FINAL EXAMINATION – DECEMBER, 2001
CSC 112
NICHOLAS R. HOWE

This is a closed-book exam. You may use two double-sided 8.5x11 sheets of notes.

All answers to this exam should be written in your exam booklet(s). Start with the questions that you know how to do, and try not to spend too long on any one question. Partial credit will be granted where appropriate. You will have two hours and twenty minutes. Good luck!

1. (10 points) Select the most appropriate data structure to help solve the following problems. Be as specific as possible; if you can supply a more specific name than a general data structure ("arithmetic expression tree" vs. "tree"), then you should do so.

a. An engineer wants to simulate train switching algorithms on a track with a single siding. Assume that cars may be added to and removed from the siding at only one end.

b. A UFO enthusiast wants to create a program to store information on UFO sightings by latitude and longitude. The program (which is to be made available over the world wide web) should allow anyone to quickly look up a particular latitude and longitude to see whether any sightings have occurred there. The locations are to be stored with great precision: any location, down to the arc minute, may be queried. However, only a few hundred sightings wil actually be stored.

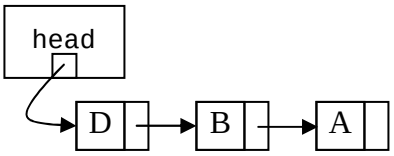
c. The state of Massachusetts wants to organize its jury duty records. State officials seek a program that will provide them with names of citizens eligible for jury duty. After serving on a jury, a citizen's name will be returned to the system, but should not come up on the eligibility list again until everybody else has taken a turn.

d. Quack, Inc. is writing a physician's assistant program to help doctors diagnose their patients. It asks a series of questions in an attempt to diagnose a patient. The sequence of questions asked will depend upon the answers to previous questions. The first question asked is always the same ("Are you feeling well?"), and the next question depends on whether the answer given is yes or no. What data structure can be used to efficiently hold the set of potential questions?

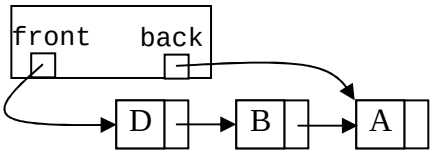
e. An online casino wants to be able to keep track of their customers by their social security numbers (SSN). At any moment, they want a program that can print out a list of current players sorted by SSN. In order to do this, the program should maintain a data structure keeping track of who is currently playing. Players should be able to join or leave at any moment.

2. (10 points) Draw a picture of the memory after the following operations, assuming the starting point is as pictured.

a. Stack. Operation: `push('C')`



b. Queue. Operation: `out()`. What value is returned?



3. (10 points) Complete the code for the following method. (Hint: Remember to account for the case where you add to an empty queue.)

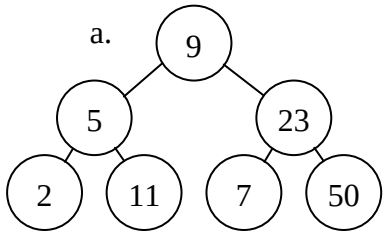
```
template <class T>
void Queue<T>::in(T new_datum) {
        // fill in code here
}
```

You may assume that the following declarations have already been made, and that the other methods have been defined elsewhere:

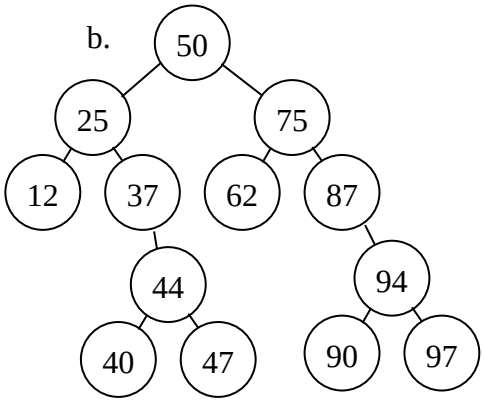
```
template <class T>
class Queue {
public:
        Queue<T>();
        Queue<T>(const Queue<T> &);
        ~Queue<T>();
        void in(T d);
        T out();
        bool isEmpty();
        bool isFull();
private:
        QueueItem<T> *front;
        QueueItem<T> *back;
};

template <class T>
class QueueItem {
public:
        QueueItem<T>(T d, QueueItem<T> *pn = NULL);
        QueueItem<T>(const QueueItem<T> &);
        ~QueueItem<T>();
        T getData();
        void setData(T d);
        QueueItem<T> *getNext();
        void setNext(QueueItem<T> *pn);
private:
        T data;
        QueueItem<T> *next;
};
```

4. (12 points) Which of the following represent valid binary search trees? Fix those that are not valid (making as few changes as possible). Assume that null leaves are not shown.

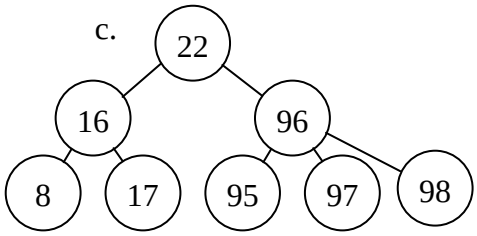
a.



b.



c.

