

C++ Programming Project

An Appointment Book

Due date: Sunday 18, 2001, at midnight

Project Description

You are asked to implement an appointment book that is able to store different appointments for a single calendar year. For simplicity, we assume that all 12 months have exactly 30 days, and that each day has 48 possible half-hour appointment slots.

As part of the assignment, you are asked to design and implement a class hierarchy that allows code reuse and enforces data encapsulation. A good design maximizes code reuse, and minimizes exposure of “internal” data structures and methods to the “outside world” (public vs. private and protected), or subclasses (protected vs. private). You are NOT allowed to use the `friends` construct to break data and/or method encapsulation.

As part of this assignment, you will receive skeletons of class definitions and implementations. All provided class members are public. You will have to define a class hierarchy between the classes, and possibly introduce new classes in order to allow code reuse. You have to decide which methods should be virtual. You may introduce additional methods, and you MUST implement the methods provided in the skeleton class definitions. You MUST NOT change the signatures of the skeleton class definitions.

As in the previous projects, we will test the functional correctness of your code through automatic testing tools. This requires you to carefully follow the output format of any `Print` method. Implementations of some `Print` methods are already provided in the handed out project code. **Please make sure that you follow the print format conventions as defined by the output of our sample main program `SampleMain.C`, which is listed in `SampleMain.output`.** You will lose points if you do not follow these conventions.

The following aspects of object-oriented programming and C++ are covered in this project:

- a class hierarchy that promotes code reuse
- virtual functions
- data and method encapsulation (public vs. protected vs. private)
- use of iterators

A good object-oriented design tries to keep things as simple as possible, while providing the highest degree of reuse and protection. This project has multiple correct answers, and you will encounter design tradeoffs. Look at our web site and newsgroup at least every other day, and **START EARLY!**

Getting Started

Copy the following files into your own project subdirectory: `DateTime.h`, `DateTime.C`, `Appointments.h`, `Appointments.C`, `Iterators.h`, `Iterators.C`, `SampleMain.C`, `SampleMain.output` and `Makefile`. The `.h` files contain class and function specifications that need to be implemented. We provide solutions for some classes in the corresponding `.C` files, and you are asked to extend the existing class definitions and their implementations in order to produce a working overall solution. `SampleMain.C` contains a sample appointment book with a few appointment entries. `SampleMain.output` shows the expected answer for a *functionally* correct solution of this project for the input used in `SampleMain.C`. You should use your own `Main.C` in order to test your implementation for different inputs of your choice. Your project will not only be graded with respect to its *functional* correctness, but also its overall *object-oriented* design.

The `Makefile` specifies the `*.h` and `*.C` files that will need to be compiled in order to generate a working executable. Just say `make` and your code will be compiled, and an executable will be generated and written into file `proj3`. You do not need to use the `Makefile` for this project; it is there only for your convenience.

The next sections give you a description of the provided class definitions and partial class definitions with their corresponding implementations.

Class Interfaces

Class Date and Class TimeSlot

The specification and implementation of classes Date and TimeSlot is provided to you. You will need these classes for other class definitions and implementations in this project. **DO NOT modify these classes.**

```
class Date {
public:
    Date(int day, int month);
    int GetDay(); // return value between 1 and 30
    int GetMonth(); // return value between 1 and 12
    bool operator ++ (); // next date, returns true if valid next date, false otherwise
    bool operator < (Date d); // defines '<' operator on dates
    void Reset(); // resets date to January 1
    void Print();
protected:
private:
    int days; // overall 360 days
};

class TimeSlot {
public:
    TimeSlot(int minute, int hour); // minutes will be "rounded DOWN" to half-hour slots
    int GetMinute(); // returns either 0 or 30
    int GetHour(); // returns values between 0 and 23
    bool operator ++ (); // next time slot, returns true if valid next slot
    void Reset();
    void Print();
protected:
private:
    int slots; // in half-hour slots, i.e., 48 slots, range 0..47
};
```

Class Appointment, Class Personal, Class Research, Class Hiring, Class AppointmentBook

The following class definitions can be found in file Appointments.h. File Appointments.C will contain your code for implementing the classes. You will have to modify both files for this project.

```
class Appointment {
public:
    Appointment(int minute, int hour, int day, int month);
        // date and time slot for meeting
    string ClassName(); // returns name of class as a string
    void Print(); // prints the object instance
protected:
private:
};

class Personal {
public:
    Personal(int minute, int hour, int day, int month, string r, int cpn);
        // date and time slot for meeting
        // string r: remark
        // int cpn : contact phone number
    string ClassName(); // returns name of class as a string
    void Print(); // prints the object instance
protected:
private:
};

class Research {
```

```

public:
    Research(int minute, int hour, int day, int month,
             string m, int np, bool c, string pn, int pid);
        // date and time slot for meeting
        // string m: name of meeting room
        // int np: number of people at meeting
        // bool c: critical meeting?
        // string pn: project name
        // int pid: project identification number
    string ClassName(); // returns name of class as a string
    void Print(); // prints the object instance
protected:
private:
};

class Hiring {
public:
    Hiring(int minute, int hour, int day, int month,
           string m, int np, bool c, string can, string pos);
        // date and time slot for meeting
        // string m: name of meeting room
        // int np: number of people at meeting
        // string can: candidate's name
        // string pos: title of open position
    string ClassName(); // returns name of class as a string
    void Print(); // prints the object instance
protected:
private:
};

class AppointmentBook {
public:
    AppointmentBook();
    bool InsertAppointment(Appointment *appointment);
    Appointment * GetAppointment(Date *date, TimeSlot *slot);
    void Print();
protected:
private:
    Appointment * table[2][24][30][12]; // [slots][hours][days][months]
};

```

Note that we are requiring you to implement the `AppointmentBook` class using the array `table`. The method `InsertAppointment` returns `false` if there is already another appointment for same date and time slot. In this case, the new appointment will not be inserted into the appointment book.

Class `AppointmentIterator`

The `AppointmentBookIterator` class is given to you, both its specification and implementation. You are asked to use this iterator to implement three functions. The signatures of the functions are provided in `Iterators.h`. The `AppointmentBookIterator` iterates over all `Appointments`. The `++` operator moves the iterator to the next `Appointment`. If no such appointment exists, the operator returns `false`, otherwise it returns `true`. The method `Current()` returns a pointer to the current `Appointment` of the iterator. File `Iterators.C` contains the implementation of the `AppointmentBookIterator`, and also will contain your code for implementing the three functions whose specification is given below.

```

class AppointmentBookIterator {
public:
    AppointmentBookIterator(AppointmentBook *aB);
    bool operator ++ (); // next Appointment, false if no more appointments
    Appointment * Current();
    void Reset();
protected:
private:
    AppointmentBook *aBook;
};

```

```

    Date *currentDate;
    TimeSlot *currentTimeSlot;
};

// Lists all non-empty appointments
void ListAllAppointments(AppointmentBook *aBook);

// Lists all appointments of a given type (no subtyping)
void ListAppointmentsOfType(AppointmentBook *aBook, string type);

// Lists all appointments between "startDate" and "endDate",
// excluding the "startDate" and "endDate".
void ListAppointmentsInRange(AppointmentBook *aBook,
                             Date startDate, Date endDate);

```

SampleMain.C

A sample `main` function is given in file `SampleMain.C`. You need to look at the sample `main` to understand how the appointment book is used, and what the output should look like. The output for the sample `main` is listed in file `SampleMain.output`. You need to follow EXACTLY the shown output format. You should use your own `main` function to develop your project code.

Makefile and Debugger

A simple UNIX Makefile is provided as part of this project. This is done for your convenience. Just say `make proj3` in order to compile your program and generate the executable named `proj3`. Note that you will have to modify the provided Makefile in order to use your own `main` function. As is, the provided project code will NOT compile. You can use `gdb` to debug your program. If you are not yet familiar with `gdb`, look at the on-line manual pages which can be accessed by saying `man gdb`.

Web Site and Newsgroup

You need look at our web site and check our newsgroup at least every other day. We will post clarifications regarding the project, if needed.

Submission

All code has to reside in the files that we provided. You will need to submit `Appointments.h`, `Appointments.C`, `Iterators.h`, and `Iterators.C`. No other file(s) should be submitted. The exact submission procedures will be posted on our CS314 web site.