Junior Qualifying Exam

Sample CS1 and CS2 Problems

Fall 2020, Computer Science Department, Reed College

On this and the following pages are some problems we gave on quals in 2019-20. They are similar problems to what you'd expect on the CSCI 121 and 221 parts of the CS Junior Qualifying Exam.

1. Consider the four different console outputs below

```
   **            **      **       **
   **            **      **       **
 ****          ****            ****
 ****          ****            ****
******        ******
******        ******
            ********
            ********
```

   The first is a staircase of height 3. The second is a staircase of height 4. The third is a staircase of height 1. And the fourth is a staircase of height 2.

   Write a Python function `def staircase(h)` that, when given a positive integer `h`, outputs the staircase of that height.

2. In programming languages that allow higher-order functions, like Python, there is a concept called "currying" (named for Haskell Curry). Currying is the process of taking a function with multiple arguments and converting it into an equivalent composition of functions that each take only one argument. For instance, suppose we have the following Python function:

```
def f(x,y):
    return 2*x + y
```

   If we "curry" `f`:

```
h = curry(f)
```

   Then `h` is a function such that `(h(x))(y)` yields the same result as `f(x,y)`.

   Write a Python function `def curry(f)` that takes a two-parameter function `f`, and returns the curried version of that function similar to `h` in the example above. Here is an example of its use:

```
>>> def f(x,y):
        return 2*x + y
>>> f(3,4)
10
>>> h = curry(f)
>>> (h(3))(4)
10
```

3. Using only the MIPS32 instructions described below, **write the code for a MIPS32 function** named with the label `numberOfBitsNotSet` that, when given a 32-bit integer parameter, returns the number of times a zero bit occurs within the 32-bit encoding of that integer. According to the standard calling conventions of MIPS32 assembly, the integer to be checked will be passed as register `a0` and the returned value resulting from the check should be stored in register `v0`.

   For example, the 32 bit sequence `00001110100100111100100101110000` encodes the value 244566384. Because this bit sequence contains 14 ones and 18 zeros, then your code should set register `v0` to 18 if `a0` is set to 244566384. If instead `a0` is set to 3 then the code should set `v0` to 30 since the bit sequence `00000000000000000000000000000011` has 2 ones and 30 zeros.

   The code for your function should be no longer than 32 instruction lines, excluding labels, and will only be given partial credit otherwise.

   MIPS32 coding guide

`li` $RD, *value*	–loads an immediate value into a register.
`add` $RD, $RS1, $RS2,	–add two registers, storing the sum in another.
`sub` $RD, $RS1, $RS2,	–subtract two registers, storing the difference in a third.
`addi` $RD, $RS, *value*	–add a value to a register, storing the sum in another.
`and` $RD, $RS1, $RS2,	–compute the bitwise AND of two registers into a third.
`andi` $RD, $RS, *value*	–compute the bitwise AND of a register and an immediate value.
`or` $RD, $RS1, $RS2,	–compute the bitwise AND of two registers into a third..
`ori` $RD, $RS, *value*	–compute the bitwise OR of a register and an immediate value.
`sll` $RD, $RS, 1	–shift one bit to the left, filling the least significant bit with 0.
`srl` $RD, $RS, 1	–shift one bit to the right, filling the most significant bit with 0.
`move` $RD, $RS	–copy a register's value to another.
`b` *label*	–jump to a labelled line.
`blt` $RS1, $RS2, *label*	–jump to a labelled line if one register's value is less than another.
`bltz` $RS, *label*	–jump to a labelled line if a register's value is less than zero.
`gt`, `le`, `ge`, `eq`, `ne`	–other conditions than `lt`

   The registers you can access are named $`v0-v1`, $`a0-a3`, $`t0-t9`, $`s0-s7`, $`sp`, $`fp`, and $`ra`.

4. **Write a Python function** `def listHasPairSum(aList,aSum)` that takes two parameters. The first parameter `aList` is a Python list of 0 or more integers. The second parameter `aSum` is an integer. The function should return `True` if any *pair of distinct indices* of `aList` hold values that sum to the value of `aSum`. It should return `False` otherwise.

   For example, if given the list `[10, 6, 1, 8, 0, 1]` and the sum value `14`, it should return `True` because element 1 and element 3 sum as $6 + 8 = 14$. If given that same list and the sum value `10`, it should return `True` because element 0 and element 4 sum as $10 + 0 = 10$. If given that same list and the sum value `2`, it should return `True` because element 2 and element 5 sum as $1 + 1 = 2$.

   With the same list `[10, 6, 1, 8, 0, 1]` it should return `False` if the sum value is 26 because no two elements of that list sum to 26. With the same list it should return `False` if the sum value is 20 because no two elements of that list sum to 20, even though the element 0 taken twice sums to 20.

