CS106B Final, Spring 2023

You have 3 hours to complete this exam

This is a closed book, closed computer exam. You are allowed only 1 page (8.5x11, both sides) of notes, and no other aids (and you are not allowed magnifying devices other than regular glasses). You don't need to `#include` any libraries, and you needn't use assert to guard against any errors. Understand that the majority of points are awarded for concepts taught in CS106B, and not prior classes. You don't get many points for for-loop syntax, but you certainly get points for proper use of Stanford library containers, efficiency of your code, and solution of the problem.

You will be given a reference sheet. If you need extra space for scratch and/or problem responses, use extra scratch paper. We will not accept any work done on scratch paper.

NAME: ____

SUNet: ____
(e.g., cgregg)

I accept the letter and spirit of the honor code. I will neither give nor receive unauthorized aid on this exam.

signed ____

Problem 1, Recursive Backtracking (1 part): 20 pts

Problem 2, Classes (2 parts): 20 pts

Problem 3, Linked Lists (1 part): 20 pts

Problem 4, Trees (2 parts): 24 pts

Problem 5, Multiple Choice (3 parts): 9 points

Total: 93 points

Problem 1: Recursive Backtracking, 20pts

For this problem, write a recursive function,

```
void printTriplets(Vector<int>vec, int targetSum)
```

that prints all combinations of three numbers from a vector whose sum is less than or equal to the `targetSum`.

Notes:

- You should use a helper function for this problem.
- To print a vector, you can use `cout << vec << endl;`.
- The order of the output is not important, nor is the order of the values in each printed vector. If there are multiple vectors with the same values, only one should be printed (e.g., `{5, 1, 3}` is equivalent to `{1, 3, 5}` and `{3, 1, 5}`).

Example:

```
printTriplets({2, 7, 4, 9, 5, 1, 3}, 10);
```

Prints out:

```
{5, 1, 3}
{4, 1, 3}
{4, 5, 1}
{2, 1, 3}
{2, 5, 3}
{2, 5, 1}
{2, 4, 3}
{2, 4, 1}
{2, 7, 1}
```

Rest of this page intentionally left blank for scratch work.

Please write your answer to Problem 1 here.

```
void printTriplets(Vector<int> vec, int targetSum) {
```

Problem 2: Classes, 20 pts

Daniel Chao, one of our awesome SLs, chose to work on a queue for his capstone project in the CS major. He calls it the `CuteQ`, and it's meant to be used for office hours signups. It functions just as the Stanford `Queue` does, allowing for `enqueue` and `dequeue` and following a First In First Out (FIFO) ordering. We can think of the class interface for the `CuteQ` as follows:

```
class CuteQ {
public:
  CuteQ(int capacity);
  ~CuteQ();
  int size() const;
  void enqueue(string student);
  string dequeue();
private:
  void swapElements(int index1, int index2);
};
```

Your task is to write the complete `CuteQ` class, including the declaration of the private member variables and the implementation of the constructor, destructor, and member functions.

Design: A `CuteQ` holds `string` values representing students. The required internal representation is a dynamic array of `string`. We require that the oldest element (which will be removed upon the next call to `dequeue`) should sit at index 0 within the array.

Constructor: In the constructor, you are to:

- Create an empty `CuteQ` with the inputted `capacity` as the capacity of the internal array. The array should be initialized to hold empty strings.
- Initialize all private instance variables and allocate any necessary dynamic memory.
- Note: you may assume that the inputted `capacity` is sufficiently large such that you will never need to resize your internal array.

Destructor: The destructor performs any needed cleanup and deallocation.

Enqueue a string: `enqueue(student)` adds a student to the `CuteQ`. This `student` should be appended to the end of the dynamic array.

- Again, you can assume that we will never need to expand the size of the internal array.

Dequeue a string: `dequeue()` removes the first student in the `CuteQ`.

