QUESTION 3

a)

```
Node* createCircle(int nodes) {
   Node* firstNode = new Node();
   int count = 1;
   Node* nextNode = firstNode;
   while (count < nodes) {
      nextNode->next = new Node();
      nextNode = nextNode->next;
      count++;
   }
   nextNode->next=firstNode;
   return firstNode;
}
```

b)

```
Node* createCircle(int nodes) {
   Node* firstNode = new Node();
   int count = 1;
   Node* nextNode = firstNode;
   while (count < nodes) {
      nextNode->next = new Node();
      nextNode = nextNode->next;
      count++;
   }
   nextNode->next=firstNode;
   return firstNode;
}
```

QUESTION 4

Complete the function **SearchItem** that searches for a given value in a regular linked list and returns the position of the value in the linked list if found, otherwise returns -1. The first item in the linked list has position 1.

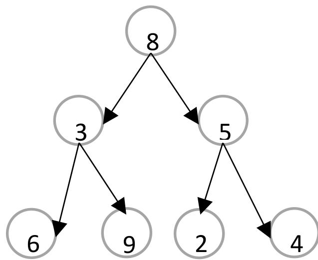
```
struct Node{
       int data;
       struct Node* next;
};


int  SearchItem(struct Node* head, int value, int index)
{


       if(head==null)
              return -1;
       else if(head->data==value)
              return index+1;
       else
              return SearchItem(head->next,value,index+1)
}
```

QUESTION 1

We will use an array in order to store a full binary tree.



Complete the given code that returns the nth node in the given level. For example, find_data(3, 2) will return 9, find_data(2, 1) will return 3, and so on. You must realize the function using the definitions given below with less than or equal to 4 code lines.

```
#define ARRAY_SIZE 7
int tree[ARRAY_SIZE];
int pow (int x, int y); //assume pow function is predefined and returns x^y.

int find_data(int level, int n){

return tree[pow(2, level-1)-1+n-1];

}
```

QUESTION 2

(5 points) One problem with an array-based implementation of a stack is the resizing we must do when the stack fills. A scheme for overcoming this drawback employs periodic resizing of the array by some constant amount c. That is, when the array of size k fills, we create a new array of size $k + c$ and copy the elements from the original array into the new one, maintaining the original order so that the stack is not corrupted. Give a complete analysis of this scheme over a sequence of n `push` operations on the stack. Assume that the stack is empty to start. Please report your answer as an average cost per `push`.

Solution: over n pushes there are n/c copying events, and that in the ith copy event, $i \cdot c$ data items are copied from the old full array to the new one. The following sum reflects the total number of copies:

$$\sum_{i=0}^{\frac{n}{c}} i \cdot c = c \sum_{i=0}^{\frac{n}{c}} i = \frac{c}{2} \cdot \frac{n}{c}(\frac{n}{c} + 1) = O(n^2)$$

Over n pushes the average cost per push is thus $O(n)$.

Rubric: 3 points for analysis, 2 points for correct answer. Two of three analysis points were awarded for any answer in which the process was decomposed appropriately into the number of copy events and the size of each, even if the values used were incorrect. Forgetting to report the average was a 1 point penalty.

QUESTION 6

a) Consider a virtual ant moves in space from point (x=0, y=0, z=0) to (x=a, y=b, z=c), where a,b and c are positive integers. **The ant moves in integer units, always in positive directions and parallel to the coordinate axis**. In other words, when it moves from (0,0,0) it can go to (1,0,0) or (0,1,0) or (0,0,1) and nowhere else. Write a **recursive** function to compute in how many **different** ways the ant can reach point (a,b,c) when it starts from (0,0,0). In other words, your function must count the number of distinct paths from (0,0,0) to (a,b,c) according to the above rules.

```
int countWays(int a, int b, int c) {
        if (x + y + z == 1)
                return 1;
        else {
                int xPath = 0, yPath = 0, zPath = 0;
                if (x > 0)
                        xPath = countWays(x - 1, y, z);
                if (y > 0)
                        yPath = countWays(x, y - 1, z);
                if (z > 0)
                        zPath = countWays(x, y, z - 1);
                return xPath + yPath + zPath;
        }
}
```

b)

```
int mod(int n, int d) {

    if (n < d)
        return n;
    else
        return mod(n-d, d);


}
```