ECE 209 Program 3: `tvguide`
Summer 2018

Due Friday, July 27, 2018 @ 11:45pm

This programming assignment must be completed **individually**. Do not share your code with or receive code from any other student. Evidence of copying or other unauthorized collaboration will be investigated as a potential academic integrity violation. **The minimum penalty for cheating on a programming assignment is a grade of -100 on the assignment.** If you are tempted to copy because you're running late, or don't know what you're doing, you will be better off missing the assignment and taking a zero. Providing your code to someone is cheating, just as much as copying someone else's work.

DO NOT copy code from the Internet, or use programs found online or in textbooks as a "starting point" for your code. Your job is to design and write this program from scratch, on your own. Evidence of using external code from any source will be investigated as a potential academic integrity violation.

In this assignment, we will create a simplified TV guide system. We will build up a database of channels, and shows that are offered by each channel. And we will be able to find shows based on their channel, name, day of the week, and time. (We will not, however, be able to delete a show or channel once it's in the system.)

As with earlier programs, we will provide the user interface code. Your job is to implement and maintain the collection of channels and shows, and to search for shows using various criteria. Because the number of channels and shows are not known ahead of time, and because the collection is built up incrementally (one channel/show at a time), you will need to use the linked list data structure.

You will also be implementing two abstract data types, in which the details of the data representation of the type is not visible to the user of the type. The user can only a pre-defined set of functions (known as the "interface") to access or change the data type values. (We will discuss this at length in class.)

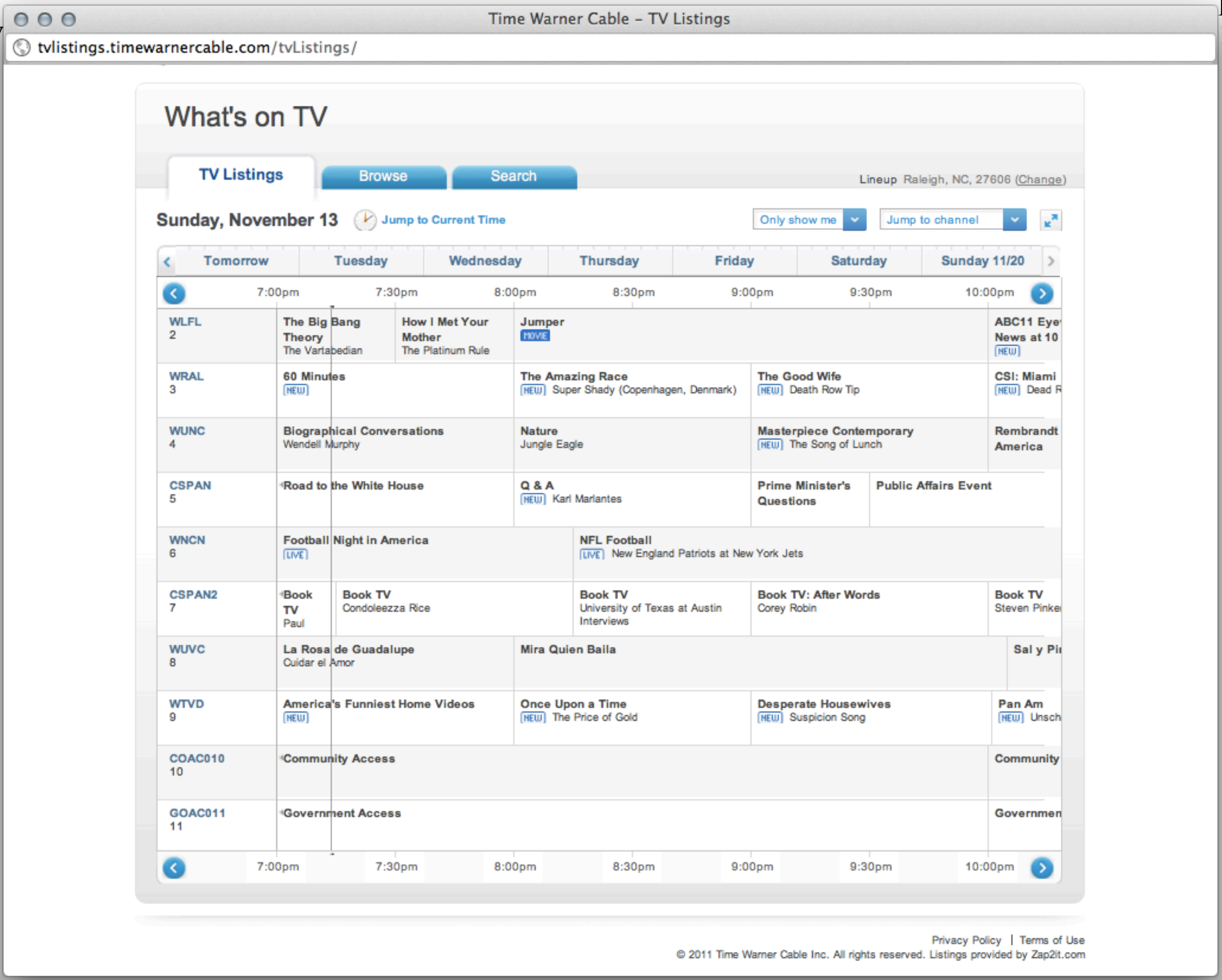
To implement this program, you will need to know how to do the following:

