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: 17. august 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: 8

Antall sider vedlegg: 0

Kontrollert av:

_____

Dato Sign

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 = 1, b = 1;
a = b;
a += b;
cout << "1) " << a << endl;

a = b = 1;
int *ap = &a;
int *bp = &b;
ap = bp;
*ap += *bp;
cout << "2) " << a << endl;
cout << "3) " << *ap << endl;

int c[] = {0, 2, 0, 6, 8};
cout << "4) " << c[4] << endl;

cout << "5)";
for (int i = 4; i > 0; --i)
  cout << " " << c[i];
cout << endl;

cout << "6)";
for (int *cp = c + 4; *cp > 0; --cp)
  cout << " " << *cp;
cout << endl;
```

```
cout << "7)";
for (int i = 0; i < 4; i++) {
  if (i == 0) {
    cout << " a";
  } else if (i == 1) {
    cout << " b";
  } else if (i == 2) {
    cout << " c";
    break;
  } else {
    cout << " x";
  }
  cout << ";";
}
cout << endl;

cout << "8)";
for (int i = 0; i < 4; i++) {
  switch (i) {
  case 0:
    cout << " a";
  case 1:
    cout << " b";
  case 2:
    cout << " c";
    break;
  default:
    cout << " x";
  }
  cout << ";";
}
cout << endl;
```

```
int d = 13, e = 3;
cout << "9) " << 13 / 3 << ", " << 13 % 3 << endl;
cout << "10) " << static_cast<float>(13 / 3)
     << ", " << static_cast<float>(13 % 3) << endl;

float f = 12 / 3 ? 1.0 : 0.1;
cout << "11) " << 12 / f-- << endl;
cout << "12) " << 12 / --f << endl;
```

b) Funksjonen **`MyPrint`** skal skrive ut innholdet av et array **`a`** av **`int`** med en gitt størrelse **`s`** til **`cout`** slik at alle elementene i arrayet kommer på samme linje og hvert element er etterfulgt av komma og mellomrom (”`, `”). *Eksempel: Gitt arrayet **`int a[] = {1, 3, 2, 5}`** så skal **`MyPrint(a, 4)`** skrive følgende streng til cout: ”`1, 3, 2, 5, `”.*
Følgende kode er et første forsøk på å implementere ***`MyPrint`***:

```
1: void MyPrint(const int *a, unsigned int s) {
2:   for (int i = 0; i <= s; ++i)
3:     cout << a[i] << ", ";
4: }
```

1) Er implementasjonen korrekt? Forklar kort hvorfor/hvorfor ikke og korriger koden dersom den er feil.

2) Generaliser implementasjonen av **`MyPrint`** slik at den nye versjonen av funksjonen kan skrive ut arrays av vilkårlig datatype.

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

c) Følgende kode er et andre forsøk på å implementere ***`MyPrint`***:

```
1:   void MyPrint(const int *a, unsigned int s) {
2:     if (s == 0)
3:       return;
4:     cout << *a << ", ";
5:     MyPrint(a + 1, s - 1);
6:   }
```

1) Hvilken teknikk er brukt i denne implementasjonen?

2) Er implementasjonen korrekt? Forklar kort hvorfor/hvorfor ikke og korriger koden dersom den er feil.

d) Gjør en minimal endring i implementasjonen av ***`MyPrint`*** fra **1c)** slik at den ikke lenger skriver ut `", "` etter siste element i arrayet.

e) Gjør en minimal endring i implementasjonen av ***`MyPrint`*** fra **1d)** slik at den nå skriver ut innholdet av arrayet baklengs. *Eksempel: Gitt arrayet* ***`int a[] = {1, 3, 2, 5}`*** *så skal* ***`MyPrint(a, 4)`*** *nå skrive følgende streng til cout:* ***`"5, 2, 3, 1"`***.

f) **1)** Vis hva som skrives ut av følgende kode og **2)** vis hvordan kall-stakken ser ut når unntaket blir kastet:

```
1:    void thrower() {
2:      throw "Exception";
3:    }
4:    void caller() {
5:      cout << "1";
6:      thrower();
7:      cout << "2";
8:    }
9:    int main() {
10:     try {
11:       cout << "1";
12:       caller();
13:       cout << "2";
14:     } catch (const char *&e) {
15:       cout << e;
16:     } catch (...) {
17:       cout << "All";
18:     }
19:     cout << endl;
20:     return 0;
21:   }
```

g) Hva skrives ut av følgende kode:

```
1:    struct A {
2:      int _a;
3:      A(int a) : _a(a) {}
4:      virtual int value() const { return _a; }
5:    };
6:    struct B : public A {
7:      int _b;
8:      B(int a, int b) : A(a), _b(b) {}
9:      int value() const { return _a + _b; }
10:   };
11:   int main() {
12:     A el1(1);
13:     B el2(2, 1);
14:     vector<A> asAndBs;
15:     asAndBs.push_back(el1);
16:     asAndBs.push_back(el2);
17:     for (vector<A>::iterator it = asAndBs.begin();
18:          it != asAndBs.end(); ++it)
19:       cout << it->value() << ", ";
20:     cout << endl;
21:     vector<A*> asAndBs2;
22:     asAndBs2.push_back(&el1);
23:     asAndBs2.push_back(&el2);
24:     for (vector<A*>::iterator it = asAndBs2.begin();
25:          it != asAndBs2.end(); ++it)
26:       cout << (*it)->value() << ", ";
27:     cout << endl;
28:   }
```

h) Hva (hvilken datatype) returneres fra følgende funksjoner:

```
1:    void f1() {
2:      cout << 1;
3:    }
4:    void f2() {
5:      cout << 2.0;
6:    }
7:    int f3() {
8:      return 3.0;
9:    }
10:   double f4() {
11:     return 4;
12:   }
```

Oppgave 2: Funksjoner (25%)

a) Deklarer følgende funksjoner:

1) **`jump`**, som tar inn to flyttall (**`angle`** og **`speed`**) og returnerer et flyttall (lengden som hoppes).
2) **`sorted`**, som tar inn et array av heltall og et heltall (**`array`** og **`size`**) og returnerer en boolsk verdi (angir om arrayet er sortert).
3) **`swap`**, som tar inn to flyttallsreferanser (**`a`** og **`b`**) og ikke returnerer noen verdi.

4) **countIf**, som tar inn en konstant **vector** av **string** referanse og en konstant **string** referanse (**array** og **comp**) og returnerer et heltall (antall ganger **comp** forekommer i **array**).
5) **max**, som tar inn to konstante referanser av en vilkårlig datatype (**a** og **b**) og returnerer en verdi av den samme datatypen (den største av de to verdiene).

b) Implementer funksjonen **bool geq(double a, double b)** som sammenligner tall **a** med tall **b** og returnerer **true** hvis **a** er større eller lik **b**. *Eksempler:* ***geq(2.5, 2.5)*** *gir* ***true****,* ***geq(2.5, -1.5)*** *gir* ***true****,* ***geq(-2.5, -1.5)*** *gir* ***false****.*

c) Implementer funksjonen **bool geq(const char *a, const char *b)** som sammenligner c-streng **a** med c-streng **b** og returnerer **true** hvis **a** er større eller lik **b**. *Eksempler:* ***geq("all", "all")*** *gir* ***true****,* ***geq("alle", "all")*** *gir* ***true****,* ***geq("alb", "all")*** *gir* ***false****.* NB! Funksjonen skal implementeres uten å bruke **strcmp** og **strncmp**.

d) Implementer funksjonen **largestIndex** som skal finne og returnere indeksen til den største verdien i en **vector** av **double** eller c-strenger (**char***). (*Eks: I* ***const char* d[] = {"AB", "BC", "CD", "DE"}*** *er 3 indeksen til den største strengen "**DE**"*). Funksjonen **largestIndex** skal ta vektoren som et argument, men du står ellers fritt til å definere funksjonens returtype og eventuelt andre parametre.

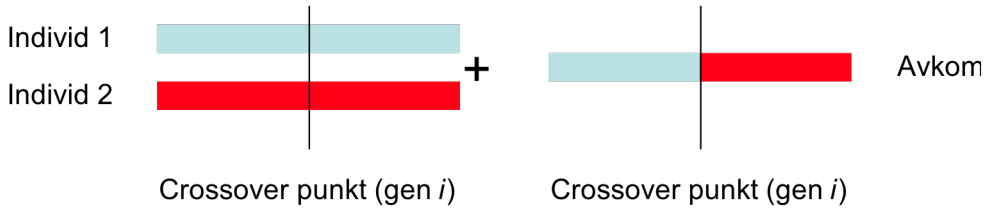
e) Implementer funksjonen **readNumbers** som skal lese inn flyttall fra en **istream** og legge tallene inn i en **vector**. Anta at det ligger kun et tall per linje i strømmen. (*Eks: Dersom innholdet i strømmen er "**1.2\n1\n-1.4\n**" så skal vektoren inneholde tallene **1.2, 1.0, og -1.4**.*). Funksjonen **readNumbers** skal ta strømmen som et argument, men du står ellers fritt til å definere funksjonens returtype og eventuelt andre parametre.

f) Lag et program som leser inn tall fra en fil og skriver ut det største tallet i filen. Programmet skal lese inn navnet på filen fra **cin**. Anta at filen har samme format som i **2e)**.

Oppgave 3: Genetisk algoritme – Minnehåndtering og klasser (25%)

I denne oppgaven og neste oppgave (Oppgave 4) skal du lage et system for å simulere evolusjon – en såkalt "genetisk algoritme". En genetisk algoritme består av en *populasjon* av kunstige *individ*, en *fitness-funksjon* som evaluerer hvor godt et gitt individ er til å løse et gitt problem, og en *seleksjons-mekanisme* som oppdaterer populasjonen basert på individenes *fitness*. Hvert individ har et *genom* som består av et bestemt antall *gener* og det er verdiene av disse genene som bestemmer hvor god individet er til å løse problemet (hvor god *fitness* individet har). Vi skal her bruke et array av flyttall mellom 0 og 1 for å representere disse genene.

To individ kan lage et nytt individ ved å *kombinere* sine gener. Denne kombinasjonen ("crossover") gjøres ved at vi først velger et tilfeldig gen i. Vi kopierer så alle genene som ligger før dette genet fra individ 1 og inn i det nye individet. Til slutt kopierer vi alle genene som ligger fra og med genet fra individ 2 og inn i det nye individet. Denne kombinasjonsoperasjonen er illustrert under:



Følgende er en mulig definisjon av en klasse for å representere individ i en genetisk algoritme:

```
1:   class Individual {
2:     unsigned int _geneCount;
3:     double *_genes;
4:   public:
5:     Individual(unsigned int geneCount);
6:     Individual(const Individual &ind);
7:     ~Individual();
8:     Individual& operator=(const Individual &ind);
9:     const Individual operator+(const Individual &ind) const;
10:    friend ostream& operator<<(ostream& o, const Individual &ind);
11:    unsigned int getGeneCount() const { return _geneCount; }
12:    double getGene(unsigned int id) const;
13:  };
```

a) Konstruktoren **`Individual(unsigned int geneCount)`** skal opprette et individ med **`n`** gener. Konstruktoren skal også initialisere alle genene til tilfeldige tall mellom 0 og 1. Implementer denne konstruktoren.

b) Implementer klassens tilordningsoperator og destruktor.

c) Funksjonen **`double getGene(unsigned int id) const`** skal returnere verdien til gen nummer **`id`**. Implementer denne funksjonen slik at den vil håndtere alle mulige verdier av argumentet.

d) Hva betyr ordet **`const`** i deklarasjonen til funksjonen **`getGene`**?

e) Operatorene **`operator+`** skal brukes for å kombinerer to individ (crossover). Implementer denne operatoren. NB! Du må også finne en måte å håndtere at to individ kan ha ulikt antall gener.

Oppgave 4: Genetisk algoritme – Klasser og arv (20%)

I denne oppgaven skal du bygge ferdig den genetiske algoritmen du begynte på i forrige oppgave (Oppgave 3). Systemet skal bestå av en klasse for å representere populasjonen, ulike måter for å beregne *fitness*, og en *seleksjons-mekanisme*. Følgende er en mulig definisjon av en klasse for å representere populasjonen i genetisk algoritme (merk at eventuelle medlemsvariable ikke er tatt med i definisjonen):

```
1:   class Population {
2:   public:
3:     Population(unsigned int size, unsigned int geneCount);
4:     void update(const Fitness &scorer);
5:     const Individual best(const Fitness &scorer) const;
6:   };
7:
```

a) Konstruktoren **`Population(unsigned int size, unsigned int geneCount)`** skal opprette en populasjon av **`size`** individ (**`Individual`**) og hvert individ skal ha **`geneCount`** gener (se klassedefinisjonen i Oppgave 3). Oppdater klassedefinisjonen med nødvendige medlemsvariable og implementer konstruktoren. Implementer destruktor, kopikonstruktor og tilordningsoperator dersom det er nødvendig.

Funksjonen **`void update(const Fitness &scorer)`** implementerer *seleksjons-mekanismen*, mens funksjonens parameter **`scorer`** er et objekt av en klasse som implementerer en måte å beregne *fitness*. Fitness klassen har medlemsfunksjonen **`double score(const Individual &ind) const`** og det er denne funksjonen **`update`** bruker for å beregne *fitness* til et gitt individ.

b) En enkel seleksjonsmekanisme, kalt "*turnering"*, kan beskrives som følgende algoritme:

i. Velg *n* tilfeldige individ fra populasjonen. Beregn *fitness* til hvert av disse *n* individene og behold det individet med høyest fitness. Dette er **`foreldre1`**.
ii. Velg *n* tilfeldige individ fra populasjonen. Beregn *fitness* til hvert av disse *n* individene og behold det individet med høyest fitness. Dette er **`foreldre2`**.
iii. Velg *n* tilfeldige individ fra populasjonen. Beregn *fitness* til hvert av disse *n* individene og bytt ut det individet som har lavest fitness med det individet du får ved å kombinere **`foreldre1`** og **`foreldre2`** (**`foreldre1 + foreldre2`**).

Implementer medlemsfunksjonen **`update`** slik at den bruker denne *turnering* seleksjonsalgoritmen.

Systemet ditt skal kunne håndtere følgende tre måter å beregne *fitness*:

i. *Largest*: Genene representerer størrelsen til individene og det individet som er størst har best fitness. Med andre ord: *fitness* for et gitt individ er summen av flyttallsverdiene til individets gener (**`_genes`** i **`Individual`** fra Oppgave 3).
ii. *Smallest*: Omvendt av *Largest.*
iii. *FirstBest*: *Fitness* er bestemt av verdien til gen 0 slik at høy verdi gir høy fitness.

c) Implementer disse tre *fitness*-strategiene (*Largest*, *Smallest*, *FirstBest*) og lag et program som leser inn populasjonsstørrelse og antall gener fra kommandolinjen (**`cin`**) og for hver av de tre *fitness*-strategiene kjører 1000 runder med seleksjon (**`update`**) og skriver ut det individet med best *fitness* (medlemsfunksjonen **`const Individual best(const Fitness &scorer) const`** i **`Population`** skal finne individet med best *fitness*).