```c
#include <stdio.h>
#include <stdlib.h>


typedef struct TreeNode {
int data;
struct TreeNode* left;
struct TreeNode* right;
} TreeNode;

TreeNode* createNode(int data) {
TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));
if (newNode == NULL) {
printf("Memory allocation failed!\n");
exit(1);
}
newNode->data = data;
newNode->left = NULL;
newNode->right = NULL;
return newNode;
}

TreeNode* insertNode(TreeNode* root, int data) {
if (root == NULL) {
return createNode(data);
}
if (data < root->data) {
root->left = insertNode(root->left, data);
} else if (data > root->data) {
root->right = insertNode(root->right, data);
}
return root;
}
```

```c
void inorderTraversal(TreeNode* root) {
if (root != NULL) {
inorderTraversal(root->left);
printf("%d ", root->data);
inorderTraversal(root->right);
}
}

void preorderTraversal(TreeNode* root) {
if (root != NULL) {
printf("%d ", root->data);
preorderTraversal(root->left);
preorderTraversal(root->right);
}
}

void postorderTraversal(TreeNode* root) {
if (root != NULL) {
postorderTraversal(root->left);
postorderTraversal(root->right);
printf("%d ", root->data);
}
}

void displayTree(TreeNode* root) {
printf("Elements in the tree (inorder traversal): ");
inorderTraversal(root);
printf("\n");
}
```

```c
int main() {
TreeNode* root = NULL;
int choice, data;
printf("\n1. Insert\n2. Inorder Traversal\n3. Preorder Traversal\n4. Postorder Traversal\n5. Display Tree\n6.
do {

printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter data to insert into the tree: ");
scanf("%d", &data);
root = insertNode(root, data);
break;
case 2:
printf("Inorder Traversal: ");
inorderTraversal(root);
printf("\n");
break;
case 3:
printf("Preorder Traversal: ");
preorderTraversal(root);
printf("\n");
break;
case 4:
printf("Postorder Traversal: ");
postorderTraversal(root);
printf("\n");
break;
case 5:
displayTree(root);
break;
case 6:
printf("Exiting...\n");
break;
default:
printf("Invalid choice! Please enter a valid option.\n");
}
} while (choice != 6);
return 0;
}
```

```
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 1
Enter data to insert into the tree: 5

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 1
Enter data to insert into the tree: 6

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 1
Enter data to insert into the tree: 4

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 1
Enter data to insert into the tree: 7

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 1
Enter data to insert into the tree: 3

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 5
Elements in the tree (inorder traversal): 3 4 5 6 7
```

```
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 2
Inorder Traversal: 3 4 5 6 7

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 3
Preorder Traversal: 5 4 3 6 7

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 4
Postorder Traversal: 3 4 7 6 5

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display Tree
6. Exit
Enter your choice: 6
Exiting...
```

Leet Code Problem:

