NTNU
Institutt for datateknikk
og informasjonsvitenskap

LØSNINGSFORSLAG TIL

TDT4102 - Prosedyre- og objektorientert programmering

Tirsdag 25. mai 2010

Kontaktperson under eksamen: Trond Aalberg (97631088)

Eksamensoppgaven er utarbeidet av Trond Aalberg
og kvalitetssikret av Hallvard Trætteberg

Språkform: Bokmål

Tillatte hjelpmidler: Walter Savitch, Absolute C++ eller Lyle Loudon, C++ Pocket Reference

Sensurfrist: Tirsdag 15 juni.

Generell introduksjon

Les gjennom oppgavetekstene nøye og finn ut hva det spørres om. Noen av oppgavene har lengre forklarende tekst, men dette er for å gi mest mulig presis beskrivelse av hva du skal gjøre.

All kode skal være C++.

Dersom du mener at opplysninger mangler i en oppgaveformulering, gjør kort rede for de antagelser og forutsetninger som du finner det nødvendig å gjøre. Hver enkelt oppgave er ikke ment å være mer krevende enn det som er beskrevet.

Oppgavesettet er arbeidskrevende og det er ikke foreventet at alle skal klare alle oppgaver innen tidsfristen. Disponer tiden fornuftig!

Oppgavene teller med den andelen som er angitt i prosent. Den prosentvise uttellingen for hver oppgave kan likevel bli justert ved sensur. De enkelte deloppgaver kan også bli tillagt forskjellig vekt.

Oppgave 1: Grunnleggende programmering (15%)

a) Hva skrives ut av følgende kode:

```
int a = 4;
int b = 5;
cout << "1) = " << a / b << endl;

int c = 4;
int d = 5;
cout << "2) = " << c % d << endl;

bool e = true;
bool f = false;
cout << "3) = " << (!e && f) << endl;

int g = 8;
int h = g++;
cout << "4) = " << g << ", " << h << endl;

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

1) = 0      //divisjon med heltall hvor 4/5 gir 0
2) = 4      //modulus som returnerer "rest" etter heltallsdivisjonen
3) = 0 (false) //evalueres til false som skrives ut som 0
4) = 9, 8   //g er inkrementert, h er tilordnet verdien i g før inkr.
5) = 5      //swap av pekere, men samme utskrift som ved swap av verdier
```

b) Implementer en funksjon **`bool match(string word, string pattern)`** som kan brukes for å teste om et ord matcher et søkeuttrykk. Søkeuttrykket (pattern-argumentet) kan bestå av bokstavene a-å/A-Å eller tegnet '.' som representerer en vilkårlig bokstav. **`match("testing", "t...i")`** skal gi **`true`** fordi ordet har 't' som første bokstav etterfulgt av tre vilkårlige bokstaver før 'i'. Tips: husk at det er mulig å bruke indeks-operatoren på string-variabler, og du kan anta at et ord bare består av bokstaver.

```
//Her er det bruk av løkker, betingelser og en fornuftig logikkk som
//er tema. Med søk mener vi at uttrykket skal matche ord eller del av
//ord, men det er greit om du har tolket dette på annen måte og f.eks.
//starter matchingen på begynnelsen av ordet (oppgaven er litt åpen)

//Enkel løsning som tester fra start av ordet og vi lar ord som er
//kortere enn uttrykket gi true
bool match2(string word, string pattern){
     for (int j = 0; j < min(word.size(), pattern.size()); j++){
       if (word[j] != pattern[j] && pattern[j] != '.'){
         return false;
       }
     }
     return true;
}
```

```
  //alternativ for de som liker å tenke enkeltuttrykk
  bool match(string word, string pattern){
       for (int j = 0; j < min(word.size(), pattern.size()); j++){
         if (pattern[j] == '.'){ //kan også løses med en enkelt if-blokk
           continue;
         }else if (word[j] != pattern[j]){
           return false;
         }
       }
       return true;
  }

//En litt mer omstendig versjon hvor vi matcher hvor som helst i ordet
//og passer på at hele søkeuttrykket matches
//Vi starter med word[0] og sjekker om hele pattern stemmer,
//hvis ikke tester vi fra word[1] og sjekker mot hele pattern osv.
//Basislogikken er en nøstet løkke hvor vi i ytterste løkke
//finner substrenger og i innerste løkke tester mot pattern.

//Det vesentlige i denne oppgaven er et relevant forslag til løsning og
løkker og betingelser som er logisk riktige (at du tenker riktig).


bool match(char c, char p){ //hjelpefunksjon for å sammenligne enkelttegn
  return (p == '.') || (c == p);
}

bool match(string word, string pattern){
  if (pattern.size() > word.size()){
    return false; //pattern er lengre en word og vil ikke matche uansett
  }
  for (int wordp = 0; wordp <=(word.size()- pattern.size()); wordp++){
    int p = 0;
    for (int w = wordp; p < pattern.size(); p++, w++){
      if (!match(word[w], pattern[p])){
        break; //pattern stemmer ikke med denne substrengen
      }
    }
    if (p >= pattern.size()){
      return true; //tester om vi har sjekket alle tegn i pattern
    }
  }
  return false;
}
```

