Kursusarbejde 4
Grundlæggende Programmering

Arne Jørgensen, 300473-2919
klasse dm032-1a

19. december 2003

Indhold

1. Kode

1.1. Header-filer

1.1.1. Queue.h

```
#ifndef QUEUE_H
#define QUEUE_H
#include <iostream>

struct frame
{
  int item;
  frame *next;
};

class Queue
{
 private:
  frame *head, *last;

 public:
  Queue();
  // pre: - post: an empty queue object is constructed

  Queue(const Queue& q);
  // pre: - post: the object and its data is copied

  virtual ~Queue();
  // pre: - post: frees all allocated memory from the heap

  Queue& operator = (const Queue& q);
  // pre: - post: the object has been assigned the values of f

  void add(const int& item);
  // pre: - post: item is added to the queue

  int remove();
  // pre: the queue must be non-empty
  // post: returns the longest waiting element and removes it from the queue

  bool isempty() const;
  // pre: - post: returns true if the queue is empty; returns false otherwise

  friend std::ostream& operator << (std::ostream& str, const Queue& q);
  // pre: if str is a file is it opened
  // post: each item is written to str followed by a blank and terminated by \n
};

#endif // QUEUE_H
```

1.1.2. Bog.h

```
#ifndef BOG_H
#define BOG_H
#include <string>
#include "queue.h"

class Bog
{
 private:
  std::string isbn, forfatter, titel;
  int antal_eksemplarer, udgivelsesaar, eksemplarer_hjemme;
  Queue reservationer;

 public:
  Bog();
  // pre: - post: default constructor; object skabt uden betydende oplysninger i

  Bog(std::string titel_);
  // pre: - post: objekt skabt med kun titel sat; primært til brug ved sammenligninger
```

```
    Bog(std::string isbn_, std::string forfatter_, std::string titel_, int antal_eksemplarer_, int udgivelsesaar_);
    // pre: – post: objekt skabt med argumenter som oplysninger; eksemplarer hjemme sat til antal eksemplarer i alt; primært til nye bøger

    Bog(std::string isbn_, std::string forfatter_, std::string titel_, int antal_eksemplarer_, int udgivelsesaar_, int eksemplarer_hjemme);
    // pre: – post: objekt skabt med argumenter som oplysninger; primært til indlæsning af bøger fra fil

    Bog(const Bog& b);
    // copy constructor

    virtual ~Bog();
    Bog& operator = (const Bog& b);
    friend bool operator < (const Bog& b1, const Bog& b2);
    friend bool operator == (const Bog& b1, const Bog& b2);

    std::string hent_isbn() const;
    std::string hent_forfatter() const;
    std::string hent_titel() const;
    int hent_antal_eksemplarer() const;
    int hent_udgivelsesaar() const;
    int hent_eksemplarer_hjemme() const;

    Queue& hent_reservationer();
    // pre: – post: returnerer reservations køen

    bool udlaan();
    // pre: – post: bogen registreres udlånt (og returnerer true) hvis der er eksemplarer hjemme; ellers false

    bool aflever();
    // pre: – post: bogen registreres hjemkommet (og returnerer true) hvis der er eksemplarer ude; ellers false

    bool reserver(int laaner_id);
    // pre: laaner_id er et unikt id for låneren (ej implementeret nærmere i denne opgave)
    // post: låner er sat i reservationskø (og returnerer true) hvis alle ekseplarer er udlånt; ellers false
};

#endif // BOG_H
```

1.1.3. Bogsamling.h

```
#ifndef BOGSAMLING_H
#define BOGSAMLING_H
#include "bog.h"
#include <fstream>

class BogSamling
{
 private:
   Bog *list;
   int used, max;
   bool grow();
   int find(const Bog& item);
   // pre: item indeholder (blot) titlen på den bog der ønskes fundet
   // post: returnerer positionen i arrayet; eller −1 hvis titlen ikke findes

   void save();
   // pre: – post: gem bøger og reservationer til næste kørsel (persistens)

   void load();
   // pre: – post: hent bøger og reservationer fra foregående kørsel (persistens)

 public:
   BogSamling();
   BogSamling(const BogSamling& sl);
   virtual ~BogSamling();
   BogSamling& operator = (BogSamling& sl);

   bool isempty() const;
   // pre: – post: returnerer true hvis der ikke er bøger i samlingen; ellers false

   bool insert(Bog& item);
   // pre: item er en bog med de nødvendige oplysninger sat
   // post: bogen indsættes hvis den ikke fandtes i forvejen (afgøres af titel); returnerer true hvis indsat; ellers false
```

```
35    bool remove(Bog& item);
      // pre: item er et objekt med (blot) titlen på den bog der ønskes fjernet
      // post: returnere true hvis bogen er fjernet; ellers (bogen fandtes ikke) false; item sættes samtidig til den fundne bog
};

40 #endif // BOGSAMLING_H
```

