Developing Real-World Programming Assignments for CS1

Daniel E. Stevenson and Paul J. Wagner
Department of Computer Science
University of Wisconsin-Eau Claire
Eau Claire, WI 54701
{stevende, wagnerpj}@uwec.edu

ABSTRACT
Instructors have struggled to generate good programming assignments for the CS1 course. In attempting to deal with this issue ourselves, we have generated two real-world programming assignments that can be solved by most students yet generate challenges for advanced students. We present our overall criteria for a quality programming assignment in CS1, details of the two example assignments, and other issues stemming from the generation and management of these assignments.

Categories and Subject Descriptors
K.3.2 [**Computers & Education**]: Computer & Information Science Education – *Computer Science Education*.

General Terms
Algorithms, Design

Keywords
CS1, Programming Assignments

1 INTRODUCTION
We, and many of our fellow instructors, have found it difficult to construct quality programming assignments in CS1. The students are at an awkward point, where they have the basic tools of data types and perhaps control structures, but are not yet able to develop their own classes. They now need to learn to work with built-in objects, and master topics along the way such as strings, external files, and containers such as ArrayLists or arrays.

The problem we see is that many example programming assignments from textbooks and historical usage do not satisfy the dual goals of allowing the students to practice concepts and helping them develop skills that scale up for the larger work to follow in CS2 and subsequent courses. Many of these example assignments have one or more of the following characteristics:

- they are simplified, relating to one aspect of a larger problem (e.g. sorting an array of words, with no context of how the sorted array will be used)
- they focus on only one particular current programming topic (e.g. strings)
- they do not relate well to the real world of software development / computer science (e.g. they focus on abstract issues such as sorting an array of strings)
- they leave little room for creativity and innovation, which at least one recent article has described as important [1]

We have worked to develop assignments for CS1 that attempt to address the above shortcomings, and satisfy our criteria for quality programming assignments in a CS1 course. Section 2 presents our criteria for a quality, real-world programming assignment in a CS1 course. Sections 3 and 4 contain two examples of assignments that we use in CS1 that we have developed based on these criteria. Section 5 discusses advanced string parsing issues which we use to build on the basic assignments. Section 6 contains several additional issues that we have found important in constructing and managing these assignments.

2 CRITERIA FOR GOOD PROGRAMMING ASSIGNMENTS
Others have looked at the issue of developing good assignments for CS1 courses; e.g. see [2, 6]. Building on past ideas and updating them for a more advanced and complex world, we have developed the following eight criteria which we consider in developing a programming assignment for our CS1 course. A good programming assignment in CS1 should:

- be based on a real-world problem. Toy projects with toy data are less motivating to the students, and convey a false sense of what the world of computer science contains.
- allow the students to generate a realistic solution to that problem. They should be able to generate a system which generates useful results, even if it might not be entirely realistic in its interface or content.
- allow the students to focus on current topic(s) from class within the context of larger program. The students still need to practice on particular topics (e.g. strings, or external files), but we want them to do that within a realistic wrapper.
- be challenging to the students. We find that our students rise to the challenge and move forward faster when they have to work to and beyond their assumed boundaries.
- be interesting to the students. A practical, current and interesting issue drives them to work harder on the problem, where an overly-theoretical or simplified project tends to decrease their effort and their learning.

ITiCSE'06, June 26-28, 2006, Bologna, Italy.

- make use of one or more existing application programming interfaces (API). This allows them to gain practice in using specific objects, as well as learning that much of their future software development work will consist of making use of other APIs rather than developing their own raw code.
- have multiple levels of challenge and achievement, thus supporting possible refactoring. The problem should have a reasonably straightforward basic solution that all students are capable of solving. It should also provide additional challenges that cause the student to solve additional problems and possibly have to refactor their original solution if it is found to be lacking.
- allow for some creativity and innovation. The opportunity should be given for some creative or innovative addition or alternate solution to the problem.

3 EXAMPLE – WEB CRAWLER

A web crawler (aka robot or spider) is a program that automatically traverses the web's hypertext structure by retrieving a document, and then recursively retrieves all documents that are referenced. Web crawlers can be used for a number of purposes, including indexing, HTML validation, link validation, "what's new" monitoring, and mirroring. We have actually released two version of the web crawler assignment. One version uses the web crawler to perform link validation (that is, looking for broken links) and the other acts as a mirroring program that copies an entire web site to a local directory. Both versions are quite similar in their basic structure.