c) Den rekursive funksjonen under er ment å skulle skrive ut tegnene i en C-streng baklengs. Hvis du kaller funksjonen med strengen"*Dette er en test*" er det meningen at utskriften skal bli "*tset ne re etteD*". Finn feil i koden og skriv riktig kode i svaret ditt.

```
void recursive(char* t){
  if (*t != '\0'){
    recursive(t++); //Feil 1: t++ gir kall med t og inkr. etterpå
  }                 //i praksis blir det uendelig rekursjon
  cout << *t;       //Feil 2: gir utskrift av '\0'
}
```

```
void recursive(char* t){
    if (*t != '\0'){
      recursive(t + 1);
      cout << *t;
    }
}
```

Oppgave 2: Programmering med funksjoner (35%)

I denne oppgaven skal du lage et program som leser innholdet i et tekstdokument (en tekstfil) og skriver ut en rapport (til en annen tekstfil) med informasjon om antall linjer i tekstdokumentet, samt hvilke bokstaver og ord som er brukt. Du kan anta at tekstdokumentet inneholder vanlige setninger hvor det er brukt skilletegn som mellomrom, komma og punktum. Du trenger ikke ta hensyn til andre tegn.

Programmet skal fungere på følgende måte:
Ved oppstart skal programmet spørre etter navnet på tekstfilen som skal leses. Hvis det ikke finnes en fil med dette navnet skal programmet skrive ut en feilmelding og avslutte. Hvis filen finnes skal programmet spørre brukeren etter navnet på ei fil som rapporten skal skrives til.

Innholdet i rapporten skal være:
1) Antall linjer i tekstdokumentet som er lest.
2) Ei liste over bokstavene a-z og hvor mange ganger hver bokstav forekommer i fila som er undersøkt (f.eks. a:10 b:12 c:0 d:2 osv). Store og små bokstaver håndteres som samme bokstav.
3) En sortert liste over alle ordene som er brukt og antall ganger hvert ord er brukt.

VIKTIG: Dette er en åpen oppgave hvor du først og fremst skal vise bruk av funksjoner og variabler samt kjennskap til nyttige typer og funksjoner i C++ biblioteket. Du trenger IKKE lage en fullstendig implementasjon av alle egendefinerte funksjonene, men skal vise nok til at den som leser koden din kan forstå hvordan du har planlagt at programmet skal fungerer. Utover beskrivelsen over står du fritt i hvordan programmet oppfører seg og hvordan rapporten skal formatteres.

Du skal i denne oppgaven svare som en samlet implementasjon. Deloppgavene lister opp noen av de spesifikke tingene vi kommer til å se etter i implementasjonen din.

a) Hvilke C++ biblioteker du benytter (vis ved hjelp av include-statements).

b) Hvilke datatyper du bruker for å lagre informasjon internt i programmet og hvordan du benytter disse. Du bør velge typer som gjør det enklest mulig å programmere løsningen).

c) Innhenting av input fra bruker (filnavnene som bruker skal skrive inn).

d) Hvilke egendefinerte funksjoner du benytter for å gjøre programmet enkelt å jobbe med (koding), enkelt å vedlikeholde (rette feil), samt enkelt å lese/forstå for andre.