1.2. Klasseimplementeringer

1.2.1. Queue.cc

```
#include <cstdlib>
#include "queue.h"

Queue::Queue()
5 {
  head = NULL;
  last = NULL;
}

10 Queue::Queue(const Queue& q)
{
  head = NULL;
  last = NULL;

15  frame *temp = q.head;
  if ( !q.isempty() )
    {
      add(temp->item);
      while ( temp->next )
20      {
          temp = temp->next;
          add(temp->item);
        };
    }
25 }

Queue::~Queue()
{
  int temp;
30  while ( !isempty() )
    temp = remove();
}

Queue& Queue::operator = (const Queue& q)
35 {
  if (this == &q)
    return *this;

  while ( !isempty() )
40   remove();

  frame *temp = q.head;
  if ( !q.isempty() )
    {
45    add(temp->item);
      while ( temp->next )
        {
          temp = temp->next;
          add(temp->item);
50      };
    }

  return *this;
}
55
void Queue::add(const int& item)
{
  frame *temp;
  temp = new frame;
60  temp->item = item;
  temp->next = NULL;
  if ( isempty() )
```

```
    head = temp;
  else
    last->next = temp;
  last = temp;
}

int Queue::remove()
{
  if ( isempty() )
    exit(1);
  int item = head->item;
  frame *temp = head;
  head = temp->next;
  delete temp;

  return item;
}

bool Queue::isempty() const
{
  return !head;
}

std::ostream& operator << (std::ostream& str, const Queue& q)
{
  using namespace std;

  frame *temp = q.head;

  while (temp)
    {
      str << temp->item << ' ';
      temp = temp->next;
    };

  return str;
}
```

1.2.2. Bog.cc

```
#include "bog.h"
#include <string>
#include "queue.h"

Bog::Bog()
{
  isbn = "";
  forfatter = "";
  titel = "";
  antal_eksemplarer = 0;
  udgivelsesaar = 0;
  eksemplarer_hjemme = 0;
}

Bog::Bog(std::string titel_)
{
  isbn = "";
  forfatter = "";
  titel = titel_;
  antal_eksemplarer = 0;
  udgivelsesaar = 0;
  eksemplarer_hjemme = 0;
}

Bog::Bog(std::string isbn_, std::string forfatter_, std::string titel_, int antal_eksemplarer_, int udgivelsesaar_)
{
  isbn = isbn_;
  forfatter = forfatter_;
  titel = titel_;
  antal_eksemplarer = antal_eksemplarer_;
  udgivelsesaar = udgivelsesaar_;
  eksemplarer_hjemme = antal_eksemplarer_;
}
```

```
35 Bog::Bog(std::string isbn_, std::string forfatter_, std::string titel_, int antal_eksemplarer_, int udgivelsesaar_, int eksemplarer_hjemme_)
   {
     isbn = isbn_;
     forfatter = forfatter_;
     titel = titel_;
40   antal_eksemplarer = antal_eksemplarer_;
     udgivelsesaar = udgivelsesaar_;
     eksemplarer_hjemme = eksemplarer_hjemme_;
   }

45 Bog::Bog(const Bog& b)
   {
     isbn = b.isbn;
     forfatter = b.forfatter;
     titel = b.titel;
50   antal_eksemplarer = b.antal_eksemplarer;
     udgivelsesaar = b.udgivelsesaar;
     eksemplarer_hjemme = b.eksemplarer_hjemme;
     reservationer = b.reservationer;
   }
55
   Bog::~Bog() { /* itentionally left blank ... */ }

   Bog& Bog::operator = (const Bog& b)
   {
60   if (this == &b)
       return *this;

     isbn = b.isbn;
     forfatter = b.forfatter;
65   titel = b.titel;
     antal_eksemplarer = b.antal_eksemplarer;
     udgivelsesaar = b.udgivelsesaar;
     eksemplarer_hjemme = b.eksemplarer_hjemme;
     reservationer = b.reservationer;
70
     return *this;
   }

   bool operator < (const Bog& b1, const Bog& b2) { return (b1.titel < b2.titel); }
75
   bool operator == (const Bog& b1, const Bog& b2) { return (b1.titel == b2.titel); }

   std::string Bog::hent_isbn() const { return isbn; }

80 std::string Bog::hent_forfatter() const { return forfatter; }

   std::string Bog::hent_titel() const { return titel; }

   int Bog::hent_antal_eksemplarer() const { return antal_eksemplarer; }
85
   int Bog::hent_udgivelsesaar() const { return udgivelsesaar; }

   int Bog::hent_eksemplarer_hjemme() const { return eksemplarer_hjemme; }

90 Queue& Bog::hent_reservationer() { return reservationer; }

   bool Bog::udlaan()
   {
     if (eksemplarer_hjemme < 1)
95     return false;
     eksemplarer_hjemme--;
     return true;
   }

100 bool Bog::aflever()
   {
     if (eksemplarer_hjemme == antal_eksemplarer)
       return false;
     eksemplarer_hjemme++;
105  return true;
   }
```

```
bool Bog::reserver(int laaner_id)
{
  if (eksemplarer_hjemme > 0)
    return false;
  reservationer.add(laaner_id);
  return true;
}
```

