IN4120 – Introduksjon til operativsystemer og datakommunikasjon

Oblig 1: Bli kjent med C

Vår 2022

I denne obligen skal du skrive en rekke mindre programmer for å bli bedre kjent med programmeringsspråket C. Oppgavene skal løses selvstendig; man kan diskutere løsningsteknikker, men å dele kode er ikke lov. Se retningslinjene for obligatoriske oppgaver og andre innleveringer ved Ifi.

Retterne vil teste innleveringer på Ifis Linux-maskiner, tilgjengelige gjennom `login.ifi.uio.no`. Programmene dine må kompilere og kjøre på disse maskinene. Dersom du lurer på noe om oppgavene, så spør gruppelæreren din.

Besvarelse leveres gjennom Devilry innen fredag 18. februar kl. 23.59.

In this mandatory assignment you will write a number of smaller programs to become better acquainted with the programming language C. The tasks must be solved independently; you can freely discuss solution techniques, but sharing code is not allowed. See retningslinjene for mandatory assignments and other submissions at Ifi.

The correctors will test submissions on Ifi's Linux machines, available through `login.ifi.uio.no`. Your programs must compile and run on these machines. If you have any questions about the assignments, ask your group teacher.

You must have submitted in Devilry by Friday 18 February at 23.59.

Oppgave 1: Vokalbytte

I denne oppgaven skal du skrive et program som bytter ut alle vokalforekomster med en annen bokstav i en setning – i tillegg til en makefile som kompilerer programmet på nytt etter oppdatering av kildefilen, som skal hete `vowelshift.c`.

Vi anbefaler at du kompilerer programmet ditt med flaggene `-g -Wall -Wextra -std=gnu11`.

In this exercise you will write a program that replaces all vowel occurrences with another letter in a sentence—in addition to a makefile that recompiles the program after updating the source file, which will be called `vowelshift.c`.

We recommend that you compiler your program with the flags `-g -Wall -Wextra -std=gnu11`.

a)

Skriv en makefile som kompilerer programmet. Sjekk at programmet blir kompilert på nytt kun hvis kildefilen er blitt endret siden den sist ble kompilert.

Write a makefile that compiles the program. Check that the program is recompiled only if the source file has been modified since it was last compiled.

Eksempel

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

b)

Programmet tar to argumenter: én setning og én bokstav. Alle fem engelske vokaler i setningen, det vil si 'a', 'e', 'i', 'o' og 'u' skal byttes ut med bokstaven. Skriv så ut den forandrede setningen til terminalen.

The program takes two arguments: one sentence and one letter. All five English vowels in the sentence, ie 'a', 'e', 'i', 'o' og 'u' must be replaced with the letter. Then print the changed statement to the terminal.

Eksempel

```
$ ./vowelshift "Lorem ipsum dolor sit amet" a
Laram apsam dalar sat amat
```

Oppgave 2: Strengoperasjoner

Denne oppgaven handler om strenger og diverse strengoperasjoner. Du skal skrive funksjonene i en egen fil med navn `stringoperations.c`, som ikke har `main`-funksjon. Bruk testprogrammet i den vedlagte filen `tests.c` for å teste funksjonene. Begge filene skal kompileres og lenkes sammen til et kjørbart program. Man kan enkelt lage et program fra to kildefilder `fil_with_main.c` og `fil_without_main.c` ved å kompilere som følger:

This assignment is about strings and various string operations. You must write the functions in a separate file named `stringoperations.c`, which does not have the `main` function. Use the test program in the attached file `tests.c` to test the functions. Both files must be compiled and linked together into an executable program. It is easy to create a program from two source files `fil_with_main.c` and `fil_without_main.c` by compiling as follows:

```
gcc fil_with_main.c fil_without_main.c -o ferdig_program
```

Kodefeil i implementasjonen kan forårsake segmenteringsfeil i programmet. I så fall anbefaler vi at du bruker Gdb og Valgrind for å finne ut hvor i programmet feilen oppstår. Merk at du må kompilere med flagget `-g` for å inkludere debuginformasjon i programmet, som både Gdb og Valgrind bruker.