e) Lesing av fil samt skriving til fil (ved hjelp av funksjoner eller ved å overlagre operatorer).

f) Riktig bruk av løkker, forgreininger og betingelser for å kontrollere programflyten.

```
#include <iostream>
#include <fstream>
#include <map>
#include <vector>
#include <string>
#include <locale>
using namespace std;

const int SIZE = 256; //size på tabellen vi bruker for å telle bokstaver

//Funksjon for å initialisere tabellen
void initCharArray(int a[]);
//Funksjon for å lese inn filnavn fra cin
string getFileName(string leadingtext);
//De følgende funksjonene er basert på at du leser linje for linje
//Teller bokstaver i en linje
void charCount(string& line, int a[]);
//Finner ordene i en linje
vector<string> parse(string& line);
//Teller ordene i en linje
void wordCount(map<string, int>& wordmap, string& line);

//Funksjoner for å skrive ut data
void printCharCount(ofstream& out, int a[]);
void printWordCount(ofstream& out, map<string, int>& wordmap);
void printLineCount(ostream& out, int count);

int main(){
     int linecounter = 0; //for å telle linjer
     int a[256]; //for å telle bokstaver
     initCharArray(a); //setter alle til 0
     map<string, int> wordmap; //for å telle ord

     //henter filnavn fra bruker og åpner fila
     string inputname = getFileName("Inputfil: ");
     ifstream input (inputname.c_str());
     if (input.fail()){
       cout << "Problemer med å åpne fila " << inputname << endl;
       exit(0); //Avslutter ved feil
     }

     string line;
     while (getline(input, line)){ //leser linje for linje til string
       linecounter++;
       charCount(line, a);
       wordCount(wordmap, line);
     }
     input.close(); //stenger inputfil
```

```
    //leser filnavn fra bruker for outputfil
    string outputname = getFileName("Output: ");
    ofstream output(outputname.c_str());
    if (output.fail()){
      cout << "Problemer med å åpne fila " << outputname << endl;
      exit(0);
    }

    //skriver output
    printLineCount(output, linecounter);
    printCharCount(output, a);
    printWordCount(output, wordmap);

    //stenger outputfil
    output.close();
}


//Det viktigste i denne oppgaven er at du har dekomponert løsningen din
//i et fornuftig sett av funksjoner og bruker relevante datatyper for å
//telle forekomster. Funksjonsdeklarasjoner er viktigere enn detaljene.
//Viktig er å vise hvordan du leser fra fil og hvordan du prosesserer
//input. Her har vi en løkke som leser linje for linje og behandler hver
//linje inne i løkka. Vi har brukt en int for å telle linjer, en int[256]
//for å telle tegn og bruker en map<string, int> for å telle ord
//Vi teller alle tegn, men skriver bare ut a-z i utskriftsfunksjonen.
//Koden over viser en grei løsning (og er nok til en bra karakter),
//men det er en fordel om du viser deler av implementasjonen av funksjoner
//også. Vanskeligste funksjon er antagelig parse-funksjonen.
//Å lese fila flere ganger i forskjellige funksjoner er IKKE noen god
//løsning, å lagre hele fila internt er heller ikke bra.

//Her er forslag til implementasjoner:

void initCharArray(int a[]){
  for (int i = 0; i < SIZE; i ++){
    a[i] = 0;
  }
}

string getFileName(string leadingtext){
  string temp;
  cout << leadingtext;
  cin >> temp;
  return temp;
}

void charCount(string& line, int a[]){
  for (unsigned int i = 0; i < line.size(); i ++){
    a[tolower(line[i])]++;
  }
}
```

```
vector<string> parse(string& line){
  vector<string> wordlist;
  string word = ""; //starter med tom streng til enkeltord
  for (int i = 0; i < line.size(); i++){
    if (isalpha(line[i])){ //hvis tegnet er en bokstav
      word += tolower(line[i]); //legger til enkeltbokstaver
    }else if (word != ""){ //hvis det ikke er bokstav og word ikke er tom
      wordlist.push_back(word); //lagrer ordet i vectoren
      word = ""; //setter word-variabelen til tom streng
    }
  }
  if (word != ""){ //spesialhåndtering for siste ord i linja
    wordlist.push_back(word); //storing last word
  }
  return wordlist;
}

void wordCount(map<string, int>& wordmap, string& line){
  vector<string> words = parse(line);
    for (int i = 0; i < words.size(); i ++){
      wordmap[words[i]]++; //oppdaterer map med ord-forekomstene
    }
}

//print funksjoner (kanskje de minst interessante)

void printCharCount(ofstream& out, int a[]){
  out << endl << "Bokstavene: " << endl;
  for (unsigned int i = 'a'; i <= 'z'; i ++){
    out << (char) i << ": " << a[i] << endl;
  }
}

void printWordCount(ofstream& out, map<string, int>& wordmap){
  out << endl << "Ordene: " << endl;
  map<string, int>::iterator it;
  for (it = wordmap.begin(); it != wordmap.end(); it++){
    out << it->first << ": " << it->second << endl;
  }
}

void printLineCount(ostream& out, int count){
  out << "Antallet linjer: " << count << endl;
}
```

