CS 216 Exam 1 – Fall 2001 – SOLUTION

Part 1 – Closed Book

Name _______________________  Section _______________________

Email Address _________________  Student ID # _________________

Pledge:

This exam is in two parts. Once you are finished with part 1, **hand it in**, and you will be given part 2. Part 1 is worth 75 points and Part 2 is worth 25 points. You will have an hour and fifty minutes total to complete both parts.

**Part 1:** Closed note, closed book. You are not to speak with anyone except the Instructor or a teaching assistant for any reason except an emergency during the exam. No calculators are allowed.

**Part 2:** Open book, open notes. You may not speak with anyone except the Instructor or a teaching assistant for any reason except an emergency during the exam. Calculators are allowed. You may not borrow notes or textbooks from other students.

Good Luck!

<table>
<thead>
<tr>
<th>PART</th>
<th>MAX</th>
<th>SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>96</td>
<td></td>
</tr>
</tbody>
</table>
CS 216 Exam 1 – Part I – Closed Book

1. a) (2 points) Define the term abstraction as it relates to this class:

Abstraction is extracting the relevant properties and ignoring the irrelevant details. The flip side to abstraction is representation/implementation. Abstraction is a mechanism for dealing with complexity. Practically everything we will study in this class could be viewed as an abstraction of some sort. The most obvious example is abstract data types. High level languages can be thought of as an abstraction that hides the details of implementation details such as memory addresses and register names.

b) (2 points) Give an example of an abstraction we have discussed in this class and why it can be considered an abstraction.

The STL vector class is one example of an abstraction. For example, you do not know how the vector class is implemented underneath – as an array or as a linked list. You do not need to know this detail as long as a vector provides the advertised functionality.

2. (2 points) What is the AVL property? The height of the left and right subtrees differ by at most one.

3. (2 points) List two advantages of choosing the correct container abstraction that we mentioned in class. Choosing an appropriate container for your program will have an impact on overall system performance, reliability, and maintainability. Selecting a queue over a more general list, for example, a) makes your design easier to understand – the fact that you are using a queue conveys some information about what sorts of operations you will perform, b) ensures that the operations you are most interested in performing will be optimized for the best performance.

4. (2 points) Usually we refer to (circle one) the best average worst case performance when we are discussing the big-Oh of an algorithm.

5. a) (3 points) Give an example of a situation when a big-O time comparison is useful:

Basically this question is testing your understanding of big-O. A big-O comparison of two algorithms is useful when you are interested in the performance of the algorithms for large values of n. If n is very small, then often constant factors will dominate and the big-O comparison is not very useful. If you are interested in comparing the predicted performance of two algorithms independent of factors such as the operating system, compiler, or underlying hardware that will eventually be used to execute the program, then a big-O comparison is useful. Big-O provides a rough estimate of performance that ignores things such as number of memory or disk accesses which can have a large effect of program performance. Nevertheless, an asymptotic analysis of two algorithms may allow you to quickly reject one as obviously less efficient( say O(2^n) vs. O(n)) or may help direct you towards portions of your code that are likely to be the bottleneck.

b) (3 points) Give an example of when a big-O time comparison is not useful.

When you know n will be very small. If you are interested in comparing the effects of details such as number of memory accesses, or the quality of code generated by two different compilers.
6. [2 points] You have a computer with 1024 addressable words of memory. What is the smallest number of bits required to address all words in your computer?

10

7. [2 points each] In the Exam1 architecture, words are 8 bits long.
a) Show how the number 7 would be represented in twos complement in an 8-bit word.

    0000 0111  (o.k. to leave off leading zeroes)

b) Show how -15 would be represented in twos complement in an 8-bit word.

    0000 1111 = 15
    1111 0000  flip
    1111 0001  add one = -15  (must include all 4 leading 1 bits for any credit)

c) Show how the result of adding 7 and –15 would be represented in an 8-bit word.

