



Institutt for datateknikk
og informasjonsvitenskap

Kontinuasjoneksamen i TDT4102 - Prosedyre- og objektorientert programmering

Faglig kontakt under eksamen: Trond Aalberg

Tlf: 97631088

Eksamendato: 12. august 2015

Eksamenstid: 15-19

Hjelpemiddelkode: C: Spesifiserte trykte og håndskrevne hjelpemidler tillatt.
Walter Savitch, Absolute C++ eller
Kyle Loudon, C++ Pocket Reference.

Målform/språk: Bokmål / Nynorsk

Antall sider: 6 sider inklusive forside

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 oppgavene.

Når det står “*implementer*” eller “*lag*” skal du skrive en fungerende implementasjon: hvis det handler om en funksjon skal du skrive deklarasjonen med returtype og parametertype(r) og hele funksjons-kroppen.

Når det står “*deklarer*” er vi kun interessert i funksjons- eller klassedeklarasjonen. Typisk vil dette være deklarasjoner du vanligvis finner i header-filer.

Hvis det står “*vis*” eller “*forklar*” står du fritt i hvordan du svarer, men bruk enkle kodelinjer og/eller korte tekst-forklaringer og vær kort og presis.

Dersom du mener at opplysninger mangler i en oppgaveformulering, gjør kort rede for de antagelser og forutsetninger du finner nødvendig.

All kode skal være i C++. Det er ikke viktig å huske helt korrekt syntaks for bibliotekfunksjoner. Oppgaven krever ikke kjennskap til andre klasser og funksjoner enn de du har blitt godt kjent med i øvingsopplegget. Det er ikke nødvendig å bruke navnerom, ha med include-statement eller vise hvordan koden skal lagres i filer.

Hele oppgavesettet er arbeidskrevende og det er ikke forventet at alle skal klare alt. Tenk strategisk i forhold til ditt nivå og dine ambisjoner! Husk at tid du bruker på å lete i boka gir deg mindre tid til å svare på oppgaver. Det er ikke noen systematisk sammenheng mellom vanskelighetsgrad og nummerering av deloppgavene.

Hoveddelene av eksamensoppgaven teller med den andelen som er angitt i prosent. De enkelte deloppgaver vil bli tillagt forskjellig vekt basert på vanskelighetsgrad og arbeidsmengde.

Oppgave 1: Kodeforståelse og funksjoner (40%)

a) Hva skrives ut av følgende kode?

```
int a = 4;
int b = 9;
cout << "1) = " << a / b << endl;

int c = 6;
int d = 4;
cout << "2) = " << c % d << endl;

cout << "3) = " << static_cast<int>(11.9/4) << endl;

bool e = true;
bool f = false;
cout << "4) = " << boolalpha << !(e && f) << endl;
//boolalph is a modifier that causes a boolean value to be
//printed as the string "true" or "false"

int g = 8;
int h = ++g;
cout << "5) = " << g++ << ", " << h << endl;

int* i = new int(5);
int* j = new int(10);
swap(i, j);
cout << "6) = " << *j << endl;

int* k = new int(5);
int* l = new int(10);
swap(*k, *l);
cout << "7) = " << *l << endl;
```

b) Hva er feil med følgende funksjon og hvordan kan den implementeres (gitt at header skal beholdes som den er)?

```
char* createAlphabet(char start, int length) {
    char alphabet[length];
    for (int i = 0; i < length; i++) {
        alphabet[i] = start++;
    }
    return alphabet;
}
```

c) Implementer funksjonen `bool isAnagram(const string& a, const string& b)` som sjekker om to tekststrenger er anagram av hverandre. Anagram er et ord, navn eller uttrykk som er blitt satt sammen ved å stokke rundt på bokstavene i et annet ord eller uttrykk (man ser vanligvis bort fra mellomrom og tegnsetting mellom bokstavene). Eksempler: "Kama Sutra" = "Austmarka", "Rivaliserende" = "Avleirende ris", "William Shakespeare" = "I am a weakish speller".

