UiT

NORGES
ARKTISKE
UNIVERSITET

EKSAMENSOPPGAVE

Eksamen i: Inf-1100
Dato: Mandag 9. desember 2013
Tid: Kl 09:00 - 13:00
Sted:
Etternavn A-L: Teorifagbygg hus 1, plan 3
Etternavn M-O: Adm.bygget, Aud Max
Etternavn P-Å: Teorifagbygg hus 1, plan 2

Tillatte hjelpemidler: Ingen

Eksamen inneholder 4 sider
inklusiv denne forside

Kontaktperson: Jan-Ove A. Karlberg
Telefon: 95281653

UNIVERSITETET I TROMSØ

Oppgave 1 – 25%

Beskriv Von Neumann modellen. Beskrivelsen bør omfatte de ulike komponentene i modellen, hvordan disse interagerer med hverandre, og hva som skjer når instruksjonene i et program utføres.

Oppgave 2 – 20%

Vi har et et array `A` med `n` positive flyttall. Vi har en funksjon som beregner gjennomsnittet av tallene i `A`:

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

Vi har også en funksjon som beregner standardavvik for tallene i `A`:

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

Bruk disse to funksjonene i løsningen av deloppgavene under. Du trenger *ikke* å skrive disse to funksjonene.

a) Lag funksjonen som beregner andelen av elementer (i prosent) i et array som er under et standardavvik fra gjennomsnittverdien av tallene i arrayet (de tallene som er innenfor standardaviket fra gjennomsnittverdien):

```
double andel(double *A, int n);
```

(Hint: Hvis gjennomsnittverdien av tallene i `A` er s og standardavviket er r, så er tallene som er under et standardavvik fra gjennomsnittverdien alle tall t hvor $t > (s - r)$ og $t < (s + r)$.)

b) Funskjonen over regner ut andelen av elementene i arrayet `A` som er mindre enn et standardavvik fra gjennomsnittverdien av tallene i `A` i *prosent*. Lag en ny funksjon som bruker funksjonen `andel` fra deloppgave a) for å regne ut *antall* elementer som er under et standardavvik fra gjennomsnittverdien av tallene i arrayet:

```
int antall(double *A, int n);
```

Du kan benytte denne avrundingsfunksjonen i din løsning:

```
int avrund(double v) {
    return (int)(v + 0.5);
}
```

(Hint: Det er ikke nødvendig å traversere alle elementene i array `A` i funksjonen `antall`.)

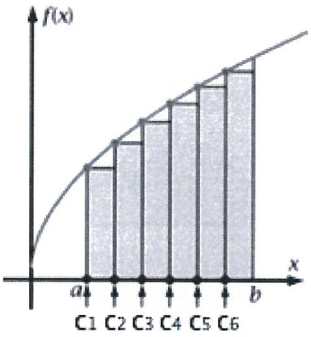
Oppgave 3 – 25%

Riemann Sum er en tilnærming for å regne ut et integral for en funksjon over et gitt intervall. Riemann Sum R_n for en funksjon f for intervallet $[a, b]$ er en sum på formen

$$R_n = \sum_{k=1}^{n} f(c_k)\Delta x_k$$

hvor $[a, b]$ er partisjonert i n subintervall med bredde Δx_k. Vi vil her bruke den samme bredden Δx for alle subintervall (med et eventuelt unntak for siste subintervall). c_k kan velges innenfor hvert subintervall, men i vår løsning bruker vi at c_k er den minste verdien i hvert subintervall (Lower Riemann Sum). Det gir oss følgende verdier for c_k

$$\begin{aligned} c_1 &= a \\ c_2 &= a + \Delta x \\ c_3 &= a + 2\Delta x \\ c_4 &= a + 3\Delta x \\ &\dots \\ c_n &= a + (n-1)\Delta x \end{aligned}$$



Lag en implementasjon av Riemann Sum for kvadratrot hvor bredden på subintervallene er konstante (med et eventuelt unntak for siste subintervall):

```
double riemann_sqrt(double a, double b, double dx);
```

Argumentene `a` og `b` gir oss intervallet, og det siste argumentet `dx` er bredden på subintervallene (Δx). Du benytter denne funksjonen for kvadratrot (f) i din løsning:

```
double sqrt(double x);
```

Oppgave 4 – 30%

I et orienteringsløp skal en deltaker besøke et gitt antall poster i en gitt rekkefølge. På hver post vil et kontrollelement som er unik for posten bli lagd når deltakeren er innom posten. Dette kontrollelementet inneholder en unik kode for posten (`kode`) og hvor lang tid deltakeren har brukt fra forrige post, eller eventuelt fra start hvis dette er første post (`strekktid`):

```
typedef struct kontrollelement kontrollelement_t;
struct kontrollelement {
    int kode;
    double strekktid;
};
```

Kontrollelementet legges til deltakerens liste over poster som er besøkt (`spor`). Mål håndteres på samme måte som alle andre poster (men skal være siste post som registreres). Når deltakeren kommer i mål vil listen av kontrollelement kunne kontroleres mot en fasit-liste hvor postene er plassert i rette rekkefølge (`fasit`).

a) Lag funksjonen som kontrollerer at deltakeren har besøkt alle postene i rett rekkefølge. Funksjonen skal returnere 1 hvis deltakeren har besøkt alle postene i rett rekkefølge og 0 hvis dette ikke er tilfelle:

```
int kontroll(list_t *spor, list_t *fasit);
```

b) Lag en funksjonen som returnere tiden deltakeren har brukt fra start til mål:

```
double tid(list_t *spor);
```

c) For alle deltakere som har besøkt alle postene i rett rekkefølge tar vi vare på deres spor i en ny liste `godkjente` (en liste hvor elementene er lister tilsvarende `spor` ovenfor). Lag en funksjon som tar `godkjente` som argument og finner den deltakeren med beste sammenlagt tid (minste sammenlagte tid). Funksjonen skal returnere denne bestetiden:

```
double bestetid(list_t *godkjente);
```

(Hint: Du kan benytte funksjonen `tid` fra deloppgaven ovenfor.)

Du kan anta at følgende listefunksjoner er tilgjengelige:

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

// Lag en ny listeiterator
list_iterator_t *list_createiterator(list_t *list);

// Frigi listeiterator
void list_destroyiterator(list_iterator_t *iter);

// Returner element som pekes på av iterator og la iterator peke på
// neste element (returner NULL når vi er kommet til enden av lista)
void *list_next(list_iterator_t *iter);
```