Oblig 1 - INF1060 - høst 2016

Denne obligen er ment å gi deg en bred men grunn innføring i C. Temaene som dekkes er i hovedsak strenger, pekere, structer og filer. Obligen er over tre uker og det vil naturligvis ikke være alle temaene som blir gjennomgått første uken, så det kan være lurt å hoppe frem og tilbake gjennom obligen og gjøre de oppgavene du har forutsetning for å løse.

Oppgavene skal løses selvstendig, se ellers Obligreglementet. Du vil finne nyttige ukesoppgaver med tilhørende forklaringer av viktige begreper på gruppelærersiden, og som i de fleste fag er semestersiden stedet for forelesningsfoiler og annen inforomasjon du kanske er på utkikk etter.

Dersom du har spørsmål underveis kan du oppsøke en orakeltime eller stille spørsmål på Piazza. Det vil (selvfølgelig) være mye relevant informasjon i plenumstime.

Husk å les hele oppgaveteksten (!), og det kan være lurt å også lese hele oppgavesettet før du begynner, så du vet hvor mye arbeid du har igjen totalt sett.

Oppgavene blir testet på Linux på Ifi sine maskiner eller tilsvarende.

Innleveringsfrist: Torsdag 15. september 23:59.

Oppgave 1

a)

Lag et enkelt program som tar inn et kommandolinjeargument og som printer det ut.

Eksempel

```
$> ./a.out "Hello world!"
Hello world!
$>
```

b)

Lag en make-fil (Makefile) som kan kjøres med programmet `make` i terminalen for å kompilere programmet fra a. Make skal lage filen `hello` som kan kjøres med `./hello` i stedet for `./a.out` som er det gcc ellers gir oss.

Eksempel

```
$> make
gcc -o hello hello.c
$>./hello "Hello world!
Hello world!
$>
```

Oppgave 2 - Strenger

Denne oppgaven handler om strenger og diverse strengoperasjoner. Du skal skrive funksjonene spesifisert i oppgavene under, og så bruke denne filen til å kjøre tester av funksjonene dine. Det er denne test-filen som blir brukt av retterne. Det betyr at funksjonene under skal ligge i en eller flere c-filer som kompileres sammen med den utleverte test-filen som inneholder main-metoden. For å få det til å kompilere må alle funksjonene eksistere. Testprogrammet kommer kanskje til å få segmentation fault hvis dere ikke har gjort ting riktig i streng-oppgavene, dere kan da inspisere test-koden for å finne ut hvor det kræsjer og hvorfor, og så fikse feilen deres. Dersom du finner en feil i testkoden blir det premie.

a)

Skriv en make-fil som du utvider etterhvert som du lager flere av oppgavene i oppgave 2.

b)

Skriv funksjonen

```
int stringsum(char* s)
```

som tar inn en char-peker og som returnerer summen av den innsendte strengen. Summen av en streng defineres som den akkumulerte verdien av alle karakterer i strengen. For denne oppgaven vil store og små bokstaver være det samme, og vi definerer at `a` (og derfor også `A`) har verdien 1, `b` har verdien 2, osv. Dersom stringen inneholder en karakter som ikke er en stor eller liten bokstav skal funksjonen returnere -1.

Det er flere måter å gjøre denne oppgaven enklere på, og det kan være lurt å lage en enkel løsning før du forbedrer den så den møter alle kriteriene i oppgaven. For eksempel kan du la vær å ta høyde for store bokstaver.

Eksempler

```
int test = stringsum("abcd"); //test har verdien 1+2+3+4 = 10
int test = stringsum("hei!"); //test har verdien -1
```

c)

Skriv funksjonen

```
int distance_between(char* s, char c)
```

som tar inn en char-peker og en char som argumenter, og som returnerer avstanden i antall tegn mellom første og andre forekomst av karakteren `c` i strengen `s`. Dersom `c` forekommer færre enn 2 ganger i teksten (som gjør det umulig å finne avstanden mellom to), skal funksjonen returnere -1.

