**Linked List FAQ**

# Reverse a Singly Linked List

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

using namespace std;

/\* Link list Node \*/

struct Node {

int data;

struct Node\* next;

};

struct Node \*reverse(struct Node \*head);

void append(struct Node\*\* head\_ref, struct Node \*\*tail\_ref, int new\_data) {

struct Node\* new\_node = new Node;

new\_node->data = new\_data;

new\_node->next = NULL;

if (\*head\_ref == NULL)

\*head\_ref = new\_node;

else

(\*tail\_ref)->next = new\_node;

\*tail\_ref = new\_node;

}

void printList(struct Node \*head) {

struct Node \*temp = head;

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->next;

}

}

/\* Driver program to test above function\*/

int main() {

int T,n,l;

cin>>T;

while(T--) {

struct Node \*head = NULL, \*tail = NULL;

cin>>n;

for (int i=1; i<=n; i++) {

cin>>l;

append(&head, &tail, l);

}

head = reverse(head);

printList(head);

cout << endl;

}

return 0;

}

/\*Please note that it's Function problem i.e.

you need to write your solution in the form of Function(s) only.

Driver Code to call/invoke your function is mentioned above.\*/

// Should reverse list and return new head.

**Node\* reverse(Node \*head) {**

**// Your code here**

**Node \* prev = NULL;**

**Node \* curr = NULL;**

**Node \* temp = NULL;**

**curr = head;**

**while(NULL != curr) {**

**temp = (curr->next);**

**(curr->next) = prev;**

**prev = curr;**

**curr = temp;**

**}**

**if(NULL != prev) {**

**head = prev;**

**}**

**else {**

**cout << "Error in linked list reverse" << endl;**

**}**

**return head;**

**}**

Input:

2

6

1 2 3 4 5 6

5

1 2 3 4 5

Output:

6 5 4 3 2 1

5 4 3 2 1

# Deletion and Reverse in Circular Linked List

Given a Circular Linked List of size N, your task is to delete the given node (excluding the first and last node) in circular linked list and then print the reverse of the circular linked list.

#include <bits/stdc++.h>

using namespace std;

/\* structure for a node \*/

struct Node {

int data;

struct Node \*next;

};

/\* Function to delete a given node from the list \*/

void deleteNode(struct Node \*head, int key);

/\* Function to reverse the linked list \*/

static void reverse(struct Node\*\* head\_ref);

/\* Function to insert a node at the beginning of a Circular linked list \*/

void push(struct Node \*\*head\_ref, int data) {

// Create a new node and make head as next of it.

struct Node \*ptr1 = (struct Node \*)malloc(sizeof(struct Node));

ptr1->data = data;

ptr1->next = \*head\_ref;

/\* If linked list is not NULL then set the

next of last node \*/

if (\*head\_ref != NULL) {

// Find the node before head and update next of it.

struct Node \*temp = \*head\_ref;

while (temp->next != \*head\_ref)

temp = temp->next;

temp->next = ptr1;

}

else

ptr1->next = ptr1; /\*For the first node \*/

\*head\_ref = ptr1;

}

/\* Function to print nodes in a given circular linked list \*/

void printList(struct Node \*head) {

struct Node \*temp = head;

if (head != NULL) {

do {

printf("%d ", temp->data);

temp = temp->next;

}

while (temp != head);

}

printf("\n");

}

/\* Driver program to test above functions \*/

int main() {

/\* Initialize lists as empty \*/

int t,a,delnode;

cin>>t;

while(t--) {

struct Node \*head = NULL;

int n;

cin>>n;

while(n--) {

cin>>a;

push(&head,a);

}

/\* Created linked list will be 2->5->7->8->10 \*/

cin>>delnode;

deleteNode(head, delnode);

// reverse(&head);

printList(head);

}

return 0;

}

/\*Please note that it's Function problem i.e.

you need to write your solution in the form of Function(s) only.

