UIA IS-105 UTKAST ICA03
Sist oppdater: 16. Februar 2017

ICA03

Besvarelsen skal evalueres med karakter (A-F). Begrunn alle svar!

Formål

Hva er nyttig å forstå om temaet?

- ❏ Kjennskap til en forenklet historisk oversikt av koding av tekst / symboler (typografiske tegn) i databehandlingsssystemer (fra hullkort til UNICODE).
- ❏ Forståelse av hvordan tekst-strenger blir representert i datasystemer.
- ❏ Forståelse for tvetydighet (duplikater, for eksempel) i standarder som UNICODE.
- ❏ Forståelse for at programmerer / systemutvikler har ansvar for å tolke koding korrekt (tekst uten kjennskap til koding er vanligvis ikke brukbar).
- ❏ Grunnleggende forståelse for Golang's design i forhold til behandling av tekst og symboler (strenger er vilkårlige sekvenser av bytes)
- ❏ Å kunne anvende spesifikt programmeringsmiljø for å kontrollere og manipulere multi-språklig tekst (konkret i dette tilfelle Golang)

Bestemme akseptabel bevis? Hva som kan telle som bevis at studentene faktisk forstår det som har vært vår intensjon at de skal forstå?

- ❏ Vise at man har forståelse for den historiske variasjonen av koding av typografiske tegn i datasystemer, ved å kunne analysere sekvenser av bytes
- ❏ Vise at man kan løse et praktisk problem ved bruk av kunnskap om koding av tekst og symboler i Golang.

Gjelder alle programmeringsoppgaver:

- ❏ Vise at man kan skrive oversiktlig kode og forklare kode med egne ord.
- ❏ Vise at man kan skrive tester for de fleste funksjonene man skriver.
- ❏ Vise at man kan kompilere kode for og utføre den på forskjellige platformer.

I denne oppgaven skal dere skrive og utføre en del kode i golang. Eksemplerfiler, som dere kan ta som utgangspunkt finnes i denne repository-en (referert videre som **REP_ICA03**):

https://github.com/uia-worker/is105-ica03

Oppgave 1

Formål:

- bli kjent med ASCII tabellen og kunne manipulere disse i et programmeringsmiljø

Hva trenger du å vite for å klare denne oppgaven:

- du må kunne å overføre filer fra din lokale datamaskin (node) til nettskyen og omvendt (alle bør nå ha en nettsky-instans som er korrekt konfigurert)
- du må forstå det grunnleggende om heksadesimalt, binært og desimalt tallsystem og overganger mellom disse
- du må forstå "slices" i golang (kun grunnleggende, dvs. hvordan deklarere dem og bruke dem, foreløpig ikke hvordan de er implementert); spesielt typen []byte
- du trenger å forstå tekststrenger "strings" ("string literals")
- du trenger å forstå at alle tekststrenger og symboler, som blir presentert i applikasjonene/programmene, består i utgangspunktet (i en lagringsenhet) av en sammenhengende sekvens av bytes, og at de som har implementert (skrevet kode for) applikasjonene/programmene, bestemmer hvordan disse sekvensene av bytes skal tolkes

Programmeringsoppgaver:

a)

- bruk filen `ascii.go` i REP_ICA03 som utgangspunkt, og skriv ut en tabell av alle tegn i "string literal" ascii (deklarert i filen ascii.go)

  Kravspesifikasjon

  Funksjonsnavn skal være `iterateOverASCIIStringLiteral(...)` og den skal ta et argument som skal være av type `string` ("string literal"). Funksjonen trenger ikke å returnere noe eksplisitt.

  Utskriftsformatet skal være følgende:

  ```
  [ascii-kode heksadesimalt med store bokstaver A-F][mellomrom][symbol for ascii-kode][mellomrom][ascii-kode binært][linjeskift]
  ```

  Eksempel

  ```
  ...
  3E > 111110
  3F ? 111111
  40 @ 1000000
  41 A 1000001
  ```

```
42 B 1000010
43 C 1000011
...
```

- analyser utskriften (spesielt for bytes fra 0x00 til 0x0F)
- utfør programmet på en instans i nettskyen (UH-IaaS) og analyser resultatet

b)

- lag en funksjon `greetingASCII()` i samme filen `ascii.go`, som skriver ut `"Hello :-)"`

  Kravspesifikasjon

  Funksjonen skal generere en utskrift fra en sekvens av bytes, dvs. av typen `[]bytes` (det betyr at du må finne den heksadesimale eller binære representasjonen av alle tegn i strengen, som skal skrives ut (inkludert anførselstegn eller "double quotes" på engelsk).