- Add items to a linked list.
- Search for items in a linked list.

Background

For this program, we want to implement a TV guide application. We'll have a collection of *channels*, and each channel will have a collection of *shows*. Conceptually, we'll be managing information like the guide provided by tvguide.com and most cable providers, as shown in Figure 1.

Each row specifies a channel, using a number and a name. Then the shows for that channel are listed, including name, start time, and end time. For our application, a channel's shows may be partially



Sunday, November 13

	7:00pm	7:30pm	8:00pm	8:30pm	9:00pm	9:30pm	10:00pm
WLFL 2	The Big Bang Theory The Vartabedian	How I Met Your Mother The Platinum Rule	Jumper MOVIE				ABC11 Eye News at 10 NEW
WRAL 3	60 Minutes NEW		The Amazing Race NEW Super Shady (Copenhagen, Denmark)		The Good Wife NEW Death Row Tip		CSI: Miami NEW Dead R
WUNC 4	Biographical Conversations Wendell Murphy		Nature Jungle Eagle		Masterpiece Contemporary NEW The Song of Lunch		Rembrandt America
CSPAN 5	Road to the White House		Q & A NEW Karl Marlantes		Prime Minister's Questions	Public Affairs Event	
WNCN 6	Football Night in America LIVE		NFL Football LIVE New England Patriots at New York Jets				
CSPAN2 7	Book TV Paul; Book TV Condoleezza Rice		Book TV University of Texas at Austin Interviews		Book TV: After Words Corey Robin		Book TV Steven Pinke
WUVC 8	La Rosa de Guadalupe Cuidar el Amor		Mira Quien Baila				Sal y Pi
WTVD 9	America's Funniest Home Videos NEW		Once Upon a Time NEW The Price of Gold		Desperate Housewives NEW Suspicion Song		Pan Am NEW Unsch
COAC010 10	Community Access						Community
GOAC011 11	Government Access						Governmen

Privacy Policy | Terms of Use
© 2011 Time Warner Cable Inc. All rights reserved. Listings provided by Zap2it.com

Figure 1. TV listing information: channels and shows.

specified -- that is, there may be time slots for which no show is scheduled. We will manage one week's worth of scheduling information: from *Monday* (first day) to *Sunday* (last day).

The scheduling information is built incrementally. The user adds one or more channels, and then adds shows scheduled for that channel. The user can also look for shows by channel, name, day, time, or any combination of those attributes.

Program Specification

User Interface (main.c)

A file named **main.c** will be provided. It contains the **main** function, which implements the user interface for the program.

The user interface allows the user to enter commands, and prompts the user for information required to carry out the command. There are four commands, each a one-letter string, defined below:

- **c** -- Add a channel. User specifies channel number and name.

  Constraints: Channel cannot be created if there already exists a channel with the same number. (Name doesn't matter.)

- **s** -- Add a show. User specifies channel number, name, day, start time, and end time.

