Når det gjelder alle programmer/filer som er relatert til oppgavene, så befinner de seg på https://github.com/Dieselgutta/ICA01

1.1

1.2 Oppgaver

1.2.1 Binære tall

Konverter følgende desimaltall til 2-tallssystemet (binært tallsystem):

(1) 1 = 1 – 1bit

(2) 2 = 10 – 2bit

(3) 5 = 101 – 3bit

(4) 8 = 1000 – 4bit

(5) 16 = 10000 – 5bit

(6) 256 = 100000000 – 8bit

For hvert svar oppgi også et antall bits som kreves for å representere det desimale tallet binært. Dere må også beskrive metoden dere brukte for å gjøre konverteringen. - For å konvertere tallene om til 2-tallssystemet brukte vi metoden hvor vi deler på 2 og setter opp 1 som rest der det ikke går. Eks:

5 : 2 = 1

2 : 2 = 0

1 : 2 = 1

Når man har regnet om til binære tall slik vi har vist så leser man svaret nedenfra og opp. Med dette mener vi at man ser på det nederste tallet i "tabellen" først, slik at du leser 1 : 2 = 1 først, så det første binære tallet er 1.

5 = 101

Konverter følgende binære tall til desimaltall (mest signifikante bit-en er til venstre):

(1) 100 = 4

(2) 1001 = 9

(3) 1100110011 = 819

- Vi gjorde det slik:

100 = 0*2^0 + 0*2^1 + 1*2^2= 0 + 0 + 4 = 4

Vi har 0 på enerplassen, 0 på toerplassen og 1 på firerplassen.

1.2.2 Informasjonsmengde

Det binære tallet kan være mellom 000 og 111, altså et tall mellom 0 og 7.

Lise får vite at tallet er et oddetall må det da enten være 1, 3, 5 eller 7. Det er altså 4/8 muligheter her. Ved å bruke formelen log2(1/(M/N)) hvor M er antall oddetall, og N er totalt antall muligheter får vi.

1) log2(1/(4/8)) = 1 bit

Man kan her tenke seg at tallet har to muligheter, enten er det et oddetall, eller så er det ikke det. (1 bit)

Videre får Per vite at tallet ikke er et multiplum av 3. (altså 0, 3 eller 6). Da er det altså ⅝ muligheter her.

2) log2(1/(5/8)) = 0.6780719051126377 bits

Her kan vi også tenke praktisk tenke oss at tallet har to muligheter, enten er det et multiplum av 3, eller så er det ikke det. (1bit)

Videre får Oskar vite at to av de binære tallene er 1, mulighetene blir da enten 011, 101 eller 110 (3, 5 og 6)

3) log2(1(3/8)) = 1 bit

Louise får vite alt de andre vet, hun vet altså at det er ett oddetall, at det ikke er et multiplum av 3, og at det enten er 011, 101 eller 110 (3, 5 eller 6).

Av disse alternativene er det bare 5 som stemmer overens med alle kriteriene. Det blir

4) log2(1/(1/8)) = 3 bits

Louise har da 3 bits med informasjon, og kan dermed finne fram til at det tresifrede binære tallet er 101 (5).

1.2.3 Arbeid med git

Kommandoer:

$ git clone https://github.com/Dieselgutta/ICA01.git

$ cd is105-uke04 $ git add hello.go.txt

$ git status $ git commit -m "endringer"

$ git push origin master

1.2.4 Samarbeid i Git og Introduksjon i Golang

1) Ved å ha en master i et hovedreposetori sikrer man seg selv for å endre i master. De som jobber sammen kan se endringene som andre man samarbeider med har gjort i prosjektet. Man sikkerhetskopierer filene, og man lar flere personer jobbe med flere ulike type kode. Eksempelvis kan noen jobbe med flere features, mens andre jobber med strukturering,opprydding og refactoring. Et eksempel på dette er dersom en ny programvare skal bli lansert så vil ikke informasjonen du har lagret gå tapt. Dette betyr også at debugging og maintenance blir enklere, siden man ikke gjør endringer i master-branchen. Man er da beskyttet mot å ødelegge prosjektet.

