Kursusarbejde 3
Grundlæggende Programmering

Arne Jørgensen, 300473-2919
klasse dm032-1a

21. november 2003

Indhold

1. Kode

1.1. forestillinger.h

```
#ifndef FORESTILLINGER_H
#define FORESTILLINGER_H
#include <string>
#include <iostream>

const int BORDE = 48;
const int PLADSER = 12;
const int MAXFORESTILLINGER = 4;

class Reservation
{
 private:
  int nr;
  int antal;

 public:
  Reservation();
  Reservation(int nr_, int antal_);
  int Nummer();
  int Antal();
};

class Forestilling
{
 private:
  std::string titel; // string−klassen i GNU C++ ligger i std namespacet
  int dato;
  int tid;
  int antal_reservationer;
  Reservation reservationer[BORDE∗PLADSER];

  int reserverede_pladser();
  // Post: returnerer antallet af reserverede pladser til forestillingen

 public:
  Forestilling();
  // default konstruktør

  Forestilling(std::string titel_, int dato_, int tid_);
  // Pre: dato_ indeholder en gyldig dato i formatet ÅÅÅÅMMDD og tid_ et gyldigt klokkeslæt i formatet HHMM
  // Post: en forestilling oprettes med de ønskede data

  int antalledige();
  // Post: resturnerer antallet af ledige pladser til forstillingen

  int reserver(int antal);
  // Pre: antal er det antal pladser der ønskes reserveret
  // Post: foretager reservationen så fremt reservation er mulig og
  // returnerer reservationsnummer eller −1 hvis reservation ikke er
  // muligt (fx ikke nok ledige pladser)

  std::string Titel();
  // Post: returnerer forestillingens titel

  int Dato();
  // Post: returnerer forestillingens spilledato som et heltal på formen ÅÅÅÅMMDD

  int Tid();
  // Post: returnerer forestillingens spilletidpunkt som et heltal på formen HHMM

  friend bool operator<(const Forestilling& f1, const Forestilling& f2);
  // Post: er sand hvis forestilling f1 finder sted før forestilling f2

  friend bool operator==(const Forestilling& f1, const Forestilling& f2);
  // Post: er sand hvis forestilling f1 og forestilling f2 finder sted samtidig

  friend std::ostream& operator« (std::ostream& str, const Forestilling& forestilling);
  // Pre: hvis str er en fil er filen åbnet
```

```
    // Post: udskriver oplysninger om forestilling på streamen med ' '
    // mellem felterne til nem, senere indlæsning med cin
};

class Spilleplan
{
 private:
   Forestilling spilleplan[MAXFORESTILLINGER];
   int antal;

 public:
   Spilleplan();
   // default kontrsuktør

   bool tilfoej(const Forestilling& forestilling);
   // Pre: forestilling indeholder et objekt med alle oplysninger om forestillingen lagt i
   // Post: returnerer true hvis forestilling er tilfoejet og sorteringen opretholdt

   bool nedlaeg(const int index);
   // Post: returnerer true hvis forestilling[i] er fjernet og sorteringen opretholdt

   int udvaelg(const int dato, const int klokkeslaet);
   // Pre: dato er på formen ÅÅÅÅMMDD og klokkeslaet på formen HHMM
   // Post: returnerer indekset på den ønskede forestilling eller −1 hvis den ikke findes

   void dump(std::ostream& forestilling);
   // Pre: hvis str er en fil er filen åbnet
   // Post: udskrivet alle oplysninger om alle forestillinger til str

   Forestilling& operator[](const int index);
   // Pre: index skal være et gyldigt indeks ( fundet med udvaelg() )
   // Post: returnerer det ønskede forestillingsobjekt
};

#endif // FORESTILLINGER_H
```

1.2. forestillinger.cc

```
#include <fstream>
#include "forestillinger.h"

Forestilling::Forestilling()
{
  titel = "";
  dato = 0;
  tid = 0;
  antal_reservationer = 0;
}

Forestilling::Forestilling(std::string titel_, int dato_, int tid_)
{
  titel = titel_;
  dato = dato_;
  tid = tid_;
  antal_reservationer = 0;
}

std::string Forestilling::Titel()
{
  return titel;
}

int Forestilling::Dato()
{
  return dato;
}

int Forestilling::Tid()
{
  return tid;
}
```

```
35  int Forestilling::reserverede_pladser()
    {
      int antal = 0;
      for (int i=0; i < antal_reservationer; i++)
        antal += reservationer[i].Antal();
40    return antal;
    }

    int Forestilling::antalledige()
    {
45    return (BORDE * PLADSER) - reserverede_pladser();
    }

    int Forestilling::reserver(int antal)
    {
50
      using namespace std;

      int reservationsnummer = ((((reserverede_pladser()+1)/12)+1)*100)
                               + ((reserverede_pladser()+1)%12);
55
      if (antal > antalledige() || antal <= 0)
        return -1;

      Reservation reservation(reservationsnummer, antal);
60    reservationer[antal_reservationer] = reservation;
      antal_reservationer++;

      ofstream reservationsfil("reservationer.dat", ios::app);
      if (!reservationsfil.fail())
65        reservationsfil << reservationsnummer << ' '
                          << dato << ' '
                          << tid << ' '
                          << antal << endl;

70    return reservationsnummer;
    }

    bool operator<(const Forestilling& f1, const Forestilling& f2)
    {
75    return ((f1.dato < f2.dato) || ((f1.dato == f2.dato) && (f1.tid < f2.tid)));
    }

    bool operator==(const Forestilling& f1, const Forestilling& f2)
    {
80    return ((f1.dato == f2.dato) && (f1.tid == f2.tid));
    }

    std::ostream& operator<< (std::ostream& str, const Forestilling& forestilling)
    {
85    using namespace std;

      str << forestilling.dato << ' '
          << forestilling.tid << ' '
          << forestilling.antal_reservationer << ' '
90        << forestilling.titel;

      return str;
    }

95  Reservation::Reservation()
    {
      nr = 0;
      antal = 0;
    }
100
    Reservation::Reservation(int nr_, int antal_)
    {
      nr = nr_;
      antal = antal_;
105 }

    int Reservation::Nummer()
```

```
{
  return nr;
}

int Reservation::Antal()
{
  return antal;
}

Spilleplan::Spilleplan()
{
  antal = 0;
}

bool Spilleplan::tilfoej(const Forestilling& forestilling)
{
  // er der overhovedet plads til en ny forestilling?
  if (antal >= MAXFORESTILLINGER)
    return false;

  // findes forestillingen allerede?
  for (int i=0; i < antal; i++)
    if (forestilling == spilleplan[i])
      return false;

  // sæt foreløbigt forestillingen ind på første ledige plads ...
  spilleplan[antal] = forestilling;

  // ... og bobble den så ned på plads
  Forestilling temp;
  for (int i=antal; i > 0; i--)
    {
      if (spilleplan[i] < spilleplan[i-1])
        {
          temp = spilleplan[i];
          spilleplan[i] = spilleplan[i-1];
          spilleplan[i-1] = temp;
        }
    }
  antal++;
  return true;
}

bool Spilleplan::nedlaeg(const int index)
{
  if (index < 0 || index > antal)
    return false;

  for (int i=index; i < antal-1; i++)
    spilleplan[i] = spilleplan[i+1];
  antal--;
  return true;
}

int Spilleplan::udvaelg(const int dato, const int klokkeslaet)
{
  Forestilling temp("", dato, klokkeslaet);

  for (int i=0; i < antal; i++)
    if (spilleplan[i] == temp)
      return i;
  return -1;
}

void Spilleplan::dump(std::ostream& forestilling)
{
  using namespace std;

  for (int i=0; i < antal; i++)
    forestilling << spilleplan[i] << endl;
 }

Forestilling& Spilleplan::operator[](const int index)
```

```
{
  if (index < 0 || index > antal)
    exit(1); // der blev forsøgt at løbe udenfor arrayet
  return spilleplan[index];
}
```

1.3. teater.cc

```
#include <iostream>
#include <fstream>
#include "forestillinger.h"

int menu();
void opretForestilling(Spilleplan& spilleplan);
void nedlaegForestilling(Spilleplan& spilleplan);
void reserverPladser(Spilleplan& spilleplan);
void ledigePladser(Spilleplan& spilleplan);

int main()
{
  Spilleplan spilleplan;

  int valg;

  do {
    valg = menu();

    switch (valg)
      {
      case 1:
        opretForestilling(spilleplan);
        break;
      case 2:
        nedlaegForestilling(spilleplan);
        break;
      case 3:
        reserverPladser(spilleplan);
        break;
      case 4:
        ledigePladser(spilleplan);
        break;
      };

  } while (valg != 9);

  std::ofstream forestilling("forestilling.dat", std::ios::app);

  if (forestilling.fail())
    exit(1); // kunne ikke åben forestillinger.dat
  spilleplan.dump(forestilling);

  return 0;
}

int menu()
{
  using namespace std;

  int valg;

  do {
    cout << "\nMenu\n----\n\n";
    cout << "1: Opret forestilling\n";
    cout << "2: Nedlæg forestilling\n";
    cout << "3: Reservér pladser til forestilling\n";
    cout << "4: Antal ledige pladser til forestilling\n";
    cout << "\n9: Afslut\n";
    cout << "\nØnske: ";
    cin >> valg;
  } while (valg != 1 && valg != 2 && valg != 3 && valg != 4 && valg != 9);
  return valg;
}
```

```
void opretForestilling(Spilleplan& spilleplan)
{
  using namespace std;

  string titel;
  int dato, klokkeslaet;

  getline(cin, titel); // ryd op på streamen før indlæsning med getline

  cout << "\nOpret forestilling\n------------------\n\n";
  cout << "Titel: ";
  getline(cin, titel);
  cout << "Dato (ÅÅÅÅMMDD): ";
  cin >> dato;
  cout << "Klokkeslæt (HHMM): ";
  cin >> klokkeslaet;

  Forestilling forestilling(titel, dato, klokkeslaet);
  if (spilleplan.tilfoej(forestilling))
    cout << '\n' << titel << " er tilføjet spilleplanen,\n";
  else
    cout << '\n' << titel << " kunne ikke tilføjes spilleplanen,\n";
}

void nedlaegForestilling(Spilleplan& spilleplan)
{
  using namespace std;

  int dato, klokkeslaet;

  cout << "\nNedlæg forestilling\n------------------\n\n";
  cout << "Dato (ÅÅÅÅMMDD): ";
  cin >> dato;
  cout << "Klokkeslæt (HHMM): ";
  cin >> klokkeslaet;

  int udvalgt = spilleplan.udvaelg(dato, klokkeslaet);
  if (udvalgt != -1)
    {
      if (spilleplan.nedlaeg(udvalgt))
        cout << "\nForestillingen er nedlagt.\n";
      else
        cout << "\nForestilling kunne ikke nedlægges.\n"; // burde ikke ske
    }
  else
    cout << "Ingen forestilling d. " << dato << " klokken " << klokkeslaet << ".\n";
}

void reserverPladser(Spilleplan& spilleplan)
{
  using namespace std;

  int dato, klokkeslaet, antal;

  cout << "\nReservér pladser til forestilling\n------------------\n\n";
  cout << "Dato (ÅÅÅÅMMDD): ";
  cin >> dato;
  cout << "Klokkeslæt (HHMM): ";
  cin >> klokkeslaet;
  cout << "Antal pladser: ";
  cin >> antal;

  int udvalgt = spilleplan.udvaelg(dato, klokkeslaet);
  if (udvalgt != -1)
    {
      int reservationsnummer = spilleplan[udvalgt].reserver(antal);
      if (reservationsnummer != -1)
        {
          cout << "\nReservationsnummer " << reservationsnummer << ":\n";
          cout << antal << " pladser til " << spilleplan[udvalgt].Titel() << " d. " << dato << " kl. " << klokkeslaet << ".\n";
        }
      else
        {
```

```
          cout << "Kunne ikke reservere " << antal << " pladser til " << spilleplan[udvalgt].Titel() << " d. " << dato << " kl. " << klokkeslaet << ".\n"
        }
    }
  else
    cout << "\nIngen forestilling d. " << dato << " klokken " << klokkeslaet << ".\n";
}

void ledigePladser(Spilleplan& spilleplan)
{
  using namespace std;

  int dato, klokkeslaet;

  cout << "\nLedige pladser til forestilling\n------------------\n\n";
  cout << "Dato (ÅÅÅÅMMDD): ";
  cin >> dato;
  cout << "Klokkeslæt (HHMM): ";
  cin >> klokkeslaet;

  int udvalgt = spilleplan.udvaelg(dato, klokkeslaet);
  if (udvalgt != -1)
    {
      cout << "\nDer er " << spilleplan[udvalgt].antalledige() << " ledige pladser";
    }
  else
    cout << "\nIngen forestilling d. " << dato << " klokken " << klokkeslaet << ".\n";
}
```

