NTNU
Institutt for datateknikk
og informasjonsvitenskap

Kontinuasjonseksamen i

TDT4102 - Prosedyre- og objektorientert programmering

Fredag 19. august 2011, 09:00

Kontaktperson under eksamen: Hallvard Trætteberg (73593443)

Eksamensoppgaven er utarbeidet av Trond Aalberg

Språkform: Bokmål

Tillatte hjelpmidler: Spesifiserte trykte og håndskrevne hjelpemidler tillatt. Bestemt, enkel kalkulator tillatt (hjelpemiddelkode C)**.**

Tilatte trykte hjelpemidler: Walter Savitch, Absolute C++ eller Lyle Loudon, C++ Pocket Reference

Sensurfrist: Fredag 9 september.

Generell introduksjon

Les gjennom oppgavetekstene 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. Fokuser på det som er det sentrale spørsmålet i hver deloppgave.

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.

Noen av oppgavene er "oppskriftsbasert" og vi spør etter forskjellige deler av et litt større helhetlig 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.

Hele oppgavesettet er arbeidskrevende og det er ikke foreventet at alle skal klare alle oppgaver innen tidsfristen. Tenk strategisk i forhold til ditt nivå og dine ambisjoner!

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: Kodeforståelse og funksjoner (30%)

a) Hva skrives ut av følgende kode?

```
int a = 0, b = 0, c = 0;

a += b + 1;
cout << "1: a = " << a << endl;

b = 5;
a = --b;
cout << "2: a = " << a << endl;

a = 6;
b = 5;
c = a % b;
cout << "3: c = " << c << endl;

a = 5;
b = 6;
c = a / b;
cout << "4: c = " << c << endl;
//1, 4, 1, 0
//(med linjeskift selvsagt)
```

b) Hva skrives ut av følgende kode?

```
int *x = new int(1);
int *y = new int(2);

int z = *x;
cout << "1: z = " << z << endl;

(*y)++;
x = y;
cout << "2: x = " << *x << endl;


//skriver ut 1, 3

//siste del av oppgaven
//*x = (*y)++; var
//kode som resulterte
//i udefinert oppførsel
//en liten tabbe rett og slett....
```

c) Lag en funksjon **`int findFirst(int tab[], int val, int size)`** som returnerer første posisjon (indeksen) i tabellen **`tab`** hvor verdien **`val`** forekommer. Hvilken verdi kan det være hensiktsmessig at funksjonen returnerer hvis verdien ikke finnes i tabellen?

```
int findFirst(int tab[], int val, int size){
  for (int i = 0; i < size; ++i){
    if (tab[i] == val){
      return i;
    }
  }
  //En grei løsning å returnere et negativt tall (-1)
  //hvis val ikke finnes i tabellen
  return -1;
}
```

d) Lag en tilsvarende funksjon **`findFirst(...)`** for tekststrenger (c-strenger) som kan brukes for å finne indeksen til første forekomst av et tegn (char) i en teksstreng.

```
//Her må vi teste mot termineringstegnet og kan bruke en while-løkke
//Også mulig å bruke strlen (men ikke baser deg på en size-variabel!).
int findFirst(char *s, char c){
  int pos = 0;
  while (*s != '\0') {
    if (*s == c){
      return pos;
    }
    s++;
    pos++;
  }
  return -1;
}
```

e) Implementer en funksjon **`bool isAnagram(char* a, char* b)`** som sjekker om to strenger er anagrammer av hverandre. Anagram (av gresk: ana og graphein = omskrive) er et ord, navn eller et fast uttrykk som er blitt satt sammen ved å stokke rundt på bokstavene i et annet ord eller uttrykk. Funksjonen skal ikke ta hensyn til skilletegn (mellomrom, komma, punktum etc.) og skal håndtere store og små bokstaver som like tegn.

