Booleaanse algebra
waar = 1, niet waar = 0
⇒ AND, (I)OR, XOR, NOT

- AND: 0 heeft voorrang
- OR: 1 heeft voorrang
- XOR: verschillend is waar, dezelfde is niet waar
- NOT: 1 is niet waar, 0 is waar

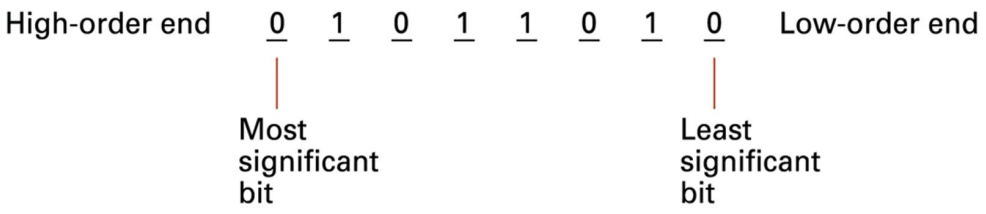


- gates:
- flip-flop: gebouwd met gates
  - VLSI: very large-scale integration (miljoenen)
- hexadecimale notatie = verkorte notatie voor het noteren van bitstromen
  - wiskundig: waarden van 0 tot F ⇒ 16 waarden
  - informatica: bit patronen als veelvoud van 4 (1 symbool voor 4 bits)

Bit pattern	Hexadecimal representation
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

2. Het werkgeheugen van de computer

- cel: groep van acht schakelingen die elk een bit kunnen bevatten = een byte
  → heeft een unieke naam (een adres)



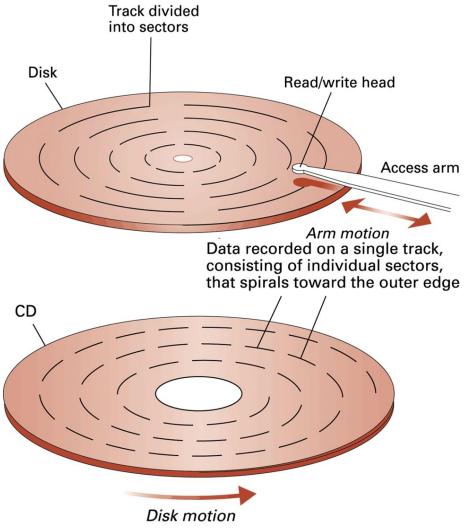
Random Access Memory (RAM)
→ elke cel heeft een eigen adres en kan dus onafhankelijk van andere cellen bereikt/bewerkt worden
Dynamisch RAM (DRAM of SDRAM)

Aanduiden van opslagcapaciteit:

- Kilobyte (KB) = 2^{10} bytes = 1024 bytes
  = +/- 1000 bytes
- Megabyte (MB) = 2^{20} bytes = 2^{10} KB
  = +/- 1 000 000 bytes
- Gigabyte (GB) = 2^{30} bytes = 2^{20} KB = 2^{10} MB
  = +/- 1 000 000 000 bytes
- Terabyte (TB) = 2^{40} bytes = 2^{30} KB = 2^{20} MB = 2^{10} GB
  = +/- 1 000 000 000 000 bytes

3. Massageheugen

- magnetisch schijfgeheugen:
  - HDD
  - een track (bevat reeksen) kan meer informatie bevatten dan nodig
  - aanbrengen van sporen en sectoren = formatteren
  - kwaliteit:
    - zoektijd (seek time)
    - rotatie-vertraging (rotation-delay)
    - toegangstijd (access time)
    - overzettijd (transformatie)
- optisch schijfgeheugen:
  - wordt gelezen door een laser → variaties in het oppervlak (van binnen naar buiten)
  - best met lange continue data strings
- flashgeheugen:
  - massageheugen zonder mechanische component → niet meer nadeel van tragere snelheid
  - bit = elektron in minuscuul vakje siliciumdioxide



⇒ opslagplaats:

gewoon magnetisch schijfgeheugen	0,5 KB tot KB's
meerdere magnetische schijfgeheugens	honderden GB's tot TB's
CD (compact disk)	+/- 700 MB

DVD (digital versatile disk)	GB's
BD (blue-ray disk)	5xDVD
flash drive: USB	max 512 GB
flash drive: SSD	max 2 TB
geheugenkaart: SD	2 GB
geheugenkaart: SDHC	32 GB
geheugenkaart: SDXC	TB's

4. Gegevens voorstellen met bitpatronen

Code met uniek bitpatroon per tekst symbool

- ASCII: 8 bits ⇒ 2^8 = 256 unieke patronen
- Unicode: 16 bits ⇒ 2^{16} = 65536 patronen
- Text file: bestand met een lange reeks symbolen gecodeerd met ASCII of Unicode

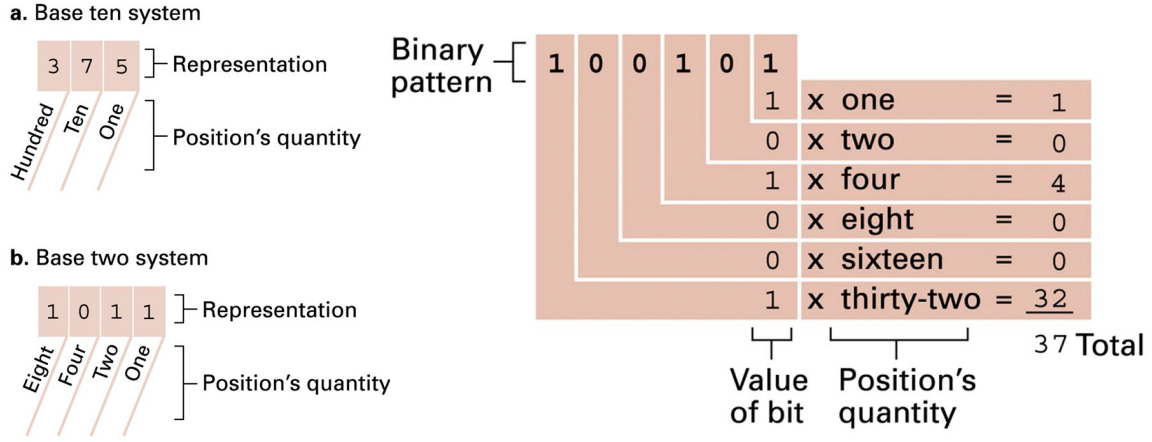
Beelden → bitmap techniek:

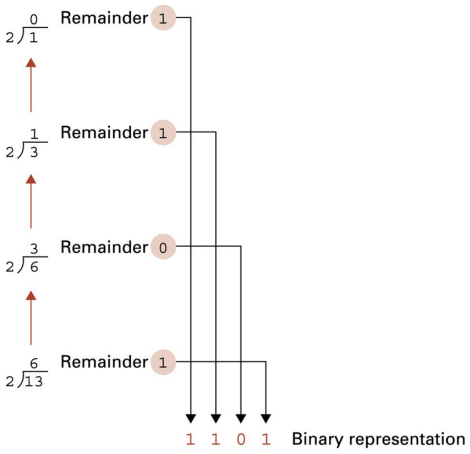
- pixels
- bitmap: reeks bits die de pixels van een afbeelding codeert
- zwart vs kleur: zwart is 1 bit per pixel
    - alternatief voor kleur: CAD → computer aided design

Geluid → sampling:

- met een frequentie van 44100 keer per seconde wordt de amplitude van de geluidsgolf gemeten
- opgeslagen 16 bits per sample (of 32 voor stereo)
- alternatief: MIDI → welk instrument welke noot hoe lang aanhoudt

5. Het binair talstelsel





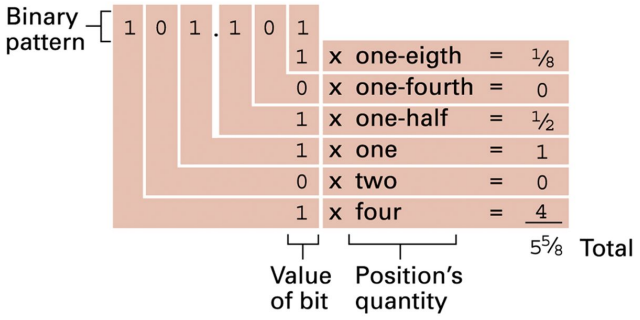
Step 1. Divide the value by two and record the remainder.

Step 2. As long as the quotient obtained is not zero, continue to divide the newest quotient by two and record the remainder.

Step 3. Now that a quotient of zero has been obtained, the binary representation of the original value consists of the remainders listed from right to left in the order they were recorded.

Optellen van binaire getallen:

```
  0     1     0     1
+ 0   + 0   + 1   + 1
  0     1     1    10
```



6. Opslaan van gehele getallen

- Gehele getallen kunnen positief of negatief zijn
- Conventie nodig voor het voorstellen van het teken
  - 2-complementnotatie
    - Gebruikt een vast aantal bits, vb. 32 bits => 2^{32} waarden in het bereik [-2147483648, 2147483648[
  - Excess-notatie
    - Variant waarbij het tekenbit omgekeerd wordt

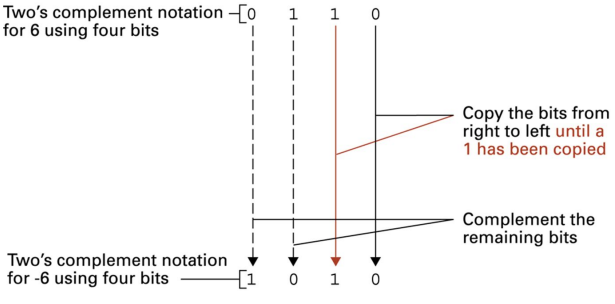
a. Using patterns of length three

Bit pattern	Value represented
011	3
010	2
001	1
000	0
111	-1
110	-2
101	-3
100	-4

Meest significante bit is het tekenbit:
• Negatief => 1
• Nul of Positief => 0

b. Using patterns of length four

Bit pattern	Value represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

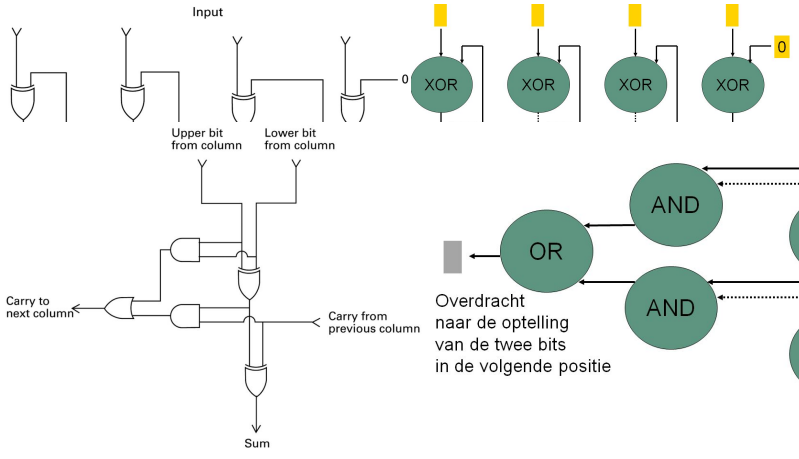


Uitleg: Eén van de nuttige eigenschappen van de 2-complementnotatie is dat een negatief (of positief) getal gemakkelijk afgeleid kan worden van het overeenkomstige positief (of negatief) getal. Vertrekkende van het lower-order end (de minst significante bits) zijn de bitpatronen gelijk aan elkaar tot en met de eerste 1. Vanaf dan zijn ze elkaars **complement** (d.w.z. alle nullen zijn énen en alle énen zijn nullen).

Problem in base ten	Problem in two's complement	Answer in base ten
3 + 2	0011 + 0010 = 0101	5
−3 + −2	1101 + 1110 = 1011	−5
7 + −5	0111 + 1011 = 0010	2

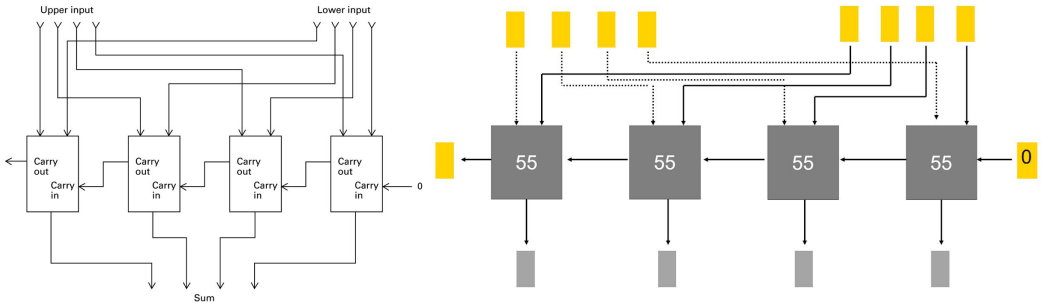
Uitleg: Wanneer gehele getallen in 2-complementnotatie voorgesteld worden, is het niet nodig om aparte schakelingen te voorzien voor het berekenen van het verschil tussen twee getallen. Het volstaat wanneer schakelingen voorzien zijn voor optelling en voor het negatief maken van een getal.

<u>Schakeling voor omkeren teken</u>:



→ pg 578-580<u>Schakeling voor optellen van 2 bits</u>: → pg 578-580

<u>Schakeling voor optellen van 2 getallen</u>:



Uitleg: Te gebruiken voor 4-bit patronen. Deze schakeling noemt men een ***ripple adder***. De “55” verwijst naar de schakeling die op slide 55 afgebeeld staat. Merk op dat we hier gebruik maken van het abstractie (of *black box*) mechanisme.

Overflow

→ het voor te stellen getal valt buiten het bereik

- Bij negatieve getallen => tekenbit = 0 en overdracht van 1 naar niet-bestaande volgende positie
- Bij positieve getallen => tekenbit = 1

Excess-notatie:
→ De tabel geeft de excess-8-notatie weer. Het enige verschil met de 2-complementnotatie voor bitpatronen van lengte 4 is dat het tekenbit omgekeerd is (dus 0 voor negatieve getallen en 1 voor nul of positieve getallen).

Bit pattern	Value represented
111	3
110	2
101	1
100	0
011	−1
010	−2
001	−3
000	−4

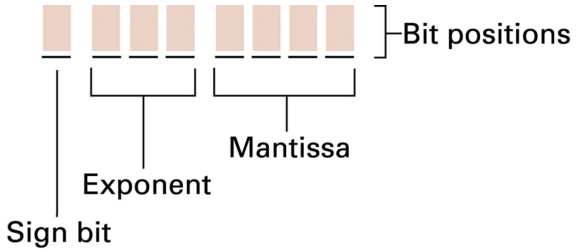
Bit pattern	Value represented
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	−1
0110	−2
0101	−3
0100	−4
0011	−5
0010	−6
0001	−7
0000	−8

7. Het opslaan van reële getallen

⇒ floating-pointnotatie

- Mantisse
  - Bitpatroon voor geheel deel en breukdeel
  - Begint altijd met 1 => genormaliseerde vorm
- Exponent
  - Positie van het radixpunt
- Tekenbit (zoals bij 2-complementnotatie)
  - 0 positief
  - 1 negatief

Voorbeeld: acht-bitindeling



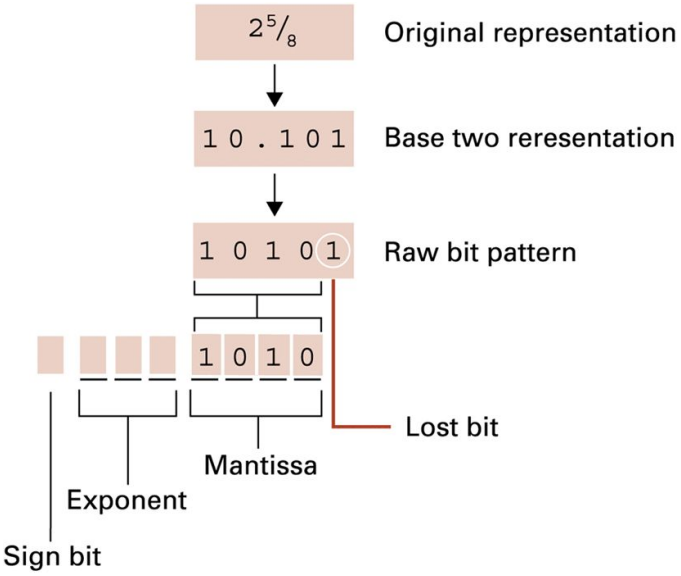
→ welke reëel getal wordt voorgesteld door 10111001?

- Mantisse = 1001
- Exponent = 011
    - Excess-4-notatie voor decimale -1
    - Radixpunt schuift in de mantisse één positie naar links
    - Mantisse wordt .01001
      wat gelijk is aan 1/4 + 1/32 = 9/32
- Tekenbit = 1
    - -9/32 = -0,28125

→ Voorbeeld 1: welke reëel getal wordt voorgesteld door 10111001?

- Mantisse = 1001
- Exponent = 011
    - Excess-4-notatie voor decimale -1
    - Radixpunt schuift in de mantisse één positie naar links
    - Mantisse wordt .01001
      wat gelijk is aan 1/4 + 1/32 = 9/32
- Tekenbit = 1
    - -9/32 = -0,28125

Afkapfouten:



Uitleg: Exponent wordt 110 en tekenbit wordt 0 dus gehele bitpatroon wordt 01101010. Als je dit terug naar decimaal omzet krijg je 2,5. Er is dus een afkapping van de minst significante bit gebeurd waardoor het getal 2,625 is afgerond naar 2,5 (dus afrondingsfout = 0,125). Om het effect van dergelijke fouten te reduceren worden minstens 32 bits (**Single Precision Floating Point**: 1 tekenbit, 8 exponentbits, 23 mantissebits) (soms 64 bits – **Double Precision Floating Point**) gebruikt voor het opslaan van getallen in floating-pointnotatie. De acht-bitindeling die hier in de cursus gebruikt wordt dient louter als voorbeeld.

Hoofdstuk 2

Gegevensverwerking

1. Computerarchitectuur

De afbeelding toont de Intel Core i7 processor (i7-4771) die gelanceerd werd in september 2013, met als belangrijkste kenmerken een registerbreedte van 64 bits, een kloksnelheid van 3,50 GHz, 4 kernen, 256KB cachegeheugen per core (= kern) en een gedeeld cachegeheugen van 8 MB.
***Cachegeheugen** bestaat uit geheugencellen binnen de CPU. Hier kan men kopieën van de inhoud van het werkgeheugen in opslaan, wat gegevensverkeer tussen de CPU en het werkgeheugen vermindert, waardoor de gegevensverwerking sneller kan gebeuren.*

→ Central Processing Unit (CPU): gebouwd uit schakelingen die gegevens bewerken en coördineren (bv. op moederbord van een computer in de vorm van microprocessor)

- **kloksnelheid**: 1 GHz, betekent dat er 1 machine cycli per seconde kan uitvoeren
- **Cachegeheugen:** geheugencellen binnen de CPU. Hier kan men kopieën van de inhoud van het werkgeheugen in opslaan, wat gegevensverkeer tussen de CPU en het werkgeheugen vermindert, waardoor de gegevensverwerking sneller kan gebeuren.

Onderdelen:

De **algemene registers** bevatten de waarden die voor de ingangen (of invoerlijnen) van de schakelingen in de wiskundige/logica-eenheid gebruikt worden. Zij worden ook gebruikt voor het opslaan van resultaten (= uitvoer van de schakelingen van de wiskundige/logica-eenheid). De wiskundige/logica-eenheid kan dus niet rechtstreeks werken op de waarden die opgeslagen zijn in de geheugencellen van het werkgeheugen. De besturingseenheid dient eerst deze waarden te kopiëren naar de algemene registers en de wiskundige/logica-eenheid te laten weten in welke registers de te bewerken gegevens staan en in welke registers de resultaten geplaatst moeten worden. Ook deze resultaten van de gegevensverwerking dienen door de besturingseenheid naar het werkgeheugen gekopieerd te worden.

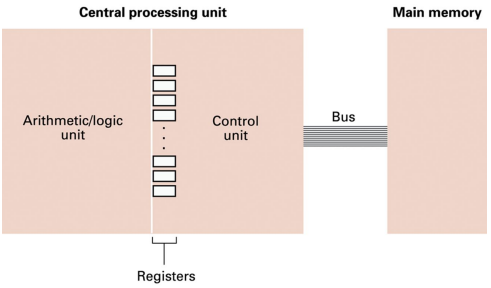
- wiskundige/logica-eenheid: schakelingen voor het uitvoeren van bewerkingen op gegevens
- besturingseenheid (Control Unit): schakelingen voor het coördineren van de activiteiten van de computer
- registers: geheugencellen voor het tijdelijk opslaan van gegevens ‘in bewerking’

Bus

De bus is een verzameling van draadjes via dewelke bitpatronen tussen de CPU en het werkgeheugen verplaatst worden. Gegevens uit het werkgeheugen halen (lees: kopiëren, want ze verdwijnen niet echt) en verplaatsen naar de CPU noemen we ‘lezen’. Dit gebeurt door het verzenden via de bus van het adres van de geheugencel plus een leessignaal. Gegevens verplaatsen van de CPU naar het werkgeheugen is dan ‘schrijven’. Dit gebeurt door het verzenden van het adres van de geheugencel plus een schrijfsignaal.
→ maakt transport mogelijk van de werkgeheugen naar de registers (lezen) of omgekeerd (schrijven)

Onderdelen:

- controle bus: gaan we lezen of schrijven?
- adres bus: de adressen van de cellen in het werkgeheugen worden geadresseerd (afhankelijk van lezen of schrijven)
- data bus: bitpatronen worden getransporteerd



Step 1. Get one of the values to be added from memory and place it in a register. → leesbewerking

Step 2. Get the other value to be added from memory and place it in another register. → leesbewerking

Step 3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result. → optelschakeling activeren, ingangen van de schakeling worden verbonden met de registers

Step 4. Store the result in memory. → controlebus zal schrijf signaal hebben (CPU naar werkgeheugen

Step 5. Stop.

Stored Program Concept

*Een computerarchitectuur die gebruik maakt van het stored program concept noemt men een **von Neumann-architectuur**. Doordat programma's net zoals gegevens opgeslagen worden in het werkgeheugen is het mogelijk dat programma's andere programma's (of zichzelf) gaan bewerken. Programma's kunnen zich dus aanpassen aan de omgeving of 'leren'.*

→ instructies zijn gewoon bitpatronen in het werkgeheugen, besturingseenheid laadt deze 'gegevens' in de CPU (zo wordt een computer programmeerbaar)
⇒ computerarchitectuur op deze wijze heet **'von Neumann-architectuur'**

2. Machinetaal

- machine-instructie: gecodeerd als een bitpatroon dat herkend kan worden door de CPU
- machinetaal: verzameling van alle soorten machine-instructies die door de CPU herkend kunnen worden

CPU-ontwerpfilosofieën:

PowerPC-processoren werden oorspronkelijk gebruikt in de Apple MacIntosh computers, maar door de daling in productiekosten van CISC-processoren gebruikt Apple nu ook Intel processoren.
Terwijl CISC processoren standaard gebruikt worden in desktop computers, worden RISC processoren (vb. van ARM – Advanced RISC Machine en o.a. geproduceerd door Texas Instruments) veel gebruikt in game controllers, digitale TV, smartphones, etc., omdat ze minder elektriciteit verbruiken.

