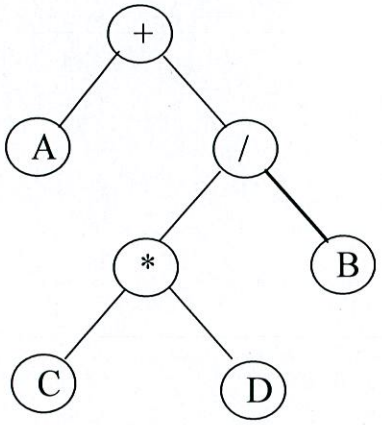
PhD Qualifying Examination
Fall 2011
Data Structures

Question II. (100 points) This question has 3 parts. The weight of each part is shown in brackets.

[40] a.) Given the **expression tree** shown on the right:

Write the expression in infix, prefix, and postfix notations.
What is the value of the expression if A=1, B=2, C=3, D=4?



[30] b.) Write a function (in C, C++ or Java) that given an expression tree builds the postfix notation on a stack of characters, named `s`. The basic operations: addition "+", subtraction "-", multiplication "*", and division "/" are supported. Assume operands are one character long, `s` is initially empty and has defined the normal stack operators, and the `expressionTree` has the usual operations of a binary tree.

```
ChStack *buildPostfixStack(BinaryTree *expressionTree, ChStack *s)
```

[30] c.) Write a function (in C, C++ or Java) that given a postfix expression stored in stack s in part b, prints out the corresponding fully-parenthesized infix expression. Do not worry about minimizing parenthesis. You will need to utilize other data structures. You may assume the normal types, operators and constructors for those data structures. Do not change the original postfix stack passed to the function.

```
void Postfix-to-Infix(ChStack *s)
```

Ph.D. Qualifying Examination

Fall 2011

Data Structures

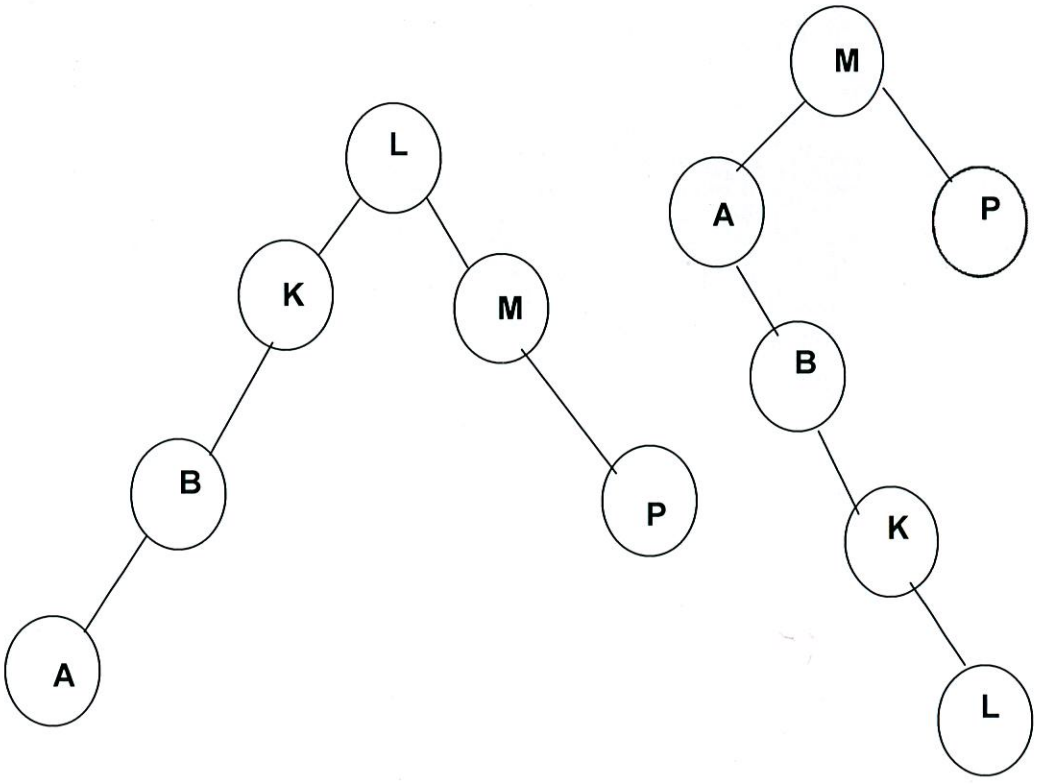
Question I (100 points)

