EXAMEN: **Computergebruik**

1e Bachelor Informatica
prof. dr. Peter Dawyndt
groep 1

maandag 09-01-2012, 08:30
academiejaar 2011-2012
eerste zittijd

Opgave 1

Brainfuck is een esoterische programmeertaal die in 1993 werd ontwikkeld door Urban Müller. Müllers doel was een Turing-complete programmeertaal te maken die met de kleinst mogelijke compiler zou kunnen geïmplementeerd worden, geïnspireerd op de 1024-byte compiler voor de programmeertaal `FALSE`. Er werden dan ook verschillende brainfuck compilers ontwikkeld die kleiner zijn dan 200 bytes. Zoals de naam reeds doet vermoeden zijn brainfuck programma's over het algemeen moeilijk te begrijpen. Daar staat tegenover dat elke Turingmachine, en daardoor ook brainfuck, elke rekentaak kan volbrengen.
De taal is gebaseerd op een zeer eenvoudige virtuele machine die — buiten het programma — bestaat uit een rij van bytes die op nul zijn gezet, een data-pointer die wijst naar een element van de rij (startwaarde van de data-pointer is de eerste byte) en twee rijen bytes die de invoer en uitvoer vormen. De acht statements waaruit de taal bestaat, en die elk worden voorgesteld door één enkel karakter, staan opgelijst in onderstaande tabel.

KARAKTER	BETEKENIS
`>`	verhoog data-pointer
`<`	verlaag data-pointer
`+`	verhoog byte waarnaar data-pointer wijst
`-`	verlaag byte waarnaar data-pointer wijst
`.`	geef byte waarnaar data-pointer wijst als ASCII-uitvoer
`,`	gebruik volgende ASCII-invoerbyte om byte waarnaar data-pointer wijst in te vullen
`[`	spring voorwaarts naar statement na corresponderende `]` indien byte waarnaar data-pointer wijst nul is
`]`	spring terug naar statement achter corresponderende `[` indien byte waarnaar data-pointer wijst geen nul is

Zoals dat gebruikelijk is bij haakjes, moeten ook in brainfuck de vierkante haakjes correct genest zijn. Onderstaande brainfuck programmacode genereert bijvoorbeeld de tekst `Hello World!`

```
++++++++++[>+++++++>++++++++++>+++>+<<<<-]>++.>+.+++++++..+++.>++.<<+++++++++++++++.>
.+++.------.--------.>+.>.
```

Op elke regel van het tekstbestand `brain.txt` staat een stukje brainfuck programmacode, gevolgd door een spatie en een woord dat enkel bestaat uit letters van het alfabet. Dit woord is de uitvoer die gegenereerd wordt door de voorafgaande brainfuck programmacode. Elk stukje brainfuck code bevat telkens één paar overeenkomstige vierkante haakjes. In onderstaande uitleg zullen we het codefragment tussen de vierkante haakjes benoemen als de *lus*, het codefragment voor het openende vierkante haakje als de *initialisatie* en het codefragment dat volgt op het afsluitende vierkante haakjes als de *uitvoer*. Gevraagd wordt:

1. Bepaal reguliere expressies voor elk van onderstaande verzamelingen, waarbij $\mathcal{B}$ de verzameling voorstelt van alle brainfuck programma's met één paar overeenkomstige vierkante haakjes. Probeer deze reguliere expressies bovendien zo kort mogelijk te houden.

(a) $\alpha = \{b \in \mathcal{B} \mid$ vierde en vierde-laatste symbool van uitvoer van b zijn gelijk$\}$

voorbeelden: `++[>+>++>+++<<<-]>++.+++..>+.<.>.` $\in \alpha$,
`++[>+>++>+++<<<-]>++.+++..>--.<-.>.` $\notin \alpha$

(b) $\beta = \{b \in \mathcal{B} \mid$ lus en uitvoer van b hebben eerste tien karakters gemeenschappelijk$\}$

voorbeeld: `++[>+++++++++>++++++>+<<<-]>+++++++++.>+.++..---.+.>.>.` $\in \beta$

(c) $\gamma = \{b \in \mathcal{B} \mid$ in uitvoer van b komen alternerend reeksen plus- en mintekens voor$\}$

opmerking: overige tekens worden gezien als scheiding tussen reeksen plus- en mintekens; ook `+-` en `-+` worden gezien als overgang tussen reeksen plus- en mintekens

voorbeelden: `++[>++>+++>+<<<-]>++++++++.---.+.>.<------.+.>.>.` $\in \gamma$,
`++[>++>+++>+<<<-]>+++++.>++++++.>++.++++.-----.>.` $\notin \gamma$