  Funksjonen `greetingASCII()` skal returnere en variabel av typen `string`, som inneholder kun ASCII-tegn (ikke utvidet ASCII).

- utfør programmet på en instans i nettskyen (UH-IaaS) og analyser resultatet

c)

Implementer en test for funksjonen `greetingASCII()` i egen fil `ascii_test.go`, som tester om returverdier (av type `string`) inneholder kun ASCII-tegn.

Teamarbeid (gjelder også for resten av oppgavene):

Oppsummer oppgaven i et dokument og illustrer hvordan programmene utfører på deres systemer. Hvis dere har tilgang på både Mac OS X og MS Windows systemer, utfør alle programmene på disse systemene og sammenligne resulater. Hvis dere ikke har tilgang på begge de mest utbredte systemene, sammenligne resultatet på instansen i nettskyen og deres eget system.

Legg inn koden i felles repository. Skriv gjerne kode to og to sammen. Den ene skriver og kommenterer hvordan man tenker og den andre observerer og kommer med innspill. Flere par i gruppen kan jobbe uavhengig, for så å møtes og sammenligne fremgangsmåter og resultater.

Evalueringskriterier (gjelder også for resten av oppgavene):

Tankegangen for implementasjon og analyse (beskrivelse) av resultater teller 30%. God og oversiktlig kode teller 40%. At alle i teamet har bidratt må fremkomme fra besvarelsen og “commits” i github repository (teller 30%).

Referanser som kan hjelpe i arbeidet med oppgaven:

- http://www.ascii-code.com/ (utvidet ASCII tabell, ISO 8859-1 og Microsoft® Windows Latin-1)

Oppgave 2

Formål:

- Bli kjent med ISO/IEC 8859 serier for 8-bits koding av typografiske symboler.
- Illustrere forskjell på ASCII og utvidet ASCII kode gjennom golang rammeverk for behandling av tekststrenger (på engelsk brukes det nesten alltid begrepet "strings" for tekststrenger).

Hva trenger du å vite for å klare denne oppgaven:

- du må forstå "slices" i golang
- at de første 256 "code points" (kodepunkter, som tilsvarer typen "rune"[1] i golang, varierer fra platform til platform og fra program til program; det finnes 15 forskjellige deler av ISO/IEC 8859 serier for 8-bits koder)

Programmeringsoppgave:

a)

- samme som i oppgave 1a, bare at funksjonen itererer (går i en løkke over) over tegn med byte-verdier fra 0x80 til 0xFF, dvs. det utvidede ASCII settet; funksjonen skal implementeres i filen `iso.go`, som dere finner i mappen `iso` i REP_ICA03

  Kravspesifikasjon

  Funksjonsnavn skal være `iterateOverExtendedASCIIStringLiteral(...)` og den skal ta et argument, som skal være av type `string` ("string literal"). Dere må generere / deklarere en `string` med alle de 128 heksadesimale verdiene (`"\x80\x81...\xFF"`) som funksjonen kan ta som argument. Funksjonen trenger ikke å returnere noe eksplisitt.

  Utskriftsformatet skal være følgende:

  ```
  [utvidet-ascii-kode heksadesimalt med store bokstaver
  A-F][mellomrom][symbol for utvidet-ascii-kode]
  [mellomrom][utvidet-ascii-kode binært][linjeskift]
  ```