You are given two binary search trees of strings. **Implement a Boolean function called "Equal"**, that determines if two binary search trees contain the same information. Considering the **Tnode** declaration below, two trees are equal if each tree contains the same label values. Assume that there are no duplicate strings in a single tree. The two binary search trees are sorted by the string field called "**label**". You may implement additional functions and/or data structures if needed. You may also use members of the Standard Template Library. However, you are not allowed to use the Binary Search Tree Template Class **BST**. Consider the following tree node declaration when implementing the function "**Equal**".

```
struct Tnode
{

     string label;
     Tnode *left, *right;
};
```

For example, the two trees below are equal.



Ph.D. Qualifying Examination

Fall 2010

Data Structures

Question I (100 points)

You are given two binary search trees of doubles. **Implement three functions, one called "Intersection", one called "Union" and the last one called "Total".** "**Intersection**" and "**Union**" will be used to print the nodes in the intersection and union of the set of numbers in two **binary search trees** between a particular range, given by "**Max**" and "**Min**", **inclusively.** "**Max**" and "**Min**" are doubles. Assume that "Max" is greater than or equal to "Min" and that there are no duplicate numbers in a single tree. The two binary search trees are sorted by the double field called "**price**". The function "**Total**" will return the sum of all the prices stored in a single binary tree. You may implement additional functions if needed. The final set of prices generated by the function Union **may not** contain duplicates. Consider the following tree node declaration when implementing the three functions.

```
struct Tnode
{
     double price;
     Tnode *left, *right;
};
```

Ph.D. Qualifying Examination

Fall 2010

Data Structures

Question II (100 points)

A well-formed parenthesized expression consists of matching '(' and ')' symbols.
For instance s_1= (( ) (( ))) and s_2= (( ) ( )) ( ) are well-formed, while
s_3= (( ) ( ))) (and s_4 = ( )) ( ) (( ) are not well-formed.

a) Write a C++ function

```
bool wellFormed(const string& s);
```

that returns *true* if the expression in string `s` is well-formed parenthesized and *false* otherwise. The function must use a character stack data structure (STL deque<char>).

b) Show the stack contents during execution of `wellFormed(s3)`, where string s3=(( ) ( ))) (

PhD Qualifying Examination

Spring 2010

Data Structures

Question II: (100 points)

Write a recursive function to convert a decimal number to any number base. For example, if the decimal number is 14 and the base is two, your function will print the number 1110. You may code your function either in C or C++ (or pseudocode). To be more specific, you are required to:

a) (50 points) Code your function with the following header syntax:

```
void convDecToAny(unsigned decimal, unsigned base);
```

b) (20 points) Show final output for the following call:

```
convDecToAny(2003, 8);
```

c) (30 points) Draw a stack to show conceptually the push and pop operations of your function for the following call:

```
convDecToAny(10, 2);
```

For simplicity in printing the final results, you may use printf with %d, if coded in C, or just cout, if coded in C++, to print your results.

Ph.D. Qualifying Examination
Spring 2009
Data Structures

Question II (100 points)

Write the code for a circular doubly linked list with a dummy node for the head. For traversal, the list uses begin() and next() functions that set the list's cursor to the first node and the next node respectively, and an isEnd() function that returns true if the cursor has advanced beyond the end of the list. The insert() and remove() functions insert a new node before the cursor, and remove the node at the cursor, respectively. To keep it simple, assume that each node holds a simple integer as its data. Class declarations for DblNode and DblList (in C++) are shown here.

```
class DblNode {
public:
      int ival;
      class DblNode* next;
      class DblNode* prior;
};

class DblList {
      class DblNode head;
      class DblNode* cursor;
public:
      DblList(void);
      bool isEmpty(void);
      int begin(void);
      bool isEnd(void);
      int insert(int value);
};
```

