

Goals and Overview

The Assignment, Part 1: Reading ...

The Assignment, Part 2: Coding

Problem Specification

Implementation Constraints an...

Getting the Given Code

Handing in your code

Good luck!

PA1 Block Chain

Due: Monday, January 28 at 11:59 PM

⚠ Lab machines

We will be grading the PAs for this course using the remote Linux machines. Any errors that you run into as a result of developing your code on [other machines](#) are your own responsibility.

Goals and Overview

In this PA (Programming Assignment) you will:

- learn about the course programming environment
- learn about vectors
- learn about pointers
- learn about linked lists
- learn about management of dynamic memory

The Assignment, Part 1: Reading and Understanding the Policies

Read and understand the following information:

- [Course Info](#)
- [Collaboration Policy](#)
- [Coding Style Policy](#)

⚠ Important Information

The information in these links is relevant to not just this PA, but every future PA as well, and you **will** be held responsible for having read and understood all of the information. So, part of this PA assignment is to read over this part 1 material, and to gain a complete understanding of it before moving on to part 2 of this assignment. This means that you should **ASK QUESTIONS ON ANY OF THIS PART 1 MATERIAL THAT YOU DO NOT UNDERSTAND**.

The Assignment, Part 2: Coding

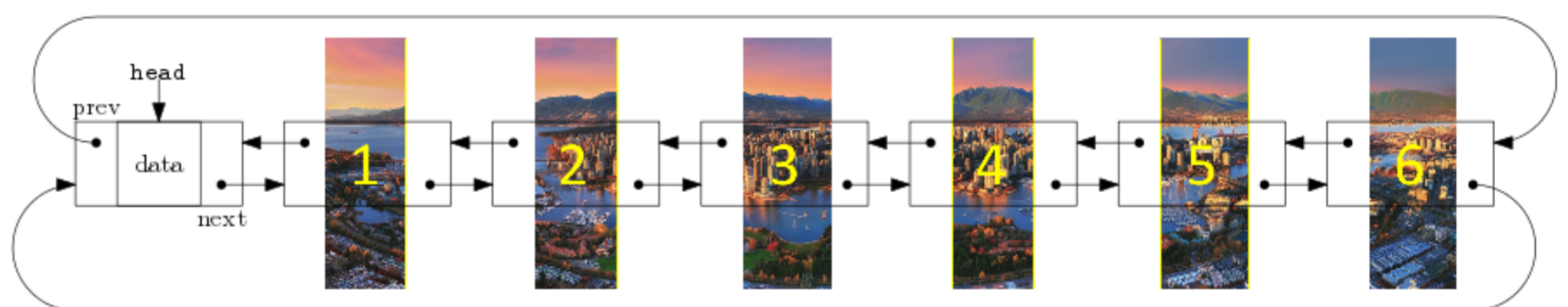
Problem Specification

A Linked List is a dynamic linear structure designed to hold any type of data. In this exercise, we develop and use a linked list to manipulate blocks of pixels from an image.

We have broken the image below into 6 Blocks.



Each Block is placed into a Node of a Chain, as shown here:



The Chain can be rearranged, and the image reassembled to create fascinating visual results.

We have provided a starting point for achieving this functionality. It is your task to complete and expand on our implementation.

Specifications for each function you write are contained in the given code. The list of functions here should serve as a checklist for completing the exercise.

In `block.cpp`

- `int width() const`; Returns the width of the current block.
- `int height() const`; Returns the height of the current block.
- `void build(PNG & im, int column, int width)`; From `im`, grabs the vertical strip of pixels whose upper left corner is at position `(column,0)`, and whose width is `width`.
- `void render(PNG & im, int column) const`; Draws the current block at position `(column,0)` in `im`.
- `void greyscale()`; This function changes the saturation of every pixel in the block to 0, which removes the color, leaving grey.

In `chain.cpp`

- `void clear()`; Helper function for destructor and assignment operator.
- `void copy(const Chain & other)`; Helper function for copy constructor and assignment operator.
- `~Chain()`; Destructor.
- `void insertBack(const Block & ndata)`; Insert a new node at the end of the Chain.
- `void moveBack(int startPos, int len, int dist)`; Move `len` nodes from `startPos` toward the end of the Chain, shifting by `dist` positions.
- `void roll(int k)`; Move `k` nodes from the end of the Chain to the front. Their order does not change.
- `void reverseSub(int pos1, int pos2)`; Reverses the order of the nodes in the chain from `pos1` to `pos2`, inclusive.
- `void weave(Chain & other)`; consumes the nodes in `other` placing them in alternating positions in the current Chain. In the final chain, the original Chain will occupy the odd nodes, and the nodes from `other` will be in the even positions.

Implementation Constraints and Advice

We will be grading your work on functionality, efficiency, and memory use. All Chain functionality, aside from the insert and the copy functions, can be achieved by moving existing nodes, rather than by allocating new ones and/or making copies. If you are tempted to use the new function when you are manipulating the Chain, ask yourself if you can achieve your goal by reassigning pointers, instead.

The Chain structure is a circular doubly-linked list with one head sentinel. The sentinel is simply a place-holding Node, whose purpose is to define the start (and end) of the list. An empty list is simply a sentinel node whose next and prev pointers both point to itself. This creates a simplifying invariant: every Node containing data is guaranteed to have a node both before and after it. This invariant eliminates the special cases typically associated with the front and end of lists, thereby reducing the number of conditionals in your code. In our solution for `chain.cpp`, we have fewer than 3 conditionals involving pointers!

Finally, you'll note that we are asking you to implement special memory management functions for the Chain class. You should ask yourself why we have not made similar requests for the Block class. Why doesn't it need a destructor, copy constructor, or assignment operator?

Getting the Given Code

Download the source files from [pa1.zip](#), and follow the procedures you learned in lab to move them to your home directory on the remote linux machines.

Handing in your code

To facilitate anonymous grading, **do not** include any personally-identifiable information (like your name, your University ID number, or your cwl) in any of your source files. Instead, before you hand in this assignment, create a file called `partners.txt` that contains only the CSIDs of people in your collaboration partnership (if one exists), one per line. If you worked alone, include only your own CSID in this file. We will be automatically processing this information, so do not include anything else in the file. (If we must manually correct your submission, you may lose points.) As always, if you're working in a group, each group member must hand in the assignment. (Failure to cite collaborators violates our academic integrity policy.)

The following files are used to grade PA1:

- `block.cpp`
- `chain.cpp`
- `chain.h`
- `partners.txt`

All other files will not be used for grading.

Detailed instructions for handing in your work will be added to this page within the next few days.

Good luck!