This exam is **closed note, closed book**. You will have an hour and fifty minutes total to complete the exam. You may **NOT** use calculators.

Good Luck!
Define and give an example of each:

1. a) (2 points) length of a path in a tree

   In a path from node a to node b, this would be the number of edges between a and b. In other words. Example: if the path from a to b is a list of 6 nodes, the length of the path is 5. Path from a to b: a, g, h, i, j, b. Length of path = 5.

   b) (2 points) AVL tree structure property

   For each node, the height of its left subtree and right subtree can differ by no more than one. Example: (see problem 12) (several will work)

   c) (2 points) normalized (in the context of floating point values)

   A format where a number is represented as matissa * 2^exponent and in the mantissa there is only one place to the left of the binary point and that digit is always one. In IEEE floating point format that digit is not stored (it is the hidden bit) but is always assumed to be there. Example: 1.01101 * 2^9

   d) (2 points) High level language

   A language at a “higher level of abstraction” than assembly language. Typically a compiler is used to compile a HLL into assembly. Examples: C++, C, Fortran, Java, Scheme

2. (2 points) Give an example of a FIFO data structure and give the names of the operations used to add and remove elements from the structure:

   Queue, enqueue and dequeue

3. (2 points) Give two examples of things that may have an effect on running time, but Big-O time comparison ignores.

   Acceptable answers:
   - Constant coefficients
   - Lower-order terms
   - Compiler optimizations, programming language
   - Hardware performance
4. [2 points] A pixel in the cs216 graphics library can be any one of 700 possible colors. You are charged with creating a data structure to represent a pixel. What is the *minimum* number of bits needed to represent the color of a pixel?

10 bits   (since \(2^9 = 512\) and \(2^{10} = 1024\)).

5. [6 points total] In the highly impractical cs216 architecture, words are 9 bits long.
   a) Show how the number 13 would be represented in twos complement in a 9-bit word.

   \[0 \ 0000 \ 1101_2\]

   b) Show how -5 would be represented in twos complement in a 9-bit word.

   \[1 \ 1111 \ 1011_2\]

   c) Show how the result of adding 13 and -5 would be represented in an 9-bit word.

   \[0 \ 0000 \ 1000_2\]

6. [4 points] Write the following infix expression as a postfix expression:
   \[a + ((b \ c) \ast (d + e)) - f\]

   \[b \ c \ d \ e + \ast \ a \ f -\]

   OR

   \[a \ b \ c \ d \ e + \ast + f -]\
7. (8 points total) Describe the running time of the following pseudocode in Big-Oh notation in terms of the variable \( n \).  

*a) int test(int k) {
    for (int i = 0; i < k; ++i) {
        if (i < 10) {
            cout << “too small”;  
        }
    }
    return k;
}
a = test(n);  \[O(n) \text{ [since test() runs in } O(k) \text{ time]}\]*

*b) for (int i = 0; i < 99; ++i) {
    if (i > 50) {
        for (int j = 0; j < i; ++j) {
            cin >> b;
            a = a + b;
        }
    }
}
\[O(1) \text{ [constant time]}\]*

*c) for (int i = 0; i < n; ++i) {
    for (int j = 0; j < i * i; ++j) {
        sum = sum + i;
        for (int k = 0; k < j * j; ++k)
            a[k] = a[k] + sum;
    }
}
\[O(n^7)\]*

*d) for (int j = 0; j < n; ++j) {
    b = a * c;
    for (int i = 0; i < j * n; ++i)
        a = a + b;
    for (int k = 0; k < n; ++k)
        c = b + c;
}
\[O(n^3)\]*
8. (6 points total) What is the representation of each of the following in the indicated radix? Be sure to show your work.

a) $241_5$ in decimal

$$2*5^2 + 4*5 + 1 = 50 + 20 + 1 = 71_{10}$$

b) $2122_3$ in hex

$$2*3^3 + 1*3^2 + 2*3 + 2 = 54 + 9 + 6 + 2 = 71_{10}$$

$$71/16 = 4 \text{ rem } 7$$

$$4/16 = 0 \text{ rem } 4 = 47_{16}$$

c) $4A7_{11}$ in radix 10

$$4*11^2 + 10*11 + 7 = 4*121 + 110 + 7 = 601_{10}$$

9. (6 points) Consider the positive binary integer represented in two’s complement: $0101101101010000_2$.

a) Express this binary number in octal

$$0 101 101 101 010 000_2 = 055520_8$$

b. Express this binary number in hexadecimal

$$0101 1011 0101 0000_2 = 0x5B50$$

c. Negate the number (i.e. give the two’s complement representation of a negative version of the same number) Use the same number of bits.

$$1010 0100 1011 0000_2$$
10. a) (3 points) Draw the binary search tree created by inserting these values in this order:

```
4 3 0 9 8 6 2 5 1 7
```

Draw T.