- Reduced Instruction Set Computing (RISC): machinetaal definieert klein aantal, eenvoudige, snelle en efficiënte soorten machine -instructies
- Complex Instruction Set Computing (CISC): machinetaal definieert groot aantal, complexe, gemakkelijke en krachtige soorten machine-instructies

Soorten instructies:

- gegevensopdrachtsgroep
  - kopiëren van gegevens van/naar werkgeheugen naar/van CPU (LOAD-instructie & STORE-instructie)
  - Input/Output activiteiten (bv. toetsenbord, printer, …)
- wiskunde/logica-groep
  - bewerkingen binnen de wiskundige/logica-unit

Voorbeeld van AND masker: zet de minst significante bit op 0

masker:	11111110
bitpatroon:	10011001
AND:	10011000

Voorbeeld van OR masker: zet de tweede bit van het high-order uiteinde op 1

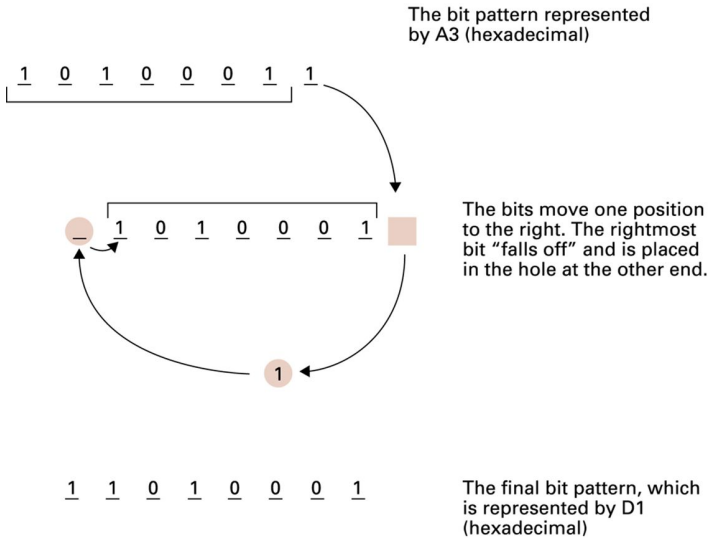
masker:	01000000
bitpatroon:	10011001
OR:	11011001

Voorbeeld van XOR masker: neem het complement van de derde en vierde bit van het high-order uiteinde

masker:	00110000
bitpatroon:	10011001
XOR:	10101001

UNIVERSITEIT GENT

14



→ voorbeeld van ROTATE bewerking

- besturingsgroep: coördineren van de uitvoering van het programma (STOP/HALT en JUMP/BRANCH)

Step 1. LOAD a register with a value from memory.

Step 2. LOAD another register with another value from memory.

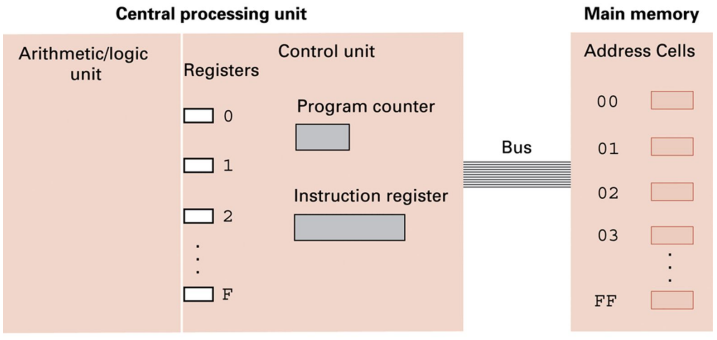
Step 3. If this second value is zero, JUMP to Step 6.

Step 4. Divide the contents of the first register by the second register and leave the result in a third register.

Step 5. STORE the contents of the third register in memory.

Step 6. STOP.

Uitleg: De machine-instructies bij stappen 1, 2 en 5 behoren tot de gegevensoverdrachtsgroep. Deze van stap 4 behoort tot de Wiskunde/Logicagroep. De instructies van stappen 3 en 6 behoren tot de besturingsgroep. Merk op dat de JUMP in stap 3 een voorwaardelijke sprong is, waarmee we een deling door nul vermijden.

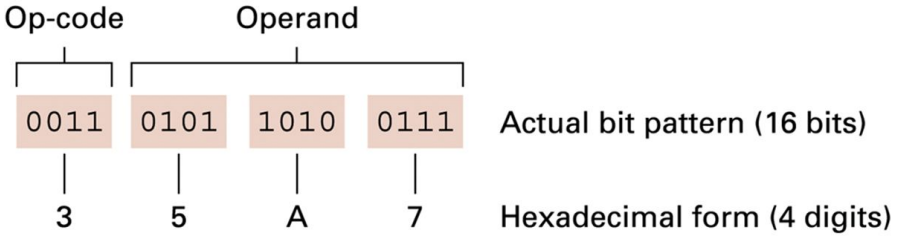


Uitleg: De machinetaal die als illustratie gebruikt wordt verondersteld een computerarchitectuur waarbij er 16 algemene registers zijn en 256 geheugencellen. De algemene registers en geheugencellen zijn elk 1 byte groot (dus 8 bits). Elke geheugencel is individueel adresseerbaar; we moeten dus 256 verschillende adressen kunnen vormen. Dit kan met bitpatronen van lengte 8. Voor het gemak zullen we deze bitpatronen in hexadecimale notatie voorstellen, dus van 00 tot FF. Ook de registers zijn individueel adresseerbaar. Omdat we hier 16 verschillende adressen moeten kunnen vormen volstaan bitpatronen van lengte 4. Deze adressen gaan we voor het gemak ook hexadecimaal voorstellen, dus van 0 tot F.

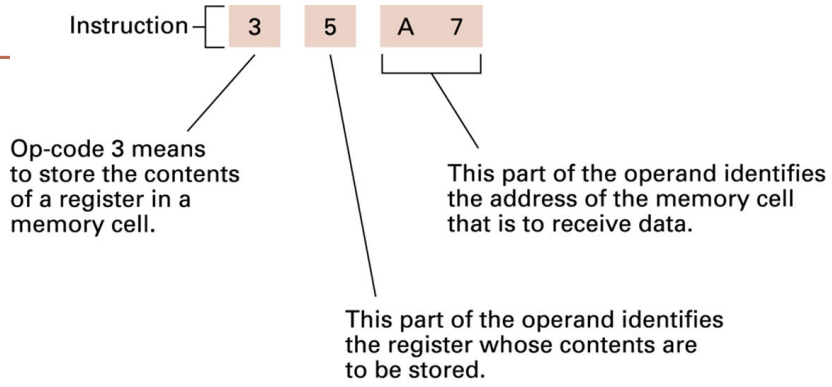
- werkgeheugen heeft 256 (2^8) cellen met eigen adressen (geadresseerd met 8 bits, van 00 tot FF)
- 16 registers (2^4) (geadresseerd met 4 bits, van 0 tot F)
    - program counter & instruction register zijn speciale registers

Structuur machine-instructies:

- lengte van 2 bytes: op-codeveld (eerste 4 bits) en operandveld (volgende 12 bits)
- 16-bitpatronen worden voorgesteld met 4 hexadecimale cijfers (worden gegeven in bijlage)



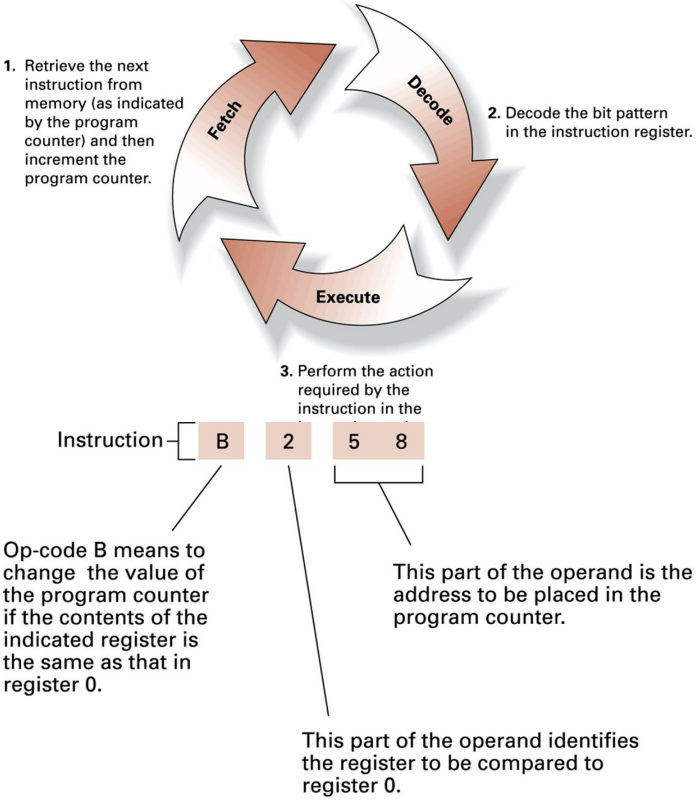
Encoded instructions	Translation
156C	Load register 5 with the bit pattern found in the memory cell at address 6C.
166D	Load register 6 with the bit pattern found in the memory cell at address 6D.
5056	Add the contents of register 5 and 6 as though they were two's complement representation and leave the result in register 0.
306E	Store the contents of register 0 in the memory cell at address 6E.
C000	Halt.



3. Programma-uitvoering

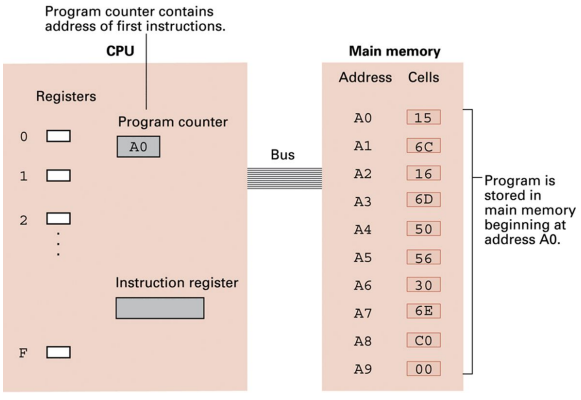
- speciale registers in besturingseenheid
    - **programmateller**: bevat het adres van de volgende instructie die uitgevoerd moet worden
    - **instructieregister**: in CPU, laadt het register in dat moet uitgevoerd worden
- machinecyclus
    - algoritme dat de besturingseenheid constant uitvoert (ophalen, decoderen en uitvoeren van één machine-instructie)

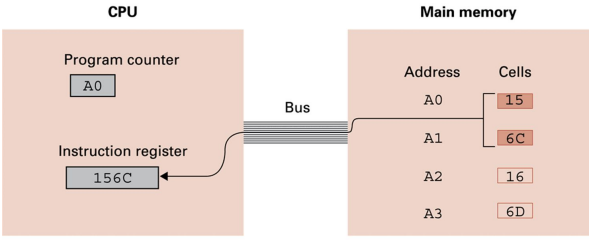
Voorwaardelijke sprong:



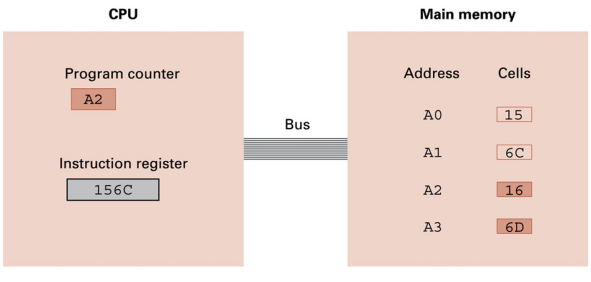
Uitleg:Omdat de machine-instructies 16 bits lang zijn, beslaan zij twee opeenvolgende geheugencellen van 8 bits. De programmateller wordt bijgevolg telkens met 2 verhoogd. Het instructieregister moet 2 bytes lang zijn om de gehele instructie te kunnen bevatten. Het decoderen van het operandveld van een instructie gebeurt op basis van de op-code. Het uitvoeren van de instructie gebeurt door het activeren van de schakelingen die bij de gewenste bewerking horen. Dit kunnen bijvoorbeeld schakelingen zijn in de wiskundige/logica-eenheid. Zodra de instructie is uitgevoerd, begint de machine-cyclus opnieuw (dus met het ophalen van de volgende instructie uit het werkgeheugen vanaf het adres aangegeven door de programmateller).
Een belangrijk kenmerk van de machinecyclus is de snelheid waarmee ze uitgevoerd wordt. Dit noemt men de **kloksnelheid** van de computer. De meeteenheid is de Hertz (Hz) wat staat voor één machine-cyclus per seconde. PCs (laptops, notebooks) werken aan kloksnelheden in de grootte-orde van één tot enkele GigaHertz (GHz), oftewel miljarden machine-cycli per seconde.

Voorbeeld van programma-uitvoering (zie boek!!!):





a. At the beginning of the fetch step the instruction starting at address A0 is retrieved from memory and placed in the instruction register.



b. Then the program counter is incremented so that it points to the next instruction.

1.Zet de adressen en de inhoud van de geheugencellen in hexadecimale notatie
2.Voer het programma uit met programmateller op adres 00
3.Wat is na uitvoering de inhoud van de geheugen?
En met adres FE?
4.Vergelijk met de inhoud van de geheugencel met adres FD en ga na welk algoritme door het programma geïmplementeerd wordt

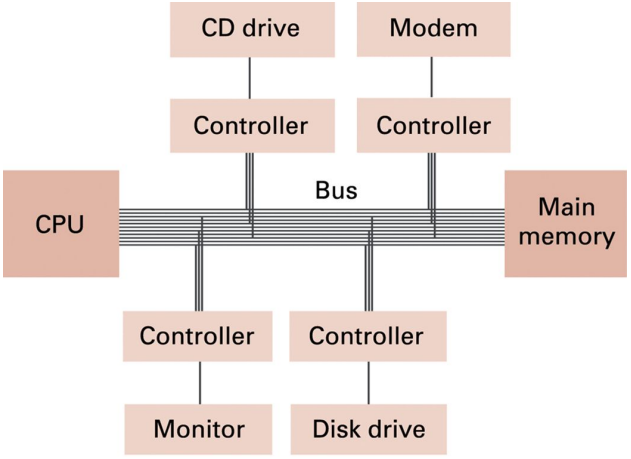
adres	cel	adres	cel	adres	cel	adres	cel
0000 0000	0010 0000	0000 1011	0010 0110	0001 0110	0001 0001	0010 0001	0001 0010
0000 0001	0000 0001	0000 1100	0010 0010	0001 0111	1111 1110	0010 0010	0011 0000
0000 0010	0011 0000	0000 1101	0000 0010	0001 1000	0101 0000	0010 0011	1111 1101
0000 0011	1111 1110	0000 1110	0001 0001	0001 1001	0001 0010	0010 0100	1011 0000
0000 0100	0011 0000	0000 1111	1111 1111	0001 1010	0011 0000	0010 0101	0000 0110
0000 0101	1111 1111	0001 0000	0101 0000	0001 1011	1111 1110	0010 0110	1100 0000
0000 0110	0001 0001	0001 0001	0001 0010	0001 1100	0010 0010	0010 0111	0000 0000
0000 0111	1111 1101	0001 0010	0011 0000	0001 1101	1111 1111	…	…
0000 1000	0010 0000	0001 0011	1111 1111	0001 1110	0001 0001	1111 1101	0000 0101
0000 1001	0000 0001	0001 0100	0001 0010	0001 1111	1111 1101	1111 1110	0000 0000
0000 1010	1011 0001	0001 0101	1111 1111	0010 0000	0101 0000	1111 1111	0000 0000

4. Gebruik van randapparatuur

Controller:

Een controller is dus eigenlijk ook een computer met eigen geheugenschakelingen en CPU. De connectoren op de computerbehulzing die verbonden zijn met de controller(s) worden ook wel poorten genoemd (hoewel we verderop een tweede betekenis van het concept 'poort' zullen tegenkomen).

- Microprocessor die met het uitvoeren van een eigen programma de activiteiten van een randapparaat bestuurt
- Aangebracht op moederbord van de computer en via bekabeling/connectoren op intern/extern randapparaat
- Verbonden met de bus tussen CPU en werkgeheugen



Uitleg: De controllers kunnen de signalen die verzonden worden tussen CPU en werkgeheugen in de gaten houden en zelf ook signalen verzenden. Oorspronkelijk was elke controller ontworpen voor een bepaald type van randapparaat. Het vervangen van een randapparaat (vb. nieuwe harde schijf installeren) betekende dan vaak dat ook de controller vervangen moest worden.

Universal serial bus (USB)

Een alternatief voor USB is FireWire (ontwikkeld door Apple). De USB technologie is goedkoper (vandaar zijn populariteit in het gamma van PC-devices zoals printers, scanners, computermuis, toetsenborden,

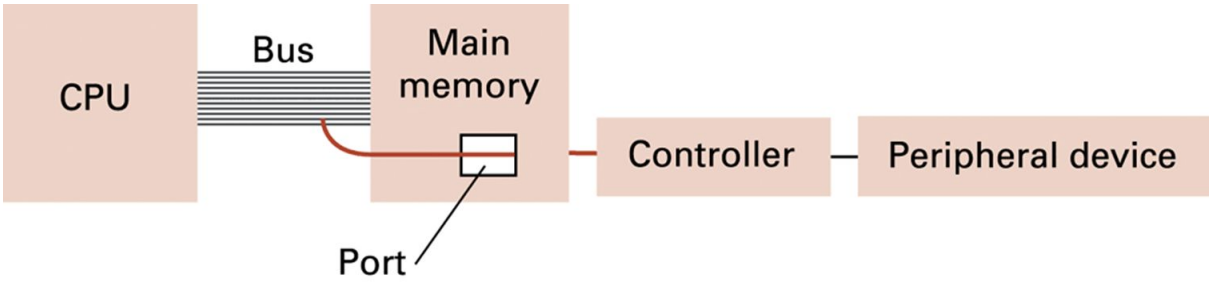
smartphones, digitale camera's, flash drives en externe harde schijven). De FireWire technologie is sneller en wordt bijvoorbeeld gebruikt bij video recorders.

- Standaard ontwikkeld door Intel waarbij controllers algemeen bruikbaar worden
    - Controller vertaalt interne computersignalen naar standaard USB signalen (en omgekeerd)
    - USB compatible randapparaat zorgt voor omzetting van standaard USB signalen naar eigen interne signalen (en omgekeerd)
    - Apparaat is aangesloten op een 'USB poort' (d.w.z. connector)
- Vervangen van een randapparaat betekent niet meer automatisch vervangen van de controller (maar apparaten moeten USB compatible zijn)

Communicatie met randapparaten

- I/O-instructies
    - Vergelijkbaar met LOAD/STORE instructies maar met adressen die verwijzen naar locaties binnen controllers
- Poort
    - Unieke set van adressen (I/O-adressen) voor een controller

Memory-mapped I/O-systeem:



Uitleg: Bij een memory-mapped I/O systeem bevat de machinetaal geen I/O-instructies maar worden dezelfde LOAD/STORE instructies gebruikt als bij de gegevensoverdracht tussen CPU en werkgeheugen. De controllers en het werkgeheugen moeten dan ontworpen worden om te reageren op berichten die verwijzen naar 'hun' bereik van adressen.

Direct Memory Access

Lezen en schrijven van/naar randapparatuur zoals de harde schijf is relatief traag ten opzichte van het uitvoeren van berekeningen binnen de CPU. De snelheden van mechanische randapparatuur worden uitgedrukt in grootte-ordes van milli- en micro seconden, terwijl de schakelingen van de CPU werken aan snelheden in het bereik van de nano-seconden. Lees/schrijf bewerkingen met randapparatuur kunnen dus duizenden tot miljoenen keren trager zijn dan lees/schrijfbewerkingen van/naar het werkgeheugen van de computer of het uitvoeren van berekeningen binnen de wiskundige/logica-unit. Een computer wordt efficiënter benut door de relatief trage I/O bewerkingen uit te besteden aan controllers, zodat de CPU niet moet wachten op het uitvoeren van deze bewerkingen en intussen ander werk kan verrichten. (zie ook hoofdstuk 3 Besturingssystemen). Een nadeel van DMA is dat de communicatie over de bus toeneemt, wat kan leiden tot de zogenaamde **von Neumann-bottleneck**.

→ Controller communiceert via de bus rechtstreeks met het werkgeheugen

- Efficiënt bij relatief trage bewerkingen
- CPU zendt lees/schrijf-opdrachten naar controller die ze uitvoert
    - Vb. schijfcontroller leest sector van harde schijf en plaatst inhoud in werkgeheugen, terwijl CPU andere programmaopdrachten uitvoert

Handshaking en statuswoord

- Gegevensoverdracht tussen CPU en randapparaat is veelal tweerichtingsverkeer
    - Informatie nodig over de status van het apparaat en voor het coördineren van de activiteiten
- Deze continue dialoog noemt men handshaking
    - Hierbij wordt een statuswoord gebruikt
- Bitmap met bits die status apparaat aangeven
    - Vb. Bij een printer, een bit voor 'geen papier', een bit voor 'printer gereed', een bit voor 'paper jam', enz.
    - Hierop wordt gereageerd door CPU of controller zelf

Communicatiemedia

Bij parallelle communicatie worden verschillende bits van een signaal gelijktijdig via verschillende lijnen verzonden. Bij seriële communicatie wordt slechts één bit tegelijk verzonden. Ethernet is een populaire verzameling van standaarden voor het bouwen van een ***Local Area Network*** *(LAN).*

- Interne computerbus
    - Parallelle communicatie
    - Snel, maar complex
- USB/FireWire (max. enkele meters)
    - Seriële communicatie
    - Traag, maar eenvoudig
- LAN (binnen een gebouw)
    - Kabels (vb. Ethernet) of draadloos (vb. WiFi)
    - Seriële communicatie

Communicatiesnelheden

- Snelheid waarmee bits van de éne computercomponent naar de andere verzonden worden
- Eenheden: bps, Kbps, Mbps, Gbps
    - USB/FireWire: honderden Mbps
    - Ethernet: 10 Mbps – 100 Gbps
- Maximum snelheid = bandbreedte

Hoofdstuk 3

Besturingssystemen

1. Rol van het besturingssysteem

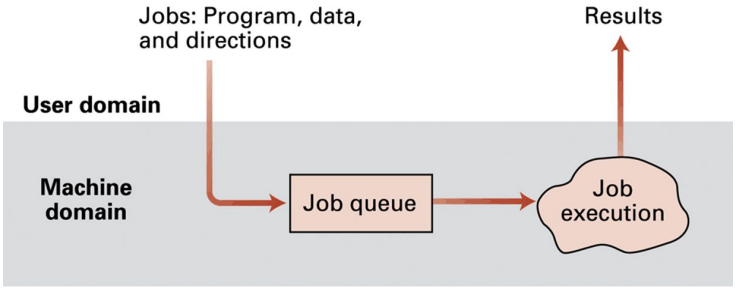
De gebruiker van een computer kan via de (doorgaans grafische) interface die het besturingssysteem voorziet programma's laten uitvoeren, deze uitvoering onderbreken of stopzetten.
→ beheren van de computer

2. Evolutie in computerbesturing

Een programma dat uitgevoerd moet worden, wordt een job of taak genoemd. FIFO (First In First Out) is het principe dat bij de uitvoering van de taken in de wachtrij gehanteerd wordt. Hierop zijn uitzonderingen mogelijk als aan taken verschillende prioriteiten toegekend worden.

Batchverwerking:
computeroperator als interface gebruiken tussen gebruikers en computer

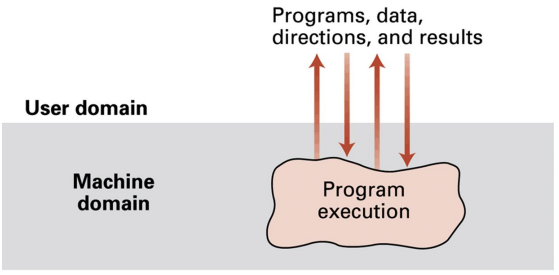
- gegevens, instructies en aanwijzingen in massageheugen opslaan
- taakwachtrij (job queue)
- taken in het massageheugen opgeslagen zonder verdere interactie met de gebruiker uitvoeren ⇒ volgorde van uitvoering = FIFO (First In First Out)
  → uitzondering: taken die prioriteit hebben gekregen



Nadeel: De gebruiker heeft geen interactie met het programma, eens het in de wachtrij geplaatst is (dit is echter geen probleem bij toepassingen waarin alle gegevens en verwerkingsbeslissingen van tevoren vaststaan).

Real-time interactieve verwerking:
Bij veel gegevensverwerkende toepassingen moeten gegevens wanneer ze ontstaan onmiddellijk verwerkt worden. Dergelijke toepassingen vergen interactie met de gebruiken waarop het programma onmiddellijk moet inspelen.