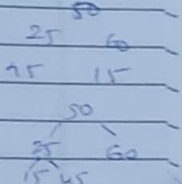
SURYA Gold
Date Page

Write a program a) to construct binary search tree b) traverse the tree using inorder, postorder, preorder c) display the elements in the tree

```
#include <stdio.h>
#include <stdlib.h>

struct BST{
        int data;
        struct BST *left, *right;
};
struct BST *root = NULL, *temp;

void create(){
  temp = (struct BST *) malloc (sizeof(struct BST));
  printf(" Enter data");
  scanf ("%d", &temp->data);
  temp->left = temp->right = NULL;
  return temp;
}

void insert (struct BST *root, struct
  BST *temp)
{
  if (temp->data < root->data)
     if (root->left != NULL)
        insert(root->left, temp);
     else
        root->left = temp;
  if (temp->data > root->data)
     if (root->right != NULL)
        insert (root->right, temp);
     else
        root->right = temp;
}

void inorder (struct BST *root)
{
  if (root != NULL)
  {
     inorder (root->left);
     printf ("%d", root->data);
     inorder (root->right);
  }
}

void postorder (struct BST *root)
{
  if (root != NULL)
  {
     postorder (root->left)
     postorder (root->right);
     printf ("%d", root->data);
  }
}
void preorder (struct BST *root)
{ if (root != NULL)
  { printf ("%d", root->data);
    preorder (root->left);
    preorder (root->right);
  }
}
```

```
void main()
{
int choice, ch;
printf("Enter operation : create \n
2. display using inorder \n
3 display using postorder \n
display using preorder \n
-1 to end \n");
while(1)
{
printf("Enter operation :");
scanf("%d", &choice");
if (choice == -1)
return;
}
else
switch (choice)
{
case 1: do{
temp = create();
if (root == NULL)
root = temp;
else
insert(root, temp);
printf("Do you want to enter more
y/n ?");
getchar();
case 1: temp = create();
if (root == NULL);
root = temp;
else
insert (root, temp);
break;
```

```
case 2: inorder (root);
break;
case 3: postorder (root);
break;
case 4: preorder (root);
break;
default: exit(0)
}
}
}
```



Output.

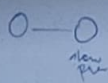
```
Enter operation 1
Enter data 50
Enter operation 1
Enter data 25
Enter operation 1
Enter data 15
Enter operation 1
Enter data 45
Enter operation 1
Enter data 60
Enter operation 2
15 25 45 50 60

Enter operation 3

15 45 25 60 50

Enter operation 4
25 15 45 60
```

SURYA Gold
Date Page

876 To delete the middle element

```
struct ListNode * middle(struct * head)
{
    struct ListNode * temp = NULL;
    struct ListNode * test = NULL;
    if (head == NULL)
        return;
    else if (head->next == NULL)
        free(head);
    else
        int i=0, n=0;
        while (temp != NULL)
        {
            i++;
            temp = temp->next;
        }
        temp = head;
        while (n != i/2)
        {
            test = temp;
            temp = temp->next;
            n++;
        }
        test->next = temp->next;
        free(temp);
}
```

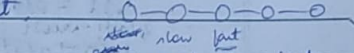
Output:
5 6 7 8 9 2 1
5 6 8 3



SURYA Gold
Date Page

```
struct ListNode * slow = head;
struct ListNode * fast, *prev = head;
while (fast->next!=NULL && fast!=NULL)
{
    fast = fast->next->next
    prev = slow;
    slow = slow->next;
}
prev->next = slow->next;
free(slow);
```

To access the middle
Odd Even linked list.




```
struct ListNode * even, * odd, * evenhead
    * oddhead;
while (odd->next!=NULL && odd!=NULL)
{
    odd = odd->next->next
    prev = even;
    even = even->next;
```

328 Odd even linked list

(1) (2) (3) (4) (5)

```
struct node *even, *odd, *evenhead, *oddhead;
odd = head;
even = head->next;
evenhead = even;
oddhead = odd;

while (even != NULL && even->next != NULL)
{
    odd->next = even->next;
    odd = odd->next;
    even->next = odd->next;
    even = even->next;
}

odd->next = evenhead;
return head;
}
```

Output:

Given: (1) (2) (3) (4) (5)

(1) (3) (5) (2) (4)

20.02.24

Description | Editorial | Pick one | Submissions

328. Odd Even Linked List

Solved

Medium | Topics | Companies

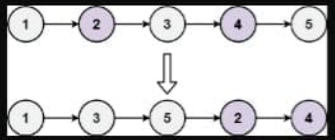
Given the `head` of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list*.

The **first** node is considered **odd**, and the **second** node is **even**, and so on.

Note that the relative order inside both the even and odd groups should remain as it was in the input.

You must solve the problem in `O(1)` extra space complexity and `O(n)` time complexity.