  Eksempel

  ```
  ...
  3E > 111110
  BD ½ 10111101
  ```

[1] inspirasjon er nok fra det eldste skriftspråket (https://en.wikipedia.org/wiki/Runes), men type i golang er laget for å unngå problemer med begrepet "tegn" (character), som i databehandlingen er fortsett sterkt bundet til en byte (8 bits, dvs. muligheter for å presenterer 256 tegn); en variabel av type rune er 32 bits (4 bytes) og er en alias for int32 type i golang; grunnen er at UNICODE klassifiseringen av tegn og symbolder bruker 32 bits verdier som "code points" (se https://unicode-table.com/en/)

```
BE ¾ 10111110
BF ¿ 10111111
C0 À 11000000
C1 Á 11000001
...
```

- analyser utskriften (spesielt for bytes fra 0x80 til 0x9F)
- utfør programmet på en instans i nettskyen (UH-IaaS) og analyser resultatet

b)

- lag en funksjon `greetingExtendedASCII()` i samme filen `iso.go`, som skriver ut `"Salut, ça va °-) €50"`

  Kravspesifikasjon

  Funksjonen skal generere en utskrift fra en sekvens av bytes, dvs. av typen `[]bytes` (det betyr at du må finne den heksadesimale eller binære representasjonen av alle tegn i strengen, som skal skrives ut (inkludert anførselstegn eller "double quotes" på engelsk).

  Funksjonen `greetingExtendedASCII()` skal returnere en variabel av typen `string`, som inneholder tegn fra Extended ASCII.

- utfør programmet på 3 platformer, - Mac OS X, MS Windows og Linux/Unix og analyser resultater

c)

Implementer en test for funksjonen `greetingExtendedASCII()` i egen fil `iso_test.go`, som tester om returverdier (av type `string`) inneholder kun tegn fra en Extended ASCII.

Referanser:

- https://unicode-table.com/en/
- https://en.wikipedia.org/wiki/Unicode (første 256 kodepunkter ("code points") er fra ISO-8859-1)
- https://en.wikipedia.org/wiki/ISO/IEC_8859-1
- https://en.wikipedia.org/wiki/ISO/IEC_8859
- https://en.wikipedia.org/wiki/UTF-8

Oppgave 3

Formål:

- forsterke kunnskapen om ISO/IEC 8859 serier for 8-bits koding
- bli kjent med UTF-8 koding, som lar oss kode teksten med tegn og symboler utøver de 256 plassene som vi har utforsket til nå og som er den mest brukte metoden for koding av tekst i datasystemer per i dag (2017)

Hva trenger du å vite for å klare denne oppgaven:

- hvor du finner tabellene med alle kodene
- historien og prinsippene bak kodetabellene
- golang's måte å håndtere koding av tekst på

Programmeringsoppgaver:

a)

- gå gjennom instruksjoner og løs gåten til slutt (gjelder konvertering til UTF-8)

  Instruksjoner

  Når man har tatt innholdet i en fil inn i datastrukturen, kan man analysere det vha. funksjoner i "fmt"-pakken. Her er noen nyttige funksjoner med et par eksempler, som man kan bruke

  %s ikke interpreterte bytes av en streng ("string") eller en skive ("slice")

  ```
  fmt.Printf("%s\n", "\xbd\xb2\x3d\xbc\x20\xe2\x8c\x98")
  Resultatet er: "??=? ⌘"
  ```

  %q representerer en tekstreng angitt med dobble anførselstegn og "flykter" (bruker ordet "escape" på engelsk når man skal fortelle parseren at et tegn skal ikke tolkes på en pre-definert måte og erstatte disse med Go's syntaks for byte) alle tegn som ikke er skrivbare; for eksempel, samsvarer ikke bytesekvensen (BD, B2 og BC er problemet) med UTF-8 kode, som Go konsekvent bruker

  ```
  fmt.Printf("%q\n", "\xbd\xb2\x3d\xbc\x20\xe2\x8c\x98")
  Resultatet er: "\xbd\xb2=\xbc ⌘"
  ```

  %+q Dette flag-et generer en representasjon hvor det "flyktet fra" både ikke-utskriftsbare tegn men også ikke-ASCII bytes som er korrekte UTF-8 tegn; resultatet er at den viser Unicode verdier ifølge UTF-8 koding for tegn som er utenfor ASCII kodesettet:

  ```
  fmt.Printf("%+q\n", "\xbd\xb2\x3d\xbc\x20\xe2\x8c\x98")
  "\xbd\xb2=\xbc \u2318"
  ```

`%X` base 16 (heksadesimalt), storebokstaver, to tegn (chars) per byte
`% X` skriver ut med et mellomrom mellom hver byte "EF DA A3 ..."
`%c` skriver ut en karakter av gangen og prøver å tolke det så godt som mulig innenfor utvidet ASCII kodesett

```
fmt.Printf("%c\n", "\xbd\xb2\x3d\xbc\x20\xe2\x8c\x98")
Resultatet er: "[½ ² = ¼  â  ]"
```

Forklar de store forskjellene i resultater av utskrift i eksemplene ovenfor (`%s, %q, %+q` og `%c`). Hva må man endre i bytesekvensen for at

```
fmt.Printf("%s", <bytesekvens>)
returnerer "½?=? ⌘"
```

Forklar endringen/konverteringen detaljert (følg eksempel i **Tillegg A** i dette dokumentet).

b)

Bruke erfaringene fra eksemplet for å analysere de tre filene i mappen `files`: lang01.wl, lang02.wl, lang03.wl

Bruk funksjonen fra REP_ICA03 `func FileToByteslice(filename string) []byte` som ligger i pakken `fileutils`. Filnavn kan være enten relativ eller absolutt[2]. Når utført, lagrer funksjonen innholdet fra en fil, byte for byte[3], i det virtuelle minne (RAM + "swap space") vha. go's datastruktur "slice" ("array", matrise o.l.). Det gir muligheter til å manipulere denne datastrukturen, mens programmet kjører. Man kan, for eksempel, søke etter spesifikke bytes og eventuelt erstatte disse med nye bytes, før man eventuelt skriver innholdet i datastrukturen ut til terminalen (stdout stream) eller til en fil (stream til en fil).

Forklar resultater basert på de forskjellige utskrift dere genererer for alle de tre filene. Illustrer med skjermbilder.
Kan man si, basert på ut-data fra funksjonen `FileToByteslice(filename string) []byte` , hvilket språk hver av disse filene er en kode for?
Er det andre måter enn programmeringsmiljøet i golang, som man kan bruke, for å finne ut hvilken kode som er brukt når man har lagret filene?

[2] fil-stien er relativ i forhold til mappen man utfører main.go fra, f.eks., hvis main.go er i C:\Documents\IS105\is105-ica03\main.go og filen, som man ønsker å jobbe med er i C:\Documents/IS105/is105-ica03/files/lang01.wl, kan man deklarere filnavn som "files/lang01.wl", istedenfor absolutt, som er "C:\Documents\IS105\is105-ica03\files\lang01.wl" (i MS Windows brukes det **'\'** "backslash" (reverse solidus U+005C) for deling mellom mappene; i Mac OS X og Linux/Unix brukes det **'/'** "slash and divide" (solidus U+002f) "/home/users/..." for deling mellom mappene;)
[3] 8 bits, for eksempel, en byte binært "1010 1011", eller heksadesimalt "\xAB", eller 0xAB; byte er den minste enheten de fleste programmeringsmiljøer operer på

De første 16 bytes fra hver av de 3 filene ser slikt ut (skrevet ut med "% X" formatet):

"EF DA A3 D2 D3 CB 0A EF DA A3 D2 D3 CB C1 0A EF ..."
"FE FD 73 6B 61 72 0A FE FD 73 6B 61 72 61 6E 61 ..."
"F8 79 65 73 70 65 73 69 61 6C 69 73 74 65 6E 0A ..."

Hvilke tegn/symboler i Unicode tilsvarer disse byte-ene? Forklar og illustrer.

c)

- tenk at dere har funnet et ark fra fortiden med en kodet melding på; dere må finne ut hvilken tekst, som skjuler seg bak denne bytesekvensen:

```
48 65 6E 72 69 6B 20 41 72 6E 6F 6C 64 20 57 65 72 67 65 6C 61
6E 64 20 28 66 F8 64 74 20 31 37 2E 20 6A 75 6E 69 20 31 38 30
38 2C 20 64 F8 64 20 31 32 2E 20 6A 75 6C 69 20 31 38 34 35 29
0A 56 69 20 65 72 65 20 65 6E 20 6E 61 73 6A 6F 6E 20 76 69 20
6D 65 64 2C 0A 20 76 69 20 73 6D E5 20 65 6E 20 61 6C 65 6E 20
6C 61 6E 67 65 2C 0A 20 65 74 20 66 65 64 72 65 6C 61 6E 64 20
76 69 20 66 72 79 64 65 73 20 76 65 64 2C 0A 20 6F 67 20 76 69
2C 20 76 69 20 65 72 65 20 6D 61 6E 67 65 2E 0A 20 56 E5 72 74
20 68 6A 65 72 74 65 20 76 65 74 2C 20 76 E5 72 74 20 F8 79 65
20 73 65 72 0A 20 68 76 6F 72 20 67 6F 64 74 20 6F 67 20 76 61
6B 6B 65 72 74 20 4E 6F 72 67 65 20 65 72 2C 0A 20 76 E5 72 20
74 75 6E 67 65 20 6B 61 6E 20 65 6E 20 73 61 6E 67 20 62 6C 61
6E 74 20 66 6C 65 72 0A 20 61 76 20 4E 6F 72 67 65 73 20 E6 72
65 73 2D 73 61 6E 67 65 2E 0A 0A 20 4D 65 72 20 67 72 F8 6E 74
20 65 72 20 67 72 65 73 73 65 74 20 69 6E 67 65 6E 73 74 65 64
73 2C 0A 20 6D 65 72 20 66 75 6C 6C 74 20 61 76 20 62 6C 6F 6D
73 74 65 72 20 76 65 76 65 74 0A 20 65 6E 6E 20 69 20 64 65 74
20 6C 61 6E 64 20 68 76 6F 72 20 6A 65 67 20 74 69 6C 66 72 65
64 73 0A 20 6D 65 64 20 66 61 72 20 6F 67 20 6D 6F 72 20 68 61
72 20 6C 65 76 65 74 2E 0A 20 4A 65 67 20 76 69 6C 20 64 65 74
20 65 6C 73 6B 65 20 74 69 6C 20 6D 69 6E 20 64 F8 64 2C 0A 20
65 69 20 62 79 74 74 65 20 64 65 74 20 68 76 6F 72 20 6A 65 67
20 65 72 20 66 F8 64 64 2C 0A 20 6F 6D 20 6D 61 6E 20 65 74 20
70 61 72 61 64 69 73 20 6D 65 67 20 62 F8 64 0A 20 61 76 20 70
61 6C 6D 65 72 20 6F 76 65 72 73 76 65 76 65 74 2E 0A
```

I filen treasure.txt, som ligger i mappen treasure, er denne sekvensen lagret som tekst, som dere kan kopiere og bruke i deres analyser.

Skriv en funksjon `PrintTreasureUTF8(treasure_string string) []byte {}` som returnerer den teksten som skjuler seg bak koden kodet som UTF-8 (det betyr at den teksten er ikke kodet som UTF-8!). For å få det til, må dere først finne ut hvilket kodesett (eller flere) er brukt for koding av teksten og konvertere teksten til UTF-8. Dere kan bruke den innebygde pakken bytes og studere funksjonen

```
func Replace(s, old, new []byte, n int) []byte
```

Denne funksjonen er beskrevet her https://golang.org/pkg/bytes/#Replace
Her er et eksempel hvor man erstetter en byte med to nye byte-s:

```
bytes.Replace(byteslice, []byte("\xe5"), []byte("\xc3\xa5"), -1)
```

Referanser som kan hjelpe til å løse oppgaven:

- https://en.wikipedia.org/wiki/KOI_character_encodings
- https://golang.org/pkg/bytes/#Replace

Oppgave 4

Formål:

- Å bli bedre kjent med Unicode tabellen.

Hva trenger du å vite for å klare denne oppgaven:

- Hvordan men representerer Unicode karakterer i golang

Programmeringsoppgaver:

a)

- Hvilken koding må man bruke og hva er bytesekvensen (heksadesimalt) for følgende strenger (inkludert høyre og venstre anførselstegn[4])?

  `“nord og sør” på islandsk er “norður og suður”`
  `“nord og sør” på japansk er “北と南”`

  Se i mappen unicode og filen unicode.go. Skriv en funksjon

  `Translate(expression string, language string) string {}`

  som tar den norske strengen som in-data sammen med en streng for språk (“jp” og “is” i dette tilfelle) og returnerer strengen for det forespurte språket som ut-data. Strengen skal ikke kopieres inn i filen, men representeres med en hexadesimal streng med “flykt” (escapes, som “\x03\x89 ...”).

b)

- analyser “rune” (symbolet med kodepunkt) U+23F0 og vis den i nettleseren vha. eksempelkode i server.go (ligger i hovedmappen).

  Du kan starte webserver fra et terminalvindu med kommandoet:

  `go run server.go`

  Du kan aksessere webserver fra din nettleser med URL:

  `http://localhost:3000`

  For å avslutte prosessen for webserver kan du bruke `Ctrl-C` tastekombinasjon (fungerer i de fleste shell).

  Prøv å bruke pakken time, for å skrive ut dato og tid til websiden.

  Lykke til!

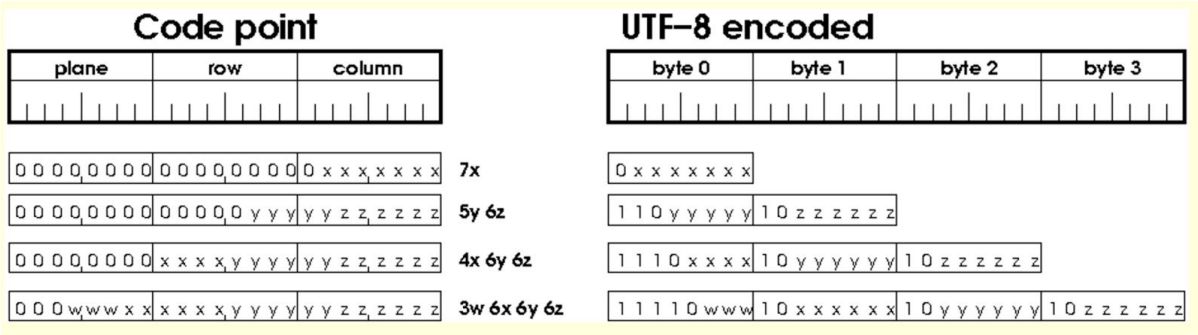
Referanser:

- http://www.alexedwards.net/blog/golang-response-snippets
- https://unicode-table.com/en/

[4] https://no.wikipedia.org/wiki/Sekund_(symbol)

Tillegg A

Tabellen viser hvordan omregning fra Unicode's "code point" (golang's rune) til UTF-8 foregår.



Unicode rom (space) er per i dag definert fra U+0000 til U+10FFFF (hele rommet er 4 bytes = 32 bits).
U+0000 - U+FFFF er "plane 0"
U+10000 - U+1FFFF er "plane 1" osv.
"planes" 1 - 16 (desimalt) er tilleggsrom (supplementary planes)
U+100000 - U+10FFFF er siste (17-de) "plane"

Eksempel:
Den norske bokstaven "å" har kode `E5` i ISO 8859-1/9/10/13/14/15 og `00E5` (skrives vanligvis U+00E5) i Unicode "plane 0" (https://unicode-table.com/en/00E5/). Den mest signifikante byte-n i kode `00E5` er `00` og kalles for "row". Den andre byte-n er `E5` og kalles for "column". En omrenging basert på bilde ovenfor foregår på følgende måte:

```
plane = 0x00 = 0b00000000
row = 0x00 = 0b00000000
column = 0xE5 = 0b11100101
-plane - -row --- -column-     -byte0-- -byte1--    -\x--
00000000 00000000 11100101 =>  11000011 10100101 => C3 A5
             yyy yyzzzzzz        yyyyy   zzzzzz
```

```
De heksadesimale tegn har følgende betydning (formatet er hex -
decimalt - binært):
A - 10 - 1010   D - 13 - 1101
B - 11 - 1011   E - 14 - 1110
C - 12 - 1100   F - 15 - 1111
```

