

```

/**
 * Programmet er en kalender der man kan legge inn heldags og
 * tidsbegrensede aktiviteter på spesifikke dager.
 *
 * Hovedfunksjonalitet:
 * - Inneholder klassen 'Aktivitet' og dens to subclasser
 *   'Tidsbegrenset' og 'Heldags'. Objekter av de to siste klassene legges
 *   inn for HVER ENKELT DAG inn i to ulike vectorer inni den selvlagede
 *   containerklassen: 'Dag'
 * - De tre første klassene inneholder alle constructorer og funksjoner
 *   for å lese og skrive alle objektets data.
 * - 'Dag' inneholder en del ulike funksjoner for å håndtere datastrukturen
 *   inni seg. Det er disse medlemsfunksjonene som kalles i fra funksjonene
 *   som startes/kalles fra 'main' for EN gitt dag.
 * - Den globale vektoren 'gDagene' inneholder ALLE DE ULIKE DAGENE
 *   med hver sine ulike aktiviteter.
 *
 * @file OBLIG2.CPP
 * @author Kristupas Kaupas
 */

#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 nærmere 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 padSiffer(int tall);
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() {
    int aktivitetInput;
    cout << "Skriv inn navn paa aktivitet \ninput: ";
    getline(cin, navn);
    cout << "Skriv inn aktivitet\n";
    cout << "0 - Jobb \n1 - Fritid\n2 - Skole\n3 - Ikke angitt\n";
    aktivitetInput = lesInt("Velg aktivitet", 0, 3);
    switch(aktivitetInput){
        case 0:
            kategori = Jobb;
            break;
        case 1:
            kategori = Fritid;
            break;
        case 2:
            kategori = Skole;
            break;
        case 3:
            kategori = ikkeAngitt;
            break;
    }
}

/**
 * Skrriver ut ALLE klassens data.
 */
void Aktivitet::skrivData() const {
    cout << "Navn | Kategori \n";
    cout << navn << " | ";
    switch(kategori){
        case 0:
            cout << "Jobb";
            break;
        case 1:
            cout << "Fritid";
            break;
        case 2:
            cout << "Skole";
            break;
        case 3:
            cout << "Ikke angitt";
            break;
    }
    cout << "\n";
}

/**
 * Leser inn ALLE klassens data, inkludert morklassens data.
 *
 * @see Aktivitet::lesData()
 * @see klokkeslettOK(...)
 */
void Tidsbegrenset::lesData() {
    int inputStartTime, inputStartMin, inputSluttTime, inputSluttMin;
    int startTotalMin, sluttTotalMin;
    bool fortsett = true; //flag for å stoppe loop

    Aktivitet::lesData();
    while (fortsett) {
        cout << "Skriv inn starttidspunkt: \n";
        inputStartTime = lesInt("Skriv inn time ", 0, 23);
        inputStartMin = lesInt("Skriv inn minutt ", 0, 59);
        if (klokkeslettOK(inputStartTime, inputStartMin)) { //Vil alltid være true
            cout << "Klokkeslett er ok. \n";
        }

        cout << "Skriv inn slutt tidspunkt: \n";
        inputSluttTime = lesInt("Skriv inn time ", 0, 23);
        inputSluttMin = lesInt("Skriv inn minutt ", 0, 59);
        if (klokkeslettOK(inputSluttTime, inputSluttMin)) { //Vil alltid være true
            cout << "Klokkeslett er ok \n";
        }
    }

    //Gjør om til min for å
    // sammennligne start og slutt tid

```

```

startTotalMin = (inputStartTime * 60) + inputStartMin;
sluttTotalMin = (inputSluttTime * 60) + inputSluttMin;
if (startTotalMin < sluttTotalMin) {
    //lagre data
    startTime = inputStartTime; startMin = inputStartMin;
    sluttTime = inputSluttTime; sluttMin = inputSluttMin;

    fortsett = false;          //Stopper while loop om alt var greit
}
else {
    cout << "Start tidspunktet kan ikke være før slutt tidspunktet! \n";
}
}
}

```

```

/**
 * 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 (
        (time >= 0 && time <= 23) &&
        (minutt >= 0 && minutt <= 59)
    )
    {
        return true;
    }
    return false;
}

```

```

/**
 * Skriver ut ALLE klassens data, inkludert morklassens data.
 *
 * @see Aktivitet::skrivData()
 * @see padSiffer(...)
 */

```

```

void Tidsbegrenset::skrivData() const {          // Skriver mor-klassens data.
    Aktivitet::skrivData();                      //skriver ut data med null
                                                // foran om det er ensifret tall
    padSiffer(startTime); cout << ":"; padSiffer(startMin); cout << "-";
    padSiffer(sluttTime); cout << ":"; padSiffer(sluttMin); cout << "\n";
}

```

```

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

```

```

void Heldags::lesData() {
    Aktivitet::lesData();
    cout << "Skriv inn en beskrivelse paa aktiviteten: \n";
    getline(cin, beskrivelse);
}

```

```

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

```

```

void Heldags::skrivData() const {
    Aktivitet::skrivData();
    cout << "Beskrivelse: " << beskrivelse << "\n";
}

```

```

/**
 * Destructor som sletter HELT begge vectorenes allokerete innhold.
 */