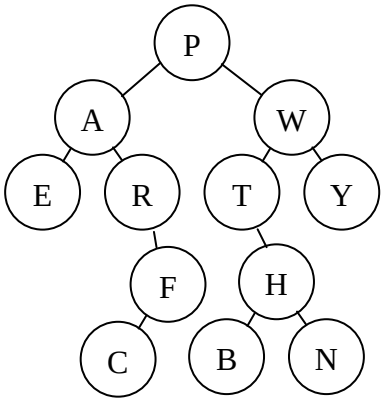


5. (15 points) Print out the string that would be created by traversing this tree in the following orders:

a. Preorder.

b. Inorder.

c. Postorder.



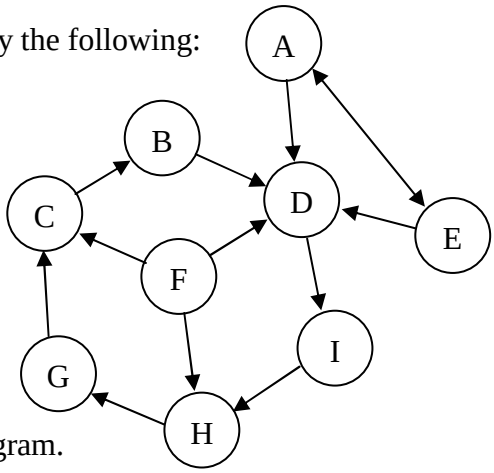
6. (10 points) Fill in code to print out a binary tree in postorder. Assume that the methods `isLeaf()`, `getData()`, `getLeftChild()`, and `getRightChild()` have already been defined.

```
Btree<T>::postorder() {
      // fill in code here.
}
```

7. (10 points) The Shorter Line Bus Company has three buses, that run on the routes described below. Draw a directed graph that represents this route structure.

Bus 1: Greenville to Haywood to Ipswich, then back to Greenville.
Bus 2: Ipswich to Jonesville and back.
Bus 3: Greenville to Klondike to Jonesville, then back to Klondike and then Greenville.

8. (8 points) Using the directed graph at right, identify the following:



   a. Neighbors of F.

   b. All cycles in the graph.

   c. Degree of G.

   d. Vertices reachable from A.

9. (15 points) Predict the output of the following program.

```
#include<iostream.h>
#include <string.h>

class Dog {
public:
	Dog(const char *n = "Rover") {
		strcpy(name,n);
		cout << "Hello " << name << "." << endl;
	};
	Dog(const Dog &orig) {
		strcpy(name,orig.name);
	};
	virtual ~Dog() {
		cout << "Goodbye " << name << "." << endl;
	};
	void describe() {
		cout << name << " is a dog." << endl;
	}
	virtual void bark() {
		cout << "Arf!  Arf!" << endl;
	};
	void bite() {
		cout << "The bark is worse than the bite." << endl;
	};
protected:
	char name[10];
};

class LittleDog : public Dog {
public:
	LittleDog(const char *n = "Fifi"):
	Dog(n)
	{
		cout << name << " is a little dog." << endl;
	}
	LittleDog(const LittleDog &orig):
	Dog(orig), lapsitter(orig.lapsitter)
	{
	};
	~LittleDog() {
		cout << "Little dogs live longer." << endl;
	}
	virtual void describe() {
		cout << name << "is a dog who ";
		if (lapsitter) {
			cout << "sits on laps." << endl;
		} else {
			cout << "acts like a cat." << endl;
		}
	}
	void bark() {
```

```
                cout << "Yip!  Yip!" << endl;
        };
        void bite() {
                cout << "This dog doesn't bite." << endl;
        };
private:
        bool lapsitter;
};

class BigDog : public Dog {
public:
        BigDog(const char *n = "Spike"):
        Dog(n)
        {
                cout << name << " is a big dog." << endl;
        }
        BigDog(const BigDog &orig):
        Dog(orig), catfriend(orig.catfriend)
        {
        };
        ~BigDog() {
                cout << "The bigger they are, the harder they fall." << endl;
        }
        virtual void describe() {
                cout << name << "is a dog who ";
                if (catfriend) {
                        cout << "likes cats." << endl;
                } else {
                        cout << "hates cats." << endl;
                }
        }
        void bark() {
                cout << "Woof!  Woof!" << endl;
        };
        void bite() {
                cout << "Ouch!" << endl;
        };
private:
        bool catfriend;
};

void main() {
        Dog* dogs[3];
        dogs[0] = new Dog("Spot");
        dogs[1] = new BigDog;
        dogs[2] = new LittleDog;

        dogs[0]->describe();
        dogs[1]->describe();
        dogs[2]->describe();
        dogs[0]->bark();
        dogs[1]->bark();
        dogs[2]->bark();
        dogs[0]->bite();
        dogs[1]->bite();
        dogs[2]->bite();

        for (int i = 0; i < 3; i++) {
                delete dogs[i];
        }
}
```