1.2.3. Bogsamling.cc

```
#include "bogsamling.h"
#include <cstdlib>
#include <iostream>
#include "bog.h"

namespace {
  int initial_size = 50;
  int grow_by = 20;
  char *bookfile = "bogsamling.txt";
  char *resfile = "reservationer.txt";
}

BogSamling::BogSamling()
{
  list = new Bog[initial_size];
  used = 0;
  max = initial_size;
  load();
}

BogSamling::BogSamling(const BogSamling& sl)
{
  if (this == &sl)
    return;

  delete [] list;
  list = new Bog[initial_size];
  for (int i=0; i < sl.used; i++)
        insert(sl.list[i]);
}

BogSamling::~BogSamling()
{
  save();
  delete [] list;
}

BogSamling& BogSamling::operator = (BogSamling& sl)
{
  if (this == &sl)
    return *this;

  delete [] list;
  list = new Bog[initial_size];
  for (int i=0; i < sl.used; i++)
    insert(sl.list[i]);

  return *this;
}

bool BogSamling::grow()
{
  Bog* temp;
  temp = new Bog[max+grow_by];
  for (int i=0; i < used; i++)
    temp[i] = list[i];
  list = temp;
  delete temp;

  max += grow_by;
  return true;
}
```

```
bool BogSamling::isempty() const
{
  return !used;
}

bool BogSamling::insert(Bog& item)
{
  if (find(item) != -1)
    return false;

  if (used == max)
    grow();

  list[used] = item;

  Bog temp;
  for (int i=used; i > 0; i--)
    if (list[i] < list[i-1])
        {
          temp = list[i];
          list[i] = list[i-1];
          list[i-1] = temp;
        }
  used++;

  return true;
}

bool BogSamling::remove(Bog& item)
{
  int index = find(item);
  if (index == -1)
    return false;

  item = list[index];

  for (int i=index; i < used-1; i++)
    list[i] = list[i+1];
  used--;

  return true;
}

int BogSamling::find(const Bog& item) // binaer soegning
{
  int start=0, stop=used; int mid;
  while (start < stop)
    {
      mid = ((stop-start)/2)+start;
      if (list[mid] == item)
        return mid;
      else if (list[mid] < item)
        start = mid+1;
      else
        stop = mid;
    }
  return -1;
}

void BogSamling::save()
{
  using namespace std;

  ofstream save_file(bookfile);
  ofstream res_file(resfile);

  if (save_file.fail() || res_file.fail())
    return;

  for (int i=0; i < used; i++)
    {
      save_file << list[i].hent_isbn() << endl;
```

```
        save_file << list[i].hent_forfatter() << endl;
        save_file << list[i].hent_titel() << endl;
        save_file << list[i].hent_antal_eksemplarer() << endl;
        save_file << list[i].hent_udgivelsesaar() << endl;
        save_file << list[i].hent_eksemplarer_hjemme() << endl;

        res_file << list[i].hent_titel() << endl;
        res_file << list[i].hent_reservationer() << endl;
      }

  res_file.close();
  save_file.close();
}

void BogSamling::load()
{
  using namespace std;

  ifstream load_file(bookfile);
  ifstream res_file(resfile);

  string isbn, forfatter, titel;
  int antal_eksemplarer, udgivelsesaar, eksemplarer_hjemme;

  if (load_file.fail() || res_file.fail()) // silently ignore missing file(s) on start up
    return;

  while (!load_file.fail())
    {
      getline(load_file, isbn);

      if (!load_file.fail())
        {
          getline(load_file, forfatter);
          getline(load_file, titel);
          load_file >> antal_eksemplarer;
          load_file >> udgivelsesaar;
          load_file >> eksemplarer_hjemme;
          Bog temp(isbn, forfatter, titel, antal_eksemplarer, udgivelsesaar, eksemplarer_hjemme);
          insert(temp);
          load_file.ignore(1000, '\n');
        }
    }
  load_file.close();

  char next;
  int laaner_id;

  getline(res_file, titel);
  while (!res_file.fail())
    {
      Bog temp(titel);
      remove(temp);
      res_file.get(next);
      while (next != '\n')
        {
          res_file.putback(next);
          res_file >> laaner_id;
          temp.reserver(laaner_id);
          res_file.get(next);
          res_file.get(next);
        }
      insert(temp);
      getline(res_file, titel);
    }
  res_file.close();
}
```

