1. Write the **recursive** function that converts an integer into its binary representation and returns the number in the binary form.

```
int  binaryConversion(int number)
{
     if (num == 0)
        return 0;
     else
        return (num%2)+10*binary_conversion(num/2);
}
```

2. Write a **recursive** function that reverses every alternate k nodes (where k is an input to the function) in a singly linked list. Using global variables or adding more parameters to the function are forbidden.

   Example:

   Inputs: 1->2->3->4->5->6->7->8->9->NULL and k=3
   Outputs: 3->2->1->4->5->6->9->8->7->NULL

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

struct node *kAltReverse(struct node *head, int k)
{

struct node *kAltReverse(struct node *head, int k)
{
  struct node* current = head;
  struct node* next;
  struct node* prev = NULL;
  int count = 0;

  /*1) reverse first k nodes of the linked list */
  while (current != NULL && count < k)
  {
    next  = current->next;
    current->next = prev;
    prev = current;
    current = next;
    count++;
  }

  /* 2) Now head points to the kth node.  So change next
     of head to (k+1)th node*/
  if(head != NULL)
    head->next = current;

  /* 3) We do not want to reverse next k nodes. So move the current pointer to skip
next k nodes */
  count = 0;
  while(count < k-1 && current != NULL )
  {
    current = current->next;
    count++;
  }

  /* 4) Recursively call for the list starting from current->next.And make rest of the list
as next of first node */
  if(current !=  NULL)
    current->next = kAltReverse(current->next, k);

  /* 5) prev is new head of the input list */
  return prev;
}
```

3. Write a function that evaluates a given prefix expression by using the given stack and the methods.

```
int stack[10];
int top=-1;
void push(int val)
{
   stack[++top]=val;
}
int pop()
{
   return(stack[top--]);
}

int evaluatePrefix(char prefix[10])
{
     int len,val,i,opr1,opr2,res;
          len=strlen(prefix);
          for(i=len-1;i>=0;i--)
          {
            switch(get_type(prefix[i]))
            {
                case 0:
                val=prefix[i]-'0';
                push(val);
                break;

           case 1: opr1=pop();
           opr2=pop();
           switch(prefix[i])
           {
                case '+': res=opr1+opr2;
                break;
                case '-': res=opr1-opr2;
                break;
                case '*': res=opr1*opr2;
                break;
                case '/': res=opr1/opr2;
                break;
           }
           push(res);

}

int get_type(char c)
{
if(c=='+'||c=='-'||c=='*'||c=='/')
return 1;
else
return 0;
```