Example 1:




```
Input: head = [1,2,3,4,5]
Output: [1,3,5,2,4]
```

Example 2:

9.6K | 107

Code

C | Auto

```
 *     struct ListNode *next;
 * };
 */
struct ListNode* oddEvenList(struct ListNode* head) {
    if (head == NULL || head->next == NULL) {
        return head;
    }

    struct ListNode *odd = head;
    struct ListNode *even = head->next;
    struct ListNode *evenHead = even;
    struct ListNode *oddHead = odd;

    while (even != NULL && even->next != NULL) {
        odd->next = even->next;
        odd = odd->next;

        if (odd != NULL) {
            even->next = odd->next;
            even = even->next;
        }
    }

    odd->next = evenHead;
```

Saved to local

Ln 28, Col 26

Testcase | Test Result

Accepted Runtime: 5 ms

Description | Editorial | Solutions | Submissions

← All Submissions

Accepted

Vignesh_Bhaskar submitted at Feb 19, 2024 19:16

Editorial

Runtime

4 ms

Beats 57.84% of users with C

Memory

6.82 MB

Beats 56.10% of users with C

Code | C

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
```

Code

C Auto

```
        return head;
    }

```

Saved to local

Ln 1, Col 1

Testcase | Test Result

Accepted Runtime: 5 ms

Case 1 Case 2

Input

head =
[1,2,3,4,5]

Output

[1,3,5,2,4]

Expected

[1,3,5,2,4]

Contribute a testcase

Description | Editorial | Solutions | Submissions

2095. Delete the Middle Node of a Linked List

Solved

Medium | Topics | Companies | Hint

You are given the `head` of a linked list. **Delete** the **middle node**, and return *the* `head` *of the modified linked list.*

The **middle node** of a linked list of size `n` is the `⌊n / 2⌋`th node from the **start** using **0-based indexing**, where `⌊x⌋` denotes the largest integer less than or equal to `x`.

- For `n` = `1`, `2`, `3`, `4`, and `5`, the middle nodes are `0`, `1`, `1`, `2`, and `2`, respectively.

Example 1:

Input: head = [1,3,4,7,1,2,6]
Output: [1,3,4,1,2,6]
Explanation:
The above figure represents the given linked list. The indices of the nodes are written below.
Since n = 7, node 3 with value 7 is the middle node, which is marked in red.
We return the new list after removing this node.

4K | 30

Code

C | Auto

```c
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* deleteMiddle(struct ListNode* head) {
    struct ListNode *fast , *slow, *prev;
    fast=head;
    slow=head;
    while(fast != NULL && fast->next != NULL)
    {
        fast=fast->next->next;
        prev=slow;
        slow=slow->next;
    }
    prev->next=slow->next;
    free(slow);
    return head;
}
```

Saved to local | Ln 1, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

Description | Editorial | Solutions | Submissions

2095. Delete the Middle Node of a Linked List

Solved

Medium | Topics | Companies | Hint

You are given the `head` of a linked list. **Delete** the **middle node**, and return *the* `head` *of the modified linked list.*

The **middle node** of a linked list of size `n` is the $\lfloor n / 2 \rfloor^{th}$ node from the **start** using **0-based indexing**, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to `x`.

- For `n` = `1`, `2`, `3`, `4`, and `5`, the middle nodes are `0`, `1`, `1`, `2`, and `2`, respectively.

Example 1:


```
Input: head = [1,3,4,7,1,2,6]
Output: [1,3,4,1,2,6]
Explanation:
The above figure represents the given linked list. The indices of the nodes are
written below.
Since n = 7, node 3 with value 7 is the middle node, which is marked in red.
We return the new list after removing this node.
```

Example 2:

Code

Testcase | Test Result

Accepted Runtime: 5 ms

• Case 1 • Case 2 • Case 3

Input

head =
[1,3,4,7,1,2,6]

Output

[1,3,4,1,2,6]

Expected

[1,3,4,1,2,6]

Contribute a testcase