*Dette kan gjøres ved å lage seg kopier av strengene hvor du har fjernet tegn som ikke er bokstaver f.eks. med hjelp av biblioteksfunksjonen **isalpha** og har konvertert til små bokstaver med **tolower**. Et string-objekt er egentlig bare en sekvens av bokstaver og bokstavene kan sorteres f.eks. ved funksjonskallet **sort(str.begin(), str.end())**. Deretter kan du sammenligne for å sjekke om string-argumentene *a* og *b* er anagrammer av hverandre.*

- d) Implementer funksjonen `bool isPalindrome(const string& str)` som sjekker om en tekststreng er et palindrom. Et palindrom er et ord, uttrykk eller tall som gir samme resultat enten det leses fra høyre eller venstre (man ser vanligvis bort fra mellomrom og tegnsetting mellom bokstavene). Eksempler på palindrom er: “dvd”, “rør”, “redder”, “trenert”, “Agnes i senga.”, “Skal varg ete gravlaks?”.
- e) Implementer en funksjon `void printWordStatistics(const string& filename)` som leser ei tekstfil og skriver ut (til `cout`) ei sortert liste av alle unike ord sammen med antallet forekomster av ordet i fila. Eksempel:
- ```
alt: 4
bra: 2
det: 3
er: 6
```
- f) Implementer funksjonen `double mean(double arr[], int size)` som regner ut det aritmetiske gjennomsnittet av verdiene i tabell `arr`. Det aritmetiske gjennomsnittet finner du ved å dele summen av alle verdiene med antallet verdier. Parameteren `size` er størrelsen på tabellen (antallet verdier i tabellen).
- g) Implementer funksjon `double median(int arr[], int size)` som finner medianen for verdiene i den sorterte tabellen `arr` (du trenger med andre ord ikke bekymre deg om sorteringen). I statistikk er median den verdien som ligger i midten av en sortert serie med verdier. Hvis antallet verdier er et oddetall, er medianen den midterste verdien. Hvis mengden av verdier er et partall er medianen gjennomsnittet av de to midterste verdiene.
- h) Implementer funksjonen `double round(double num, int precision)` som skal returnere verdien `num` avrundet til antallet desimaler angitt med `precision`. Eksempelvis skal `round(0.886, 2)` gi `0.89` og `round(-0.788, 1)` skal gi `-0.8`. Funksjonen under viser hvordan et flyttall kan avrundes til et heltall (med riktig avrunding) og denne kan du basere deg på i løsningen din (denne er det basert på implisitt kasting pga. returtypen).

```
int round(double num){
 if (num >= 0.0){
 return num + 0.5;
 }else{
 return num - 0.5;
 }
}
```

- i) I medlemsfunksjonen `func` i klassen `Foo` under er nøkkelordet `const` brukt på tre plasser. Forklar hva effekten og hensikten er for hver enkelt av disse.

```
class Foo{
 private:
 set<string> dummyvar;
 public:
 const string& func (const string& p) const;
};
```

- j) I samme funksjon (medlemsfunksjonen `func` i oppgaven over) er tegnet `&` brukt to plasser. Hva spesifiserer dette tegnet og forklar effekt og hensikt.

## Oppgave 2: Lenket liste av dynamisk allokerede tabeller (40%)

I denne oppgaven skal du lage en objektorientert datastruktur som ligner litt på en todimensjonal array av `int`, men hvor du dynamisk kan legge til nye rader av forskjellig lengde (mens programmet kjører). Datastrukturen skal være basert på en dobbelt-lenket liste hvor nodene er objekter med hver sin dynamisk allokeret tabell. Det er ikke lov å bruke `vector` eller andre klasser fra biblioteket i denne oppgaven.

En foreløpig deklarasjon av klassene `DynamicMultiArray` og `RowNode` er vist under, men du må selv bestemme om det er behov for andre variabler eller funksjoner samt legge til funksjonene vi ber om i deloppgavene).

```
class DynamicMultiArray{
private:
 RowNode *first;
 RowNode *last;
public:
 DynamicMultiArray();
 ~DynamicMultiArray();
 void add(int size);
 void remove(int row);
 int get(int row, int column);
 void set(int row, int column, int val);
};

class RowNode{
private:
 int *arr;
 RowNode* next;
 RowNode* prev;
 int size;
public:
 RowNode(int size);
 ~RowNode();
 friend class DynamicMultiArray;
};
```

Hver rad i tabellen er en instans av klassen `RowNode`. Denne klassen har de klassiske `next`- og `prev` (ious)-peker-medlemsvariablene som brukes for å implementere en dobbelt-lenket liste. `next` skal peke til rad-noden som kommer etter og `prev` skal peke til rad-noden som kommer før. Siden radene skal være dynamisk allokerede tabeller har vi også en peker `arr` som skal peke til den dynamisk allokerede tabellen hvor dataene i raden skal lagres. I variabelen `size` lagres vi størrelsen på raden.

