CS2303 In Class Exercises

Jan 21, 2020

January 9, 2020

Abstract

Today's goal is that you can analyze a statement of requirements (including those whose solution is a C executable) and get started creating a solution.

Name and WPI email: Kush Shah kshah2@wpi.edu

1 Requirements Statement

Homework assignments that call for code to be written are one form of requirements statement for an executable. We shall use homework as our examples of requirements statements.

1.1 Identify Compartments Using Nouns and Noun Phrases

Though C is not an object-oriented language, we can use C to obtain some of the benefits of object-orientation. Knowing that our C code is made up of data structure definitions and functions, we will apply a divide-and-conquer mindset; we will think about how to separate concerns by making compartments in our software. A compartment will be represented by a .c file and a .h file, of the same name, such as production.c and production.h. Using the nouns and noun phrases in the requirements statement, we will choose names for these compartments.

Example Homework Assignment:
Open a file whose name is given as the first command line argument (argv[1]), for reading. This is the input file.
Open a file whose name is given as the second command line argument for writing. This is the output file.
The input file is a binary file rather than a text file.
The output file is a text file.
Read the input file byte by byte.
For each byte, separate it into two four bit quantities.
For each four bit quantity, interpret it as a number in the range [0-15], and represent it as an ASCII character in the range {[0-9],[A-F]}.

Write the ASCII character in the output file.
Remember to close the files when the executable has finished reading the input file.

We analyze the text of the requirements statement to find nouns and noun phrases. We select among these nouns and noun phrases for ideas for the software compartments.

One compartment that makes sense for this software project is file input and output. Another compartment is byte conversion. Using these two compartments, we can begin to create the files into which our data structure definitions and other statements will reside.

All programming assignments in this course will have these files:

- The .c file containing the main function
- The .c and .h file called test
- the .c and .h file called production

Beyond those above, by analysis of the specific requirements we have inferred:

- A .c and .h file for fileHandling
- A .c and .h file for byteConversion

Functions in the fileHandling and byteConversion compartments are production code, as is the code found in the compartment production. These functions will be invoked by test code, found in the test compartment.

Exercises: (As always, if you have questions, ask. After this occasion, this material becomes background knowledge.)

1. Identify compartments in the following homework assignment:
   The program is to search a house, accumulating treasure from the several rooms in the house.
   A floor plan of the house is provided.
   The search order (from room to room) must be possible given the floor plan (no going through walls).
   Print the order of the rooms searched, and the successive subtotals of treasure accumulated as the search is carried out.

   Search need to be able to query house and room
   in order to find rooms and look for treasure.
   House needs to be able to query Layout in order to get # of rooms.
   Layout needs to be able to query Room in order to get room layout.

2. What are the .c and .h files you would use for the above homework assignment?

.h	.c
Search.h	Search.c
House.h	House.c
Layout.h	Layout.c
Room.h	Room.c

1.2 Identify Functions Using Verbs

We know that we construct code from data structure definitions and other statements. We know that C code contains functions. For any given problem we need to identify the functions we shall construct for its solution. We can find the verbs in the requirements statement and use these as a point of departure for identifying the functions we shall code.

The verbs (and objects of the verbs) in the problem of byte processing include:

1. read the command line values
2. open a file for reading, another for writing
3. read bytes from a binary file
4. analyze bytes
5. create character values
6. write characters to a text file
7. close files

We need to have some functions in our solution to this problem.
Some functions are already written, we can use them insofar as we know about them.
Other functions we will need to code.
There is a function "fopen". We will use it, that is, we will invoke it.
I've written starter code for you that processes command line arguments. You should read it for understanding, and consider how you might need to change it to serve this purpose.
Code for analyzing the bytes corresponds to one or more functions you need to code (sometime).
Code for creating character values corresponds to one or more functions you need to code (sometime).
Writing characters to a text file involves invoking a function, "fprintf".
Closing files can be done with the function "fclose".

So, we've derived a list of functions we need to create.

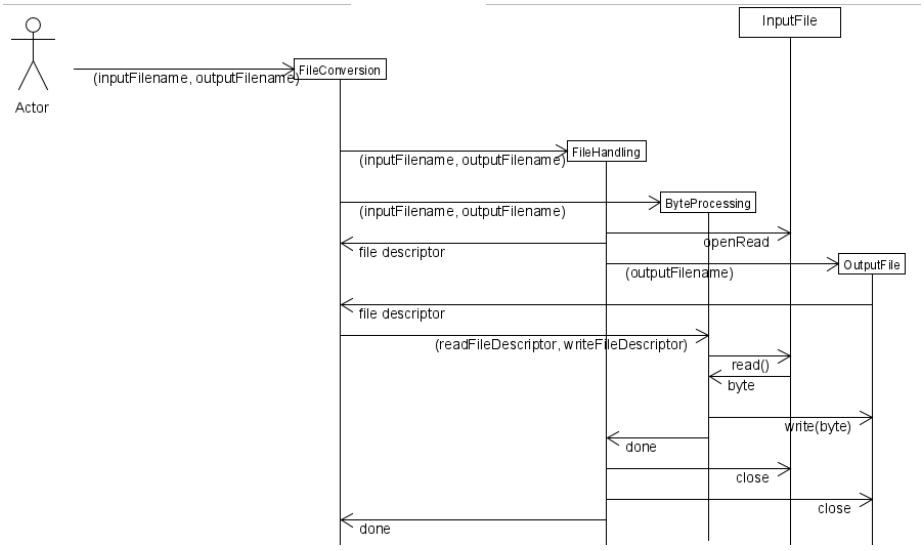

Figure 1: The sequence diagram shows schematically the interactions between the components of the solution.

- get input
- analyze bytes
- create character values
- produce output

Now, we need to associate the components with the functions. It makes sense to put the get input and produce output in the fileHandling component. It makes sense to put the analyze bytes and create character values in the byteConversion component.

2 Design a Solution at a High Level of Abstraction

The design of the application uses interaction between the components to generate a solution.

Now we can draw a sequence diagram, depicting how these functions interact.

Sequence diagrams show the components, in boxes. The lifetime of the component is shown by the arrow going down from the box. This line is called the lifeline. that get created during the execution of the program have arrows coming in to the left side of the box. Function calls on the components (invocations of functions defined within the components) are shown by arrows coming in to

the lifeline. Paramaters delivered in function invocations are mentioned on the arrow. Datatypes of returned values are shown on arrows going back in response to a function invocation.

Practice drawing sequence diagrams: (As always, if you have questions, ask. Developing sequence diagrams is a creative act. There can be many right answers. Like writing a new poem or painting a new painting, creators can feel blocked. Be compassionate with yourself, and draw something. After this occasion, this material becomes background knowledge.)

1. Draw a sequence diagram for the process of making a bed.
2. Draw a sequence diagram for the process of entering a new friend's contact information into your phone.