5. Here, we build a digital circuit for the "carry out" bit of a 2-bit addition. That is, suppose we are adding two unsigned integers encoded as the bit pair $x_1 : x_0$ and the bit pair $y_1 : y_0$. Consider the logic for c, the carry bit that results from that addition. For example, if we are adding 3 to 2, that's the addition of `11` to `10`. The result is 5, i.e. `101`. Because the result's leftmost bit is 1, then the carry bit c is 1. If instead we are adding 1 to 2 (that is, `01` to `10`) the result is `011`. Because that result's leftmost bit is 0, the carry c is 0.

   (a) **Give the truth table** for the logic of c.

   (b) **Give the boolean expression** for c. It should use only the AND, OR, and NOT operations.

   (c) **Draw the circuit** for c. Label the gates as AND, OR, and NOT.

6. Pat likes to walk around with a bucket carrying their collection of red and white balls. Sometimes they'll stop on their walk to lay out the contents of their bucket in a line on the sidewalk.

   Write Python code for `def bucketLines(numWhite,numRed)` that outputs to the console all the possible line-ups of the balls in their bucket when there are `numWhite` white ones and `numRed` red ones. Each possible line-up should appear once. Depict a line-up with a series of `w` and `r` characters, for example

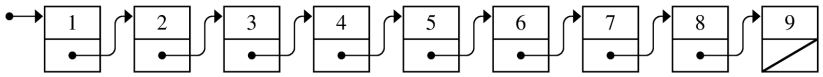
   ```
   wrr
   rwr
   rrw
   ```

   would be reasonable output for `bucketLines` when there is 1 white ball and there are 2 red balls in the bucket.

7. The following class is typical for defining linked list node objects:

```
class Node:
    def __init__(self, value, link=None):
        self.data = value
        self.next = link
```

Consider a representation of integer sequences where an empty sequence is just the Python value `None`. A non-empty sequence is represented by a `Node` object that holds the first item. The subsequent items in the sequence are reached via `next` links. For example, here is a picture of a linked list storing the sequence of integers from 1 to 9:



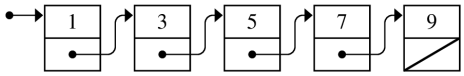
Write a Python function

```
def filterSequence(fn, oldSeq):
```

that takes a sequence `oldSeq` and returns a new sequence of its elements for which the predicate function `fn` returns `True`. The new sequence's values should be in the same order as those in `oldSeq`. **The structure of `oldSeq` must remain unchanged after the call.** For example, suppose that the variable `NUMBERS` refers to the head of the list depicted above and we call

```
filterSequence(lambda n: n % 2 == 1, NUMBERS)
```

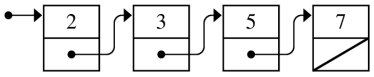
This returns the head node of the following linked list:



Similarly, if `isPrime` is a predicate checking whether a number is prime then

```
filterSequence(isPrime, NUMBERS)
```

should return the head node of this list:

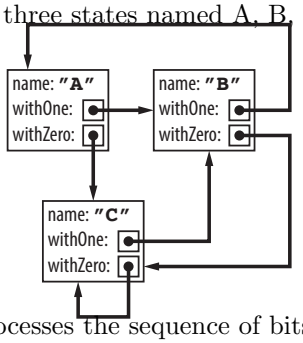


Your code should handle the case where the `oldSeq` is `None`, that is, is an empty sequence. Your code should also return `None` if no values in `oldSeq` make `fn` return `True`.

8. Let's use C++ to devise a state machine data structure. The data structure can be used to represent finite state machines that process sequences of bits. **Define a `State` struct** that contains the three attributes listed just below.

   - `name`: a string that names this state of the machine.
   - `withOne`: a pointer to a `State`. This gives the next state when a 1 is processed by a machine in this state.
   - `withZero`: a pointer to a `State`. This gives the next state when a 0 is processed by a machine in this state.

   A collection of such linked `State` structs defines a state machine. For example, consider the following diagram for a state machine that has three states named A, B, and C.



   If the machine is in state C and it processes the sequence of bits 0,1,1,1 it ends up in state B.

   (a) Consider the following snippet of C++ code that builds three `State` structs:

   ```
   State A {"A",nullptr,nullptr};
   State B {"B",nullptr,nullptr};
   State C {"C",nullptr,nullptr};
   ```

   Complete the code so that their `withOne` and `withZero` attributes mimic the structure of the diagram above.

   (b) **Write a C++ function** `process` that takes three parameters: a pointer to a `State` struct, a pointer to an integer array containing 0s and 1s, and a integer describing the length of that array. It should return the name of the state that the machine would end up in, having processed that array of bits in order. For example, if we feed `process` the address of `C`, an array of length four with the sequence 0,1,1,1, and the integer 4, then it would return the string `"B"`.