- If the client attempts to `dequeue` and the `CuteQ` is empty, you should raise an `error`.
- You should remove the element at the 0th index, since it was enqueued furthest ago.
- Once the 0th element has been removed, you should shift all elements to the right of that over by one. What was previously at index 1 should move to index 0, what was previously at index 2 should move to index 1, etc. You should include the necessary safeguards to ensure you're not accessing invalid memory.
- We've provided a `swapElements` function that you can use and assume works correctly. It takes in two indices and swaps the `string` values stored there within your internal array.

Below is a sample client use of `CuteQ`. The comments show the contents of the internal array after each line of code executes.

```
CuteQ cq(5);                    // [ "", "", "", "", "" ]
cout << cq.size() << endl;      // prints 0
cq.enqueue("Chris");            // [ "Chris", "", "", "", "" ]
cq.enqueue("Clinton");          // [ "Chris", "Clinton", "", "", "" ]
cq.enqueue("Daniel");           // [ "Chris", "Clinton", "Daniel", "", ""]
cout << cq.size() << endl;      // prints 3
cout << cq.dequeue() << endl;   // prints "Chris"
                                // [ "Clinton", "Daniel", "", "", "" ]
cout << cq.dequeue() << endl;   // prints "Clinton"
                                // [ "Daniel", "", "", "", ""]
```

2a

Complete the `CuteQ` class implementation by declaring the private member variables.

```
class CuteQ {
public:
  CuteQ(int capacity);                 // Constructor
  ~CuteQ();                            // Destructor
  int size() const;                    // Returns the number of students in the CuteQ
  void enqueue(std::string student);   // Adds a student to back of the CuteQ
  string dequeue();                    // Removes and returns the student at index 0
private:
  // Swap strings at index1 and index 2 within internal array
  void swapElements(int index1, int index2);

  // TODO: declare private member variables
```

2b

Write the full implementation of the `CuteQ` class.

```
CuteQ::CuteQ(int capacity) {









}

CuteQ::~CuteQ() {









}

int CuteQ::size() const {









}
```

```
}
```

Please write your answer to Problem 2b here.

```
void CuteQ::enqueue(string student) {











}

string CuteQ::dequeue() {











}
```

Problem 3: Linked Lists, 20 pts

Your task is to write the following function:

```
void moveToFront(Cell*& front, Cell* toMove);
```

This function takes in two `Cell*` parameters. The first is the front of a doubly-linked list. The second is a node in the doubly-linked list that is to be moved to the front. When the function closes, `front` should be updated to be pointing to `toMove`, and `toMove` should be moved to the front of the list. You should account for all necessary pointer rewiring. Some notes on this problem:

- You can assume both `front` and `toMove` are not `nullptr`
- `toMove` could be any node in the list
- Note that `front` is passed in by reference. That variable should be updated to point to the new front of the list after the function closes
- `Cell` is a doubly-linked node:

  ```
  struct Cell {
    int value;
    Cell* next;
    Cell* prev;
  };
  ```

Consider breaking this into two steps:

1. Slice out `toMove` from its current position in the linked list by performing the necessary rewirings
2. Shift `toMove` to the beginning of the linked list by making the necessary rewirings to `front`

Rest of this page intentionally left blank for scratch work

Please write your answer to Problem 3 here.

```
void moveToFront(Cell*& front, Cell* toMove) {
```

Problem 4: Trees, 24pts

Part 1: Highest product of siblings in a binary tree

Write the function, `int highestSiblingProduct(TreeNode *root)` that returns the highest product of any two siblings in a binary tree with at least two nodes in the tree. Siblings in a binary tree are two nodes that share a parent node.

Notes:

- There will be *at least* 2 nodes in the tree.
- Each node in the tree can have 0, 1, or 2 children.
- Nodes that are `nullptr` can be considered to have a data value of 0.
- If there is a node without a sibling, that sibling product is 0.
- You do not need, and are not allowed to use a helper function for this problem.
- You may use the `max(int a, int b)` function to determine the maximum of two values.

Example 1:

```
   4
 /   \
2     5

highest product: 10
```

Example 2:

```
    4
   / \
  9   2
 / \   \
3   5   7

highest product: 18
```

Example 3:

```
   8
 /
3

highest product: 0
```

Rest of this page intentionally left blank for scratch work.