(d) $\delta = \{b \in \mathcal{B} \mid$ aantal mintekens tussen opeenvolgende `<` en `>` in b is altijd oneven$\}$

opmerking: met opeenvolgende `<` en `>` wordt bedoeld dat daartussen geen andere `<`-tekens of `>`-tekens staan; volgorde is belangrijk: aantal mintekens tussen opeenvolgende `>` en `<` speelt bijvoorbeeld geen rol; opeenvolgende `<` en `>` zonder tussenliggende mintekens ook niet toegelaten

voorbeelden: `++[>++>+++>+<<<-]>+.>++++++++.<-.----.+++++++.>>.` $\in \delta$
`++[>++>+++>+<<<-]>+++.>++++++.<<-.-.>>--.<---.>>.` $\notin \delta$

Gebruik een commando uit de `grep` familie om enkel die regels van het bestand `brain.txt` te selecteren met fragmenten brainfuck programmacode die behoren tot de opgegeven verzameling. Vermeld in je antwoordbestand voor elke verzameling het gebruikte selectiecommando, en geef telkens ook aan hoeveel regels je gevonden hebt.

2. Beschouw de verzamelingen α, β, γ en δ zoals hierboven gedefinieerd. Gebruik nu deze verzamelingen om op de volgende manier een boodschap bestaande uit vier woorden te achterhalen:

   (a) het eerste woord staat op de unieke regel met programmacode uit de verzameling $\alpha \cap \beta$

   (b) het tweede woord staat op de unieke regel met programmacode uit de verzameling $\beta \cap \gamma$

   (c) het derde woord staat op de unieke regel met programmacode uit de verzameling $\gamma \cap \delta$

   (d) het vierde woord staat op de unieke regel met programmacode uit de verzameling $\delta \cap \alpha$

   Vermeld in je antwoordbestand de gevonden woorden, samen met het unix commando (of de commandosequentie) dat je gebruikt hebt om elk van deze woorden te vinden. Elk commando of elke commandosequentie moet dus als resultaat één van de gezochte woorden naar standaard uitvoer schrijven (zonder de brainfuck programmacode die aan het woord voorafgaat).

Opgave 2

Gegeven is een tekstbestand `access.log` dat de historiek van de activiteiten van een webserver bevat. Elke regel van dit bestand bevat informatie over een vraag aan de webserver. Deze informatie wordt weergegeven als de reeks velden die hieronder staan opgelijst, en die van elkaar worden gescheiden door een spatie. Merk hierbij op dat sommige velden zelf ook spaties kunnen bevatten (zie beschrijving).

- *aanvraagadres*: adres van aanvrager; kan geen spaties bevatten
- *gebruikersnaam*: gebruikersnaam gebruikt bij aanvraag; aangegeven met `-` indien geen gebruikersnaam vermeld werd; kan geen spaties bevatten

- *wachtwoord*: wachtwoord gebruikt bij aanvraag; aangegeven met `-` indien geen wachtwoord gebruikt werd; kan geen spaties bevatten
- *timestamp*: tijdstip van aanvraag, afgebakend door vierkantje haakjes; binnen vierkantje haakjes kunnen spaties voorkomen
- *request*: geeft aan wat er precies werd aangevraagd, afgebakend door dubbele aanhalingstekens; binnen dubbele aanhalingstekens kunnen spaties voorkomen
- *statuscode*: getal dat `http` statuscode aangeeft
- *grootte*: grootte van antwoord van webserver, uitgedrukt in bytes; aangegeven met `-` indien geen antwoord werd teruggegeven

Gevraagd wordt om, gebruik makend van de teksteditor `vi` (of `vim`), een reeks (substitutie)commando's op te stellen die achtereenvolgens de volgende opdrachten uitvoeren. Elke opdracht kan met één enkel commando opgelost worden.

1. Zorg er voor dat de velden gescheiden worden door een asterisk (`*`). Zorg er ook voor dat de velden met gebruikersnaam en wachtwoord verwijderd worden. Zo moeten de regels

   ```
   plexus.v2.nl - - [22/Jan/2003:19:04:01 +0100] "GET / HTTP/1.0" 200 395
   dyn143.nl - - [23/Jan/2003:13:34:11 +0100] "GET /html/index.html HTTP/1.1" 404 287
   ```

   omgezet worden naar

   ```
   plexus.v2.nl*[22/Jan/2003:19:04:01 +0100]*"GET / HTTP/1.0"*200*395
   dyn143.nl*[23/Jan/2003:13:34:11 +0100]*"GET /html/index.html HTTP/1.1"*404*287
   ```