Implement the following 5 DblList functions (most require only a few lines of code):

a. The **default constructor** for the DblList class. The constructor takes no arguments. The effect of the constructor is to initialize the list and the dummy head for an empty list.
b. An **isEmpty()** function that takes no arguments. It returns true if the list is empty, else it returns false.
c. A **begin()** function that points the cursor at the first node in the list and returns its data value. If the list is empty, the cursor points at the head and returns the value 0.
d. An **isEnd()** function that returns true if the cursor is pointing beyond the end of the list, or the list is empty. Otherwise, it returns false.
e. An **insert()** function that takes an integer value for the new node's data. It puts a new node in the list in front of the current cursor location, and sets the cursor to point at the new node. If the list is empty, it inserts the node as the list's only node. This function returns the same integer value that was passed in.

Notes:
- Write your answers in C++ language. Your answers will be graded based on the correctness and the style of your implementation of the core functionalities.
- Be sure to properly comment your program; explain how the solution works.
- If you use a different language other than C++, please write a complete and working code that is functionally equivalent to the problem requirements.
- You code should work but you may not use native data structures (you must write your own). Read each description carefully. You must implement all of the details in each description.

Ph.D. Qualifying Examination
Spring 2009
Data Structures

Question I (100 points)

1. (40 points) Draw all possible binary search trees containing the four elements ("A", "B", "C", "D"). Each binary search tree should contain four nodes.

2. (60 points) Write a recursive function called "**Nodes_Between**" to count the number of nodes in a binary search tree between **key1** and **key2**, inclusive. Consider all possible cases and assume that **key1** is less than or equal to **key2**. Use the following function header and structure given below when answering this question.

```
struct tree_node
{
        string    item;
        tree_node  *l_child,
        tree_node  *r_child;
}
```

`*root` is a pointer to the root of the binary search tree.

```
int  Nodes_Between( tree_node * root , string & key1, string  & key2 )
```

Ph.D. Qualifying Examination

Spring 2008

Data Structures

Question II (100 Points)

1. [15 points] Fill in the blank for linked list:
   a. If a node has no successor, its link is set to a special _____ value
   b. In a linked list, insertion at a special place in the list is an O(_____) operation
   c. In a linked list, deletion at a special place in the list is an O(_____) operation
2. [30 points] Assume that the following declarations are used to process singly-linked lists:

```
Class Node
{
public:
        int data;
        Node * next;
};
Node  * p1, * p2, * p3;
```

Assume also that the following statements have been executed:

```
p1 =  new  Node;
p2 =  new Node;
p3 = new Node;
```

Tell what will be displayed by each of the code segments, or explain why an error occurs.

i.
```
p1-> data = 12;
p2->data = 34;
p1  = p2;
cout << p1->data << " " << p2->data << endl;
```

ii.
```
p1-> data = 12;
p2->data = 34;
*p1  = * p2;   //assume default copy operator, not overloaded
cout << p1->data << " " << p2->data << endl;
```

iii.
```
p1-> data = 123;
p2->data = 456;
p1->next  = p2;
p2->next = 0;
cout << p2->data << " " << p2->next->data << endl;
```

3. Consider a singly-linked list with integer values in its nodes. Assume each node has two parts:

   - *ptr->data* that stores an element of the list pointed by *ptr*
   - *ptr->next* refers to the location of the node containing the next list element

   If there is no element, then *Null-value* is used as special value

   Assume node **first** is created.

[30 points] Write a C/C++ program segment to determine whether the data items in a linked list with the first node pointed to by **first** are in ascending order. Please include proper error checking.

[25 points] Write a C/C++ program segment to insert a new node into a linked list with the first node pointed to by **first** after the *n*th node in this list for a given integer *n*. Please include proper error checking.

Ph.D. Qualifying Examination

Fall 2008

Data Structures

Question I (100 points)

Given a series of *n* daily price quotes for a stock, the *span* of the stock's price on a certain day is the maximum number of consecutive days up to the current day that the price of the stock has been less than or equal to its price on that day.

