Print Your Name **: _____ Exam # _____

****By submitting your exam for grading, you certify that you have worked independently of others in the completion of this exam, and you agree to be bound by the UAH academic misconduct policies as outlined in the UAH Student Handbook and the CPE212 course syllabus.**

- **This exam is Closed Book and Closed Notes.**
- **No Cell Phones, PDAs, Laptops, MP3 Players, calculators, Bluetooth earpieces, headphones, smart watches, or other computing/communication devices**
- **There should be nothing on your table other than your exam, your bubble sheet, pencils, and erasers.**
- **You may write on the Exam ==> Do Not Disassemble the Exam <==**
- **You Must Return Both The Exam Paper Intact And The Bubble Sheet.**

Section I – Stacks Problems 01-07 – Questions on **Stack ADT** concepts.

Stack ADT Function Prototypes: void Push(int n); void Pop( );

01. Suppose that a **stack** object contains the following values **Top { 4, 6, 2, 8 } Bottom**. What is the **contents of the stack** after the following operation? **Push(3)**

A) **Top { 4, 6, 2, 8 } Bottom**
B) **Top { 3, 4, 6, 2, 8 } Bottom**
C) **Top { 6, 2, 8 } Bottom**
D) **Top { 4, 6, 2, 8, 3 } Bottom**
E) None of the answers provided

02. Suppose that a **stack** object contains the following values **Top { 4, 6, 2, 8 } Bottom**. What is the **contents of the stack** after the following operation? **Pop( )**

A) **Top { 4, 6, 2 } Bottom**
B) **Top { 6, 2, 8 } Bottom**
C) **Top { 4, 6, 2, 8 } Bottom**
D) **Top { 6, 2 } Bottom**
E) None of the answers provided

03. Suppose that a **stack** object contains the following values **Top { 4, 6, 2, 8 } Bottom**. What is the **contents of the stack** after the following sequence of operations are executed left-to-right? **Push(3), Pop( )**

A) **Top { 3, 4, 6, 2, 8 } Bottom**
B) **Top { 4, 6, 2, 8, 3 } Bottom**
C) **Top { 6, 2, 8, 3 } Bottom**
D) **Top { 3, 4, 6, 2 } Bottom**
E) None of the answers provided

04. Suppose that a **stack** object contains the following values **Top { 4, 6, 2, 8 } Bottom**. What is the **contents of the stack** after the following sequence of operations are executed left-to-right? **Pop( ), Push(3)**

A) **Top { 6, 2, 8, 3 } Bottom**
B) **Top { 3, 4, 6, 2 } Bottom**
C) **Top { 3, 6, 2, 8 } Bottom**
D) **Top { 3, 4, 6, 2, 8 } Bottom**
E) None of the answers provided

05. What is the complexity of the **MakeEmpty()** operation for the array implementation of **Stack ?**

A) **O(1)** B) **O(N)** C) **O(logN)** D) **O(N^2)** E) None of the answers provided

06. What is the complexity of the **IsEmpty()** operation for a linked node implementation of a **Stack**?

A) **O(1)** B) **O(N)** C) **O(logN)** D) **O(N^2)** E) None of the answers provided

07. Which acronym accurately describes the access policy of a **stack** container?

A) **LOL** B) **LIFO** C) **FIFO** D) **LIFI** E) None of the answers provided

Problems 08-14 utilize the following **Stack class** declaration
Your solution must have no memory leaks!!

```
// stack.h - complete header file for Stack ADT
#include <iostream>
using namespace std;
class StackEmpty          { /* Empty exception class */ };
class StackFull           { /* Empty exception class */ };
class StackInvalidPeek    { /* Empty exception class */ };

const int MAXSIZE = 5;    // Maximum number of values stored in Stack

class Stack               // Static array implementation of Stack ADT
{
 private:
  int  data[MAXSIZE];     // Allocated array that holds stack data
  int  top;               // Index of top element on stack

 public:
  Stack();                // Initializes private variables of stack object

  ~Stack();               // Destructor

  void Push(int n);       // Adds integer n to top of stack;
                          // Throws StackFull if already full

  int Pop();              // Removes and returns top integer from stack
                          // Throws StackEmpty if stack is empty

  int Peek(int n) const; // Returns the value n positions down from top element
                          // Throws StackEmpty if stack is empty
                          // Throws StackInvalidPeek if n is otherwise out of range

  int Available() const; // Return the number of unused array elements


};  // End Class Stack
```