På en annen side er git-flow en vrien måte å bruke om man jobber i et agilt prosjekt, der man enten med laste opp eller ned flere ganger om dagen. Man må også legge til endringer, commite og pushe når man skal endre i prosjektet, dette kan være komplisert fordi man må gjøre det i riktig rekkefølge.På den andre siden kan dette også være en enkel og nyttig funksjon dersom man bruker det riktig, og ofte nok. Å dele opp prosjekter i forskjellige grener eller “branches” forsikrer brukerne mot å endre på feil ting.

Mange problemer med å bruk av github kan komme av dårlig kommunikasjon i gruppen. Hvis flere skal endre på en kode, blir det dumt om alle endrer på den samme tingen. Det er derfor viktig å kommunisere på en god måte hva, når og hvor du gjør endringer.

2) Windows: Portabel Executables, for eksempel (.exe, .dll .acm), Mac: Mach-O, Linux: COFF (Common Object File Format), for eksempel .o eller .obj. På Linux er det ikke lenger COFF som er vanlig. Dette er fordi alle filer på Linux kan kjøres. Dette kommer fra filosofien “everything on Linux is a file”. Dette betyr at endingen, eller filformatet, kun beskriver hva filen gjør/hvordan den blir kjørt.

De har forskjellige filer for å optimalisere systemet. Filene lagres og hentes ut på forskjellige måter i de forskjellige systemene, og det er derfor naturlig at de opererer annerledes.
Når man prøver å kjøre en fil i Windows vil operativsystemet kun se på filnavnet for å bestemme hvordan den skal kjøre den. Dersom det ikke finnes noe filnavn, eller filnavnet er feil, vil ikke filen bli kjørt riktig. Dette skjer ikke i Mac OS X eller Unix/Linux. Her er informasjon om filen lagret i begynnelsen av selve filen, istedenfor på slutten i form av et filnavn. På grunn av dette kan en fil bli kjørt i Mac eller Linux uansett om den har et filnavn eller ikke. Dette kalles en MIME-type, og blir ikke brukt av Windows-systemer.

3) Konstruksjonen er ulik java. I java bruker man semikolon “;” i slutten av en linje, noe man ikke gjør i Golang. Det er også forskjeller som fmt. istedenfor “System.out.” for print-funksjoner osv. fmt må importeres før bruk, noe man ikke må i Java. Man har heller ikke felt og constructorer i golang. I Golang kommer også variabel-typen etter variabel-navnet, her trenger man også bare å skrive variabel-typen bak den siste variabelen i en rekke (så lenge det er samme variabel). Arbeid med variabler i Golang er mye simplere, de kan alle konstrueres på rekke og rad uten å erklære variabelen i begynnelsen. Variabler får også en nullverdi automatisk hvis man ikke definerer den til annet (0, false, “ ”). Man kan også returnere flere resultater med en return, i motsetning til java hvor man bare kan returnere ett resultat. Man kan også navngi “return”-verdier for å gjøre dokumentasjonen enklere.

4) Når man bruker et programmeringsmiljø kommuniserer det direkte med plattformen man er på. Den vil også illustrere at man kan hente ut informasjon fra filer som ligger i akkurat samme mappe. Dette gjør det mer strukturert og er da lettere å hente ut informasjon, uten å måtte lete gjennom hele koden.

5) Ettersom github gjør deling av arbeid enklere kan det være hensiktsmessig å legge denne filen til et repository. Det blir lettere for andre å gjøre endringer til filen, eller å bruke filen i kombinasjon med andre program. Siden logcli kan brukes som et utgangspunkt i logbcli kan det også være greit å ha den opplastet til github.

6) Pakken log inneholder for det meste funksjoner mens pakken fmt inneholder en rekke forkortelser og definisjoner. Pakken log må vi selv implementere og må vite hvor vi oppretter filene og hvor vil velger å legge dem på datamaskinen. Den inneholder også (type Logger). fmt inneholder print-funksjoner, og ikke noe særlig mer, den printer kun det som er definert.