```cpp
/**
 *  Obligatorisk oppgave nr 2.
 *
 *  @file OBLIG2.CPP
 *  @author Loke Svelland
 */


#include <iostream>                 //  cout, cin
#include <string>                   //  string
#include <vector>                   //  vector
#include "LesData2.h"
using namespace std;


/*
 *  Enum 'aktivitetsType' (med hva slags aktivitet dette er).
 */
enum aktivitetsType {Jobb, Fritid, Skole, ikkeAngitt};


/**
 *  Baseklassen 'Aktivitet' (med navn og aktivitetstype).
 */
class Aktivitet  {
  private:
    string navn;
    aktivitetsType kategori;
  public:
    Aktivitet()  {  navn = ""; kategori = ikkeAngitt; }
    void lesData();
    void skrivData() const;
};


/**
 *  Subklassen 'Tidsbegrenset' (med tidspunkter for start/stopp av aktivitet).
 */
class Tidsbegrenset : public Aktivitet  {
  private:
    int  startTime, startMin, sluttTime, sluttMin;
    bool klokkeslettOK(const int time, const int minutt) const;
  public:
    Tidsbegrenset()  { sluttMin = sluttTime = startTime = startMin = 0; };
    void lesData();
    void skrivData() const;
};


/**
 *  Subklassen 'Heldags' (med nÃ¦rmere beskrivelse av aktiviteten).
 */
class Heldags : public Aktivitet  {
  private:
    string beskrivelse;
  public:
    Heldags()  { beskrivelse = ""; };
    void lesData();
    void skrivData() const;
};


/**
 *  Selvlaget container-klasse 'Dag' (med dato og ulike aktiviteter).
 */
class Dag  {
```

```
  private:
    int dagNr, maanedNr, aarNr;
    vector <Tidsbegrenset*> tidsbegrensedeAktiviteter;
    vector <Heldags*> heldagsAktiviteter;

  public:
    Dag() {  };
    Dag(const int dag, const int maaned, const int aar) {
                           dagNr = dag;  maanedNr = maaned;  aarNr = aar; };
    ~ Dag();
    bool harDato(const int dag, const int maaned, const int aar) const;
    void nyAktivitet();
    void skrivAktiviteter() const;
    void skrivDato() const;
};


bool dagOK(const int dag, const int maaned, const int aar);
Dag* finnDag(const int dag, const int maaned, const int aar);
void frigiAllokertMemory();
void nyAktivitet();
void skrivDager(const bool inkludertAktiviteter);
void skrivEnDag();
void skrivMeny();


vector <Dag*> gDagene;           ///<  Dager med aktiviteter


/**
 *  Hovedprogrammet:
 */
int main ()  {
    char kommando;

    skrivMeny();
    kommando = lesChar("\nKommando");

    while (kommando != 'Q')  {
        switch (kommando)  {
          case 'N': nyAktivitet();      break;
          case 'A': skrivDager(true);   break;
          case 'S': skrivEnDag();       break;
          default:  skrivMeny();        break;
        }
        kommando = lesChar("\nKommando");
    }

    frigiAllokertMemory();

    return 0;
}


// ------------------------------------------------------------------------
//                       DEFINISJON AV KLASSE-FUNKSJONER:
// ------------------------------------------------------------------------

/**
 *  Leser inn ALLE klassens data.
 */
void Aktivitet::lesData() {

  cout << "\n\tAktivitets navn: ";    getline(cin, navn);
  int aktivitetstype = lesInt("\n\tType aktivitet (1=Jobb) (2=Fritid) (3=Skole) (4=ikke
Angitt)", 1, 4);
```

```
                        // Leser inn kategori
  switch(aktivitetstype) {
    if        (aktivitetstype == 1) this->kategori = Jobb;
    else if   (aktivitetstype == 2) this->kategori = Fritid;
    else if   (aktivitetstype == 3) this->kategori = Skole;
    else if   (aktivitetstype == 4) this->kategori = ikkeAngitt;
  }
}


/**
 *  Skriver ut ALLE klassens data.