Eksempler:
"I am Lord Voldemort" er et anagram av "Tom Marvolo Riddle"
"Balkongdjevelen Kim" er et anagram av "Kjell Magne Bondevik"

```
//Eksempel på løsning finnes i LF for konten 2009
//ja dette var klipp og lim (sjeldent brukt i dette emnet ;-)
//Her legger vi vekt på koding og ryddig logikk.
//Viktig at funksjonen ikke endrer på argumentene (vanligvis ville disse
//vært deklarert med const, men dette var en test på om du skjønte at du
//ikke bør endre disse).
//En strategi er å lage kopier av strengene basert på bare bostavene
//i lowercase, sortere og sammenligne tegn for tegn
//En annen strategi som flere har brukt er å rett og slett telle
//forekomsten av hvert enkelt tegn for deretter å sammenligne tallene
```

f) I medlemsfunksjonen **`dummyfunction`** i klassen **`Foo`** under er nøkkelordet **`const`** brukt på tre plasser. Forklar hva effekten er for hver enkelt av disse.

```
class Foo{
  private:
    set<string> dummyvar;
  public:
    const string& dummyfunction (const string& p) const;
};
//Første const: referansen som returnere er constant og da kan du ikke
//endre på variebeln det refereres til direkte (f.eks. ved å ha funksjonen
//på venstre side i en tilordning
//Andre const: argumentet const og kan ikke endres i funkjsonen
//Tredje const: funksjonen endrer ikke noen medlemmer i objektet
```

Oppgave 2: Funksjoner, unntakstyper, minnehåndtering m.m. (35%)

I denne oppgaven skal du lage et program som slår sammen flere tekstfiler og lagrer disse i ei enkelt fil. Programmet skal først be brukeren skrive inn en serie filnavn det skal leses fra og deretter be brukeren skrive inn et filnavn det skal skrive til.

I main() skal det være en variabel **`vector<string> infilenames`** for å lagre et dynamisk antall filer som programmet skal lese og en variabel **`string outfilename`** for filnavnet det skal skrives til.

Programmet ditt skal bestå av diverse funksjoner som beskrevet i deloppgavene og eller oppføre seg som spesifisert i deloppgavene. Der parameterlista eller returtypen ikke er spesifisert er det opp til deg å bestemme funksjonsheadere.

a) Lag en funksjon **`void getInfileNames(...)`** som brukes for å be brukeren skrive filnavnene som det skal leses fra og lagre disse i infilenames. Deretter skal funksjonen be om navnet til fila det skal lagres i. Du kan anta at filnavnene kun inneholder bokstaver og punktumtegnet (f.eks. "testfil.txt"), og eksempelvis lese fra cin til en string og avslutte ved at brukeren skriver inn bokstaven 'x'. Her er vi primært ute etter bruk av **`cout`**, **`cin`**, kontrollstrukturen du bruker, hvordan du får lagret i main-variabelen **`infilenames`** og **`outfile`**.

```
void getFileNames(vector<string> &infiles, string &outfile){
  string temp = "";
  while (true) {
    cout << "Skriv inn filnavn det skal leses fra: ";
    cin >> temp;
    if (temp == "x"){
      break;
    }
    infiles.push_back(temp);
  }
  cout << "Skriv inn filnavn det skal skrives til: ";
  cin >> outfile;
  //Vi spør ikke etter mer enn dette i oppgaven
  //Husk call by reference, og bruk av løkke for å lese inn
  //flere filnavn
}
```

b) Lag en funksjon **`void copy(....)`** som leser alle filene i **`infilenames`** og lagrer filinnholdet fra alle disse filene i **`outfilename`**.

```
void copy(vector<string> &infilenames, string &outfilename){
  ofstream outfile(outfilename.c_str());
  //Åpne og sjekk om fila kan skrives til
  if (outfile.fail()){
    string msg = "Feil ved åpning av output fila: ";
    msg += outfilename;
    throw outputFileException(msg);
  }

  //Leser en og en fil og skriver til outfile
  for (int i = 0; i < infilenames.size(); ++i){
    ifstream infile(infilenames.at(i).c_str());
    //Åpne og sjekk om fila kan skrives til
    if (infile.fail()){
      string msg = "Feil ved åpning av input fila: ";
      msg += infilenames.at(i);
      throw inputFileException(msg);
    }
    //Kopiering kan gjøres på mange måter,
    //her leser vi tegn for tegn.....
    char c;
    while (infile.get(c)){
      outfile.put(c);
    }
    infile.close();

  }
  outfile.close();
}

//Husk å åpne og stenge filer, samt teste for feil
//Unntakskastingen her er for oppgave c)
//Husk løkke så du leser alle filene og skriver til en utfil
```