Oppgave 3: Klasser (25%)

I denne oppgaven skal du lage en klasse kalt **`Timer`**. Denne klassen skal kunne brukes for å måle tidsbruk omtrent på samme måte som du vil bruke en Stoppeklokke. Du bør lese igjennom alle deloppgavene for å få en fullstendig oversikt over hvordan klassen skal virke.

Timer-klassen skal benytte funksjonen **`int clock()`** fra **`ctime`**-biblioteket for å beregne tidsbruk[1]. Denne funksjonen returnerer antallet taktslag som har gått siden et program startet (hvor mange ganger prosessoren har "tikket" siden oppstart av programmet). Selve tidtakingen styres med eksplisitte kall til Timer-klassens medlemsfunksjoner **`void start()`** og **`void stop()`**. Disse skal implementeres slik at man kan starte og stoppe tidtakingen flere ganger etter hverandre for å akkumulere tidsbruk. Klassen skal også ha medlemsfunksjoner som returnerer tidsbruken som henholdsvis *minutter*, *sekunder* eller *millisekunder* (tusendels sekunder). Vi kan bruke konstanten **`CLOCKS_PER_SEC`** fra **`ctime`**-bibliotek for å regne om fra taktslag til sekunder.

Klassen skal implementeres som beskrevet i deloppgavene, men du velger selv om du vil presentere kode under hver deloppgave eller som en helhetlig og samlet implementasjon. Velger du en samlet implementasjon bør du kommentere koden slik at det går tydelig fram hvilke deler du har svart på.

```
class Timer{

private:
     int ticks;            //Det er ikke behov
     int starttick;        //for å lagre sekunder, minutter etc.
     bool running;

public:
     Timer(): ticks(0), starttick(0), running(false) {};
     int minutes(){ return (ticks/CLOCKS_PER_SEC)/60; };
     int seconds(){return ticks/CLOCKS_PER_SEC;};
     int ms(){return (ticks/ (CLOCKS_PER_SEC/1000) );};
     void start();
     void stop();
     bool isRunning();
     bool operator ==(Timer& t){return ticks == t.ticks;};
};
```

a) Hvilke medlemsvariabler (en eller flere) må klassen ha for å kunne ta vare på tidsbruk og annen informasjon som er nødvendig.

Må ha en variabel for å lagre antall taktslag som er gått mellom start og stop, og en variabel for å lagre start-tidspunktet slik at vi kan regne ut antall taktslag når stop kallles. I tillegg har vi i LF tatt med en variabel for tilstanden (om tidtakingen er i gang eller ikke).

1. De som har vært borte i denne funksjonen tidligere vil kanskje legge merke til at vi har forenklet litt.

b) Hva er innkapsling (generelt) og hvordan har du brukt innkapsling i Timer-klassen?

Innkapsling er å skjule medlemsvariabler (sette disse som private) og kontrollere lesing/skriving ved hjelp av funksjoner. Her brukes private medlemsvariabler som kun endres internt i klassens egne funksjoner. Vi leser kun minutter, sekunder eller millisekunder.

c) Implementer en av følgende medlemsfunksjoner: En funksjon som returnerer tidsbruken som hele minutter, en funksjon som returnerer tidsbruken som hele sekunder eller en funksjon som returnerer tidsbruken som millisekunder (tusendels sekunder) .

