FINAL EXAMINATION ANSWER KEY– DECEMBER, 2003
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!

Data Structure Selection

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 ("binary search tree" vs. "tree"), then you should do so.

a. You want to write a program that will store and compute the value of boolean expressions using the binary boolean operators OR and AND, such as ((*P* AND *Q*) OR (*P* AND *R*)).

Binary tree (or binary expression tree)

b. An operating system must keep track of the set of active processes, which are stored in no particular order. From time to time, it must check each of the processes to update its state. Sometimes a process will terminate, in which case its record will be eliminated. Other times a process will split in two, in which case an identical record must be created beside the first.

List (hash table also accepted)

c. A *reversing buffer* works as follows: a sequence of characters is stored one at a time in the buffer. When finished, the characters may be retrieved from the buffer in the opposite order from which they were stored.

Stack

d. Internet routers receive incoming packets and quickly forward them on to their destinations. If they get a heavy burst of incoming packets, they temporarily store them and process them in the order in which they were received.

Queue

e. A geographic system must efficiently locate data associated with two-dimensional coordinate keys (i.e., the location where the measurement was taken).

R-Tree (hash table also accepted)

2. (10 points) Write a short (several paragraph) response to each of the questions below.

a. Discuss *encapsulation*, giving examples of mechanisms within C++ that help to achieve it. Explain how encapsulation helps make a class more abstract, using a specific example.

Programs may be viewed at a number of levels, ranging from a high-level view covering the broad program concepts, to a low-level view concerned with the details of the implementation. Encapsulation refers to the hiding of low-level details so that they do not interfere with the high-level concepts. C++ encourages encapsulation by allowing classes to declare certain properties and methods to be private, meaning that they are effectively hidden from any program using the class. The program then needs to be concerned only with the class's public interface, which can be written at a higher conceptual level. As a specific example, consider the implementation of a class for handling cash values. The class may store the cash value in a number of ways: as a float, as an integer number of pennies, or as two integers representing dollars and cents. With encapsulation, it makes no difference to the program using the class which of these choices is actually used, because the public interface is abstract enough to apply to any of the three.

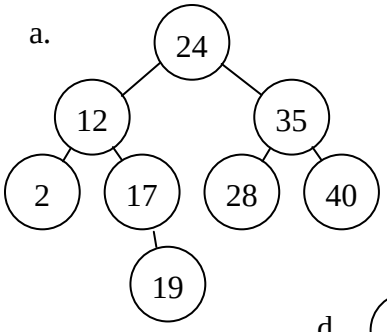
b. Discuss the advantages and disadvantages of using exceptions, in comparison with other methods of handling errors.

Exceptions are a means by which a program can signal an error and allow a different part of the program to handle it appropriately. They are particularly well-suited for writing encapsulated classes, since the class can handle the error detection but leave the choice of how to handle the error up to the main program. Separating the detection from the handling can be a disadvantage in some cases. If the main program neglects to provide proper error-handling code, then the when an error is detected the program will have no choice but to terminate.

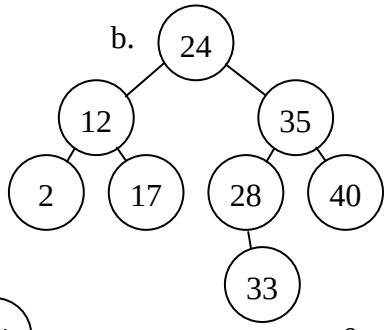
Trees

3. (10 points) Given the binary search tree shown below, draw the data structure that would result from the following operations. (For each part of the problem, assume that you start with the original tree.)

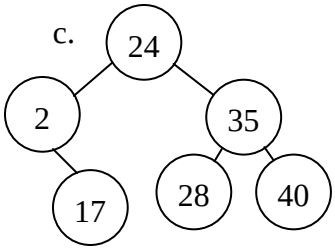
a.



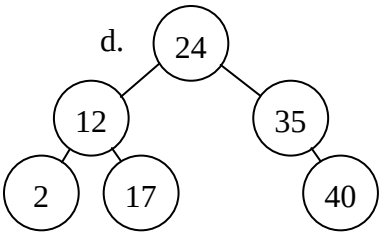
b.



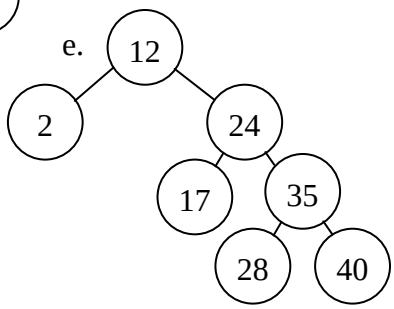
c.



d.



e.



Heaps

4. (12 points) The diagram below shows the contents of a heap at a particular point in time, stored within an array as discussed in class. Show the result of each operation listed below, by giving the value returned (if any) and drawing the state of the data structure after the operation is complete. Assume that each operation is applied successively, so that part (b) uses the result of part (a), etc. (Hint: It may help to draw the structure as a tree so you can see what is happening. However, in your answer you must show the data as an array.)