2. Het *request*-veld bevat tussen dubbele aanhalingstekens zelf drie informatievelden die van elkaar worden gescheiden door een spatie: *i*) een `http`-methode, *ii*) het adres van wat precies werd aangevraagd en *iii*) de gebruikte versie van het `http`-protocol. Zorg er voor dat de dubbele aanhalingstekens rond dit veld verwijderd worden. Verwijder ook de gebruikte versie van het `http`-protocol en vervang de spatie tussen de `http`-methode en het adres door een asterisk (`*`). De twee regels uit het voorgaande voorbeeld moeten dus omgezet worden naar

   ```
   plexus.v2.nl*[22/Jan/2003:19:04:01 +0100]*GET*/*200*395
   dyn143.nl*[23/Jan/2003:13:34:11 +0100]*GET*/html/index.html*404*287
   ```

3. Verwijder de vierkante haakjes rond het *timestamp*-veld, laat de tijdzone weg en plaats een spatie tussen datum en tijdsaanduiding in plaats van een dubbelpunt (`:`). Wijzig ook de volgorde van de velden. Plaats het adres van de opgevraagde pagina vooraan, gevolgd door de `http`-methode, de statuscode, de grootte, het tijdstip en het adres van de aanvrager. Dit levert het volgende resultaat op

   ```
   /*GET*200*395*22/Jan/2003 19:04:01*plexus.v2.nl
   /html/index.html*GET*404*287*23/Jan/2003 13:34:11*dyn143.nl
   ```

4. Aanvragen die niet door mensen uitgevoerd werden, willen we liever niet in rekening brengen bij het analyseren van de historiek van de webserver. Verwijder daarom de regels waar `bot` of `crawl` voorkomt in het adres van de aanvrager.

5. Vervang (vanuit de teksteditor `vi` of `vim`) het logbestand door een overzicht van hoeveel keer elke pagina werd opgevraagd. Dit overzicht moet aflopend gesorteerd worden op het aantal aanvragen. Dit geeft onder meer volgende regels

   ```
   2679 /robots.txt
   2044 /logs/clients.log
   ```

Probeer voor elke opdracht zo weinig mogelijk commando's te gebruiken en zorg er voor dat elk van deze commando's bestaat uit zo weinig mogelijk tekens. Alle wijzigingen moeten na elkaar uitgevoerd worden.

Opgave 3

1. Geef LaTeX-code die een reconstructie maakt van onderstaande tabel, samen met de bijhorende tekst. Zorg er daarbij voor dat de opmaak zo getrouw mogelijk behouden blijft.

De tabel

links		**midden**	**rechts**	
1	2	3	2	1
4	5	6	5	4
7	8	9	8	7

namaken in LaTeX kan moeilijker zijn dan je denkt.

2. Geef LaTeX-code die precies hetzelfde resultaat oplevert als het tekstfragment in onderstaand kader. Maak hierbij omgevingen aan voor `Stelling` en `Lemma`. Zorg er voor dat formules en eigen omgevingen automatisch genummerd worden, en gebruik waar mogelijk verwijzingen naar deze nummeringen.

Lemma 1 *De formule*

$$\lim_{\alpha\to 0} \log_\alpha\left(1+\frac{(\alpha^x-1)(\alpha^y-1)}{\alpha-1}\right) = \min\{x,y\}$$

is geldig voor alle $(x,y)\in[0,1]^2$.

Stelling 2 *Voor alle* x *en* y *in* $[0,1]$ *geldt:*

$$\lim_{\alpha\to 0} \log_\alpha\left(1+\frac{(\alpha^x-1)(\alpha^y-1)}{\alpha-1}\right) = \begin{cases} x & \text{als } x\le y \\ y & \text{anders} \end{cases}$$

Stel dat $T_{\mathbf{M}}$ en $S_{\mathbf{M}}$ de $[0,1]^2\to[0,1]$ afbeeldingen zijn die gegeven worden door:

$$T_{\mathbf{M}}(x,y) = \min\{x,y\} \tag{1}$$
$$S_{\mathbf{M}}(x,y) = \max\{x,y\} \tag{2}$$

voor alle $(x,y)\in[0,1]^2$. De gelijkheid $S_{\mathbf{M}}(x,y) = 1-T_{\mathbf{M}}(1-x,1-y)$ geldt dan voor willekeurige x en y in $[0,1]$, vermits:

$$\begin{aligned} 1-T_{\mathbf{M}}(1-x,1-y) &= 1-\min\{1-x,1-y\} && \text{want (1)} \\ &= 1+\max\{-1+x,-1+y\} \\ &= \max\{1-1+x,1-1+y\} \\ &= \max\{x,y\} \\ &= S_{\mathbf{M}}(x,y) && \text{want (2)} \end{aligned}$$

Plaats een PDF bestand met daarin de gecompileerde LaTeX fragmenten in het ZIP-bestand dat je indient via Indianio.

