CS 2110 Homework 7
Recursive Assembly

Youna Hoshi, Daniel Becker, Sam Gilson, Joshua Viszlai, and Madeleine Brickell

Spring 2019

Contents

1 Overview

In this homework, you will be writing assembly code for the following tasks:

- `decimalStringToInt()`: Convert a string to an integer
- `checkTreeEquality()`: Check to see if two binary trees are equivalent
- `reverse_ll()`: Reverse a linked list

You should follow the LC-3 calling convention when writing these functions. A detailed description of the calling convention is available in Part 5 of this PDF.

2 Instructions

2.1 Part 1

For the first part of this homework, we will be writing the `decimalStringToInt` function that was in homework 2. In order to do this, we need to write three subroutines: `mult(), length(),` and finally `decimalStringToInt()`

2.1.1 Mult

For the mult subroutine, we will be taking two parameters and multiplying them together. Some pseudocode to help you implement this function is seen below:

```
mult(int a, int b)
{
    var result = 0;
    while(b > 0) {
        result += a;
        b--:
    }
    return result;
}
```

2.1.2 Length

The length subroutine takes an address as a string as its only parameter and calculates the length of the string. Pseudocode:

```
length(String a)
{
    int length = 0;
    while(a.charAt(length) != 0) {
        length++;
    }
    return length;
}
```

2.1.3 decimalStringToInt

Finally, we will write the `decimalStringToInt` subroutine. This function takes in an address to a string that is a positive number and returns an integer representation.
Example:

$$\text{decimalStringToInt}(\text{"125"}) = 125$$

Our function will call both the mult() and length() subroutines as seen in the pseudocode below:

```
decimalStringToInt(String decimal)
{
    int ret = 0
    for (int i = 0; i < length(decimal); i++) {
        ret = mult(ret, 10);
        ret = ret + decimal.charAt(i) - 48;
    }
    return ret;
}
```

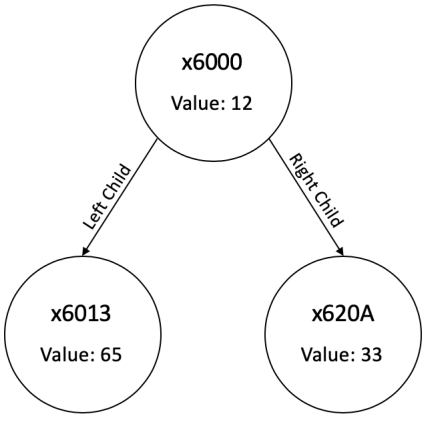
2.2 Part 2

For this part, you will be given two binary trees and tasked with determining whether or not they are equal. A binary tree is made up of nodes, each with a left child, a right child, and a data value. The two arguments passed in will be the addresses of each binary tree's root node (i.e the top most one). The nodes will be formatted as shown below. Two binary trees are "equal" if they have the same node structure and the same data values in their corresponding nodes. If two trees are equal, return 1, otherwise return 0.

2.2.1 Tree Format

Here is an example of how a binary tree will be laid out for this assignment. Each node consists of 3 contiguous memory words. The first is the address of the left child. The second is the address of the right child. The third is the data held at that node. Here is an example of binary tree in this notation:

Address	Value	Description
x6000	x6013	Address of x6000->left
x6001	x620A	Address of x6000->right
x6002	12	Data value for x6000
...	...	...
x6013	0	Address of x6013>left
x6014	0	Address of x6013>right
x6015	65	Data value for x6013
...	...	...
x620A	0	Address of x620A>left
x620B	0	Address of x620A>right
x620C	33	Data value for x620A



2.2.2 Pseudocode

Here is the pseudocode for checking if two binary trees are equal:

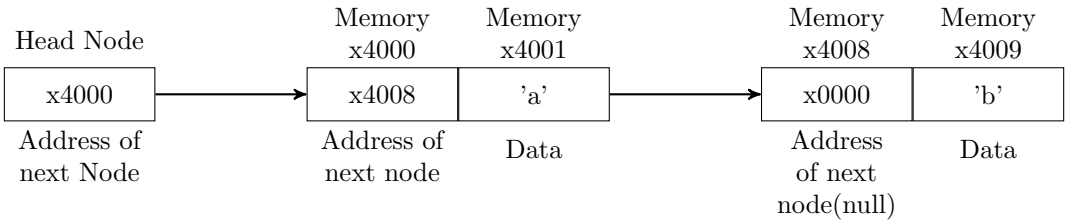
```
int checkTreeEquality(Node b1, Node b2)
{
    // Both are empty, so return 1
    if (b1 == 0 && b2 == 0) {
        return 1;
    }

    // If both are non-empty, return 1 if their data values are the same

    // and their children are equal. Return 0 otherwise
    if (b1 != 0 && b2 != 0)
    {
        if (b1.data != b2.data)
            return 0
        else if (checkTreeEquality(b1.left, b2.left) == 0)
            return 0
        else if (checkTreeEquality(b1.right, b2.right) == 0)
            return 0
        else
            return 1
    }

    // If one is empty and the other isn't, return 0
    return 0;
}
```

