

Eksamen INF-1100
Innføring i programmering og
datamaskiners virkemåte
Høst 2012

Eksamenssettet består av 4 oppgaver.

Der oppgaven ber om at du skriver en funksjon kan du bruke C lignende pseudo-kode. Husk også at du kan referere tilbake til funksjoner du tidligere har definert.

Oppgave 1 - 20%

Hvilken verdi vil funksjonen *ukjent* nedenfor returnere ved følgende kall:

a) *ukjent*(2, 1)

Funksjonen utfører multiplikasjon.

Løsningsforslag 1a:

2

b) *ukjent*(5, 3)

Løsningsforslag 1b:

15

```
int ukjent(int a, int b)
{
    int p;

    p = 0;
    while (b != 0) {
        if ((b & 0x1) != 0) {
            p = p + a;
        }
        a = a << 1;
        b = b >> 1;
    }
    return p;
}
```

Oppgave 2 - 25%

- a) Gi en kort beskrivelse av komponentene i von Neumann modellen.
- b) Beskriv kort hvordan I/O utføres i en von Neumann-basert datamaskin.

Oppgave 3 - 20%

Gitt et array A som inneholder n flyttall (float eller double i C).

- a) Skriv en funksjon som beregner gjennomsnittet av tallene i A :

```
double gjennomsnitt(double *A, int n)
```

Gjennomsnitt kan beregnes med følgende formel:

$$\frac{1}{n} \sum_{i=0}^{n-1} x_i$$

- x_i er tallet i posisjon i i array A .

Løsningsforslag 3a:

```
double gjennomsnitt(double *A, int n)
{
    int i;
    double snitt;

    snitt = 0;
    for (i = 0; i < n; i++) {
        snitt = snitt + A[i];
    }
    return snitt/n;
}
```

b) Skriv en funksjon som beregner standardavvik for tallene i A :

```
double standardavvik(double *A, int n)
```

Standardavvik kan beregnes med følgende formel:

$$\sqrt{\frac{1}{n} \sum_{i=0}^{n-1} (x_i - \bar{x})^2}$$

- x_i er tallet i posisjon i i array A .
- \bar{x} er gjennomsnittet av alle tallene i A .
- For å beregne kvadratroten kan du anta at det eksisterer en funksjon `double sqrt(double n)`.

Løsningsforslag 3b:

```
double standardavvik(double *A, int n)
{
    int i;
    double avvik;
    double snitt;
    double xi;

    snitt = gjennomsnitt(A, n);
    for (i = 0; i < n; i++) {
        avvik = avvik + (A[i] - snitt)*(A[i] - snitt);
    }
    return sqrt(avvik/n);
}
```

Oppgave 4 - 35%

Et datamaskinsystem støtter typisk hierarkisk organisering av filer ved hjelp av kataloger (mapper/directories). Kataloger har navn, på samme måter som filer har navn. For eksempel kan en lage en katalog med navn "a" og lagre en fil med navn "f" i denne katalogen.

Anta at en *sti* er definert som en tekststreng som angir hvilke kataloger en må besøke for å finne en bestemt fil. Med utgangspunkt i eksempelet over vil en sti til filen "f" være tekststrengen "a/f". Merk at tegnet skråstrek ('/') benyttes for å skille katalognavn fra filnavn. Merk også at en kan lage kataloger inne i kataloger og i en slik sti vil skråstrek benyttes for å skille katalognavn fra hverandre. For eksempel, dersom "a" ligger inne i en katalog "b" vil sti til filen "f" være "b/a/f".

- a) Gitt en sti, skriv en funksjon som avgjør hvor mange kataloger en må besøke for å finne en bestemt fil:

```
int antallkataloger(char *sti)
```

Løsningsforslag 4a:

```
int antallkataloger(char *sti)
{
    int i;
    int num;

    num = 0;
    for (i = 0; sti[i] != 0; i++) {
        if (sti[i] == '/')
            num++;
    }
    return num;
}
```

- b) Anta at en sti er delt opp i *komponenter* og plassert i en liste. Med komponenter menes katalognavn og filnavn. For eksempel, dersom en deler opp stien “b/a/f” i komponenter vil en få en liste hvor første element er tekstrengen “b”, andre element “a” og siste element “f”. Skriv en funksjon som tar utgangspunkt i en liste med komponenter og rekonstruerer den originale stien:

```
void listetilsti(char *sti, list_t *stikomponenter)
```

Skriv den rekonstruerte stien til arrayet gitt ved argumentet *sti*. Du kan anta at arrayet er stort nok til å holde alle tegnene til stien.

Løsningsforslag 4b:

```
void listetilsti(char *sti, list_t *stikomponenter)
{
    int i;
    int k;
    char *komponent;

    i = 0;
    komponent = list_removefirst(stikomponenter);
    while (komponent != NULL) {
        for (k = 0; komponent[k] != 0; k++) {
            sti[i] = komponent[k];
            i++;
        }

        komponent = list_removefirst(stikomponenter);
        if (komponent != NULL) {
            sti[i] = '/';
            i++;
        }
    }
    sti[i] = 0;
}
```

- c) Anta at det eksisterer en spesiell katalog med navn “..”. Forekomst av denne spesielle katalogen i en sti skal ha betydningen: gå tilbake til katalogen over. For eksempel, stien “b/a/./a” skal tolkes som gå til “b”, deretter “a”, deretter tilbake til “b”, deretter til “a”.

Skriv en funksjon som *normaliserer* en sti uttrykt som komponenter i en liste. Normalisering innebærer at forekomster av “..” fjernes i henhold til betydningen gitt over. For eksempel, dersom listen inneholder “b”, “a”, “..”, “a” og “f” skal listen etter normalisering inneholde “b”, “a” og “f”. Funksjonen skal returnere en normalisert liste:

```
list_t *normalisersti(list_t *stikomponenter)
```

Du kan anta at forekomster av ‘.’ aldri vil referere til kataloger som ikke er med i stien (ingen stier som “../..” osv.). Hint: en nyttig tilnærming vil være å opprette en ny liste og flytte komponenter fra *stikomponenter* til den nye listen samtidig som en sjekker etter forekomster av “..” og håndterer disse passende..

Løsningsforslag 4c:

Antar at det eksisterer en funksjon *strlen* som returnerer lengden på en streng.

```
list_t *normalisersti(list_t *stikomponenter)
{
    list_t *normsti;
    char    *komponent;

    normsti = list_create();
    komponent = list_removefirst(stikomponenter);
    while (komponent != NULL) {
        if (strlen(komponent) >= 3 &&
            komponent[0] == '.' && komponent[1] == '.' && komponent[2] == 0) {
            list_removefirst(normsti);
        } else {
            list_addlast(normsti, komponent);
        }
        komponent = list_removefirst(stikomponenter);
    }
}
```

Husk at alle tekststrenger er avsluttet med verdien 0.
Du kan anta at følgende listefunksjoner er tilgjengelige:

Bokmål

```
// Lag en ny liste
list_t *list_create(void);

// Sett inn et element sist i en liste
int list_addlast(list_t *list, void *item);

// Fjern første element i en liste.  NULL returneres
// dersom listen er tom
void *list_removefirst(list_t *list);

// Fjern siste element i en liste.  NULL returneres
// dersom listen er tom
void *list_removelast(list_t *list);
```

Bokmål