9. A "turtle" is an object that lives on a 2-D integer grid, sitting at a point represented by integer coordinates x and y, and moves around on this grid. A turtle faces one of four compass directions: north, south, east, or west. A turtle can either walk forward one unit, turn clockwise 90 degrees (e.g. switch from facing north to facing east), or turn counter-clockwise 90 degrees (e.g. switch from facing north to facing west).

   For each of these, give C++ code for describing the following two object classes that describe two kinds of turtle objects. This means that you should give the specification of the class (what would normally be in the header file) along with the code that implements the class methods.

   (a) Invent a class, named `Turtle`, that represents a turtle object. Choose the attributes as you see fit. Turtles are constructed at the origin, with x and y set to 0, facing north. In addition to a constructor, you should write four other methods: `forward`, `turnCW`, `turnCCW`, and `run`. The first three change the state of the turtle in the appropriate way. The last method `run` takes a string and applies a series of operations coded in that string. For example, the code `"+F--FF"` makes the turtle turn clockwise, move forward one unit, turn around (with two counter-clockwise turns) and then move forward two units.

   (b) **Devise a subclass of** `Turtle` named `StackTurtle` for a turtle that keeps track of a stack of locations on the grid. In addition to the methods of `Turtle` it has two additional methods:

   - `save`, which pushes the turtle's current location on its stack
   - `restore`, which pops the top location off the stack, then moves the turtle to that location

   For its `run` method, the character `'s'` within the string is used to save/push, and the character `'r'` within the string is used to restore/pop.

   You can use any standard sequence/container from the C++ libraries that you deem appropriate to represent the turtle's stack.

10. The Reed catalog lists the prerequisites for each course. CSCI 378 has this:

> Full course for one semester. This course is an introduction to deep neural architectures and their training. Beginning with the fundamentals of regression, optimization, and regularization, the course will then survey a variety of architectures and their associated applications. Students will develop projects that implement deep-learning systems to perform various tasks. *Prerequisites: MATH 201 and 202, and CSCI 221.*

But this doesn't give the whole story. Some prerequisites have prerequisites of their own. CSCI 221 has CSCI 121 as a prerequisite. Imagine developing Python code to help enumerate the entire set of courses needed before taking a given course. Assume the following Python class is defined:

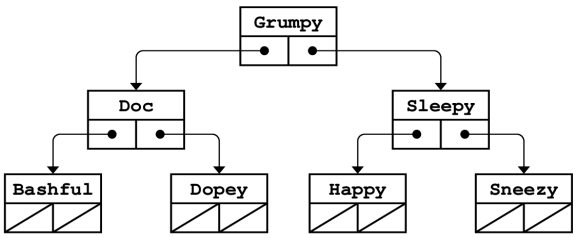
```
class Course:
    def __init__(self, id, prereqs):
        self.id = id
        self.prereqs = prereqs
```

where `id` is the numeric ID of the course, and `prereqs` is a list of `Course` instances that are its listed prerequisites in the Reed catalog.

Define a Python function `def allPrereqs(c)` that takes a `Course` instance `c` as its argument. It should return a Python list of all numeric course IDs that an incoming first-year student would need to take to qualify for the course. The list should have no repeats but can have any ordering. Here is an example of that function's use:

```
>>> math111 = Course(111, [])
>>> math112 = Course(112, [math111])
>>> math113 = Course(113, [])
>>> math201 = Course(201, [math112])
>>> math202 = Course(202, [math201])
>>> csci121 = Course(121, [])
>>> csci221 = Course(221, [csci121])
>>> csci378 = Course(378, [csci221, math202])
>>> csci382 = Course(382, [csci121, math112, math113])
>>> allPrereqs(csci378)
[111, 112, 121, 201, 202, 221]
>>> allPrereqs(csci382)
[111, 112, 113, 121]
```

11. A binary search tree's (a BST's) performance depends on how balanced it is laid out. For example, the BST pictured below has height 3 and is perfectly balanced with 7 nodes.



If an application knows all the keys ahead of time, it can build a balanced BST from a sorted list by choosing the middle element for the root then building the left and right subtrees recursively. The BST pictured above could be built from the Python list

```
DWARVES=['Bashful','Doc','Dopey','Grumpy','Happy','Sleepy','Sneezy']
```

When given $2^h - 1$ keys a perfectly balanced BST can be built with height h. For $h > 0$, its left subtree will be perfectly balanced and of height $h - 1$, as will its right subtree.