1.4. Om koden

Hovedprogrammet (`teater.cc`) indeholder en kort main-løkke der kalder frie funktioner afhængig af brugerens valg fra en menu (også fri funktion).

De frie funktioner tager sig af indlæsning af oplsyninger fra brugeren og kalder dernæst den(de) rette funktioner på spilleplanen (objekt).

Klasserne (reservation, foredrag, spilleplan) er forsøgt holdt på et generelt niveau. Det vil sige de oplysningerne om foredrag mm. er kapslet ind i objekterne og kan kun tilgås gennem funktioner der „skjuler“ den underliggende implementation.

Fx var min umiddelbare tanke at implementere foredragsklassen med et 48-array til at repræsentere pladserne ved bordene. Men da vi ikke har brug for særlige oplysninger om det enkelte plads er det kun informationen om hvorvidt pladsen er reserveret eller ej der skal repræsenteres i et sådant array (fx ved en bool). Da pladserne dog fordeles fra bord 1, plads 1 og fremefter kan vi dog nøjes med oplysningen om hvormange pladser der er reserveret (som jeg også gør i denne opgave).

Selvom første version af min opgaveløsning indeholdt et 48-arraym kunne klassen programmeres om til den nuværende løsning uden at ændre klassens interface.

Spilleplanen er designet så man kan udvælge en enkelt forestilling efter dato og klokkeslæt og herefter udføre operationer på denne forestilling (gennem overload af `[]`). Denne metode er valgt fremfor generelt at sende dato og klokkeslæt gennem et lag i spilleplansklassen.

2. Datafiler

2.1. `forestilling.dat`

Filen indeholder en linje per forestilling med oplysninger: dato, klokkelæt, antal reservationer og titel.

```
20031122 1400 2 Dyrene i Hakkebakkeskoven
20031124 1930 0 Phantom of the opera
20031125 1930 0 Den ugudelige farce
```

Ved at lægge titlen sidst er det muligt at indlæse dataene i filen igen med læsninger fra inputstrømmen med `>>` og dernæst resten af linjen med string-klassens `getline()`.

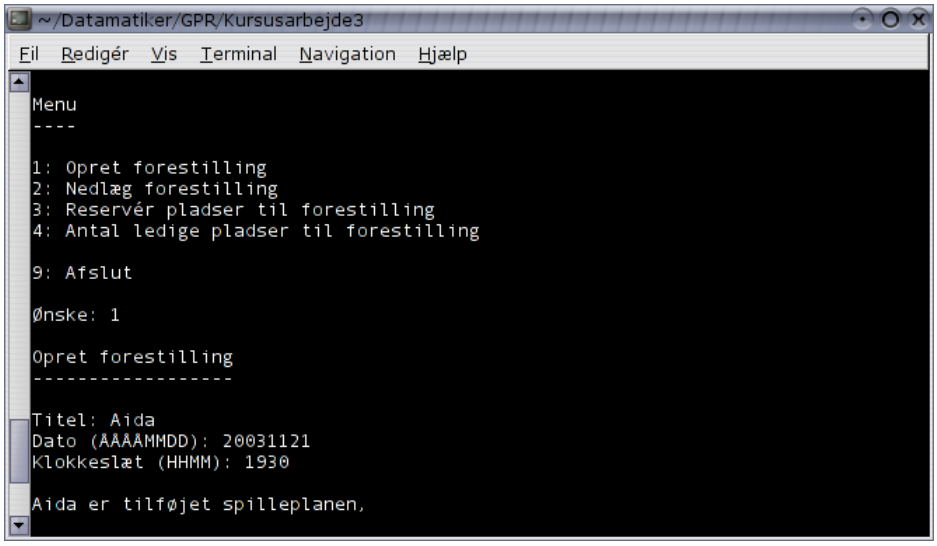
2.2. `reservationer.dat`

```
101 20031122 1400 20
209 20031122 1400 54
```

