**DSA ASSIGNMENT-4**

**Name:-** Manali Naik

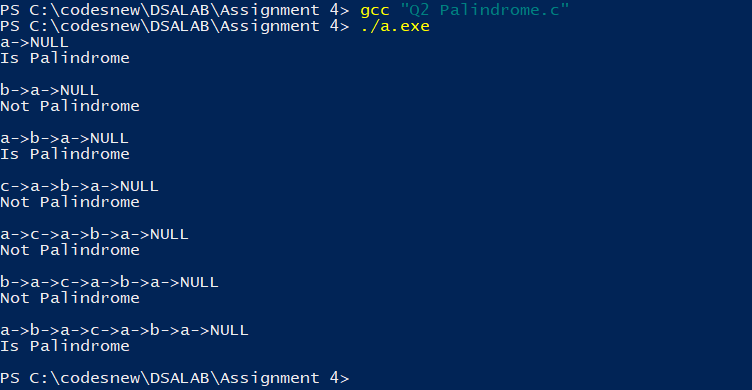
**Roll No.:-**1706333

**Branch:-** IT

**Class:-** IT-4

**Q.1. Split the single linked list at the middle without calculating the**

**length.**



#include<stdio.h>

#include<stdlib.h>

/\* Link list node \*/

struct Node

{

int data;

struct Node\* next;

};

/\* Function to get the middle of the linked list\*/

void printMiddle(struct Node \*head)

{

struct Node \*slow\_ptr = head;

struct Node \*fast\_ptr = head;

if (head!=NULL)

{

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

{

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

slow\_ptr = slow\_ptr->next;

}

printf("The middle element is [%d]\n\n", slow\_ptr->data);

}

}

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

{

/\* allocate node \*/

struct Node\* new\_node =(struct Node\*) malloc(sizeof(struct Node));

/\* put in the data \*/

new\_node->data = new\_data;

/\* 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;

}

// A utility function to print a given linked list

void printList(struct Node \*ptr)

{

while (ptr != NULL)

{

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

ptr = ptr->next;

}

printf("NULL\n");

}

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

int main()

{

/\* Start with the empty list \*/

struct Node\* head = NULL;

int i;

for (i=5; i>0; i--)

{

push(&head, i);

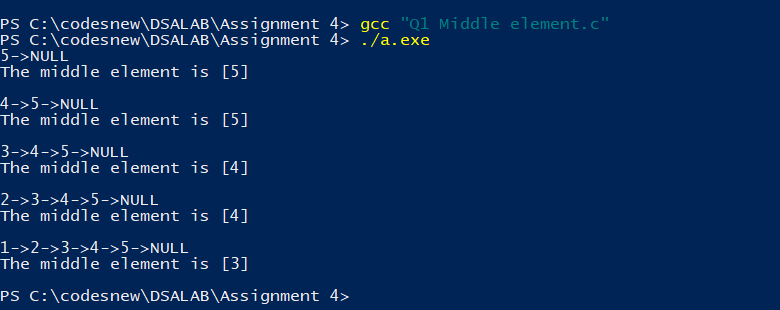
printList(head);

printMiddle(head);

}

return 0;

}



**Q.2. Check the single linked list palindrome or not.**

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

/\* Link list node \*/

struct Node

{

char data;

struct Node\* next;

};

void reverse(struct Node\*\*);

bool compareLists(struct Node\*, struct Node \*);

/\* Function to check if given linked list is

palindrome or not \*/

bool isPalindrome(struct Node \*head)

{

struct Node \*slow\_ptr = head, \*fast\_ptr = head;

struct Node \*second\_half, \*prev\_of\_slow\_ptr = head;

struct Node \*midnode = NULL; // To handle odd size list

bool res = true; // initialize result

if (head!=NULL && head->next!=NULL)

{

/\* Get the middle of the list. Move slow\_ptr by 1

and fast\_ptrr by 2, slow\_ptr will have the middle

node \*/

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

{

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

/\*We need previous of the slow\_ptr for

linked lists with odd elements \*/

prev\_of\_slow\_ptr = slow\_ptr;

slow\_ptr = slow\_ptr->next;

}

/\* fast\_ptr would become NULL when there are even elements in list.