- interactieve verwerking: tijdens de uitvoering van het programma wordt een dialoog met de gebruiker onderhouden
- real-time verwerking: de handelingen van de computer worden afgestemd op de handelingen van de gebruiker



Uitleg: Real-time verwerking is bij interactieve verwerking geen probleem als er maar één gebruiker/programma is. De uitdaging van computerbesturing is om interactieve verwerking 'in real-time' aan te bieden aan verschillende gebruikers tegelijk.

Multiprogramming:

Timesharing is efficiënt wanneer de uit te voeren programma's veel interactie met de gebruikers vergen omdat I/O-activiteiten toch relatief veel tijd in beslag nemen vergeleken met het door de CPU uitvoeren van programma-instructies.

- timesharing: besturingssysteem laat computer verschillende toepassingen 'gelijktijdig' verwerken door de jobs afwisselend na elkaar even uit te voeren (meestal multi-usersystemen)
- multi-tasking: timesharing bij een systeem met één gebruiker die verschillende taken wil uitvoeren

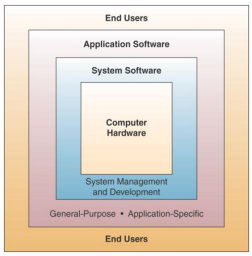
Multiprocessing:

Bij multiprocessing kunnen ook de verschillende instructies van eenzelfde programma door meerdere processoren uitgevoerd worden om zo de uitvoering van het programma te versnellen. Het opdelen van taken in subtaken in overeenstemming met het beschikbaar aantal processoren noemt men scaling. Met load balancing wordt ervoor gezorgd dat alle processoren efficiënt gebruikt worden.

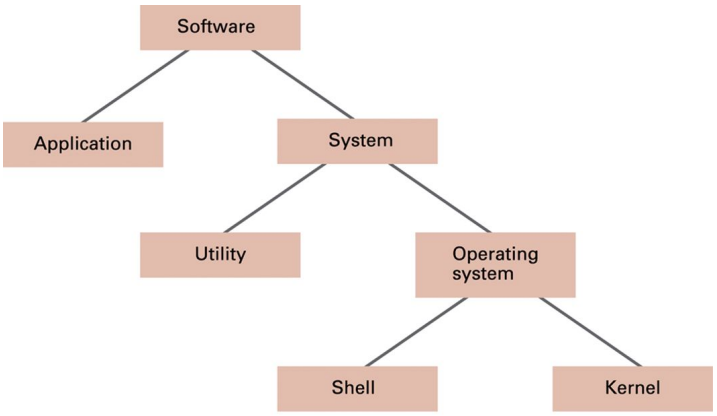
- multiprocessor machines: verschillende taken aan verschillende processoren toewijzen zodat ze echt gelijktijdig uitgevoerd kunnen worden (coördinatie van de verschillende processoren nodig)
    - load balancing
    - scaling

3. Typen van software

- toepassingssoftware: programma's voor het uitvoeren van taken die samenhangen met het gebruik van de computer (vb. gegevensverwerking)
- systeemsoftware: het uitbaten van het computersysteem ter ondersteuning van de gegevensverwerkende toepassingen



Uitleg: De systeemsoftware levert de infrastructuur die de toepassingssoftware en de eindgebruikers nodig hebben om met de computer te kunnen werken.



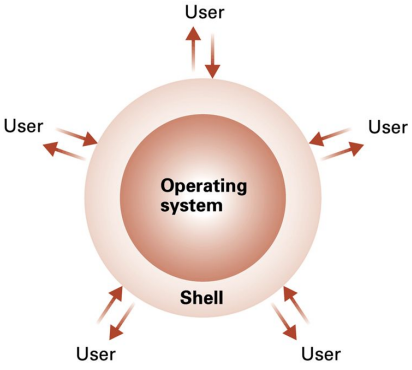
Uitleg: Utility software bestaat uit programma's voor het uitvoeren van activiteiten die van belang zijn voor een goede en efficiënte werking van de computer, maar die niet in het besturingssysteem zelf zitten. Utility software vergroot de mogelijkheden van het besturingssysteem en laat een modulaire opbouw van de systeemsoftware toe. Voorbeelden van utility software zijn programma's voor het comprimeren en decomprimeren van gegevens, voor het defragmenteren van de harde schijf en voor het nemen van backups.

4. Het besturingssysteem: de Shell

Afbeeldingen die een programma in een venster aan een gebruiker wil tonen, worden door de windowmanager in het venster geplaatst. Omgekeerd, wanneer de gebruiker met de computermuis klikt in een venster, berekent de windowmanager de locatie van de muisaanwijzer (Engels: cursor) en maakt deze klik en locatie kenbaar aan het programma.

→ Deel van het besturingssysteem dat communiceert met de gebruiker.

- communicatie met de gebruiker via een grafische gebruikersinterface (GUI)
- belangrijk onderdeel: de windowmanager
  - programma dat blokken beeldschermruimte toewijst aan programma's
  - bij interactieve verwerking van een programma via vensters, moet het programma voor deze communicatie de windowmanager inschakelen



Uitleg: Besturingssystemen bezitten vaak meerdere shells waaruit de gebruiker kan kiezen, bijvoorbeeld een windowgebaseerde GUI of een commandogestuurde shell.

5. Het besturingssysteem: de Kernel

→ Programma's die de basisfuncties van het besturingssysteem uitvoeren.

De filemanager:

Bij een PC bestaat het massageheugen in de eerste plaats uit de harde schijf van de computer.

- coördineert het gebruik van het massageheugen
  → beheert de bestanden opgeslagen in massageheugen (naam en type van het bestand, locatie op het geheugenmedium en toegangsrechten)
- beheert ook het nog vrije deel van het massageheugen

Bestandsbeheer:

Wanneer een programma toegang wil tot een bestand (voor lees/schrijf-bewerkingen), dan moet deze toegang verzocht worden aan de filemanager ('bestand openen'). Als de toegang verleend wordt, dan slaat de filemanager de file descriptor op in een deel van het werkgeheugen dat toegankelijk gemaakt wordt voor het programma. Het programma dient voor lees/schrijf-bewerkingen op het bestand gebruik te maken van verwijzingen naar de file descriptor.

- directory of map: hiërarchische structuur voor het groeperen van bestanden met gemeenschappelijke kenmerken
- pad: positie van een bestand binnen een hiërarchie van mappen
- file descriptor: informatie die een programma nodig heeft om een bestand te vinden en te bewerken

Device drivers:

Device drivers zijn een mooi voorbeeld van de toepassing van abstractie. De machinetaal van de CPU kan hiermee verschillen van de machinetalen van de randapparatuur aangesloten op de CPU. De device drivers zorgen voor de vertaling van instructies van de éne in de andere taal. Bij het vervangen van randapparatuur (vb. nieuwe printer aankoppelen) moet daarom enkel maar een nieuwe device driver geïnstalleerd worden opdat de computer van de nieuwe randapparatuur gebruik kan maken.

- programma's die met controllers communiceren om I/0-bewerkingen op de randapparatuur uit te voeren
- specifiek voor een bepaald apparaat
  - vertalen I/O-instructies naar de instructies waarmee de controller van het randapparaat werkt
  - abstract gereedschap om het besturingssysteem generiek te houden

De memory manager:

Pagina's zijn enkele KB groot. Door van uit te voeren programma's en te verwerken gegevensbestanden enkel die pagina's in het werkgeheugen te laden die op een bepaald moment voor de uitvoering van het programma nodig zijn, kunnen meer en grotere programma's uitgevoerd worden dan dat er in het werkgeheugen passen. Op die manier ontstaat een ***virtueel geheugen*** *dat groter is dan het eigenlijke werkgeheugen, wat van essentieel belang is voor multi-user en multi-tasking computersystemen.*

→ Coördineert het gebruik van het werkgeheugen en beheert de geheugencapaciteit.

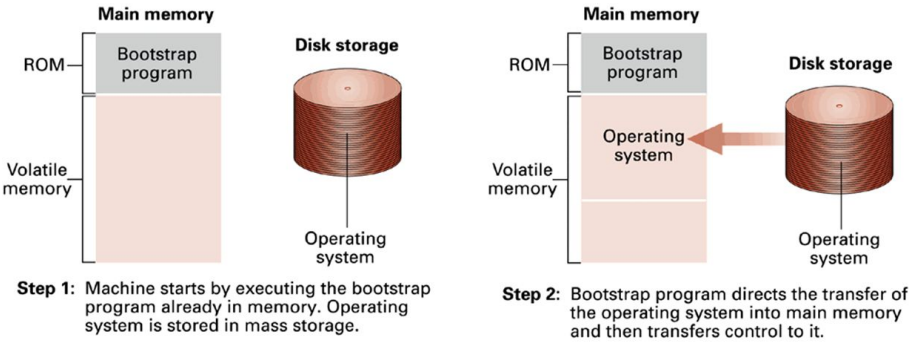
- uit te voeren programma's en te verwerken gegevensbestanden opdelen in pagina's
- bijhouden van welke pagina's waar in het werkgeheugen opgeslagen zijn
- uitwisselen van pagina's tussen werkgeheugen en massageheugen

Booten:

Het besturingssysteem is nodig voor het op de computer uitvoeren van toepassingsprogramma's, maar hoe kan het besturingssysteem zichzelf helpen uitvoeren? Antwoord: via de boot loader. Dit programma is aanwezig in het ROM deel van het werkgeheugen, d.w.z. dat deel van het werkgeheugen dat niet vluchtig is (dus niet afhankelijk van stroomtoevoer), meestal omdat het bestaat uit flashgeheugen (en dus in principe wel wijzigbaar is).

→ Bootstrapping

- procedure die uitgevoerd wordt wanneer de computer gestart wordt
- kopieert het besturingssysteem van het massageheugen naar het werkgeheugen
- boot loader is aanwezig in het Read-Only Memory (ROM) deel van het werkgeheugen



Uitleg: Wanneer de computer opgestart wordt, verwijst de programmateller van de CPU steeds naar dezelfde plaats in het ROM deel van het werkgeheugen. Op deze plaats (en in de daaropvolgende geheugencellen) staat de boot loader. Dit programma start dus automatisch bij het opstarten van de computer. De taak van de boot loader (bootstrap program in de figuur op de slide) bestaat erin om het besturingssysteem dat opgeslagen is in het massageheugen te kopiëren naar het werkgeheugen en dan de CPU opdracht te geven om naar dat gebied in het werkgeheugen te springen. Op dat moment neemt het besturingssysteem de besturing van de computer over.

Firmware:

Wanneer bij het opstarten fouten optreden, kan de boot loader de gebruiker van de computer hiervan verwittigen via firmware programma's. Het is dus essentieel dat de firmware opgeslagen wordt in het ROM deel van het werkgeheugen.

- programma's voor I/O opgeslagen in het ROM
- kan door de boot loader gebruikt worden voor elementaire I/O-activiteiten voordat het besturingssysteem de controle overneemt
    - weergeven van berichten op het beeldscherm, ontvangen van informatie van het toetsenbord, lezen van gegevens van massageheugen
    - communicatie met de gebruiker bij het booten

Proces:

Eénzelfde programma kan aan meerdere processen gekoppeld zijn, bijvoorbeeld wanneer een gebruiker tegelijkertijd twee documenten aan het bewerken is met een tekstverwerkingsprogramma.

- programma: reeks van instructies voor het uitvoeren van een taak door een computer
- proces: de activiteit van het uitvoeren van een programma
- processtatus: de toestand van een programma in uitvoering

→ procesbeheer: coördineren van de processen, processen kunnen de uitvoering van toepassingssoftware, utilities en de programma's van het besturingssysteem betreffen

De scheduler:

- houdt bij welke processen actief zijn
- introduceert een nieuw proces aan deze verzameling (wanneer de gebruiker verzoekt een programma uit te voeren
- verwijdert een proces dat voltooid is

De procestabel:

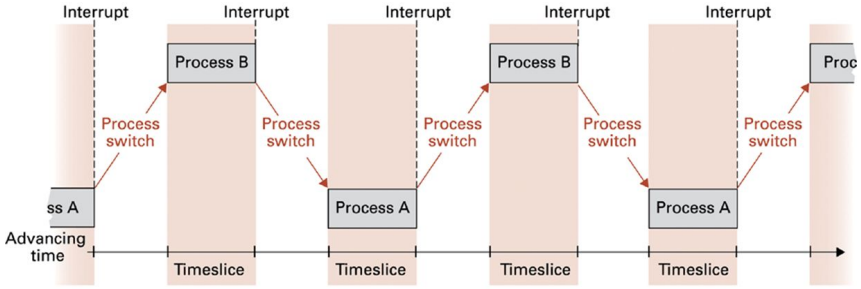
Een proces kan wachten omdat bijvoorbeeld een I/O-bewerking die het gebruik van randapparatuur inhoudt nog niet voltooid is.

- per actief proces een record
    - geheugengebied ter beschikking van het proces (toegewezen door de memory manager)
    - prioriteit van het proces
    - toestand van het proces (gereed = proces kan verder worden uitgevoerd, wacht = externe gebeurtenis nodig vooraleer proces verder uitgevoerd kan worden)
- ruimte in de procestabel wordt beheerd door de scheduler

De dispatcher:

→ De tijdsduur van een tijdslot wordt gemeten in milliseconden, soms slechts microseconden.

- beheert tijdslots en wijst deze toe aan processen
- tijdslot = tijd gedurende dewelke het proces 'beschikt' over de CPU
- procesomschakeling = verandering van toewijzing van CPU



De interrupt handler:

De interrupt handler is een programma dat een onderdeel is van de dispatcher en beschrijft hoe de dispatcher reageert op een interrupt signaal. De interrupt handler is opgeslagen in een vooraf bepaalde en door de CPU gekende locatie van het werkgeheugen. Een Interrupt wordt gegenereerd door een timer die door de dispatcher bij het begin van het tijdslot gestart wordt. Timesharing is (meestal) efficiënt. Wanneer een proces een I/O activiteit doet (die relatief traag is) zal de scheduler de toestand op 'wacht' zetten waardoor het proces zijn tijdslot niet meer verder benut en ook niet meer meedoet aan de strijd om een tijdslot te bemachtigen. De dispatcher zal een ander, 'gereed' proces uit de procestabel kiezen en hieraan een nieuw tijdslot toewijst. Wanneer de controller van het randapparaat aangeeft dat de I/O activiteit beëindigd is, zal de scheduler de toestand van het proces terug op 'gereed' zetten, waardoor het terug kan meedingen naar tijdslots.

- interrupt:
  - signaal voor het einde van een tijdslot
  - CPU maakt machinecylcus af, slaat processtatus op, voert interrupt handler uit
- interrupt handler:
  - geeft scheduler de mogelijkheid de procestabel bij te werken
  - selecteert 'gereed' proces met hoogste prioriteit
  - start timer
  - laat CPU de processtatus herstellen en het geselecteerde proces verder afwerken

Vlaggen:

De printer is een voorbeeld van een systeembron die maar door één proces tegelijk gebruikt kan worden; als twee of meer processen tegelijkertijd de printer zouden gebruiken zouden de afgedrukte gegevens van de verschillende processen door elkaar lopen.

- bit in het werkgeheugen voor systeembronnen die maar door één proces tegelijk gebruikt kunnen worden
  - waarde 0: clear (vrij)
  - waarde 1: set (bezet)
- proces wil toegang tot zo'n bron
  - test-instructie: vlag clear?
  - set-instructie: als vlag clear dan wijs bron toe aan het proces en zet daarna vlag op set

Probleem van concurrentie

Door de interrupt en procesomschakeling tussen het uitvoeren van de test-instructie en de set-instructie kan het zijn dat meerdere processen toch toegang krijgen tot dezelfde systeembron.

→ veronderstel time-sharing met processen X en Y

- de vlag van de bron is *clear*
- X test of deze vlag *clear* is
- interrupt en procesomschakeling van X naar Y
- Y test of dezelfde vlag *clear* is
- Y krijgt bron toegewezen en zet vlag op *set*
- interrupt en procesomschakeling van Y naar X
- X krijgt bron toegewezen en zet vlag op *set*

Semaforen:

Een alternatief voor de interrupt-disable en interrupt-enable instructies is het opnemen van een test-en-set instructie in de machinetaal.

→ vlag waarbij geen interrupt mogelijk is tussen de test en set-instructie

- kritiek gebied: reeks instructies die maar door één proces tegelijk mogen uitgevoerd worden
- semafoor voldoet aan de eis van wederzijdse uitsluiting
- machinetaal bevat interrupt-disable en interrupt-enable instructies

Semaforen oplossing voor concurrentie

→ veronderstel time-sharing met processen X en Y

- de vlag van de bron is clear
- X voert interrupt-disable uit
- X test of deze vlag clear is
- X krijgt bron toegewezen en zet vlag op set
- X voert interrupt-enable uit
- interrupt en procesomschakeling van X naar Y
    - als Y nu test of dezelfde vlag clear is, dan krijgt Y de bron niet toegewezen
    - de toestand van Y wordt op 'wacht' gezet totdat X de bron terug vrijgeeft

Deadlock:

*Deadlock kan ook ontstaan bij **forking**, d.w.z. twee processen willen elk een subproces starten om hun taak te voltooien, maar de scheduler heeft geen ruimte meer vrij in de procestabel.*

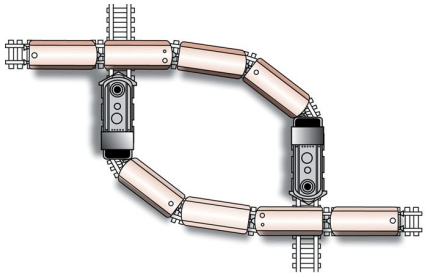
→ een proces wacht op het vrijkomen van een bron die toegewezen is aan een ander proces dat wacht op het vrijkomen van een bron toegewezen aan het eerste proces

⇒ deadlock ontstaat als tegelijk is voldaan aan:

- de gewenste bronnen zijn niet-deelbaar
- meerdere bronnen zijn nodig om een activiteit uit te voeren, maar moeten niet in één keer gevraagd en toegewezen worden
- toegewezen bronnen kunnen niet voortijdig afgepakt worden

Vermijden & oplossen van deadlocks:

- deadlockdetectie en herstel: pak toegewezen bron af (vb. van proces met laagste prioriteit)
- deadlockpreventie: eis dat processen alle bronnen nodig voor het uitvoeren van een activiteit tegelijk aangevraagd en toegewezen worden



Spooling:
→ deadlockpreventie mechanisme

- maak niet-deelbare bronnen deelbaar
- bij output/schrijven
    - wijs randapparaat toe aan één proces
    - schrijf uit te voeren gegevens van andere processen weg naar massageheugen
    - eens randapparaat vrijkomt, lees uit te voeren gegevens van massageheugen en stuur naar randapparaat

Hoofdstuk 4

Netwerken en internet

1. Classificaties van soorten netwerken

 classificatie op basis van reikwijdte

- Local Area Network (LAN): netwerk binnen één enkel gebouw of complex van gebouwen
- Metropolitan Area Network (MAN): netwerk binnen afgebakend geografisch gebied zoals een gemeente
- Wide Area Network (WAN): netwerk dat ongelimiteerd kan zijn qua reikwijdte

 classificatie op basis van eigenaarschap

De werking van een open netwerk is gebaseerd op een ontwerp dat zich in het publieke domein bevindt. De standaarden die de communicatie overheen het netwerk regelen zijn vrij beschikbaar en iedereen kan er gratis van gebruik maken om een dergelijk computernetwerk op te zetten. Voor het opzetten van een gesloten netwerk moet een licentieovereenkomst (veelal betalend) afgesloten worden met de ontwikkelaar van de standaarden.

- Open netwerk (bv. Internet)
- Gesloten (of proprietary) netwerk

 classificatie op basis van netwerktopologie

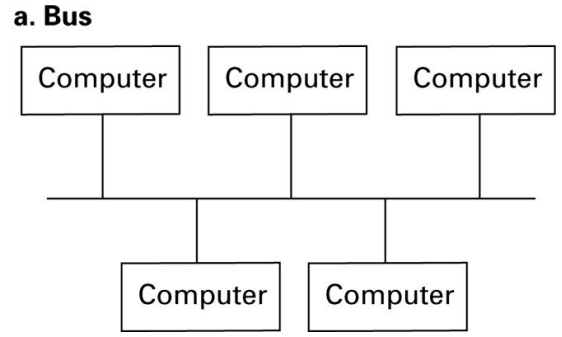
De netwerktopologie verwijst naar de vorm van het network in termen van het patroon waarmee computers in het netwerk verbonden worden.

- Buspatroon (bv. Ethernet netwerken)
- Sterpatroon (bv. Draadloze netwerken met centraal toegangspunt)

<u>Buspatroon:</u>

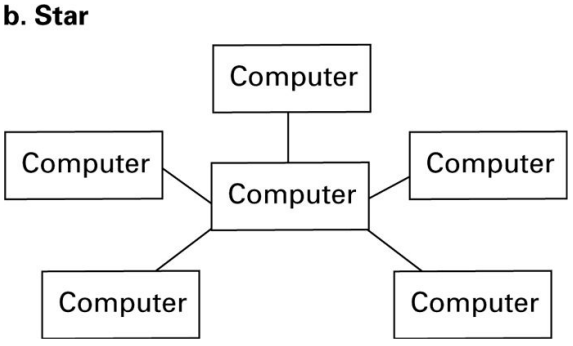
Alle computers zijn verbonden via een gemeenschappelijke communicatielijn die bus wordt genoemd. In een busnetwerk communiceren de aangesloten computers rechtstreeks met elkaar via de gemeenschappelijke bus. Deze bus kan op zich zeer 'kort' zijn, vb. vaak worden de computers met kabels aan elkaar verbonden op een centrale locatie via een ***hub****; dit is een apparaat waarop dan alle kabels aangesloten worden. Hoewel deze topologie sterk lijkt op deze van het sternetwerk (zie volgende side), zal een hub elk signaal dat het ontvangt (van één van de aangesloten computers) doorsturen, eventueel na versterking, naar alle andere aangesloten computers.*

Ethernet is een populaire verzameling van standaarden die de communicatie over een netwerk met bustopologie regelen. Ethernetnetwerken behoren tot de meest populaire computernetwerken die vandaag in gebruik zijn. Oorspronkelijk werden de computers in een Ethernetnetwerk met coaxkabels verbonden. Nu zijn ook andere technologieën in gebruik, vb. UTP kabels (unshielded twisted pair kabels) en glasvezelkabels.



Ster netwerkpologie:

*In een sternetwerk dient één computer als een centraal punt waarop alle andere computers zijn aangesloten. Alle communicatie verloopt via deze centrale computer. Een populaire toepassing van de stertopologie zijn de draadloze netwerken waar het centraal punt een **access point** genoemd wordt (vb. de Telenet hotspots).*



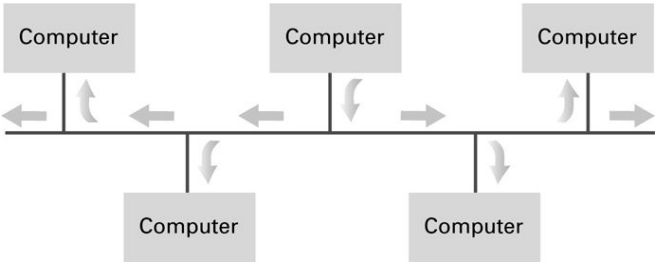
2. Netwerkprotocollen

- Een verzameling van regels die de werking van het netwerk reguleren (bv. Protocollen om het verzamelen van berichten over een netwerk te coördineren)
- Vaak gestandaardiseerd zodat fabrikanten/leveranciers producten kunnen maken die compatibel zijn met andere producten

CSMA/CD:

 Carrier Sense, Multiple Access with Collision Detection (CSMA/CD): protocol gebruikt bij Ethernet (bus)

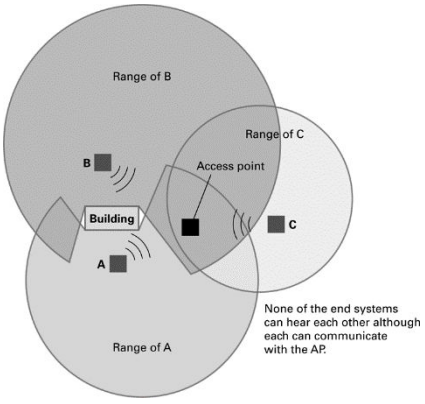
- Elke computer ontvangt elke verstuurde boodschap, maar verwerkt enkel de boodschap die aan hem geadresseerd is
- Boodschappen worden verstuurd wanneer de bus niet bezet is (‘stil’ is)
- Wanneer twee boodschappen tegelijk verstuurd worden (= ‘botsing’), pauzeren beide verzendende computers eventjes (gedurende een toevallige gekozen tijdsinterval) en proberen dan opnieuw



 Carrier Sense, Multiple Access with Collision Avoidance (CSMA/CA): protocol gebruikt bij sternetwerk (als WiFi)

- Wanneer een computer een boodschap wil versturen en het communicatiekanaal is ‘stil’, dan wordt eerst even gewacht
- Wanneer bij dit wachten het kanaal ‘stil’ blijft, mag daarna de boodschap verstuurd worden
- Wanneer bij dit wachten het kanaal ‘busy’ wordt, dan wacht de computer gedurende een toevallig gekozen tijdsinterval opnieuw, vooraleer nog eens te proberen; het versturen mag dan in ieder geval en andere computers moeten wachten

Het ‘Hidden Terminal’ probleem:



Uitleg: Het CSMA/CD protocol dat botsingen detecteert, werkt niet bij een stertopologie omdat de computers in het netwerk het verzenden van boodschappen tussen andere computers en het centrale access point niet altijd kunnen detecteren, vb. omdat er een obstakel in de weg staat (= ‘hidden terminal’) of omdat ze buiten het bereik vallen van een deel van het netwerk.