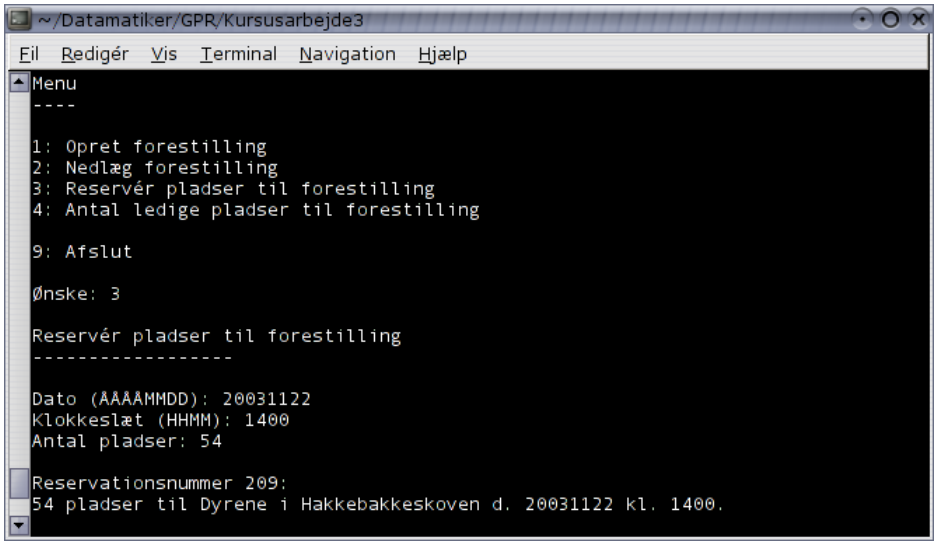
3. Afprøvning

Programmet er afprøvet med en række testdata. I figur 1 til 3 på den følgende side ses skærmbilleder fra et par af afprøvningerne Resultatet af et par flere yderligere afprøvninger fremgår af `forestilling.dat` og `reservationer.dat`, se afsnit 2.1 til 2.2 på foregående side.

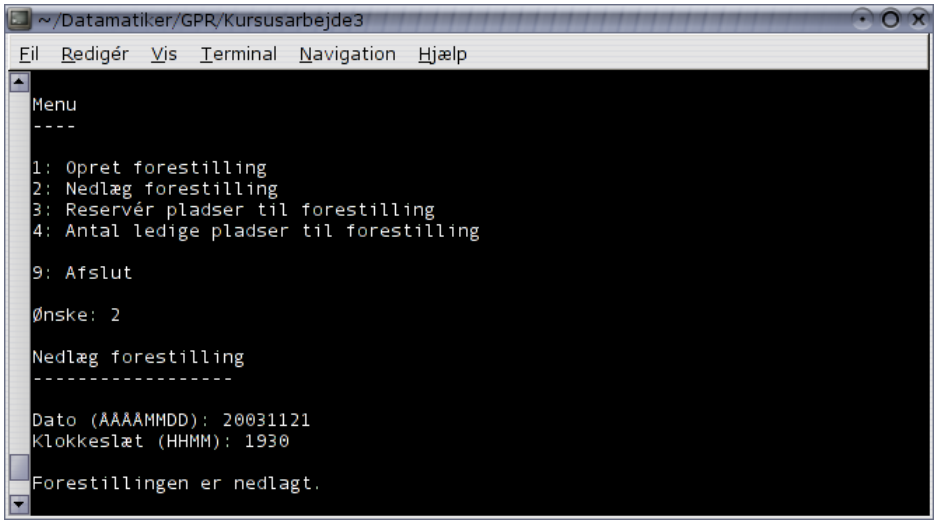
Programmets beregninger, konstrolstrukturer, etc., er endvidere under udviklingen afprøvet ved hjælp af diverse driver-programmer. Disse er ikke vedlagt.



Figur 1: En forestilling oprettes i spilleplanen).



Figur 2: Et antal pladser reserveres til en forestilling.



Figur 3: En forestilling i spilleplanen nedlægges.