And not NULL for odd elements. We need to skip the middle node

for odd case and store it somewhere so that we can restore the

original list\*/

if (fast\_ptr != NULL)

{

midnode = slow\_ptr;

slow\_ptr = slow\_ptr->next;

}

// Now reverse the second half and compare it with first half

second\_half = slow\_ptr;

prev\_of\_slow\_ptr->next = NULL; // NULL terminate first half

reverse(&second\_half); // Reverse the second half

res = compareLists(head, second\_half); // compare

/\* Construct the original list back \*/

reverse(&second\_half); // Reverse the second half again

// If there was a mid node (odd size case) which

// was not part of either first half or second half.

if (midnode != NULL)

{

prev\_of\_slow\_ptr->next = midnode;

midnode->next = second\_half;

}

else prev\_of\_slow\_ptr->next = second\_half;

}

return res;

}

/\* Function to reverse the linked list Note that this

function may change the head \*/

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

{

struct Node\* prev = NULL;

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

struct Node\* next;

while (current != NULL)

{

next = current->next;

current->next = prev;

prev = current;

current = next;

}

\*head\_ref = prev;

}

/\* Function to check if two input lists have same data\*/

bool compareLists(struct Node\* head1, struct Node \*head2)

{

struct Node\* temp1 = head1;

struct Node\* temp2 = head2;

while (temp1 && temp2)

{

if (temp1->data == temp2->data)

{

temp1 = temp1->next;

temp2 = temp2->next;

}

else return 0;

}

/\* Both are empty reurn 1\*/

if (temp1 == NULL && temp2 == NULL)

return 1;

/\* Will reach here when one is NULL

and other is not \*/

return 0;

}

/\* Push a node to linked list. Note that this function

changes the head \*/

void push(struct Node\*\* head\_ref, char new\_data)

{

/\* allocate node \*/

struct Node\* new\_node =(struct Node\*) malloc(sizeof(struct Node));

/\* put in the data \*/

new\_node->data=new\_data;

/\* link the old list off the new node \*/

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

/\* move the head to pochar to the new node \*/

(\*head\_ref)=new\_node;

}

// A utility function to print a given linked list

void printList(struct Node \*ptr)

{

while (ptr!=NULL)

{

printf("%c->",ptr->data);

ptr=ptr->next;

}

printf("NULL\n");

}

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

int main()

{

/\* Start with the empty list \*/

struct Node\* head = NULL;

char str[] = "abacaba";

int i;

for (i = 0; str[i] != '\0'; i++)

{

push(&head, str[i]);

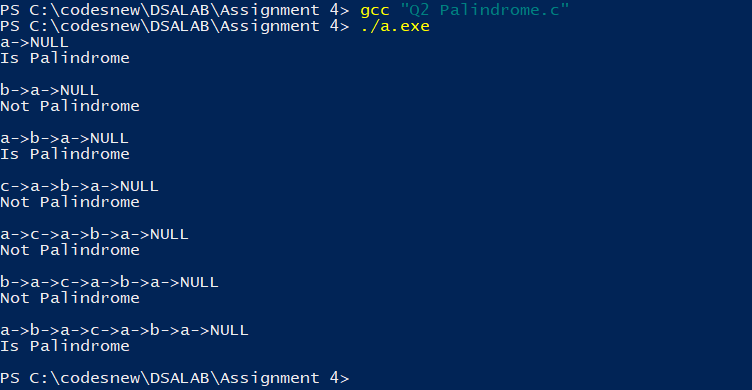
printList(head);

isPalindrome(head)? printf("Is Palindrome\n\n"):printf("Not Palindrome\n\n");

}

return 0;

}



**Q.3. Display the kth node of the single linked list from the last.**

#include<stdio.h>

#include<stdlib.h>

/\* Link list node \*/

struct Node

{

int data;

struct Node\* next;

};

/\* Function to get the nth node from the last of a linked list\*/

void printNthFromLast(struct Node \*head, int n)

{

struct Node \*main\_ptr = head;

struct Node \*ref\_ptr = head;

int count = 0;

if(head != NULL)

{

while( count < n )

{

if(ref\_ptr == NULL)

