**CptS 122 – Data Structures**

**Summer 2021**

***Final Exam***

**Your Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**ID Number: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**TA’s Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

READ THE FOLLOWING INSTRUCTIONS:

This exam is take-home. It should take ~ 2 hours to complete. You must work **individually** on this exam. You may use your book and notes if necessary. Please either handwrite or type your answers into this document. Late exam solutions will **not** be accepted! **Finding a solution online for any given problem could lead to zero credit for the problem.**

You must submit the exam through Blackboard by Friday, July 30, midnight PST

|  |  |  |
| --- | --- | --- |
| **Part** | **Points Possible** | **Points Earned** |
| I | 56 |  |
| II | 20 |  |
| III | 30 |  |
| IV | 29 |  |
| **Total** | **135** |  |

***Part I: Multiple-Choice (2 points each)***

1. Given a pointer to a node X in a singly linked list. Only one pointer is given, a pointer to the head node is NOT given, can we delete the node X from given linked list?
2. **Possible if X is not the last node. Use the following two steps (a) Copy the data of the next of X to X. (b) Delete next of X.**
3. Possible if size of linked list is even.
4. Possible if X is not first node. Use the following two steps (a) Copy the data of next of X to X. (b) Delete next of X.
5. Possible if size of linked list is odd.

2. In the linked-list version of the Stack class, with a pointer to the top, which operations require linear time (O(n)) for their worst-case behavior?

A. isEmpty ()

B. peek ()

C. pop ()

D. push ()

E. **None of these operations require linear time.**

1. What are the user-defined types called that are created by C++ programmers?

**A. Structures**

**B. Classes**

C. Stacks

D. Queues

E. Lists

1. Suppose you were going to quickly implement a dynamic Stack class. What class would be the best base class?

A. Binary search tree

B. Queue

**C. List**

D. Graph

1. Consider this class definition:

class quiz

{

public:

quiz( );

int f( );

int g( ) const;

private:

double score;

};

Which functions can carry out an assignment score = 1.0; to the private member variable score?

A. Both f and g can carry out the assignment

**B. f can carry out the assignment, but not g**

C. g can carry out the assignment, but not f

D. Neither f nor g can carry out the assignment

1. Which kind of functions can access private member variables of a class?

A. Friend functions of the class

B. Private member functions of the class

C. Public member functions of the class

D. **All of the above**

1. What is the worst case time complexity for search, insert, and delete operations in a general Binary Search Tree?
2. O(Log n) for all
3. **O(n) for all**
4. O(Log n) for search and insert, and O(n) for delete
5. O(Log n) for search, and O(n) for insert and delete
6. None of the above
7. In the worst case scenario, which data structure performs the best when looking for an item?

A. Ordered list

B. Queue

C. Stack

**D. Balanced BST**

E. All of the above data structures perform the same.

1. Which of the following does an object-oriented approach to problem solving make use of?
2. Polymorphism
3. Polynomials
4. Inheritance
5. Compiling
6. Encapsulation
7. **(A), (C) and (E) above.**
8. Which of the following traversal outputs the data in sorted order in a BST?

A. Levelorder

B. Postorder

**C. Inorder**

D. Preorder

1. Suppose cursor points to a node in a linked list (using the node definition with member functions called data and link). What statement changes cursor so that it points to the next node?

A. cursor++;

B. cursor = link( );

C. cursor += link( );

**D. cursor = cursor->link( );**

1. What kind of list is best to answer questions such as "What is the item at position n?"

**A. Lists implemented with an array**

B. Singly-linked lists that are created dynamically

C. You cannot solve this problem with a list

1. Consider the following pseudocode that uses a stack

|  |
| --- |
| declare a stack of characters  while ( there are more characters in the word to read )  {     read a character     push the character on the stack  }  while ( the stack is not empty )  {     pop a character off the stack     write the character to the screen  } |

What is output for input "CptS122"?

1. CptS122221StpC
2. CptS122
3. **221StpC**
4. None of the above
5. What does the function print() do in general? The function print() receives the root of a Binary Search Tree (BST) and a positive integer k as arguments.

|  |
| --- |
| // A BST node  struct node {      int data;      struct node \*left, \*right;  };    int count = 0;    void print(struct node \*root, int k)  {      if (root != NULL && count <= k)      {          print(root->right, k);          count++;          if (count == k)            printf("%d ", root->data);         print(root->left, k);      }  } |

1. Prints the kth smallest element in the BST
2. Prints the leftmost node at level k from the root
3. **Prints the kth largest element in the BST**
4. Prints the rightmost node at level k from the root
5. None of the above
6. I have implemented the queue with a linked list, keeping track of a front pointer and a rear pointer. Which of these pointers will change during a deletion from a NON-EMPTY queue?

