```
/**
 *  @file OBLIG2reinnlevering.CPP
 *  @author Sebastian Instanes Skylstad.
 *
 *  Hovedfunksjonen med programmet.
 *  Programmet er en kalender/aktivitet tracker som kan holde
 *  styr på oppkommende eller tidligere gjennomførte aktiviteter.
 *  Dette gjøres ved hjelp av fire klasser.
 *  -   Baseklassen Aktivtet og to subklasser (Tidsbegrenset og Heldags)
 *  -   Aktivitet-Objektene kan legges inn i klassen Dag sine to vectorer,
 *      sammen med dato.
 */


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

    frigiAllokertMemory();

    return 0;
}


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

/**
 *  Leser inn ALLE klassens data.
 */
```

```
void Aktivitet::lesData() {
    char svar;      //Temp variabel

    cout << "\tAktivitet navn: ",   getline (cin, navn); //aktivitet navn
    //Looper så lenge urelevant svar
    do {
        svar = lesChar("\tJ(jobb), F(Fritid), S(Skole), I(ikke Angitt)");
}while(svar != 'J' && svar != 'F' && svar != 'S' && svar != 'I');

    // Bruker switch for å konvertere input til riktig enum output
    switch(svar) {
        case 'J': kategori = Jobb;          break;
        case 'F': kategori = Fritid;        break;
        case 'S': kategori = Skole;         break;
        default : kategori = ikkeAngitt;    break;
    }

}


/**
 *  Skriver ut ALLE klassens data.
 *  Skriver ut aktivtitetens daga, navn, kategori
 */
void Aktivitet::skrivData() const {
    cout << "\tAktivitetnavn: " << navn <<"\n";
    cout << "\tAktivitets kategori: ";
    //Switch for å konvertere enum verdi til riktig output
    switch (kategori) {
        case Jobb   :   cout << "Jobb\n";           break;
        case Fritid :   cout << "Fritid\n";         break;
        case Skole  :   cout << "Skole\n";          break;
        default     :   cout << "ikke Angitt\n";    break;
    }
}


/**
 *  Leser inn ALLE klassens data, inkludert morklassens data.
 *
 *  @see   Aktivitet::lesData()
 *  @see   klokkeslettOK(...)
 *
 *  Siden en aktivitet ikke kan slutte etter midnatt
 *  antar jeg att den heller ikke kan starte ved/etter midnatt
 *  av den grunn velger jeg å sette start tiden til tidligst 6 om morgenen.
 *
 *
 *  Bruker nå lesInt til å sørge for at sluttMin er +1 større enn startMin
 *  hvis sluttTime og startTime har samme verdi
 */
void Tidsbegrenset::lesData() {
    Aktivitet::lesData();           // Henter aktivitet klassens lesData

    // Sjekker først lovlige verdier for startTime og startMin
    do {        //do while, sikkrer lovlige verider
        startTime   = lesInt ("Start time", 6, 23);         //Time start
        startMin    = lesInt ("Start minutt", 0, 59);       //minutt start
    } while (!klokkeslettOK(startTime, startMin));

    do {
        sluttTime   = lesInt ("Slutt time", startTime, 23); //time slutt
        //Sørger for at startMin er større enn sluttMin dersom timeSlutt
        //og timeStart har samme verdi.
        if (startTime == sluttTime) {
            sluttMin = lesInt ("Slutt minutt", startMin+1, 59); //Minutt slutt
```

```
            }
        else {      //Hvis sluttTime > startTime
            sluttMin = lesInt ("Slutt minutt", 0, 59);      //minutt slutt
        }
    } while (!klokkeslettOK(sluttTime, sluttMin));
    //Looper så lenge den returnerer false.
}


/**
 *  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
 *
 *  Sjekker at start og slutt Time er innenfor intervallet 0-23
 *  Sjekker at start og slutt minutt er innenfor intervallet 0-59
 *  Hvis det returneres true godkjennes det av funksjonen over.
 *
 *
 */
bool Tidsbegrenset::klokkeslettOK(const int time, const int minutt) const {

    if (time >= 0 && time <= 23 && minutt >= 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.
    Aktivitet::skrivData(); //Skriver ut morklassens data.

    //skriver ut aktivitet start på dette formatet (hh:mm);
    cout << "\tAktivitet start: ";
    cout << ((startTime < 10)   ? "0" : "") << startTime << ":";
    cout << ((startMin < 10)    ? "0" : "") << startMin << "\n";

    cout << "\tAktivitet slutt: ";
    cout << ((sluttTime < 10)   ? "0" : "") << sluttTime << ":";
    cout << ((sluttMin < 10)    ? "0" : "") << startMin << "\n\n";
}


/**
 *  Leser inn ALLE klassens data, inkludert morklassens data.
 *
 *  @see   Aktivitet::lesData()
 *  leser inn klassens data, deretter beskrivelse av klassen
 */
void Heldags::lesData() {
    Aktivitet::lesData();   //Henter aktivitet klassens lesData
    cout << "\tBeskrivelse: ";  getline(cin,beskrivelse);
}


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