{

printf("%d is greater than the no. of "

"nodes in list", n);

return;

}

ref\_ptr = ref\_ptr->next;

count++;

} /\* End of while\*/

while(ref\_ptr != NULL)

{

main\_ptr = main\_ptr->next;

ref\_ptr = ref\_ptr->next;

}

printf("Node no. %d from last is %d ",

n, main\_ptr->data);

}

}

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

{

/\* allocate node \*/

struct Node\* new\_node =(struct Node\*) malloc(sizeof(struct Node));

/\* put in the data \*/

new\_node->data = new\_data;

/\* 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;

}

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

int main()

{

/\* Start with the empty list \*/

struct Node\* head = NULL;

push(&head, 20);

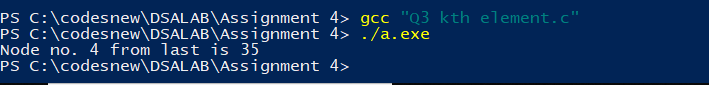
push(&head, 4);

push(&head, 15);

push(&head, 35);

printNthFromLast(head, 4);

}



**Q.4. Merge Two sorted linked list.**

#include<stdio.h>

#include<stdlib.h>

#include<assert.h>

/\* Link list node \*/

struct Node

{

int data;

struct Node\* next;

};

/\* pull off the front node of the source and put it in dest \*/

void MoveNode(struct Node\*\* destRef, struct Node\*\* sourceRef);

/\* Takes two lists sorted in increasing order, and splices

their nodes together to make one big sorted list which

is returned. \*/

struct Node\* SortedMerge(struct Node\* a, struct Node\* b)

{

/\* a dummy first node to hang the result on \*/

struct Node dummy;

/\* tail points to the last result node \*/

struct Node\* tail = &dummy;

/\* so tail->next is the place to add new nodes

to the result. \*/

dummy.next = NULL;

while (1)

{

if (a == NULL)

{

/\* if either list runs out, use the

other list \*/

tail->next = b;

break;

}

else if (b == NULL)

{

tail->next = a;

break;

}

if (a->data <= b->data)

MoveNode(&(tail->next), &a);

else

MoveNode(&(tail->next), &b);

tail = tail->next;

}

return(dummy.next);

}

/\* UTILITY FUNCTIONS \*/

/\* MoveNode() function takes the node from the front of the

source, and move it to the front of the dest.

It is an error to call this with the source list empty.

Before calling MoveNode():

source == {1, 2, 3}

dest == {1, 2, 3}

Affter calling MoveNode():

source == {2, 3}

dest == {1, 1, 2, 3} \*/

void MoveNode(struct Node\*\* destRef, struct Node\*\* sourceRef)

{

/\* the front source node \*/

struct Node\* newNode = \*sourceRef;

assert(newNode != NULL);

/\* Advance the source pointer \*/

\*sourceRef = newNode->next;

/\* Link the old dest off the new node \*/

newNode->next = \*destRef;

/\* Move dest to point to the new node \*/

\*destRef = newNode;

}

/\* Function to insert a node at the beginging of the

linked list \*/

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

{

/\* allocate node \*/

struct Node\* new\_node = (struct Node\*) malloc(sizeof(struct Node));

/\* put in the data \*/

new\_node->data = new\_data;

/\* 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;

}

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

void printList(struct Node \*node)

{

while (node!=NULL)

{

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

node = node->next;

}

}

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

int main()

{

/\* Start with the empty list \*/

struct Node\* res = NULL;

struct Node\* a = NULL;

struct Node\* b = NULL;

/\* Let us create two sorted linked lists to test

the functions

Created lists, a: 5->10->15, b: 2->3->20 \*/

push(&a, 15);

push(&a, 10);

push(&a, 5);

push(&b, 20);

push(&b, 3);

push(&b, 2);

/\* Remove duplicates from linked list \*/

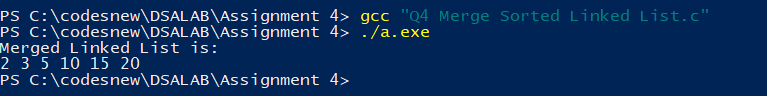
res = SortedMerge(a, b);

printf("Merged Linked List is: \n");

printList(res);

return 0;