![Binary Search Tree Diagram]

b) (2 points) Give a pre-order traversal of your tree shown above:

```
4 3 0 2 1 9 8 6 5 7
```
11. (3 points each) The cs216list class is implemented as a doubly linked list (similar to the one you used in lab #1). For each operation give 1) How you would most efficiently implement the operation in the cs216list, 2) WHAT exactly is the worst case scenario and 3) what is the worst case Big-Oh running time of this scenario. State any assumptions you make about the structure of a cs216list or the implementation of its operations.

a) Reading an element of a cs216list with the [ ] operator.

1. Traverse the linked list starting from the head pointer until you find the element requested.
2. Worse case: The user requests the last element in the list.
3. O(n)

b) push_back(T val) – inserts a copy of val after the last element in the cs216list.

1. Assuming you have a pointer to the element at the tail of the list (as we did in lab), you merely allocate a new node, assign it the value val, set its previous pointer to point to the current tail, set the current tail’s next pointer to point to the new node, and point the tail pointer to the new node.
2. Worse case: all cases are the same.
3. O(1) – constant time

c) size() – returns the size of the cs216list.

1. If the size is maintained internally each time elements are added to or removed from the list (as it was in lab), size() just returns the value of that internal state variable.
2. Worse case: all cases behave the same
3. O(1) - constant time

d) insert(iterator pos, T val) – inserts a copy of val prior the element in the cs216list referred to by pos.

1. Allocate a new node and assign it the value val. Set its previous pointer to point to the previous pointer of the element referred to by pos. Set its next pointer to point to the element referred to by pos. Set
2. Worse case: all cases behave the same
3. O(1) - constant time
12. (3 points) Given the following tree:

```
    20
   /|
  13 77
 /|
2 15 78
|
14 19
```

Is it an AVL tree? If not, circle the node(s) where the AVL property is violated. **Why or why not** (must answer for any credit)?

This is an AVL tree because the AVL property holds at every node. (At each node the height of its left and right subtrees differs by no more than one). (In addition the ordering property holds (descendants in the left subtree are less than current value and descendants in the right subtree are greater than the current value.)
13. (5 points) In C++ it is possible to declare a function as follows:

```c
1  int test (int a[][100]) {
2    int c;
3    c = a[3][6];
4    return c;
5  }
```

a) (3 points) What does the code the compiler generates for line 3 look like? (pseudocode is o.k.)

```
c = mem[&a + (3 * 100 * sizeof(int)) + 6 * sizeof(int)]
```

OR more generally, for `a[x][y]`:

```
&a + (x * number of columns * sizeof(int)) + (y * sizeof(int))
```

b) (2 points) If we changed the parameter `a` to a 3-dimensional array, which dimensions of the array must be specified? (give an example of the function declaration (line 1) for the new function) Why will this new declaration work?

In C++ (which does row-major allocation of arrays), you must specify all dimensions but the first one. Example:

```c
int test (int a[][100][200]) {
```

The address for a reference `a[i][j][k]` could be calculated as follows:

```
&a + (i* 100 * 200 * sizeof(int)) + (j* 200 * sizeof(int)) + k * sizeof(int)
```

Note that the size of the first dimension is not needed for this calculation.

Or more generally:

```
S1 = size of a plane = S2 * num_rows_per_plane
S2 = size of a row = sizeof(element) * num_elements_per_row
S3 = size of an individual element

&a + i * S1 + j * S2 + k * S3
```

OR

```
&a + (i* num_cols*sizeof(int)*num_rows_per_plane) +
(j*num_cols*sizeof(int)) + (k*sizeof(int))
```
14. (7 points) Assume we are using the 32-bit IEEE single precision floating point format as described in class and used in lab. The mantissa has 24 bits including the hidden bit, there is one sign bit, and there are eight exponent bits. The exponent is stored in excess 127.

What decimal floating point number is represented by the following 32 bits? SHOW YOUR WORK!

0011 0000 1001 1000 0000 0000 0000 0000

a) Is this a positive or negative number?  positive

b) What is the exponent (in base 10)?