Opgave 4

Gegeven zijn de bestanden `latex.txt`, `LATEX.txt`, `LATEX1.txt`, `LATEX2.txt`, `LATEX_c.txt`, `replace`, `alphabet.txt` en `chains.txt`.

1. (a) Implementeer de volgende (eenvoudige) commando's aan de hand van shell files:
   i. `./caps`
   Vervangt alle kleine letters die binnenkomen via standaard invoer door hoofdletters, en stuurt het resultaat naar standaard uitvoer.
   ii. `./drop <n>`
   Stuurt de standaard invoer naar standaard uitvoer vanaf de *<n>*-de regel. Met andere woorden, de eerste *<n>* − 1 regels worden weggelaten.
   iii. `./lines`
   Zet alle karakters die binnenkomen via standaard invoer op een aparte regel, en stuurt het resultaat naar standaard uitvoer.

   (b) Combineer deze commando's vervolgens tot (telkens) één samengesteld commando dat:
   i. het bestand `latex.txt` omzet naar `LATEX.txt`.
   ii. het bestand `LATEX1.txt` genereert, vertrekkende van `latex.txt`.
   iii. het bestand `LATEX2.txt` genereert, vertrekkende van `LATEX1.txt`.

2. (a) Gebruik opnieuw shell files om de volgende (meer complexe) commando's te implementeren:
   i. `./combine <bestand 1> <bestand 2> ... <bestand n>`
   Plakt de overeenkomstige regels van standaard invoer en de opgegeven bestanden aan elkaar, gescheiden door een TAB, en stuurt het resultaat naar standaard uitvoer.

   Voorbeeld: Stel dat de bestanden `b.txt` en `c.txt` beide slechts één regel bevatten, en dat die regels respectievelijk uit het karakter "b" en "c" bestaan (gevolgd door een newline). Het commando `./combine b.txt c.txt` zal dan de regel "b c" (en daarna een newline) uitschrijven. Als dit commando bovendien de regel "a" ontvangt via standaard invoer, dan wordt niet "b c" maar "a b c" uitgeschreven.
   ii. `./chains <bestand 1> <bestand 2> ... <bestand n>`
   Creëert karaktersequenties door de overeenkomstige regels van de opgegeven bestanden aan elkaar te plakken, zonder scheidingsteken, en bepaalt vervolgens welke sequenties er het meeste opduiken. De 26 meest voorkomende karaktersequenties worden naar standaard uitvoer geschreven. Dit gebeurt onder de vorm van een lijst van sequenties die geordend is volgens aantal voorkomens. Met andere woorden, als een sequentie a vóór een andere sequentie b staat in de teruggegeven lijst, dan wil dat zeggen dat a meer voorkomt dan b.

   Voorbeeld: Het bestand `chains.txt` bevat de uitvoer van het commando `./chains LATEX1.txt LATEX2.txt`.

   (b) De shell file `replace` krijg je cadeau. Deze file implementeert het commando `./replace <bestand>`, waarmee karaktersequenties in *<bestand>* kunnen vervangen worden. De regels die dit commando ontvangt via standaard invoer, moeten bestaan uit twee karaktersequenties gescheiden door een TAB. Elk voorkomen in *<bestand>* van de sequentie voor de TAB zal dan telkens vervangen worden door de sequentie erna. Het resultaat wordt uitgeschreven naar standaard uitvoer (en wordt dus niet opgeslagen in *<bestand>*).

   Combineer `./replace` nu met `./combine` en `./chains` tot (telkens) één samengesteld commando dat:
   i. Het gegeven bestand `LATEX_c.txt` genereert, vertrekkende van `LATEX.txt`, `LATEX1.txt`, `LATEX2.txt` en `alphabet.txt`. Dit houdt in dat de 26 meest voorkomende sequenties van twee karakters in `LATEX.txt` moeten vervangen worden door een kleine letter. De

sequentie die het meeste voorkomt wordt vervangen door “a”, de op één na meest voorkomende door “b”, etc.

ii. Het gegeven bestand `LATEX.txt` genereert, vertrekkende van `LATEX_c.txt`, `chains.txt` en `alphabet.txt`.

Merk op dat `LATEX_c.txt` en `chains.txt` samen 12% minder geheugenruimte innemen dan `LATEX.txt`, terwijl we `LATEX.txt` wel volledig kunnen reconstrueren uit `LATEX_c.txt` en `chains.txt` (wetende dat we `alphabet.txt` moeten gebruiken bij die reconstructie). Met andere woorden, de hierboven besproken commando’s implementeren een (zeer primitieve) vorm van compressie.

