CSCI 230 — Data Structures and Algorithms
Project 1 — Meows and Mutts

January 25, 1999

Preliminary Submission Deadline: Tues, Feb 2 at 11:59:59 pm
Final Submission Deadline: Tues, Feb 9 at 11:59:59 pm

The Meows and Mutts (M&M) animal clinic and kennel has decided to automate the process of scheduling animals for boarding. They want a new program and they are hiring you to write it for them. Since you have negotiated your contract with M&M so that you retain the rights to the software and since you would like sell it to other kennels, you need to make your program flexible.

M&M, or any other kennel, will provide you with a list of cages, numbered sequentially starting from 0. Each cage has a weight range of the animals that can fit in it. Once the list of cages is known (assume it will not change), you are ready to start scheduling animals. Only accept reservations for the current calendar year (1-1-99 to 12-31-99).

The scheduling must handle several operations.

S chedule an animal starting at a given date. If there are no cages available, add the animal to the waiting list. If the animal is too big or too small to fit into any of the cages, output an appropriate message and add the animal to neither the waiting list nor the schedule. If the animal is scheduled, output a confirmation message.

C ancel an animal from the schedule or from the waiting list. If the cancellation is from the schedule, go down the queue of waiting animals, and determine if one can fill in the space in the cage. Try animals in order from the waiting list to fill the time slot. This may result in more than one animal being scheduled (e.g. if Fido had a two week time slot, then Alphie might be able to use the first week, and Buffy the second). Output messages indicating the changes to the schedule. Output an appropriate message if the animal is unknown.

R evise a reservation if possible. If the revision (change) is possible, make the change and output a message. (Hint: within your program, changing a reservation will involve a temporary cancellation and then a check to see if space is available.) If a change cannot be made, do not add the animal to the waiting list. If a change is made by shortening the animal's scheduled time, check to see if any animals from the waiting list can now be scheduled. Output messages appropriate to the operations performed. (This operation is the most complicated; leave it for last!)

A nimal status: check the status of an animal. Is it scheduled or on the waiting list? When is it scheduled?

P rint the schedule of animals for a given day.

Design and Implementation

There are many ways this program could be implemented, many different data structures and algorithms that could be used, and many ways the project description could be extended and made more realistic. It will be interesting to think about these issues during the project and as we explore more sophisticated data structures. For now, however, try to keep it as simple as possible. Use the sequential container classes (lists, vectors, deques, stacks and queues) from the standard (STL) library where needed, and use the STL algorithms whenever you can.

Pay particular attention to the design of your classes. Here is an outline of classes you might have:

- A class that represents the date. It should have a constructor that builds a date from a month and day, less-than and greater-than operators, and an addition operator that adds a number of days. It should also have a function that converts from the month and day to the Julian day—the numbered day of the year (1 to 365). We have provided this class for you in

  Project 1 Auxiliary Files Directory[1]

  This directory is also accessible directly on the RCS file system as

  ```
  /dept/cs/cs230/public/projects/project1
  ```

- A class that represents a cage and keeps a schedule for that cage.
- A class that maintains the entire schedule and contains the cages. It should also maintain lists of scheduled and waiting animals.
- A class that stores and provides information about each animal, including its name, weight, and if/when it is scheduled.

The member functions defined for each of these class objects should support the above outlined scheduling operations. They should do so in such a way that the implementation details (the lists, queues, etc.) are hidden from the outside world. Start your programming task by determining which classes do what operations and then decide on the implementations within each class (some will be simple, others not).

Program synopsis and input format

The program will be run with the command-line

```
animals cages.txt
```

(You can set program arguments in Developer Studio using the Project->Settings menu, under the Debug tab.) After reading the cages from `cages.txt` (or whatever name is provided), the program should interactively query the user for input. All output should be to `std::cout` or `std::cerr`.

The format of `cages.txt` is simple. Each line will give the the minimum and maximum weights (integers) for that cage. The cage numbers will be implicit in the ordering of the input file. The number of cages will be determined by the number of input lines. There will be no mistakes in the file.

The interactive queries will have the following formats. First the user should type one of the characters ‘S’, ‘C’, ‘R’, ‘A’, ‘P’, or ‘Q’ (to quit). Small letters should be allowed as well so the program isn’t case sensitive.

- For ’S’, the program should request the animal and then request the starting date and duration. The animal will be a name and a weight, such as

  ```
  Fido 11
  ```

  with the name ending with a whitespace and the weight being an integer. The starting date and duration will be described by 3 integers: a month, a day, and a number of days.

- For ‘C’, a cancellation, the program should just request the animal name (which must match exactly the name in the schedule).
- For ‘R’, a revision, the program should request the animal name and the new starting date and duration.
- For ‘P’, the program should request the starting date (month and day) and the number of days to print the schedule.