1.3. Hovedprogram – bibliotek.cc

```
#include "bogsamling.h"
#include <iostream>
```

```
int main_menu();
void opret_bog(BogSamling& samling);
void slet_bog(BogSamling& samling);
void udlaan_bog(BogSamling& samling);
void aflever_bog(BogSamling& samling);
void reserver_bog(BogSamling& samling);
void vis_bog(BogSamling& samling);

int main()
{
  BogSamling samling;
  int valg;

  do
    {
      valg = main_menu();

      switch (valg)
        {
        case 1:
          opret_bog(samling);
          break;
        case 2:
          slet_bog(samling);
          break;
        case 3:
          udlaan_bog(samling);
          break;
        case 4:
          aflever_bog(samling);
          break;
        case 5:
          reserver_bog(samling);
          break;
        case 6:
          vis_bog(samling);
          break;
        }

    } while (valg != 7);

  return 0;
}
```

2. Om koden

Opgaveløsningen er delt op i hovedprogram med tilhørende frie funktioner og tre klasser indeholdende containere mm. (bogsamling, bog og reservationskø).

Bogsamlingen implementerer et sorteret, dynamisk array der kan indeholder bøger (objekter af klassen Bog). Det dynamiske array er valgt da det giver fleksibilitet med hensyn til størrelse i forhold til et statisk array. Bøgerne indsættes i arrayet sorteret efter titel hvilket kræver at operatorerne == og < er overloaded til at måle på titlen i bogklassen. Indsættelse og fjernelse af elementer i arrayet foretages i lineær tid, $O(n)$ og søgning i arrayet kan foretages i tiden $O(\log_2 n)$.

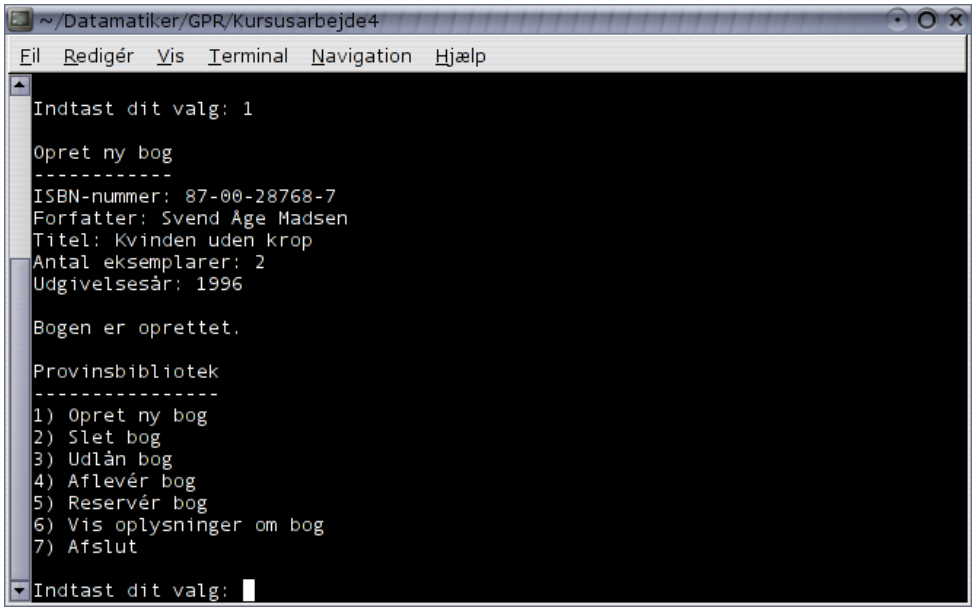
I bogklassen er implementeret de nødvendige attributter og medlemsfunktioner. Reservationskøen er implementeret som en attribut direkte i bogklassen (men dog med detaljerne i sin egen klasse). Det er valgt da reservationerne naturligt hører til som egenskaber ved bogen.