Code errors in the implementation can cause segmentation errors in the program. In that case, we recommend that you use Gdb and Valgrind to find out where in the program the error occurs. Note that you must compile with the `-g` flag to include debug information in the program, which both Gdb and Valgrind use.

a)

Skriv en makefile som du utvider etter hvert som du løser deloppgavene. For å få `tests.c` og filen med funksjonene til å kompilere, må alle funksjonene som brukes av `tests.c` være definerte. Dette kan du løse ved å lage midlertidige tomme implementasjoner av alle funksjonene, som f.eks.:

Write a makefile that you expand as you solve the sub-tasks. For `tests.c` and the function file to compile, all the functions used by `tests.c` must be defined. You can solve this by creating temporary empty functions first, like this:

```
int stringsum (char* s)
{
  return -1;
}
```

b)

Skriv funksjonen

Write the function

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

som tar en char-peker som argument og returnerer en int-verdi som vi kaller strengsummen. Vi definerer strengsummen som den akkumulerte verdien av alle tegn i strengen, hvor verdien tilsvarer alfabetisk posisjon: 'a' har verdien 1, 'b' har verdien 2 og så videre. Vi skiller ikke mellom små og store bokstaver. Mellomrom og den avsluttende nullbyten inngår ikke i summen. Dersom strengen inneholder noen tegn som ikke er en stor eller liten bokstav eller mellomrom, så skal funksjonen returnere -1.

which takes a char pointer as an argument and returns an int value that we call the string sum. The string sum is defined as the accumulated value of all characters in the string, where the value corresponds to the alphabetical position: 'a 'has the value 1,' b' has the value 2 and so on. We do not distinguish between lowercase and uppercase letters. Spaces and the final null byte are not included in the sum. If the string contains any character that is not a uppercase or lowercase letter or space, then the function should return -1.

Det er flere måter å forenkle denne oppgaven på; det kan være lurt å lage en enkel løsning før du forbedrer den så den møter alle kriteriene i oppgaven.

There are several ways to simplify this task; you may want to create a simple solution before you improve it so that it meets all the criteria in the task.

Tips: En char er en énbytes tallverdi, som kan ha verdier fra -128 til 127, og som oftest tolkes som en bokstav ved hjelp av ASCII-tabellen. Dette kan man utnytte for en effektiv løsning – ved å beregne tallverdiene som trenges for å løse oppgaven fra bokstavenes ASCII-verdier.

Tip: A char is a one-byte numeric value, which can have values from -128 to 127, and is most often interpreted as a character using the ASCII table. This can be exploited for an efficient solution—by computing the numeric values needed to solve the task from the characters' ASCII values.

Eksempler

```
int result = stringsum("Lorem ipsum dolor sit amet");
// result har verdien 292
int result = stringsum("L0rem 1psum d0l0r s1t amet");
// result har verdien -1
```

c)

Skriv funksjonen

Write the function

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

som tar en char-peker og en char som argumenter og returnerer avstanden (altså differansen mellom posisjonene) fra første til siste forekomst av tegnet i strengen. Dersom tegnet forekommer kun én gang så skal funksjonen returnere 0. Dersom tegnet ikke forekommer, så skal funksjonen returnere -1.

which takes a char pointer and a char as arguments and returns the distance (i.e. the difference between the positions) from the first to the last occurrence of the character in the string. If the character occurs only once then the function must return 0. If the character does not occur, then the function must return -1.

Eksempler

```
int result = distance_between("Lorem ipsum dolor sit amet", 'm');
// result har verdien 19
int result = distance_between("Lorem ipsum dolor sit amet", 'u');
// result har verdien 0
int result = distance_between("Lorem ipsum dolor sit amet", 'y');
// result har verdien -1
```

d)

Skriv funksjonen

Write the function

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

som tar en char-peker og en char som argumenter og returnerer delstrengen som befinner seg mellom første og siste forekomst av tegnet i strengen. Dersom tegnet forekommer kun én gang, så skal funksjonen returnere den tomme strengen. Dersom tegnet ikke forekommer, så skal funksjonen returnere nullpekeren.

Her må du bruke `malloc` for å allokere plass i heapen for den nye strengen som du skal returnere. Ikke glem å frigjøre plassen på rett sted.

which takes a char pointer and a char as arguments and returns the substring that is located between the first and last occurrence of the character in the string. If the character occurs only once, then the function should return the empty string. If the character does not appear, then the function should return the null pointer.

