Question 1 (8 marks)

Suppose that somebody owes X dollars. At the end of every year the debt is recalculated as explained here:

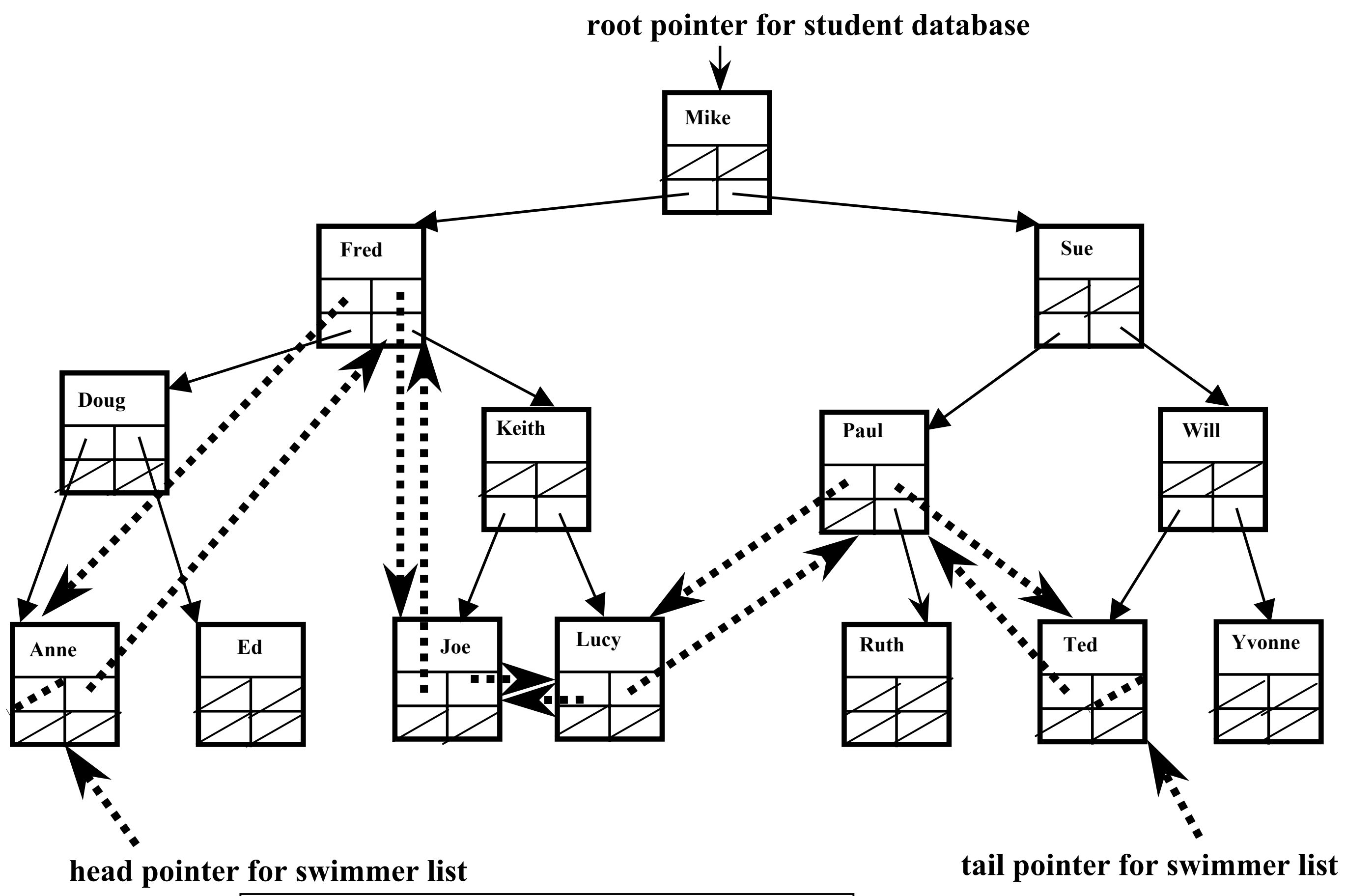
1. First the debt is increased by X * (R / 100), where R is the interest rate.
2. Secondly, the debt is then reduced by the annual payment of Y dollars.

Write a RECURSIVE function that, given X, R, and Y, returns the number of years it will take to pay off the debt. You may assume that the values are such that the debt will eventually get paid off.

Hint: How much debt remains after one year? How many years will it take to pay off this debt?

Question 2 (28 marks)

A student database is organized as a binary tree SORTED BY STUDENT NAME ("aardvark" < "betty" < "zirconda"). Some of the students in the database are on the university swim team. An auxiliary linked list is used to keep track of which students are on the swim team. The head pointer for this linked list points to the tree node for the first student on the swim team, and so on. The linked list is doubly linked, and there is also a tail pointer. The diagram below should make things clear.


root pointer for student database
Mike
Fred
Sue
Doug
Keith
Paul
Will
Anne
Ed
Joe
Lucy
Ruth
Ted
Yvonne
head pointer for swimmer list
tail pointer for swimmer list

Note: The students currently on the swim team are: Anne, Fred, Joe, Lucy, Paul and Ted.