0110 0001 is stored in excess 127 (i.e. any value is represented as value + 127)

Thus 64 + 32 + 1 = 97
If value + 127 = 97, then value must be -30

The exponent is -30.

c) What is the value of the mantissa (in base 10)

The mantissa is: 1.0011₂ which translates to:
1 + 1/8 + 1/16 = 19/16
1 + .125 + .0625 = 1.1875₁₀

d) What is the total value?
Note: you may leave your answer in the form: \(value₁₀ * base^{exponent}\)
Where you specify value, base and exponent.

\(1.1875₁₀ * 2^{-30}\)
15. (24 points total) a) (10 points) Implement a stack ADT in C++. The underlying representation of the stack should be an array. Your stack should store integers and should handle errors. You will be graded mostly on the correctness of the ideas of your solution rather than exact C++ syntax, but your solution should be clear. Correct C++ code is the best way to ensure we understand your solution.

You should implement the following:

Stack class declaration
Stack class constructor

You should implement the following member functions:

push(int val) - pushes val onto the stack
top() - returns the value on the top of the stack (without removing it from the stack)
pop() - returns the value on the top of the stack and removes it from the stack
isEmpty() - returns true if the stack contains no elements, false otherwise
isFull() - returns true if the stack is full, false otherwise
size() - returns the number of elements currently in the stack

const int MaxSize = 100;

class stack {
public:
    stack ();
    void push (int val);
    int top();
    int pop ();
    bool isEmpty ();
    bool isFull ();
    int size();

private:
    int data [MaxSize];
    int top_of_stack; // index of val on top of stack
};
stack::stack () {
    for (int i = 0; i < MaxSize; i++)
        data[i] = 0;
    top_of_stack = -1;
};

void stack::push(int val) {
    if (! isFull()) {
        top_of_stack++;
        data[top_of_stack] = val;
    } else {
        cout << "stack is full, cannot add more values"
    }
};

int stack::pop() {
    if (! isEmpty()) {
        top_of_stack--;
        return data[top_of_stack+1];
    } else {
        cout << "stack is empty"
    }
};

int stack::top() {
    if (! isEmpty()) {
        return data[top_of_stack];
    } else {
        cout << "stack is empty"
    }
};

int stack::isEmpty() {
    return (top_of_stack == -1);
};

int stack::isFull() {
    return (top_of_stack == MaxSize -1);
};

int stack::size() {
    return (top_of_stack + 1 );
};
b) (9 points) Implement a non-member function `print_stack(stack S)` in C++. `print_stack` should print out all elements currently in the stack in the order they were originally inserted into the stack. For example:

```cpp
S.push(1);
S.push(2);
S.push(3);
print_stack(S);
```

Should print: **1 2 3**

`print_stack` cannot access private data members and functions of the stack class. When you return from `print_stack`, the original stack should be unchanged.

**People did this several ways:**

**Option A:** (Use a temporary stack)
1) pop all values off of S and onto push them onto a temporary stack
2) pop all values off of the temporary stack and print them as you pop them
   (optionally – push the values back onto the original stack, since S would be passed by value, in most cases the original S would not be modified)

**Option B:**
Recursive solution – if empty then return, else {pop a value, make recursive call to self, print the previously popped value}.

**Option C:** (Use a temporary array)
Similar to option A, but instead of pushing values onto a temp stack, they put them into an array. Then they read the values backwards from the array.

Printing the values in the wrong order was completely wrong for this question!
c) (3 points) Give the worst case big-O running time of your implementation for each of the following operations (in terms of n where n is the number of elements currently in the stack). For partial credit, explain briefly how you got your answer.

\[
push(\text{int val}) - O(1) \\
pop() - O(1) \\
size() - O(1)
\]

d) (2 points) Give the worst case big-O running time of your implementation of \text{print_stack} (in terms of n where n is the number of elements currently in the stack). \textit{For full credit you must explain briefly how you got your answer.}

Option A: temp stack
1) \(O(n)\) to pop and push each value once
2) \(O(n)\) to pop and print each value once
3) \(O(n)\) to (optionally) push the value back on the original stack.
\[O(n + n + n) = O(n)\]

Option B: recursive
Makes \(n\) recursive calls(one per element) and does constant work at each step. \(O(n)\)

Option C: temp array
4) \(O(n)\) to pop and store each value once into temp array
5) \(O(n)\) to read each value from the array and print it
6) \(O(n)\) to (optionally) push the value back on the original stack.
\[O(n + n + n) = O(n)\]

On my honor, I have neither given nor received unauthorized aid on this exam.

_________________________
Sign your name