08. Which if any of the following code segments correctly implements the constructor for **Stack** ?

A)
```
Stack::Stack()
{

  int top = -1;
}
```

B)
```
Stack::Stack()
{
    data = new int[MAXSIZE];
    int top = -1;
}
```

C)
```
Stack::Stack()
{
  top = -1;
}
```

D)
```
void Stack::Stack()
{
    top = 0;
}
```

E) None of the answers provided

09. Which if any of the following code segments correctly implements the method **Push** ?

A)
```
void Stack::Push(int n)
{
  if ( top == MAXSIZE-1 )
    throw StackFull();
  else
  {
    data[n] = top;
    top++;
  }
}
```

B)
```
void Stack::Push(int n)
{
   if ( IsFull() )
     throw StackFull();
   else
   {
     top++;
     data[top] = n;
   }
}
```

C)
```
void Stack::Push(int n)
{
  if ( top == MAXSIZE-1 )
    throw StackEmpty();
  else
  {
    data[top] = n;
    top++;
  }
}
```

D)
```
void Stack::Push(int n)
{
   if ( top == MAXSIZE-1 )
     throw StackFull();
   else
   {
     top++;
     data[top] = n;
   }
}
```

E) None of the answers provided

10. Which if any of the following code segments correctly implements the method **Available** ?

A)
```
void Stack::Available() const
{
  return MAXSIZE;
}
```

B)
```
int Stack::Available() const
{
    return top - MAXSIZE;
}
```

C)
```
int Stack::Available() const
{
  return top;
}
```

D)
```
int Stack::Available() const
{
    return MAXSIZE - 1;
}
```

E) None of the answers provided

11. Which if any of the following code segments correctly implements the method **~Stack** ?

A)
```
Stack::~Stack()
{
   delete [ ] data;
}
```

B)
```
void ~Stack::Stack()
{
   delete [ ] data;
}
```

C)
```
Stack::~Stack()
 {
   delete data [ ];
 }
```

D)
```
Stack::~Stack()
 {
   /* No code required here */
 }
```

E) None of the answers provided

12. Which if any of the following code segments correctly implements the method **Pop** ?

A)
```
int Stack::Pop( )
{
  if ( top == -1 )
    throw StackEmpty();
  else
  {
    top--;
    return data[top];
  }
}
```

B)
```
int Stack::Pop( )
{
  if ( top == -1 )
     throw StackEmpty();
  else
  {
     top++;
     return data[top];
  }
}
```

C)
```
int Stack::Pop( )
{
  if ( top == -1 )
    throw StackEmpty();
  else
  {
    return data[top];
    top--;
  }
}
```

D)
```
int Stack::Pop( )
{
   if ( top == -1 )
     throw StackEmpty();
  else
  {
    top = data[top];
    return top;
  }
}
```

E) None of the answers provided

13. Which if any of the following code segments correctly implements the method **Peek** ?

A)
```
int Stack::Peek(int n) const
{
  if ( top == -1 )
    throw StackEmpty();
  else if ((top < n) || (n < 0))
    throw StackInvalidPeek();
  else
    return data[n - top];
}
```

B)
```
int Stack::Peek(int n) const
{
  if ( top == -1 )
     throw StackInvalidPeek();
  else if ((top < n) || (n < 0))
    throw StackEmpty();
  else
     return data[top - n];
}
```

C)
```
void Stack::Peek(int n) const
{
  if ( top == -1 )
    throw StackEmpty();
  else if ((top > n) || (n < 0))
    throw StackInvalidPeek();
  else
    return data[top - n];
}
```

D)
```
void Stack::Peek(int n) const
 {
   if ( top == -1 )
     throw StackEmpty();
  else if ((top < n) || (n < 0))
    throw StackInvalidPeek();
  else
     return data[top - n];
}
```

E) None of the answers provided

14. Assuming that the **Stack** object named **s** has been declared correctly in the client program, which of the following code segments correctly adds the integer value **5** to the top of stack **s** assuming that **s** is not full?

A) **s = 5;** C) **s.Push(5);** E) None of the answers provided

B) **s.Pop(5);** D) **s = s + 5;**

Section II – Queues **Problems 15-21** Questions on **Queue ADT** concepts

Queue ADT Function Prototypes: void Enqueue(int n); void Dequeue(int& n); Queue();

15. Suppose that a **queue** object contains the following values **Front { 1, 3, 5, 7 } Rear**.
What is the **contents of the queue** after the following operation? **Enqueue(6)**

A) **Front { 1, 3, 5, 7, 6 } Rear**
B) **Front { 6, 1, 3, 5, 7 } Rear**
C) **Front { 1, 3, 5, 6, 7 } Rear**
D) **Front { 1, 3, 5, 7 } Rear**
E) None of the answers provided