EXAMEN: **Computergebruik**

1e Bachelor Informatica
prof. dr. Peter Dawyndt
groep 2

maandag 09-01-2012, 14:00
academiejaar 2011-2012
eerste zittijd

Opgave 1

Brainfuck is een esoterische programmeertaal die in 1993 werd ontwikkeld door Urban Müller. Müllers doel was een Turing-complete programmeertaal te maken die met de kleinst mogelijke compiler zou kunnen geïmplementeerd worden, geïnspireerd op de 1024-byte compiler voor de programmeertaal `FALSE`. Er werden dan ook verschillende brainfuck compilers ontwikkeld die kleiner zijn dan 200 bytes. Zoals de naam reeds doet vermoeden zijn brainfuck programma's over het algemeen moeilijk te begrijpen. Daar staat tegenover dat elke Turingmachine, en daardoor ook brainfuck, elke rekentaak kan volbrengen.
De taal is gebaseerd op een zeer eenvoudige virtuele machine die — buiten het programma — bestaat uit een rij van bytes die op nul zijn gezet, een data-pointer die wijst naar een element van de rij (startwaarde van de data-pointer is de eerste byte) en twee rijen bytes die de invoer en uitvoer vormen. De acht statements waaruit de taal bestaat, en die elk worden voorgesteld door één enkel karakter, staan opgelijst in onderstaande tabel.

KARAKTER	BETEKENIS
`>`	verhoog data-pointer
`<`	verlaag data-pointer
`+`	verhoog byte waarnaar data-pointer wijst
`-`	verlaag byte waarnaar data-pointer wijst
`.`	geef byte waarnaar data-pointer wijst als ASCII-uitvoer
`,`	gebruik volgende ASCII-invoerbyte om byte waarnaar data-pointer wijst in te vullen
`[`	spring voorwaarts naar statement na corresponderende `]` indien byte waarnaar pointer wijst nul is
`]`	spring terug naar statement achter corresponderende `[` indien byte waarnaar pointer wijst geen nul is

Zoals dat gebruikelijk is bij haakjes, moeten ook in brainfuck de vierkante haakjes matchen. Onderstaande brainfuck programmacode genereert bijvoorbeeld de tekst `Hello World!`

```
++++++++++[>+++++++>++++++++++>+++>+<<<<-]>++.>+.+++++++..+++.>++.<<+++++++++++++++.>
.+++.------.--------.>+.>.
```

Op elke regel van het tekstbestand `brain.txt` staat een stukje brainfuck programmacode, gevolgd door een spatie en een woord dat enkel bestaat uit letters van het alfabet. Dit woord is de uitvoer die gegenereerd wordt door de voorafgaande brainfuck programmacode. Elk stukje brainfuck code bevat telkens één paar overeenkomstige vierkante haakjes. In onderstaande uitleg zullen we het codefragment tussen de vierkante haakjes benoemen als de *lus*, het codefragment voor het openende vierkante haakje als de *initialisatie* en het codefragment dat volgt op het afsluitende vierkante haakjes als de *uitvoer*. Gevraagd wordt:

1. Bepaal reguliere expressies voor elk van onderstaande verzamelingen, waarbij $\mathcal{B}$ de verzameling voorstelt van alle brainfuck programma's met één paar overeenkomstige vierkante haakjes. Probeer deze reguliere expressies bovendien zo kort mogelijk te houden.

(a) $\alpha = \{b \in \mathcal{B} \mid$ codefragment tussen eerste twee punten van uitvoer van b is minstens negen karakters lang, en komt ook voor in lus van $b\}$

voorbeelden: `++[>++>+++>++++++++++>+<<-]>++.>+++++++++.>+.---.>.` $\in \alpha$,
`++[>++>+++>++++>+++++>+<<-]>++.>+++++++++.>+.---.>.` $\notin \alpha$

(b) $\beta = \{b \in \mathcal{B} \mid$ uitvoer van b bevat puntloze reeks r van minstens twee karakters lang die minstens drie keer na elkaar herhaald wordt met tussenliggende punten$\}$

opmerking: puntloze reeks r is substring van uitvoer die zelf geen punten bevat, en die start aan het begin van de uitvoer of begint met een punt en stopt aan het einde van de uitvoer of eindigt met een punt (*longest match principe*)

voorbeeld: `++[>++>+++>+<<<-]>+++++++++.>+.<--..--.--.--.>>.` $\in \beta$ met $r =$ `--`

(c) $\gamma = \{b \in \mathcal{B} \mid$ elke reeks opeenvolgende `<`-tekens heeft steeds oneven lengte, en ook elke reeks opeenvolgende `>`-tekens heeft steeds oneven lengte$\}$

(d) $\delta = \{b \in \mathcal{B} \mid$ elk zevende symbool van b is een plusteken$\}$

voorbeeld: `+++++++++[>++++++++>+++++>+<<<-]>+++.>++++++.<-.+.>>.` $\in \delta$

Gebruik een commando uit de `grep` familie om enkel die regels van het bestand `brain.txt` te selecteren met fragmenten brainfuck programmacode die behoren tot de opgegeven verzameling. Vermeld in je antwoordbestand voor elke verzameling het gebruikte selectiecommando, en geef telkens ook aan hoeveel regels je gevonden hebt.