  Constraints: Show cannot be created if there is already a scheduled show anytime during the specified time period (on the same channel). Cannot be created if specified channel doesn't exist, or if the day string is illegal (see below). Cannot be created if the end time is before the start time. (We do not allow a show to extend into another day -- must end at 23:59 or earlier.)

- **a** -- All channels. Prints a list of all channels that have been added to the schedule.
- **f** -- Find shows. User specifies channel, name, day, and time. A wildcard (*) can be entered for any or all of these search parameters. (The time parameter is not a start time; it is any time during the span of a show. For example a time of 9:30 will match a show that starts at 9:00 and ends at 10:00.) Only a full, case-sensitive match of the name is supported -- no searching for strings within the name.
- x -- "Execute" a command file. The user specifies the name of a file, which must be in the same directory as the executing program. The file contains a sequence of commands, formatted as described above. This is a "shortcut" for setting up a schedule; it allows a sequence of commands to be processed quickly without having the user type them in. Once the commands have all been executed, the program will prompt for another command. (More details on the command file format are given in a later section of this document.)
- **q** -- Quit.

Times are specified in "military" 24-hour format, from 00:00 (12am midnight) to 23:59 (11:59pm).

All strings (channel names and show names) are limited to 20 characters. Day strings are the first three characters of each day, i.e., MON, TUE, WED, etc. They do not have to be entered as all caps (but the user interface will convert them to all caps for consistency).

The "f" command (find shows) may result in zero or more shows being found. The matching shows are printed in a specific order: by channel number (smallest to largest), and (within a channel) from earliest to latest start times. Remember: Monday is the first day in the schedule, and Sunday is the last, so a Monday show has an earlier start time than a Sunday show.

The user interface code will print the shows in the order provided by your function. You are responsible for returning a correctly-ordered list of shows.

Data Types (sched.h)

A file named **sched.h** will be provided. It contains struct definitions and typedefs for some data types that will be needed for this program.

Two of these types (`Channel` and `Show`) are not fully specified. They are only defined as pointers to other types (`struct channel` and `struct show`, respectively), which you will define in your implementation file (**sched.c**).

You are responsible for defining `struct channel` and `struct show` in **sched.c**. You are free to implement these structs however you choose, as long as you can provide the information required by the interface functions described below.

These are the two main types in the program. Since the user of the type does not have information about the details of the struct definition, it must use functions to create instances of the struct, or to read change information stored in the instance. This is known as an abstract data type -- the details are abstracted away from the user.

The Channel type is defined by the following functions:

- **`addChannel(int num, const char* name)`**
  Creates a new Channel and adds it to the system. Returns a pointer to the newly-created object. (NOTE: The name string must be copied into the channel instance. You cannot simply save the pointer that's passed in by the caller -- that string belongs to the caller and may be changed later.) If a channel cannot be added because a channel with number already exists, the function returns NULL (and no channel is added to the schedule). The name string is 20 characters or less.
- **`channelNum(Channel c)`**
  Returns the number (integer) associated with c.
- **`channelName(Channel c)`**
  Returns the name (string) associated with c.

The Show type is defined by the following functions:

- **`addShow(Channel c, const char* name, const char* day,`**
  `        struct showTime startTime, struct showTime endTime)`
  Creates a new Show and adds it to the schedule. Returns a pointer to the newly-created object. (NOTE: The name string must be copied into the show instance. You cannot simply save the pointer that's passed in by the caller -- that string belongs to the caller and may be changed later. The day pointer may be copied, since it points to a constant literal string.) If the show cannot be created, because the channel doesn't exist, or because a show is already scheduled for that channel during the designated time period, the function returns NULL (and nothing is added to the schedule). The name string is 20 characters or less. You may assume that the day and times are legal -- the user interface will prevent a bogus day or an illegal time span.
- **`showChannel(Show s)`**
  Returns Channel on which s appears.
- **`showName(Show s)`**
  Returns name (string) of s.
- **`showDay(Show s)`**
  Returns day (string) when s is scheduled.

- **`showStart(Show s)`**
  Returns starting time (struct showTime) of s.
- **`showEnd(Show s)`**
  Returns ending time (struct showTime) of s.

(Many abstract types have functions that allow the user to change information about an instance, but we won't need that for this program.)