The essential elements of the header for the database class are given on the next page. Note that elements that are not relevant to this question have been omitted.

```
class StudentDB {
private:
  class TNode {
  public:
    string name;
    int number;
    bool onSwimTeam;
    TNode *priorSwimmer, *nextSwimmer; // both NULL if student not on swim team
    TNode *left, *right;
  };

  TNode *root, // root pointer for tree
          *head, *tail;  // head and tail pointers for linked list of swimmers

  // searches the tree for the node containing the specified name and returns a pointer
  // to this node.  returns NULL if the name does not appear in the tree.
  TNode *findNode (const string &name) const;

  // detaches the node containing the last (i.e. highest) name from the tree and returns a
  // pointer to this node.
  TNode *detachLastNode (TNode *&root);

public:
  // removes the student from the swim team (but not from the database)
  void dropStudentFromSwimTeam (const string &name);

  // outputs the names of swim team members to "cout" (one per line)
  void outputSwimTeam () const;
};
```

For all parts of this question you may or may not use recursion, as you see fit. You may assume additional functions but, if you do, you must write them.

(a) Write private member method "findNode". Given a student's name, it returns a pointer to the node for the student (NULL if the student is not in the database). (6 marks)

(b) Write member method "dropStudentFromSwimTeam". You will find that method "findNode" is useful. Note that the "dropStudentFromSwimTeam" method does not remove the student from the student database. It only removes him/her from the swim team list. Have your function throw an exception if the student does not exist. (7 marks)

(c) We can easily print out the students on the swim team by traversing the linked list. However, if the linked list did **not** exist, it would still be possible to implement method "outputSwimTeam". Show how this could be done. (7 marks)

(d) Implement method "detachLastNode". The last node is the one containing the highest name. Note that the node is to be removed from the tree but **not** deleted – a pointer to the detached node is returned by this method. (7 marks)

(e) Explain how method "detachLastNode" might be useful in implementing another tree operation (1 mark).

Question 3 (8 marks)

Imagine that the police want to maintain a registry of stolen cars. Because fast lookups are very desirable, hashing is appropriate. The essence of a suitable class header is given below. Note that the header assumes hashing with buckets, i.e. data nodes include a "next in bucket" pointer. Struct "otherData" is assumed to be globally defined and to contain things like the date on which the car was stolen and so on. The details of this structure are not relevant to this question.

```
class HotCarRegistry {
private:

   class DNode {
   public:
      int licenseNumber;
      otherData data;
      DNode *nextInBucket;
   };

   // returns a value between 0 and tableSize – 1;
   int hash (int licenseNumber, int tableSize);

   DNode **table; // pointer to dynamically allocated hash table
   int tableSize, // size of hash table
       numberOfCars; // number of cars in registry

   HotCarRegistry (const HotCarRegistry &otherRegistry);
   HotCarRegistry &operator= (const HotCarRegistry &otherRegistry);

public:

   HotCarRegistry (int expectedNumberOfCars);
   ~HotCarRegistry;

   void insert (int licenseNumber, const otherData &data);

   // returns true if the license number is in the registry and false otherwise
   bool lookup (int licenseNumber);

};
```

Assuming that THE LINKED LISTS FOR EACH BUCKET ARE SORTED BY ASCENDING LICENSE NUMBER, write the implementation of method "insert". If the license number is already in the registry, your function should just update the information associated with it.

Question 4 (8 marks)

Here is the class definition for an elastic Queue implemented using a ring array:

```
template <class T>
class Queue {
private:
   T *ring; // dynamically allocated ring array
   int capacity, count, head, tail;

public:
   Queue();
   void enqueue (T item);  // adds the item to the queue, enlarging it if necessary
   T dequeue ();  // throws “overflow_error” if queue empty
   int size ();   // returns number of items in queue
   T look (); // as for dequeue but item is left in queue
};
```

Here is a partial implementation of enqueue:

```
template <class T>
void Queue<T>::enqueue (T item) {
   int i;
   T *temp;
   if (count == capacity) { // increase the capacity of the ring array by 10
      // this code has been omitted
   }
   ring[tail] = item; count++;
   tail = (tail + 1) % capacity;
}
```

The omitted code increases the capacity of the queue by 10. Basically it has to enlarge the array, somehow preserve the original data, and leave "head" and "tail" containing appropriate values. You are to write the omitted code. Hint: We only enlarge the ring array if it is full. What is the relationship between head and tail if the queue is full? Should this still be the case after the array is enlarged?

Question 5 (8 Marks)

Given the following grammar:

```
<name> ::= { a | b | c | d | …| z }+      // a name consists of one or more letters
<spaces> ::= { ‘ ’ }+  // a collection of one or more spaces
<item> ::= * <name> | <name>
<list> ::= <item> | <item> , <list>
<goal> ::= <name> <spaces> <list> ;  // *** the ; is part of the grammar ***
```

(a) Describe, using words and/or examples, the kinds of strings defined by this grammar. (2 marks)

(b) Give the parse tree for the following string: **int a, *b;** (2 marks)

(c) Assuming the usual "iterator" methods (iterator.get() and iterator.look()), and that somebody else has written "recognizeItem" and so on, write method "recognizeList". Have your function return true on success and false on failure. (4 marks)