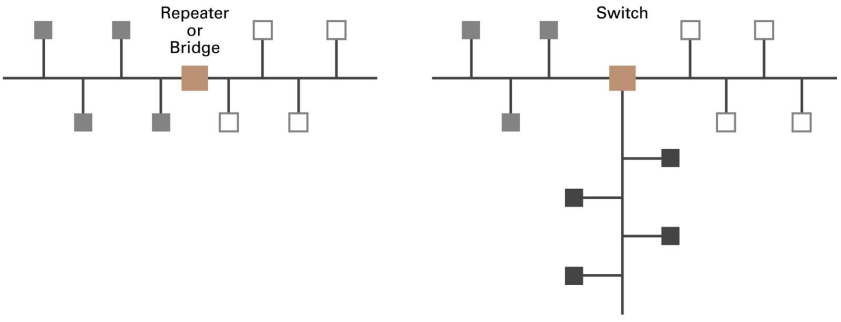
Merk op dat CSMA/CA botsingen probeert te vermijden, door prioriteit te geven aan computers die al langer wachten, het ‘hidden terminal’ probleem niet oplost omdat computers niet altijd kunnen nagaan of het communicatiekanaal stil of bezet is. Daarom wordt dit protocol vaak gebruikt in combinatie met het verzenden van aanvragen naar het Access Point om het communicatiekanaal te mogen gebruiken. Het centrale Access Point zal dergelijke aanvragen negeren wanneer het bezig is met het communiceren met computers in het netwerk.

3. Netwerken van netwerken

Voor het connecteren van netwerken om een groter computernetwerk te vormen is bijkomende apparatuur nodig. Bij het gebruik van repeaters, bridges en switches is het nodig dat de geconnecteerde netwerken hetzelfde netwerkprotocol gebruiken.

 netwerken met hetzelfde protocol (bv. Voor bustopologie)

- Repeater: stuurt signalen door tussen twee geconnecteerde bussen, eventueel na versterking van deze signalen
- Bridge: stuurt alleen signalen door die bedoeld zijn voor een computer van het andere busnetwerk
- Switch: zoals een bridge, maar voor het connecteren van meer dan twee busnetwerken



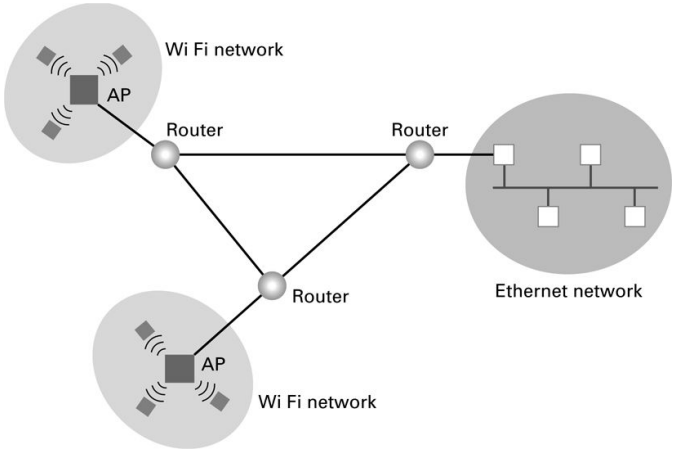
a. A repeater or bridge connecting two buses

b. A switch connecting multiple buses

 internetten = netwerken met verschillende protocollen (bv. Voor connecteren WiFi en Ethernet)

*Indien netwerken die werken onder verschillende netwerkprotocollen verbonden moeten worden, dient een **internet** gebouwd te worden (het Internet is hier slechts één voorbeeld van). In een internet kan elk verbonden netwerk met het eigen protocol blijven werken omdat de apparaten die de netwerken connecteren voor een vertaling zorgen van de boodschappen.*

- Router: computer voor het doorsturen van boodschappen naar het juiste netwerk in de vorm die door het protocol van dit netwerk verwacht wordt, op basis van een voor het internet unieke addressering
- Gateway: apparaat dat de functies van WiFi Access Point en router combineert



4. Communicatiemodellen

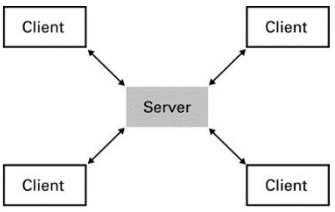
Met een communicatiemodel beschrijven we de manier waarop processen die op verschillende computers in het netwerk uitgevoerd worden met elkaar communiceren om hun activiteiten op elkaar af te stemmen.

Client/Server:

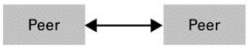
- Proces uitgevoerd op een computer (de cliënt) verzoekt een proces uitgevoerd op een andere computer (de server) om een dienst te leveren (bv. Netwerkprinter)
- Server bedient meerdere clients en moet continu beschikbaar zijn

Peer-to-peer (P2P):

- Processen uitgevoerd op verschillende computers die aan mekaar diensten vragen en leveren
- Processen hoeven niet continu uitgevoerd te worden omdat andere peers in het netwerk hun rol kunnen overnemen



a. Server must be prepared to serve multiple clients at any time.



b. Peers communicate as equals on a one-to-one basis.

5. Gedistribueerde systemen

Met gedistribueerde systemen kan de kracht (vb. rekenkracht, opslagcapaciteit) van meerdere computers tegelijk gebruikt worden om veeleisende gegevensverwerkende taken uit te voeren. Vaak is de kostprijs van een dergelijke verzameling van computers samen met het hogesnelheidsnetwerk dat hen verbindt een stuk lager dan de kostprijs van een supercomputer, en dit met een hogere betrouwbaarheid en onderhoudbaarheid.

Bij cluster computing gaat het om clusters van computers die speciaal bedoeld zijn voor het opzetten van een gedistribueerd systeem.

Bij grid computing gaat het om 'gewone' computers die ook voor andere taken gebruikt worden, maar wanneer ze niet benut worden, ingezet kunnen worden om samen te werken aan grote taken.

Bij cloud computing gaat het om grote hoeveelheden van gedeelde computers die aan clients kunnen toegewezen worden waar en wanneer nodig voor gegevensopslag of gegevensverwerking, zonder dat de clients besef hebben van welke en hoeveel computers hiervoor gebruikt worden en waar deze gesitueerd zijn.

 systemen die bestaan uit software-eenheden die als processen op verschillende genetwerkte computers uitgevoerd worden

Types:

- Cluster computing
- Grid computing
- Cloud computing

Hoofdstuk 5

Algoritmen

1. Wat is een algoritme?

*Gestructureerd betekent dat tijdens de uitvoering van het algoritme ten allen tijde duidelijk is wat de volgende uit te voeren instructie is. Bij **parallelle algoritmen** is het algoritme zodanig gestructureerd dat reeksen van instructies onafhankelijk van elkaar uitgevoerd kunnen worden, vb. via multiprocessing. Dergelijke algoritmen beschrijven ook de synchronisatie van parallelle stromen van instructies, m.a.w. wanneer onafhankelijke paden in de uitvoering terug samen komen (vb. om de resultaten te vergelijken of te integreren). De precisie van een algoritme garandeert dat de uitvoering van de instructies éénduidig is, dus niet voor interpretatie vatbaar. Verschillende personen of machines zullen het algoritme op eenzelfde manier uitvoeren, zodat bij dezelfde invoer dezelfde uitvoer bekomen wordt. Een algoritme is effectief als het uitvoerbaar is, dus in een eindig aantal stappen tot een oplossing leidt. Een algoritme moet ook expliciet beschrijven wanneer de uitvoering ervan stopt.*

 Een algoritme is een geordende reeks ondubbelzinnige, uitvoerbare stappen die een eindig proces beschrijven (computer is niet intelligent, het kan gewoon zeer snel rekenen)

Kenmerken:

- Gestructureerd
- Precies
- Effectief

Niveaus van abstractie in algoritme

meest abstract	Algemene oplossing voor een bepaald type van probleem
↕	Algoritmische beschrijving van deze oplossing
↕	Representatie van deze beschrijving in een vorm geschikt voor communicatie naar het doelpubliek (vb. programma)
minst abstract	Activiteit van het uitvoeren van een algoritme (vb. proces)

Uitleg: De algoritmische beschrijving van de algemene oplossing voor een bepaald type van probleem bestaat uit een geordende, ondubbelzinnig, uitvoerbare reeks instructies die een eindig proces beschrijven. (def. algoritme)

2. Representatie van algoritmen

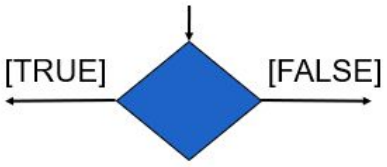
***De primitieven van de taal zijn de bouwstenen** waarmee representaties van algoritmen worden samengesteld. Als de taal informeel is, bijvoorbeeld natuurlijke taal, dan zijn er meerdere interpretaties mogelijk wanneer een primitief gebruikt wordt omdat de betekenis die de 'schrijver' voor ogen had niet automatisch duidelijk is voor de 'lezer' (of 'uitvoerder' van het algoritme). Een ander probleem dat kan optreden is dat instructies in verschillende mate van detail beschreven worden waardoor er voor sommige gebruikers te weinig informatie gegeven wordt om een instructie uit te voeren, terwijl voor andere gebruikers nu net overbodige informatie gegeven wordt. De taal die gebruikt wordt voor het representeren van een algoritme moet formeel gedefinieerd zijn en afgestemd zijn op het soort van gebruiker dat men voor ogen heeft. Soms is het nodig om eenzelfde algoritme meerdere keren te representeren (in meerdere talen dus) wanneer er grote verschillen zijn tussen de gebruikers.*

- Instructies beschrijven in termen van een taal
    - Vocabularium: primitieven (woorden) van de taal (**primitief naar betekenis is een bijectie**)
    - Grammatica: regels voor het combineren van primitieven
- Formeel gedefinieerde talen
    - Elk primitief heeft één ondubbelzinnige unieke betekenis (**geen synoniemen, homoniemen** → anders geen bijectie meer!)
    - Uniforme detaillering van de algoritmen → anders zullen verschillende algoritmen, verschillende graden van detaillering gebruiken

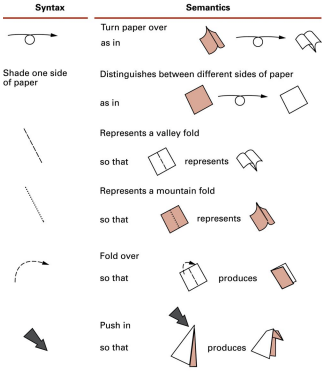
Primitieven:

 bouwstenen van de taal

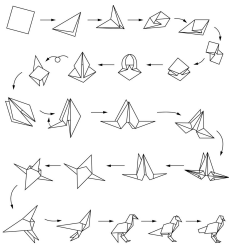
- Syntax: symbolische representatie (zie afb 2)
- Semantiek: betekenis (zie afb 2)
- Booleaanse conditie: afhankelijk van het voldaan zijn van de voorwaarde wordt één van twee mogelijke reeksen instructies uitgevoerd (zie afb 1).



Uitleg: De voorbeeld primitief kan gebruikt worden bij de grafische representatie van een algoritme (een stroomschema of *flow chart*) en is het besturingsconcept van een selectiestructuur.

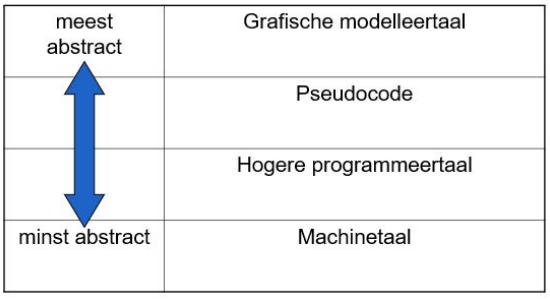


Uitleg: Getoond wordt een origami algoritme om een vogel te vouwen. De primitieven in termen waarvan de instructies beschreven zijn vind je op de volgende slide.



→ voorbeeld van een algoritme (stappen die uitgevoerd moeten worden om een bepaald resultaat te bekomen)

Niveaus van abstractie bij talen voor het voorstellen van algoritmen



Uitleg: Elke primitief van een taal op een bepaald niveau van abstractie kan vertaald worden naar één primitief of naar een combinatie van primitieven van een taal op een onderliggend niveau van abstractie. Algoritmen gerepresenteerd in een machinetaal kunnen direct uitgevoerd worden door de computers waarvoor die taal ontwikkeld werd. Voor het ontdekken, communiceren en presenteren van algoritmen zijn meer abstracte talen dan machinetaal geschikter. Hogere programmeertalen komen in hoofdstuk 6 aan bod. Hier, in hoofdstuk 5, zullen we **pseudocode**, d.i. een abstracte en wat minder formeel gedefinieerde programmeertaal gebruiken.

Pseudocode:

*De pseudocode die we hier zullen gebruiken is gebaseerd op de programmeertaal **Python**.*

- Representeren van algoritmen
  - Tijdens het ontdekken (of ontwikkelen?) van het algoritme
  - Voor het op een compacte manier communiceren van het algoritme
  - Zonder ons te committen tot één of andere programmeertaal
- Pseudocode primitieven
  - Syntax: beknopte, consistente notatie
  - Semantiek: terugkerende semantische structuren

 pseudocode primitieven:

- Toekenning: naam = expressie
  betekenis: de waarde die het resultaat is van de expressie wordt aan de naam gekoppeld
- Keuze:
  betekenis: als de voorwaarde voldaan is, doe dan activiteit_1; als de voorwaarde niet voldaan is, doe dan activiteit_2, vb.:
  if (voorwaarde)**:**
  activiteit_1
  else:
  activiteit_2
- Herhaling (terwijl-lusstructuur)
  betekenis: als de voorwaarde voldaan is doe dan de activiteit en test vervolgens opnieuw de voorwaarde; als de voorwaarde niet voldaan is, sla dan de activiteit over
  while(voorwaarde):
  activiteit

Voorbeelden:

```
RemainingFunds = CheckingBalance + SavingsBalance

if(sales have decreased):
  lower the price by 5%

if(year is leap year):
  daily total = total / 366
else:
  daily total = total / 365

while(tickets remain to be sold):
  sell a ticket
```

```
if(not raining):
    if(temperature == hot):
        go swimming
    else:
        play golf
else:
    go fishing
    drink lots of beer
```

- Functie:
  betekenis: de functie naam is een abstract gereedschap dat verwijst naar een stuk pseudocode dat een bepaalde taak uitvoert; desgewenst wordt deze taak uitgevoerd op de waarde(n) waarnaar verwezen wordt via parameter(s).
  def naam():
  activiteit

Voorbeelden:

```
def Greetings():
    Count = 3
    while (Count > 0):
        print('Hello')
        Count = Count - 1
```

Uitleg: Bij het gebruik van de functie (ook wel de **aanroep** van de functie genoemd) moet enkel maar naar de naam ervan verwezen worden, eventueel dienen ook de waarde(n) waarop de functie wordt toegepast als parameter(s) gespecificeerd te worden.

```
Greetings()
```

p 228-234 zelf lezen!!

3. Fasen bij het algoritmisch oplossen van problemen

Dit zijn geen te volgen stappen bij het oplossen van problemen door middel van algoritmen en programma's, maar fasen waarin men zich bij dit proces kan bevinden. In het ideale geval kan men deze fasen achtereenvolgens doorlopen; in de realiteit is dit zelden het geval. Er bestaat geen algoritme voor het oplossen van problemen! → dit is geen stappen plan

1. Begrijp het probleem
2. Bedenk hoe een algoritmische procedure het probleem kan oplossen
3. Formuleer het algoritme en representeer het
4. Controleer op nauwkeurigheid en ga na of het ook andere problemen kan oplossen

Probleemoplossingsstrategieën:

Vooral stapsgewijze verfijning (ook wel eens de 'verdeel-en-heers' strategie genoemd) is populair bij het ontwikkelen van programma's omdat er gebruik wordt gemaakt van abstractie en omdat het op natuurlijke wijze leidt tot modulair gestructureerde programma's.

- Werk van achter naar voren
  - Vertrek van alle mogelijke oplossingen en ga na welke van deze op basis van de invoer mogelijk is
- Vertrek van de oplossing van een vergelijkbaar, makkelijker probleem
  - Los eerst stukken van het probleem op
  - Bottom-up werkwijze

- Stapsgewijze verfijning (divide en concur)
    - Abstraheer het probleem in een aantal stappen tot een geheel van kleinere, eenvoudigere problemen die gemakkelijker op te lossen zijn
    - Top-down werkwijze

4. Iteratieve structuren

De kracht van computers wordt duidelijk wanneer programma's met iteratieve structuren uitgevoerd worden. Het herhaaldelijk uitvoeren van reeksen instructies (tot vele duizenden keren toe) zal in een mum van tijd gebeuren, daar waar dit manueel (mentaal?) geheel onmogelijk is.

 Lus

- Vaak voorkomende structuur in algoritmen/programma's
- Een reeks instructies die steeds opnieuw herhaald wordt
- Het herhalen van de reeks instructies stopt wanneer aan de eindigingsvoorwaarde voldaan is

Voorbeeld: Sequential search

 Er wordt bij het gebruik van deze functie verondersteld dat de lijst met waarden oplopend gesorteerd is (vb. van klein naar groot, alfabetisch).

```
def Search(List, TargetValue):
    if(List is empty):
        Declare search a failure
    else:
        Select the first entry in List to be TestEntry
        while(TargetValue > TestEntry and entries remain):
            Select the next entry in List as TestEntry
        if(TargetValue == TestEntry):
            Declare search a success
        else:
            Declare search a failure
```

Lusbesturing:

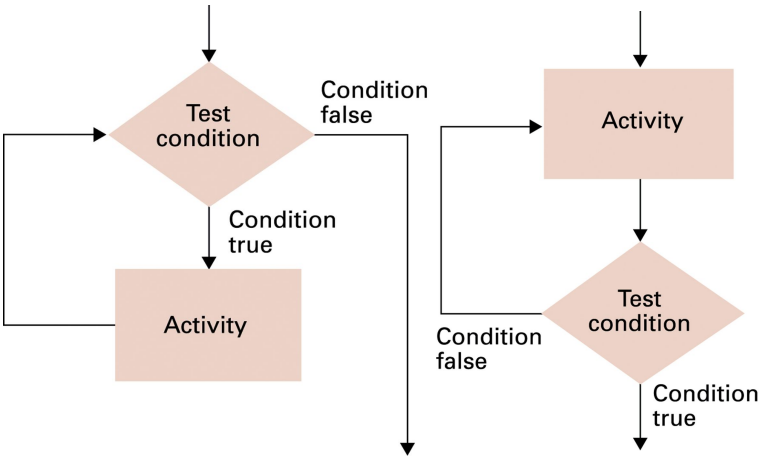
termination condition = eindigingsvoorwaarde

Initialize:	Establish an initial state that will be modified toward the termination condition
Test:	Compare the current state to the termination condition and terminate the repetition if equal
Modify:	Change the state in such a way that it moves toward the termination condition

 bij de terwijllusstructuur: een lus waarbij de eindigingsvoorwaarde nooit voldaan zal zijn is een oneindige lus

- Voorwaarde = complement van eindigingsvoorwaarde
    - Voorbeeld bij sequentieel zoeken (eindigingsvoorwaarde: TargetValue ≤ TestValue OF er zijn geen te testen waarden meer in de lijst)
- Initialisatie- en modificatiestappen moeten ervoor zorgen dat eindigingsvoorwaarde ooit voldaan zal zijn

Alternatieve iteratiestructuren:



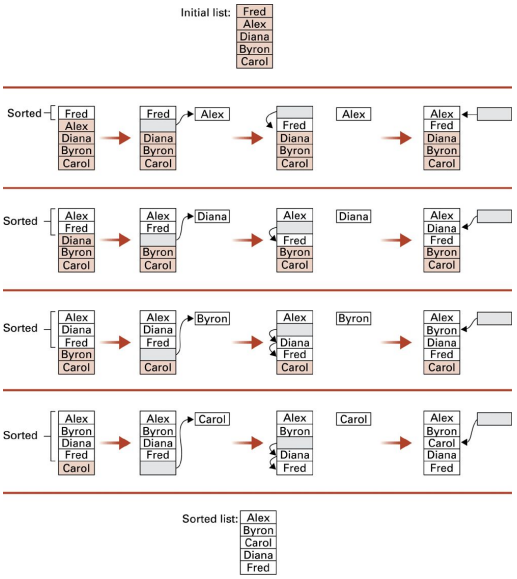
Uitleg: Deze iteratiestructuren noemen we de **terwijl-lusstructuur** (links) en de **herhaal-lusstructuur** (rechts). Het zijn respectievelijk een **pretest-lus** en een **posttest-lus**. Het verschil tussen beide varianten zit erin dat bij de posttest-lus de body van de lus (dit zijn de instructies die herhaald worden) altijd minstens één keer uitgevoerd wordt.

Pseudocode herhaal-lusstructuur:

De voorwaarde die getest wordt is de eindigingsvoorwaarde, dus het complement van de voorwaarde die gebruikt wordt bij een equivalente terwijl-lusstructuur.

Noot: Python heeft geen ingebouwde herhaal-lusstructuur, dus het 'repeat until' primitief dat we toevoegen aan de pseudo-code is ontleend aan andere programmeertalen.

- Herhaal-lusstructuur
  betekenis: de activiteit wordt gedaan en vervolgens wordt de voorwaarde getest; als de voorwaarde niet voldaan is doe dan opnieuw de activiteit en test vervolgens opnieuw de voorwaarde; als de voorwaarde voldaan is, ga dan verder met de uitvoering van de volgende instructie
  repeat:
  activiteit
  until(voorwaarde)



Uitleg: Insertion sort is een sorteeralgoritme dat beetje bij beetje een lijst met waarden (vb. namen) sorteert door het stukje lijst dat gesorteerd is telkens met één waarde groter te maken. Dit gebeurt door de nieuwe waarde telkens op de juiste plaats in het reeds gesorteerde deel in te voegen.

Voorbeeld: Insertion sort

→ Alex is de eerste spil of de pivot (hiermee wordt Fred vergeleken)

De **spil** (Engels: **pivot**) is de waarde die op de juiste plaats ingevoegd moet worden in het reeds gesorteerde deel van de lijst.

```
def Sort(List):
    N = 2
    while(N ≤ length of List):
        Pivot = Nth entry in List
        Remove Nth entry leaving a hole in List
        while(there is an Entry above the hole and Entry > Pivot):
            Move Entry down into the hole leaving
            a hole in the list above the Entry
        Move Pivot into the hole
        N = N + 1
```

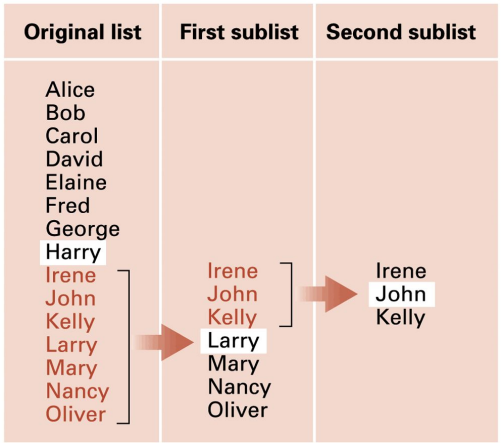
5. Recursieve structuren

Bij een recursieve structuur is elke fase in de herhaling van een reeks instructies een subtaak van de vorige fase. Bij een iteratieve structuur volgt elke fase in de herhaling van een reeks instructies op de vorige fase.