Driver Code to call/invoke your function is mentioned above.\*/

/\* structure for a node

struct Node {

int data;

struct Node \*next;

};

\*/

/\* Function to delete a given node from the list \*/

**void deleteNode(struct Node \*head, int key) {**

**// Your code goes here**

**struct Node \* prev = NULL;**

**struct Node \* curr = head;**

**while(key != (curr->data)) {**

**prev = curr;**

**curr = (curr->next);**

**}**

**if(key == (curr->data)) {**

**(prev->next) = (curr->next);**

**(curr->next) = NULL;**

**free(curr);**

**}**

**}**

/\* Function to reverse the linked list \*/

**static void reverse(struct Node\*\* head\_ref) {**

**// Your code goes here**

**struct Node \* prev = NULL;**

**struct Node \* curr = NULL;**

**struct Node \* temp = NULL;**

**curr = (\*head\_ref);**

**while((\*head\_ref) != temp) {**

**temp = (curr->next);**

**(curr->next) = prev;**

**prev = curr;**

**curr = temp;**

**}**

**if( ((\*head\_ref) == temp) && ((\*head\_ref)==(curr->next)) ) {**

**((\*head\_ref)->next) = curr;**

**(\*head\_ref) = curr;**

**}**

**else {**

**cout << "Error in reverse" << endl;**

**}**

**}**

Input:

2

7

43 230 374 422 920 785 538

422

23

645 591 506 140 955 787 670 83 543 465 198 508 356 805 349 612 623 829 300 344 747 569 341

670

Output:

538 785 920 374 230 43

341 569 747 344 300 829 623 612 349 805 356 508 198 465 543 83 787 955 140 506 591 645

# Pairwise swap elements of a given linked list

Given a singly linked list, write a function to swap elements pairwise.

Example, if the linked list is 1->2->3->4->5 then

the function should change it to 2->1->4->3->5

#include<stdio.h>

#include<stdlib.h>

/\* A linked list node \*/

struct Node {

int data;

struct Node \*next;

};

#define SUCCESS 0

#define FAILURE -1

int push(struct Node\*\* head\_ref, int new\_data);

int pairWiseSwap(struct Node \*head);

int printList(struct Node \*node);

int push(struct Node\*\* head\_ref, int new\_data) {

// printf("push\n");

struct Node \* temp = NULL;

temp = (struct Node \*)malloc(sizeof(struct Node));

if(NULL == temp) {

fprintf(stdout, "Error in memory allocation");

return FAILURE;

}

(temp -> data) = new\_data;

if(NULL == (\*head\_ref)) {

(temp -> next) = NULL;

(\*head\_ref) = temp;

}

else {

(temp -> next) = (\*head\_ref);

(\*head\_ref) = temp;

}

return SUCCESS;

}

**int pairWiseSwap(struct Node \*head) {**

**struct Node \* prev = NULL;**

**struct Node \* curr = NULL;**

**int temp = 0;**

**if(NULL == head) {**

**return FAILURE;**

**}**

**if(NULL == (head -> next)) {**

**return SUCCESS;**

**}**

**prev = head;**

**curr = (head -> next);**

**while(NULL != curr) {**

**temp = (curr -> data);**

**(curr -> data) = (prev -> data);**

**(prev -> data) = temp;**

**prev = (curr -> next); // prev = ((prev -> next) -> next);**

**if(NULL != prev) {**

**curr = (prev -> next); // curr = ((curr -> next) -> next);**

**}**

**else {**

**curr = NULL; // since prev is NULL curr will also become NULL**

**}**

**}**

**return SUCCESS;**

**}**

int printList(struct Node \*head) {

printf("printList\n");

while(NULL != (head -> next)) {

fprintf(stdout, "%d ", (head -> data));

head = (head -> next);

}

if(NULL == (head -> next)) {

fprintf(stdout, "%d", (head -> data));

}

return SUCCESS;

}

int main() {

struct Node \*start = NULL;

int a, x, i;

scanf("%d", &a);

for (i = 0; i < a; i++) {

scanf("%d",&x);

push(&start, x);

}

printf("List: "); printList(start); printf("\n");

pairWiseSwap(start);

printf("List: "); printList(start); printf("\n");

return 0;

}

Input:

5

5 4 3 2 1

Output:

List: 1 2 3 4 5

List: 2 1 4 3 5

# Swap nodes in a linked list without swapping data

Given a linked list and two keys in it, swap nodes for two given keys. Nodes should be swapped by changing links. Swapping data of nodes may be expensive in many situations when data contains many fields.