```
 *
 *  @see    Aktivitet::skrivData()
 *  skriver ut klassens data, deretter beskrivelse av klassen
 */
void Heldags::skrivData() const {
    Aktivitet::skrivData(); //Henter aktivitet klassens skrivData
    cout << "\tHeldags beskrivelse: " << beskrivelse << "\n";

}


/*
 *  Destructor som sletter HELT begge vectorenes allokerte innhold.
 */
Dag :: ~ Dag() {
    //Looper så lenge den ikke er tom
    while(!tidsbegrensedeAktiviteter.empty()){
        delete tidsbegrensedeAktiviteter[tidsbegrensedeAktiviteter.size() - 1];
        tidsbegrensedeAktiviteter.pop_back();   //fjerner bakerste element
    }
    //Looper så lenge den ikke er tom
    while(!heldagsAktiviteter.empty()){
        delete heldagsAktiviteter[heldagsAktiviteter.size() - 1];
        heldagsAktiviteter.pop_back();
    }
}


/**
 *  Finner ut om selv er en gitt dato eller ei.
 *
 *  @param   dag     -  Dagen som skal sjekkes om er egen dag
 *  @param   maaned  -  MÃ¥neden som skal sjekkes om er egen mÃ¥ned
 *  @param   aar     -  Ã…ret som skal sjekkes om er eget Ã¥r
 *  @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 {
    //Hvis match, return true;
    if(dagNr == dag && maanedNr == maaned && aarNr == aar) {
            return true;
        }
    else {
        return false;   //Hvis den ikke finner en match
    }
}


/**
 *  Oppretter, leser og legger inn en ny aktivitet pÃ¥ dagen.
 *
 *  @see    Tidsbegrenset::lesData()
 *  @see    Heldags::lesData()
 */
void Dag::nyAktivitet()  {
    char svar;
    cout << "\tHvilken aktivited skal opprettes?\n";
    //Looper så lenge svaret ikke er T eller H
    do {
        svar = lesChar("\tT(idsbegrenset), H(eldags)");
    } while (svar != 'T' && svar != 'H');

    //Opretter ett nyy objekt, objektet for lest inn sin data
    //Deretter pushbacker i vektor..
    if (svar == 'T') {
        Tidsbegrenset* t; t = new Tidsbegrenset;
        t->lesData();
```

```
        tidsbegrensedeAktiviteter.push_back(t);
    }
    //Samme framgangsmåte
    if (svar == 'H') {
        Heldags* h; h = new Heldags;
        h->lesData();
        heldagsAktiviteter.push_back(h);
    }
}


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

    //Skriver ut alle tidsbegrensede aktiviteter
    if (!tidsbegrensedeAktiviteter.empty()){
        cout << "\nSkriver Tidsbegrensede Aktiviteter: \n";
        for (int i = 0; tidsbegrensedeAktiviteter.size(); i++){
            tidsbegrensedeAktiviteter[i]->skrivData();
        }
    }
    //Skriver ut alle heldags aktiviteter
    if (!heldagsAktiviteter.empty()){
        cout << "\nSkriver Heldags Aktiviteter: \n";
        for (int i = 0; heldagsAktiviteter.size(); i++) {
            heldagsAktiviteter[i]->skrivData();
        }
    }
}


/**
 *  Skriver KUN ut egen dato.
 *  skriver ut dato på dette formatet: dd.mm/år
 */
void Dag::skrivDato() const {
    cout << "\t Skriver ut dagens dato\n";
    cout << dagNr << "." << maanedNr << "/" << aarNr << "\n";