}



**Q.5. Implement Stack using queue.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\* next;

};

struct queue

{

struct node\* st1,\*st2;

};

void initQueue(struct queue\*\* q)

{

(\*q)->st1=NULL;

(\*q)->st2=NULL;

}

void enqueue(struct queue\*\* q,int n)

{

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

temp->data=n;

temp->next=(\*q)->st1;

(\*q)->st1=temp;

return;

}

int dequeue(struct queue\*\* q)

{

struct node\* s1=(\*q)->st1;

struct node\* s2=(\*q)->st2;

if((\*q)->st2==NULL)

{

if((\*q)->st1==NULL)

{

printf("Empty Queue\n");

return -999;

}

while((\*q)->st1!=NULL)

{

struct node\* t=(\*q)->st1->next;

(\*q)->st1->next=(\*q)->st2;

(\*q)->st2=(\*q)->st1;

(\*q)->st1=t;

}

}

int v=(\*q)->st2->data;

(\*q)->st2=(\*q)->st2->next;

return v;

}

int main()

{

struct queue\* q=(struct queue\*)malloc(sizeof(struct queue));

initQueue(&q);

while(1)

{

printf("1.Enqueue\n2.Dequeue\n3.exit\nchoose: ");

int op;

scanf("%d",&op);

if(op==3)

break;

else if(op==1)

{

int val;

printf("Enter value: ");

scanf("%d",&val);

enqueue(&q,val);

}

else if(op==2)

{

int val=dequeue(&q);

if(val!=-999)

printf("Popped out: %d\n",val);

}

else

printf("Wrong choice! try again.\n");

}

return 0;

}



**Q.6. Implement Queue using stack.**

#include<stdlib.h>

#include<stdio.h>

struct stack

{

struct queue\* q1;

struct queue\* q2;

int curr\_size;

};

struct queue

{

struct node \*front,\*rear;

};

struct node

{

int val;

struct node\* next;

};

void initQueue(struct queue\*\* q)

{

(\*q)->front=NULL;

(\*q)->rear=NULL;

return;

}

void initStack(struct stack\*\* s)

{

(\*s)->q1=(struct queue\*)malloc(sizeof(struct queue));

(\*s)->q2=(struct queue\*)malloc(sizeof(struct queue));

initQueue(&((\*s)->q1));

initQueue(&((\*s)->q2));

(\*s)->curr\_size=0;

return;

}

void enqueue(struct queue\*\* q,int val)

{

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

temp->val=val;

temp->next=NULL;

if((\*q)->front==NULL && (\*q)->rear==NULL)

{

(\*q)->front=temp;

(\*q)->rear=temp;

return;

}

(\*q)->rear->next=temp;

(\*q)->rear=temp;

return;

}

int dequeue(struct queue\*\* q)

{

if((\*q)->front==NULL)

{

return -999;

}

int v=(\*q)->front->val;

(\*q)->front=(\*q)->front->next;

if((\*q)->front==NULL)

(\*q)->rear=NULL;

return v;

}

void push(struct stack\*\* s,int val)

{

struct stack\* st=\*s;

st->curr\_size++;

enqueue(&(st->q2),val);

int v=dequeue(&(st->q1));

while(v!=-999)

{

enqueue(&(st->q2),v);

v=dequeue(&(st->q1));

}

struct queue\* temp=st->q1;

st->q1=st->q2;

st->q2=temp;

return;

}

int pop(struct stack\*\* s)

{

int v=dequeue(&((\*s)->q1));

if(v==-999)

printf("Stack underflow!\n");

return v;

}

int main()

{

struct stack\* st=(struct stack\*)malloc(sizeof(struct stack));

initStack(&st);

while(1)

{

printf("1.Push\n2.Pop\n3.exit\nchoose: ");

int op;

scanf("%d",&op);

if(op==3)

break;

else if(op==1)

{

int val;

printf("Enter value: ");

scanf("%d",&val);

push(&st,val);

}

else if(op==2)

{

int val=pop(&st);

if(val!=-999)

printf("Popped out: %d\n",val);

}

else

printf("Wrong choice! try again.\n");

}

return 0;

}