- Implementer konstruktøren til `RowNode`. Konstruktøren skal sørge for at det allokeres minne til en `int`-array av størrelsen `size`. Vis bruk av initialiseringsliste.
- Implementer destruktøren til `RowNode`.
- Hva betyr følgende linje i `RowNode`: `friend class DynamicMultiArray` og hvilken nytteverdi har dette i implementasjonen av denne datastrukturen?

Klassen `DynamicMultiArray` er den overordnede klassen for den lenkede listen og den eneste klassen brukere (den som benytter `DynamicMultiArray` i et program) trenger å kjenne til. Medlemsvariablene `first` og `last` peker hhv. til første og siste node (rad). Enkelt-verdier i radene leses og skrives med `get`- og `set`-funksjoner hvor første argument (`row`) er raden du skal lese fra, og andre argument (`column`) er indeksen til elementet du skal lese i denne raden.

- Implementer konstruktøren til `DynamicMultiArray`. Denne skal opprette en instans uten rader (en tom tabell).
- Implementer medlemsfunksjonen `void add(int size)`. Nye rader skal legges til på slutten og `size` angir størrelsen på raden.
- Implementer medlemsfunksjonen `void remove(int row)`. Denne brukes til å slette rader fra tabellen hvor parameter `row` angir indeksen til raden som skal slettes. Hvis rad med denne indeksen ikke finnes skal funksjonen kaste et unntak av typen `std::invalid_argument`.
- Implementer `get`- og `set`-funksjonene i `DynamicMultiArray` for å lese og endre verdier i tabellen. Parametrene `row` og `column` er her indekser tilsvarende syntaksen `[row][column]` for todimensjonale tabeller. Hvis denne posisjonen ikke finnes skal funksjonen kaste unntak av typen `std::invalid_argument`.

- h) Implementer destruktøren i **DynamicMultiArray**.
- i) Hvilken mekanisme kan du bruke for å lage en generisk versjon av **DynamicMultiArray** (slik at du kan ha **DynamicMultiArray** variabler for å lagre andre typer enn **int**).

### Oppgave 3: Minnehåndtering (20%)

Her fortsetter vi med klassene **DynamicMultiArray** og **RowNode** fra oppgave 2.

- a) Implementer en medlemsfunksjon **void RowNode::resize(int increase)** for å øke størrelsen på en rad med **increase** antall elementer.
- b) Implementer det som er nødvendig for klassene **DynamicMultiArray** og **RowNode** for å støtte “deep copy”. Dvs. at det ved tilordning (eller kall til kopikonstruktør) for objekter av typen **DynamicMultiArray** skal lages en fullstendige kopi av tabellen. Du trenger kun implementere/visse funksjoner/operatorer som må legges til eller vesentlig endres.
- c) Lag alternative versjoner av klassene hvor du implementerer “shallow copy” og “referanse-lignende” oppførsel. Dette betyr at objekter av typen **DynamicMultiArray**, som er kopier av hverandre, deler minne/data. Ved tilordning eller kall til kopikonstruktør skal det ikke allokeres nytt minne, men kopien skal peke til samme minne/data som objektet det er kopi av. Dette kalles “referanse-lignende” oppførsel fordi objektene da vil oppføre seg som referanser/pekere. Endringer som gjøres på kopien vil også gjelde for originalen og vice versa. Dette kan implementeres med en delt medlemsvariabel som teller hvor mange objekter som deler minne. Når det opprettes en ny kopi skal variabelen inkrementeres, når en kopi slettes skal variabelen dekrementeres. Hvis variabelen blir 0 betyr det at det ikke er noen kopier igjen og minnet som er i bruk kan frigis.