# **Solution for Exam 1**

*This is a closed book and closed notes test.* You are not allowed to have anything on your desk other than pencil and this exam paper during the test; this includes *calculators* or *electronic assistance* of any kind – ***especially smartphones***.

*You may not leave to go to the restroom.* Please go before the exam starts.

*You may not ask questions.* If something is confusing, write a note beside the question and explain your assumptions.

*You must show all of your work on this exam.* You will not be allowed to turn in additional sheets of paper.

*Read and sign the following statement.*  Failure to sign the statement will result in a **zero** on the exam.

*I have neither given nor received unauthorized assistance on this test. I have notified the proctor of any violations of the above policies.*

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| **Problem** | **Score** |
| 1 | / 30 |
| 2 | / 30 |
| 3 | / 20 |
| 4 | / 20 |
| **Total** | **/ 100** |

*Points divided evenly among parts of a problem unless otherwise specified.*

1. (30 points) Evaluate the following C++ code snippets.

class Pizza {

private:

int diameter;

int slices;

public:

bool large;

// approximate pi as 3

double area() { return (diameter\*diameter/4)\*3; }

public:

Pizza(int d=12,

int s=8) {

diameter = d;

slices = s;

if (diameter > 14) large = true;

else large = false;

}

int getSlices() { return slices; }

int getDiameter() { return diameter; }

double areaPerSlice() {

return area() / slices;

}

};

Consider these variables declared within a function:

Pizza large(16);

Pizza personal(6, 4);

Pizza medium;

Pizza small(10.0,6);

Evaluate the following expressions. Or, if the expression is illegal, explain why.

|  |  |  |
| --- | --- | --- |
| **#** | **Expression** | **Evaluate or if illegal write “Syntax error” and explain why** |
| 1 | large.getDiameter() | **16** |
| 2 | medium.getSlices() | **8** |
| 3 | large.areaPerSlice() > small.areaPerSlice() | **true** |
| 4 | personal.getSlices() | **4** |
| 5 | medium.areaPerSlice() | **13.5** |
| 6 | medium.large | **false** |
| 7 | large.large | **true** |
| 8 | large.areaPerSlice() | **24** |
| 9 | medium.medium | **Syntax error! No field named medium.** |
| 10 | personal.area() | **27** |

2. (30 points) Design a class to implement a Queue of integers. Your class should support push, pop, peek, and empty. You must use a linked list as your implementation of the queue, and may not assume some other code implements the linked list. Your code must include that implementation.

|  |
| --- |
| class Queue {  private:  // node for the linked list  class Node {  public:  Node(int d):data(d),next(NULL) {}  int data;  Node\* next;  };  Node \*head;  Node \*tail;  public:  Queue() { head=NULL; tail=NULL; }  bool empty() { return head==NULL; }  // Push to tail  void push(int data) { // push to tail  Node \*node = new Node(data);  if(head==NULL) {  head = tail = node;  } else {  tail->next = node;  tail = node;  }  }  // Remove from head  void pop() {  Node \*tmp = head;  head = head->next;  if(head==NULL) tail = NULL;  delete tmp;  }  // return the first element of the list.  // peek should never be called on an empty list!  int peek() {  return head->data;  }  }; |

3. (20 points) Consider the following operations on a singly linked list of length N. State the big-O complexity of each operation and justify your answer.

1. Insert a new node before a node that you have a pointer to.

|  |
| --- |
| This is O(N). You must traverse the list to just before the node, and insert there. In the worst case, the node could be at the end of the list. |

1. Append an item to the end of a list given a pointer to the tail of the list.

|  |
| --- |
| O(1). All you have to do is set the tail’s next pointer to the new item. |

1. Insert an item in the middle of the list, given the head pointer.

|  |
| --- |
| O(N). You have to traverse to the middle of the list, which requires N/2 steps. Hence, O(N). |

1. Print the nodes of the linked list in reverse.

|  |
| --- |
| Two answers I will accept here, depending on the rationale provided.  O(N\*N). On a singly linked list, you can’t go backwards. To get to the previous element, you must start at the beginning and go forwards until just before the previous node printed. This requires N steps. Then, you have to do this for all elements in the list, which is another factor of N, since we have to traverse N items (worst case) for each of the N items. Altogether, this is N\*N steps. O(N\*N).  O(N). Traverse the linked list forward using a recursive function. This would take O(N) to reach the end. As the recursive function returns, print each node. This only requires O(1). So, in total, O(N). Here’s the code:  void print(head) {  if(head!=NULL)  print(head->next);  print(“%d”, head->data);  }  If your answer lacked justification, you lost points. |

4. (20 points) Give an example of C++ code for each concept below and briefly (in one comment/sentence) explain why it demonstrates the concept. You do not need to show a full implementation. Use comments to fill in gaps to avoid writing a lot of code.

1. Abstract Base Class.

|  |
| --- |
| class A {  virtual void f() = 0; // abstract base class has a pure virtual function  }; |

1. A default class constructor that takes a parameter.

|  |
| --- |
| class B {  public:  B(int x = 5); // x will have 5 as it’s default value if called without an argument  }; |

1. A private class member.

|  |
| --- |
| class C {  private:  int i; // private i  }; |

1. Is-A relationship between classes.

|  |
| --- |
| // Is-A means public inheritance  class D {  public:  int d;  };  // DerivedfromD is-a D  class DerivedFromD : public D {  public:  int another;  }; |

**C++ Keywords**

In common with C:

auto const double float int short struct unsigned  
break continue else for long signed switch void  
case default enum goto register sizeof typedef volatile  
char do extern if return static union while

Unique to C++:

asm dynamic\_cast namespace reinterpret\_cast try  
bool explicit new static\_cast typeid  
catch false operator template typename  
class friend private this using  
const\_cast inline public throw virtual  
delete mutable protected true wchar\_t

Reserved words:

and bitand compl not\_eq or\_eq xor\_eq  
and\_eq bitor not or xor

**ASCII Table**