2. Beschouw de verzamelingen α, β, γ en δ zoals hierboven gedefinieerd. Gebruik nu deze verzamelingen om op de volgende manier een boodschap bestaande uit vier woorden te achterhalen:

   (a) het eerste woord staat op de unieke regel met programmacode uit de verzameling $\alpha \cap \beta$

   (b) het tweede woord staat op de unieke regel met programmacode uit de verzameling $\beta \cap \gamma$

   (c) het derde woord staat op de unieke regel met programmacode uit de verzameling $\gamma \cap \delta$

   (d) het vierde woord staat op de unieke regel met programmacode uit de verzameling $\delta \cap \alpha$

   Vermeld in je antwoordbestand de gevonden woorden, samen met het unix commando (of de commandosequentie) dat je gebruikt hebt om elk van deze woorden te vinden. Elk commando of elke commandosequentie moet dus als resultaat één van de gezochte woorden naar standaard uitvoer schrijven (zonder de brainfuck programmacode die aan het woord voorafgaat).

Opgave 2

1. Geef LaTeX-code die een reconstructie maakt van onderstaande tabel, samen met de bijhorende tekst. Zorg er daarbij voor dat de opmaak zo getrouw mogelijk behouden blijft.

De tabel

Product	1	2	3	4	5
Prijs	€124	€136	€85	€156	€23
Garantie {jaar}	1	2	-	3	1
Score	89%	84%	51%		45%
Aanbevolen	ja	ja	neen	neen	neen

bevat een vergelijking van de verschillende types RAM geheugen.

2. Geef LaTeX-code die precies hetzelfde resultaat oplevert als het tekstfragment in onderstaand kader. Maak hierbij omgevingen aan voor `Definitie` en `Recursieve definitie`. Zorg er voor dat formules en eigen omgevingen automatisch genummerd worden, en gebruik waar mogelijk verwijzingen naar deze nummeringen.

Definitie 1 *Als $z, f(z) \in \mathbb{R}^m$, dan is de M-de Frechet afgeleide van f, voorgesteld door $f^{(M)}(z)$, een operator op $\mathbb{R}^m \times \mathbb{R}^m \times \ldots \times \mathbb{R}^m$ (M maal), die lineair is in elk van de operandi, met als waarde*

$$f^{(M)}(z)\underbrace{(K_1, K_2, \ldots, K_M)}_{operandi} = \sum_{i=1}^{m}\sum_{j_1=1}^{m}\sum_{j_2=1}^{m}\cdots\sum_{j_M=1}^{m} {}^{i}f_{j_1 j_2 \ldots j_M}\, {}^{j_1}K_1\, {}^{j_2}K_2 \ldots {}^{j_M}K_M e_i, \qquad (1)$$

waarbij z het argument is en

$$K_t = [{}^{1}K_t, {}^{2}K_t, \ldots, {}^{m}K_t]^T \in \mathbb{R}^m, \qquad t = 1, 2, \ldots, M\,,$$
$${}^{i}f_{j_1 j_2 \ldots j_M} = \frac{\partial^M\, {}^{i}f(z)}{\partial\, {}^{j1}z \partial\, {}^{j2}z \ldots \partial\, {}^{jM}z}\,,$$
$$e_i = [0, 0, \ldots, 0, 1, 0, \ldots, 0]^T \in \mathbb{R}^m\,,$$

met 1 op de i-de plaats in e_i.

Recursieve definitie 2 *De elementaire differentialen $F_s : \mathbb{R}^m \to \mathbb{R}^m$ van f en hun orde worden recursief gedefinieerd als:*

(a) f is de enige elementaire differentiaal van de orde 1, en

(b) als F_s ($s = 1, 2, \ldots, M$) respectievelijk elementaire differentialen zijn van de orde r_s, dan is de Frechet afgeleide

$$f^{(M)}(F_1, F_2, \ldots, F_M) \qquad (2)$$

een elementaire differentiaal van de orde

$$1 + \sum_{s=1}^{M} r_s\,. \qquad (3)$$

Plaats een PDF bestand met daarin de gecompileerde LaTeX fragmenten in het ZIP-bestand dat je indient via Indianio.