Se inline-implementasjonene i klassedeklarasjonen.

d) Hva er formålet med en klasses konstruktør(er)? Implementer en egnet konstruktør for Timer-klassen.

Formålet er å sørge for at objekter blir riktig initialisert ved instansiering. Her setter vi alle ticks og stattick til 0, og running til false. Hvis vi ikke gjør dette vil de få "tilfeldig" verdi ved oppstart og klassen vil ikke fungere riktig.

e) Implementer medlemsfunksjonene **`start`** og **`stop`**. Disse skal implementeres slik at man kan starte og stoppe tidtakingen flere ganger for å akkumulere tidsbruk. Det skal derfor være mulig å gjøre funksjonskall-sekvensen **`t.start(); t.stop(); t.start(); t.stop();`**
Dette skal gjøre at **`t`** lagrer tiden mellom første **`start`** og **`stop`** og etterpå legger til tiden mellom andre **`start`** og **`stop`**.

```
//Her løser vi både e) og f)
//For e) er løsningen å huske starttidspunktet i start(),
//regne ut tidsbruk og legge til tiden i stop()
void Timer::start(){
  if(!running){
    starttick = clock();
    running = true;
    cout << starttick << endl;
  }
}

void Timer::stop(){
  if(running){
    cout << clock() << endl;
    ticks += clock() - starttick;
    running = false;
    starttick = 0;
  }
}
```

f) I praksis er kall til **`stop`** kun meningsfullt hvis tidtakingen allerede er startet (det har vært utført et kall til start). Kall til **`start`** er kun meningsfullt hvis tidtakingen ikke allerede er startet. Dette kan gi potensielt gi problem f.eks. hvis man gjør kallsekvensen **`t.start(); t.start(); t.stop(); t.stop();`**. Forklar hvordan du velger å håndtere dette (dette er en åpen oppgave og vi legger primært vekt på gode argumenter for ditt valg).

Her er det flere mulige løsninger og det er opp til deg å argumentere for ditt valg.
a) Du kan velge å ikke gjøre noe som helst og overlate til den som bruker klassen å bruke den riktig
b) Du kan ha en funksjon som bruker kan kalle å sjekke om tidtagingen er i gang eller ikke
c) Du kan implementere en spesifikk håndtering av dette. Ved gjentatte start kan du enten velge å bruke første (og ignorere repetive kall) eller du kan velge å bruke siste. For stop er det i tilfellet enklest å bare gjøre noe ved første kall til stop. Dette er logikken som er implementert i e)
d) Kaste et unntak, men dette er litt på kanten av hva man vanligvis bruker unntak til.

g) Overlagre en (fritt valgt) sammenligningsoperator som medlem av klassen. Operatoren skal sammenligne tidsbruken som 2 Timer-objektene representerer. Du kan anta at operatoren bare benyttes når tidtakingen i begge objekter er "stoppet" opp. Objektene skal sammenlignes mht. antall taktslag.

Se inline implementasjon i deklarasjonen av klassen.
Her er det viktig å lage en enkel og grei implementasjon og at du vet hvordan du implementerer som medlem (kun ett parameter). Et viktig poeng er at selv om ticks er private så gjelder dette på "klassenivå". Du kan derfor lese t.ticks. Det var en gjennomgående feil blant svarene å bruke to parameter, men her spør vi spesifikt etter "medlemsoperator".

Oppgave 4: En dynamisk todimensjonal tabell (25%)

I denne oppgaven skal du lage en objektorientert datastruktur som ligner litt på en todimensjonal array, men hvor du dynamisk kan legge til nye rader av forskjellig lengde (mens programmet kjører). Datastrukturen skal være basert på en lenket liste hvor radene er noder med hver sin peker til et dynamisk allokerte array. Det er med andre ord ikke lov å bruke vector eller andre klasser fra Standard Template Library. Se vedlegget for en illustrasjon.