Reservationskøen er implementeret som en hægtet liste. Den hægtede liste har det dynamiske arrays fordel med hensyn til fleksibilitet. I modsætning til et dynamisk array er det med en kø implementeret som en hægtet liste muligt både at foretage indsættelse og fjernelse af elementer i konstant tid, $O(1)$.

Persistens er opnået ved at lade bogsamlingens konstruktor indlæse de persistente data fra filer og lade sammes destruktor gemme dataene. Datafilernes opbygning kan ses og er beskrevet nærmere i appendiks A på den følgende side.

Hovedprogrammet er opdelt ved at lade main-funktionen stå for kontrollen og de frie funktioner for brugerdialog udførelsen af den ønskede funktionalitet.

De frie funktioner udfører generelt deres operationer på et bogobjekt ved at tage bogen ud af samlingen. Udfører de nødvendige operationer/manipulationer og derefter sætter bogen ind i samlingen igen. Netop til



Figur 1: En bog oprettes.

denne brug er remove-funktionen i bogsamlingsklassen (bogsamling.h, linje 35) implementeret med en call-by-reference parameter.

Af hensyn til pladsforbruget er pre- og post-betingelser på de mest simple og intuitive funktioner udeladt (fx på get- og set-funktioner, samt udvalgte konstruktorer, destruktorer og *the big three*). Desuden er de frie funktioner i hovedprogrammet flyttet til appendiks B på side 13.

3. Afprøvning

Såvel det færdige program som de enkelte programdele har gennemgået en grundig afprøvning.

Løbende under udviklingen er funktionaliteten af de enkelte klasser og komponenter afprøvet med mindre driver-programmer der tester grænsetilfælde mm. Et eksempel på sådan et test-program og dets udførsel kan ses i appendiks C på side 16. Som det fremgår viser kørslen det ønskede og forventede resultat.

Ved at holde designet „rent“ og grænseflader mellem programmets dele klare har det været muligt at føre grundig test af de enkelte dele og det har været nemt at lokalisere hvilke kodedele fejl har befundet sig i.

Afprøvningen af det færdige program har ligeledes budt på diverse kørsler der afprøver grænsetilfælde mm. Fx er testen af persistensen bla. foretaget ved at gemme, indlæse og atter gemme dataene og derefter sammenligne datafilerne før og efter. Den færdige version består denne test

Af figurerne 1 til 3 på side 11–12 ses en række testscenarier og i appendiks A findes de tilhørende datafiler.

De afsluttende test har ikke afsløret fejl eller mangler.

A. Datafiler

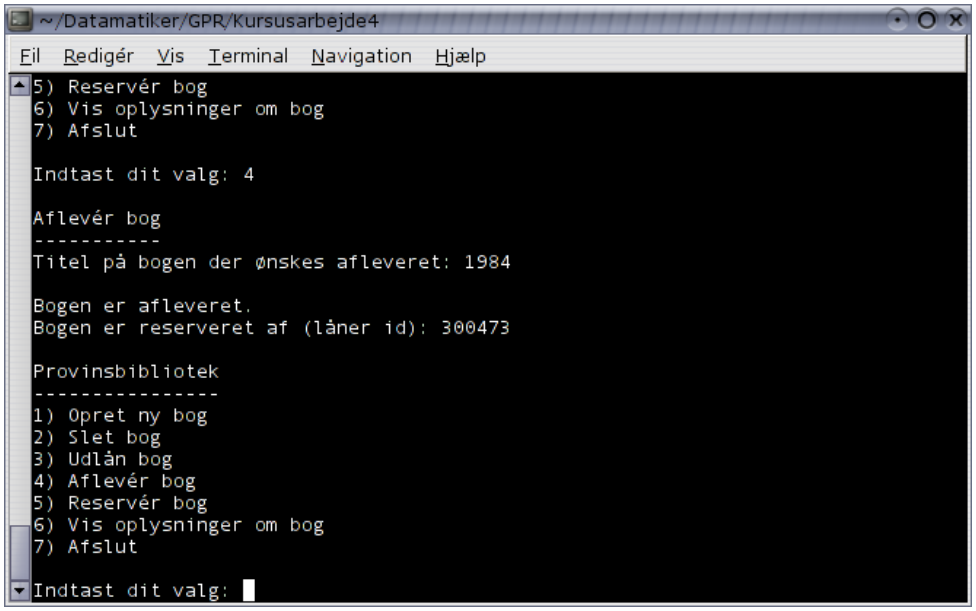
De persistente data skrives til to filer – bogsamling.txt og reservationer.txt.

