

CMSC 212
Project #6
Due 5/10/2007 8:00 PM

Background

In this assignment you will build a set of functions to manipulate graphs. The graphs in this assignment will be for handling airline schedules. We will provide an API for you to implement as well as the data structures required to store the graph. You must use our data structures, and you may not change either of the supplied header files `schedule.h` and `scheduleP.h`.

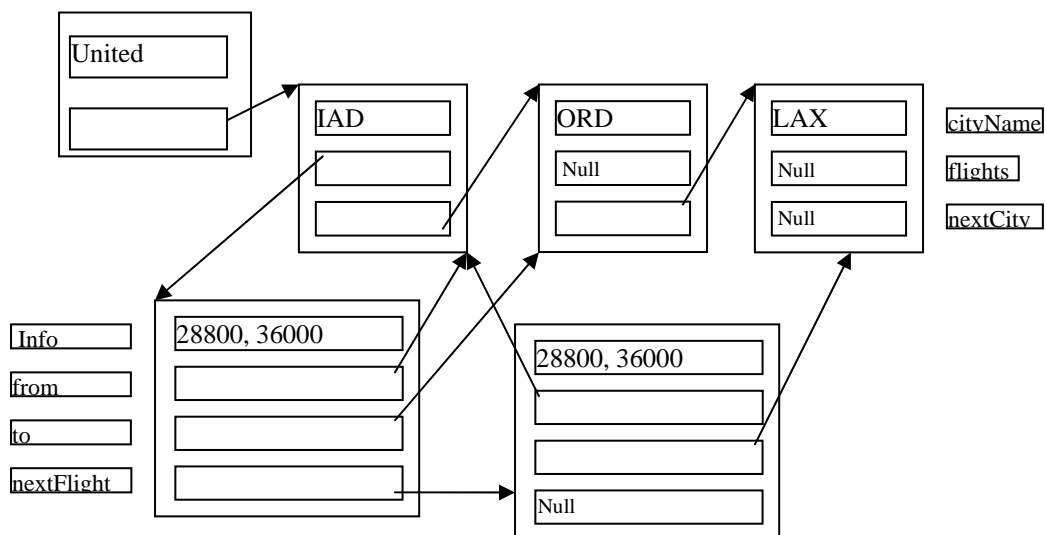
The three main data structures are `schedule`, `city` and `flight`.

`Schedule` is an abstraction of a single airlines schedule. It contains the name of the airline (`airlineName`) and a linked list of cities the airline serves (`cities`).

`City` is the information about a single city served by a single airline (if two airlines serve the same city there will be two instances of this data structure). The fields of `city` are the name of the city (`cityName`), a linked list of flights leaving that city (`flights`), a next pointer for the next city served by the airline (`nextCity`) and a variable `visited` that will be used in computing routes through the schedule.

`Flight` is a structure that represents the information about a single flight on one airline from one city to another. Its fields are: flight arrival and departure times (`info`), the starting city for the flight (`from`), the destination city for the flight (`to`), and the next flight out of the `from` city on that airline (`nextFlight`).

A picture of the data structure associated with `test1` would look like:



API

In general in this API, unless otherwise noted, any failure of a memory allocation call or passing of a NULL pointer should result in an error return value. The error return value is typically NULL for functions that return a pointer and a negative number those that return an integer.

```
schedule *createSchedule(char *airline)
```

Create a new empty schedule for the passed airline. Return's NULL on failure, or NULL city being passed.

```
void destroySchedule(schedule *g)
```

Destroy the passed schedule. It should delete all existing cities and flights for this schedule.

```
city *addCity(schedule *s, char *name)
```

Add the passed city to the passed schedule. Return a pointer to the newly created city. If the city with the passed name already exists, it should return a pointer to that city. It should return NULL if the parameters are invalid.

```
int dropCity(schedule *s, city *c)
```

Drop service for the passed city. This should drop all flights to and from this city for the passed airline. It should return 0 if the drop was succeed, and a negative value if an error occurs.

```
city *findCity(schedule *s, char *name)
```

Find the passed city. It should return NULL if the city can't be found, or if the parameters are invalid.

```
char *getCityName(city *s)
```

Return the name of the passed city. Returns NULL for any error condition.

```
flight *addFlight(schedule *s, city *from, city *to, flightInfo info)
```

Add a new flight from city `from` to city `to` for the passed airline `s`. The flight should leave at `info.departTime` and arrives at `info.arrivialTime`. This should return NULL if there are errors with any of the parameters.

```
int dropFlight(schedule *s, flight *f)
```

Drop the passed from the passed schedule. Returns 0 if the drop succeeded and -1 if there is any error (including the flight not being part of schedule `s`).

```
char *getFromCityName(flight *f)
```

Return the name of the city that the passed flight originates in. Returns NULL for any error condition.

```
char *getToCityName(flight *f)
```

Return the name of the city that the passed flight travels to. Returns NULL for any error condition.

```
int getFlightInfo(flight *f, flightInfo *info)
```

Return the flight info for the passed flight. Return 0 if there is no error and -1 if there are any errors.

```
int getCitySchedule(schedule *s, char *name, flight* found[],  
int max)
```

Return the flights that leave the passed city name. Pointers (not copies of the flights) to the flights from that city should be put into found. At most max flights should be returned. The return value should be the number of flights copied into the array found. If there are any errors, the return value should be negative.

```
int printRoutes(schedule *s, city *from, city *to, int maxHops)
```

Print all practical direct and multi-hop flights from city from to city to that take less than or equal maxHops flight segments that travel only on the passed airline. For a connecting flight to be practical, it must depart at least one second after the arrival time of the inbound flight. There will be no red-eye flights (i.e. all flights will start and finish on the same day and not cross midnight). You should not include flights that visit the same city twice. The format of the output should be one line per flight and be of the form:

```
IAD -(28800,36000)> ORD -(37000,46960)> LAX
```

Where IAD is the from city, ORD, and intermediary city, and the destination is LAX. The first flight leaves at time 28800 and arrives at time 36000. The second flight segment leaves at time 37000 and arrives at 46960. The punctuation and spacing must be exactly as shown. The function should return the number of valid flights (routes from from to to), or a negative value if there are any errors.

```
int printInterLineRoutes(schedule *s[], int numAirlines, char  
*from, char *to, int maxHops)
```

Print all practical (that don't require time travel) direct and multi-hop flights from city from to city to that take less than or equal maxHops flight segments and that travel on any of the passed airlines s (of which there are numAirlines). The output format should be:

```
IAD -(United,0,200)-> LAX -(Southwest,288,360)-> SJC
```

The format is the same as for `printRoutes` except that the name of the airline used is printed before the departure time. You do not need to worry about handling the case of routing a passenger back through the same city twice on two different airlines (we will not test this case).

FILES

The interface for using this API is supplied in the file `schedule.h`. The internal data structures for the project are defined in `scheduleP.h`. We will supply both of these files (in the file `p6.tar.gz`). All of your implementation needs to be in the file `schedule.c` (we provide a blank `schedule.c` in `p6.tar.gz`).