Deklarasjonen av klassene **`DynamicMultiArray`** og **`RowNode`** er vist under (men du må selv bestemme om det er behov for andre variabler eller funksjoner). Funksjonen **`addRow`** brukes for å legge til en og en rad. Medlemsvariablene **`first`** og **`last`** peker hhv. til første og siste element i den lenkede listen av rader. Enkelt-verdier kan leses med funksjonen **`at`** hvor første argument (**`row`**) er raden du skal lese fra (nummerert fra 0 og oppover), og andre argument (**`column`**) er indeksen til elementet du skal lese i denne raden.

```
class DynamicMultiArray{

private:
   RowNode *first;
   RowNode *last;
public:
   DynamicMultiArray();
   ~DynamicMultiArray();
   void addRow(int row[], int size);
   int at(int row, int column);
};
```

```
class RowNode{

private:
   int *row;
   RowNode* next;
   int rowsize;
public:
   RowNode(int row[], int size);
   ~RowNode();
   friend class DynamicMultiArray;
};
```

a) Implementer konstruktøren til **`RowNode`**. Konstruktøren skal initialisere objektene riktig og kopiere innholdet i row-argumentet over i en dynamisk allokert array som medlemsvariabelen row peker til.

```
//Initialiser next til NULL
//alloker en ny tabell og kopier

RowNode::RowNode(int row[], int size){
   rowsize = size;
   next = NULL;
   this->row = new int[size];
   for (int i = 0; i < size; i++){
      this->row[i] = row[i];
   }
}
```

b) Implementer konstruktøren til **`DynamicMultiArray`**.

```
//Sett first og last til NULL
DynamicMultiArray::DynamicMultiArray(){
   first = NULL;
   last = NULL;
}
```

c) Implementer medlemsfunksjonen **addRow**. Nye rader skal legges til på slutten av lista.

```
//Instansier nytt radobjekt
//Trenger å spesialhåndtere tom liste, eller er det bare
//å bruke last og legge til ny rad

void DynamicMultiArray::addRow(int arr[], int size){
  RowNode *temp = new RowNode(arr, size);
  if (first == NULL && last == NULL){
    first = temp;
  }else{
    last->next = temp;
  }
  last = temp;
}
```

d) Implementer medlemsfunksjonen **at**. Parametrene **row** og **column** er her indekser tilsvarende syntaksen [row][column] for todimensjonale tabeller. Funksjonen skal returnere verdien som finnes på denne posisjonen. Hvis denne posisjonen ikke finnes skal funksjonen kaste et unntak av typen **InvalidIndex** (du må også deklarere denne typen).

```
//Viser både d) og e) her.
//Svaret på d) blir en enklere utgave uten testingen og throw
//For e) har vi gjort et lite triks som ikke kommer fram i koden over.
//Vi har brukt en medlemsvariabel kalt size i DynamicMultiArray
//som inkrementeres hver gang vi legger til en rad.
//Da slipper vi å gå igjennom alle radene og telle før vi finnner ut at
//raden ikke finnes.

int& DynamicMultiArray::at(int row, int column){
  if (row < size){
    RowNode *current = first;
    for (int i = 0; i < row; i++){
      current = current->next;
    }
    if (column < current->rowsize){
      return current->row[column];
    }else{
      InvalidIndex i;
      throw i;
    }
  }else{
    InvalidIndex i;
    throw i;
  }
}
//Viktig i svaret er at du tester for feil og bruker throw.
//Løser du uten en size-variabel i DMA må du iterere deg framover med en
//løkke og telle noder. Hvis du kommer til siste node uten at antallet
//noder > row-parameteren skal du kaste et unntak.
```

e) Hvordan må funksjonen **at** deklareres hvis du skal kunne bruke denne til å endre på enkelt-elementer i tabellen slik som vist i denne setningen **d.at(3,3) = 5;**
(hvor **d** er en variabel av typen **DynamicMultiArray**)

```
//Se oppgaven over. Her er løsningen at returverdien må våre en
//referanse og det eneste du trenger å gjøre er å deklarere
//returtypen som int& at(int row, int column);
//hvis du returnerer en peker er det mye mer som må endres
```

f) Hva betyr følgende linje i RowNode: **friend class DynamicMultiArray**?