In addition, there are some fully-defined types. Times are represented using struct showTime, which includes an hour and a min field. The hour field holds an integer between 0 and 23. The min field holds an integer between 0 and 59.

Finally, there are types used to build linked lists of Channels and Shows.

```
struct channelNode {
    Channel c;  /* points to a struct channel */
    struct channelNode *next;
};

typedef struct channelNode * ChannelList;

struct showNode {
    Show s;  /* points to a struct show */
    struct showNode *next;
};

typedef struct showNode * ShowList;
```

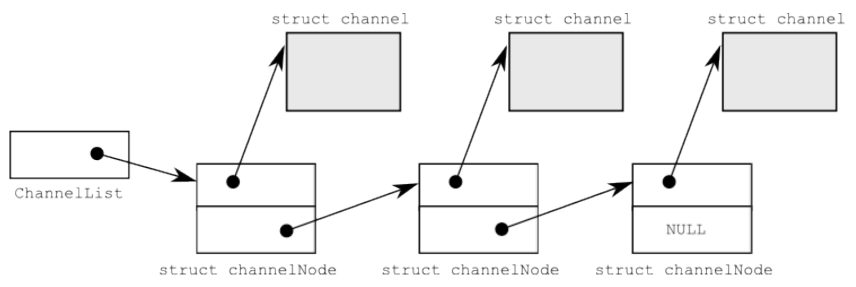


Figure 2. A list of channels.

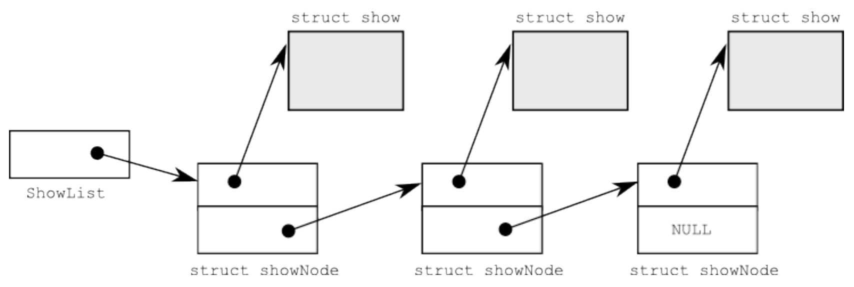


Figure 3. A list of shows.

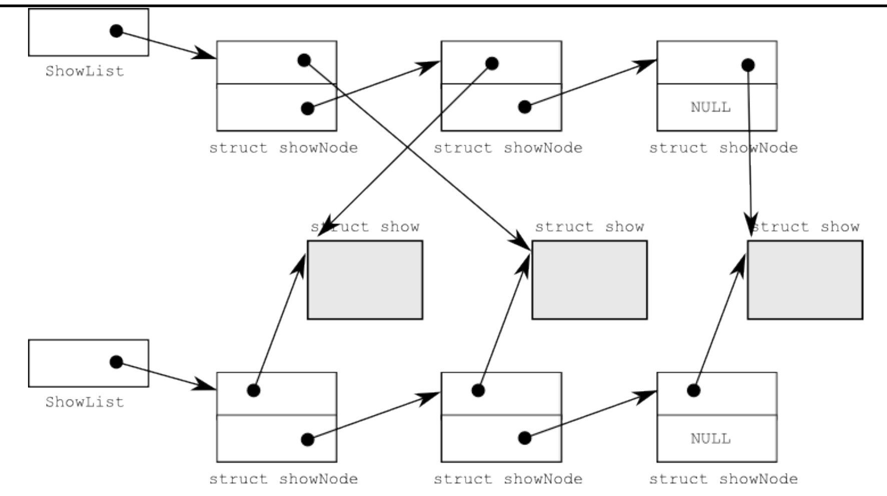


Figure 4. Two lists, containing the same shows in a different order.

Note that the data portion of each linked list node is a pointer. The data is not part of the node; it is referenced (pointed to) by the node. This means that a specific instance of a type (e.g., a struct show) can be present in more than one list at a time.

Figures 2 and 3 illustrate what a linked list of channels and a linked list of shows look like. Figure 4 illustrates what was mentioned in the previous paragraph: multiple lists that refer to the same shows. Why would this happen? For example, when you create a list of shows to return from the **findShows** function (see below), you can do that without copying the struct show objects and without disturbing the original list of shows on each channel.