More formally, assume that price quotes begin at day 0 and that p_i denotes the price on day i. The span s_i is equal to the maximum integer k, such that $k <= i + 1$ and $p_j <= p_i$ for $j = i - k + 1, \ldots, i$. Given the prices $p_0, p_1, \ldots, p_{n-1}$, the stock span problem is to compute all the spans $s_0, s_1, \ldots, s_{n-1}$

Part A (50 points)
Design pseudo-code that uses dynamic allocated memory array(s) as the data structure(s) to solve the stock span problem.

Part B (50 points)
Design pseudo-code that uses dynamically allocated memory stack(s) as the data structure(s) to solve the stock span problem.

Ph.D. Qualifying Examination

Fall 2008

Data Structures

Question II (100 points)

Part A (20 points)

1. Describe 2 essential characteristics of a good hashing algorithm for a hash table.
2. Describe 2 alternative solutions to handle hash collisions in a hash table.

Part B (80 points)

The following is a pseudocode class declaration for a simple binary tree of integer values without duplicates.

```
Declare Class TreeOfIntegers
    *** The root node is a pointer to a NodeWithInteger object.    ***
    Pointer-to-NodeWithInteger root
    *** The insertInteger function takes an integer as an argument ***
    *** and returns a Boolean value as the result.                 ***
    *** It inserts an integer value in the tree if it is not       ***
    *** already there, and returns TRUE.                           ***
    *** It return FALSE if the value is already in the tree        ***
    boolean insertInteger ( integer value )
    *** The findInteger function takes an integer as an argument   ***
    *** and returns a Boolean value as the result.                 ***
    *** It searches for the value in the tree and returns TRUE if  ***
    *** the value exists in the tree, else it returns FALSE        ***
    boolean findInteger ( integer value )
    *** The insertInteger function takes an integer as an argument ***
    *** and returns a Boolean value as the result.                 ***
    *** It removes the value from the tree and then returns TRUE.  ***
    *** If the value to be removed was not found, it returns FALSE ***
    boolean removeInteger ( integer value )
end-of Declare Class
```

1. Write a declaration of the `NodeWithInteger` class and its class constructor with initialization in any computer language (or as readable pseudode). Note: a good constructor can simplify the answer to the next question.

2. Write the body of the `boolean insertInteger(int value)` function in any computer language (or a readable pseudocode). Keep the tree sorted. Do not worry about memory exceptions or about balancing the tree.

3. Write the body of the `boolean findInteger(int value)` function in any computer language (or readable pseudocode).

4. Write the body of the `boolean removeInteger(int value)` function in any computer language (or readable pseudocode).

Ph.D. Qualifying Examination

Spring 2007

Data Structures

Question I

(Part A - 20 points)

The Euler tour traversal of a binary tree T can be defined as a 'walk' around T, starting by going from the root toward its left child, viewing the edges of T as being walls that are always kept to the left. Each node of T is visited by the Euler tour:

On the left (before the Euler tour of a node's left subtree)
From below (between Eurler tours of a node's two subtrees)
On the right (after the Euler tour of a node's right subtree)

If the node is external, the visits all happen at the same time. We start an Euler tour by initializing a counter to zero, and then increment the counter each time we visit a node. To get the number of descendents of a node, we get the difference between the values of the counter when the node is visited on the left and when it is visited on the right and add one.

Exactly, how many times each node of T is encountered during an Euler tour?

(Part B - 40 points)

Define a *class or structure* (in C++ or C) for a generic Euler tour of a binary tree.

(Part C - 40 points)

Define a method/function (in C++ or C) for the Class or Structure defined in **Part B** that recursively traverses the binary tree and accumulates the results. This method must be part of the Class or Structure defined in **part B**.

Ph.D. Qualifying Examination

Spring 2007

Data Structures

Question (100 points)

Problem: Consider a singly linked list with integer values in its nodes.

