Programming assignment 1 (120 points)

Submit Assignment

Due Feb 11 by 11:59pm **Points** 120 **Submitting** a file upload **File Types** tar
Available after Jan 20 at 12am

Introduction to C

Prof. Yipeng Huang

Rutgers University

2021 Spring

0. Introduction and setup

Learning goals

You will gain experience programming in C while reviewing data structures and algorithms concepts. You will practice using the Linux command line to compile, run, debug, and test programs.

Resources

▶ tarted early! You can post questions to the class Piazza (please avoid sending emails to the TAs and instructor). You can discuss the assignment with your classmates, but you cannot copy code from your classmates. We will be checking the assignments for identical and/or plagiarized code using automated tools. See the academic integrity policy at: **https://nbprovost.rutgers.edu/academic-integrity-students (https://nbprovost.rutgers.edu/academic-integrity-students)** . We will report any violations.

The computer science student space, CAVE, is closed but they are live and online remotely. They are available to help you in this class and your other CS classes. CAVE hours have morphed into long Code Red sessions as students come for tutoring and to discuss things they have problems with. The CAVE page is up to date (**https://resources.cs.rutgers.edu/docs/rooms-equipment/cave/**

(https://resources.cs.rutgers.edu/docs/rooms-equipment/cave/)
(https://resources.cs.rutgers.edu/docs/rooms-equipment/cave/))

Setup

If you do not already have access to ilab, activate your account here:
https://services.cs.rutgers.edu/accounts/activate/activate
(https://services.cs.rutgers.edu/accounts/activate/activate) .

For information on how to access the ilab Linux machines using ssh or x2go from a Windows or Mac machine, see: https://resources.cs.rutgers.edu/docs/new-users/beginners-info/
(https://resources.cs.rutgers.edu/docs/new-users/beginners-info/) .

Access the programming assignment files for this assignment by cloning this Github repository:

```
git clone https://github.com/yipenghuang0302/2021_0s_211.git
```

The files for this assignment are in the directory 2021_0s_211/pa1/.

---

1. goldbach: Goldbach's weak conjecture (22 points)

Goldbach's conjecture is a famous unsolved problem in mathematics. It states that every even number greater than two is the sum of two prime numbers. While the statement has been shown to be true for quintillions of even numbers, mathematicians have not yet devised a rigorous proof of the statement.

In this part of the assignment you will write a C program that demonstrates the related Goldbach's weak conjecture, which states that every odd number greater than 5 is the sum of three primes. This weaker claim is implied by the more famous Goldbach's conjecture, and this weaker claim has potentially been proven by mathematicians.

Your task is to write a C program called goldbach which will take an odd number greater than 5 as a command line argument:

```
./goldbach 7
```

Your program should then output to the command line a valid equation showing that the given number is the sum of three primes:

```
7 = 2 + 2 + 3
```

You will receive 5 points for a program that compiles, incremental points for passing a few test cases, and full marks if your program successfully passes 100 randomized inputs.

How to compile, run, and test your code

First, you should rename goldbach_provided.c to goldbach.c by entering:

```
mv goldbach_provided.c goldbach.c
```

Your program should be written in the file goldbach.c.

To compile your source code goldbach.c, you can type:

```
gcc -Wall -Werror -fsanitize=address -std=c99 -o goldbach goldbach.c -lm
```

We've provided several important useful compiler flags.

Alternatively, to build the executable, you can type:

```
make
```

Once your program is successfully compiled, you can run your program:

```
./goldbach 7
```

You should make sure your C program returns 0 indicating successful program termination.

To use the autograder script to test your program, you can type:

```
./autograder
```

or

```
python3 autograder.py
```

We will be automatically building, testing, and grading your assignment. Make sure that we can build your assignment by just using the Makefile and that we can run the program by invoking our autograder. Make sure to follow the required output format; any extraneous output such as debugging statements will confuse the grading program. You can check if you have accidentally modified your autograder:

```
git diff autograder.py
```

---

2. maximum: finding the N maximum elements of a list (22 points)

Your task in this part of the assignment is to write a C program that returns the largest elements of a list of signed integers. Your program should take as a command line input the path to an input file:

```
./maximum tests/test0.txt
```

The first line of this text file says how many numbers will be in the list. The second line of this text file says how many of the largest elements you should return. The third line is a list of positive and negative integers from which you should select the largest numbers.

You program should print to the command line the list of the largest numbers in any order, separated by spaces.

The corresponding expected outputs are in the answers directory: answers/answer0.txt.

---

3. matMul: matrix multiplication (22 points)

Matrix multiplication is a fundamental task in almost every area of computer science, including machine learning and solving scientific problems. You may wish to review the rules of matrix multiplication. Suppose matrix A has size L rows x M columns, matrix B has size M x N, then the matrix multiplication of the two matrices A x B = C, where matrix C has size L x N.