    0000 0111  = 7
    1111 0001  = -15
    ----------------
    1111 1000  = -8

8. [2 points] Which of the following is true about overflow:
   a) The compiler will never allow overflow to occur.
   b) Overflow cannot occur with integer values.
   c) **Overflow cannot occur when adding a positive and a negative value.**
   d) The program will halt when overflow occurs.

9. [2 points] In C++ the sizeof() function returns:
   a) The number of bytes in a word on this machine
   b) **The size in bytes of a variable or type.**
   c) The length of an address in bytes on this machine.
   d) The maximum value for a variable or type.

10. [4 points total] In 32-bit IEEE single precision floating point, the exponent is stored in _____ 8 _____ (fill in number of bits) bits. The _____ mantissa _____ (fill in name of field) stores the fractional value.

11. [3 points] Write the following infix expression as a postfix expression:

    ((a + b) * c) –d

    a b + c * d -
12. (8 points total) Describe the running time of the following pseudocode in Big O notation:

a) for (int i = 0; i < n; ++i) {
    for (int j = 0; j < i; ++j) {
        a = a + b;
    }
}
\[O(n^2)\]

b) for (int j = 0; j < n; ++j) {
    for (int i = 0; i < j; ++i) {
        a = a + b;
    }
    for (int i = 0; i < n; ++i) {
        c = b + c;
        cin >> b;
    }
}
\[O(n^2)\]

c) for (int i = 0; i < 999999; ++i) {
    for (int j = 0; j < n; ++j) {
        a[i] = a[i] + b + c;
    }
    for (int k = 0; k < n; ++k) {
        cout << i;
    }
}
\[O(n)\]

d) for (int i = 0; i < n; ++i) {
    for (int j = 0; j < i * i; ++j) {
        sum = sum + i;
        cout << sum << endl;
    }
    for (int k = 0; k < n; ++k) {
        cin >> sum;
    }
}
\[O(n^3)\]
13. (6 points) What is the representation of each of the following in the indicated radix? Be sure to show your work.

a) \(730_8\) in hex

\[
11101100010_2 \text{ in binary, group the digits into 4s to get hex:}
\]

\[
11101100010_2 = \text{EC2}_{16}
\]

b) \(1BA_{13}\) in decimal

\[
(1 \times 13^2) + (11 \times 13) + (10 \times 1) \\
169 + 143 + 10 = 322_{10}
\]

c) \(140_{10}\) in radix 7

\[
\begin{align*}
140/7 &= 20 \text{ rem } 0 \\
20/7 &= 2 \text{ rem } 6 \\
2/7 &= 0 \text{ rem } 2
\end{align*}
\]

\[
= 260_7
\]

14. (6 points) Consider the positive binary integer \(0111100110111000_2\). Express

a) this binary number in octal

\[
74570_8
\]

b. this binary number in hexadecimal

\[
7978_{16}
\]

c. the negative of the number, assuming a two’s complement representation (same number of bits)

\[
1000011010001000
\]
15. (3 points) Consider a binary search tree that is built by inserting the sequence:
24, 78, 55, 11, 17, 16, 28, 4, 42, 3
into an initially empty binary tree.

Draw the tree constructed by this insertion sequence.

```
        24
       /   \
      11    78
     /   /  \
    4    55  28
   /   /  \
  3    17  42
```

16. (5 points) Given the following tree:

```
        10
       /   \
      3     46
     /   /  \
    2    5  78
    /      /  \
   9       9   
```

a) Is it an AVL tree? If not, circle the nodes that violate the AVL property. Why or why not (must answer for any credit)?

Yes, it is an AVL tree. The AVL property holds at every node.

b) List the order in which the nodes are visited for a preorder traversal:

```
10  3  2  5  9  46  78
```
17. (2 points) Draw the BST tree after removing the value 10 from the tree. Use one of the two methods we described in class.