(a) Write the C/C++ class/struct(s) that implement(s) such a linked list. [20 points]

(b) Construct an elegant and efficient recursive C/C++ function that takes such a linked list and returns the number of nodes with integer values that are even. [40 points]

(c) Redo part Question 1 part (b) but make your C/C++ function iterative. [40 points]

Notes:

- Make sure your solution is constructed clearly and idiomatically, so that it adheres to the commonly accepted definition of good coding style.
- Be sure to properly comment your program; explain how the solution works and why you selected particular algorithm(s) and data structure(s).

Ph.D. Qualifying Examination

Fall 2007

Data Structures

Question I

(Part A - 20 points)
Consider a file system tree *T*, where external nodes represent files and internal nodes represent directories.

Suppose we want to compute the disk space used by a directory, which is recursively given by the sum of: the sizes of the files in the directory and the space used by any sub-directories. (Assume that the size of the directory itself is computed as part of the disk space used by its parent.)

What kind of tree traversal should be used for the above disk space computation from the tree *T*? (Please provide the *name* of the traversal method **and** explain how the calculation itself would need to happen as the tree is traversed).

(Part B - 40 points)
Define a class or structure in (C++ or C) for a tree that can accommodate the above file system tree *T*.

(Part C - 40 points)
Define a method/function (in C++ or C) within the tree class/structure defined in **part B**, which implements the tree traversal and disk space calculation from **part A**. This method must be part of the class or associated with the structure defined in **part B**.

Ph.D. Qualifying Examination

Fall 2007

Data Structures

Question II (100 Points)

Consider a singly linked list with integer values in its nodes. Assume each node has two parts:

- *ptr->data* that stores an element of the list pointed by *ptr*
- *ptr->next* refers to the location of the node containing the next list element
- If there is no element, then *Null-value* is used as special value

Using the notations given above, answer the following questions:

1. [40 points] Write a program segment to reverse a linked list with the first node pointed to by *first*. Do not copy the list elements; rather, reset links and pointers to that first points to the last node and all links between nodes are reversed.
2. [60 points] The *shuffle –merge* of two lists $x_1, x_2, \ldots, x_n$ and $y_1, y_2, .., y_m$ is the list

   $z = x_1, y_1, x_2, y_2, \ldots, x_n, y_n, y_{n+1}, \ldots, y_m$ if $n < m$,

   $z = x_1, y_1, x_2, y_2, .., x_m, y_m, x_{m+1}, .., x_n$ if $n > m$

   or

   $z = x_1, y_1, x_2, y_2, ..., x_n, y_n$ if $n = m$

   Write a program segment to shuffle-merge two linked lists with their first nodes pointed to by *first1* and *first2*, respectively.

   Note: Do not copy the items. Just change links in the two lists (thus destroying the original lists) to produce the merged list.

Notes:

- Write your answer in C language. Your answers will be graded based on the correctness and style of your implementation of the core functionalities.
- Make sure your solution is constructed clearly and idiomatically, so that it adheres to the commonly accepted definition of good coding style.
- Be sure to properly comment your program; explain how the solution works.

Ph.D. Qualifying Examination

Spring 2006

Data Structures

Question I (100 points)

Define a pointer-based linked list, class and corresponding source code for operations and methods using C++ that stores the current date in the following format: February 23 2006. Basic operations and methods must include a default constructor, an explicit-value constructor, a copy constructor, a destructor, and a basic ADT's list method for empty list checks.

The class must be defined as a *SINGLE* linked list only. Use of arrays, counters, vectors, stacks, queues, or C++ built-in classes, etc. is not acceptable.

Ph.D. Qualifying Examination

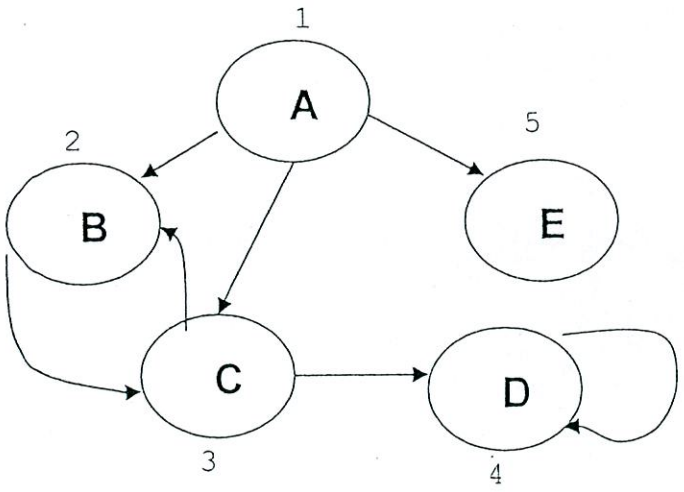
Spring 2006

Data Structures

Question II (100 points)