 */
void Aktivitet::skrivData() const {
                        // Skriver ut kategori
  cout << "\naktiviteten: " << this->navn << " er av kategori ";
  switch(kategori) {
    case 1:          cout << "Jobb\n";          break;
    case 2:          cout << "Fritid\n";        break;
    case 3:          cout << "skole\n";         break;
    case 4:          cout << "ikke Angitt\n";   break;
  }
  cout << '\n';
}


/**
 *  Leser inn ALLE klassens data, inkludert morklassens data.
 *
 *  @see   Aktivitet::lesData()
 *  @see   klokkeslettOK(...)
 */
void Tidsbegrenset::lesData() {

                        // leser innn felles data
  Aktivitet::lesData();

                        // leser inn start timer og minutt og slutt tid
  startTime = lesInt("\n\tStart time på aktivitet: ", 0, 23);
  startMin  = lesInt("\n\tStart minutt på aktivitet: ", 0, 59);
  sluttTime = lesInt("\n\tSlutt time på aktivitet: ", startTime, 23);
  sluttMin  = lesInt("\n\tSlutt minutt på aktivitet: ", 0, 59);

                        // sjekker at tid er lovlig
  klokkeslettOK(startTime, startMin);
  klokkeslettOK(sluttTime, sluttMin);
}


/**
 *  Privat funksjon som finner ut om input er et lovlig klokkeslett.
 *
 *  @param   time    -  Timen som skal evalueres til mellom 0 og 23
 *  @param   minutt  -  Minuttet som skal evalueres til mellom 0 og 59
 *  @return  Om parametrene er et lovlig klokkeslett eller ei
 */
bool Tidsbegrenset::klokkeslettOK(const int time, const int minutt) const {

  if((0 <= time <= 23) && (0 <= minutt  <= 59)) {
    return true;
  } else return false;
}


/**
 *  Skriver ut ALLE klassens data, inkludert morklassens data.
```

```
 *
 *  @see   Aktivitet::skrivData()
 */
void Tidsbegrenset::skrivData() const {         //  Skriver mor-klassens data.

                        // skriver ut felles data
  Aktivitet::skrivData();

  cout << "Aktiviteten starter: ";
  if((startTime <= 9) && (startMin <= 9)) {
    cout << "0" << startTime << ":" << "0" << startMin;
  } else if((startTime <= 9) && (startMin >= 10)) {
     cout << "0" << startTime << ":" << startMin;
  } else if((startTime >= 10) && (startMin >= 10)) {
     cout << startTime << ":" << startMin;
  } else cout << startTime << ":" << "0" << startMin;

  cout << '\n';

  cout << "Aktiviteten slutter: ";
  if((sluttTime <= 9) && (sluttMin <= 9)) {
    cout << "0" << sluttTime << ":" << "0" << sluttMin;
  } else if((sluttTime <= 9) && (sluttMin >= 10)) {
     cout << "0" << sluttTime << ":" << sluttMin;
  } else if((sluttTime >= 10) && (sluttMin >= 10)) {
     cout << sluttTime << ":" << sluttMin;
  } else cout << sluttTime << ":" << "0" << sluttMin;
}


/**
 *  Leser inn ALLE klassens data, inkludert morklassens data.
 *
 *  @see   Aktivitet::lesData()
 */
void Heldags::lesData() {

  Aktivitet::lesData();

  cout << "\n\tBeskriv aktiviteten: ";    getline(cin, beskrivelse);
}


/**
 *  Skriver ut ALLE klassens data, inkludert morklassens data.
 *
 *  @see   Aktivitet::skrivData()
 */
void Heldags::skrivData() const {

  Aktivitet::skrivData();

  cout << "\nAktivitetens beskrivelse: \n\t" << this->beskrivelse;
}


/**
 *  Destructor som sletter HELT begge vectorenes allokerte innhold.