Original:
```
     10
   /   \
  3     46
 /   \   /   \n2     5 78  9
```

Replace 10 with smallest val on RHS:
```
     46
   /   \
  3     78
 /   \   /   \n2     5 3   9
```

Replace 10 with largest val on LHS:
```
     9
   /   \
  3     46
 /   \   /   \
2     5 78  9
```

For 18 and 19: 0 pts if wrong answer, 1 pt if right answer but wrong reason or no reason, 3 pts if right answer and right reason.

18. (3 points) What is the worst case running time (in big-O notation) to insert an element into a Binary Search Tree? And Why?

O(n) – This corresponds to a list of values inserted in order into the BST. Creating what in essence a linked list. The worst case is where inserting the next element requires following a path of length n-1 for a tree containing n nodes.

19. (3 points) What is the worst case running time (in big-O notation) to print all the values in an AVL tree? And Why?

O(n) – You must visit each node once, and at each node you do a constant amount of work.
While working on the remainder of Exam 1, you may consult your notes or text. You may use a calculator. You may NOT borrow notes or texts from other students or talk to other students.

20. (5 points) Assume we are using the 32-bit IEEE single precision floating point format as described in class and used in lab.

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

0100 0001 0110 0000 0000 0000 0000 0000

0   100 0001 0    110 0000 0000 0000 0000 0000

sign bit = 0 = positive number

exponent = 8 bits in excess 127, that is, the value K is represented as K + 127.

= 1000 0010 = 128 + 2 = 130 in excess 127.

= the value represented by 130 is 130-127 = 3. 

= the exponent is three.

The matissa = .11

Put it all together, including the hidden bit:

1.11₂ * 2³ = (1 + ½ + ¼) * 2³ = 1.75 * 8 = 14
21. (20 points) The data structure called a deque (pronounced either “deck” or “DQ” – short for a double-ended queue) is a list in which entries can be added or removed from either the first or last position of the list, but no changes may be made elsewhere in the list. We will define the fundamental operations on a deque to be:

append_front – appends an item on the front of the deque
append_back - appends an item on the back of the deque
pop_front – removes and returns an item from the front of the deque
pop_back – removes and returns an item from the back of the deque

a) (4 points) Write (and or draw) the general idea of how you would implement the deque class.

Use a doubly linked list with a pointer to the front and the back of the list.

b) (4 points) Write a C++ class header file for your deque class.

```cpp
class Node{
    public:
        int Val;
        Node *Prev;
        Node *Next;
    }

class deque {
    public:
        deque() : Front(NULL), Back(NULL) {} // Constructor
        void append_front(int val);
        void append_back(int val);
        int pop_front();
        int pop_back();
    private:
        Node *Front;
        Node *Back;
    }
```
(2 points) Implement the constructor for your deque class.

(Implementation on previous page)

c) (4 points) Implement your append_front method.

```c
void deque::append_front(int val) {

    // Create a new node.
    Node *temp = new Node;
    temp->Next = NULL;
    temp->Prev = NULL;
    temp->Val = val;

    if (!Front) { // Add the first item to the list.
        Front = Back = temp;
    } else {
        temp->Next = Front;
        Front->Prev = temp;
        Front = temp;
    }
}
```

- Possible problems: Not handling case of empty list at all, or not updating back.

(2 points each)

d) What is the big-O running time of your append_front method and why?

O(1) - none of the operations in append_front depend on the number of items in the deque. We are only doing a constant number of pointer manipulations.

e) What is the big-O running time of your pop_front method and why?

O(1) - none of the operations in pop_front depend on the number of items in the deque. Similar to append_front, we would only do a constant number of pointer manipulations.

f) What is the big-O running time of your append_back method and why?

O(1) - none of the operations in append_back depend on the number of items in the deque. The code for append_back would be incredibly similar to the code for append_front, we would basically do the same (constant) number of pointer manipulations for the Back of the deque rather than the front.