The **sched.h** file also defines several macros that you may use to refer to constant strings to represent days of the week. A day string is a three-character string containing the first three letters of the day: MON, TUE, WED, etc. (The user interface functions will point to these literal strings when a day is required. Your structs can copy those pointers, if you want, rather than allocating your own character array to hold the day for a show.)

Query Functions (sched.h)

In addition to the data types and functions described above, **sched.h** also declares three functions that allow us to query the schedule database. In two cases, the function returns a list. In all cases, if nothing is found that matches the query, a null pointer (NULL) must be returned.

- **`allChannels()`**
  Returns a list of all channels in the schedule, in increasing order by number. A new list must be created; the nodes (but not the channels) will be deleted by the caller.
- **`findChannel(int num)`**
  Returns a pointer to the channel with the specified number. If no such channel, returns NULL.
- **`findShows(Channel c, const char *name, const char *day,`**
  `struct showTime *time)`
  Returns a list of all shows that match the channel, name, day, and time. A new list must be created;

the nodes (but not the shows) will be deleted by the caller. Any of the arguments may be NULL, which means that any show matches that search criteria. (If all values are NULL, all shows are returned.) The shows must be ordered by channel number (low to high) and then by day and starting time (earliest first).

Implementation (sched.c)

In the **sched.c** file, you are responsible for (a) defining `struct show` and `struct channel`, and (b) implementing all functions described in the previous two sections (as declared in **sched.h**).

To implement these functions, you will be required to create and maintain one or more data structures that track the channels and shows in the TV schedule. The functions **addChannel** and **addShow** have the side effect of adding information to that data.

Because of the incremental nature of the data, a linked list (or a collection of linked lists) is a good choice. You choose how to organize the data. Some possibilities include:

- A single, unordered list of shows. (You would need to put data in the proper order for the **allChannels** and **findShows** functions.)
- An ordered list of shows, ordered by channel and start time.
- A list of channels, where each channel contains a list of shows.
- A list of shows for each day.
- Separate lists of channels and shows.
- A list of channels, with multiple lists of shows, one per day.

As you can see, the possibilities are numerous.

It is necessary for these data structures to be stored as **global variables**. (At last, a good reason!) The data must be maintained beyond a single function's execution, and must be visible to several different functions.

Your global functions will only be used by functions in **sched.c**. If you want to ensure that they are not visible from other files, you can add the `static` qualifier to the declaration. (See CPP, Chapter 12.) This is not required, but it is a common practice. It avoids accidental name conflicts in large programs with many different programmers.

When a function returns a list, it is necessary to <u>*create a new linked list*</u>. The nodes of the list can (and should!) point to same channel and show objects that are stored in your lists. There is no need to copy a show or channel.

NOTE: Your channel and show objects <u>*must allocate storage*</u> for name strings. Don't just copy the pointer that is passed into addChannel or addShow -- copy the characters into the new object's character array. The caller will reuse its string storage; if you simply copy the pointer, then every show/channel will end up with the same name!

All name strings will be limited to 20 characters or less.

The strings for days of the week, on the other hand, are simply pointers to literal strings (as seen in the macros defined in **sched.h**.) These pointers can be used without copying the data, because no one can change the contents of a constant literal string.

Command File

A command file (mentioned in the User Interface section) is simply a list of commands and arguments that is read by the program, if a file name is specified on the command line. This is a quick way to set up a schedule every time you run the program. Once the commands in the file have been processed, control is turned over to the user (via stdin), unless there's a "q" (quit) command in the file.

One note about the command file format: A channel/show name must end with a linefeed character. Spaces are allowed in the name strings, so we use a linefeed to designate the end of the string. (See the **readline** function in **main.c**.) This is true even if you are specifying "*" as the show name in the "f" command. This means that commands that include a show name (**s** and **f**) must be split across multiple lines. All other command parameters may be on a single line, separate by at least one whitespace character.

The **x** command is not available for execution in a command file. It will be ignored.

The use of the command file is optional. It's only there for convenience. It has no impact on the part of the code that you are writing; it is only part of the user interface.