Valget af to datafiler skyldes primært opgave formuleringen. I forhold til den valgte implementering i øvrigt ville det være lige så naturligt – og tilmed mere overskueligt – at skrive oplysninger om reservationerne på en bog sammen med bogens øvrige data.

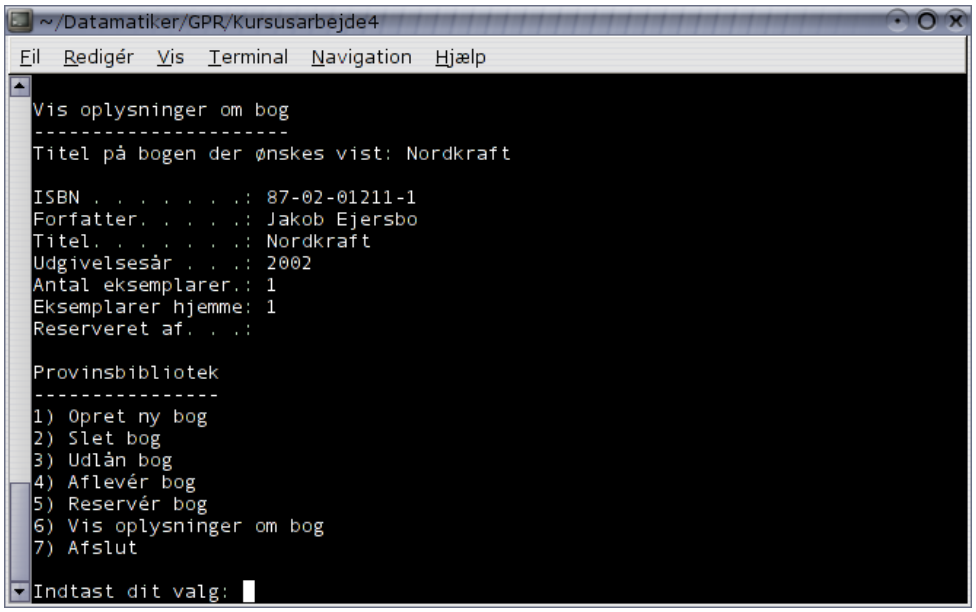
A.1. `bogsamling.txt`

Dataene om en bog skrives med en oplysning per linje i filen. Rækkefølgen er betydende og de kan derfor indlæses igen i samme rækkefølge (det antages at filen altid er korrekt opbygget).

```
87-00-01872-4
George Orwell
1984
```



Figur 2: Et reserveret bog bliver afleveret.



Figur 3: Oplysninger vises om en bog.

```
1
1949
0
87-00-28768-7
Svend Åge Madsen
Kvinden uden krop
2
1996
2
87-02-01211-1
Jakob Ejersbo
Nordkraft
1
2002
1
```

A.2. `reservationer.txt`

Reservationerne skrives til filen over to linjer. Første linje indeholder titlen på bogen og anden linje lånernes id efterfulgt af et mellemrum.