Your task is to write a C program that takes two command line inputs, indicating two files to load. The first file contains matrix A and the second file contains matrix B. For example:

```
./matMul tests/matrix_a_0.txt tests/matrix_b_0.txt
```

In both of these input files, the format is such that the first line contains the number of rows in the matrix, and the second line contains the number of columns in the matrix. Then, the file continues for several lines, each line corresponding to a row of the input matrix. In each row, the columns of the matrix are listed, each number separated by a space.

Your program should load these files and perform matrix multiplication of the two matrices. Then, your program should output L*N number of numbers reading out the elements of the product matrix C in row-major order: first print out the elements in the first row, separated by spaces, followed by printing the elements of the second row, and so forth.

The corresponding expected outputs are in the answers directory: e.g., answers/matrix_c_0.txt.

---

4. balanced: checking if braces are balanced or not using a stack (22 points)

You may have noticed that your favorite text editor or integrated development environment (IDE) can help you check if open parentheses, brackets, and braces are properly closed. That is, whether each open '<', '(', '[', and '{' is properly closed with a corresponding '>', ')', ']', and '}' without any intervening open parentheses, brackets, or braces.

For example, these are balanced expressions:

<({})>

({<>(){}})

But these are not:

({)

▶ <()>]

Your task in this part of the assignment is to write a C program using structs and pointers to build a stack data structure, and use that stack to check whether a string is properly balanced. Your program should take as a command line input the path to an input file:

```
./balanced tests/test0.txt
```

Your program should check whether the expression in the input file is balanced, and then print to the command line "yes" if the expression is balanced, and "no" if it is not.

The corresponding expected outputs are in the answers directory: answers/answer0.txt.

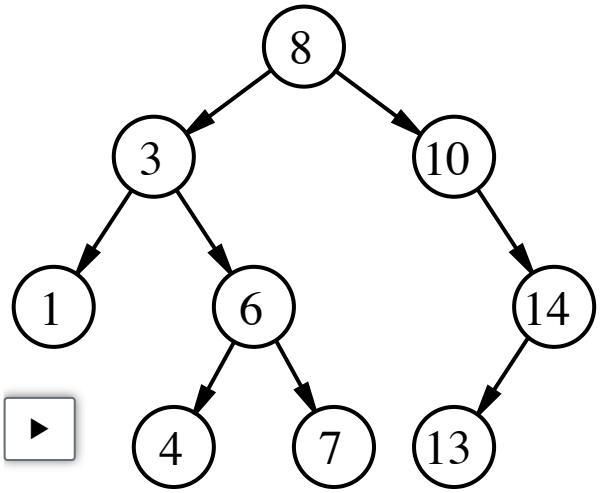
5. bstReverseOrder: reverse order traversal of a binary search tree (22 points)

Your task in this part of the assignment is to write a C program that constructs a binary search tree from a list of input numbers, and then print out the binary search tree in a reverse order traversal of the tree. You may find it helpful to review the properties of a binary search tree, and the various flavors of tree traversal order. In a binary search tree, the key in each node is greater than all keys in its left subtree, and is lesser than all keys in its right subtree.

Your program should take as a command line input the path to an input file:

```
./bstBreadthFirst tests/test0.txt
```

Each line of the input file lists a number to be inserted into the binary search tree. If a number has already been inserted, you can ignore the duplicate insertion. Since we are not performing tree balancing, everyone should arrive at the same binary search tree structure for any given input sequence. For example, an input sequence of 8,3,6,1,10,4,14,7,13 would lead to this unique binary search tree (image credit wikimedia):



Once the binary search tree is constructed, your program should print out the nodes in a depth-first, reverse in-order traversal of the tree. That is, for every node you visit, you should visit the right subtree first, then print the root node, and finally visit the left subtree. A reverse order traversal of the example tree above would return the numbers in descending order: 14, 13, 10, 8, 7, 6, 4, 3, 1.

The corresponding expected outputs are in the answers directory: answers/answer0.txt.

How to submit

From the pa1/ directory, you can run this command to check on the outputs of our autograder script.

```
./assignment_autograder
```

or

```
python3 assignment_autograder.py
```

When you are ready to submit, from the 2021_0s_211/ directory where you see the pa1/ directory, run this command:

```
tar cvf pa1.tar pa1/
```

Upload the file pa1.tar here on Canvas.

We will not be accepting late assignments. The Canvas submission site will close to enforce the deadline, so be sure to submit early.

If anything is not clear, reach out to your classmates and the instructors on the class Piazza!