Example, if the linked list is 1->2->3->4->5->6 then

the function should change it to 2->1->4->3->6->5

#include<stdio.h>

#include<stdlib.h>

/\* A linked list node \*/

struct Node {

int data;

struct Node \*next;

};

#define SUCCESS 0

#define FAILURE -1

int push(struct Node\*\* head\_ref, int new\_data);

int pairWiseSwap(struct Node \*\*head);

int printList(struct Node \*node);

int push(struct Node\*\* head\_ref, int new\_data) {

// printf("push\n");

struct Node \* temp = NULL;

temp = (struct Node \*)malloc(sizeof(struct Node));

if(NULL == temp) {

fprintf(stdout, "Error in memory allocation");

return FAILURE;

}

(temp -> data) = new\_data;

if(NULL == (\*head\_ref)) {

(temp -> next) = NULL;

(\*head\_ref) = temp;

}

else {

(temp -> next) = (\*head\_ref);

(\*head\_ref) = temp;

}

return SUCCESS;

}

**int pairWiseSwap(struct Node \*\*head) {**

**struct Node \* prev = NULL;**

**struct Node \* curr = NULL;**

**struct Node \* temp = NULL;**

**if(NULL == (\*head)) {**

**return FAILURE;**

**}**

**if(NULL == ((\*head) -> next)) {**

**return SUCCESS;**

**}**

**// swap first two nodes separately to take care of head**

**prev = (\*head);**

**curr = ((\*head) -> next);**

**(prev -> next) = (curr -> next);**

**(curr -> next) = prev;**

**(\*head) = curr;**

**// swap all other nodes**

**temp = prev;**

**prev = (prev -> next);**

**if(NULL != prev) {**

**curr = (prev -> next);**

**}**

**else {**

**curr = NULL;**

**}**

**while(NULL != curr) {**

**(prev -> next) = (curr -> next);**

**(curr -> next) = prev;**

**if(NULL != temp) {**

**(temp -> next) = curr;**

**}**

**temp = prev;**

**prev = (prev -> next);**

**if(NULL != prev) {**

**curr = (prev -> next);**

**}**

**else {**

**curr = NULL;**

**}**

**}**

**return SUCCESS;**

**}**

int printList(struct Node \*head) {

while(NULL != (head -> next)) {

fprintf(stdout, "%d ", (head -> data));

head = (head -> next);

}

if(NULL == (head -> next)) {

fprintf(stdout, "%d", (head -> data));

}

return SUCCESS;

}

int main() {

struct Node \*start = NULL;

int a, x, i;

scanf("%d", &a);

for (i = 0; i < a; i++) {

scanf("%d",&x);

push(&start, x);

}

printf("List: "); printList(start); printf("\n");

pairWiseSwap(&start);

printf("List: "); printList(start); printf("\n");

return 0;

}

Input:

6

6 5 4 3 2 1

Output:

List: 1 2 3 4 5 6

List: 2 1 4 3 6 5

# Find the middle of a given linked list

Given a singly linked list, find middle of the linked list. For example, if given linked list is 1->2->3->4->5 then output should be 3.

If there are even nodes, then there would be two middle nodes, we need to print second middle element. For example, if given linked list is 1->2->3->4->5->6 then output should be 4.

## Count no of nodes

Traverse the whole linked list and count the no. of nodes. Now traverse the list again till count/2 and return the node at count/2.

## Fast Pointer and Slow Pointer

Traverse linked list using two pointers. Move one pointer by one and other pointer by two. When the fast pointer reaches end slow pointer will reach middle of the linked list.