Bemærk at det er et efterfølgende mellemrum og der derfor også er et mellemrum efter det sidste låner id hvilket nødvendiggør en ekstra get under indlæsningen i bogsamling.cc, linje 195.

```
1984
300473 281280 200377
Kvinden uden krop

Nordkraft
```

B. Hovedprogrammets frie funktioner

```
int main_menu()
{
  using namespace std;

  int valg;

  cout << "\nProvinsbibliotek\n";
  cout << "----------------\n";
  cout << "1) Opret ny bog\n";
  cout << "2) Slet bog\n";
  cout << "3) Udlån bog\n";
  cout << "4) Aflevér bog\n";
  cout << "5) Reservér bog\n";
  cout << "6) Vis oplysninger om bog\n";
  cout << "7) Afslut\n";

  cout << "\nIndtast dit valg: ";
  cin >> valg;

  return valg;
}

void opret_bog(BogSamling& samling)
{
  using namespace std;

  string isbn, forfatter, titel;
  int antal_eksemplarer, udgivelsesaar;

  cin.ignore(1000, '\n'); // ryd op på input streamen
  cout << "\nOpret ny bog\n";
  cout << "------------\n";
```

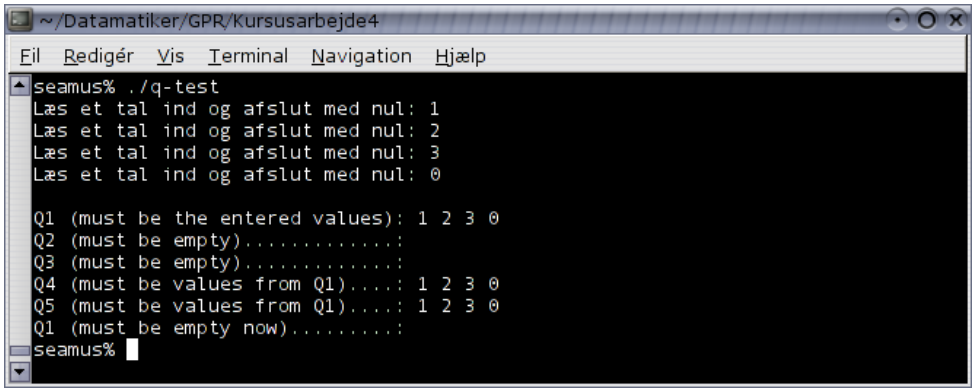
```
80    cout << "ISBN-nummer: ";
      getline(cin, isbn);
      cout << "Forfatter: ";
      getline(cin, forfatter);
      cout << "Titel: ";
85    getline(cin, titel);
      cout << "Antal eksemplarer: ";
      cin >> antal_eksemplarer;
      cout << "Udgivelsesår: ";
      cin >> udgivelsesaar;
90
      Bog temp(isbn, forfatter, titel, antal_eksemplarer, udgivelsesaar);

      if (samling.insert(temp))
        cout << "\nBogen er oprettet.\n";
95    else
        cout << "\nBogen er IKKE oprettet (den findes i forvejen).\n";
    }

    void slet_bog(BogSamling& samling)
100 {
      using namespace std;

      string titel;

105   cin.ignore(1000, '\n'); // ryd op på input streamen
      cout << "\nSlet bog\n";
      cout << "--------\n";
      cout << "Titel på bogen der ønskes slettet: ";
      getline(cin, titel);
110
      Bog temp("", "", titel, 0, 0);

      if (samling.remove(temp))
        cout << "\nBogen er slettet.\n";
115   else
        cout << "\nBogen findes ikke og er derfor IKKE slettet.\n";
    }

    void udlaan_bog(BogSamling& samling)
120 {
      using namespace std;

      string titel;

125   cin.ignore(1000, '\n'); // ryd op på input streamen
      cout << "\nUdlån bog\n";
      cout << "---------\n";
      cout << "Titel på bogen der ønskes lånt: ";
      getline(cin, titel);
130
      Bog temp("", "", titel, 0, 0);

      if (!samling.remove(temp))
        {
135       cout << "\nBogen findes ikke og er derfor IKKE lånt.\n";
          return;
        }

      if (temp.udlaan())
140     cout << "\nBogen er nu udlånt.\n";
      else
        cout << "\nBogen kunne ikke lånes da der ikke er flere eksemplarer hjemme.\n";

      if (!samling.insert(temp))
145     cout << "\nUkendt fejl (ved indsættelse af udlånt bog).\n";
    }

    void aflever_bog(BogSamling& samling)
    {
150   using namespace std;

      string titel;
```

```
  cin.ignore(1000, '\n'); // ryd op på input streamen
  cout << "\nAflevér bog\n";
  cout << "-----------\n";
  cout << "Titel på bogen der ønskes afleveret: ";
  getline(cin, titel);

  Bog temp("", "", titel, 0, 0);

  if (!samling.remove(temp))
    {
      cout << "\nBogen findes ikke og kan IKKE afleveres.\n";
      return;
    }

  if (temp.aflever())
    {
      cout << "\nBogen er afleveret.\n";
      if ((temp.hent_reservationer()).isempty())
        cout << "Bogen er IKKE reserveret." << endl;
      else
        cout << "Bogen er reserveret af (låner id): " << (temp.hent_reservationer()).remove() << endl;
    }
  else
    cout << "\nBogen kunne IKKE afleveres da der ikke er nogen eksemplarer udlånt.\n";

  if (!samling.insert(temp))
    cout << "\nUkendt fejl (ved indsættelse af afleveret bog).\n";
}

void reserver_bog(BogSamling& samling)
{
  using namespace std;

  string titel;
  int laaner_id;

  cin.ignore(1000, '\n'); // ryd op på input streamen
  cout << "\nReservér bog\n";
  cout << "------------\n";
  cout << "Titel på bogen der ønskes reserveret: ";
  getline(cin, titel);

  Bog temp("", "", titel, 0, 0);

  if (!samling.remove(temp))
    {
      cout << "Bogen findes ikke og er derfor IKKE reserveret.\n";
      return;
    }

  cout << "Reserver til (låner id): ";
  cin >> laaner_id;

  if (temp.reserver(laaner_id))
    cout << "\nBogen er reserveret.\n";
  else
    cout << "\nBogen er IKKE reserveret da der er flere eksemplarer hjemme.\n";

  if (!samling.insert(temp))
    cout << "\nUkendt fejl (ved indsættelse af reserveret bog).\n";
}

void vis_bog(BogSamling& samling)
{
  using namespace std;

  string titel;

  cin.ignore(1000, '\n'); // ryd op på input streamen
  cout << "\nVis oplysninger om bog\n";
  cout << "----------------------\n";
  cout << "Titel på bogen der ønskes vist: ";
```

```
  getline(cin, titel);

  Bog temp(titel);

  if (!samling.remove(temp))
    {
      cout << "Bogen findes ikke og oplysninger kan derfor IKKE vises.\n";
      return;
    }

  cout << endl;
  cout << "ISBN . . . . . .: " << temp.hent_isbn() << endl;
  cout << "Forfatter. . . . .: " << temp.hent_forfatter() << endl;
  cout << "Titel. . . . . .: " << temp.hent_titel() << endl;
  cout << "Udgivelsesår . . .: " << temp.hent_udgivelsesaar() << endl;
  cout << "Antal eksemplarer.: " << temp.hent_antal_eksemplarer() << endl;
  cout << "Eksemplarer hjemme: " << temp.hent_eksemplarer_hjemme() << endl;
  cout << "Reserveret af. . .: " << temp.hent_reservationer() << endl;

  if (!samling.insert(temp))
    cout << "\nUkendt fejl (ved indsættelse af vist bog).\n";
}
```