Overgangen fra E5 (Utvidet ASCII kodesett) til C3 A5 (UTF-8 kodesett) kan analyseres på følgende måte i golang og beviser at golang følger UTF-8 koding disiplinert:

```
fmt.Printf("%s", "\xe5")
Resultatet er "?"

fmt.Printf("%s", "\xc3\xa5")
Resultatet er "å"

fmt.Printf("%c", "\xc3\xa5") // er ikke lovlig, siden char er ikke string
Resultatet er "%!c(string=å)"

fmt.Printf("%c", byte[]("\xc3\xa5"))
Resultatet er "[Ã ¥]"
siden a-tilde (0xc3) og yen-tegn (0xa5) finnes i ISO 8859-1
```

Tillegg B

Basert på https://blog.golang.org/slices

"Arrays"
"Array" - en ordnet anordning, som en samling av bokstaver, for eksempel; [...] brukes i gramatikken til de fleste programmeringsspråkene som betegnelse for en "array", - ["a", "b", "c"]

Spørsmål, som ofte blir stilt når "array"[5] (), er

- skal man bruke en fast størrelse eller en dynamisk størrelse?
- skal størrelse være en del av typedefinisjon?
- hvordan ser multidimensjonale "arrays" ut?
- skal en tom "array" ha en mening?

Under design av Golang brukte man mer enn et år på å svare på disse spørsmålene. Det ble gjort en avgjørelse om å bruke begrepet **"slices"** som en betegnelse på en innebygd (i programmeringsspråket) type for "arrays" med en fast størrelse. Viktige faktorer var utvidbarhet og fleksibilitet.

Datastrukturer ("array", "map", "dictionary", "collection" osv.) i kombinasjon med datatyper dreier seg om lagring i det flyttige minne. I reelle applikasjoner er det vanlig at utvikler ikke vet, hvor store datastrukturer man vil trenge, når programmet utføres. Derfor må alle programmeringsmiljøer ha datastrukturer som gir mulighet til å bestemme størrelse under utførelsen av programmet, dvs. dynamisk utvide datastrukturen hvis nødvendig. I Golang er "append" funksjonen sentral for dette formålet.

"append" (no. "tilføye")
Et eksempel på en "array" eller **slice** i Golang er

```
var buffer [256]byte
```

Type for "array" med navn "buffer" inneholder dets størrelse (256 bytes). For eksempel, `[128]byte` blir en annen datatype i Golang.

Med slike definisjoner etterspør utvikleren om å lagre data i minne.

```
buffer: byte byte ... <opp til 256 ganger> .... byte
```

Utvikleren kan få tilgang til hvert element i "array" vha. indekser, -

```
buffer[23], buffer[123], buffer[255]
```

Man kan også ta ut "skriver" fra den store "skiven"

```
buffer[23:25], buffer[123:255], buffer[1:10]
```

[5] oversettelse til norsk kan være misvisende, men "matrise" er den mest anbefalte direkte-oversettelse; en annen mulighet er å bruke "tabell", men i dette dokumentet bruker vi det engelske ordet "array"

Indekseringen er bestemt i designet av programmeringsmiljøet og for Golang starter det på 0, dvs. for buffer har vi tilgjengelige elementer fra 0 til 255.

Les mer om “slices” her https://blog.golang.org/slices

Referanser

De fleste refferansene er gitt i oppgaver. Dette er tillegslitteratur, som kan være interessant.

Ahmed W. (2014), “Getting started with Go and Test Driven Development”, https://www.binpress.com/tutorial/getting-started-with-go-and-test-driven-development/160, sist sett 2017-02-02

V. Driessen. (2010). A successful Git branching model. Retrieved from ULR http://nvie.com/posts/a-successful-git-branching-model/, sist sett 2017-01

GitHub Help, https://help.github.com/, sist sett 2017-01-24

P. Krill. (2017). “Go tops Java, C, Python for programming language of the year”, http://www.infoworld.com/article/3155924/application-development/go-tops-java-c-python-for-programming-language-of-the-year.html?imm_mid=0ec6d0&cmp=em-prog-na-na-newsltr_20170121, sist sett 2017-01

Golang profiling. https://golang.org/pkg/runtime/pprof/

SLUTT.

JG/2017