1. [50 points] Consider a binary tree with integer values in its nodes.
(a) Write the class/struct(s) that implement(s) a binary tree.
(b) Write a recursive function that takes such a binary tree and returns the number of nodes with even integer values.

2. [50 points] Describe two different ways in which a directed graph can be implemented as a data structure. Compare and contrast the advantages and disadvantages of each. Using the following graph as an example, show how it can be represented by the two different data structures in your answer.



Ph.D. Qualifying Examination

Fall 2006

Data Structures

Question III (100 points)

Background:

A **Heap** is a binary tree with the following properties: (1) It is *complete*; that is, each level of the tree is completely filled, except possibly the bottom level, and in this level, the notes are in the leftmost position; (2) It satisfies the *heap-order* property: the data item stored in each node is greater than or equal to the data items stored in its children. If the data items are records, then some key field in these records must satisfy this condition.

1. [30 points] If an *array* is used to implement a **heap**, answer the following:
   a. List the location of the children of the *i*th node (at location *i*);
   b. List the location of the parent of the *i*th node.

2. [70 points] Consider a heap of integers. Given the following declarations to support the **heap** implementation:

```
struct HeapType
{
 void PercolateDown(int root, int bottom);
 void PercolateUp(int root, int bottom);
 int * elements; // Array to be allocated dynamically
 int numElements;
 };
```

Write a complete code in C/C++ to implement function *PercolateDown (int root, int bottom)*. The function should satisfy the following conditions:

Function: Restores the order property of heaps to the tree between root and bottom
Precondition: The order property of heaps may be violated only by the root node of the tree.
Postcondition: The order property applies to all elements of the heap.

Ph.D. Qualifying Examination

Spring 2005

Data Structures

Question I (100 points)

Using a pointer-based Linked List (no arrays, vectors, stacks, or queues, etc.), write a program using C or C++ that allows the manual entry of positive or negative integers (one per line, must hit <enter> key after each number). The current entered number must be placed in the right location within the existing List before accepting another number entry without destroying the List every time. The data entry operation stops when a special character is detected, e.g. the letter Q for "quit".

Note: The Linked List is ALWAYS in a sorted order. Before and after the entry of a number, the Linked List is ALREADY sorted in ascending or descending order, without destroying and rebuilding it after each new entry. The program must use a *SINGLE* Linked List only. Use of arrays, counters, vectors, stacks, queues, etc. is not acceptable.

Ph.D. Qualifying Examination
Spring 2005
Data Structures

Question II. (100 points)
This question is about operations on a binary tree. For this question every node of the binary tree holds an integer value. For any node, all values in nodes of the left subtree are less than the value of the parent node. All values in nodes of the right subtree are greater than the value in the parent node. Duplicates (two different nodes holding the same value) are not allowed.. You are to write two functions that perform operations on a sorted binary tree data structure: "measure" and "delete." In both cases you may write the function in any form – recursive, non-recursive, as a single function, broken into sub-functions, etc. But it must work. You may use C++ or C++-like pseudo-code. A C++ prototype for the node and tree classes are provided. Suggested prototypes for the Tree and TreeNode classes are shown below. You may assume the existence of insertNode and findNode methods on a TreeNode. You may add additional methods to either class, but you must implement any method that you add.

```
class TreeNode {
private:
    integer value;
    TreeNode* left;
    TreeNode* right;
public:
    void insertNode(TreeNode* node);
    TreeNode* findNode(int search_value);
    // add methods here if you wish
};

class Tree {
private:
    TreeNode* root;
public:
    // Implement these two functions
    int measure(int* height);
    bool delete(int omit_value);
};
```

1. (50 points) Write a function that computes the size and height of the binary tree. The size of the tree (number of nodes) should be returned as the result of the function. The depth should be assigned to its one passed argument.