C. Driverprogram til test af kø

```
#include "queue.h"
#include <iostream>

int main()
{
  using namespace std;

  Queue Q1;

  Queue Q2(Q1);
  Queue Q3;
  Q3 = Q1;

  int temp;

  // test add
  do
    {
      cout << "Læs et tal ind og afslut med nul: ";
      cin >> temp;
      Q1.add(temp);
    } while (temp != 0);

  // test copy constructor
  Queue Q4(Q1);
  Queue Q5;
  Q5 = Q1;

  // test remove
  cout << "\nQ1 (must be the entered values): ";
  while ( !Q1.isempty() )
    cout << Q1.remove() << ' ';

  cout << "\nQ2 (must be empty).............: ";
  while ( !Q2.isempty() )
    cout << Q2.remove() << ' ';

  cout << "\nQ3 (must be empty).............: ";
  while ( !Q3.isempty() )
    cout << Q3.remove() << ' ';

  cout << "\nQ4 (must be values from Q1)....: ";
  while ( !Q4.isempty() )
    cout << Q4.remove() << ' ';

  cout << "\nQ5 (must be values from Q1)....: ";
  while ( !Q5.isempty() )
```



Figur 4: Driverprogram der tester en kø.

```
    cout << Q5.remove() << ' ';

50  cout << "\nQ1 (must be empty now).....: ";
    while ( !Q1.isempty() )
      cout << Q1.remove() << ' ';
    cout << endl;
}
```