 */
Dag :: ~ Dag() {

                        // går gjennom hver av vectorene og sletter elementene
                        // og all allokert minne
  for(int i = 0; i < tidsbegrensedeAktiviteter.size(); i++)
    delete tidsbegrensedeAktiviteter[i];
  tidsbegrensedeAktiviteter.clear();
```

```
  for(int i = 0; i < heldagsAktiviteter.size(); i++)
    delete heldagsAktiviteter[i];
  heldagsAktiviteter.clear();
}


/**
 *  Finner ut om selv er en gitt dato eller ei.
 *
 *  @param   dag     -  Dagen som skal sjekkes om er egen dag
 *  @param   maaned  -  Måneden som skal sjekkes om er egen måned
 *  @param   aar     -  året som skal sjekkes om er eget år
 *  @return  Om selv er en gitt dato (ut fra parametrene) eller ei
 */
bool Dag::harDato(const int dag, const int maaned, const int aar) const {

  if((dag == dagNr) && (maaned == maanedNr) && (aar == aarNr)) {
    return true;
  } else return false;

}


/**
 *  Oppretter, leser og legger inn en ny aktivitet på dagen.
 *
 *  @see   Tidsbegrenset::lesData()
 *  @see   Heldags::lesData()
 */
void Dag::nyAktivitet()  {
                        // lager pekere til vectorene
  Tidsbegrenset* tidsbeg;
  Heldags* heldag;

  char valg;

  valg = lesChar("\n\tHvilken aktivitet skal opprettes (T/H) (T(idsbegrenset/H(elsdags))"
);
                        // hvis T leser inn data til denne typen
  if(valg == 'T') {
    tidsbeg = new Tidsbegrenset;
    tidsbeg->lesData();
    tidsbegrensedeAktiviteter.push_back(tidsbeg);

                        // hvis H leser inn data til denne typen
  } else if(valg == 'H') {
    heldag = new Heldags;
    heldag->lesData();
    heldagsAktiviteter.push_back(heldag);
  }

}


/**
 *  Skriver ut ALLE aktiviteter på egen dato (og intet annet).
 *
 *  @see   Heldags::skrivData()
 *  @see   Tidsbegrenset::skrivData()
 */
void Dag::skrivAktiviteter() const {

  int i;
                        // skriv3er ut alt innhold i vecotrene
  for(i = 0; i < tidsbegrensedeAktiviteter.size(); i++) {
    cout << "\nTidbegrenset aktiviteter på denne dagen: "; tidsbegrensedeAktiviteter[i]->
```

```
skrivData();
  }
  for(i = 0; i < heldagsAktiviteter.size(); i++) {
    cout << "\nHeldagsaktiviteter på denne dagen: ";
    heldagsAktiviteter[i]->skrivData();
  }

}


/**
 *  Skriver KUN ut egen dato.
 */
void Dag::skrivDato() const {

  cout << "\nDagens dato: " << dagNr << '.' << maanedNr << '.' << aarNr;
}


// ------------------------------------------------------------------------
//                      DEFINISJON AV ANDRE FUNKSJONER:
// ------------------------------------------------------------------------

/**
 *  Returnerer om en dato er lovlig eller ei.
 *
 *  @param   dag     -  Dagen som skal sjekkes
 *  @param   maaned  -  Måneden som skal sjekkes
 *  @param   aar     -  året som skal sjekkes
 *  @return  Om datoen er lovlig/OK eller ei
 */
bool dagOK(const int dag, const int maaned, const int aar)  {

  if((dag <= 31) && (maaned <= 12) && (1990 <= aar <= 2030)) {
    return true;
  } else return false;
}


/**
 *  Returnerer om mulig en peker til en 'Dag' med en gitt dato.
 *
 *  @param   dag     -  Dagen som skal bli funnet
 *  @param   maaned  -  Måneden som skal bli funnet
 *  @param   aar     -  året som skal bli funnet
 *  @return  Peker til aktuell Dag (om funnet), ellers 'nullptr'
 *  @see     harDato(...)