c) Utvid programmet med unntakstyper og kasting av unntak. Funksjonen copy skal kaste en type unntak hvis en av filene i infilenames ikke kan åpnes, og en annen type unntak hvis fila outfilename ikke kan åpnes. Begge unntakstypene skal arve fra samme unntakstype og kunne inneholde navnet til fila som forårsaket feilen. Vis hvordan du kan fange opp disse unntakene i main og skrive ut forskjellige informasjon avhengig av hvilken type unntak som oppstod.

```
class superexc{
private:
   int i;
   string message;
public:
   superexc(const string &s): message(s){}
   string getMsg(){return message;}
   int getI() {return i;};
};

//To ting av interesse:
//En supertype vi kan arve fra som har en string variabel
//for å lagre melding
//Siden message er privat i supertypen bruker vi
//initialiseringsliste og kaller
//supertypens konstruktør for å sette denne via subtypene
//Noe mer enn dette trengs ikke.....
//Navngivingen her er likegyldig

class inputFileException : public superexc{
public:
   inputFileException(const string &s): superexc(s){}
};

class outputFileException : public superexc{
public:
   outputFileException(const string &s): superexc(s){}
};

//Eksempel på å fange unntak i main:
int main(){
    try{
       //kode
       //som
       //skal
       //kjøres
    }catch(inputFileException &exc){
        cout << "oops - inputFileError" << endl;
        cout << exc.getMsg();
    }catch(outputFileException &exc){
        cout << "oops - outputFileError" << endl;
        cout << exc.getMsg();
    }
}
```

d) Lag en ny versjon av funksjonen copy hvor du lagrer innholdet i filene i en (eller flere) variabler av dynamisk størrelse (du kan bruke typen char slike variabler). Når alle filer er lest (og lagret i minne) skal du lagre alt til fila outfilename og frigjøre minne som er brukt før programmet avslutter. Siden filene kan være store, er du i fare for å bruke opp alt tilgengelig minne. Hvis det ikke er mulig å allokere nok minne vil operatoren **new** kaste et unntak av typen bad_alloc. Implementer unntakshåndtering for dette i main, men ta også hensyn til at programmet også ved slike unntak skal frigjøre alt allokert minne før det avslutter.

Oppgaven sier ikke noe om hvilken type variabel du skal lagre i. Du kan bruke en char[] som du øker størrelsen på selv ved behov (gir mye kopiering) eller du can bruke en vector<char>.

Det viktigste er at du fanger opp bad_alloc i main. Hvis du bruker en lokal (automatisk/vanlig) vector-variabel i copy-funksjonen til å mellomlagre, vil funksjonen selv rydde opp. Hvis du allokerer minne selv er det du som må rydde opp. Da må du fange opp bad_alloc i copy, frigjøre allokert minne og videresende bad_alloc ved hjelp av throw inne i catch-delen.

```
void copy(vector<string> &infiles, vector<char> &temp, string &outfile);

int main(){
  try{
    vector<string> infilenames;
    string outfilename;
    getFileNames(infilenames, outfilename);
    vector<char> temp;
    copy(infilenames, temp, outfilename);
    ofstream output(outfilename.c_str());
    if(output.fail()){
      string msg = "Feil ved åpning av output fila: ";
      msg += outfilename;
      throw outputFileException(msg);
    }
    vector<char>::iterator it;
    for(it = temp.begin(); it != temp.end(); ++it){
      output << *it;
    }
    output.close();
  }catch(bad_alloc){
    cout << "Tomt for minne!";
  }catch(superexc &e){
    cout << e.getMsg();
  }
}

void copy(vector<string> &infiles, vector<char> &temp, string &outfile){
  vector<string>::iterator it;
  for(it = infiles.begin(); it != infiles.end(); ++it){
    ifstream input;
    input.open(it->c_str());
    if(input.fail()){
      string msg = "Feil ved åpning av input fila: ";
```

```
        msg += *it;
        throw inputFileException(msg);
      }
      char next;
      input.get(next);
      while(!input.eof()){
        temp.push_back(next);
        input.get(next);
      }
      input.close();
    }
}
```

Oppgave 3: Klasser og bruk av STL collection typer (35%)

I appendiks 2 finner du eksempler på bussruter i Trondheim. I denne oppgaven skal du implementere en klasse **`BusRoute`** som kan brukes for å opprette og manipulere (endre, legge til, slette) slike bussruter.

Viktige elementer i bussruten er:

- Nummeret til bussruten
- Navnet til bussruten
- Tidstabellen over bussavganger. Alle bussruter har en eller flere avganger hver time. I bussruteeksempleen er det bare avganger for timene 06-23, men for enkelthets skyld kan du lage en tidstabell som støtter alle døgnets timer.
- Til enkeltavganger kan det være knyttet noter for mer informasjon om denne avgangen.
- Liste over busstopper med tidligste ankomst (antall minutter etter avgang). Merk at rekkefølgen av disse er viktig.

- Du trenger ikke ta hensyn til annen informasjon en det som er spesifisert over.

Merk at det kan være nødvendig å lage seg flere klassetyper i tillegg for de enkelte elementene i bussruta og at det kan være hensiktsmessing å lage seg hjelpefunksjoner. I denne oppgaven gjelder det å velge seg hensiktsmessige datatyper for eksempel fra STL for å unngå unødvendig tungvinte løsninger. Les alle deloppgaver før du bestemmer typer.

Lag en klasse som tilfredstiller kravene i deloppgavene under. Det trenger ikke implementere mer enn det som spesifikt er spurt etter i deloppgavene. Der det ikke er spesifisert navn, returtype og parameterliste er det opp til deg å definere funksjonsheaderen.

```
//NB Dette er bare et tentativt løsningsforslag som ikke er utførlig testet
//Klipp og lim av koden gjøres på eget ansvar ;-)
//Det vi er ute etter i denne oppgaven er at du skal velge enkle og
//greie løsninger, samt vise
//din kunnskap om bruk av STL collection typene
```

a) Medlemsvariabler for nummeret til bussruten og navnet til bussruten og en konstruktør som tar inn navn og nummer og setter disse i en initialiseringsliste.

```
class BusRoute{
private:
   string name;
   int number;

public:
   BusRoute(const string &name, int number):name(name), number(number){}
};
```

b) Get og set-funksjoner for nummeret til bussruten og navnet til bussruten. Det skal kun være mulig å endre nummer og navn ved hjelp av set-funksjonene.

```
//Her skal selvsagt medlemsvariablene være private
//Legger til følgende funksjoner under klassens public del:
void setName(const string &name){this->name = name;}
void setNumber(int number){this->number = number;}
```

c) Medlemsvariabel (eller medlemsvariabler) for tidstabellen. Husk at antallet avganger per time kan variere, samt at det kan være en eller flere noter knyttet til hver avgang.

```
//Det enkleste er ofte det beste.
//Lag deg f.eks en egen klasse for tidspunktene

//Det er tidspunktene og notene vi må lagre, den tabulariske
//presentasjonen kan vi generere automatisk.
//Her lager vi en løsning som gjør det enkelt å programmere
//innsetting og sletting etc, men utskriftsfunksjonen kan det hende blir
//ganske komplisert. Siden du ikke skal lage en utskriftsfunksjon,
//trenger du selvsagt ikke designe koden din mht. til denne ;-)

class DepartureTime{
private:
  int day; //f.eks. 1 = ukedag, 6 er lørdag og 7 er søndag,
           //kan også bruke enum til å definere lovlige typedager
  int hour;
  int minute;
  set<string> notes;
public:
  DepartureTime(int day, int hour, int minute):
  day(day), hour(hour), minute(minute){}
  friend class BusRoute;
  bool operator <(const DepartureTime &d) const{
    //her laget med if-else, kan skrives på flere måter
    if (day < d.day){
      return true;
    }
    if (day == d.day){
      if (hour < d.hour){
        return true;
      }
      if (hour == d.hour){
        return minute < d.minute;
      }
    }
    return false;
  }
};

//I Busroute kan vi legge til følgende private medlemsvariabel:
  set<DepartureTime> departures;
```

d) En medlemsfunksjon for å legge til nye avganger **`addDeparture(...)`**.

```
//Bruken av set gir oss det vi trenger
void BusRoute::addDeparture(int day, int hour, int minute){
  departures.insert(DepartureTime(day, hour, minute));
};
```

e) En medlemsfunksjon for å legge til en note til en avgang **`addDepartureNote(...)`**.

```
void BusRoute::addDepartureNote(int day, int hour, int minute,
                                const string &note){
  set<DepartureTime>::iterator it =
           departures.find(DepartureTime(day, hour, minute));
  DepartureTime temp = *it; //Lager en kopi (inkl. tidligere noter)
  departures.erase(it);
  temp.notes.insert(note);
  departures.insert(temp);
};

//Må lete opp eksisterende avgangstid som vi skal legge til note for
//Her håndterer vi ikke situasjoner som kan oppstå hvis avgangstiden
//ikke finnes fra før (var ikke noe krav i oppgaven)
//Merk at vi må lage en kopi av eksisterende objekt, fjerne denne og
//deretter lagre på nytt siden objekter som er lagret i et set ikke kan
//endres.
```