Write a Python function

```
def createBSTfromList(values):
```

that takes a Python list and returns a balanced BST storing the same elements. Here, an empty binary search tree is represented by the value `None`, and a non-empty binary search tree is an instance of the following class:

```
class BSTNode:
    def __init__(self, key, left=None, right=None):
        self.key = key
        self.left = left
        self.right = right
```

You can assume `values` is sorted (you can use $<$ and $>$ to compare its elements) and that it has length $2^h - 1$ for some integer $h \geq 0$. Your code should handle the case when `values` has a length of 0. In that case, `createBSTFromList` should return `None`, an empty BST.

12. In this problem you'll complete MIPS code that analyzes an array of *unsigned integers* stored in computer memory. The unsigned integers are 32 bits long and the start of the array is at a known address, also 32 bits long.

    The code should determine whether or not the array contains a *permutation* of the values from $\{0, \ldots, n-1\}$. Note that an array forms a permutation exactly when each value in the array is less than n, and when there are no duplicate values in the array. One strategy for checking this is to (STEP 1) first check that each of the array values are each less than n, and then (STEP 2) check that there are no duplicate values.

    The code assumes that the address of the array sits in register `$a0` and that the length of the array n sits in register `$a1`. When the code is done executing, register `$v0` should be set to `0` if the array is not a permutation, and to `-1` if the array is a permutation.

    Below is the uncompleted code. It performs STEP 1, but it is missing the code for STEP 2.

```
STEP1:
            move $t0, $a0               # we use $t0 to address the array
            li $t1, 0                   # we use $t1 to count what we've checked
loop1:
            beq $t1, $a1, STEP2
            lw t2, ($t0)                # checks whether an array value is less than n
            bge $t2, $a0, isntPerm      #
            addi $t0, $t0, 4
            addi $t1, $t1, 1
            b loop1
STEP2:
            ...your code goes here...   # code that checks for duplicates


isPerm:
            li $v0, -1
            b stop
isntPerm:
            li $v0, 0
stop:
```

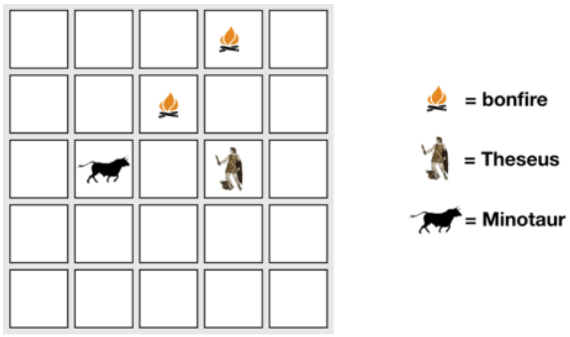
Junior Qualifying Exam

Sample Discrete Math Problems

Fall 2020, Computer Science Department, Reed College

Below are some problems we gave on quals in 2019-20. They are similar problems to what you'd expect on the MATH 113 part of the CS Junior Qualifying Exam.

1. With how many zeros does 200! end?
2. For this problem, consider working with the integers modulo 113.
   (a) **What integer** between 0 and 113 is a multiplicative inverse for 14?
   (b) Is this number unique for 14? If so, **explain why**. If not, **explain why not.**
3. Theseus and the Minotaur are in the following grid of rooms:



   Every minute, both Theseus and the Minotaur choose a direction (N, S, E, W) uniformly at random and move one room in that direction (if they hit the edge of the grid, then they stay in the same room). If Theseus and the Minotaur ever end up in the same room, then the Minotaur eats Theseus. Additionally, some rooms have bonfires. If Theseus is directly north, south, east, or west of a bonfire, then he can smell the smoke (note: it is fine to enter a room with a bonfire, and Theseus can smell smoke in those rooms as well).
   (a) **What is the probability** that Theseus is still alive (i.e. uneaten) after 1 move?
   (b) **What is the probability** that Theseus is still alive after 2 moves?
   (c) Suppose after the first move, Theseus is still alive, but cannot smell smoke. **What is the probability** that he will still be alive after his second move?
4. Say we have a sequence $a0, a1, \ldots$. The sequence is defined as having $a_0 = 1$ and for all other values $a_{n+1} = 3a_n + 2$. **Prove that** for any n we have $a_n = 2 \cdot 3^n - 1$.
5. Bob is creating strings of length 8 that consist of the letters A, B, and C. (E.g., one such string is ABCCABAA.) For each of the following constraints, say how many possible strings he can create.
   (a) There must be exactly two Bs, and they can't be adjacent to each other.
   (b) There must be three As, three Bs, and two Cs, in any order.
6. Alice and Bob are two students. One is a math major and one is a history major, but you have no information about which is which. You give them each a math problem to do, and you know that math majors get it right 80% of the time, while history majors get it right 40% of the time. Alice gets the problem right, but Bob does not.

   What is the probability that Alice is the math major?