Oblig 1 – IN2140 – Vår 2021

I denne oppgaven skal du skrive en rekke mindre programmer for å bli bedre til å programmere i C.

Oppgavene skal løses selvstendig, se reglene for obligatoriske oppgaver på Obligreglementet. Dersom du har spørsmål underveis kan du oppsøke en orakeltime/gruppetime.

Retterne vil teste innleveringer på de Linux-basertr login-maskinene til IFI (maskiner som deler navnet login.ifi.uio.no seg imellom). Programmene dine **MÅ** kunne kompileres og kjøres på ifi sine login-maskiner (login.ifi.uio.no).

Innlevering i Devilry innen 10. februar 23:59.

Oppgave 1 - Bytte bokstav

I denne oppgaven skal du skrive et program som bytter ut alle forekomster av en bokstav med en annen bokstav i en setning, og en makefile som kompilerer programmet på nytt etter en oppdatering av kildefilen.

Skriv programmet

Programmet skal ta imot 3 argumenter: en setning og to bokstaver. Alle forekomster av den første bokstaven i setningen skal byttes ut med den andre bokstaven. Skriv så ut den nye setningen til terminalen.

Kall kildefilen 'oppgave1.c'.

Eksempelbruk:

```
$ ./oppgave1 "Dette er en setning" e a
Datta ar an satning
```

Lag makefile

Lag en makefile som kompilerer programmet automatisk hvis kildefilen har forandret seg. Sjekk at programmet blir kompilert kun hvis det er blitt gjort endringer i kildefilen siden sist kompilering.

Eksempelkjøring:

```
$ make
gcc oppgave1.c -o oppgave1
$ make
make: Nothing to be done for 'all'.
$ touch oppgave1.c
$ make
gcc oppgave1.c -o oppgave1
```

Oppgave 2 - Strenger i C

Denne oppgaven handler om C-strenger og diverse strengoperasjoner. Du skal skrive funksjonene spesifisert i oppgavene under, og så bruke den vedlagte test-filen (oblig1_tests.c) til å kjøre tester av funksjonene dine. Du skal skrive funksjonene dine i en annen fil uten main-funksjon. Du skal så kompilere både din fil og filen oblig1_tests.c og binde de sammen (linke de) til et eksekverbart program.

Hvis du gjør feil i implementasjonene kan testprogrammet få segmentation fault. Da anbefaler vi at du bruker gdb til å finne ut hvor i programmet feilen skjer. Merk at du må kompilere med flagget -g for å få debug-informasjon inkludert i programmet som gdb bruker.

a)

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

For å få oblig1_tests.c og filen med funksjonene til å kompilere må alle funksjonene som brukes av oblig1_tests.c være definerte. Dette kan du få til ved å lage skall-implementasjoner av alle funksjonene som vist under.

```
int stringsum (char *s){ return 0; }
```

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. 0-byten som avslutter strengen inngår ikke i summen. 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 være å ta høyde for store bokstaver.

Tipp: Datatypen char fungerer ikke bare som bokstav. Den funker samtidig som en tallverdi som kan ha verdier fra -128 til 127. Man kan utnytte dette for en effektiv løsning.

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 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 - Multiple choice: Teorispørsmål

Operativsystemer

Hvilke påstander er sanne om operativsystemer generelt? Velg ett eller flere alternativer

1. Et OS er en samling av programmer eller funksjoner som opererer som et mellomlag mellom hardware og brukerene av systemet.
2. Et mikrokjerne OS har minimal funksjonalitet hvor andre tjenester implementeres som server-prosesser kjørende i brukerområdet ("user space").
3. Et OS gir brukerne direkte tilgang til hardware.
4. I et monolitisk OS har de fleste vanlige brukerprogrammer full aksess til alle ressurser og instruksjoner på datamaskinen.
5. Uten et OS kan man ikke kjøre programmer på datamaskinen.

fork()

Hvilke påstander er sanne?
Systemkallet fork() ...

1. oppretter en ny prosess som er en kopi av prosessen som gjorde kallet.
2. starter et helt nytt program som er gitt av en parameter som peker på en eksekverbar fil.
3. genererer flere schedulerings-køer.
4. forgreiner eksekveringen til et program og kjører hver forgrening sekvensielt.
5. muliggjør at en prosess kan dele minne med en annen prosess.

Schedulering

Hvilke svar er riktige?
Ikke-preemptiv schedulering ...

1. avbryter prosessor bare etter at de har brukt opp sin tildelte tidsskive.
2. vil forhindre at en prosess kan avbrytes av en interrupt.
3. betyr at en ny prosess kan kjøre bare hvis den tidligere prosessen enten avslutter eller gi fra seg retten å kjøre.
4. vil tillate at en prosess kan avbrytes av en exception.

minne i C

I funksjonen `main()` i et C-programm kompilert for IFI sine login-maskiner finner vi følgende deklarasjonen av et array:

```
int numbers[10];.
```

Hvilke påstander om den er sanne?

1. Adressen til `numbers[4]` er 1 bytes større enn adressen til `numbers[3]`
2. Adressen til `numbers[4]` er 4 bytes større enn adressen til `numbers[3]`
3. Adressen til `numbers[4]` er 8 bytes større enn adressen til `numbers[3]`
4. Størrelse av numbers[2] er 1 byte
5. Størrelse av numbers[2] er 4 bytes
6. Størrelse av numbers[2] er 8 bytes

Levering

1. Legg alle filene i en mappe med ditt brukernavn

   ```
   $ mkdir brukernavn
   $ cp oppgave*.c Makefile brukernavn/
   ```

2. Lag så en komprimert arkiv som du leverer

   ```
   $ tar czf brukernavn.tar.gz brukernavn/
   ```

3. (Strekt anbefalt!) Test innleveringen ved å laste den leverte tar.gz-filen ned til en IFI-maskin, pakk ut, kompiler og kjør.

   ```
   $ tar xzf brukernavn.tar.gz
   $ cd brukernavn
   $ make
   $ ./oblig1_tests
   ```