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

using namespace std;

/\* Link list Node \*/

struct Node {

int data;

struct Node\* next;

};

void print\_Middle\_Node(Node \*head);

void append(struct Node\*\* head\_ref, struct Node \*\*tail\_ref, int new\_data);

void printList(struct Node \*head);

/\* Driver program to test above function\*/

int main() {

int n, l;

struct Node \*head = NULL, \*tail = NULL;

cin>>n;

for (int i=1; i<=n; i++) {

cin>>l;

append(&head, &tail, l);

}

printList(head);

print\_Middle\_Node(head); cout << endl;

return 0;

}

void append(struct Node\*\* head\_ref, struct Node \*\*tail\_ref, int new\_data) {

struct Node\* new\_node = new Node;

new\_node->data = new\_data;

new\_node->next = NULL;

if (\*head\_ref == NULL)

\*head\_ref = new\_node;

else

(\*tail\_ref)->next = new\_node;

\*tail\_ref = new\_node;

}

void printList(struct Node \*head) {

struct Node \*temp = head;

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->next;

}

cout << endl;

}

/\*Please note that it's Function problem i.e. you need to write your solution in the form of Function(s) only. Driver Code to call/invoke your function is mentioned above.\*/

// Should reverse list and return new head.

**void print\_Middle\_Node(Node \*head) {**

**Node \*fast\_ptr = NULL;**

**Node \*slow\_ptr = NULL;**

**if(NULL == head)**

**return;**

**slow\_ptr = head;**

**fast\_ptr = head;**

**while((NULL != fast\_ptr) && (NULL != (fast\_ptr -> next))) {**

**slow\_ptr = slow\_ptr -> next;**

**fast\_ptr = fast\_ptr -> next -> next;**

**}**

**if(NULL != slow\_ptr) {**

**cout << "Middle element: " << (slow\_ptr -> data) << endl;**

**}**

**return;**

**}**

Output:

2 3 4 5 6 7

Middle element: 5

## Using Only Slow Pointer

Initialize mid element as head and initialize a counter as 0. Traverse the list from head, while traversing increment the counter and change mid to mid->next whenever the counter is odd. So the mid will move only half of the total length of the list.

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

using namespace std;

/\* Link list Node \*/

struct Node {

int data;

struct Node\* next;

};

void print\_Middle\_Node(Node \*head);

void append(struct Node\*\* head\_ref, struct Node \*\*tail\_ref, int new\_data);

void printList(struct Node \*head);

/\* Driver program to test above function\*/

int main() {

int n, l;

struct Node \*head = NULL, \*tail = NULL;

cin>>n;

for (int i=1; i<=n; i++) {

cin>>l;

append(&head, &tail, l);

}

printList(head);

print\_Middle\_Node(head);

cout << endl;

return 0;

}

void append(struct Node\*\* head\_ref, struct Node \*\*tail\_ref, int new\_data) {

struct Node\* new\_node = new Node;

new\_node->data = new\_data;

new\_node->next = NULL;

if (\*head\_ref == NULL)

\*head\_ref = new\_node;

else

(\*tail\_ref)->next = new\_node;

\*tail\_ref = new\_node;

}

void printList(struct Node \*head) {

struct Node \*temp = head;

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->next;

}

cout << endl;

}

/\*Please note that it's Function problem i.e. you need to write your solution in the form of Function(s) only. Driver Code to call/invoke your function is mentioned above.\*/

// Should reverse list and return new head.

**void print\_Middle\_Node(Node \*head) {**

**Node \*slow\_ptr = NULL;**

**int count = 0;**

**if(NULL == head)**

**return;**

**slow\_ptr = head;**

**while(NULL != head) {**

**if(1 & count) {**

**slow\_ptr = slow\_ptr -> next;**

**}**

**count = count + 1;**

**head = head -> next;**

**}**

**if(NULL != slow\_ptr) {**

**cout << "Middle element: " << (slow\_ptr -> data) << endl;**

**}**

**return;**

**}**

Output:

2 3 4 5 6 7

Middle element: 5

# END