A. Neither will change

B. Only front\_ptr changes

C. Only rear\_ptr changes

**D. Both could change**

1. What are different application(s) of a stack?
2. Tracking function calls
3. Reversal of a string
4. Conversion of infix expressions to postfix expressions
5. A parentheses balancing program
6. Tracking of local variables at run time
7. Complier Syntax Analyer
8. **All of the above**
9. None of the above
10. In the linked-list version of the queue class with a pointer to the head and a pointer to the tail, which operations require linear time (O (n)) for their worst-case behavior?

A. dequeue ( )

B. enqueue ( )

C. isEmpty ( )

**D. None of these operations require linear time**

E. All of the operations require linear time

1. Which of the following is true about linked list implementation of stack?
2. In the push operation, if new nodes are inserted at the end, then in the pop operation, nodes must be removed from the end.
3. In the push operation, if new nodes are inserted at the front, then in the pop operation, nodes must be removed from the front.
4. **Both of the above.**
5. None of the above.
6. Following is C like pseudo code of a function that takes a Queue as an argument, and uses a stack S to do processing.

|  |
| --- |
| void fun(Queue \*Q)  {      Stack S;  // Say it creates an empty stack S        // Run while Q is not empty      while (!isEmpty(Q))      {          // deQueue an item from Q and push the dequeued item to S          push(&S, deQueue(Q));      }        // Run while Stack S is not empty      while (!isEmpty(&S))      {        // Pop an item from S and enqueue the poppped item to Q        enQueue(Q, pop(&S));      }  } |

What does the above function do in general?

1. Makes Q empty
2. **Reverses the Q**
3. Removes the last node from Q
4. Keeps Q the same as it was before the call

14

/ \

2 11

/ \ / \

1 3 10 30

/ /

7 40

1. Given the tree to the right, which is a binary tree (but not a binary search tree), what is the order of nodes visited using a pre-order traversal?

A. 1 2 3 7 10 11 14 30 40

B. 14 1 3 2 7 10 11 30 40

C. 1 3 2 7 10 40 30 11 14

**D. 14 2 1 3 11 10 7 30 40**

E. None of the above

1. Which of the following is a disadvantage to implementing a dynamic singly linked list?

A. Random access is not allowed. We have to access elements sequentially starting from the first node. So we

cannot do binary search with linked lists.  
B. Extra memory space for a pointer is required with each element of the list  
C. Cannot in constant time determine what is at item *n* in the list

**D. All of the above are disadvantages**

1. Consider the following function to traverse a linked list.

|  |
| --- |
| void traverse(struct Node \*head)  {     while (head->next != NULL)     {         printf("%d  ", head->data);         head = head->next;     }  }  Which of the following is **FALSE**about the above function? |

1. **The function is implemented incorrectly because it changes head**
2. The function may crash when the linked list is empty
3. The function doesn’t print the last node when the linked list is not empty
4. The operation for adding an entry to a stack is traditionally called:

A. add

B. append

C. insert

**D. push**

1. The following function reverse() is supposed to reverse a singly linked list. There is one line missing at the end of the function.

|  |
| --- |
| /\* Link list node \*/  struct node  {      int data;      struct node\* next;  };    /\* head\_ref is a double pointer which points to head (or start) pointer    of linked list \*/  static void reverse(struct node\*\* head\_ref)  {      struct node\* prev   = NULL;      struct node\* current = \*head\_ref;      struct node\* next;      while (current != NULL)      {          next  = current->next;          current->next = prev;          prev = current;          current = next;      }  **/\*ADD A STATEMENT HERE\*/**  } |

What should be added in place of "/\*ADD A STATEMENT HERE\*/", so that the function correctly reverses a linked list?

1. **\*head\_ref = prev;**
2. \*head\_ref = current;
3. \*head\_ref = next;
4. \*head\_ref = NULL;
5. Why may a *tree* class have two versions of the insert member function?