		60	48	57	3	41	50			
a. *Insert(77)*		77	48	60	3	41	50	57		
b. *Insert(5)*		77	48	60	5	41	50	57	3	
c. *RemoveRoot()*		60	48	57	5	41	50	3			(Returns 77)
d. *RemoveRoot()*		57	48	50	5	41	3				(Returns 60)
e. *Insert(26)*		57	48	50	5	41	3	26		

STL

5. (12 points) The program below was written using the `DL_IntList` class developed for this course. Rewrite it so that it uses the templated `list` class and iterators from the Standard Template Library to accomplish the same effect.

```
#include <list>
#include <iostream>
using namespace std;

list<int>::iterator minpos(list<int> &ls) {
  list<int>::iterator result = ls.begin();
  list<int>::iterator item;
  for (item = ls.begin(); item != ls.end(); item++) {
    if (*result > *item) {
      result = item;
    }
  }
  return result;
}

int main() {
  list<int> ls2;

  ls2.push_back(5);
  ls2.push_back(3);
  ls2.push_back(-2);
  ls2.push_back(0);
  ls2.push_back(-2);
  cout << *minpos(ls2) << endl;
}
```

Graphs

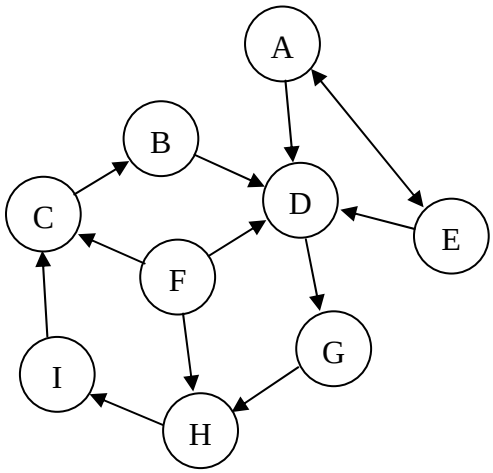
6. (12 points) Identify the order in which vertices would be visited for the specified traversal of the **directed** graph below. Also identify any vertices that are unreachable from the starting vertex. Where there is a choice of more than one vertex, choose the one that is first in alphabetical order.

a. Breadth-first traversal starting at F.

F, C, D, H, B, G, I.
Not accessible: A, E.

b. Depth-first traversal starting at A.

A, D, G, H, I, C, B, E.
Not accessible: F.

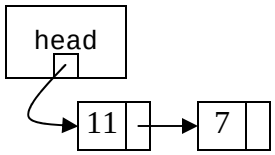


Stacks and Queues

7. (12 points) For the following sequence of operations, give the result returned by each `pop()` and `out()` operation, in order, and draw the final state of the resulting data structure with links shown. Assume that both stacks and queues are implemented as described in class, and that the data structures are initially empty.

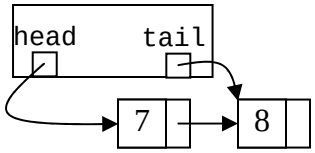
a. Stack: push(7); push(6); pop(); push(4); pop(); push(11); push(7); push(8); pop(); pop().

Output: 6, 4, 8, 7. Status:



b. Queue: in(7); in(6); out(); in(4); out(); in(11); in(7); in(8); out (); out ().

Output: 7, 6, 4, 11. Status:



Recursion

8. (12 points) Consider the following code. What would the output be?

```
#include <iostream>
using namespace std;

int recursive(int x) {
  cout << x << ", ";
  if (x%2 == 0) {
    return
recursive(x/2)*recursive(x/2);
  } else {
    return x;
  }
}

int gcd(int x, int y) {
  cout << "(" << x << "," << y
<< "), ";
  if (x == y) {
    return x;
  } else if (x < y) {
    return gcd(x,y-x);
  } else {
    return gcd(x-y,y);
  }
}
```

```
int main() {
  cout << "Part a:  ";
  int x = recursive(12);
  cout << x << endl;
  cout << "Part b:  ";
  int y = gcd(12,40);
  cout << y << endl;
}
```

Output:

Part a: 12, 6, 3, 3, 6, 3, 3, 81
Part b: (12,40), (12,28), (12,16), (12,4), (8,4), (4,4), 4

Lists

9. (10 points) A singly-linked list of integers can be implemented using two arrays. The values of the two arrays at a particular index ***i*** describe one of the elements in the linked list, as follows: The first array (***data***) holds the integer stored at that element of the list. The second array (***next***) holds the index ***j*** where the next element of the list may be found. A separate variable ***start*** gives the index of the first element in the list, and a negative index indicates the end of the list. For example, the arrays shown below correspond to the list 3, 1, 4, 1, 5, 9:

start | 4 | ***empty*** | 6 |

data	5	1	1	9	3	4	0	0	0	0

next	3	0	5	-1	2	1	7	8	9	-1

Empty spaces in the array are also linked together in a stack-like structure so that they may be found easily. A variable called empty gives the index of the first empty space. When an element is inserted into the list, the first space from the empty list is used. When an element is deleted, its index is added to the front of the empty list.

a. Draw the state of the data structure described above if the number 7 is inserted into the list following the number 4.

start | 4 | ***empty*** | ***7*** |

data	5	1	1	9	3	4	***7***	0	0	0

next	3	0	5	-1	2	***6***	***1***	8	9	-1

b. Draw the state of the data structure if the number 5 is deleted from the original list (i.e., the list before part (a) above).

start | 4 | ***empty*** | ***0*** |

data	5	1	1	9	3	4	0	0	0	0

next	***6***	***3***	5	-1	2	1	7	8	9	-1

c. Comment on the advantages and disadvantages of this implementation of lists, compared with that used previously in class.

The maximum size of the list is limited. On the other hand, some of the overhead of memory allocation and deallocation is avoided.