[1] http://www.rpi.edu/dept/cs/cs230/public/projects/project1/

Sample input and output are provided in

Project 1 Auxiliary Files Directory[2]

This directory is also accessible directly on the RCS file system as

```
/dept/cs/cs230/public/projects/project1
```

Notes and assumptions

- Each animal may be in the schedule or on the waiting list only once. Therefore, your program should check to see if an input animal is already scheduled or waiting; if so, do not permit a new scheduling operation. If the animal is canceled it may be rescheduled, however.
- An animal occupies a cage all day for each day it is scheduled. Therefore, if it is scheduled for 7 days starting on 8/3 it will occupy the cage for 8/3, 8/4, 8/5, 8/6, 8/7, 8/8, 8/9. (Note that it does not occupy the cage for 8/10!)
- The month and day values that your program reads will be correct and will not go past the end of the year, even at the end of an animal's scheduled stay.
- Be sure to use the `string` class from the standard library instead of `char*` for the animal names. It will simplify life considerably. If you haven't used `string` before, look in a recent reference book on C++.

Implementation environment

You may use either the Visual Studio environment or the GNU (g++ compiler) environment to implement your project. Please be very specific in indicating which one we should use in grading! Regardless of which you submit you **must submit a makefile**. You will need to write your own if you are working with the GNU compiler. Visual Studio will create one for you but you must explicity tell it to make it available (under the Project menu, select Export Makefile).

Submission dates and guidelines

A preliminary submission is required (see the beginning of the project description for the Preliminary Submission Deadline): each student must submit a brief outline of the design of the classes for the project. Submission should be via email to the TA for your section. This outline should indicate the class declarations, member function declaration (but not implementation) and member variables. It must also include a drawing (done with a text editor) of your classes and their interrelationships. Each submission will be briefly critiqued and returned (electronically). Students not submitting an outline or submitting an obviously thoughtless one by the indicated date will have a 10 point penalty taken from the project grade. During final implementation of the project, students should feel free to revise the design (in part based on the feedback).

The Final Submission Deadline is given at the beginning of the project description. Projects submitted within 24 hours after this deadline will have 10 points taken from the grade (maximum of 100). Projects submitted within 24-48 hours after this deadline will have 20 points taken from the grade. Projects submitted later than this will *not* be accepted for credit.

To submit the project, create a subdirectory named `cs230` of your home directory on the RCS file system, and within that create a subdirectory named `project1`. Within that subdirectory place all of your source files (.cpp and .h) in a directory, along a file named `README`, and your makefile. The `README` file should include any notes about your project that we should know, as well as which compiler you used to build the project. You do not need your workspace and project files if you developed on Visual C++.

You must give us access to the directory, so that we can read the files. To do so, execute the following commands in your project subdirectory.

[2]http://www.rpi.edu/dept/cs/cs230/public/projects/project1/

```
fs sa . conwam rl
fs sa . karans rl
fs sa . rajagp rl
fs sa . mussed rl
fs sa . valoij rl
```

Or just execute `/dept/cs/cs230/public/projects/setpermissions`. (Note that these are *not* the usual Unix commands for setting access permissions; they are AFS permission commands. Unix commands will not work! For more information see Brian Osman's AFS Permissions and Groups page.[3]) This allows us (the TAs and instructors) to look at, but not change your files. Once you have done this, send an email message to the TA for your lab section, with the subject line 'DSAPROJ1' saying that your project files are ready. Use the email address from the following table:

Section	Instructor	Lab	Room	TA	Email Address
1	Valois	F 8am - 10am	Sage 3101	Priya	rajagp@rpi.edu
2	Valois	F 10am - noon	Sage 3101	Conway	conwam@rpi.edu
3	Valois	F noon - 2pm	Sage 3101	Priya	ragagp@rpi.edu
4	Musser	F 2pm- 4pm	Sage 3101	Shri	karans@rpi.edu
5	Musser	R 6pm - 8pm	Sage 3101	Conway	conwam@rpi.edu
6	Musser	F noon - 2pm	VCC North	Shri	karans@rpi.edu

You may change the files up to the deadline; we will not copy your files for grading until after the deadline. You should not change your files after the deadline at least until they have been graded, because if there is any problem in accessing them we might accept the timestamp on the files as evidence of the time they were actually ready. (This is at the discretion of the instructors and TAs, who incidently have means of detecting any tampering with the timestamps.)

If you submit later than the deadline, then you should send an email message to the TA for your lab section stating that your files are ready to be copied. As stated above, if we receive a submission within 24 hours after the deadline, 10 points will be taken from the grade. Projects submitted within 24-48 hours after the deadline will have 20 points taken from the grade. Projects submitted later than this will *not* be accepted for credit.

At no point should you make your files more widely accessible than to the TAs and instructors.

Be sure to check the course web site for any updates to the submission guidelines.

Grading criteria

The program grade will be grade 25% for successful compilation, 40% for correctness of the program, and 35% for program structure. Criteria for program correctness may be discerned from the foregoing description and will solely be based on the output from your program. Program structure includes class organization, member function design, code readability (indentation, variable names, etc.), and commentary. There should be comments describing the purpose of each class and the member functions and member variables. The comments about each class **must** also include a discussion of the flexibility of the class design and how well it will support future changes. Keep other comments, especially within the body of each function, short and concise. Avoid comments that just state in English what is obvious from the code. One bad example is

```
   i++;   //  increment i
```

Academic integrity

Students may discuss the program design, especially the design of the classes. Implementation of class member functions and the main program must be done individually. Sharing of code is expressly forbidden and will be detected by code comparison tools. Projects from all six sections will be graded together by the TAs.

[3]http://www.rpi.edu/~osmanb/afs/index.html