Search an element in a Linked List (Iterative and Recursive)

Write a C function that searches a given key 'x' in a given singly linked list. The function should return true if x is present in linked list and false otherwise.

```
    bool search(Node *head, int x)
```

For example, if the key to be searched is 15 and linked list is 14->21->11->30->10, then function should return false. If key to be searched is 14, then the function should return true.

Iterative Solution

```
2) Initialize a node pointer, current = head.
3) Do following while current is not NULL
    a) current->key is equal to the key being searched return true.
    b) current = current->next
4) Return false
```

Following is iterative C implementation of above algorithm to search a given key.

```
// Iterative C program to search an element in linked list
#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct node
{
    int key;
    struct node* next;
};

/* Given a reference (pointer to pointer) to the head
  of a list and an int, push a new node on the front
  of the list. */
void push(struct node** head_ref, int new_key)
{
    /* allocate node */
    struct node* new_node =
            (struct node*) malloc(sizeof(struct node));

    /* put in the key  */
    new_node->key  = new_key;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref)    = new_node;
}

/* Counts no. of nodes in linked list */
bool search(struct node* head, int x)
{
    struct node* current = head;  // Initialize current
    while (current != NULL)
    {
        if (current->key == x)
            return true;
        current = current->next;
    }
    return false;
}

/* Drier program to test count function*/
int main()
{
    /* Start with the empty list */
    struct node* head = NULL;
    int x = 21;

    /* Use push() to construct below list
     14->21->11->30->10  */
    push(&head, 10);
    push(&head, 30);
    push(&head, 11);
    push(&head, 21);
    push(&head, 14);

    search(head, 21)? printf("Yes") : printf("No");
    return 0;
}
```

Output:

```
Yes
```

Recursive Solution

```
bool search(head, x)
1) If head is NULL, return false.
2) If head's key is same as x, return true;
2) Else return search(head->next, x)
```

Following is recursive C implementation of above algorithm to search a given key.

```
// Recursive C program to search an element in linked list
#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct node
{
    int key;
    struct node* next;
};

/* Given a reference (pointer to pointer) to the head
  of a list and an int, push a new node on the front
  of the list. */
void push(struct node** head_ref, int new_key)
{
    /* allocate node */
    struct node* new_node =
            (struct node*) malloc(sizeof(struct node));

    /* put in the key  */
    new_node->key  = new_key;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref)    = new_node;
}

/* Counts no. of nodes in linked list */
bool search(struct node* head, int x)
{
    // Base case
    if (head == NULL)
        return false;

    // If key is present in current node, return true
    if (head->key == x)
        return true;

    // Recur for remaining list
    return search(head->next, x);
}

/* Drier program to test count function*/
int main()
{
    /* Start with the empty list */
    struct node* head = NULL;
    int x = 21;

    /* Use push() to construct below list
     14->21->11->30->10  */
    push(&head, 10);
    push(&head, 30);
    push(&head, 11);
    push(&head, 21);
    push(&head, 14);

    search(head, 21)? printf("Yes") : printf("No");
    return 0;
}
```

Output:

```
Yes
```