 Recursieve functie

- Alternatief voor iteratieve structuren
- Een reeks instructies die steeds opnieuw herhaald wordt
    - NIET door een reeks instructies opnieuw te herhalen wanneer ze volledig uitgevoerd zijn
    - WEL door een reeks instructies uit te voeren als subtaak van zichzelf
    - Eén of meerdere van de instructies in de reeks houden een aanroep van de volledige reeks instructies in
- Het herhaaldelijk aanroepen van de reeks instructies stopt wanneer aan de eindigingsvoorwaarde voldaan is

Voorbeeld: we zoeken 'John' in de lijst



Uitleg: Binary search is een zoekalgoritme dat doorgaans sneller resultaat oplevert dan sequential search omdat het te doorzoeken deel van de lijst in elke stap gehalveerd wordt. Het zoekalgoritme wordt dus herhaaldelijk toegepast op een kleiner stuk van de lijst, totdat de op te zoeken waarde gevonden wordt of totdat er niets meer te doorzoeken valt. Het herhaaldelijk toepassen van het zoekalgoritme binnen de uitvoering van het zoekalgoritme zelf noemen we recursie.

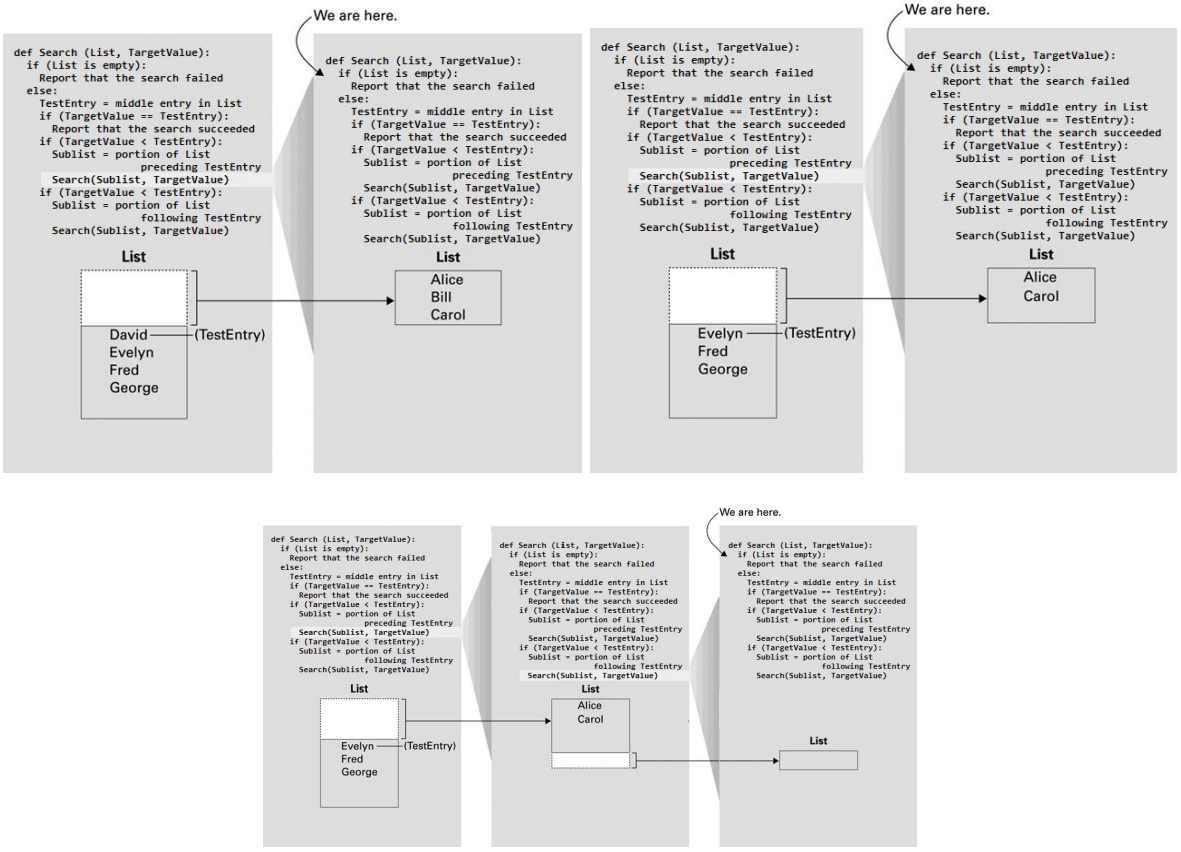
Voorbeeld: Binary search

```
if(List is empty):
    Report that the search failed
else:
    TestEntry = middle entry in the List
    if(TargetValue == TestEntry):
        Report that the search succeeded
    if(TargetValue < TestEntry):
        Search the portion of List preceding TestEntry for
        TargetValue, and report the result of that search
    if(TargetValue > TestEntry):
        Search the portion of List following TestEntry for
        TargetValue, and report the result of that search
```

```
def Search(List, TargetValue):
    if(List is empty):
        Report that the search failed
    else:
        TestEntry = middle entry in the List
        if(TargetValue == TestEntry):
            Report that the search succeeded
        if(TargetValue < TestEntry):
            Sublist = portion of List preceding TestEntry
            Search(Sublist, TargetValue)
        if(TargetValue > TestEntry):
            Sublist = portion of List following TestEntry
            Search(Sublist, TargetValue)
```

Uitleg: Door van het zoekalgoritme een functie te maken hebben we een abstract gereedschap gecreëerd dat in de reeks instructies aangeroepen kan worden. De instructies worden dus als subtaak van zichzelf uitgevoerd telkens wanneer de functie aangeroepen wordt.

Voorbeeld: we zoeken ‘Bill’ (links) en ‘Diane’ (rechts) in de lijst



Recursieve besturing:

- Keuzestructuur met test of eindigingsvoorwaarde voldaan is
    - Voldaan als probleem triviaal geworden is
- Als niet voldaan is, aanroep van de recursieve functie
    - Maar nu toegepast op iets eenvoudiger probleem
    - Door het vereenvoudigen van het probleem zal de eindigingsvoorwaarde ooit voldaan zijn en het zich herhaaldelijk aanroepen stoppen

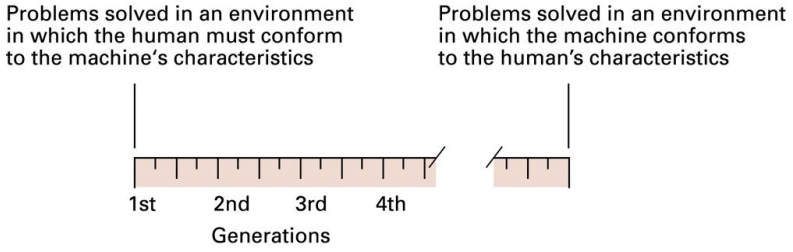
Hoofdstuk 6

Programmeertalen

De pseudo code waarmee algoritmen uitgedrukt worden in hoofdstuk 5 is begrijpelijk voor mensen, maar niet automatisch vertaalbaar naar één of andere machinetaal. De machinetaal van hoofdstuk 2 en bijlage C kan gebruikt worden om algoritmen voor te stellen zodat ze onmiddellijk in het werkgeheugen van de computer (die een architectuur heeft waarvoor de machinetaal ontworpen werd) opgeslagen kunnen worden en vervolgens door de CPU uitgevoerd kunnen worden. Het nadeel van machinetalen is dat de binaire codes voor de instructies van een programma moeilijk hanteerbaar zijn voor menselijke gebruikers. Het is zeer lastig om deze binaire codes foutloos te lezen en te kopiëren, zeker voor lange programma's met een complexe structuur (vb. geneste lussen). Hexadecimale notatie helpt, maar is toch ver verwijderd van de concepten gebruikt bij het algoritmisch denken.
Voor het ontdekken van algoritmen en het ontwikkelen van een eerste voorstelling ervan is pseudocode geschikt. Voor het laten uitvoeren van een programma door één bepaald type van computer is machinetaal geschikt. In dit hoofdstuk 6 bestuderen we een derde representatievorm, programmeertalen, die qua begrijpbaarheid niet veel onderdoet voor pseudocode, terwijl ze toch een automatische vertaling van het programma in een machinetaal mogelijk maakt. Programmeertalen zijn ideaal als voorstellingswijze voor algoritmen eens het algoritmisch denken over de oplossing van het probleem achter de rug is, maar vooraleer een definitieve keuze van op welk type computer(s) het programma uit te voeren gemaakt is.

1. Generaties programmeertalen

*Machinetalen vormen de eerste generatie van programmeertalen. Latere generaties maken in toenemende mate abstractie van specifieke computerarchitecturen en leunen meer en meer aan bij de natuurlijke talen in termen waarvan mensen over problemen en hun oplossingen nadenken. Bij de ontwikkeling van dergelijke 'gebruiksvriendelijke' **hogere programmeertalen** blijft de omzetting (liefst automatisch) van de programma instructies in machine instructies een belangrijke overweging.*



Eerste generatietalen: Machinetalen

Het spreekt voor zich dat het programmeren van algoritmen met bitreeksen zeer foutgevoelig is, wat je bij het inoefenen van hoofdstuk 2 waarschijnlijk aan den lijve ervaren hebt.

- Binaire code
    - Vb. 0101000001010110
    - Verkort (hexadecimaal): 5056

- Ontworpen voor één welbepaalde computerarchitectuur
    - Vb. De registers met adressen 5 en 6 bevatten elk een geheel getal in 2-complementnotatie (8 bits), die we optellen en waarvan we de som plaatsen in het register met adres 0
- Programmeur moet denken zoals de computer uitvoert

Tweede generatietalen: assemblertalen

Het concept 'mnemonisch' wordt ook wel eens beschreven als 'mnemotechnisch'. Mnemonische codes zijn 'geheugensteuntjes' voor binaire codes; zij helpen programmeurs bij het onthouden van de betekenis van de bitpatronen die bij machinetalen gebruikt worden. Programma's in mnemonische code uitgedrukt lezen wat makkelijker dan dezelfde programma's uitgedrukt in hexadecimale code (en in ieder geval binaire code). De mnemonische naam voor een op-code leunt aan bij de natuurlijke taal waarmee we de semantiek van de op-code definiëren.

 Mnemonische code (Vb. **LD R5, prijs** in plaats van 156C of 0001010101101100)

- Mnemonische namen voor op-codes (Vb. **LD** in plaats van op-code 0001 (LOAD instructie)
- CPU-registers benoemen (Vb. **R5** in plaats van register met adres 0101)
- Identifiers (betekenisvolle namen) voor het verwijzen naar de geheugencellen waarin de te verwerken gegevens opgeslagen zijn (Vb. **prijs** in plaats van adres 01101100 om te verwijzen naar de waarde die in de geheugencel met dat adres opgeslagen is)

Voorbeeld:

In deze denkbeeldige assemblertaal staat LD voor LOAD, ST voor STORE, ADDI voor ADD Integers en HLT voor HALT.

Machinetaal (hexadecimaal):	Assemblertaal:
156C	LD R5, prijs
166D	LD R6, verzendkost
5056	ADDI R0, R5 R6
30CE	ST R0, totaal
C000	HLT

Eigenschappen van assemblertalen:

*Het belangrijkste verschil en tegelijk voordeel met de eerste generatietalen is dat de syntax van de tweede generatietalen gebruiksvriendelijker is. Het is duidelijk dat assemblertalen slechts een 'opstapje' betekenen voor de ontwikkeling van hogere programmeertalen, die **machine onafhankelijk** zijn.*

- Eén-op-één overeenkomst tussen machineinstructies en programma instructies
    - Programmeur moet nog steeds denken zoals de computer uitvoert
    - Inherent verbonden aan één welbepaalde computerarchitectuur
- De vertaling naar machineinstructies gebeurt automatisch, door een assembler.

Derde generatietalen: Hogere programmeertalen

*Een **compiler** (= een vertaler) vertaalt het ganse programma naar machinetaal, waarna het uitgevoerd kan worden. Een interpreter (= een tolk) vertaalt programma instructie per programma instructie naar machine instructies en voert ze onmiddellijk uit.*

- Primitieven zijn abstracter dan deze van machine/assemblertalen
    - Bijna vergelijkbaar met abstractieniveau van pseudocode
    - Een programma instructie komt overeen met een reeks van één of (meestal) meerdere machine instructies
    - Programmeur moet niet langer denken zoals de computer uitvoert
- Hogere programmeertalen zijn machine-onafhankelijk
    - Eenzelfde programma kan (na vertaling) uitgevoerd worden op computers met verschillende architecturen
- De vertaling naar machine instructies (in de machinetaal naar keuze) gebeurt automatisch, door een compiler of interpreter (vertaler of tolk)

Voorbeeld:

Assemblertaal:
```
LD R5, prijs
LD R6, verzendkost
ADDI R0, R5 R6
ST R0, totaal
HLT
```

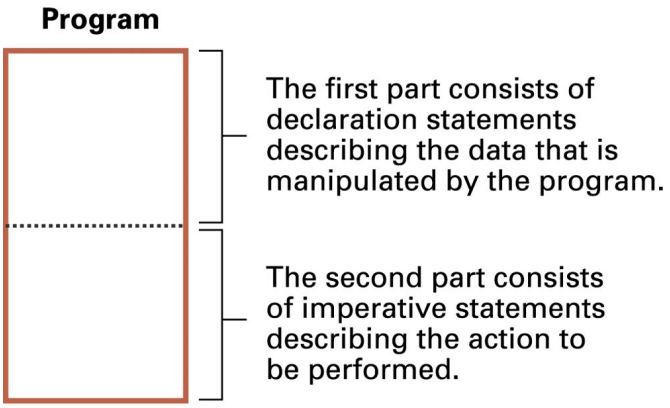
Hogere programmeertaal (Python)
```
totaal = prijs + verzendkost
```

Programmeerparadigma's: (niet kennen! p276-280)

2. Programmeerconcepten (imperatief paradigma)

 Programma's hebben 3 soorten van 'instructies' (statements)

- Declaratieve statements
    - Definiëren terminologie die in het programma gebruikt zal worden (vb. variabelen)
- Imperatieve statements
    - Beschrijven stappen in het algoritme
- Commentaren
    - Enkel om het programma te documenteren
    - Worden niet vertaald naar machine instructies



 Samenstelling van een typisch imperatief programma of imperatieve programma-eenheid

Variabelen en gegevenstypen:

In een declaratieve statement kunnen verschillende variabelen (naam gegeven aan gegevens) van hetzelfde type gedefinieerd worden. Ook is het mogelijk om bij het 'declareren' van variabelen initiële waarden toe te kennen. Vertaalprogramma's zoals compilers en interpreters maken van de kennis opgeslagen in declaratieve statements gebruik om een programma in een hogere programmeertaal te vertalen naar een programma in een machinetaal.

- Variabele
  - Verwijzing naar een locatie in het werkgeheugen door middel van een identifier i.p.v. adres
  - Als waarde opgeslagen in de locatie wijzigt, wijzigt ook de waarde gekoppeld aan de identifier
    ⇒ identifier is 'variabel'

- Gegevenstype
  - Hoe gegeven coderen? + welke bewerkingen toegelaten?
  - Definitie van een variabele met een declaratieve statement waarin naast de identifier ook beschreven wordt welk type van gegeven in de locatie wordt opgeslagen
  - Primitieve gegevenstypen: integer (gehele getallen), real (float) (reële getallen), char (symbolen), Boolean (Booleaanse waarden)

Voorbeelden:

```
float Length, Width;
int Price, Tax, Total;
char Symbol;
```

```
length = 3.5
price = 40
symbol = 'q'
```

Uitleg: De linkse voorbeelden gelden voor talen zoals C, C++, C# en Java. Er bestaan ook imperatieve talen waarin het niet nodig is om de gegevenstypen van variabelen expliciet te definiëren met declaratieve statements, vb. Python. Het gegevenstype van een variabele wordt dan afgeleid uit de waarde die aan de variabele is toegekend.

Gegevensstructuren:

- Gekoppeld aan de definitie van variabelen
- Beschrijft de conceptuele vorm van een verzameling van gegevens
    - Vb. tekst = lange reeks tekens, dus variabelen van type char
- Vergemakkelijkt het werken met een grote groep variabelen die gegevens beschrijven die logisch samenhangen
    - Vb. een lijst of tabel met waarden

Arrays

Homogene Array

Homogene arrays worden net zoals variabelen gedefinieerd met behulp van declaratieve statements.

- Blok van gegevens van hetzelfde type
    - 1 dimensie: lijst
    - 2 dimensies: table
    - >2 dimensies: hogere orde gegevensstructuren
- Bij definitie naam, gegevenstype, aantal dimensies en lengte van de dimensies opgeven
- Naar componenten van homogene arrays verwijzen door middel van indices

Voorbeeld:

In C

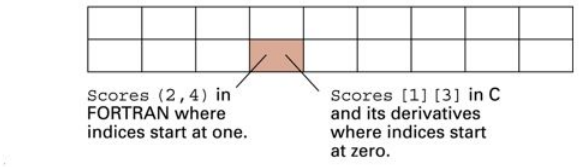
```
int Scores[2][9];
```

In FORTRAN

```
INTEGER Scores(2,9)
```

In Python

```
>>> scores_1 = [11, 14, 12, 9, 14, 17, 19, 10, 8]
>>> scores_2 = [10, 20, 15, 16, 11, 12, 12, 0, 13]
>>> scores = [scores_1, scores_2]
>>> scores[1][3]
16
>>>
```



Uitleg: Scores is een twee-dimensionele homogene array van integers. Afhankelijk van de gebruikte taal starten indices vanaf 0 of 1.

Python kent geen arrays, maar wel het gerelateerde concept van een lijst (confer hoofdstuk 8). Lijsten kunnen genest worden om meerdimensionele homogene arrays te creëren.

Heterogene array

Heterogene arrays worden eerst met een declaratief statement als een nieuw gegevenstype gedefinieerd. Vervolgens kunnen dan variabelen voor dit nieuwe gegevenstype gedefinieerd worden.

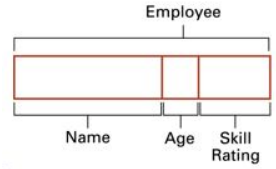
- Blok van gegevens van mogelijks verschillende typen
- Bij definitie naam van de array en van elke component ook de naam en het gegevenstype opgeven
- Naar componenten van heterogene arrays verwijzen door middel van ‘naam van de array **.** naam van de component’ (Vb. **Medewerker.Leeftijd)**

Voorbeeld:

a. The array declaration

```
struct
{ char Name [8];
  int Age;
  float SkillRating;
} Employee;
```

b. The conceptual organization of the array




```
>>> employee = 'J. Claes', 35, 0.9
>>> employee[0]
'J. Claes'
>>>
```

Uitleg: Het voorbeeld links is in C geschreven.
Voorbeeld van gebruik:
Employee werknemer;
...
Print(werknemer.Name, werknemer.Age, werknemer.SkillRating);
...
Het voorbeeld rechts is een heterogene array in Python (daar 'tuple' genoemd) die direct als variabele gebruikt kan worden.

Constanten en literals:

- Literal
    - Letterlijke waarde gebruikt in een programma (Vb. **PrijsInclusief = PrijsExclusief * (1 + 0.21)**)
- Constante
    - Identifier gekoppeld (met declaratieve statement) aan een onveranderlijke literal (Vb. **const float btwLuxeProduct = 0.21**)
    - Bevordert leesbaarheid en aanpasbaarheid (Vb. **PrijsInclusief = PrijsExclusief * (1 + btwLuxeProduct)**)

Toekenningsstatements:

 Assignment statement

*De toekenningsoperator was ook één van de primitieven van de pseudocode beschreven in hoofdstuk 5. Rekenkundige operatoren zoals de + kunnen overladen zijn, zoals bijvoorbeeld in Java (Engels: **overloading**).*

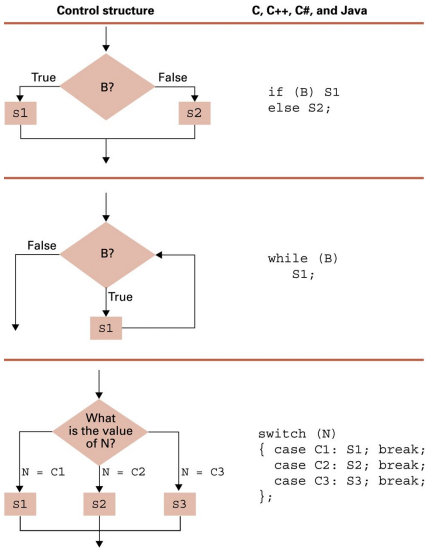
- Basis imperatief statement
- Voor het opslaan van een waarde in de geheugenlocatie die gekoppeld is aan de variabele
- Toe te kennen waarde kan een literal of een expressie zijn!
    - Rekenregels bepalen de volgorde van het toepassen van de operatoren
    - Operatoren kunnen overladen zijn (Vb. **'abra' + 'cadabra'** levert **'abracadabra'** op)

Besturingsopdrachten:

Om gestructureerd programmeren toe te laten moet een programmeertaal minstens één besturingsopdracht van het type 'keuze' en één besturingsopdracht van het type 'herhaling' aanbieden.

- Control statement (Engels)
    - Imperatief statement dat de volgorde van uitvoering van programmaopdrachten bepaalt

- Gestructureerd programmeren
    - Gebruik van een beperkt aantal mogelijke besturingsopdrachten om duidelijke, goed leesbare programma's te ontwikkelen

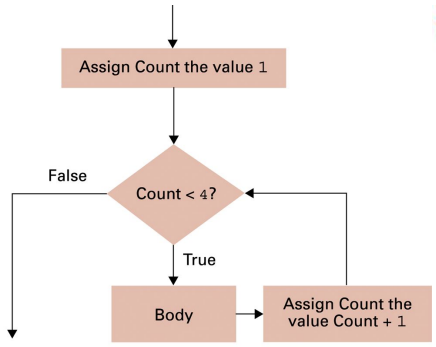



```
if getal % 2 == 0:
    print(str(getal) + ' is even.')
else:
    print(str(getal) + ' is oneven.')
```

```
teller = 1
while teller <= x:
    print(str(teller))
    teller = teller + 1
```

```
if len(list) == 0:
    print('Empty list')
elif len(list) == 1:
    print('Singleton')
elif len(list) == 2:
    print('Tuple')
elif len(list) == 3:
    print('Triple')
elif len(list) == 4:
    print('Quadruple')
else:
    print('Other type of list')
```

Uitleg: De besturingsopdrachten worden schematisch afgebeeld met primitieven uit een taal voor het opstellen van stroomschema's (flow charts). Er wordt ook telkens een Python voorbeeld gegeven. Python heeft geen switch-statement.




```
for (int Count = 1; Count<4; Count++)
    body ;
```

```
for teller in range(1, x + 1):
    print(str(teller))
```

Uitleg: Voorbeeld links voor C++, C# en Java. Rechts een Python voorbeeld dat de getallen 1 t.e.m. x afdrukt.

3. Procedurele eenheden

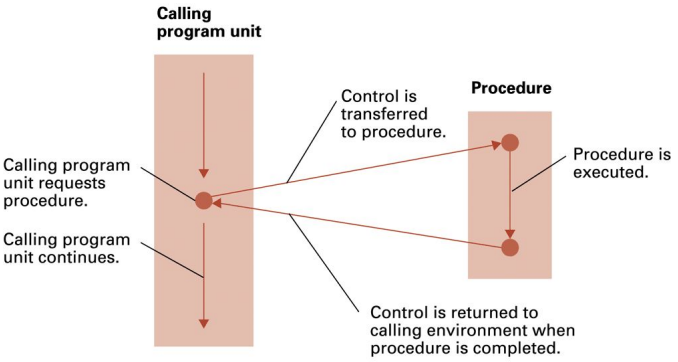
In de pseudocode van Hoofdstuk 5 werden procedurele eenheden gedefinieerd met het functie primitief. In Hoofdstuk 6 behouden we de term functie voor een specifiek type van procedurele eenheid (confer infra) en gebruiken we de meer generieke term procedure voor het benoemen van een procedurele eenheid als verzameling instructies die een bepaalde taak uitvoert.

***Lokale variabelen** zijn enkel maar gedefinieerd voor één of andere procedure; er kan niet naar verwezen worden buiten deze procedure. Variabelen die niet lokaal zijn aan een procedure worden **globale variabelen** genoemd.*

→ Abstract concept voor een verzameling instructies die een bepaalde taak uitvoert

- Programma 'in het klein'
    - Declaratieve statements (vb. lokale variabelen)
    - Imperatieve statements
- Procedure header
    - Declaratieve statement voor het definiëren van een naam voor de procedure
- Procedure aanroep
    - De verzameling instructies waarvoor de procedure staat kan uitgevoerd worden door te verwijzen naar de naam van de procedure.
    - Besturing wordt overgedragen aan deze procedure; na uitvoering van de procedure wordt de besturing weer overgedragen aan de aanroepende programma-eenheid.

Overdragen van besturing bij procedure aanroep:

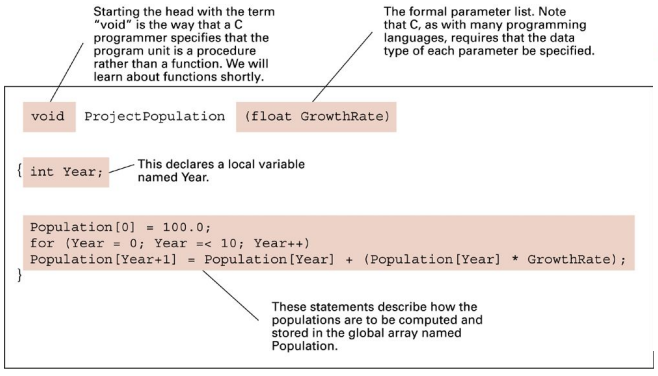


Uitleg: Met het overdragen van besturing wordt bedoeld het veranderen van de reeks programma instructies die uitgevoerd wordt. De overdracht van de besturing wordt in een machinetaal gerealiseerd met een **JUMP** (sprong) opdracht, waardoor het werkgeheugen adres van de volgende uit te voeren instructie in de programmateller geplaatst wordt (confer hoofdstuk 2).

Parameters:

- Formele parameter in procedure header
    - Declaratie van een lokale variabele die een waarde krijgt bij het aanroepen van de procedure (Vb. **def ProjectPopulatie(Groeisnelheid)**)
- Actuele parameter in procedure aanroep
    - Waarde die gebruikt wordt voor een formele parameter bij de procedure aanroep (Vb. **ProjectPopulatie(0.03)**)

Voorbeeld:




```
def projectPopulation(growthRate):
    population = [100.0]
    for year in range(0,10):
        population.append(population[year] + (population[year] * growthRate))
    for year in range(0,11):
        print('Population in year ' + str(year) + ': ' + str(population[year]))

projectPopulation(0.03)
```

```
Population in year 0: 100.0
Population in year 1: 103.0
Population in year 2: 106.09
Population in year 3: 109.2727
Population in year 4: 112.550881
Population in year 5: 115.92740743
Population in year 6: 119.4052296529
Population in year 7: 122.98738542487
Population in year 8: 126.67700813876161
Population in year 9: 130.47731838292447
Population in year 10: 134.3916379344122
>>>
```

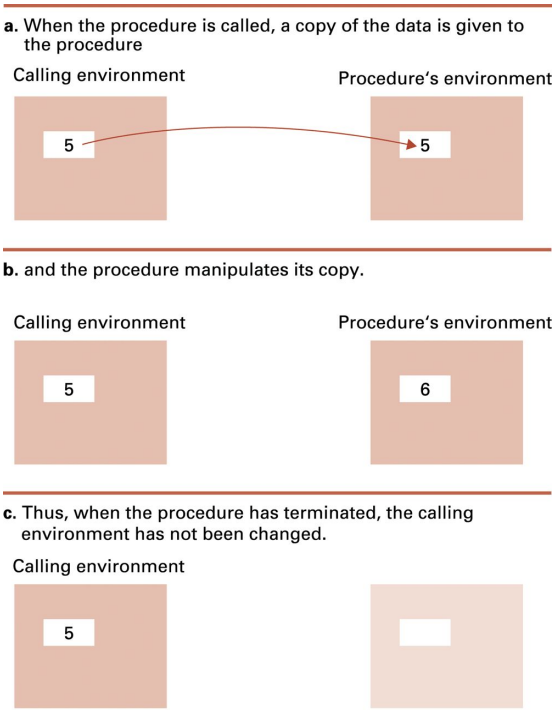
Parameter passing:

- By Value
    - De waarde van de actuele parameter wordt gekopieerd naar de formele parameter → originele parameter wordt niet veranderd
- By Reference
    - De formele parameter verwijst naar dezelfde werkgeheugen locatie als de actuele parameter → originele parameter wordt mee veranderd

De volgende twee voorbeelden veronderstellen een procedure met procedure header def Incrementeer(Formeel): waarin één toekenning statement Formeel = Formeel + 1 zit, en veronderstellen verder ook een procedure aanroep Incrementeer(Actueel), waarbij Actueel een actuele parameter voorstelt van eenzelfde numeriek gegevenstype als de formele parameter Formeel in de procedure Incrementeer.

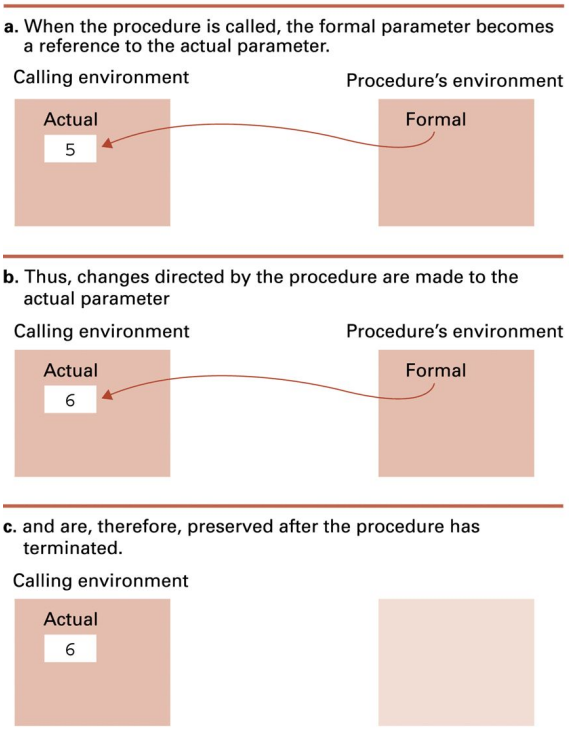
```
def Incrementeer(Formeel):
Formeel = Formeel + 1
...
Actueel = 5
Incrementeer(Actueel)
Print(strt(Actueel))
...
```

Passed by value:



Uitleg: In dit voorbeeld wordt bij de overdracht van Actueel naar Formeel **passed by value** verondersteld.

Passed by reference:



Uitleg: In dit voorbeeld wordt bij de overdracht van Actueel naar Formeel **passed by reference** verondersteld.

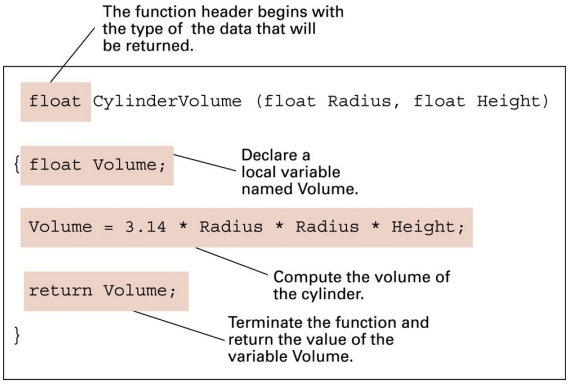
Functies:

- Procedurele eenheid die een waarde retourneert naar de aanroepende programma-eenheid
- Geretourneerde waarde is het resultaat van de bewerking die de functie uitvoert
- In de aanroepende programma-eenheid kan de functie aanroep gebruikt worden als ware het de waarde die het resultaat van de bewerking voorstelt.

Voorbeeld:

```
def Incrementeer(Formeel):
    Formeel = Formeel + 1
    return Formeel
...
Actueel = 5
Actueel = Incrementeer(Actueel)
print(str(Actueel))
...
```

Voorbeeld:




```
def cylinderVolume(radius, heigth):
    volume = 3.14 * radius * radius * heigth
    return volume

print(str(cylinderVolume(2, 3)))
```

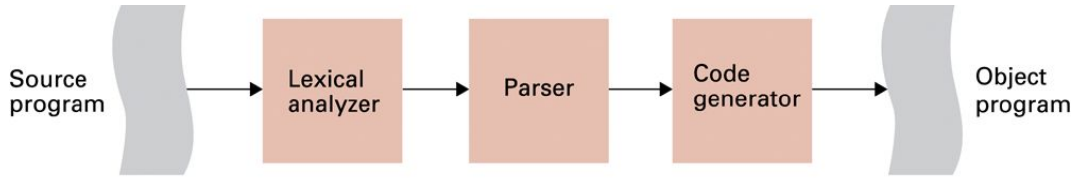
```
37.68
>>>
```

4. Vertaalproces

Vertaalprogramma's kunnen compilers of interpreters zijn (om van hogere programmeertaal naar machinetaal te gaan) of assemblers (om van een assemblertaal naar machinetaal te gaan).

- Vertalen
    - Proces waarmee een programma van de éne taal naar een andere wordt omgezet
- Bronprogramma
    - Programma in de oorspronkelijke vorm (Vb. programma geschreven in een hogere programmeertaal)
- Doelprogramma
    - Programma in de vorm die de vertaling beoogt (Vb. programma geschreven in de machinetaal die hoort bij de computer waarop we het programma willen uitvoeren)

Activiteiten bij het vertaalproces:



Uitleg: De lexicale analyzer, parser en codegenerator zijn drie onderdelen van het vertaalprogramma.

Lexicale analyse:

De lexicale analyzer slaat bij het vormen van tokens alle commentaren over. Deze worden dus niet overhandigd aan de parser.

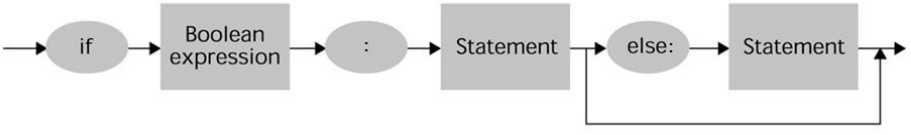
- Voor het interpreteren van reeksen symbolen in het bronprogramma als eenheden
    - Symbool voor symbool bronprogramma inlezen
    - Bepalen welke groepen symbolen één eenheid vormen
    - Eenheden classificeren als getallen, woorden, wiskundige operatoren, etc.
- Token
    - Bitpatroon die een herkende eenheid voorstelt
    - Wordt door lexicale analyzer aan parser overhandigd

Parsen:

Het parsen van een programma komt dus neer op het samenstellen van de parse-boom voor het bronprogramma. Hierbij worden dus de tokens die aangeleverd werden door de lexicale analyzer, geïnterpreteerd aan de hand van de syntaxdiagrammen.

- Tokens samenstellen tot opdrachten
    - Herkennen van de grammaticale structuur van het programma
    - Door middel van syntaxregels, voorgesteld met syntaxdiagrammen, die de syntax van de programmeertaal definiëren
- Parse-boom
    - Representatie van de manier waarop de parser de grammaticale samenstelling van het programma interpreteert

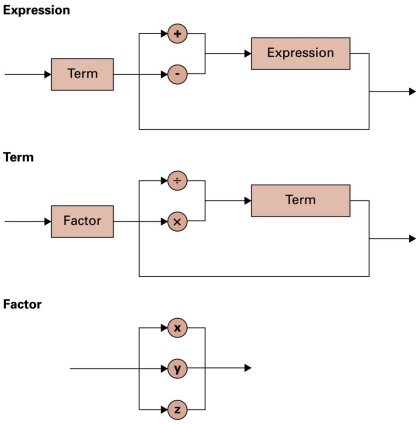
Syntaxdiagram voor keuze primitief van de pseudocode:



Uitleg: De termen in een syntaxdiagram die in ovalen of cirkels staan zijn **terminals**; de termen in vierkanten of rechthoeken zijn **nonterminals**. De nonterminals dienen nog verder beschreven te

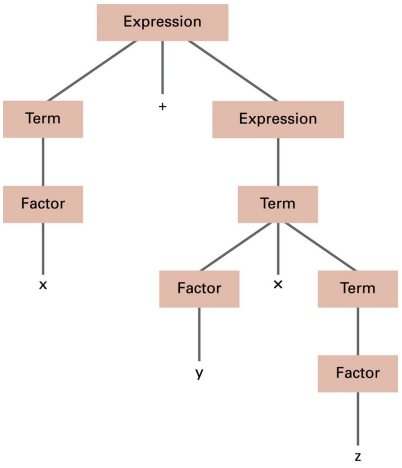
worden in syntaxdiagrammen, terwijl de terminals volledig gedefinieerd zijn. Het syntaxdiagram van het voorbeeld zegt dat een keuze-opdracht moet beginnen met if, gevolgd door een Booleaanse uitdrukking, gevolgd door een dubbele punt, gevolgd door een opdracht (Engels: statement). Deze combinatie kan (maar moet niet) gevolgd worden door else met dubbele punt en een opdracht

Voorbeeld: Syntaxdiagram

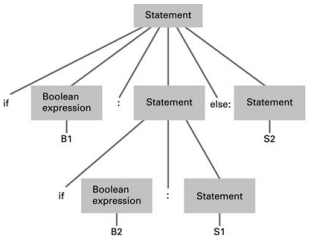


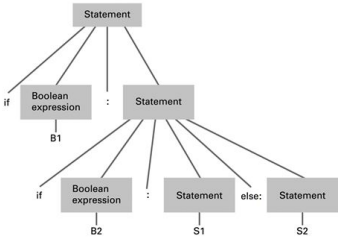
Uitleg: Het voorbeeld syntaxdiagram geldt voor een eenvoudige algebraïsche expressie in een niet nader gedefinieerde programmeertaal. Merk op dat een factor beschreven kan worden als één symbool waarbij enkel gekozen kan worden uit x, y of z (wat natuurlijk een sterke vereenvoudiging van de realiteit inhoudt).

Voorbeeld: Parse-boom voor X+Y*Z



Uitleg: Het voorbeeld betreft een parse-boom die de parser zou samenstellen voor de reeks tokens x + y * z die de lexicale analyzer doorstuurde, waarbij vertrokken wordt van de syntaxdiagrammen op de vorige slide.





Uitleg: Twee alternatieve parse-bomen voor de opdracht ‘if B1: if B2: S1 else: S2’. De semantiek van beide vormen is verschillend omdat de syntaxregel voor de keuzeprimitief van de pseudocode niet éénduidig is. In de pseudocode zoals gedefinieerd in hoofdstuk 5 treedt dit verschil in interpretatie niet op omdat ook indentatie gebruikt wordt.

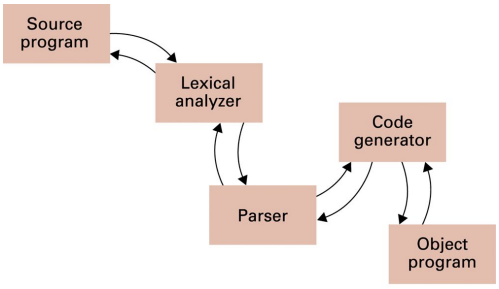
Symbooltabel:

- Parser analyseert declaratieve statements..
  - ..en slaat in symbooltabel op:
    - Identifiers van gedeclareerde variabelen
    - Gegevenstypen
    - eventuele gegevensstructuren die bij de variabelen horen
- Deze informatie wordt gebruikt voor het analyseren van imperatieve opdrachten
  - Vb. betekenis van + in **z = x + y** hangt af van de gegevenstypen van x en y

Generen van code:

- Samenstellen van machinetaalinstructies voor het implementeren van de door de parser herkende opdrachten
- Optimaliseren van de code
  - Vb. **z = x + y**
    w = z + y
    z en y zijn al in algemene CPU registers aanwezig na het uitvoeren van de eerste opdracht
    ⇒ niet meer terug inlezen vanuit werkgeheugen

Vertaalproces in actie:



Uitleg: Het bronprogramma, de lexicale analyzer, de parser, de codegenerator en het doelprogramma zijn objecten die met elkaar communiceren door berichten over en weer te sturen terwijl elke component van het vertaalprogramma zijn eigen taak uitvoert. De parser hoeft hierbij niet te wachten totdat de lexicale analyzer alle tokens heeft doorgestuurd want hij kan op basis van de grammaticale analyse van de reeds doorgestuurde tokens zelf bepalen of een opdracht volledig is of niet. Van zodra een opdracht volledig is doet de parser een verzoek aan de code-generator voor het samenstellen van machinetaalinstructies. De code-generator voegt na dit samenstellen de instructies toe aan het doelprogramma.

Linken en laden:

Uitleg: De **linker** koppelt doelprogramma's, routines van het besturingssysteem en andere utility software samen tot een volledig, uitvoerbaar programma (een **load module**) dat dan vervolgens in het massageheugensysteem wordt opgeslagen. De **loader** (een onderdeel van de scheduler) plaatst het programma in het werkgeheugen van de computer en past eventuele expliciete verwijzingen naar geheugenadressen aan. Tegenwoordig is het linken veelal een onderdeel van het vertaalproces geworden en zijn technieken voorhanden om expliciete verwijzingen naar geheugenadressen te vermijden.

Hoofdstuk 8

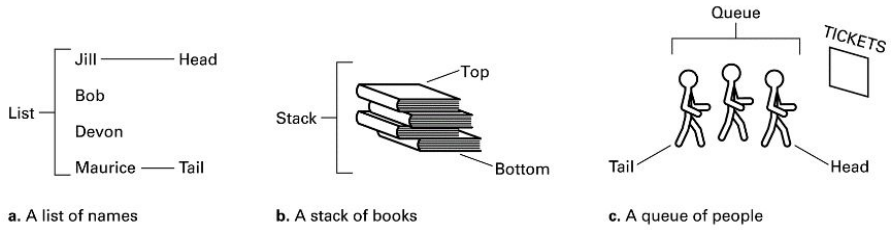
Gegevensabstracties

1. Basis gegevensstructuren

Een lijst is een verzameling gegevens waarvan de items achter elkaar, in een bepaalde volgorde (sequentieel), gerangschikt zijn. Een boom is een verzameling gegevens die hiërarchisch georganiseerd is. Er bestaan twee speciale soorten van lijsten: bij stacks (Nederlands: stapels) zal het laatste item dat toegevoegd wordt ook altijd het eerste item zijn dat verwijderd wordt. Stacks werken dus volgens het principe Last In First Out (LIFO). Bij wachtrijen zal het item dat eerst toegevoegd werd ook het eerste zijn dat verwijderd wordt. Wachtrijen werken dus volgens het First In First Out (FIFO) principe.

- Homogene arrays (confer Hfst 6)
    - Lijsten: homogene arrays van één dimensie, sequentieel georganiseerd (de volgorde is van belang)
        - Stacks: lijsten die werken volgens LIFO (last in first out)
        - Wachtrijen: lijsten die werken volgens FIFO (first in first out)
    - Bomen: homogene arrays van één dimensie, hiërarchisch georganiseerd (bepaalde elementen staan boven anderen)
- Heterogene arrays (confer Hfst 6)

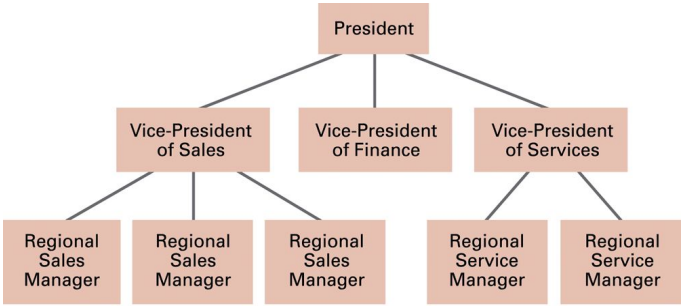
Voorbeelden van lijsten, stacks en wachtrijen:



a. A list of names **b.** A stack of books **c.** A queue of people

a. volgorde is van belang (alfabetische lijst)
b. LIFO
c. FIFO

Voorbeeld van een boomstructuur: het organigram (geen binaire boom, deze heeft elk punt maar 2 vertakkingen)



Terminologie voor lijsten:

- Kop: eerste item van de lijst (begin van de lijst)
- Staart: laatste item van de lijst (einde van de lijst)

Terminologie voor stacks:

- Top: kop van een stack
- Onderkant of basis (Engels: base): staart van een stack
- Toevoegen/verwijderen kan alleen aan de top:
    - Pushen: toevoegen van een item aan de top
    - Poppen: verwijderen van het item aan de top

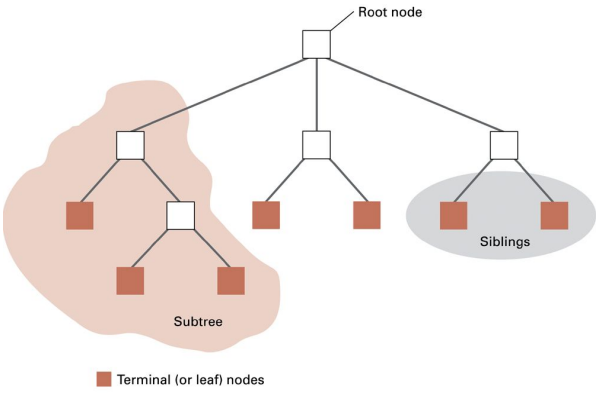
Terminologie voor wachtrijen:

- Dezelfde terminologie als bij 'gewone' lijsten
- Toevoegen/verwijderen van items:
    - Toevoegen van items aan de staart
    - Verwijderen van items aan de kop

Terminologie voor bomen:

Met 'node' kan verwezen worden naar een positie in een hiërarchische structuur, alsook naar het item dat op deze positie terug te vinden is. De termen 'top', 'onderkant', 'boven' en 'onder' zijn relatief t.o.v. de gebruikelijke oriëntatie bij hiërarchische structuren, waarbij vertakkingen naar beneden gebeuren i.p.v. naar boven (zoals bij bomen in de natuur meestal het geval is).

- Node
    - Een item in een boom
    - Een positie in een hiërarchische structuur
- Root node (vb. president)
    - Uniek item aan de 'top' van de boom
    - Er zijn geen nodes 'boven' de root node
- Eind node (Engels: leaf)
    - Item aan de 'onderkant' van de boom
    - Er zijn geen nodes 'onder' een eind node
    - Een boom heeft één of meerdere eind nodes
- Subboom
    - Elke node in een boom is de root node van een boom met alle nodes 'onder' deze node



Uitleg: Een gegevensstructuur zoals een boom zal moeten gesimuleerd worden in het werkgeheugen van de computer omdat het werkgeheugen bestaat uit één lange rij van opeenvolgend geadresseerde geheugencellen. Bij het gebruik van gegevensstructuren kan abstractie gemaakt worden van de manier waarop die structuren in het werkgeheugen gesimuleerd wordt.

- Ouder van een node
    - De node onmiddellijk ‘boven’ de beschouwde node
- Kinderen van een node
    - De nodes onmiddellijk ‘onder’ de beschouwde node
- Siblings
    - Alle nodes met dezelfde ouder
- Voorouders van een node
    - Alle nodes ‘boven’ de beschouwde node
- Nakomelingen van een node
    - Alle nodes ‘onder’ de beschouwde node
- Binaire boom
    - Boom waarin elke node maximaal twee kinderen heeft
- Diepte van een boom
    - Aantal nodes op het **langste pad** van de root node naar een eind node

2.1 Statische versus dynamische gegevensstructuur

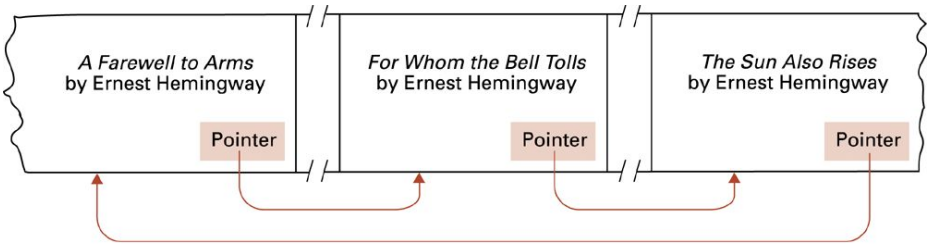
Dynamische gegevensstructuren zijn moeilijker te beheren dan statische structuren omdat de geheugenruimte die in beslag genomen wordt varieert.

- Statisch (bv. weekkalender bevat 7 dagen)
    - De grootte van de gegevensstructuur is onveranderlijk
- Dynamisch (bv. wachtrij aan een kassa)
    - De grootte van de gegevensstructuur is veranderlijk

2.2 Pointers (belangrijk in de informatica)

- Pointer = geheugenlocatie waarin het adres van een andere geheugenlocatie is opgeslagen
    - Kan gebruikt worden om gegevens te vinden en op te halen
- Instructie pointer = alternatieve benaming voor de programmateller van de CPU

Voorbeeld: lijst met romans aan elkaar gekoppeld met pointers ⇒ blockchain!



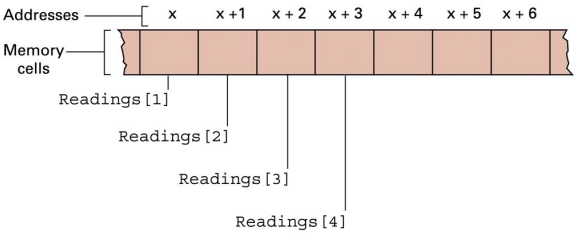
Uitleg: Stel dat de lijst met romans alfabetisch op titel gesorteerd is. Om nu alle romans van een bepaalde auteur terug te vinden kunnen we gebruik maken van pointers. In het blok van geheugencellen voor het opslaan van de gegevens van een roman reserveren we één geheugencel (of meerdere cellen afhankelijk van de lengte van de adressen) als pointer naar een volgend blok van geheugencellen met daarin de gegevens van een roman van dezelfde auteur. Dit blok is dan weer met een pointer gekoppeld aan een volgend blok, enzovoort. De pointer in het blok voor de laatste roman van die auteur kan terugverwijzen naar het blok met de eerste roman van dezelfde auteur. Van zodra een boek van een bepaalde auteur in de lijst gevonden is, kunnen we alle boeken van die auteur terugvinden door het volgen van de pointers.

3.1 Arrays opslaan

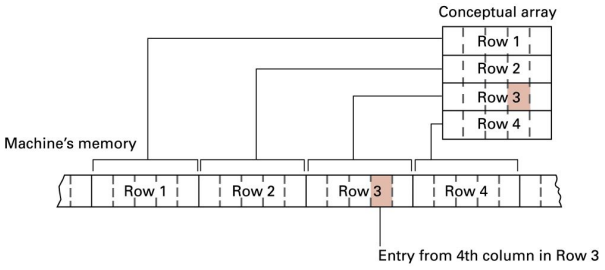
*Sommige programmeertalen (vb. VB.NET) laten toe dat bij de declaratie van een homogene array geen grootte wordt opgegeven, m.a.w. een dynamische gegevensstructuur wordt gedeclareerd. De uitdrukking x + (k.(i-1)) + (j-1) wordt de **adrespolynoom** of **adresveelterm** genoemd. In de voorbeelden op de slide wordt verondersteld dat elk array element één geheugencel in beslag neemt (vb. een geheel gtal in 8-bit 2-complement notatie is).*

- Homogene arrays => aaneengesloten reeks van geheugencellen gebruiken
- Statische gegevensstructuur: grootte bij declaratie opgeven
    - Vb. **int Rij[24];**
      int [][] scoresTable = new int [10, 20];
- Op welke adressen zijn array elementen opgeslagen?
    - 1 dimensie: Als adres van het eerste array element *x* is, dan is het adres van het *i*-de array element *x* + (*i* – 1)
    - 2 dimensies: Als adres van het eerste array element *x* is, dan is adres van het (*i*,*j*)-de array element *x* + (*k*.(*i*-1)) + (*j*-1), waarbij *k* het aantal kolommen is.

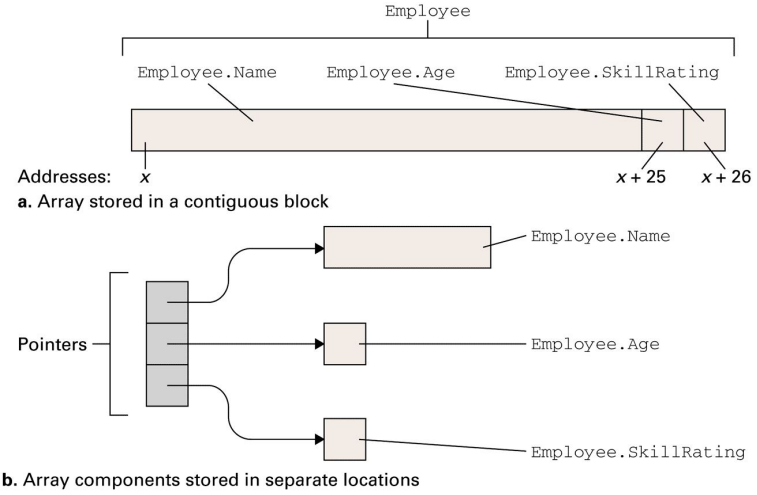
Voorbeeld: één dimensie



Voorbeeld: twee dimensies



<u>Heterogene arrays opslaan</u>:



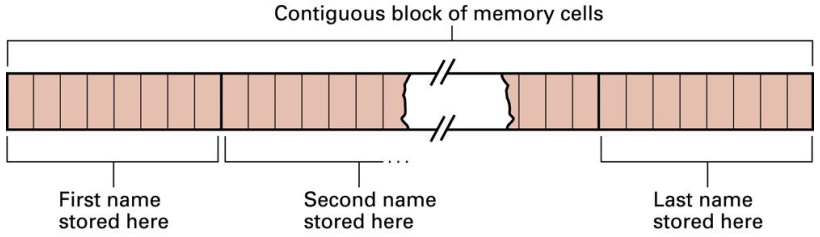
 heterogene arrays worden ook wel eens aggregaten genoemd

3.2 Lijsten opslaan

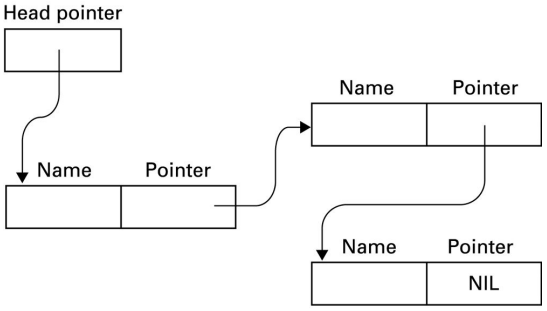
In de voorbeelden die volgen veronderstellen we dat we een lijst met namen moeten opslaan, waarbij elke naam maximaal 8 letters lang is. Indien we ASCII gebruiken volstaat het dus om voor elke letter één geheugencel (1 byte of 8 bits) te reserveren. Elke naam neemt 8 geheugencellen in. Indien namen korter zijn dan 8 letters, dan vullen we de lege geheugencellen op met een default teken, de ASCII code voor een spatie. Een namenlijst is statisch indien we op voorhand kunnen aangeven hoe groot de lijst is. Een namenlijst is dynamisch indien we dat niet kunnen.

- Aaneengesloten lijst → voor statische lijsten
    - Lege plaatsen opvullen met een default teken
- Gekoppelde lijst → voor dynamische lijsten
    - Items in de lijst kunnen verspreid zitten in het werkgeheugen
    - Items worden gekoppeld door middel van pointers
        - Voor elk item een extra geheugencel voorzien als pointer naar het volgende item in de lijst
        - Kop pointer: Pointer naar de kop van de lijst
        - NIL pointer: Pointer van de staart van de lijst

Voorbeeld: Aaneengesloten lijst

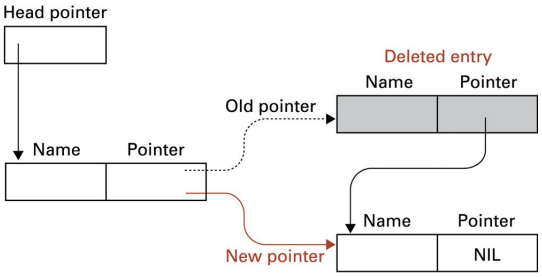


Voorbeeld: Gekoppelde lijst



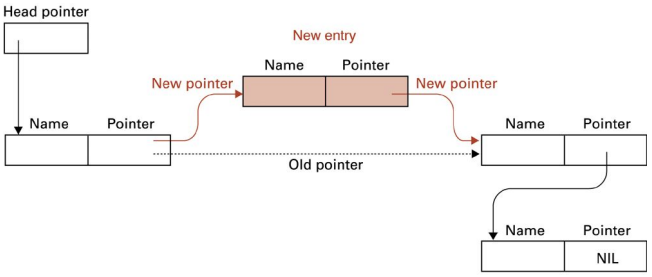
Uitleg: De waarde NIL kan een adres zijn in het werkgeheugen waar geen items opgeslagen kunnen worden, bijvoorbeeld het adres 0000 0000. De koppointer (Engels: head pointer) neemt één of meerdere extra geheugencellen in beslag, afhankelijk van de lengte van de adressen van de geheugencellen.

 Een item verwijderen uit een lijst:



Uitleg: Het verwijderen van een item uit de lijst kan gedaan worden door het veranderen van de pointer van het item dat voorafgaat aan het te verwijderen item. De nieuwe pointer waarde is het adres van het item dat volgt op het te verwijderen item.

 Een item toevoegen aan een lijst:



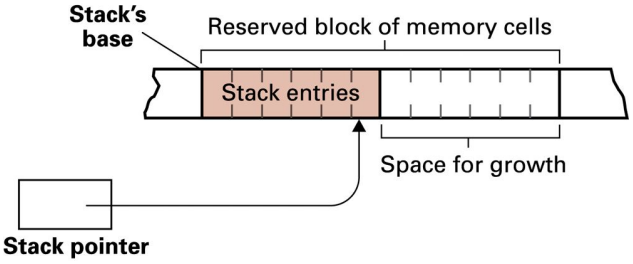
Uitleg: Items in een lijst zijn sequentieel georganiseerd, vb. een alfabetisch gesorteerde namenlijst. Nieuwe items moeten daarom vaak tussengevoegd worden in plaats van toegevoegd aan de staart van de lijst. Bij het tussenvoegen van een item moet ook de pointer van het item dat voorafgaat aan het nieuwe item veranderen.

3.3 Stacks opslaan

De ruimte in het werkgeheugen gereserveerd voor de stack mag ook niet te groot zijn, want dan wordt opslagcapaciteit verspild.

- Aaneengesloten blok geheugencellen
  - Groot genoeg om groei van de stack op te vangen
  - Eén uiteinde is de basis
    - Hier wordt eerste item gepusht
    - Volgende items hiernaast gepusht in richting van het andere uiteinde
  - Stack pointer
    - Pointer naar de top van de stack
    - Top schuift heen en weer bij pushen/poppen

Voorbeeld: Stack



Uitleg: Als een nieuw item gepusht wordt, wordt eerst de stack pointer aangepast zodanig dat deze verwijst naar de eerstvolgende lege locatie aan de top van de stack; daarna wordt het item in deze locatie opgeslagen. Bij het poppen wordt de stack pointer gevolgd om het item aan de top te vinden; daarna passen we de stack pointer aan zodanig dat deze verwijst naar het eerstvolgende item in de richting van de basis van de stack.

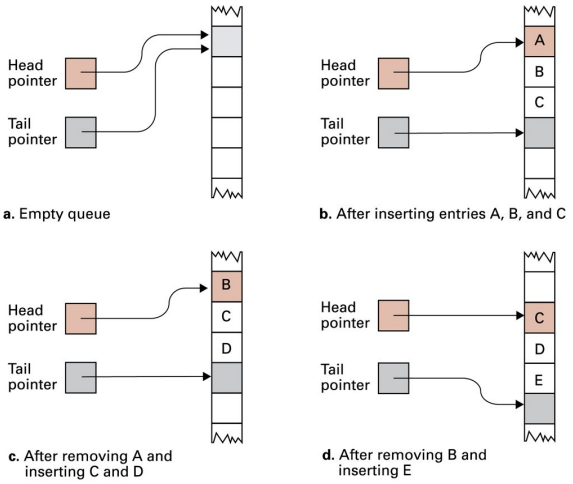
3.4 Wachtrijen opslaan

*Bij een **cirkelvormige wachtrij** verwijst de staart pointer naar het begin van het blok geheugencellen wanneer een item opgeslagen werd in de laatste locatie van het blok.*

 Aaneengesloten blok geheugencellen

- Groot genoeg om groei van de wachtrij op te vangen
- Eén uiteinde is de staart
  - Staartpointer verwijst naar lege locatie volgend op staart
  - Hier wordt een nieuw item toegevoegd
- Ander uiteinde is de kop
  - Koppointer verwijst naar deze locatie
  - Bij verwijderen wordt het item waarnaar de koppointer verwijst verwijderd

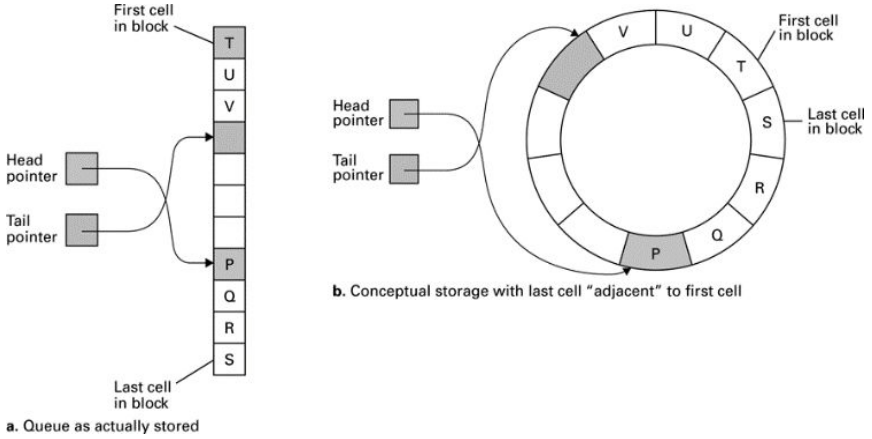
Voorbeeld: Wachtrij



Uitleg: Als de wachtrij leeg is verwijzen zowel koppointer als staartpointer naar dezelfde lege geheugenlocatie. Bij het gebruik van de wachtrij verschuift het deel van de geheugenlocaties dat door items bezet wordt doorheen het blok van geheugencellen dat voor de wachtrij gereserveerd werd.

(bij C: C weg, enkel D)

Voorbeeld: Circulaire wachtrij → wanneer je meer dan 20 cellen nodig hebt



3.5 Binaire bomen opslaan

 Gekoppelde structuur

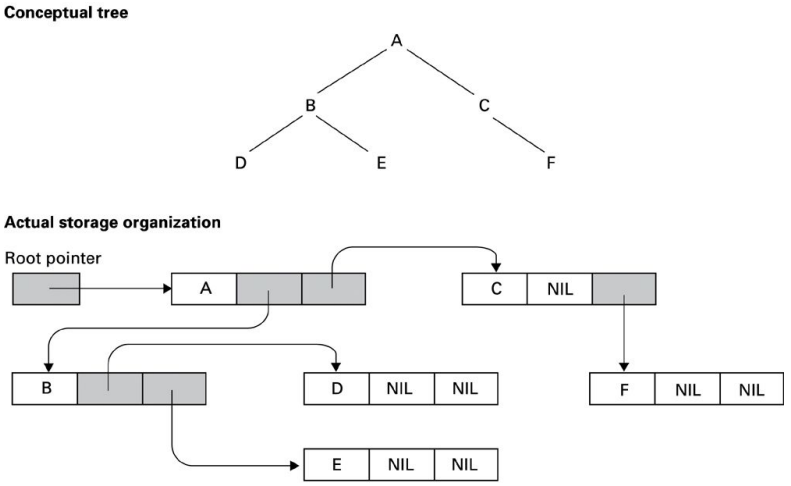
Binaire bomen kunnen opgeslagen worden als een gekoppelde structuur.

- Elk item bestaat uit drie componenten
  - Het gegeven
  - Linker pointer (bevat het adres van geheugenlocatie waar 'eerste' kind is opgeslagen of NIL)
  - Rechter pointer (bevat het adres van geheugenlocatie waar 'tweede' kind is opgeslagen of NIL)
- Rootpointer
  - Geheugencel met adres van de rootnode

Structuur van een node in een binaire boom:

Cells containing the data	Left child pointer	Right child pointer

Voorbeeld: Binaire boom als gekoppelde structuur

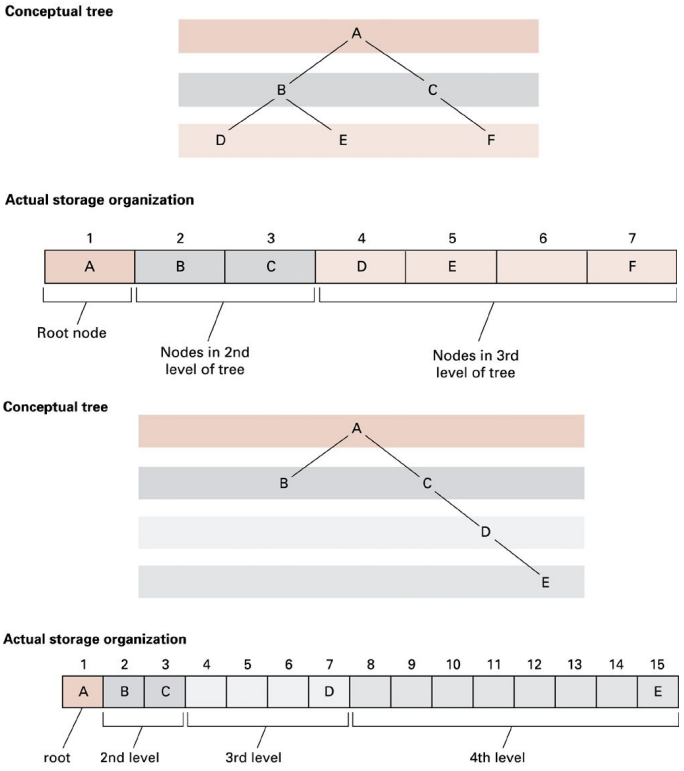


 aaneengesloten blok geheugencellen

..of als een aaneengesloten blok geheugencellen. Ook in dit voorbeeld veronderstellen we dat elke node één geheugencel zal beslaan.

- Nodes hiërarchisch, per niveau sequentieel opslaan
  - Rootnode in 1ste cel van het blok
  - Voor een node in de *n*-de cel van het blok
    - Linker kind in 2*n* - de cel
    - Rechter kind in 2*n*+1 – ste cel
- Ongebruikte locaties worden gemarkeerd met een default bitpatroon

Voorbeeld: Binaire boom als een aaneengesloten structuur



3.6 Bewerken van gegevensstructuren

De gegevensstructuur (die voor een programma gedefinieerd kan worden met een declaratief statement) samen met de verzameling procedurele eenheden die toelaten om met deze gegevensstructuur te werken, vormt een abstract gereedschap dat bij het programmeren in een hogere programmeertaal gebruikt kan worden, zonder ons te hoeven bekommeren om de manier waarop de gegevensstructuur echt is opgeslagen.

- Abstractie van fysieke opslagstructuur
- Procedurele eenheden om met gegevensstructuur te werken
    - Vb. voor stack minstens
        - een procedurele eenheid voor pushen
        - een procedurele eenheid voor poppen

Voorbeeld: Functie (in pseudocode) om een gekoppelde namenlijst af te drukken

```
def PrintList(List):
    CurrentPointer = List.Head
    while(CurrentPointer is not None):
        print(CurrentPointer.Value)
        CurrentPointer = CurrentPointer.Next
```

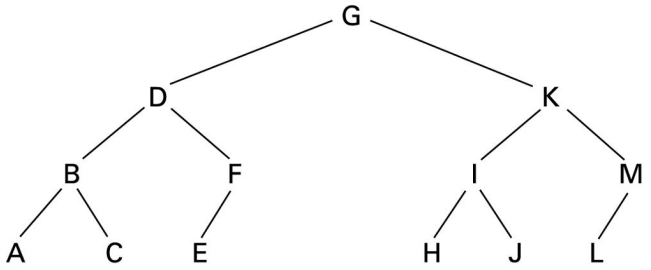
In de Python-gebaseerde pseudocode van Hoofdstuk 5 gebruiken we 'None' om te verwijzen naar de NIL-pointer.

4. Gevallenstudie: Alfabetisch gesorteerde lijsten

Omdat de lijst binair doorzocht moet kunnen worden is het handiger om hier, in plaats van een lijst, een binaire boom als gegevensstructuur te gebruiken, waarbij het middelste item van de lijst de rootnode wordt. Elke daaropvolgende node is dan het middelste item van de lijst die gevormd wordt door deze node als rootnode van een subboom te beschouwen. Omdat de lijst dynamisch is zullen we als opslagstructuur een gekoppelde boomstructuur (dus pointers) gebruiken.

- We wensen een alfabetisch gesorteerde lijst met items bij te houden
- Volgende bewerkingen moeten mogelijk zijn
    - Zoeken naar de aanwezigheid van een item
    - Afdrukken van de lijst
    - Een nieuw item aan de lijst toevoegen
- Verdere specificaties
    - De lijst is dynamisch
    - Lijst moet binair doorzocht kunnen worden

Voorbeeld: De letters A t.e.m. M alfabetisch gesorteerd in een binaire boom

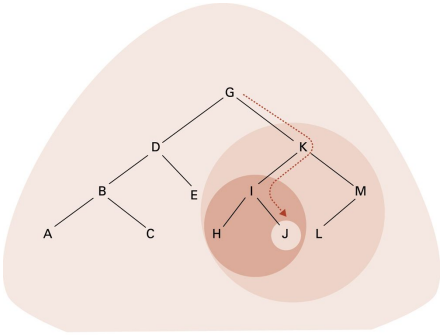


Functie voor binair zoeken:

```
def Search(Tree, TargetValue):
    if(Tree is None):
        return None        # Search failed
    elif(TargetValue == Tree.Value):
        return Tree        # Search succeeded
    elif(TargetValue < Tree.Value):
        # Continue search in left subtree
        return Search(Tree.Left, TargetValue)
    elif(TargetValue > Tree.Value):
        # Continue search in right subtree
        return Search(Tree.Right, TargetValue)
```

Uitleg: Merk op dat deze functie afgeleid is van de recursieve functie voor binair zoeken uit hoofdstuk 5. De 'elif' is een verkorte notatie voor 'else: if ..' (op Python gebaseerd). De parameter Tree is een rootpointer die verwijst naar de rootnode van de boom die doorzocht moet worden.

Voorbeeld: Binair zoeken van J

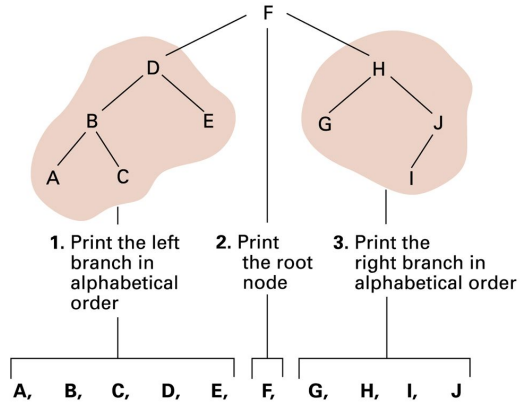


Procedure voor afdrukken:

```
def PrintTree(Tree):
    if(Tree is not None):
        PrintTree(Tree.Left)
        print(Tree.Value)
        PrintTree(Tree.Right)
```

Uitleg: Merk op dat hier opnieuw van recursie gebruik gemaakt wordt. Voor elke (sub)boom is de rootnode het middenste item dat afgedrukt moet worden.

Voorbeeld: Afdrukken van alfabetische lijst met letters A t.e.m. J

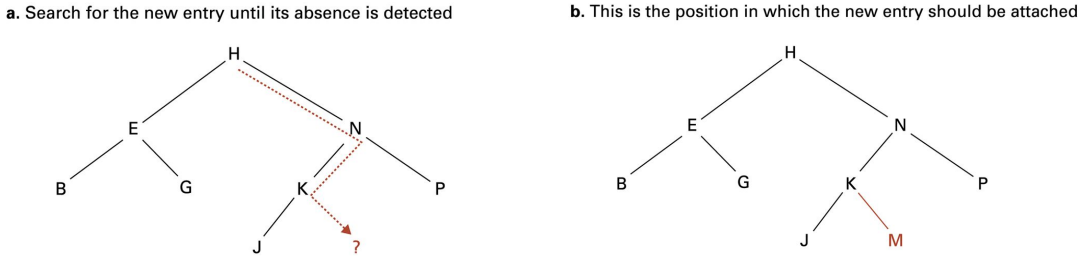


Functie voor toevoegen:

```
def Insert(Tree, NewValue):
    if(Tree is None):
        # Create a new leaf with NewValue
        Tree = TreeNode()
        Tree.Value = NewValue
    elif(NewValue < Tree.Value):
        # Insert NewValue into the left subtree
        Tree.Left = Insert(Tree.Left, NewValue)
    elif(NewValue > Tree.Value):
        # Insert NewValue into the right subtree
        Tree.Right = Insert(Tree.Right, NewValue)
    else:
        # Make no change
    return Tree
```

Uitleg: Ook hier wordt van recursie gebruik gemaakt. Merk op dat de functie controleert of het toe te voegen item al aanwezig was.

Voorbeeld: M invoegen in alfabetisch gesorteerde reeks B, E, G, H, J, K, N, P



5. Door gebruikers gedefinieerde gegevenstypen

- Zelf gemaakt op basis van
  - primitieve gegevenstypen..
  - ..of eerder gedefinieerde gegevenstypen
- Variabelen kunnen gedeclareerd worden met deze 'nieuwe' gegevenstypen
  - In plaats van voor elk van deze variabelen een heterogeen array met telkens dezelfde structuur te declareren

Voorbeeld

Definitie gegevenstype:

```
definieer type MedewerkerType als
{ char Naam [24];
  int Leeftijd;
  real FunctieCode;
}
```

Declaratie variabelen:

```
MedewerkerType medewerker1, medewerker2;
```

*In het voorbeeld definiëren we een nieuw gegevenstype voor het beschrijven en opslaan van de gegevens van medewerkers in een organisatie. Vervolgens declareren we twee variabelen, medewerker1 en medewerker2, die MedewerkerType als gegevenstype hebben. Deze variabelen zijn **instanties** van het nieuw gedefinieerde gegevenstype.*

<u>Abstracte gegevenstypen</u>:

Uitbreiding van een gebruiker gedefinieerd gegevenstype met procedurele eenheden voor bewerkingen op gegevens van het type.

Hoofdstuk 9

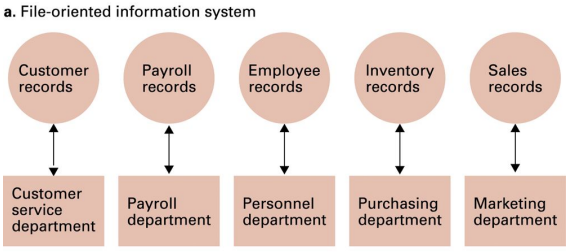
Databasesystemen

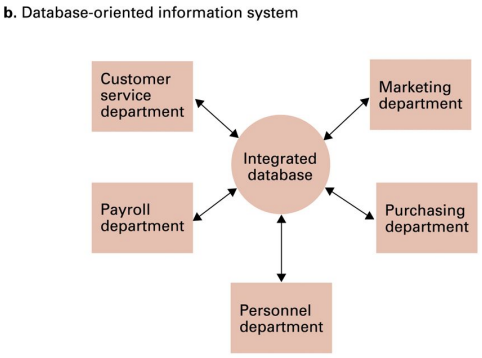
1. Basisprincipes van databases

Als studentengegevens opgeslagen worden in een flat file op een massageheugensysteem, bijvoorbeeld op alfabetische volgorde van naam, dan kunnen deze gegevens later enkel maar teruggevonden worden op basis van de naam van een student. Als studentengegevens opgeslagen worden in een database op een massageheugensysteem, dan kunnen zij teruggevonden worden op basis van eender welk kenmerk van een student waarvoor gegevens bewaard worden (vb. naam, stamnummer, studierichting, studieresultaten, ...).

- Database
    - Verzameling van gegevens met meerdere dimensies
    - Gegevens kunnen vanuit verschillende perspectieven benaderd worden
- Flat file (Nederlands: plat bestand)
    - Eéndimensionele verzameling gegevens
    - Gegevens kunnen maar vanuit één perspectief benaderd worden

<u>Bestandsorganisatie versus databaseorganisatie van gegevens</u>:





Uitleg: Bij een bestandsgeoriënteerd informatiesysteem bewerkt elk toepassingsprogramma een eigen bestand met gegevens. Al deze bestanden bestaan onafhankelijk van elkaar ook al bevatten ze gedeeltelijk dezelfde gegevens. Relaties tussen gegevens van verschillende bestanden worden niet bijgehouden.

In een databasegeoriënteerd informatiesysteem wordt een geïntegreerde verzameling van gegevens bijgehouden. Alle toepassingsprogramma's werken op dezelfde verzameling van gegevens. Ze delen met andere woorden eenzelfde database.

<u>Databaseschema's</u>:

Het delen van gegevens door een ganse reeks toepassingen brengt het gevaar met zich mee dat toepassingen andere gegevens gaan raadplegen (of erger nog, wijzigen) dan deze waarvoor de toepassing bedoeld is. Door het gebruik van een subschema kan het deel van de database dat voor een bepaalde toepassing/gebruiker niet relevant is, afgeschermd worden. De toepassing heeft m.a.w. enkel weet van de gegevens die in zijn subschema gedefinieerd zijn.

- Schema
    - Beschrijving van de structuur van de database
    - Wordt gebruikt om de database te beheren

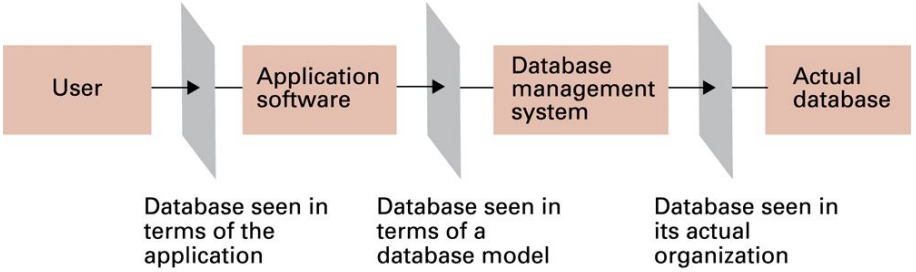
- Subschema
    - Beschrijving van de structuur van een deel van de database, nodig voor een bepaalde toepassing of gebruiker
    - Wordt gebruikt om de toegang tot de gegevens te beperken

Databasemanagementsysteem:

*Het DBMS is een abstract gereedschap voor toepassingsprogamma's. Het DBMS biedt procedures aan via dewelke toepassingsprogramma's kunnen werken met een database zonder zich te hoeven bekommeren om de manier waarop de gegevens fysiek zijn opgeslagen (bijvoorbeeld, zonder bewust te zijn van een eventuele keuze die gemaakt werd tussen een **gedistribueerde database** of een **gecentraliseerde database**). Het DBMS kan de toegang tot de gegevens door toepassingsprogramma's controleren door de beperkingen die de subschema's opleggen af te dwingen. Zelf maakt het DBMS gebruik van het gehele databaseschema. Met gegevensonafhankelijkheid wordt bedoeld dat wijzigingen aan de structuur van de database enkel maar effect hebben op die subschema's die gegevens beschrijven waarvan de structuur of definitie gewijzigd werd. Enkel de programma's die van deze subschema's gebruik maken moeten ook gewijzigd worden.*

- Programma voor het beheren van een database
- Voert zoek- en updateopdrachten uit voor toepassingsprogramma's (of gebruikers)
- Ondersteunt
    - Abstractie
    - Gegevensbeveiliging
    - Gegevensonafhankelijkheid

Het DBMS als database interface voor toepassingssoftware:



Uitleg: **Databasetoepassingen** zijn gegevensverwerkende programma's die gebruik maken van in een database opgeslagen gegevens. Deze toepassingen bestaan uit twee lagen: een toepassingsprogramma via hetgene gebruikers (fysieke personen of andere programma's) kunnen werken met de gegevens van de database en het DBMS dat door dit toepassingsprogramma gebruikt wordt om zoek- of updateopdrachten uit te voeren. Het DBMS is voor het toepassingsprogramma een interface naar de database, terwijl het toepassingsprogramma voor zijn gebruikers een interface is naar de database. Zoals de figuur toont wordt bij dit systeem uitgebreid gebruik gemaakt van het principe van abstractie.

Toepassingsprogramma's en DBMS worden meestal op verschillende genetwerkte computers opgeslagen en uitgevoerd, d.w.z. het DBMS speelt de rol van **server** en de toepassingssoftware speelt de rol van **client**.

Databasemodel:

Hogere programmeertalen laten veelal toe om de routines van het DBMS, die procedures implementeren voor het opzoeken en updaten van gegevens in een database, te gebruiken in programma's.

- Het DBMS schermt de interne structuur van de database af voor de toepassingsprogramma's..
- ..door middel van een databasemodel
    - Conceptueel beeld van de database
    - Routines voor het werken met gegevens

2. Het relationele databasemodel

Elk gegeven in een relatie opgeslagen is de waarde voor een attribuut van een tupel, bijvoorbeeld in een relatie voor medewerkersgegevens is "G. Jerry Smith" de waarde van het attribuut 'name' voor de tupel die deze medewerker voorstelt.

- Database georganiseerd als een verzameling van onderling gekoppelde tabellen
- Concepten
    - Relatie
        - Tabel voor het opslaan van gegevens m.b.t. een bepaalde soort van object
    - Tupel
        - Rij in een relatie met gegevens over één object
    - Attribuut
        - Kolom in een relatie die een kenmerk voorstelt van het soort van object voorgesteld door de relatie

Voorbeeld: Relatie voor het opslaan van medewerkersgegevens

Empl Id	Name	Address	SSN
25X15	Joe E. Baker	33 Nowhere St.	111223333
34Y70	Cheryl H. Clark	563 Downtown Ave.	999009999
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555
•	•	•	•
•	•	•	•
•	•	•	•

Uitleg: De relatie MEDEWERKER bevat de gegevens van de medewerkers van een organisatie. Voor elke medewerker (tupel) worden gegevens bijgehouden over de kenmerken (attributen) Id, naam, adres en SSN (social security number).

Database ontwerp: mogelijke problemen

De relaties die in een database opgenomen worden zijn niet willekeurig gekozen. Het ondoordacht kiezen van relaties kan tot problemen van gegevensredundantie en gegevensverlies leiden zoals het voorbeeld op de volgende slide illustreert. In dit voorbeeld gaan we ervan uit dat we voor elke medewerker ook gegevens over het arbeidsverleden van die medewerker willen bijhouden:

- *Functietitel*
- *Functiecode*
- *Kwalificatie nodig voor die functie*
- *Afdeling waaronder de functie valt*
- *Periode waarin de medewerker die functie heeft vervuld (of desgevallend nog steeds vervult)*

 Als meer dan één soort van object in een relatie bijgehouden wordt

- Gegevensredundantie
    - Gegevens moeten meer dan één keer opgeslagen worden
- Gegevensverlies
    - Het weglaten van een tupel kan het ongewenst neveneffect hebben dat ook andere gegevens verloren gaan.

Voorbeeld: Een relatie met redundante

Empl Id	Name	Address	SSN	Job Id	Job Title	Skill Code	Dept	Start Date	Term Date
25X15	Joe E. Baker	33 Nowhere St.	111223333	F5	Floor manager	FM3	Sales	9-1-2002	9-30-2003
25X15	Joe E. Baker	33 Nowhere St.	111223333	D7	Dept. head	K2	Sales	10-1-2003	*
34Y70	Cheryl H. Clark	563 Downtown Ave.	999009999	F5	Floor manager	FM3	Sales	10-1-2002	*
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555	S25X	Secretary	T5	Personnel	3-1-1999	4-30-2001
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555	S26Z	Secretary	T6	Accounting	5-1-2001	*
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.

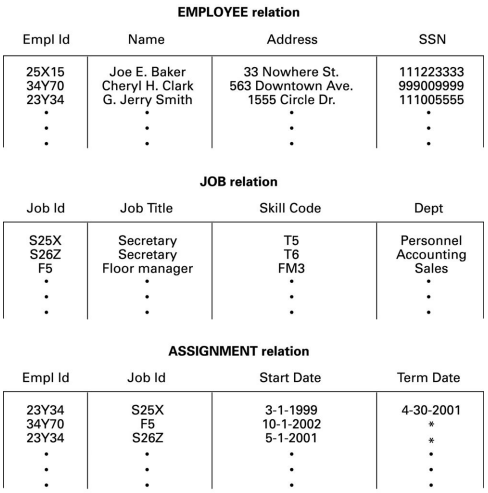
Uitleg: Als een medewerker een arbeidsverleden heeft met twee of meer functies, dan worden de medewerkergegevens herhaald in de tabel. Als een functie door twee of meer medewerkers vervuld werd (of nog vervuld wordt) dan worden de functiegegevens herhaald. Deze redundantie maakt de gegevensopslag weinig efficiënt. Bovendien kan het tot ongewenst gegevensverlies aanleiding geven. Bijvoorbeeld, stel dat “Joe E. Baker” ontslag neemt en dat we zijn gegevens uit de relatie verwijderen. Indien nu de functie met Job id D7 niet meer zou voorkomen in de tabel, dan verliezen we ook de informatie dat functie D7 een functie was binnen het Sales departement, dat voor functie D7 kwalificatie K2 nodig is en dat de bijhorende functietitel die van “Dept. head” is.

2.1 Decompositie

Voor elk concept (soort van object) waarover we gegevens wensen bij te houden zouden we een relatie moeten creëren. Indien we gegevens over werknemers en hun arbeidsverleden wensen te bewaren hebben we een tabel nodig voor de werknemergegevens, een tweede tabel voor functiegegevens en een derde tabel voor het bewaren van de toewijzingen van functies aan werknemers. Door het dupliceren van sommige kolommen worden deze drie tabellen aan elkaar gekoppeld, zodat van elke werknemer het arbeidsverleden kan opgezocht worden en van elke functie de werknemers opgezocht kunnen worden die deze functie bekleedden (of nog bekleden).

- Decompositie
    - Het verdelen van de kolommen van een relatie over twee of meer relaties
    - Kolommen dupliceren indien deze gebruikt kunnen worden voor het koppelen van tupels uit verschillende relaties
- *Losless/nonloss decomposition*
    - Decompositie zonder gegevensverlies

Voorbeeld: Relaties zonder redundantie



EMPLOYEE relation

Empl Id	Name	Address	SSN
25X15	Joe E. Baker	33 Nowhere St.	111223333
34Y70	Cheryl H. Clark	563 Downtown Ave.	999009999
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555

JOB relation

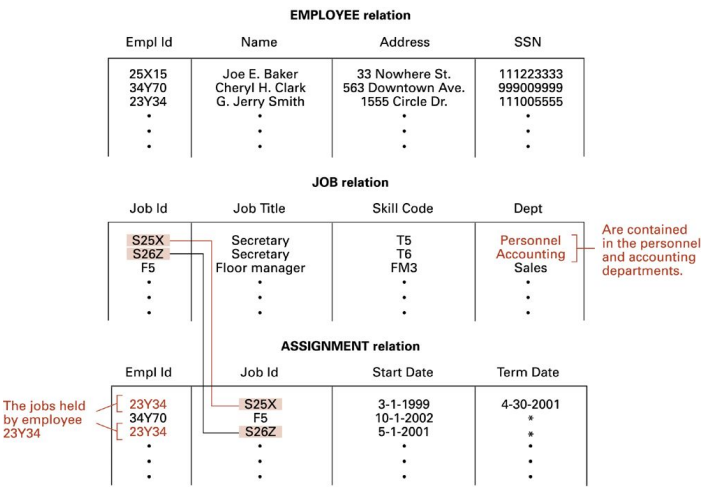
Job Id	Job Title	Skill Code	Dept
S25X	Secretary	T5	Personnel
S26Z	Secretary	T6	Accounting
F5	Floor manager	FM3	Sales

ASSIGNMENT relation

Empl Id	Job Id	Start Date	Term Date
23Y34	S25X	3-1-1999	4-30-2001
34Y70	F5	10-1-2002	*
23Y34	S26Z	5-1-2001	*

Uitleg: De koppeling tussen de WERKNEMER en JOB relaties komt tot stand via de ASSIGNMENT relatie, waarin de kolom Empl Id van WERKNEMER en de kolom Job Id van JOB herhaald worden. Gegevens van werknemers zoals naam, adres en SSN komen slechts één keer voor. Ook gegevens van functies zoals functie titel, kwalificatie en afdeling waaronder de functie resorteert, komen slechts één keer voor. Het verwijderen van een werknemer leidt nu niet langer tot verlies aan informatie m.b.t. functies.

Voorbeeld: Zoeken van afdelingen voor een medewerker



Uitleg: De afdelingen waarop medewerker 23Y34 heeft gewerkt of nog werkt kunnen gevonden worden door voor elke tupel voor die medewerker in de ASSIGNMENT tabel het job id op te zoeken in de JOB tabel en daar de waarde van Dept af te lezen.

2.2 Relationele bewerkingen

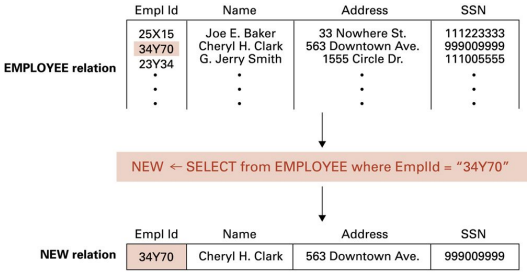
Een relationeel DBMS biedt routines (functies) aan om SELECT-, PROJECT- en JOIN-bewerkingen op een relationele database uit te voeren. Toepassingsprogramma's kunnen deze DBMS routines aanroepen om op die manier informatie uit de database te halen.

Er bestaat een standaardtaal voor relationele DBMSen die deze (en andere) routines aanbiedt: de ***Structured Query Language*** *(****SQL****).*

Hoe informatie uit een relationele database halen?

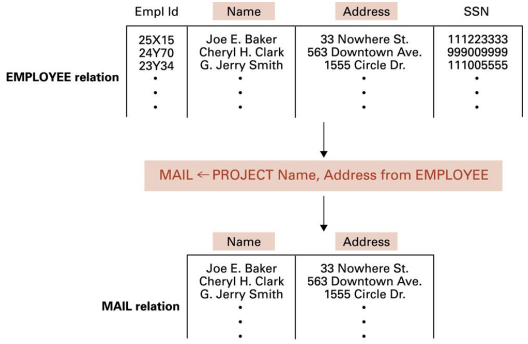
- SELECT-bewerking: tupels uit een relatie selecteren
- PROJECT-bewerking: relatie projecteren op één of meerdere kolommen
- JOIN-bewerking: twee relaties combineren tot één relatie, veelal om er vervolgens SELECT en/of PROJECT op toe te passen

Voorbeeld: Select

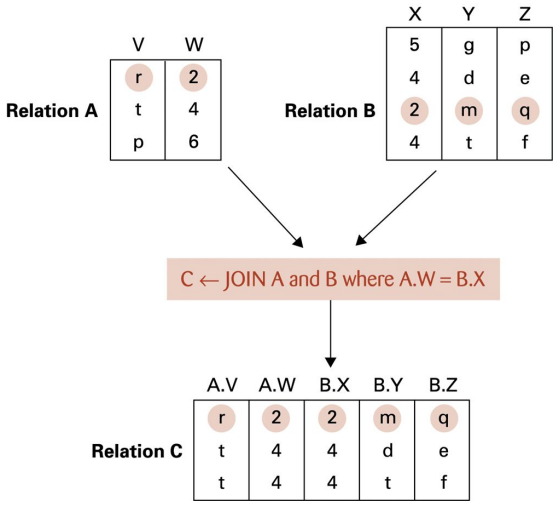


Uitleg: Wanneer de SELECT-bewerking toegepast wordt op een relatie ontstaat een nieuwe relatie met daarin enkel de tupels die aan de selectieconditie(s) voldoen. In het voorbeeld worden uit EMPLOYEE de gegevens geselecteerd van de medewerker met Empl Id "34Y70". Het resultaat van een SELECT-bewerking kan een lege relatie zijn, een relatie met één tupel zijn (zoals in dit voorbeeld) of een relatie met meerdere tupels zijn.

Voorbeeld: Project

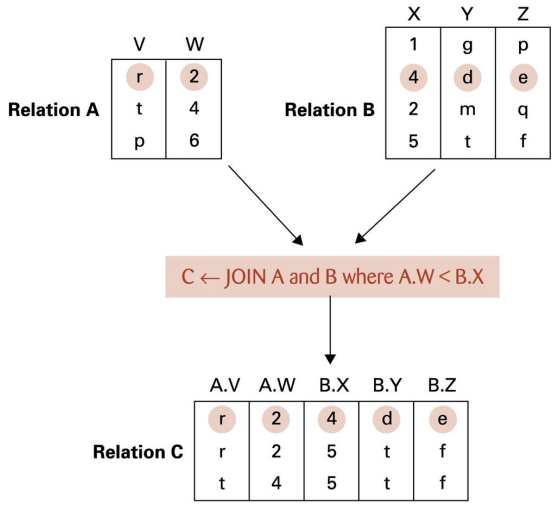


Uitleg: In het voorbeeld wordt de relatie EMPLOYEE geprojecteerd op twee van zijn kolommen: naam en adres. Dit betekent dat een nieuwe relatie (MAIL) gecreëerd wordt met voor elke tupel uit EMPLOYEE enkel de gegevens van de kolommen naam en adres.



Voorbeeld: Join

Uitleg (1): In het voorbeeld is C een nieuwe relatie die ontstaat uit het samenvoegen van de relaties A en B. De kolommen van C zijn de kolommen van A gevolgd door de kolommen van B. Om verwarring te vermijden wordt elke kolomnaam voorafgegaan door de naam van de relatie waaruit de kolom kwam. De tupels van C zijn de tupels die voldoen aan de JOIN-conditie. In het voorbeeld stelt de JOIN-conditie dat een tupel van A gecombineerd wordt met een tupel van B indien voor die tupel van A de waarde van de W kolom gelijk is aan de waarde van de X kolom voor de beschouwde tupel uit B.



Uitleg (2): de JOIN-conditie hoeft niet noodzakelijk een gelijkheid tussen twee kolomwaarden voorop te stellen. Zoals het voorbeeld toont, kunnen ook andere voorwaarden gesteld worden.

Uitleg (3): In de JOIN-conditie wordt vaak verwezen naar die kolommen die gebruikt worden om koppelingen tussen relaties tot stand te brengen. In het voorbeeld zien we dat de kolom Job Id (of functiecode) aanwezig is in zowel de ASSIGNMENT relatie als in de JOB relatie. Tupels uit beide relaties

zijn gerelateerd indien ze dezelfde waarden hebben voor deze kolommen.