Please write your answer to Problem 4.1 here.

```
int highestSiblingProduct(TreeNode *root) {
```

Part 2: Ternary tree: sum of all nodes

For this part of the question, assume the following definition for a `TernaryNode`:

```
struct TernaryNode {
    int data;
    TreeNode *left;
    TreeNode *middle;
    TreeNode *right;
};
```

A ternary tree is a tree where all nodes can have either 0, 1, 2, or 3 children (i.e., any of the children of a node could be `nullptr`).

Write the function, `int sumNodes(TernaryNode *root)` that returns the sum of all of the data fields in the nodes in a ternary tree, where `root` is the root node of the tree. You do not need, and are not allowed to use a helper function for this problem.

Example: the following tree has a node data count of 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 45:

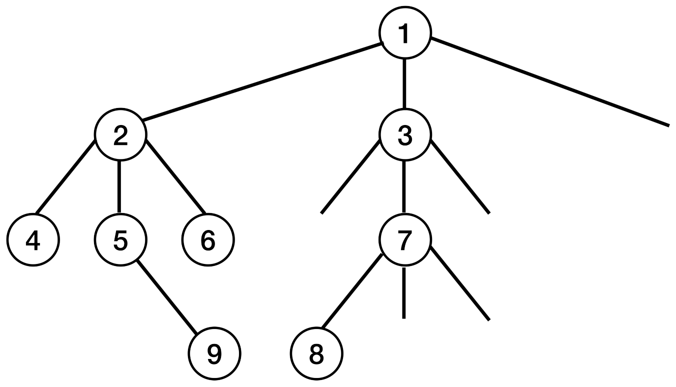

Figure 1:

Rest of this page intentionally left blank for scratch work.

Please write your answer to Problem 4.2 here.

```
int sumNodes(TernaryNode *root) {
```

Problem 5: Multiple Choice, 9 points

Note: There are three parts to this problem

5.1 Which of the following sorting algorithms would be the best one to use if you knew that a list of numbers was *mostly sorted already*?

A) Insertion Sort
B) Selection Sort
C) Quicksort with the pivot chosen as the first element each time.
D) Quicksort with the pivot chosen as the last element each time.

Answer to 5.1:

5.2 Which of the following characteristics is typically associated with a good hash function?

A) Collisions occur frequently, allowing for efficient distribution of data.
B) Hash values are generated using a random number generator, allowing for efficient distribution of data.
C) The output hash values are highly predictable and easily reversible.
D) Small changes in the input data produce significant changes in the resulting hash value.
E) Hash values are constant, regardless of the input data.

Answer to 5.2:

5.3 Given the graph below, the following table is a partial solution to Dijkstra's algorithm, starting from node A.

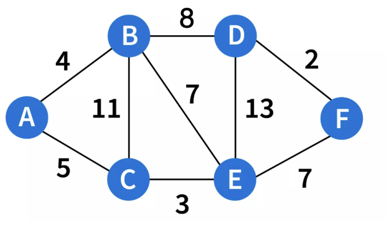

Figure 2: graph

	A	B	C	D	E	F
Distance to A	0	4	5	12	8	∞
From	-	A	A	B	C	
Visited?	Y	Y	Y	N	N	N

Which of the following is the *next step* in Dijkstra's algorithm?

		A	B	C	D	E	F
	Distance to A	0	4	5	21	8	15
A)	From	-	A	A	E	C	E
	Visited?	Y	Y	Y	N	Y	N

		A	B	C	D	E	F
	Distance to A	0	4	5	12	8	14
B)	From	-	A	A	B	C	D
	Visited?	Y	Y	Y	N	N	Y

		A	B	C	D	E	F
	Distance to A	0	4	5	12	8	14
C)	From	-	A	A	B	C	D
	Visited?	Y	Y	Y	Y	N	N

		A	B	C	D	E	F
	Distance to A	0	4	5	12	8	15
D)	From	-	A	A	B	C	E
	Visited?	Y	Y	Y	N	Y	N

Answer to 5.3: