There are 10 questions and you have 35 minutes to complete the test. Some questions may have more than one correct answer, in which case you must circle all correct answers to get full marks.

(1) A B C D
(2) A B C D
(3) A B C D
(4) A B C D
(5) A B C D
(6) A B C D
(7) A B C D
(8) A B C D
(9) A B C D
(10) A B C D
Good luck
For each of the following items, please enter *one or more* answer A, B, C or D, on the sheet provided.

1. Which of the following is not an Abstract Data Type
   (a) StackCharBlock
   (b) QueueChar
   (c) Deque
   (d) LinkedList

2. Assume that MyStack has been defined as a subclass of Stack, and implements the interface Multipop. Which of the following lines of code will cause an error
   (a) MyStack m = new Stack();
   (b) Multipop m = new MyStack();
   (c) Stack m = (Stack) new MyStack();
   (d) MyStack m = (MyStack) new Multipop()

3. Chas can write code at a rate of 12 lines per minute, and he can debug his code at a rate of 4 lines per minute. At what amortized rate can Chas produce debugged code?
   (a) 3 lines per minute
   (b) 4 lines per minute
   (c) 6 lines per minute
   (d) 8 lines per minute

4. Consider a cyclic singly linked implementation of a Queue ADT that has a single instance variable referring to the link that contains the first element in the queue: Which of the following statements is true:
   (a) enqueue must take time $O(n)$, where $n$ is the size of the queue.
   (b) dequeue must take time $O(n)$, where $n$ is the size of the queue.
   (c) there is an implementation where both enqueue and dequeue take constant time.
   (d) None of the above are true.
5. Suppose that we are writing a recursive method `reverse` that takes an array of characters, and two integers `i` and `j` as arguments, and then reverses the order of characters between the indexes `i` and `j` inclusive. The method is written as:

```java
public static void reverse(char[] arr, int i, int j) throws OutOfBounds{
    if ( i<0 || j<0 || i>= arr.length || j>= arr.length){
        throw new OutOfBounds('Index out of bounds');
    }
    else{
        // Missing code
    }
}
```

Which is the correct implementation of the missing code

(a) if( i <= j ) {
    char temp = arr[i];
    arr[j] = temp;
    arr[i] = arr[j];
    return reverse(arr, i+1, j-1);
}

(b) if( i < j ) {
    char temp = arr[j];
    arr[j] = arr[i];
    arr[i] = temp;
    reverse(arr, i+1, j-1);
}

(c) if( i <= j ) {
    char temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    return reverse(arr, i+1, j-1);
}

(d) if( i < j ) {
    char temp = arr[j];
    arr[j] = arr[i];
    arr[i] = temp;
    reverse(arr, j-1, i+1);
}
6. Recall that in the RMaths program you were given a constant time method `decrement`, and a method `add` that was linear in its second argument (for example, the time for `add(x,y)` could be expressed as $by+a$ for some constants $b$ and $a$).

Consider the following recursive method:

```java
/**
 * double a positive integer
 * @param x the integer to double (assumed >=0)
 * @return double the value of x
 */
public static int double(int x) {
    if (x == 0) return 0;
    else return add(double(decrement(x)),2);
}
```

What is the (smallest) ‘big O’ that describes this method?

(a) $O(\log n)$  
(b) $O(n)$  
(c) $O(n^2)$  
(d) $O(n^3)$

7. Which of the following is a necessary and sufficient condition for showing that $f(n)$ is $O(g(n))$ for some functions $f$ and $g$?

(Note: *sufficient* means if the condition holds, then the result must hold; *necessary* means the condition *must* hold in order for the result to hold.)

(a) there exists a value $c > 0$ and $n_0 \geq 1$ such that $f(n_0) \leq cg(n_0)$  
(b) there exists a value $c > 0$ and $n_0 \geq 1$ such that $f(n_0) \leq cg(n_0)$, and $g(n)$ always increases with increasing $n$ for $n \geq n_0$  
(c) there exists a value $c > 0$ and $n_0 \geq 1$ such that $f(n_0) \leq cg(n_0)$, and $cg(n)$ is always greater than or equal to $f(n)$ for $n \geq n_0$  
(d) there exists a value $c > 0$ and $n_0 \geq 1$ such that $f(n_0) \leq cg(n_0)$, and $cg(n)$ always increases faster than $f(n)$ with increasing $n$ for $n \geq n_0$

8. Which of the following best matches the description of a *Last-In-First-Out Buffer*?

(a) A Stack  
(b) A Deque  
(c) A Queue  
(d) A List
9. The following method searches an array (stored in `block`) to see if the same item appears twice.

```java
public boolean hasMatch (int[] block) {
    boolean found = false;
    for (int i=0; i<block.length; i++) {
        for (int j=0; j<block.length; j++)
            found = found || (i != j && block[i]==block[j]);
    }
    return found;
}
```

If the function $f(x)$ describes the time performance of this method, where $x$ denotes the size of the block, which of the following is the smallest ‘big O’ for $f(x)$?

(a) $f(x)$ is $O(1)$
(b) $f(x)$ is $O(\log n)$
(c) $f(x)$ is $O(n)$
(d) $f(x)$ is $O(n^2)$
10. A block implementation of a List contains the following instance variables:

- `block` — an array of objects that stores the items in the list
- `before` — a reference to the before-first position
- `after` — a reference to the after-last position

It is used with a window class that contains one variable:

- `index` — the position in the list of the window item

The `insertBefore` method can be implemented as follows:

```java
public void insertBefore (Object e, WindowBlock w) throws OutOfBounds, Overflow {
    if (!isFull()) {
        if (!isBeforeFirst(w)) {
            // missing code
        } else throw new OutOfBounds("Inserting before start of list.");
    } else throw new Overflow("Inserting in full list.");
}
```

Which of the following is the best implementation of the missing code?

(a) for (int i=block.length-1; i>=w.index; i--) block[i+1] = block[i];
    after++;
    block[w.index] = e;
    w.index++;

(b) for (int i=after-1; i>=w.index; i--) block[i+1] = block[i];
    w.index++;
    block[w.index] = e;
    after++;

(c) for (int i=after-1; i>=w.index; i--) block[i+1] = block[i];
    w.index++;
    block[w.index] = e;
    w.index++;

(d) for (int i=w.index; i<block.length; i++) block[i+1] = block[i];
    after++;
    block[w.index] = e;
    w.index++;