f) En medlemsfunksjon for å slette en avgang **`removeDeparture(...)`**.

```
void BusRoute::removeDeparture(int day, int hour, int minute){
  departures.erase(departures.find(DepartureTime(day, hour, minute)));
};
//kompakt, men gjør det vi trenger
```

g) Medlemsvariabel (eller medlemsvariabler) for stoppesteder (husk at dette skal inkludere navnet på stedet og tidspunktet bussen passerer) slik at disse lagres i riktig rekkefølge.

```
class Stop{
private:
  int time;
  string name;
public:
  Stop(int time, const string &name): time(time), name(name){};
  friend class BusRoute;
};

//I BusRoute som privat medlemsvariabel bruker vi list siden vi trenger
//en organsiert rekkefølge og list gir muligheten for å sette inn
//midt i på en effektiv måte
  list<Stop> stops;
```

h) En medlemsfunksjon for å legge til nye busstopper **`addStop(...)`**.

```
//Egentlig trenger vi en funksjon for å sette inn på slutten
//og en funksjon for å sette inn på en spesifikk plass
void BusRoute::addStop(int time, const string &place){
  stops.push_back(Stop(time, place));
}

//Oppgaven sa lite som indikerte at dette var funksjonen vi egentlig
//tenkte på, men det de mer drevne skjønner fort at det
//er nyttig å kunne legge til nye stop hvor som helst i listen
void BusRoute::addStopBefore(int atime, const string &aplace,
                             int ntime, const string &nplace){
  list<Stop>::iterator it;
  //Finner stoppestedet vi skal sette inn før
  for (it = stops.begin(); it != stops.end(); ++it){
    if ((it->time == atime) && (it->name == aplace)){
      break; //avslutter løkka hvis vi har funnet atime/aplace
    }
  }
  if (it != stops.end()){
    stops.insert(it, Stop(ntime, nplace));
  }else{
    stops.push_back(Stop(ntime, nplace));
  }
}
```

i) Gitt at du skal lagre en eller flere bussruter i en variabel av typen **`set<BusRoute>`**. Hvilke egenskaper har et STL set? Hva må du implementere for klassen for at dette skal kompilere (og fungere).

Et set sikrer at alle objekter er unike samt at de kan hentes ut i sortert rekkefølge (med sammenligning basert på operatoren <). Typen du skal lagre i et set må implementere operatoren < (less than).

Appendix 1: Classes and functions that may be of interest

Functions common for all container classes	
begin	Return iterator to beginning (public member type)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
size	Return size (public member function)
empty	Test whether e.g. vector is empty (public member function)
Iterator find (Iterator first, Iterator last, const T& value); Returns an iterator to the first element in the range [first,last) that compares equal to value, or last if not found.	
void sort (Iterator first, Iterator last); Sort elements in range (can be used to sort arrays as well using pointers instead of iterators e.g. sort(tab, tab+10)	
vector	
operator[]	Access element (public member function)
at	Access element (public member function)
front	Access first element (public member function)
back	Access last element (public member function)
push_back	Add element at the end (public member function)
set	
insert	Insert element (public member function)
find	Get iterator to element (public member function)
list	
front	Access first element (public member function)
back	Access last element (public member function)
push_front	Insert element at beginning (public member function)
pop_front	Delete first element (public member function)
push_back	Add element at the end (public member function)
insert	iterator insert (iterator position, const T& x); The list container is extended by inserting new elements before the element at position.
map	