16. Suppose that a **queue** object contains the following values **Front { 1, 3, 5, 7 } Rear**.
What is the **contents of the queue** after the following operation (**x** is **int**)? **Dequeue(x)**

A) **Front { 1, 3, 5, 7 } Rear**
B) **Front { 1, 3, 5 } Rear**
C) **Front { 3, 5, 7 } Rear**
D) **Front { } Rear**
E) None of the answers provided

17. Suppose that a **queue** object contains the following values **Front { 1, 3, 5, 7 } Rear**.
What is the **contents of the queue** after the following sequence of operations are executed left-to-right (**x** is **int**)? **Enqueue(6), Dequeue(x)**

A) **Front { 3, 5, 6, 7 } Rear**
B) **Front { 1, 3, 5, 6 } Rear**
C) **Front { 6, 3, 5, 7 } Rear**
D) **Front { 1, 3, 5, 7 } Rear**
E) None of the answers provided

18. Suppose that a **queue** object contains the following values **Front { 1, 3, 5, 7 } Rear**.
What is the **contents of the queue** after the following sequence of operations are executed left-to-right (**x** is **int**)? **Dequeue(x), Enqueue(x)**

A) **Front { 1, 3, 5, 7 } Rear**
B) **Front { 7, 1, 3, 5 } Rear**
C) **Front { 3, 5, 7, 1 } Rear**
D) **Front { 3, 5 } Rear**
E) None of the answers provided

19. Suppose the client code wishes to create a **Queue** object named **Q**. Which of the following code segments successfully completes that task?

A) **Q = Queue;**
B) **~Queue Q;**
C) **Queue Q;**
D) **Queue Q(10);**
E) None of the answers provided

20. Suppose the client code includes a **Queue** object named **Q**. Which of the following code segments successfully uses the **Enqueue** method to add the integer value **3** to the object **Q** ?

A) **Enqueue(Q, 3);**
B) **Q.Enqueue(3);**
C) **Q.Enqueue('3');**
D) Both B and C
E) None of the answers provided

21. Suppose the client code includes a **Queue** object named **Q**. Which of the following code segments successfully uses the **Dequeue** method to remove the next integer value stored in **Q** and place that integer into the integer variable **W ?**

A) **W = Dequeue(Q);**
B) **W = Q.Dequeue();**
C) **Q.Dequeue(W);**
D) **W.Dequeue(Q);**
E) None of the answers provided

Problems 22-28 are based on the following **Queue class** code segment

Your solution must have no memory leaks!!

```
class QueueFull   { /* Empty exception class */ };
class QueueEmpty  { /* Empty exception class */ };
struct Node               // Node structure
{
  int   data;             // Holds an integer
  Node* next;             // Pointer to next node in the queue
};
class Queue               // Linked node implementation of Queue ADT
{
 private:
  Node* front;            // Pointer to front node of queue
  Node* rear;             // Pointer to last node of queue

 public:
  Queue();                // Default constructor initializes queue to be empty
  ~Queue();               // Deallocates all nodes in the queue
  void Add(int n);        // Enqueues value n; if full, throws QueueFull exception
  int  Remove();          // Dequeues and returns next value from the queue
                          // If empty, throws QueueEmpty exception
  bool IsFull() const;    // Returns true if queue is full; return false otherwise
  bool IsEmpty() const;   // Returns true if queue is empty; return false otherwise
  int  Size() const;      // Returns number of data values stored in queue
};  // End Queue class
```

22. Which if any of the following code segments correctly implements the Queue **constructor**?

A)
```
Queue::Queue()
{
  front = NULL;
  rear = NULL;
}
```

B)
```
Queue::Queue()
{
 front = new Node;
 rear = new Node;
}
```

C)
```
Queue::Queue() const
{
  front = NULL;
  rear = NULL;
  count = 0;
}
```

D)
```
Queue::Queue()
{
  front = rear;
}
```

E) None of the answers provided

23. Which if any of the following code segments correctly implements the **IsEmpty** method ?

A)
```
bool Queue::IsEmpty() const
{
  return (front == rear);
}
```

B)
```
bool Queue::IsEmpty()
{
   return (count == 0);
}
```

C)
```
bool Queue::IsEmpty() const
{
  return (rear == NULL);
}
```

D)
```
bool Queue::IsEmpty()
{
  return (front == NULL);
}
```

E) None of the answers provided

24. Which if any of the following code segments correctly implements the **Add()** method?

A)
```
void Queue::Add(int n)
{
  if (IsFull())
     throw QueueFull();
  else
  {
    Node* temp = new Node;
    temp->data = n;
    temp->next = NULL;
    if (rear == NULL)
      front = temp;
    else
      rear->next = temp;
  }
}
```