```

```

Dag::~~Dag() {
    for (Tidsbegrenset* aktivitet : tidsbegrensedeAktiviteter) {
        delete aktivitet;
    }

    for (Heldags* aktivitet : heldagsAktiviteter) {
        delete aktivitet;
    }

    //tømmer selve pointer vektorene
    tidsbegrensedeAktiviteter.clear();
    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;
    }
    return false;
}

```

```
/**
 * Oppretter, leser og legger inn en ny aktivitet på dagen.
 *
 * @see Tidsbegrenset::lesData()
 * @see Heldags::lesData()
 */
void Dag::nyAktivitet() {
    char input;
    bool run = true; //flag for å stoppe loop
    while (run) {
        cout << "Velg aktivitetstype:\n" <<
            "\tT - Tidsbegrenset aktivitet\n" <<
            "\tH - Heldags aktivitet\n";
        input = lesChar("input");

        if (input == 'T') {
            run = false;
            Tidsbegrenset* nyAktivitet = new Tidsbegrenset();
            nyAktivitet->lesData();
            tidsbegrensedeAktiviteter.push_back(nyAktivitet);
        }
        else if (input == 'H') {
            run = false;
            Heldags* nyAktivitet = new Heldags();
            nyAktivitet->lesData();
            heldagsAktiviteter.push_back(nyAktivitet);
        }
    }
}

/**
 * Skriver ut ALLE aktiviteter på egen dato (og intet annet).
 *
 * @see Heldags::skrivData()
 * @see Tidsbegrenset::skrivData()
 */
void Dag::skrivAktiviteter() const {
    cout << "\n----- Tidsbegrensede aktiviteter ----- \n";
    for (int i = 0; i < tidsbegrensedeAktiviteter.size(); i++) {
        tidsbegrensedeAktiviteter[i]->skrivData();
    }
    cout << "\n";
    cout << "----- Heldags aktiviteter ----- \n";

    for (int i = 0; i < heldagsAktiviteter.size(); i++) {
        heldagsAktiviteter[i]->skrivData();
    }
}

/**
 * Skriver KUN ut egen dato.
 */
void Dag::skrivDato() const {
    cout << 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 >= 1 && dag <= 31) &&
        (maaned >= 1 && maaned <= 12) &&
        (aar >= 1990 && aar <= 2030)
    )
    {
        return true;
    }
    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)) {
            return gDagene[i]; //return av funnet pointer
        }
    }
    return nullptr; //ingen dager funnet med lik dato
}

/**
```

```
* Frigir/sletter ALLE dagene og ALLE pekerne i 'gDagene'.
*/
void frigiAllokertMemory() {
    for (Dag* dag : gDagene) {
        delete dag;                //sletter dataen pointers peker til
    }
    gDagene.clear();              //tømmer hele vektoren
}

/**
 * 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* dagPtr;
    skrivDager(false);           //Skriver først ut alle lagrede datoer SkrivDager()

    dag = lesInt("skriv inn dag ", 0, 31);
    maaned = lesInt("Skriv inn maaned ", 0, 12);
    aar = lesInt("Skriv inn aar ", 1990, 2030);
    if (dagOK(dag, maaned, aar)) { //vil alltid gi true pga lesint
        dagPtr = finnDag(dag, maaned, aar);
        if (dagPtr == nullptr) { //opprett ny dag med dato
            Dag* nydag = new Dag(dag, maaned, aar); //allokerer ny memory
            gDagene.push_back(nydag); //lagrer nydag
            dagPtr = nydag; //gir pointer til ny dag hvis den ikke eksisterte
        }

        dagPtr->nyAktivitet(); //leses inn ny aktivitet
    }
}

/**
 * Skriver ut ensifrete tall med 0 foran.
 *
 * @param tall
 */
void padSiffer(int tall) {
    if (tall < 10) {
        cout << "0" << tall;
    }
    else {
        cout << tall;
    }
}

/**
 * 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) {
    int size = gDagene.size();
    if (size == 0) {
        cout << "Det er ingen dager lagret \n";
    }
    else {
        cout << "Dag\n" << "-----\n";
        for (int i = 0; i < size; i++) {
            cout << "\n";
            gDagene[i]->skrivDato();
            if (inkludertAktiviteter) { gDagene[i]->skrivAktiviteter(); }
            cout << "\n";
        }
    }
}

/**
 * Skriver ut ALLE data om EN gitt dag.
 *
 * @see    skrivDager(...)
 * @see    dagOK(...)
 * @see    finnDag(...)
 * @see    Dag::skrivAktiviteter()
 */
void skrivEnDag() {
    int dag, maaned, aar;
    Dag* dagPtr;
    skrivDager(false);           //skriv ut alle dager uten aktiviteter
                                   //leses inn en lovlig dato dagOK()

    dag = lesInt("skriv inn dag ", 0, 31);
    maaned = lesInt("Skriv inn maaned ", 0, 12);
    aar = lesInt("Skriv inn aar ", 1990, 2030);
    if (dagOK(dag, maaned, aar)) { //vil alltid gi true pga lesint
        dagPtr = finnDag(dag, maaned, aar);
        if (dagPtr != nullptr) { //Skriv ut info for dagen
            dagPtr->skrivDato();
            dagPtr->skrivAktiviteter();
        }
        else {
            cout << "Dagen finnes ikke! \n";
        }
    }
}

/**
 * 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";  
}
```