2.3 Part 3

For this part of the homework, you will be writing a function to reverse a linked list. This function will be recursive.

2.4 Linked List Data Structure

The below figure depicts how each node in our linked list is laid out. Each node will have two attributes: the next node, and a value for that node.



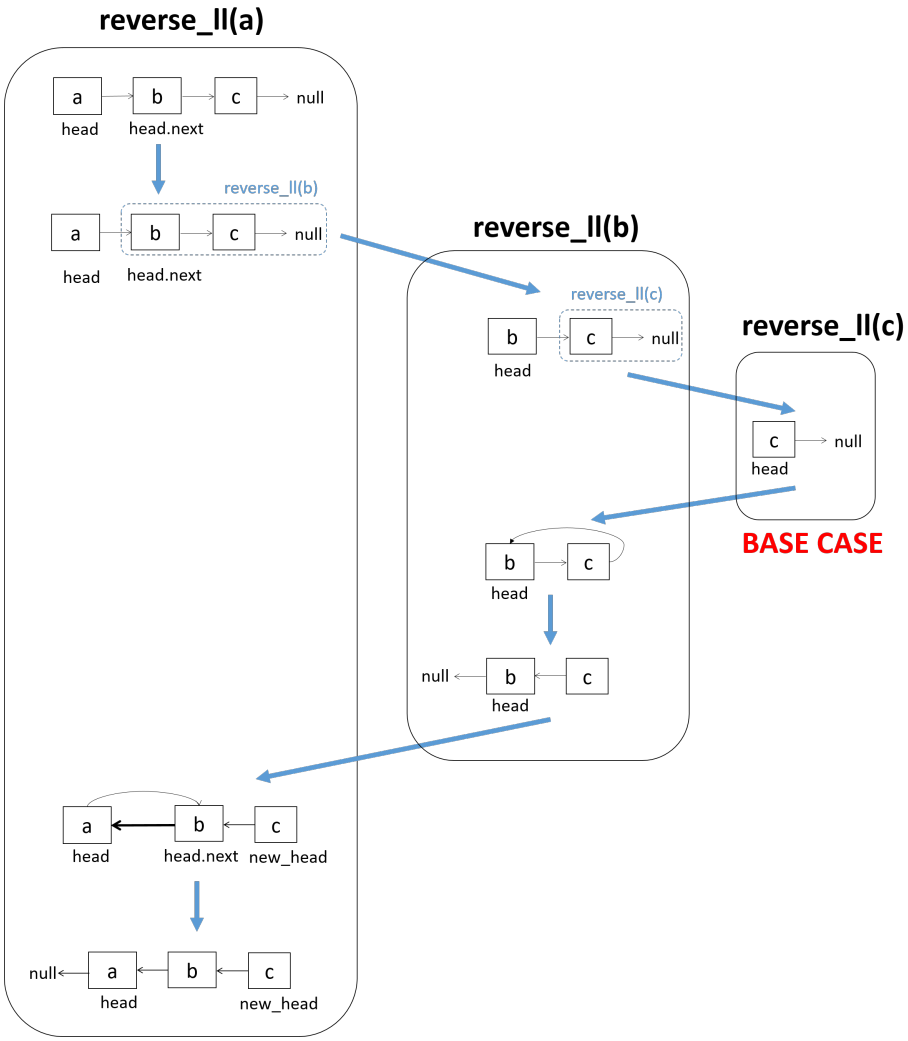
2.4.1 Pseudocode

The pseudocode for this function is as follows. **Note: null has a value of 0.** Since memory address zero is used to hold part of the trap vector, we can use zero as a value distinguished from all of the memory

addresses in our linked list.

```
reverse_ll(Node head)
{
    if (head == null || head.next == null)
    {
        return head;
    }

    new_head = reverse_ll(head.next);
    head.next.next = head;
    head.next = null;

    return new_head;
}
```

Below is a visual representation of how this function works. Reversing a linked list is equivalent to reversing all the nodes after the head node, and then appending the head node to the end of the reversed portion.



2.4.2 Helper Functions

The `reverse_ll.asm` file comes with a subroutine, `print_ll` which will print the values in the linked list to the console. This subroutine assumes that the R1 will hold the first node. So, you will load the first node into R1, and then call the subroutine with `JSR print_ll`.

3 Checker

To run the autograder locally, follow the steps below depending upon your operating system:

- Mac/Linux Users:
  1. Navigate to the directory your homework is in. **In your terminal, not in your browser**
  2. Run the command `sudo chmod +x grade.sh`
  3. Now run `./grade.sh`
- Windows Users:
  1. On **docker quickstart**, navigate to the directory your homework is in
  2. Run `./grade.sh`

Note: The checker may not reflect your actual grade on this assignment. We reserve the right to update the checker as we see fit when grading.

4 Deliverables

Please turn in the following files to gradescope:

1. `decimalStringToInt.asm`
2. `checkTreeEquality.asm`
3. `reverse_ll.asm`

5 LC-3 Assembly Programming Requirements

5.1 Overview

1. Your code must assemble with **NO WARNINGS OR ERRORS**. To assemble your program, open the file with Complx. It will complain if there are any issues. **If your code does not assemble you WILL get a zero for that file.**
2. **Comment your code!** This is especially important in assembly, because it's much harder to interpret what is happening later, and you'll be glad you left yourself notes on what certain instructions are contributing to the code. Comment things like what registers are being used for and what less intuitive lines of code are actually doing. To comment code in LC-3 assembly just type a semicolon (;), and the rest of that line will be a comment.
3. Avoid stating the obvious in your comments, it doesn't help in understanding what the code is doing.

   Good Comment

```
ADD R3, R3, -1          ; counter--
BRp LOOP                ; if counter == 0 don't loop again
```

Bad Comment

```
ADD R3, R3, -1          ; Decrement R3
BRp LOOP                ; Branch to LOOP if positive
```

4. **DO NOT assume that ANYTHING in the LC-3 is already zero.** Treat the machine as if your program was loaded into a machine with random values stored in the memory and register file.
5. Following from 3. You can randomize the memory and load your program by doing File - Randomize and Load.
6. Use the LC-3 calling convention. This means that all local variables, frame pointer, etc... must be pushed onto the stack. Our autograder will be checking for correct stack setup.
7. Start the stack at xF000. **The stack pointer always points to the last used stack location.** This means you will allocate space **first**, then store onto the stack pointer.
8. Do NOT execute any data as if it were an instruction (meaning you should put .fills after **HALT** or RET).
9. Do not add any comments beginning with @plugin or change any comments of this kind.
10. **Test your assembly.** Don't just assume it works and turn it in.

6 Rules and Regulations

6.1 General Rules

1. Starting with the assembly homeworks, any code you write must be meaningfully commented. You should comment your code in terms of the algorithm you are implementing; we all know what each line of code does.
2. Although you may ask TAs for clarification, you are ultimately responsible for what you submit. This means that (in the case of demos) you should come prepared to explain to the TA how any piece of code you submitted works, even if you copied it from the book or read about it on the internet.
3. Please read the assignment in its entirety before asking questions.
4. Please start assignments early, and ask for help early. Do not email us the night the assignment is due with questions.
5. If you find any problems with the assignment it would be greatly appreciated if you reported them to the author (which can be found at the top of the assignment). Announcements will be posted if the assignment changes.

6.2 Submission Conventions

1. All files you submit for assignments in this course should have your name at the top of the file as a comment for any source code file, and somewhere in the file, near the top, for other files unless otherwise noted.

2. When preparing your submission you may either submit the files individually to Canvas/Gradescope or you may submit an archive (zip or tar.gz only please) of the files. You can create an archive by right clicking on files and selecting the appropriate compress option on your system. Both ways (uploading raw files or an archive) are exactly equivalent, so choose whichever is most convenient for you.

3. Do not submit compiled files, that is .class files for Java code and .o files for C code. Only submit the files we ask for in the assignment.

4. Do not submit links to files. The autograder does not understand it, and we will not manually grade assignments submitted this way as it is easy to change the files after the submission period ends.

6.3 Submission Guidelines

1. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency let us know **IN ADVANCE** of the due time supplying documentation (i.e. note from the dean, doctor's note, etc). Extensions will only be granted to those who contact us in advance of the deadline and no extensions will be made after the due date.