Opgave 3

Gegeven is een tekstbestand `kijkcijfers.txt` waarin de 100 meest bekeken tv-programma's van 2010 staan opgelijst. Van elk programma worden de volgende gegevens op afzonderlijke regels opgelijst: rangnummer, naam, genre, zender, datum, start, lengte, percentage kijkende vlamingen, aantal kijkers (in duizendtallen telkens tot 2 cijfers na de komma), en percentage marktaandeel. Gevraagd wordt om, gebruik makend van de teksteditor `vi` (of `vim`), een reeks (substitutie)commando's op te stellen die achtereenvolgens de volgende opdrachten uitvoeren:

1. Zorg er voor dat de verschillende informatievelden van hetzelfde programma op één enkele regel gezet worden, van elkaar gescheiden door een puntkomma (`;`). Zo moeten de regels

```
3
WITSE
Fiktie/Series
```

```
EEN
05/12/2010
21:42:09
00:55:56
33,2
1.942,48
66,2
14
SPORTWEEKEND
Sport/Sport-Magazines
EEN
17/01/2010
19:22:21
00:33:39
22,7
1.325,79
52,8
```

omgezet worden naar

```
3;WITSE;Fiktie/Series;EEN;05/12/2010;21:42:09;00:55:56;33,2;1.942,48;66,2
14;SPORTWEEKEND;Sport/Sport-Magazines;EEN;17/01/2010;19:22:21;00:33:39;22,7;1.325,79;52,8
```

2. Het genre bevat een hoofdcategorie, gevolgd door een slash (`/`) en een subcategorie. Verwijder de subcategorie, inclusief de slash. Voeg ook een procentteken (`%`) toe op het einde van de twee velden die een percentage uitdrukken. De twee regels uit het voorgaande voorbeeld moeten dus omgezet worden naar

   ```
   3;WITSE;Fiktie;EEN;05/12/2010;21:42:09;00:55:56;33,2%;1.942,48;66,2%
   14;SPORTWEEKEND;Sport;EEN;17/01/2010;19:22:21;00:33:39;22,7%;1.325,79;52,8%
   ```

   Opmerking: indien je er (nog) niet in geslaagd bent om de eerste opdracht uit te voeren, dan kan je voor deze tweede opdracht verder werken op het bestand `kijkcijfers2.txt` dat resulteert uit de eerste opdracht.

3. Het aantal kijkers wordt uitgedrukt in duizendtallen. Pas dit aan zodat dit gewoon wordt uitgedrukt in aantal kijkers, en plaats dit veld als tweede in de lijst van informatievelden. Let op het gebruik van punten (`.`) als scheidingsteken tussen de duizendtallen. Toegepast op het vorige voorbeeld wordt dit

   ```
   3;1.942.480;WITSE;Fiktie;EEN;05/12/2010;21:42:09;00:55:56;33,2%;66,2%
   14;1.325.790;SPORTWEEKEND;Sport;EEN;17/01/2010;19:22:21;00:33:39;22,7%;52,8%
   ```

4. De lengte van een programma wordt weergegeven in een formaat waarbij uren, minuten en seconden telkens voorgesteld worden door 2 cijfers, die van elkaar worden gescheiden door een dubbelpunt (`:`). Pas dit formaat aan zodat na het aantal uren een `u` komt te staan, na het aantal minuten een `m` en na het aantal seconden een `s`. De dubbelpunten worden weggelaten. Het aantal uren moet weggelaten worden als een programma minder dan een uur duurt. Bij getallen kleiner dan 10 moet de voorste 0 weggelaten worden. `00:37:04` wordt bijvoorbeeld omgezet in `37m4s`. Dit geeft

   ```
   3;1.942.480;WITSE;Fiktie;EEN;05/12/2010;21:42:09;55m56s;33,2%;66,2%
   14;1.325.790;SPORTWEEKEND;Sport;EEN;17/01/2010;19:22:21;33m39s;22,7%;52,8%
   ```

5. Zorg er voor dat de naam van een programma begint met een hoofdletter, gevolgd door kleine

letters. Toegepast op het vorige voorbeeld wordt dit

```
3;1.942.480;Witse;Fiktie;EEN;05/12/2010;21:42:09;55m56s;33,2%;66,2%
14;1.325.790;Sportweekend;Sport;EEN;17/01/2010;19:22:21;33m39s;22,7%;52,8%
```

Probeer voor elke opdracht zo weinig mogelijk commando's te gebruiken en zorg er voor dat elk van deze commando's bestaat uit zo weinig mogelijk tekens. Alle wijzigingen moeten na elkaar uitgevoerd worden.