**A. One is public, the other is private to hide private pointer information**

B. One is to use with a const pointer, the other with a regular pointer

C. One returns the forward link, the other returns the backward link

D. One returns the data, the other returns a pointer to the next node

1. You are given pointers to first and last nodes of a singly linked list, which of the following operations are dependent on the length of the linked list?
2. Delete the first node
3. **Delete the last node of the list**
4. Insert a new node as the first node
5. Add a new node at the end of the list
6. What are the “Big Three” in C++?
7. Setters, getters, and constructor
8. Setters, destructor, and constructor
9. Copy constructor, constructor, and destructor
10. **Destructor, copy constructor, and overloaded copy assignment operator**
11. I have implemented the queue with a linked list, keeping track of a front node and a rear node with two reference variables. Which of these reference variables will change during an insertion into a NONEMPTY queue?

A. Neither changes

B. Only front changes.

**C. Only rear changes.**

D. Both change.

***Part II: Fill-In-The-Blank (2 pts each)***

1. A dynamic \_\_\_\_\_doubly\_\_\_\_\_\_\_\_\_\_\_ linked list can be traversed both in the forward and backward direction.
2. The C++ \_\_\_\_\_\_new\_\_\_\_\_\_\_\_\_\_\_\_ operator dynamically allocates memory for an object of a specified type and returns a \_\_\_\_\_\_\_\_pointer\_\_\_\_\_\_\_\_ to that type.
3. The keyword \_\_\_\_\_\_operator\_\_\_\_\_\_\_\_\_ introduces an overloaded operator method or function definition.
4. When deriving a class from a base class with public inheritance, public members of the base class become \_\_\_\_\_\_\_public\_\_\_\_\_\_\_\_\_\_ members of the derived class and protected members of the base class become \_\_\_\_\_\_\_\_\_\_\_\_ protected \_\_\_\_\_\_\_\_\_\_\_ members of the derived class.
5. Having multiple functions with the same function name that operate on different types and numbers of parameters is called \_\_\_\_\_\_\_function overloading\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
6. A \_\_\_\_binary search tree\_\_\_\_\_ is a non-linear data structure, which stores data in a way that makes it easy to search.
7. Given the following function, what is the worst-case Big-O time complexity?

\_\_\_\_\_\_\_O( 1 )\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

bool Queue::enqueue(const string &newString)

{

bool success = false;

Node \*pMem = new Node(newString);

if (pMem != nullptr)

{

success = true;

// works if queue is empty

if (this->mpHead == nullptr)

{

this->setHeadPtr(pMem);

this->setTailPtr(pMem);

//this->mpHead = this->mpTail = pMem;

}

else

{

// not empty - insert at the tail

this->mpTail->setNextPtr(pMem);

this->setTailPtr(pMem);

}

}

return success;

}

1. Inheritance enables \_\_\_\_\_\_software reuse\_\_\_\_\_\_\_, which saves time in development and encourages using previously proven high-quality software.
2. A class with at least one pure virtual function is called a(n) \_\_\_\_\_abstract\_\_\_\_\_ class.
3. If items need to be stored in a first-come, first-served method, then a(n) \_\_\_\_\_\_queue\_\_\_\_\_\_ should be used.

***Part III: True or False (2 points each)***

1. **T F** Inserting data, from an unsorted array into a BST, and then traversing the BST in order, while writing the data back to the array will produce a sorted array. (True)
2. **T F** Inheriting from two or more base classes is called multiple inheritance. (True)
3. **T F**  A stack enforces FIFO. (False)
4. **T F** A destructor is a member function that is automatically called when an object is destroyed by leaving scope or by calling keyword *delete* explicitly. (True)
5. **T F** Inserting an item at the front of a non-empty linked list is more efficient than inserting an item at the front (index 0) of an array implementation of a non-empty list. (True)
6. **T F** Overridable functions are declared with the keyword *operator*. (False)

1. **T F** All *virtual* functions in an *abstract* base class must be declared as pure virtual functions. (False)
2. **T F** Treating a base-class object as a derived-class object can cause errors. (True)
3. **T F**  It’s possible to implement a queue with an array. (True)
4. **T F** Inheritance is a fundamental way of encapsulating the “has a” relationship between two objects. (False)
5. **T F** An object’s member functions have access to a “self pointer” to the object called the *this* pointer. (True)
6. **T F** Class diagrams, which are part of UML, may be used to illustrate relationships between various classes during the design phase. (True)
7. **T F** Casting a base-class pointer to a derived-class pointer is called *downcasting*. (True)
8. **T F** In C++, the precedence of an operator may not be changed. (True)
9. **T F**  Division by zero is a *runtime\_error* exception. (True)

***Part IV: Application***

1. (29 pts) Compare and contrast linked lists, linked stacks, linked queues, and linked BSTs. Discuss insertions, deletions, and searches. In your discussion be sure to mention the Big-O complexities of these operations. Also, discuss applications for each data structure.