CLion users: To use a command file, it must be placed in the **cmake-build-debug** folder. This is where the executable is located.

Compiling

A **Makefile** will be provided for you. On a Linux system, put **Makefile** in the same directory with your other files (**main.c**, **sched.h**, and **sched.c**) and type "make". This will compile the C files (.c), as needed, to create object files (.o), and will then link the object files to create an executable file named **tvguide**.

If the Makefile is not working for you, here's how to compile:

```
gcc -c -g -Wall -pedantic main.c
gcc -c -g -Wall -pedantic sched.c
gcc -o tvguide -g main.o sched.o
```

If you are using CLion, then you don't need the Makefile. Create a new project, rename the **main.c** file to **sched.c**, and remove the **main** function. Copy the **main.c** and **sched.h** files to the same directory where your project files exist, then update the **add_executable** statement in the **CMakeLists.txt** file to include both **main.c** and **sched.c**. The system knows that these source code files are all part of the same project, and it will compile and link them together, as needed.

If you're using some other IDE, then you'll have to figure this out yourself. There should be a process to add an existing file to the project. If you can't figure it out, then just create new files yourself, and copy the content from the files provided.

Hints and Suggestions

- *Decide first how you want to represent your data.* There are many possible choices. Some choices are easy to create, but harder to search. Other choices make you do more work up front when a new item is added, but make finding shows/channels easier. Don't be afraid to change your data representation after you start coding. Don't choose a representation just because other people are using it -- make your own decision about how you want your data to be organized.
- You don't have to use linked lists for everything, but it's certainly reasonable to do so. Think about the differences between arrays and linked lists as you decide on your data representation.
- Use the linked list structures provided in **sched.h**. Don't roll your own. The user interface will also use these structures, so they must be used for the lists that are returned by **allChannels** and **findShows**.
- Don't overcomplicate the program.
- Work incrementally. Don't write the entire program before you try to compile or run it. It's better to get one part working, then add the next part, and so on. You are less likely to be overwhelmed with compiler errors, and your debugging efforts can focus on the small part of the program that you just implemented.
- Start early! This allows plenty of time to understand the assignment and to ask questions about the parts that aren't clear.
- For compiler errors, look at the source code statement mentioned in the error. Try to figure out what the error is telling you. Try to fix the first error first, and then recompile. Sometimes, fixing the first error will make all the other errors go away. (Because the compiler got confused after the first error.)
- Use a source-level debugger, like *kdbg*, to step through the program if the program behavior is not correct. If you are using NetBeans on your own computer, the debugger is integrated with the editor and compiler, so there's no excuse for not using it.
- For general questions or clarifications, use the Message Board, so that other students can see your question and the answer.
- For code-specific questions, email your code (as an attachment) to the support list: **ece209-001-sup@wolfware.ncsu.edu**. Attach your entire program, not just a snippet of code, so that we can compile and run your program the same way you do. Don't send to my individual email address; you're likely to get a faster response by sending to the list.

Administrative Info

Updates or clarifications on the Message Board:

Any corrections or clarifications to this program spec will be posted on the Message Board. It is important that you read these postings, so that your program will match the updated specification.

What to turn in:

- Source file – it must be named **sched.c**.

Grading criteria:

Partial credit is available in all areas, except where noted.

10 points:	File submitted with the proper name. (No partial credit! You will lose 10 points if you submit a file with an incorrect name!!!)
10 points:	Compiles with no warnings and no errors. (If the program is not complete, then only partial credit is available here. In other words, you won't get all 10 points for compiling a few trivial lines of code.)
10 points:	Proper coding style, comments, and headers. No <u>unnecessary</u> global variables. No goto. (See the Programs web page for more information.)
5 points:	Channel access functions (**channelNum**, **channelName**) are correct.
20 points:	Correctly creates and maintains a list of channels (**addChannel**, **allChannels**, **findChannel**). Newly-created list of channels, in the proper numerical order, is returned by **allChannels**.
15 points:	Show access functions (**showChannel**, **showName**, **showDay**, **showStart**, **showEnd**) are correct.
15 points:	**addShow** correctly creates and adds a show to the schedule.
15 points:	**findShows** works correctly for any set of inputs (search criteria), returning a newly-created list of matching shows in the proper order.