Opgave 4

Gegeven zijn de bestanden `TEX.txt`, `tex.txt`, `tex1.txt`, `tex2.txt`, `tex_c.txt`, `replace`, `ALPHABET.txt` en `sequences.txt`.

1. (a) Implementeer de volgende (eenvoudige) commando's aan de hand van shell files:
   i. `./part <n>`
   Stuurt de standaard invoer naar standaard uitvoer vanaf de `<n>`-de regel. Met andere woorden, de eerste `<n>` -1 regels worden weggelaten.
   ii. `./tolines`
   Zet alle karakters die binnenkomen via standaard invoer op een aparte regel, en stuurt het resultaat naar standaard uitvoer.
   iii. `./tolower`
   Vervangt alle hoofdletters die binnenkomen via standaard invoer door kleine letters, en stuurt het resultaat naar standaard uitvoer.

   (b) Combineer deze commando's vervolgens tot (telkens) één samengesteld commando dat:
   i. het bestand `TEX.txt` omzet naar `tex.txt`.
   ii. het bestand `tex1.txt` genereert, vertrekkende van `TEX.txt`.
   iii. het bestand `tex2.txt` genereert, vertrekkende van `tex1.txt`.

2. (a) Gebruik opnieuw shell files om de volgende (meer complexe) commando's te implementeren:
   i. `./fuse <bestand 1> <bestand 2> ... <bestand n>`
   Plakt de overeenkomstige regels van de opgegeven bestanden en van standaard invoer aan elkaar, gescheiden door een TAB, en stuurt het resultaat naar standaard uitvoer.

   Voorbeeld: Stel dat de bestanden `1.txt` en `2.txt` beide slechts één regel bevatten, en dat die regels respectievelijk uit het karakter "1" en "2" bestaan (gevolgd door een newline). Het commando `./fuse 1.txt 2.txt` zal dan de regel "1 2" (en daarna een newline) uitschrijven. Als dit commando bovendien de regel "3" ontvangt via standaard invoer, dan wordt niet "1 2" maar "1 2 3" uitgeschreven.
   ii. `./sequences <bestand 1> <bestand 2> ... <bestand n>`
   Creëert karaktersequenties door de overeenkomstige regels van de opgegeven bestanden aan elkaar te plakken, zonder scheidingsteken, en bepaalt vervolgens welke sequenties er het meeste opduiken. De 26 meest voorkomende karaktersequenties worden naar standaard uitvoer geschreven. Dit gebeurt onder de vorm van een lijst van sequenties die geordend is volgens aantal voorkomens. Met andere woorden, als een sequentie a vóór een andere sequentie b staat in de teruggegeven lijst, dan wil dat zeggen dat a meer voorkomt dan b.

   Voorbeeld: Het gegeven bestand `sequences.txt` bevat de uitvoer van het commando `./sequences tex1.txt tex2.txt`.

(b) De shell file `replace` krijg je cadeau. Deze file implementeert het commando `./replace <bestand>`, waarmee karaktersequenties in *<bestand>* kunnen vervangen worden. De regels die dit commando ontvangt via standaard invoer, moeten bestaan uit twee karaktersequenties gescheiden door een TAB. Elk voorkomen in *<bestand>* van de sequentie voor de TAB zal dan telkens vervangen worden door de sequentie erna. Het resultaat wordt uitgeschreven naar standaard uitvoer (en wordt dus niet opgeslagen in *<bestand>*).

Combineer `./replace` nu met `./fuse` en `./sequences` tot (telkens) één samengesteld commando dat:

i. Het gegeven bestand `tex_c.txt` genereert, vertrekkende van `tex.txt`, `sequences.txt` en `ALPHABET.txt`. Dit houdt in dat de 26 meest voorkomende sequenties van twee karakters in `tex.txt` moeten vervangen worden door een hoofdletter. De sequentie die het meeste voorkomt wordt vervangen door "A", de op één na meest voorkomende door "B", etc.

ii. Het bestand `tex.txt` genereert, vertrekkende van `tex_c.txt`, `tex1.txt`, `tex2.txt` en `ALPHABET.txt`.

Merk op dat `tex_c.txt` en `sequences.txt` samen 15% minder geheugenruimte innemen dan `tex.txt`, terwijl we `tex.txt` wel volledig kunnen reconstrueren uit `tex_c.txt` en `sequences.txt` (wetende dat we `ALPHABET.txt` moeten gebruiken bij die reconstructie). Met andere woorden, de hierboven besproken commando's implementeren een (zeer primitieve) vorm van compressie.