Appendix 1: Classes and functions that may be of interest

operator[]	Access element (public member function) T& operator[] (const key_type& x); Creates a new key-value pair if key does not exist. Example: map<char,string> mymap; mymap['a']="an element"; Alternative access based on map iterator: (*it).first; (*it).second;
insert	Insert element (public member function) Example: mymap.insert (pair<char,int>('z',500));
string	
operator[]	Get character in string (public member function)
at	Get character in string (public member function)
operator+=	Append to string (public member function)
append	Append to string (public member function)
c_str	Get C string equivalent (public member function)
size	Return length of string (public member function)
find	Find content in string (public member function)
rfind	Find last occurrence of content in string (public member function)
find_first_of	Find character in string (public member function)
find_last_of	Find character in string from the end (public member function)
substr	Generate substring (public member function)
Various functions	
istream& getline (istream& is, string& str, char delim); Get line from stream (function)	
istream& getline (istream& is, string& str); Get line from stream (function)	
int get(); Extracts a character from the stream and returns its value (casted to an integer).	
istream& get (char& c); Extracts a character from the stream and stores it in c.	
istream& get (char* s, streamsize n);Extracts a character from the stream and stores it in c.	
int isalnum (int c); Check if character is alphanumeric. A value different from zero is returned (i.e., true) if indeed c is a white-space character. Zero (i.e., false) otherwise.	
int isalpha (int c); Check if character is alphabetic (function)	
int tolower (int c); Convert uppercase letter to lowercase	
int toupper (int c); Convert lowercase letter to uppercase	
int ispunct (int c); Check if character is a punctuation character	
int isspace Check if character is a white-space (function)	

Appendix 2: Bus routes (buss ruter)

46 Pirbadet - Munkegata - Tiller - Sandmoen

Avgangstider fra Pirbadet

Operatør: Nettbuss Trondheim

Time	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Mandag til fredag	20 35 50	05 20A) 35 50	05 20 35 50	05 20 40	00 20 40	00 20 40	00 20 40	00 20 40	00 20 35 50	05 20 35 50	05 20 35 50	05 20 40	00 20 40	00 30	00 30	00 30	00 30	00 30	00
Lørdag	35	05 35	05 35	05 35	00 20 40	00 20 40	00 20 40	00 20 40	00 20 40	00 20 40	00 20 40	00 30	00 30	00 30	00 30	00 30	00 30	00 30	00
Søndag					00 30	00 30	00 30	00 30	00 30	00 30	00 30	00 30	00 30	00 30	00 30	00 30	00 30	00 30	00

A) Holdeplassene fom. KVT tom. Rognbudalen betjenes ikke.
Istedet betjenes Tiller v.g.s. og Ivat Skjånes veg. Deretter vanlig trasé fra Ole Ross veg.

Tidligste passering av holdeplassene etter avgang fra **Pirbadet** i minutter. I perioder med mye trafikk vil bussen kunne passere noen minutter senere.

Pirbadet	Tollbua	Brattøra	Pohlhuset	Trondheim sentralst. hpl.10	Sandre gate 22	Munkegata M1	Prinsen kinosenter	Studentersamfundet	Einar Tambarskjelves gate	Prof. Brochs gate	Valøyvegen	Bratsbergvegen	Kroppan bru	Tonstadkrysset	Rostengrenda	John Aaes' veg	City Syd vestre	KVT	City Syd østre	Tonstadgrenda	Torvmyra	Moltmyra	Moltmyra øst	Ingeborg Ofstads veg	Martin Kreignes veg	Koieflata	Rognbudalen	Ole Ross veg	Torvtaket	Isdamveien	Djupmyra	Løvåsveien	Sandmoen
0	1	1	2	2	3	10	13	14	15	16	17	18	19	22	23	24	24	25	25	26	26	27	27	27	28	29	30	31	32	32	33	34	36

52 Vestlia/Othilienborg - Munkegata

Avgangstider fra Vestlia endeholdeplass

Operatør: Nettbuss Trondheim

Time	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Mandag til fredag	05 35	05 35	05 35	05 35	25	15	05 55	45	20 50	20 50	20 50	30	20	10	00 50	40	30	20
Lørdag	30	30	30	30	30	15 45	15 45	15 45	15 45	15 50	40	30	20	10	00 50	40	30	20
Søndag					00 50	40	30	20	10	00 50	40	30	20	10	00 50	40	30	20

Tidligste passering av holdeplassene etter avgang fra **Vestlia** i minutter. I perioder med mye trafikk vil bussen kunne passere noen minutter senere.

Vestlia endeholdeplass	Arne Bergsgårds veg	Edgar B. Schieldrups veg	Høeggen skole	Nardosenteret	Bjarne Ness veg	Einar Øfstis veg	K.O. Thornæs veg	Othilienborg	K.O. Thornæs veg	Einar Øfstis veg	Tors veg	Nardokrysset	Fiolsvingen	Dydalsvegen	Prestegårdsjordet	Gløshaugen syd	Gløshaugen nord	Høgskoleringen	Volladbakken	Studentersamfundet	Prinsen kinosenter	Munkegata M1
0	0	1	1	2	2	3	3	4	5	6	6	7	8	9	9	9	10	11	11	12	13	16