}


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

/**
 *  Returnerer om en dato er lovlig eller ei.
 *
 *  @param   dag     -  Dagen som skal sjekkes
 *  @param   maaned  -  MÃ¥neden som skal sjekkes
 *  @param   aar     -  Ã…ret som skal sjekkes
 *  @return  Om datoen er lovlig/OK eller ei
 */
bool dagOK(const int dag, const int maaned, const int aar)  {
    // returnerer true om dag er mellom 1-31,
    // maaned er mellom 1-12,
    // aar er mellom 1990-2030
    // ellers returnerer false
    if (dag <= 31 && dag >= 1 && maaned <= 12
            && maaned >= 1 && aar <= 2030 && aar >= 1990) {
```

```
        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  -  MÃ¥neden som skal bli funnet
 *  @param   aar     -  Ã…ret 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)  {
    // itererer gjennom, leter etter medsendte parametere i Dag klasse,
    // returnerer dagen den fant om det er en match
    for (int i = 0; i < gDagene.size(); i++ ) {
        if (gDagene[i]->harDato(dag,maaned,aar)){
            return gDagene[i];
        }
    }
    return nullptr;
}



/**
 *  Frigir/sletter ALLE dagene og ALLE pekerne i 'gDagene'.
 */
void frigiAllokertMemory()  {
    // Looper så lenge det er noe i vektor
    while (!gDagene.empty()) {
        delete gDagene[gDagene.size() - 1]; // Frigir allokert memory
        gDagene.pop_back();                 // Fjerner bakerste element
    }
}



/**
 *  Legger inn en ny aktivitet pÃ¥ en (evt. ny) dag.
 *
 *  @see   skrivDager(...)
 *  @see   dagOK(...)
 *  @see   finnDag(...)
 *  @see   Dag::nyAktivitet()
 */
void nyAktivitet()  {
    int aar, mnd, dag;  //temp variabler

    skrivDager(false);  //Skriver ut dagenes dato

    //Looper så lenge bruker skriver inn ulovelige verdier
    do {
        aar = lesInt("Aar",1990,2030);
        mnd = lesInt("Maand",1,12);
        dag = lesInt("dag",1,31);
    }while(!dagOK(dag,mnd,aar));    //Sender med som parameter for sjekk

    //Hvis dagen ikke finnes må vi lage ett nytt objekt
    if (!finnDag(dag,mnd,aar)) {
```

```
        cout << "Opretter ny dag\n";
        cout << "Legg til aktivitetre\n";
        Dag* d; d = new Dag(dag,mnd,aar);
        d->nyAktivitet();
        gDagene.push_back(d);

    }
    //Hvis dagen finnes men vi ønsker flere aktiviteter
    else {
        cout <<"Dagen finnes allerede\n";
        cout <<"legger til flere aktiviteter\n";
        Dag* finnes;        //temp variabel hvis dagen finnes
        finnes = finnDag(dag,mnd,aar);
        finnes->nyAktivitet();
    }
}


/**
 *  Skriver ut ALLE dagene (MED eller UTEN deres aktiviteter).
 *
 *  @param   inkludertAktiviteter - Utskrift av ALLE aktivitetene ogsÃ¥, eller ei
 *  @see     Dag::skrivDato()
 *  @see     Dag::skrivAktiviteter()
 */
void skrivDager(const bool inkludertAktiviteter)  {
    //Hvis det er tomt for dager
    if (gDagene.empty()) {
        cout << "\tDet er tomt for dagr\n";
    }

    // Skriver bare ut dagens dato HVIS dagen ikke inneholder aktiviteter
    if (!inkludertAktiviteter) {
        for (int i = 0; i < gDagene.size(); i++){
            gDagene[i]->skrivDato();
        }
    }
    //Ellers itererer gjennom alle dagene of skriver ut dataen.
    else {
        for (int i = 0; i < gDagene.size(); i++) {
            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, mnd, aar;  // temp variabler
    Dag* enDag;         //temp peker
                        //Skriver ut alle lagrede datoer.
    skrivDager(false);

    // Spør etter innput til temp variabler
    do {
        dag = lesInt("Dag", 1, 31);
        mnd = lesInt("Mnd", 1, 12);
        aar = lesInt("Aar", 1990, 2030);
    } while (!dagOK(dag,mnd,aar));
```

```
    //Søker etter dag med finnDag funksjon
    enDag = finnDag(dag, mnd, aar);
    //Hvis vi ikke finner dag
    if (!enDag) {
        cout << "\tFinner ikke dagen\n\n";
    }
    //Hvis vi finner dagen
    else {
        enDag->skrivAktiviteter();
    }
}


/**
 *  Skriver programmets menyvalg/muligheter pÃ¥ 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";
}
```