 */
Dag* finnDag(const int dag, const int maaned, const int aar)  {

  for(int i = 0; i < gDagene.size(); i++) {
    if(gDagene[i]->harDato(dag, maaned, aar) == true) {
      return gDagene[i];
    } else return nullptr;
  }

}


/**
 *  Frigir/sletter ALLE dagene og ALLE pekerne i 'gDagene'.
 */
void frigiAllokertMemory()  {

  for(int i = 0; i < gDagene.size(); i++)
    delete gDagene[i];
```

```
  gDagene.clear();

}


/**
 *  Legger inn en ny aktivitet på en (evt. ny) dag.
 *
 *  @see   skrivDager(...)
 *  @see   dagOK(...)
 *  @see   finnDag(...)
 *  @see   Dag::nyAktivitet()
 */
void nyAktivitet()  {

  int dag,
      maaned,
      aar;

  Dag* nydag;           // peker til vector

  skrivDager(false);    // skriver ut alle dagar registrert

                        // leser inn dato til aktivitet
  cout << "\nlegg inn dato aktiviteten skal holdes: ";
  dag     = lesInt("\nDag: ", 1, 31);
  maaned  = lesInt("\nMåned: ", 1, 12);
  aar     = lesInt("\nÅr: ", 1990, 2030);

                        // hvis dagOK skal den lese inn data
  if(dagOK(dag, maaned, aar) == true) {
    if(finnDag(dag, maaned, aar) == nullptr) {
      nydag = new Dag(dag, maaned, aar);
      gDagene.push_back(nydag);
      nydag->nyAktivitet();
      nydag->skrivDato();
    } else {
      for(int i = 0; i < gDagene.size(); i++) {
        if(gDagene[i] == finnDag(dag, maaned, aar)) {
          gDagene[i]->nyAktivitet();
        }
      }
    }
  }


}


/**
 *  Skriver ut ALLE dagene (MED eller UTEN deres aktiviteter).
 *
 *  @param   inkludertAktiviteter - Utskrift av ALLE aktivitetene også, eller ei
 *  @see     Dag::skrivDato()
 *  @see     Dag::skrivAktiviteter()
 */
void skrivDager(const bool inkludertAktiviteter)  {

  if(gDagene.size() == 0) {
    cout << "\n\tIngen dager lagt til\n\n\n";
  } else {              // hvis false blir sendt med skal den bare skrive dato
    for(int i = 0; i < gDagene.size(); i++) {
      if(inkludertAktiviteter == false) {
        gDagene[i]->skrivDato();
      } else {          // hvis true så blir dato og data skrevet ut
        gDagene[i]->skrivDato();
```

```
        gDagene[i]->skrivAktiviteter();
      }
    }
  }

}


/**
 *  Skriver ut ALLE data om EN gitt dag.
 *
 *  @see   skrivDager(...)
 *  @see   dagOK(...)
 *  @see   finnDag(...)
 *  @see   Dag::skrivAktiviteter()
 */
void skrivEnDag()  {
  int dag,
      maaned,
      aar;

  skrivDager(false);

                        // legger inn dato som skal sjekkes
  cout << "\nHvilken dag vil du sjekke";
  dag     = lesInt("\nDag: ", 1, 31);
  maaned  = lesInt("\nMåned: ", 1, 12);
  aar     = lesInt("\nÅr: ", 1990, 2030);

  if(dagOK(dag, maaned, aar) == true) {
    if(finnDag(dag, maaned, aar) == nullptr) {
      cout << "\nDag finnes ikke\n\n\n";
    } else {
      for(int i = 0; i < gDagene.size(); i++) {
                          // hvis dag finnes skrives ut data
        gDagene[i]->skrivAktiviteter();
      }
    }
  }

}


/**
 *  Skriver programmets menyvalg/muligheter på skjermen.
 */
void skrivMeny()  {
   cout << "\nDisse kommandoene kan brukes:\n"
        << "\tN - Ny aktivitet\n"
        << "\tA - skriv ut Alle dager med aktiviteter\n"
        << "\tS - Skriv EN gitt dag og (alle) dens aktiviteter\n"
        << "\tQ - Quit / avslutt\n";
```