2. (50 points) Write a function to find and delete a node from the binary tree. In this function, should return true if a node with the given value was found and deleted. The function should return false if no node was found that matched the given value. The delete operation must free the memory used by the deleted node, and it must preserve the property that the binary tree is sorted. DO NOT WORRY ABOUT BALANCE OR BALANCING.

Ph.D. Qualifying Examination

Fall 2005

Data Structures

Question II (100 points)

Part 1: (55 points) Write the C++ implementation of a linked list that is singly linked and has only a head pointer and a current pointer. A prototype for the LinkedList class is given below, including a nested Node class. Write the code the 5 methods shown in bold, below, for the LinkedList class. Implementations for the constructor, isEmpty(), and begin() methods are given.

```
class LinkedList {
      class Node {
            Node* next_node;
            void* data;
      public:
            Node(void* ptr_data) {data = ptr_data; next_node = NULL;}
      };
      Node* head_node;
      Node* current_node;
public:
      LinkedList() { head_node=NULL; current_node=NULL; };
      Bool isEmpty() { return NULL==head_node; };
      // Method: void* begin()
      // returns a NULL pointer if the list is empty.
      // makes the head node the current node and returns its data pointer
      void* begin();
      // Method: void* advance();                                    // 10 pts
      // returns a NULL pointer if the list is empty or the current node is the tail
      // makes the node after the current node the current node and returns its data pointer
      void* advance();
      // Method: insertHead(void*);                                  // 8 pts
      // adds a new node as the head node and makes the new node the current node
      void insertHead(void* ptr_data);
      // Method: insertNext( void* );                                // 10 pts
      // if the list is empty, adds a new node as the head node
      // else adds a new node immediately after the current node
      // in all cases, the new node becomes the current node
      void insertNext(void* ptr_data);
      // Method: insertTail( void* );                                // 12 pts
      // adds a new node as the tail node and makes the new node the current node
      void insertTail(void* ptr_data);
      // Method: remove();                                           // 15 pts
      // if the list is empty, does nothing
      // deletes the current node and makes the node after it become the current node
      // if deleted node was the last node, the new last node becomes the current node
      void remove();
};
```

Node* Node* ptr_data

```
void* LinkedList::begin() {
  if( isEmpty() )
     return NULL;
  else {
     current_node = head_node;
     return current_node->data;
  }
};
```

Part 2: (45 points) For each of the 3 choices below, describe the relative advantages and disadvantages of the two alternatives, in terms of

a. usage (i.e., application suitability and relative efficiency of specific operations), and
b. presence or absence of special cases to be handled in the implementation

1. doubly linked lists vs. singly linked lists
2. circular linked lists vs. linear linked lists
3. using a dummy node vs. a pointer for the head of the list

Ph.D. Qualifying Examination

Fall 2005

Data Structures

Question I (100 points)

a) The median of a set of numbers is the middle number after the set of numbers have been sorted in increasing order. For an even number of points, the average of the two middle numbers is taken. Write a function template in C++ that finds the median of 3 numbers (must handle any type of number, ie integer, floating point, etc). 20 points

b) Assume the following declarations:

```
vector<double> xValue;
vector<int> number(5,1);
```

Describe the values in vector<T> after the statements are executed, each is worth 20 points:

a)
```
for (int i = 0; i <= 4; i++)
        xValue.push_back(double( i ) / 2.0);
```

b)
```
for (int i = 0; i < 5; i++ )
        if ( i % 2 == 0)
                number.push_back(2 * i);
        else
                number.push_back(2 * i + 1) ;
```

c)
```
for (i = 1; i < 5; i++)
        number.push_back(2 * number[i – 1]);
```

d)
```
for (int i = 1; i <= 3; i++)
        number.pop_back();
        for ( int i = 1; i <= 3; i++)
                number.push_back(2);
```