2. You are also responsible for ensuring that what you turned in is what you meant to turn in. After submitting you should be sure to download your submission into a brand new folder and test if it works. No excuses if you submit the wrong files, what you turn in is what we grade. In addition, your assignment must be turned in via Canvas/Gradescope. Under no circumstances whatsoever we will accept any email submission of an assignment. Note: if you were granted an extension you will still turn in the assignment over Canvas/Gradescope.

3. There is a 6-hour grace period added to all assignments. You may submit your assignment without penalty up until 11:55PM, or with 25% penalty up until 5:55AM. So what you should take from this is not to start assignments on the last day and plan to submit right at 11:54AM. You alone are responsible for submitting your homework before the grace period begins or ends; neither Canvas/Gradescope, nor your flaky internet are to blame if you are unable to submit because you banked on your computer working up until 11:54PM. The penalty for submitting during the grace period (25%) or after (no credit) is non-negotiable.

6.4 Syllabus Excerpt on Academic Misconduct

Academic misconduct is taken very seriously in this class. Quizzes, timed labs and the final examination are individual work.

Homework assignments are collaborative, In addition many if not all homework assignments will be evaluated via demo or code review. During this evaluation, you will be expected to be able to explain every aspect of your submission. Homework assignments will also be examined using computer programs to find evidence of unauthorized collaboration.

What is unauthorized collaboration? Each individual programming assignment should be coded by you. You may work with others, but each student should be turning in their own version of the assignment. Submissions that are essentially identical will receive a zero and will be sent to the Dean of Students' Office of Academic Integrity. Submissions that are copies that have been superficially modified to conceal that they are copies are also considered unauthorized collaboration.

You are expressly forbidden to supply a copy of your homework to another student via electronic means. This includes simply e-mailing it to them so they can look at it. If you supply an electronic copy of your homework to another student and they are charged with copying, you will also be charged. This includes storing your code on any site which would allow other parties to obtain your code such as but not limited to public repositories (Github), pastebin, etc. If you would like to use version control, use github.gatech.edu

6.5 Is collaboration allowed?

Collaboration is allowed on a high level, meaning that you may discuss design points and concepts relevant to the homework with your peers, share algorithms and pseudo-code, as well as help each other debug code. What you shouldn't be doing, however, is pair programming where you collaborate with each other on a single instance of the code. Furthermore, sending an electronic copy of your homework to another student for them to look at and figure out what is wrong with their code is not an acceptable way to help them, because it is frequently the case that the recipient will simply modify the code and submit it as their own.

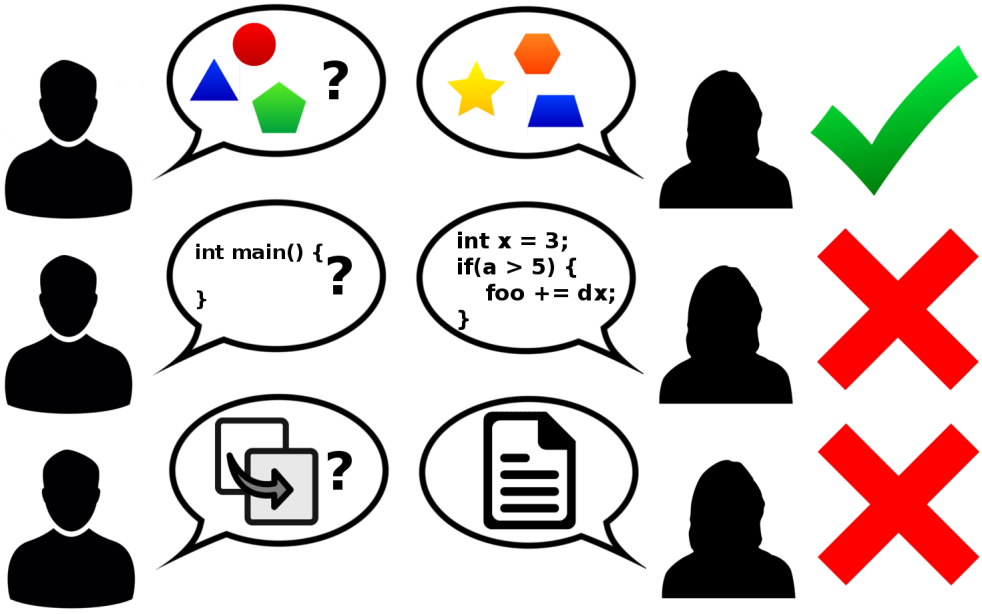


Figure 1: Collaboration rules, explained colorfully