FINAL EXAMINATION – DECEMBER, 2002
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 ("arithmetic expression tree" vs. "tree"), then you should do so.

a. Application must insert and delete quickly, and must be able to produce a sorted list quickly.

b. Application must insert, delete, and look up extremely quickly, but doesn't ever need to produce a sorted list.

c. Expert systems work by asking a series of questions in an attempt to diagnose a situation. The questions asked depend upon the answers to the previous questions. What data structure would you use if to store the questions for an expert system that asks only yes/no questions?

d. If outgoing e-mail must be stored temorarily by a mail server before transmission, what data structure should be used to store it?

e. Word processors often store the sequence of edits made to a file, so that a user can undo them one by one if she chooses. What data structure might be most likely used to store the edit history?

C++ Language

2. (12 points) Why are redundancy and repetition in computer code a bad thing? Write a few paragraphs on this subject, giving at least three examples of mechanisms in C++ that are designed to help avoid unnecessary repetition of code.

Stacks and Queues

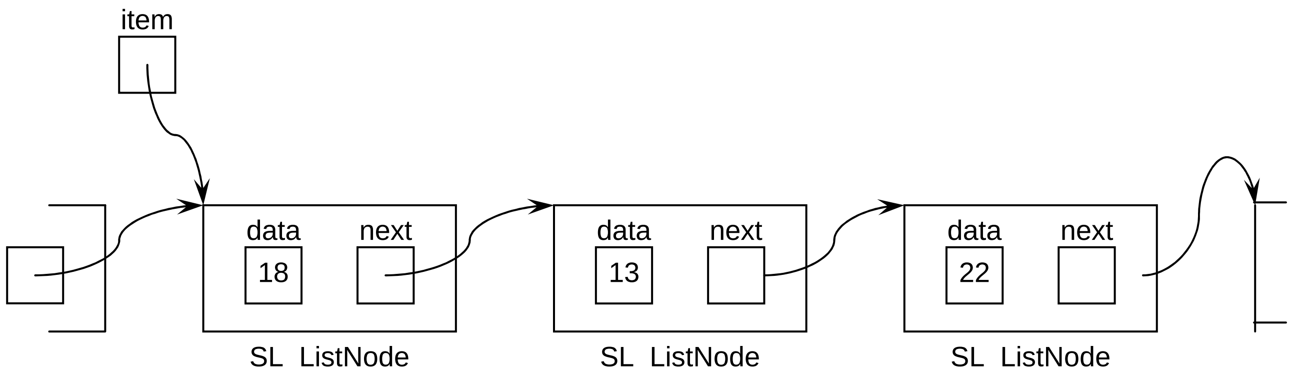
3. (16 points) For the following sequence of operations, give the result returned by each `pop()` and `out()` operation, in order, and draw the final state of the data structure specified. Assume that both stacks and queues are implemented as described in class, and that the data structures are initially empty.

a. Stack: push(5); push(8); push(4); pop(); push(11); pop(); push(7); push(8); pop(); pop();

b. Queue: in(5); in(8); in(4); out(); in(11); out (); in(7); in(8); out (); out();

Lists

4. (8 points) Assume that the diagram below represents a portion of a singly linked list. The variable `item` is a pointer to one of the nodes in the list. Write fragments of code for the following operations, assuming the naming conventions given in the diagram. You may assume for purposes of this problem that all the properties are declared public; i.e., they may be accessed directly by name without using accessors or manipulators. Make sure that your code updates all pointers as necessary to leave the list in a consistent state, and allocates/deallocates memory properly. (You need not put your code fragments inside functions; just list the statements required to accomplish the specified tasks.)

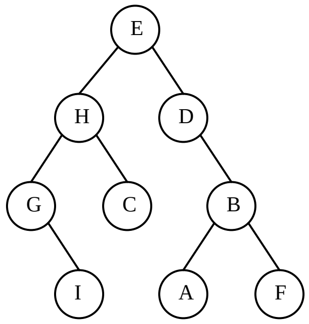


a. Delete the node containing 13.

b. Insert a node containing 8 immediately following the node containing 18.

Trees

5. (12 points) In what order would the nodes be visited for the tree at right for the following traversal methods?



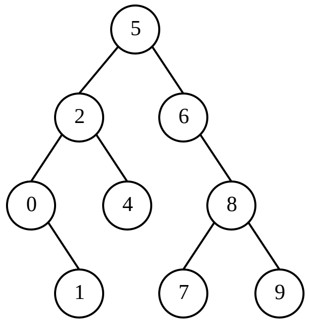
a. Inorder traversal

b. Breadth first traversal

c. Preorder traversal

d. Postorder traversal

6. (10 points) For the binary search tree at right, indicate the tree that would result from each of the following operation. You should assume that each part starts with the tree pictured. (In other words, the changes are not cumulative.)



a. Insert 3.

b. Delete 2, using right merge if necessary.

c. Delete 6, using left merge if necessary.

d. Delete 5, using right copy if necessary.

e. Delete 5, using left copy if necessary.

Hash Tables

7. (8 points) Suppose that a particular application uses a hash table of size five, with linear probing. The hash function is simple modular hashing. Draw the contents of the table after the following sequence of insertions:

Key: 179; **Data**: Alan Turing
Key: 381; **Data**: John Von Neumann
Key: 440; **Data**: Ada Lovelace
Key: 381; **Data**: Charles Babbage
Key: 514; **Data**: Grace Hopper

0		
1		
2		
3		
4		

Recursion

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

```
#include <iostream>

int recursive(int x) {
  cout << x << ", ";
  if (x <= 1) {
    return 0;
  } else if (x%2 == 1) {
    return recursive(x-1);
  } else {
    return recursive(x/2)+1;
  }
}

int fib(int n) {
  cout << n << ", ";
  if (n <= 1) {
    return 1;
  } else {
    return (fib(n-1)+fib(n-2));
  }
}

void main() {
  cout << "Part a:  ";
  int x = recursive(13);
  cout << x << endl;
  cout << "Part b:  ";
  int y = fib(7);
  cout << y << endl;
}
```

Inheritance

9. (12 points) Consider the following definition for class `Shape`, designed to keep track of the details of geometric shapes for display on the screen. You are to design a new class called `Circle` derived from `Shape`. It will have one new property, `radius`, plus appropriate constructors, accessors and manipulators.

```
enum Color = {Black, White, Red, Yellow, Orange, Green, Blue, Purple};

class Shape {
public:
  Shape(float x = 0, float y = 0, Color body = Black, Color border = Black);
  Shape(const Shape &);
  virtual ~Shape();

  // accessors:
  float getCenterX() const;
  float getCenterY() const;
  Color getBodyColor() const;
  Color getBorderColor() const;

  // manipulators:
  void setCenterX(float);
  void setCenterY(float);
  void setBodyColor(Color);
  void setBorderColor(Color);

  // other methods:
  virtual float area() const;       // computes shape's area
  virtual float perimeter() const;  // computes shape's perimeter
  virtual void draw() const;        // draws the shape on the screen

protected:
  float centerX;      // x coordinate of figure's center
  float centerY;      // y coordinate of figure's center
  Color bodyColor;    // color of shape body
  Color borderColor;  // color of shape border
};
```

a. What new methods should be added for class `Circle`? What existing methods of class `Shape` should be overridden? Write a class declaration for class `Circle`, taking advantage of inheritance as you do so.

b. Write the copy constructor for class `Circle`, using initializers where appropriate.