EECE.3220 Spring 2017: Exam 3
Class Definitions

BST class definition for Question 1 (1b & 1c)

```
class BST {
public:
    BST();
    void add(int v);            // Add d to binary search tree
    void delete(int v);         // Delete v from tree
                                // You must write part of delete()
                                //  in Question 1c
    void print(ostream &os);    // Print tree contents to os
private:
    class BNode {
    public:
        int data;           // Data stored in node
        BNode *left;        // Left child
        BNode *right;       // Right child
    };

    BNode *root;            // Root of tree

    // Helper function for recursively printing tree contents
    void printtree(ostream &os, BNode *st);
};
```

Function definitions used in Question 1b:

```
void BST::print(ostream &os) {
    printtree(os, root);
    os << "\n";
}

void BST::printtree(ostream &os, BNode *st) {
    if (st == NULL)
        return;
    else {
        printtree(os, st->left);
        printtree(os, st->right);
        os << st->data << " ";
    }
}
```

EECE.3220 Spring 2017: Exam 3
Class Definitions

Heap class definition for Question 2c

```
class Heap {
public:
    Heap();                              // Default constructor
    void add(int v);                     // Add v to heap
    void percolate_up(unsigned pos);     // Percolate up value at
                                         //  position pos within
                                         //  array

    // Space to add additional functions
    //   that aren't necessary for this exam

private:
    int heaparr[1024];    // Actual heap data storage
    unsigned size;        // Number of values stored in heap
};
```

EECE.3220 Spring 2017: Exam 3
Class Definitions

HashTable class definition for Question 4b and 4c
```
template <typename T>
class HashTable {
public:
    HashTable();
    bool add(T v);
    void print(ostream &os);
private:
    T tab[10];          // Actual data storage
    bool free[10];      // free[i] = true if tab[i]
                        //   available to store data
};
```

Function definitions used in Question 4b:

```
template <typename T>
HashTable <T>::HashTable() {
    for (unsigned i = 0; i < 10; i++)
        free[i] = true;
}

template <typename T>
bool HashTable <T>::add(T v) {
    unsigned L = v % 10;
    while (L < 10 && free[L] == false)
        L++;

    if (L == 10)
        return false;
    else {
        tab[L] = v;
        free[L] = false;
        return true;
    }
}

template <typename T>
void HashTable <T>::print(ostream &os) {
    for (unsigned i = 0; i < 10; i++) {
        if (!free[i])
            os << tab[i] << "\n";
    }
}
```