NTNU
Det skapende universitet

Institutt for Datateknikk og Informasjonsvitenskap

# Eksamensoppgave i TDT4102 Prosedyre- og objektorientert programmering

**Faglig kontakt under eksamen: Pål Sætrom**

**Tlf.: 98203874**

**Eksamensdato: 16. mai 2013**

**Eksamenstid (fra-til): 09:00-13:00**

**Hjelpemiddelkode/Tillatte hjelpemidler: C**

Walter Savitch, Absolute C++ eller Lyle Loudon, C++ Pocket Reference. (Egne notater i boka er ikke tillatt, men understrekning, utguling, eller kapittelmarkering vha. f.eks. "post-its" er lov.)

Egen utskrift av kapittel 20 ”Patterns and UML” fra Walter Savitch, Absolute C++ (20 sider).

Bestemt, enkel kalkulator tillatt.

**Annen informasjon:**

**Målform/språk: Bokmål**

**Antall sider: 7**

**Antall sider vedlegg: 0**

**Kontrollert av:**

_____

Dato Sign

Merk! Studenter finner sensur i Studentweb. Har du spørsmål om din sensur må du kontakte instituttet ditt. Eksamenskontoret vil ikke kunne svare på slike spørsmål.

## Generell introduksjon

Les gjennom oppgavetekstene nøye. Noen av oppgavene har lengre tekst, men dette er for å gi kontekst, introduksjon og eksempler til oppgaven.

Når det står "implementer" eller "lag" skal du skrive en implementasjon. Hvis det står "vis" eller "forklar" står du fritt i hvordan du svarer, men bruk enkle kodelinjer og/eller korte forklaringer og vær kort og presis. Dersom det er angitt en begrensning i hvor langt et svar kan være vil lengre svar telle negativt. I noen oppgaver er det brukt nummererte linjer for koden slik at det skal være lett å referere til spesifikke linjer. Dersom du mener at opplysninger mangler i en oppgaveformulering, gjør kort rede for de antagelser og forutsetninger som du finner nødvendig.

Hver enkelt oppgave er ikke ment å være mer omfattende enn det som er beskrevet. Noen oppgaver fokuserer bare på enkeltfunksjoner og da er det utelukkende denne funksjonen som er tema. Andre oppgaver er "oppskriftsbasert" og vi spør etter forskjellige deler av en klasse, samarbeidende klasser eller et program. Du kan velge selv om du vil løse dette trinnvis ved å ta del for del, eller om du vil lage en samlet implementasjon. Sørg for at det går tydelig frem hvilke spørsmål du har svart på hvor i koden din.

Det er ikke viktig å huske helt korrekt syntaks for biblioteksfunksjoner. Oppgaven krever ikke kjennskap til andre klasser og funksjoner enn de du har blitt kjent med i øvingsopplegget. All kode skal være i C++.

Hele oppgavesettet er arbeidskrevende og det er ikke foreventet at alle skal klare alt. Tenk strategisk i forhold til ditt nivå og dine ambisjoner! Velg ut deloppgaver som du tror du mestrer og løs disse først. Slå opp i boka kun i nødsfall. All tid du bruker på å lete i boka gir deg mindre tid til å svare på oppgaver. Deloppgavene i de "tematiske" oppgavene er organisert i en logisk rekkefølge, men det betyr IKKE at det er direkte sammenheng mellom vanskelighetsgrad og nummereringen av deloppgavene.

Oppgavene teller med den andelen som er angitt i prosent. Den prosentvise uttellingen for hver oppgave kan/vil likevel bli justert ved sensur basert på hvordan oppgavene har fungert. De enkelte deloppgaver kan også bli tillagt forskjellig vekt.

## Oppgave 1: Kodeforståelse (30%)

**a)** Hva skrives ut av følgende kode?

```
int a = 3;
int b = 4;

if (a == b) {
  cout << "1) a == b = "
       << (a == b) << endl;
} else {
  cout << "1) a != b = "
       << (a != b) << endl;
}

cout << "2) ";
switch (b - a) {
case 0:
  cout << "0 ";
case 1:
  cout << "1 ";
default:
  cout << "all ";
}
cout << endl;

cout << "3) ";
do {
  cout << a-- << " ";
} while (a > 0);
cout << endl;

cout << "4) ";
while (++a < b) {
  cout << a << " ";
}
cout << endl;
```

```
int c[] = {0, 2, 4, 6, 8};
cout << "5) " << c[0]
     << ", " << *c << endl;
cout << "6) " << ((c + 1) == &(c[1]))
     << endl;

int s = 0;
for (int i = 0; i < 5; i++)
  s += c[i];
cout << "7) " << s << endl;
cout << "8) " << ((int)&b == *(c + 2))
     << endl;

double d = 3.5;
int e = 3;
cout << "9) " << d * e
     << ", " << d / e << endl;
cout << "10) " << static_cast<int>(d) * e
     << ", " << static_cast<int>(d) / e
     << endl;

cout << "11)";
int f = 5;
while (f > 0) {
  if (f = 1) {
    cout << " (last round)";
  }
  cout << " " << f-- % e;
}
cout << endl;
```

**b)** Funksjonen **`ArrayAlloc`** skal opprette et array av **`int`** med en gitt størrelse **`n`** og initialisere alle elementene i arrayet til en gitt verdi **`v`**. Følgende kode er et første forsøk på å implementere **`ArrayAlloc`**:

```
1: int* ArrayAlloc(int n, int v) {
2:   int ret[n];
3:   for (int i = 0; i < n; i++)
4:     ret[i] = v;
5:   return ret;
6: }
```

**1)** Forklar med en setning (maks 20 ord) hva som er feil med denne implementasjonen.

**2)** Vis hvilke endringer som må gjøres i koden for å få riktig oppførsel.

**3)** Generaliser implementasjonen av **`ArrayAlloc`** slik at den nye versjonen av funksjonen kan opprette arrays av vilkårlige datatyper.

**4)** Forklar kort hva din generaliserte implementasjon av **`ArrayAlloc`** forutsetter når det gjelder de datatypene **`ArrayAlloc`** skal kunne håndtere.

**c)** Klassen **`Tree`** under brukes for å lage et binært tre. I klassedefinisjonen under er det deklarert en konstruktor og to medlemsfunksjoner. Implementasjonen av den ene medlemsfunksjonen er gitt under klassedefinisjonen.

```
1:    class Tree {
2:      Tree *_left;
3:      Tree *_right;
4:      string _name;
5:    public:
6:      Tree(string name) : _left(NULL), _right(NULL), _name(name) {}
7:      void insert(string name);
8:      void insert(const Tree &tree);
9:      friend ostream& operator<<(ostream&, const Tree&);
10:   };
11:
12:   void Tree::insert(string name) {
13:     Tree **destination = &_right;
14:     if (name < _name) {
15:       destination = &_left;
16:     }
17:     if (*destination == NULL)
18:       (*destination) = new Tree(name);
19:     else
20:       (*destination)->insert(name);
21:   }
```

**1)** Forklar med en setning (maks 20 ord) hva medlemsfunksjonen **`insert(string name)`** gjør.

**2)** Tegn hvordan treet blir seende ut etter følgende kodesnutt (bruk bokser med tekst for noder og piler for pekere):

```
1:    Tree t("m");
2:    t.insert("a");
3:    t.insert("c");
4:    t.insert("g");
5:    t.insert("v");
6:    t.insert("b");
7:    t.insert("a");
8:    t.insert("h");
```

**3)** Deklarasjonen av medlemsfunksjonen **`insert(string name)`** er ikke optimal. Forklar kort hvorfor og skriv en ny og forbedret deklarasjon av **`insert`**.

**4)** Forklar kort hvorfor klassedefinisjonen av **`Tree`** over inneholder setningen på linje 9 (starter med **`friend`**).

**5)** Operatoren **`operator<<(ostream&, const Tree&)`** skal skrive ut innholdet i treet oppgitt som argument (dvs. medlemsvariabelen **`_name`**) i noderekkefølge (”in-order”). Innholdet fra hver node skal være separert med komma og mellomrom (”, ”). Følgende implementasjon er et feilet forsøk på å få til dette. Forklar kort hva denne implementasjonen av operatoren skriver ut. Vis hvordan denne operatoren kan brukes for å skrive ut innholdet av treet **`t`** fra deloppgave **2)** og vis hva som skrives ut.

```
1:    ostream& operator<<(ostream &out, const Tree &tree) {
2:      out << tree._name;
3:      if (tree._left != NULL)
4:        out << ", " << *tree._left;
5:      if (tree._right != NULL)
6:        out << ", " << *tree._right;
7:      return out;
8:    }
```

**6)** Lag en korrekt implementasjon av **`operator<<(ostream&, const Tree&)`**.

**7)** Medlemsfunksjonen **`insert(const Tree &tree)`** skal sette inn innholdet i argumentet **`tree`** inn i treet funksjonen blir kalt opp på; se et eksempel på hvordan funksjonen skal kunne brukes under. Implementer denne medlemsfunksjonen.

```
1:    Tree t1("a");
2:    Tree t2("b");
3:    t1.insert(t2);
```

**8)** Den nåværende klassedefinisjonen og implementasjonen av **`Tree`** har en stor og kritisk mangel. Forklar kort hvilken mangel dette er og hvorfor denne er kritisk og vis hvordan denne mangelen kan utbedres (Hint: hvilke funksjoner blir automatisk definert og implementert av kompilatoren?).

**9)** Lag en generalisert klassedefinisjon av **`Tree`** slik at den nye versjonen kan håndtere innhold av en vilkårlig datatype. Denne generaliserte klassedefinisjonen skal inneholde to medlemsfunksjoner som tilsvarer de to i **`Tree`** for å sette inn et element (**`void insert(string name)`**) og et tre (**`void insert(const Tree &tree)`**).

**10)** Implementer det som vil tilsvare medlemsfunksjonen **`void insert(string name)`** i ditt generaliserte **`Tree`**.

**11)** Forklar kort hva ditt generaliserte **`Tree`** forutsetter når det gjelder de datatypene **`Tree`** skal kunne håndtere.

## Oppgave 2: Funksjoner (25%)

**a)** Implementer funksjonen **`int ceil(double d)`** som runder av et flyttall opp til nærmeste heltall (returnerer en **`int`**). Funksjonen skal også håndtere negative tall. *Eksempler: **`ceil(2.5)`** gir 3, **`ceil(1.01)`** gir 2, **`ceil(-2.5)`** gir -2.*

**b)** Implementer funksjonen **`int* rotate(int *array, unsigned int size)`** som roterer innholdet i heltallsarrayet **`array`** ved å flytte alle tall i arrayet et steg til høyre og siste element i arrayet fram til første posisjon i arrayet. *Eksempler: Gitt arrayet **`int a[] = {1, 3, 2, 5}`**, så skal innholdet i **`a`** etter et kall til **`rotate(a, 4)`** være **`{5, 1, 3, 2}`**.*

**c)** Implementer funksjonen **`int* rotateN(int *array, unsigned int size, unsigned int n)`** som roterer innholdet i heltallsarrayet **`array`** ved å flytte alle tall i arrayet **`n`** steg til høyre. *Eksempler: Gitt arrayet **`int a[] = {1, 3, 2, 5}`**, så skal innholdet i **`a`** etter et kall til **`rotateN(a, 4, 1)`** være **`{5, 1, 3, 2}`**. Likeledes, gitt samme array (**`int a[] = {1, 3, 2, 5}`**) så skal innholdet etter **`rotateN(a, 4, 2)`** være **`{2, 5, 1, 3}`**, etter **`rotateN(a, 4, 3)`** være **`{3, 2, 5, 1}`**, etter **`rotateN(a, 4, 4)`** være **`{1, 3, 2, 5}`**, og etter **`rotateN(a, 4, 1000001)`** være **`{5, 1, 3, 2}`**.* Merk at ***`rotateN`*** kan implementeres enkelt ved gjenbruk fra **b)**, men en slik løsning vil ikke være spesielt effektiv. (Hint: hvor lang tid vil kallet ***`rotateN(a, 4, 1000001)`*** ta?). Vi er her på jakt etter en effektiv løsning.

**d)** Implementer funksjonen **`int* rotateLeft(int *array, unsigned int size)`** som roterer innholdet i heltallsarrayet **`array`** ved å flytte alle tall i arrayet et steg til venstre og første element i arrayet tilbake til siste posisjon i arrayet. *Eksempler: Gitt arrayet **`int a[] = {1, 3, 2, 5}`**, så skal innholdet i **`a`** etter et kall til **`rotateLeft(a, 4)`** være **`{3, 2, 5, 1}`**.*

**e)** Implementer funksjonen **`range`** som skal finne og returnere minimum og maksimum i et **`double`** array (*Eks: I **`double d[] = {1.1, -3.0, 2.0, 5.1}`** er **`-3.0`** minimum og **`5.1`** maksimum*). Funksjonen **`range`** skal ta arrayet som et argument, men du står ellers fritt til å definere funksjonens returtype og eventuelt andre parametre.

**f)** Utvid løsningen din fra **e)** slik at **`range`** kan finne og returnere minimum og maksimum i arrays av følgende datatyper: **`int`**, **`string`**, **`char`** og **`char*`** (c-strenger).

### Oppgave 3: Minnehåndtering og klasser (25%)

Polynomer er uttrykk av typen $4x^3 + 2x + 1$, $7x^{1001} + 2x^{501}$ og $x^2 - 2x + 1$. Alle polynom med en variabel $x$ kan representeres på den generelle formen $a_nx^n + a_{n-1}x^{n-1} + ... + a_1x + a_0$, hvor $n$ er et positivt heltall og $a_n, a_{n-1}, ... a_1, a_0$ er konstante koeffisienter. Vi kan dermed representere alle polynom med en variabel ved å bruke et array av koeffisienter (*Eks: $4x^3 + 2x + 1$ kan representeres med **`{4, 0, 2, 1}`** og $x^2 - 2x + 1$ med **`{1, -2, 1}`***).

Følgende er en mulig definisjon av en klasse for å representere polynom:

```
1:   class Polynomial {
2:     double *_coefficients;
3:     unsigned int _n;
4:   public:
5:     Polynomial(unsigned int n);
6:     Polynomial(const double *coefficients, unsigned int size);
7:     Polynomial(const Polynomial &pol);
8:     ~Polynomial();
9:     const Polynomial operator-() const;
10:    const Polynomial operator+(const Polynomial &pol) const;
11:    const Polynomial operator-(const Polynomial &pol) const;
12:    Polynomial& operator=(const Polynomial &pol);
13:    double compute(double x) const;
14:  };
```

I denne definisjonen skal medlemsvariabelen **`_coefficients`** lagre polynomets koeffisienter. Medlemsvariabelen **`_n`** skal holde styr på antall koeffisienter.

**a)** Konstruktoren **`Polynomial(unsigned int n)`** skal opprette et polynom av grad **`n`** (dvs. at polynomet skal ha **`n`** + 1 koeffisienter). Konstruktoren skal også initialisere alle koeffisientene til **`0`**. Implementer denne konstruktoren og forklar med en setning hvorfor parameteren **`n`** har datatypen **`unsigned int`**.

**b)** Konstruktoren **`Polynomial(const double *coefficients, unsigned int size)`** skal opprette et polynom som består av koeffisientene i arrayet **`coefficients`**. Parameteren **`size`** angir størrelsen på arrayet. Implementer denne konstruktoren slik at du alltid vil kunne opprette et lovlig polynom.

**c)** Implementer klassens kopikonstruktor og destruktor.

**d)** Operatorene **`operator+`** og **`operator-`** skal henholdsvis addere og subtrahere to polynom og returnere resultatet. Implementer disse to operatorene.

**e)** Medlemsfunksjonen **`double compute(double x) const`** skal beregne verdien av polynomet for en gitt verdi av polynomets variabel **`x`**. (*Eks: Gitt **`double d[] = {2.0,`***

*0}* er ***Polynomial(d, 2).compute(0) == 0*** *mens* ***Polynomial(d, 2).compute(2) == 4***). Implementer **`compute`**.

## Oppgave 4: Arv, strømmer og STL (20%)

I denne oppgaven skal du lage to funksjoner **`computeTotalArea`** og **`computeTotalPerimeter`** som beregner og returnerer henholdsvis totalt areal og total omkrets av en samling av ulike sirkler og rektangler. Anta at samlingen av disse sirklene og rektanglene ligger lagret i en **`vector`** fra STL og at dette er input til både **`computeTotalArea`** og **`computeTotalPerimeter`**.

**a)** Vis hvordan dette problemet kan løses ved hjelp av arv og implementer **`computeTotalArea`** og **`computeTotalPerimeter`** basert på din løsning. (Husk at areal og omkrets til en sirkel med radius *r* er $\pi r^2$ og $2\pi r$ og areal og omkrets til et rektangel med lenge *l* og bredde *b* er *lb* og 2(*l*+*b*).)

**b)** Du skal nå lage en funksjon **`readShapes(const char *filename)`** som leser inn samlingen av sirkler og rektangler fra filen gitt av **`filename`**. I filen ligger informasjonen om hvert geometriske objekt på en egen linje. Linjer med sirkler inneholder et flyttall (radiusen til sirkelen) mens linjer med rektangler inneholder to flyttall (rektangelets lenge og bredde) separert med et mellomrom (”space”). Funksjonen **`readShapes`** skal returnere en **`vector`** som skal kunne brukes som argument til **`computeTotalArea`** og **`computeTotalPerimeter`**. Implementer **`readShapes`**.
*Eksempel: Blokken under viser en fil som beskriver en samling av 2 sirkler og 1 rektangel. Sirklene har radius* ***`2.5`*** *og* ***`3.1`*** *mens rektangelet har lengde* ***`0.1`*** *og bredde* ***`10`****.*

```
2.5
0.1 10
3.1
```