Det betyr i praksis at DynamicMultiArray-funksjonene har tilgang til det som er privat i RowNode (gjør at vi kan programmere direkte mot RowNode sine medlemsvariabler). Merk at det er DMA som har tilgang til RowNode og ikke omvendt.

g) Hvordan kan du deklarere at **addRow** ikke endrer på verdiene til argumentet **int row[]**?

Ved å deklarere denne parameteren som const.

```
void addRow(const int row[], int size);
```

Ekstrapoeng hvis du også ser at du dermed må endre konstruktørparameteren til const

h) Implementer destruktoren i begge klasser slik at alt minne som er allokert blir frigjort når et **DynamicMultiArray**-objekt slettes (enten automatiske variabel som går ut av scope eller dynamisk allokerte variabel som eksplisitt slettes med delete).

```
//Viktig å gå igjennom alle noder og slette disse
//Husk å ikke slette for du har flyttet peker til neste, gjøres ved å
//bruke en temp-peker til objektet som skal slettes
DynamicMultiArray::~DynamicMultiArray(){
  RowNode* current = first;
  RowNode* temp;
  while (current != NULL){
    temp = current;
    current = current->next;
    delete temp;
  }
}

//Her må du eksplisitt slette
//den allokerte tabellen
RowNode::~RowNode(){
  delete [] row;
}
```

```
//Oppgaven kan også løses med bruk av rekursjon.
DynamicMultiArray::~DynamicMultiArray(){
  delete first;
}

RowNode::~RowNode(){
  delete [] row;
  delete next;
}

//Rekursjonen svsluttes når next == NULL siden delete av
//en NULL-peker er lov.
```

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).

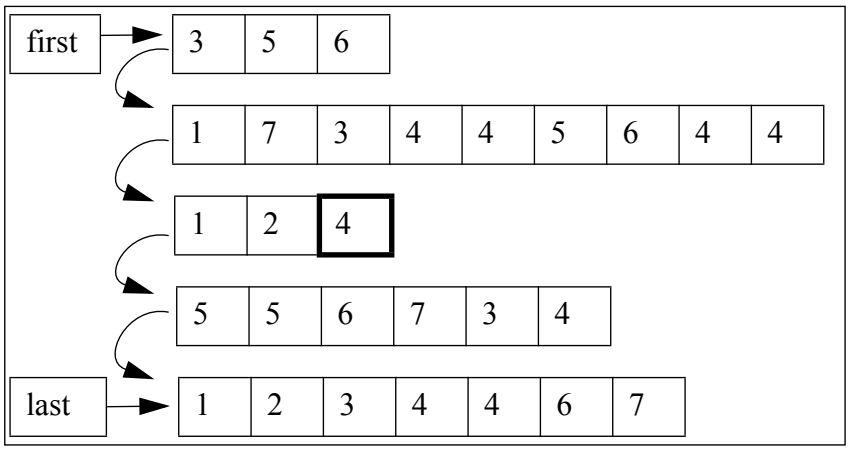
```
Vi kan lage template-klasser av DynamicMultiArray og RowNode hvor vi pa-
rameteriserer int-datatypen som brukes: T row[] og T row*
```

VEDLEGG

Eksempel som illustrerer bruken av DynamicMultiArray.

```
DynamicMultiArray *d = new DynamicMultiArray();

int a[3] = {3, 5, 6};
d ->addRow(a, 3);

int b[9] = {1, 7, 3, 4, 4, 5, 6, 4, 4};
d->addRow(b, 9);

int c[3] = {1, 2, 4};
d ->addRow(c, 3);

int d[6] = {5, 5, 6, 7, 3, 4};
d->addRow(d, 6);

int e[7] = {1, 2, 3, 4, 4, 6, 7};
d ->addRow(e, 7);
```

//datastrukturen etter at radene er lagt til med addRow:




```
cout << d->at(2,2) << endl;
//skriver ut tallet i ruten med fet linjetype (oppgave 4d)

d ->at(2,2) = 10;
//endrer innholdet i elementet som er merket med fet linjetype (oppgave 4e)

delete d;
//sletter d-variabelen og alt minne som brukes
```