Eksempler

```
int test = distance_between("a1234a", 'a'); //test har verdien 5
int test = distance_between("hei!", 'a'); //test har verdien -1
```

d)

Skriv funksjonen

```
char* string_between(char* s, char c)
```

som tar inn en char-peker og en char som argumenter, og som returnerer en ny streng som er den som er mellom første og andre forekomst av karakteren `c` i strengen `s`. Dersom `c` forekommer færre enn 2 ganger i teksten (som gjør det umulig å finne avstanden mellom to), skal funksjonen returnere NULL.

Her er du nødt til å bruke `malloc()` for å heap-allokere plass til den nye strengen du skal returnere.

Eksempler

```
char* test = string_between("a1234a", 'a'); //test har verdien "1234"
char* test = string_between("hei!", 'a'); //test har verdien NULL
```

e)

Skriv funksjonen

```
char** split(char* s)
```

som tar inn en char-peker som argument og som returnerer en array med strenger der hvert ord i den innsendte strengen er satt på hver sin plass i arrayet. Den innsendte strengen `s` skal ikke endres. Det returnerte arrayet må termineres med en nullpeker for å vise at det ikke er flere ord igjen.

Her er du nødt til å bruke `malloc()` for å heap-allokere plass til det doble arrayet du skal returnere.

Eksempler

```
char** test = split("hei du der");
/*test har nå verdien {"hei", "du", "der", NULL}, som også kan skrives på formen
{{'h', 'e', 'i', '\0'}, {'d', 'u', '\0'}, {'d', 'e', 'r', '\0'}, NULL}}.*/
```

f)

Tegn en tegning av char-arrayene før og etter kallet på split i eksempelet over. Hvordan ser strengen "hei du der" ut i minnet, og hvordan ser det ut i minnet etter at split har kjørt?

g)

Skriv en ny versjon av funksjonen stringsum, men denne gangen som

```
void stringsum2(char* s, int* res)
```

som ikke returnerer noen verdi, men som legger resultatet av utregningen (strengsummen) i int-en pekt på av res.

Eksempler

```
int res;
stringsum2("abcd", &res); //res har nå verdien 10
stringsum2("ab!", &res); //res har nå verdien -1
```

Oppgave 3 - Structer

Her skal du som til de andre to oppgavene også lage en make-fil.

a)

Definer strukten

`struct datetime`

som representerer en kombinasjon av tid og dato. Strukten skal kunne lagre tidspunkt i timer, minutter og sekunder, og dato med årstall, måned og dato. Merk at du ikke skal lagre ukedag, men dag i måneden.

b)

Skriv funksjonen

```
void init_datetime(struct datetime* dt, ...)
```

som initialiserer en datetime-strukt med passende verdier. Det er opp til deg hvilke argumenter funksjonen tar inn, men den må minst ta inn en peker til en datetime-strukt, den som skal initialiseres.

Eksempel

```
struct datetime dt;
init_datetime(&dt, ...); //dt er nå initialisert
```

c)

Skriv funksjonene

```
void datetime_set_date(struct datetime* dt, ...)
void datetime_set_time(struct datetime* dt, ...)
```

som begge tar inn en peker til en datetime-strukt og endrer henholdsvis dato og tidspunkt i strukten. Eventuelle andre argumenter bestemmer du her også selv.

d)

Skriv funksjonen

```
void datetime_diff(struct datetime* dt_from,
                   struct datetime* dt_to,
                   struct datetime* dt_res)
```

som begge tar inn tre pekere til datetime-strukter. Funksjonen skal regne ut avstanden i år, måneder, dager, timer, minutter og sekunder mellom de to struktene pekt på av `dt_from` og `dt_to`, og fylle inn resultatet i strukten pekt på av den siste pekeren, `dt_res`.

Levering

1. Lag en mappe med ditt brukernavn: `mkdir bnavn`
2. Lag en mappe for hver oppgave (1, 2, 3, ...) under den første mappen.
3. Kopier alle filene som er en del av innleveringen inn i sin tilhørende mappe:
   `cp x.c bnavn/OppgaveX/` (f.eks.)
4. Komprimer og pakk inn mappen:
   `tar -czvf bnavn.tgz bnavn/`
5. Logg inn på Devilry
6. Lever under INF1060 Oblig 1