

# Eksamenssettet består av 4 oppgaver.

## Eksamenssettet består av 4 oppgaver.

### Eksamenssettet består av 4 oppgaver.

#### Eksamenssettet består av 4 oppgaver.

##### Eksamenssettet består av 4 oppgaver.

###### Eksamenssettet består av 4 oppgaver.

# Eksamenssettet består av 4 oppgaver.

## Eksamenssettet består av 4 oppgaver.

### Eksamenssettet består av 4 oppgaver.

#### Eksamenssettet består av 4 oppgaver.

##### Eksamenssettet består av 4 oppgaver.

###### Eksamenssettet består av 4 oppgaver.

# Eksamenssettet består av 4 oppgaver.

## Eksamenssettet består av 4 oppgaver.

# Eksamenssettet består av 4 oppgaver.

## Eksamenssettet består av 4 oppgaver.

*Eksamenssettet består av 4 oppgaver.*

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

## **Oppgave 1 - 30%**

De fleste av dagens datamaskiner er strukturert i henhold til en modell foreslått av John Von Neumann i 1946. Beskriv denne modellen. Beskrivelsen bør vektlegge hvordan programmer utføres.

## Oppgave 2 - 20%

Gitt  $x$  (ulik 0) som en approksimasjon til kvadratroten av  $n$ , så vil  $(x + n/x)/2$  være en bedre approksimasjon. Dette kan man bruke til å beregne kvadratroten av et tall med ønsket nøyaktighet, ved å begynne med et vilkårlig estimat og deretter forbedre det gjentatte ganger. Skriv en iterativ funksjon *mysqrt*, som beregner kvadratroten av et tall  $n$  med en nøyaktighet på minst 5 desimaler:

```
float mysqrt(float n, float x)
```

Eksempel:

```
x = 20, n = 100, funksjonen krever 5 iterasjoner
iterasjon 1: (20 + 100/20)/2 = 12.5
iterasjon 2: (12.5 + 100/12.5)/2 = 10.25
iterasjon 3: (10.25 + 100/10.25)/2 = 10.01190476
iterasjon 4: (10.01190476 + 100/10.01190476)/2 = 10.0000070777
iterasjon 5: (10.0000070777 + 100/10.0000070777)/2 = 10.0000000000
```

Hint: Den absolutte differansen mellom approksimasjonen i iterasjon 5 og 4 er mindre enn 0.00001.

## Oppgave 3 - 30%

Sieve of Eratosthenes er en algoritme for å finne alle primtall opp til et gitt tall  $n$ . Algoritmen fungerer ved å initiert anta at alle tall  $2..n$  er primtall, for så å systematisk eliminere sammensatte tall. Den kan implementeres som følger:

1. Lag et array  $A$  som kan indekseres fra 2 til  $n$ , hvor alle verdier initiert er satt til 1.
2. Med utgangspunkt i indeks 2, for alle multipler av 2 ( $2 * 2, 2 * 3, osv.$ ) sett verdien med tilsvarende indeks til 0 (slik at  $A[4] = 0, A[6] = 0, ..$ ).
3. Finn den neste indeksen i arrayet hvor verdien er lik 1 (denne indeksen er et primtall).
4. For alle multipler av indeksen du fant i steg 3, sett verdien med tilsvarende indeks til 0.
5. Repeter steg 3 og 4 helt frem til du når en indeks som er større enn kvadratroten av  $n$ .
6. Alle indekser  $i$  hvor  $A[i]$  har verdien 1 er primtall.

Skriv en funksjon *finnprimtall* som returnerer en liste med alle primtall opp til et gitt tall  $n$ :

```
list_t *finnprimtall(int n)
```

Du kan anta at følgende listefunksjoner er tilgjengelige:

```
// 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);
```

## Oppgave 4 - 20%

Alle naturlige tall som ikke er primtall kan faktoriseres i to eller flere primtallsfaktorer. For eksempel, dersom et tall  $n$  kan skrives som et produkt av to primtall  $a$  og  $b$  ( $n = a * b$ ), så er  $a$  og  $b$  primtallsfaktorene til tallet  $n$ . Skriv en funksjon som returnerer en liste med alle primtallsfaktorer til et tall  $n$ :

```
list_t *finnprimtallsfaktorer(int n)
```

Her kan du benytte funksjonen fra oppgave 3 for å finne de primtallene som er potensielle faktorer til  $n$ . Husk at samme primtall kan være faktor flere ganger ( $20 = 5 * 2 * 2$ ). For å sjekke om et primtall  $a$  er en faktor i  $n$  kan du sjekke om resten ved divisjon er 0. I programmeringspråket C gjør du dette ved å bruke % operatoren:

```
if ((n % a) == 0) {  
    // a er en faktor i n  
}
```

Du kan anta at følgende listefunksjoner er tilgjengelige:

```
// 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);  
  
// Lag en ny listeiterator  
list_iterator_t *list_createiterator(list_t *list);  
  
// Returner element som pekes på av iterator og  
// la iterator peke på neste element  
void *list_next(list_iterator_t *iter);
```