B)
```
void Queue::Add(int n)
{
  if (IsFull())
     throw QueueFull();
  else
  {
    Node* temp = new Node;
    temp->data = n;
    temp->next = NULL;
    if (front == NULL)
      front = temp;
    else
      rear = temp;
  }
}
```

C)
```
void Queue::Add(int n)
{
  if (IsFull())
     throw QueueFull();
  else
  {
    Node* temp = new Node;
    temp->data = NULL;
    temp->next = n;
    if (rear == NULL)
      front = temp;
    else
      rear->next = temp;
    rear = temp;
  }
}
```

D)
```
void Queue::Add(int n)
{
  if (IsFull())
     throw QueueFull();
  else
  {
    Node* temp = new Node;
    temp->data = n;
    temp->next = rear;
    if (front == NULL)
      front = temp;
    else
      rear->next = temp;
    rear = temp;
  }
}
```

E) None of the answers provided

25. Which if any of the following code segments correctly implements the Queue destructor?

A)
```
Queue::~Queue()
{
  delete front;
  delete rear;
}
```

B)
```
~Queue::Queue()
{
   delete front;
   delete rear;
}
```

C)
```
Queue::~Queue()
{
  int temp;
  while ( !IsEmpty() )
    temp = Remove();
}
```

D)
```
Queue::~Queue()
{

   delete [] front;

}
```

E) None of the answers provided

26. Which if any of the following code segments correctly implements the **IsFull()** method?

A)
```
bool Queue::IsFull() const
{
  return (rear == front);
}
```

B)
```
bool Queue::IsFull() const
{
  return (rear == NULL);
}
```

C)
```
bool Queue::IsFull() const
{
  try
  {
    Node* temp;
    return false;
  }
  catch (...)
  {
    return true;
  }
}
```

D)
```
bool Queue::IsFull() const
{
  try
  {
    Node* temp = new Node;
    return true;
  }
  catch (...)
  {
    return false;
  }
}
```

E) None of the answers provided

27. Which if any of the following code segments correctly implements the **Remove()** method?

A)
```
int Queue::Remove()
{
  if (IsEmpty())
    throw QueueEmpty();
  else
  {
    Node* temp = front;
    front = front->next;
    delete temp;
  }
}
```

B)
```
int Queue::Remove()
{
  if (IsEmpty())
    throw QueueEmpty();
  else
  {
    Node* temp = new Node;
    front = front->next;
    delete temp;
  }
}
```

C)
```
int Queue::Remove()
{
  if (IsEmpty())
    throw QueueEmpty();
  else
  {
    Node* temp = new Node;
    front = front->next;
    int value = temp->data;
    delete temp;
    return value;
  }
}
```

D)
```
int Queue::Remove()
{
  if (IsEmpty())
    throw QueueEmpty();
  else
  {
    Node* temp = front;
    front = front->next;
    int value = temp->data;
    delete temp;
    return value;
  }
}
```

E) None of the answers provided

28. Which if any of the following code segments correctly implements the **Size()** method?

A)
```
int Queue::Size() const
{
   int count = 0;
   while (front != NULL)
   {
     count++;
     front = front->next;
   }
   return count;
}
```

B)
```
int Queue::Size() const
{
   int count;
   while (rear != NULL)
   {
      count++;
      front = front->next;
   }
    return count;
}
```

C)
```
int Queue::Size() const
{
   int count = 0;
    Node* temp = front;
   while (temp != NULL)
   {
     count++;
     temp = temp->next;
   }
   return count;

}
```

D)
```
int Queue::Size() const
{
   int count = 0;
   Node* temp = front;
   while (temp != NULL)
   {
      count++;
      front = front->next;
   }
    return count;

}
```

E) None of the answers provided

Section III – Lists

29. When **inserting** a new data value into an **Unsorted Linked List**, where is the least efficient position to add the new data value?

A) After the last element currently stored in the container
B) Before the first element currently stored in the container
C) In between two elements currently stored in the container
D) None of the answers provided

30. What is the complexity of the **inserting** a data value into an **Sorted Array-Based List**?

A) **O(1)** C) **O(logN)** E) None of the answers provided
B) **O(N)** D) **$O(N^2)$**

31. What is the complexity of the constructor operation for a **Sorted Array-Based List**?

A) **O(1)** B) **O(N)** C) **$O(\log_2 N)$** D) **$O(N^2)$** E) None of the answers provided

32. What is the complexity of the destructor operation for an **Unsorted Linked List**?

A) **O(1)** B) **O(N)** C) **$O(\log_2 N)$** D) **$O(N^2)$** E) None of the answers provided