3.1 ASSIGNMENT SPECIFICATION

In either version, a basic web crawler needs to be built. For succinctness we will refer to the link validation version in the following description. A web crawler by definition is recursive since we never know how deep our web sites will be. However, given this is a relatively early assignment for CS1, recursion has obviously not been covered yet nor do we intend to cover it until CS2. Thus, we show the students how to build an iterative version of the web crawler that uses a queue structure (FIFO processing) to perform a breadth first search of the web site. In particular, they will need to create two structures. The first needs to keep track of which URLs the web crawler still needs to look at, which we call the "to-do" list. The other needs to keep track of the URLs they have already processed since HTML links can refer back to previous HTML pages and we don't want to get caught in circular loops. This is called the "processed" list. The to-do list needs to be a queue. In the past we have simply used a Vector and treated it as a queue. However, in Java 1.5 there is now a Queue interface so we now teach them to use the proper data structure for the job. The processed list can simply be a List.

Once these structures have been set up, the students simply need to follow the following algorithm:

- For each HTML page that the crawler need to look at:
  - Get the HTML contents of the page from across the network
  - Parse the HTML contents to find all the links on the page
  - For each found link:
    - Determine if the link is broken, recording that information in the broken link report
    - Determine if the link is one we need to recursively look at

In order to get the contents of the HTML page from across the network the students have to learn how to use the Java networking classes, in particular the URL class. At this point in the semester the students have already learned how to open up a BufferedReader to a file in order to read/write its contents. Using the URL class is very similar:

```
URL aURL = new URL("http://foo/default.htm");
URLConnection con = aURL.openConnection();
InputStream is = con.getInputStream();
InputStreamReader isr =new
InputStreamReader(is);
BufferedReader reader = new
BufferedReader(isr);
```

They can use the BufferedReader just like they did with file IO, thus driving home the fact that IO in Java is just about streams even if the other end is not directly connected to a local file. This will prove a useful concept to know in CS2 when we talk extensively about sockets and distributed computing. As a starting point, we teach them to simply read in the entire file line by line and concatenate them together to form a String that represents the entire HTML page. There are some variations on this based on XHTML specs and more advanced parsers, but those extensions will be discussed later.

The next step is to identify all the links on the page that was just read. Again, there are several more advanced parsing options that one can use (discussed later), but as a simple starting step we teach the students to use a combination of StringTokenizers and substring searching to identify the anchor tags. A StringTokenizer can easily be configured to hand back tokens that match HTML tags in the document. Thus, the job simply becomes searching for which tokens are anchor tags. Once an anchor tag is identified then substring searching can be used on that tag (which is now a string by itself) to parse out the destination of the href attribute.

The next stage is to determine if the link is broken. We classify broken links into 3 categories:

1. Malformed link
2. Well-formed link, but points to a server that doesn't exist
3. Well-formed link and points to a valid server, but the requested file doesn't exist.

The first two are fairly easy to handle as they simply involve handling exceptions (MalformedURL and IO respectively) at the different points in connecting. However, the 404 error (#3 in our list) is harder since an actual web page will be returned. This page consists of HTML tags that state that you just tried to access a nonexistent page. Each web server has its own way of generating this page, so you can't count on uniformity. Thus, the students need to drop down to the HTTP protocol level that is doing the actual transfer of the HTML page and check the response code that is returned. While this is not hard to accomplish it does generate a good discussion of protocols and

the layers of abstraction that are placed on top of the low-level protocols.

Links that are marked as broken are then recorded in an external file for later viewing by an administrator. Links that are not broken are then put on the to-do list as long as they don't already exist on the processed list. We also add the restriction that links to different domains not be followed as this utility is generally only run on a single domain. The processing continues until the "to-do" list becomes empty.

3.2 ASSIGNMENT DISCUSSION

Doing the parsing using low-level string manipulation has pros and cons. On the good side it really drives home how to manipulate strings since the parsing problem is certainly not trivial. However, it can become messy when given a real web site. For example, links can be placed in more than just anchor tags (JavaScript, framed pages, etc.), not all anchor tags represent links, within an anchor tag there can be lots of variation in the formation of the href attribute (e.g. spacing, quotes), etc.

