Project Volunteer Information

Thank you for volunteering your time to help me with my senior project. In this document I will attempt to cover my expectations from you as well as any resources that you might find useful while working on my project. Please let me know if you have any questions; I'd be happy to discuss the project with you!

Expectations

Over the course of this project, you will be implementing a series of programs that illustrate various aspects of systems programming: such as memory management, pointer manipulation, and conditional compilation. Each volunteer is expected to have

- Some experience with C++
- Very little experience with either D or Rust

Error Logs

For my project, I am interested in the types of errors that are common to each of the languages you will be working in. Specifically, here are the types of errors I am studying:

Category	Description
Syntax error	Any error caused by your program failing to parse. Could be a typo in a variable name, forgetting to put braces, etc.
Logic error	Your program parses correctly, but it does not run as intended due to human error. Off-by-one errors are a common example of a logic error.
Resource error	The incorrect use of memory management. For example, your program dereferences a `NULL` pointer or uses deallocated memory.

In order to obtain data on the types of errors that you run into, I would like you to keep a log of every error you run into while working on each project. Each log should contain the type of error you ran into, a brief description of the error, and whether the error was caught at compile-time or run-time. Here's an example of a log:

Category	Description	When caught
Syntax error	Wrote "sring" instead of "string"	Compile-time
Logic error	Accidentally iterated one time too many	Run-time
Resource error	Used a freed pointer	Run-time

In short, if the compiler spits out an error, or you have to change your code after you noticed something didn't work, log it! The error logs are key to my study so please take care that they are accurate.

Rust Note

Since Rust is rather unstable, you might receive warnings about deprecated or unstable features. You may fix these, and you do *not* have to include them in your logs. However, if you are unsure if an error should be included or not, please include it anyways.

Implementation

Each volunteer will be selected to implement their series of programs in D or Rust.

Resources

In order to keep the learning environment as free of bias as possible, the resources you may use to learn Rust and D should be limited to the following sites (i.e., no StackOverflow, no IRC, etc.):

Additional Help

If you would like additional help, you may contact me in which case I will decide to offer help at my own discretion. I will offer help with:

- Algorithmic Questions
- Clarification
- Installing the languages
- Locating information about the languages in the books or docs

I will not:

- Look at your code
- Help you solve compile errors

You are free to use whatever editor you wish. Please install a syntax highlighting package for your editor; I know that packages exist for Sublime Text, emacs, and vim. Please do *not* install any autocomplete packages, as these may skew the results of the study.

Development

There will be test cases provided in order for you to test your implementation. However, there is no problem if your program does not pass all cases. As long as you spend a significant amount of time on each program (say, 90 minutes), it doesn't matter if your program "works" in the end.

I may use snippets of your code in my final paper, in which case your work will be kept anonymous.

Installation

It's probably easiest to use your own computer to install each language's compiler. However, if you have a Windows PC, to save yourself some installation headache I recommend either dual-booting or running a VM of a Linux distro such as Ubuntu, or using the lab Macs. I've set up `project` with the needed dependencies of each language as well.

You may find language-specific installation guides in the Rust resources and D resources documents.

Programming

You will be asked to implement the following programs (listed in order from easiest to hardest).

Note: While I don't expect you to handle *all* errors that your program might run into, if you run into exceptional conditions (that is, empty input, invalid indices, etc.), please handle the errors as you feel fit. This could include throwing an exception or returning an `Error` object.

Sentence Splitter (source)

Define a function capable of splitting a text into sentences. The standard set of heuristics for sentence splitting includes (but isn't limited to) the following rules:

Sentence boundaries occur at one of "." (periods), "?" or "!", except that

- Periods followed by whitespace followed by a lower case letter are not sentence boundaries.
- Periods followed by a digit with no intervening whitespace are not sentence boundaries.
- Periods followed by whitespace and then an upper case letter, but preceded by any of a short list of titles are not sentence boundaries. Sample titles include Mr., Mrs., Dr., Jr.
- Periods internal to a sequence of letters with no adjacent whitespace are not sentence boundaries (for example, www.aptex.com, or e.g).
- Periods followed by certain kinds of punctuation (notably comma and more periods) are probably not sentence boundaries.

Write a function `splitSentences(string input)` that, given a string, prints each sentence on a separate line. I am only concerned with the function, so you may pass your test input to the function in any way, whether it be hard-coded, or through stdin, etc.

Note: For simplicity, we will assume that we only have one space between each sentence.

Skills covered

- String manipulation
- Branching

Integer Linked List

Define a linked list data structure that operates on integers. The linked list should support:

- Insertion in O(n) time.
- Deletion in O(n) time.
- Retrieval in O(n) time.
- A method to retrieve the size.
- A method to retreive the head of the list.
- A method to retrieve the tail of the list.

The IntegerLinkedList class should implement the following Java-like interface:

```
/**
 * A linked list containing only integers.
 */
interface IntegerLinkedList {
  /**
   * Inserts an element into the linked list at the specified index, shifting
   * any elements already in its position down.
   */
  public void insert(int index, int element);

  /**
   * Removes an element from the specified index, shifting any elements already
   * in the list back up.
   * @return The element that was removed.
   */
  public int remove(int index);

  /**
   * Retrieves an element at the specified position
   */
  public int get(int index);

  /**
```

```
    * Returns the number of integers in the linked list.
    */
  public int size();

  /**
    * Returns the first element of the linked list.
    */
  public int head();

  /**
    * Returns the last element of the linked list.
    */
  public int tail();
}
```

Skills covered

- Object-orientation
- Pointer manipulation
- Memory allocation

Generic Array List

Implement an array list class (also known as a dynamic array). Unlike the linked list class, this class should support generic elements, like ArrayList in Java. You should use templates (D) or generics (Rust) to implement this class.

The array list should support:

- Insertion in amortized O(1) time.
- Deletion in O(n) time.
- Retrieval in O(1) time.
- A method to retrieve the size.

The ArrayList class should implement the following Java-like interface:

```
/**
 * An array that dynamically resizes as elements are added.
 */
interface ArrayList<E> {
  /**
    * Inserts an element into the array at the specified index, shifting any
    * elements already in the list down.
    */
  public void insert(int index, E element);

  /**
    * Removes an element from the specified index, shifting any elements past its
    * position back up.
    * @return The element that was removed.
    */
  public E remove(int index);

  /**
    * Retrieves an element at the specified index.
    */
  public E get(int index);

  /**
    * Returns the number of elements in the array.
    */
  public int size();
}
```

Skills covered

- Object-orientation
- Generic programming
- Memory allocation

Parallel Mergesort

Implement the parallel mergesort algorithm. This algorithm should use std.parallelism (D) or std::thread (Rust). You should also implement a sequential threshold of 10, meaning that if the number of elements in a subarray is less than 10, you should not fork a new thread.

Your function should take in an array.

Skills covered

- Recursion
- Concurrency

Brainfsck Interpreter

Write an interpreter for the brainfsck programming language (thankfully, you are not required to write any brainfsck programs on your own).

Your interpreter should take in a file name as the first argument to the program, and respond to standard input and output as specified in this Wikipedia article. Hint: your interpreter may need to perform multiple passes on the input to handle brackets.

Skills covered

- File I/O
- Pointer manipulation

Completion

Upon completion of each program, you may email your code and error log to me or upload it to a source-hosting website such as GitHub or BitBucket.

Conclusion

Thank you again for volunteering! If you would like to contact me, I may be reached on Facebook, through email at acr02011@mymail.pomona.edu, or through my phone at (314) 440-8830.