Here you must use `malloc` to allocate space on the heap for the new string to be returned. Don't forget to deallocate that space at the right place.

Eksempler

```
char* result = string_between("Lorem ipsum dolor sit amet", 'o');
// result peker på strengen "rem ipsum dol"
char* result = string_between("Lorem ipsum dolor sit amet", 'a');
// result peker på den tomme strengen
char* result = string_between("Lorem ipsum dolor sit amet", 'y');
// result er nullpekeren
```

e)

Skriv funksjonen

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

som fungerer som `stringssum`, men heller enn å returnere strengsummen legger den i int-en som `res` peker på; returnerer 0 ved suksess og -1 ved feil.

Write the function

which behaves like `stringsum`, but rather than returning the string sum, puts it into the int that `res` points to. The functions returns 0 for success and -1 for failure.

Eksempler

```
int result;
stringsum2("Lorem ipsum dolor sit amet", &result);
// result har verdien 292
stringsum2("L0rem 1psum d0l0r s1t amet", &result);
// result har verdien -1
```

Oppgave 3: Marken i eplet

Du har følgende variabel:

You have the following variable:

```
char* apple = "                      a\n"
              "                     ppl\n"
              "                   eappl\n"
              "                  eapple\n"
              "                  apple\n"
              "                  appl\n"
              "          eappleappleappleappl\n"
              "      eappleappleappleappleapple\n"
              "    appleappleappleappleappleapple\n"
              "  appleappleappleappleappleappleappl\n"
              " eappleappleappleappleappleappleapple\n"
              "appleappleappleappleappleappleappleapp\n"
              "leappleappleappleappleappleappleapplea\n"
              "ppleappleappleappleappleappleappleappl\n"
              "eappleappleappleappleappleappleappleap\n"
              "pleappleappleappleappleappleappleapple\n"
              "appleappleappleappleappleappleappleapp\n"
              "leappleappleappleappwormpleappleapplea\n"
              " ppleappleappleappleappleappleappleap\n"
              "  pleappleappleappleappleappleapplea\n"
              "    ppleappleappleappleappleappleapp\n"
              "     leappleappleappleappleappleap\n"
              "       pleappleappleappleappleapp\n"
              "         leappleappleappleapple\n"
              "           appleappleappleapp\n";
```

Du finner strengen i filen `the_apple.c` og headerfilen `the_apple.h`.

Filen som inneholder ditt program skal hete `apple.c`

Det er bare en mark i eplet, men marken kan vokse seg stor ved å duplisere sine bokstaver. Da ser den kanskje slik ut:

```
"leappleappleappleappwwwwooooooooorrrrrr\n"
" rrmmmmmleappleappleappleappleappleap\n"
```

You find the string in the file `the_apple.c` and the header file `the_apple.h`.

The file containing your application must be named `apple.c`

There is only a single worm in the apple, but that worm can grow very long by repeating its characters. It could look like this:

a)

Skriv funksjonen

```
int locateworm(char* apple)
```

som finner marken i eplet ved kun å bruke grunnleggende språkkonstruksjoner som `if ... else`, `for`, `while`, `do` og så videre – altså ingen funksjoner, som `strchr` eller `strstr`.

Nærmere bestemt skal funksjonen returnere tallet som er array-indeksen til den første bokstaven som tilhører marken.

Eksempel

```
int result = locateworm(apple);
// result har verdien 565
```

Write the function

```
int locateworm(char* apple)
```

that finds the worm in the apple by using only basic language constructions such as `if ... else`, `for`, `while`, `do` and so on - that is, no functions, such as `strchr` or `strstr`.

More specifically, the function must return the number that is the array index of the first character that belongs to the worm.

b)

Skriv funksjonen

```
int removeworm(char* apple)
```

som skjærer ut marken av eplet ved å overskrive den med mellomrom. Funksjonen returnerer markens lengde (antallet bytes som ble erstattet med mellomrom) eller 0 hvis marken ikke finnes.

Eksempel

```
int result = removeworm(apple);
// result har verdien 4
int result = removeworm(apple);
// result har verdien 0
```

Write the function

```
int removeworm(char* apple)
```

which cuts the worm out of the apple by overwriting it with whitespace characters. The function returns the length of the worm (the number of bytes that were replaced by writespace characters) or 0 if the worm does not exist.