To minimize these issues and keep the focus on basic string parsing, we provide a test web site for which the students to run their code. This site has very well defined HTML and makes the above parsing issues non-existent. The test site is also quite useful as a benchmark for testing their code against since it is a known quantity. In real-world assignments, this is a very important feature as it can help you guide your students through their debugging much easier since you know the fixed answer for the test site.

Additionally, it provides another benefit in our opinion. After they have their crawler working on the basic test site, they need to try it on some real sites. The students very quickly realize that things are not as structured and neat in the real world and their crawlers quickly break. This seems to come as a shock to many of them since they failed to realize the variation that exists in the real-world and had not coded their project to be able to handle it.

However, just noting that their code doesn't work in these more complex situations is not enough. We make them rework parts of their code so it can handle some of the more advanced variation in real sites. The choice as to what they handle and how they modify their code is up to them, but they need to make it handle at least two additional problems found in real sites if they want full credit. They need to write up a small document that describes the issue that caused the problem and what they did in their code to handle it.

We have found that this sort of structured test problem to real-world transition is a very powerful technique to help them learn about code robustness and error checking – things that are often left untouched in basic CS1 programs. Also the fact that they correct at least two problems of their choice gives them a sense of creativity that they may not experience normally.

Overall this assignment was a big success from our point of view in terms of what they learned and the quality of their solutions. But it was also successful in terms of student interest, as they found it quite fun to build a real tool that they could run on actual sites. They really liked telling us when one of the CS department web pages had a broken link!

4 EXAMPLE – SPAM EVALUATOR

A spam evaluator is a program that processes a mail folder to determine the probability that each message within the folder is unwanted bulk email, or spam. A spam evaluator would normally be used as part of a larger system (a spam filter) which would actually move messages to another Junk Mail folder. In order to keep the assignment manageable in a CS1 context and avoid issues with possible loss of moved email messages, we have focused on the spam evaluator portion of the overall spam filter.

4.1 ASSIGNMENT SPECIFICATION

The basic algorithm for a spam evaluator is straightforward:

- Connect to the mail store
- Open the mail folder and get the messages
- For each message in the folder
    - Determine the likelihood that this message is spam

There are many methods that spam filters use to determine if a particular message is spam; for an example, see [7]. Most use some sort of probabilistic combination of the presence and/or absence of certain spam indicators. Fundamental indicators often include, but are not limited to, the following:

- the inclusion of certain keywords (in the subject header and/or the message body), such as the word "Viagra".
- the original source IP address or name matching a "blacklisted" IP address or name, that is normally kept in separate file or other data structure.
- the inclusion of other significant features in the message, such as multiple images or a large image.
- the presence of spam avoidance techniques, such as inserting a period character between words, using digits or special characters to substitute for certain letters (e.g. using '!' for the letter 'I'.

While the number and type of indicators and the probabilistic formulas for the likelihood of a message being spam can be quite complex, we have found that a simplified combination of these features can still be quite realistic and accurate in evaluating mail messages for their spam content.

The above indicators rely heavily on string matching, and several of the indicators also involve working with external files (e.g. a keyword file, a blacklisted site file, and possibly an output file for the results). This makes this assignment fit naturally at the point when the String class and external files (including classes like BufferedReader and BufferedWriter) are introduced.

To access a mail store and mail folder in Java, the students need to understand and use the Java Mail API [3]. We first gave this assignment using JavaMail 1.2, which uses a relatively straightforward set of mail API calls to connect to a mail store, open a given folder, and to pull out a message from within a folder. A stripped-down version of the basic code is below:

```
Properties props = System.getProperties();
Session session = Session.getDefaultInstance(props, null);
Store store = session.getStore("imap");
store.connect(server, user, pass);
folder = store.getDefaultFolder();
folder = folder.getFolder(mailbox);
folder.open(Folder.READ_ONLY);
Message [ ] msgs = folder.getMessages();
```

We have now moved to using JavaMail 1.3, which requires a slightly more sophisticated model of providers. With either version of JavaMail, we have given basic explanations of the API and given examples of how to open a store and folder and display one or more messages, and this is easily understood by most students. At a minimum, they should be able to read the text of each message line by line and concatenate these text lines into one large String that represents the entire message body. This body can then be analyzed for spam content.

The next step is analyzing the mail headers. We show them that the various mail headers (e.g. From, Subject, To, Date, Received, Content-Type) are basically name/value pairs, with the name being the type and the value being the actual content. There may be multiple headers with the same name; e.g. the set of Received headers shows the path of the message across the Internet by indicating the host chain that is used in transmission of the message. We again explain how to get a single or a group of headers as an Enumeration, iterate through the names and values, and build a Java String containing the header content.

Now that the students can get the header or body content, the next step is basically doing matching between possible content (e.g. keywords, blacklist addresses) and actual content. While most students start by manually manipulating and comparing Strings using substring searching, this also provides us with an opportunity to talk about more sophisticated pattern matching, as we discuss below.

4.2 ASSIGNMENT DISCUSSION

At this point, the students should be able to process and analyze well-formed messages, and we provide a test email account (set up by our system staff, and populated by the instructor) with a set of both good and spam email messages. The students can also analyze their own Inbox, adding to the realism of the assignment. However, after the assignment has been released for a certain amount of time and most students have the basic functionality, we provide the final set of mail messages, which contains variations and issues that may cause their spam evaluator program to fail. These issues include:

- empty subject (generally creating a null pointer exception in their program, unless they have checked to make sure the Subject header they pull off of the message is not null)
- message with attachments (which leads to a discussion of message parts)
- message with empty body (e.g. just with an attachment)
- messages with a different content type than plain text (e.g. text/html)

The students are required to solve as many of these problems as possible in order to maximize their grade. Most are solved fairly quickly, demanding some additional information and reasoning beyond the base case.

This assignment is also a popular one with the students. Not only do they enjoy learning how real spam filters work, but they also like being able to access their own email through a program they wrote.

5 ADVANCED STRING PARSING

In both of the example assignments a significant amount of String and HTML parsing was needed. The entry level versions of these assignments simply use StringTokenizers and substring searching to accomplish these tasks. These parsing and string manipulation techniques are good basic techniques to know, but the students soon see that they have their limitations. This opens the door for discussing more complex parsing and searching techniques that, while on the surface seem to require more setup and understanding of their associated APIs, will pay off in a big way in the more complex situations. It has been our experience that students often don't see the need to use "over the top" parsing if it is introduced too early. They need some motivation to want to use it. These assignments also provide a nice backdrop for doing refactorings using these advanced concepts later in the course or in following courses. The following sections list some of the advanced techniques that can be used.

5.1 SPLIT AND REGULAR EXPRESSIONS

The Java API does declare the StringTokenizer class to be maintained for backward compatibility at this point, and recommends the use of the String class split method and the java.util.regexp package. We still talk about StringTokenizer first to introduce the general concept of Iterators, but now show them the comments regarding StringTokenizer being at least unofficially deprecated and then show them the newer split functionality. The Java 1.5 Scanner class provides another option.

The more powerful tool for manipulating String data is the java.util.regexp package. While this is a complex topic, we still have introduced the package by example, in order to provide tools for the advanced students and for future, more complex work. For example, split can be used to pull apart the pieces of a numeric internet address, and a regular expression can be used to see if each piece has a valid format (one to three digits – more work must be done to see if these numbers are valid).

```
String ip = "137.28.109.29";
String [] pieces = ip.split("\\.");    // escape the period; not wildcard
Pattern p = Pattern.compile("\\d{1,3}");   // 1 to 3 digits
for (int i = 0; i < pieces.length; i++) {
   if (p.matcher(pieces[i]).matches()) {
      System.out.println("valid form - IP number piece"); } }
```

5.2 HTML PARSING VIA SWING

Especially when trying to find links in an HTML document, one would like to have an automatic parser. Java provides one as part of Swing [5]. One simply needs to create a parser and set up the callbacks for the tags of interest. However, while this may be simple in concept, there are a couple of catches to using it early in CS1. First it is difficult to obtain an HTMLEditorKit.Parser class in the first place. One needs to create a new class that overrides a constructor. The same is true for the callbacks with respect to having to override certain methods.

While "magic lines" of code can certainly be given to the students to get them started, the idea of extending and overriding is generally handled later in the semester and thus may be to conceptually abstract for them to handle at this stage. However, this does make a nice follow-on assignment or lab later in the semester. They already have done, for example, the link

validating web crawler with simple String manipulation and seen some of the issues as the data got complex. Thus, they have the motivation necessary for trying a refactoring with something more complex like this once they have learned how to implement inheritance and override methods.

5.3 XML

Another advanced option for HTML parsing is to use one of the XML parsing technologies such as DOM [4]. This, of course, assumes that the HTML you are processing is XHTML. While this generally can't be something that is assumed on the web, one can make a simple test site that does meet this specification. This will also vastly simplify the reading in of the document as well, basically reducing it to a single line of code. And the XML DOM classes have powerful searching mechanisms build into them so finding the tags you are interested is simplified as well.

However, in order to use DOM one needs to understand the tree abstraction of an XML document. Again, while this is not a complex concept, it can be a bit abstract for the start of CS1 in our opinion. We usually save this information for our CS2 class. But at that point, a refactoring of an assignment such as the web crawler using this new technology makes a great assignment.

6 REAL-WORLD ASSIGNMENT ISSUES

We are repeatedly challenged by several concerns in developing real-world assignments, and address three of them here.

First, it is more important than usual for the instructor to develop a solution in advance for larger real-world assignments. It is too easy for time-pressed instructors to come up with a good idea and write it up as an assignment before solving it themselves. However, with a larger, real-world assignment, we are more likely to run up against real-world complexities, including subtle features in the APIs being used or interactions of the used components. As an instructor, developing a solution first avoids problems that may frustrate the students and potentially interfere with completion by the given deadline.

Second, a related benefit of pre-developing the solution is that it gives us as instructors the opportunity to improve the program's design through analysis and refactoring. Students at this level can certainly not be expected to design their own solutions, and may make poor decisions if allowed to do so. We try to provide a good design based on our own solution, and basically can then remove sections of our code in areas relevant to the topics which we are trying to have them practice. This also allows us as instructors to gauge the size and complexity of the work assigned.

Third, in real-world assignments there is a delicate balance that needs to be struck between using toy data and real-world data. Our approach is to start with simple test data to get the students to a place where they understand the basics of the problem and have generally working code before moving on to the real-world data. This leads to more students getting at least a basic working project, which in turn builds their confidence. However, it still educates them as to how messy the real world is when they see their code break on what is fairly simple real-world data. It also introduces them to the world of refactoring which is a powerful software engineering technique. Finally, it fosters a sense of creativity by allowing them to try and solve whichever complex problems they are most motivated by in order to get their code to work on some advanced form of real-world data.

7 SUMMARY

At the start of CS1 one needs to focus on string manipulation, arrays and lists, and using built-in objects. It can be difficult to come up with interesting real-world assignments that focus here.

We feel that both the web crawler and the spam evaluator cover the criteria we set forth in defining a good CS1 assignment. They are real-world problems that are relevant and interesting to students, as they understand the need for spam filtering and for web crawling since they see them in use every day. The way we have constructed the assignments allows the focus to be on those basic string manipulation and array/List processing topics presented in class, without making it seem as if the assignment is just on these topics. New APIs must be used (e.g. JavaMail, networking) in order to complete the assignment, giving them valuable experience in learning how to use existing code rather than always generating their own code. Having the students create a base solution to a small set of test data followed by moving to real-world data allows most students to complete the basic solution while the real-world parts add challenge and creativity to entertain even the best students. Finally, the resulting programs can actually be used in the real world to find broken links or to filter their email. This gives the students a strong sense of accomplishment as they have produced something real.

We encourage others to use variations on these assignments (materials available upon request) as well as develop new assignments based on the principles we have outlined.

REFERENCES

1. Denning, Peter J., and McGettrick, Andrew, "*Recentering Computer Science"*, Communications of the ACM, Vol. 48, No. 11, November 2005, pp. 15-19.
2. Feldman, Todd J. and Zelenski, Julie D., "*The Quest for Excellence in Designing CS1/CS2 Assignments*", Proceedings of the 27th SIGCSE Technical Sympoisum on Computer Science Education, 1996, pp. 319-323.
3. Java Mail API, http://java.sun.com/products/javamail
4. Java XML Tutorial, http://java.sun.com/xml/tutorial_intro.html
5. Parsing HTML with Swing, http://www.samspublishing.com/articles/article.asp?p=31059&seqNum=1
6. Reed, David, "*The Use of Ill-Defined Problems For Developing Problem-Solving and Empirical Skills in CS1*", Journal of Computing in Small Colleges, Vol. 18, No. 1, October 2002, pp. 121-133.
7. Robinson, Gary, "*A Statistical Approach to the Spam Problem*", Linux Journal online, http://www.linuxjournal.com/article/6467