1. **Singly Linked List Creation.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

int main()

{

struct node \*head, \*second, \*third, \*fourth, \*fifth;

head = malloc(sizeof(struct node));

printf("Enter data for first node: ");

scanf("%d", &head->data);

head->next = NULL;

second = malloc(sizeof(struct node));

printf("Enter data for second node: ");

scanf("%d", &second->data);

second->next = NULL;

head->next = second;

third = malloc(sizeof(struct node));

printf("Enter data for third node: ");

scanf("%d", &third->data);

third->next = NULL;

second->next = third;

fourth = malloc(sizeof(struct node));

printf("Enter data for fourth node: ");

scanf("%d", &fourth->data);

fourth->next = NULL;

third->next = fourth;

fifth = malloc(sizeof(struct node));

printf("Enter data for fifth node: ");

scanf("%d", &fifth->data);

fifth->next = NULL;

fourth->next = fifth;

struct node \*temp = head;

while (temp != NULL)

{

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

temp = temp->next;

}

printf("NULL\n");

return 0;

}

1. **Singly Linked List Insertion at Beginning.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

int main()

{

struct node \*head, \*second, \*third, \*fourth, \*fifth;

head = malloc(sizeof(struct node));

printf("Enter data for first node: ");

scanf("%d", &head->data);

head->next = NULL;

second = malloc(sizeof(struct node));

printf("Enter data for second node: ");

scanf("%d", &second->data);

second->next = NULL;

head->next = second;

third = malloc(sizeof(struct node));

printf("Enter data for third node: ");

scanf("%d", &third->data);

third->next = NULL;

second->next = third;

fourth = malloc(sizeof(struct node));

printf("Enter data for fourth node: ");

scanf("%d", &fourth->data);

fourth->next = NULL;

third->next = fourth;

fifth = malloc(sizeof(struct node));

printf("Enter data for fifth node: ");

scanf("%d", &fifth->data);

fifth->next = NULL;

fourth->next = fifth;

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

printf("Enter data for new node at beginning: ");

scanf("%d", &begin->data);

begin->next = head;

head = begin;

struct node \*temp = head;

while (temp != NULL)

{

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

temp = temp->next;

}

printf("NULL\n");

return 0;

}

1. **Singly Linked List Insertion at Middle.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

int main()

{

struct node \*head, \*second, \*third, \*fourth, \*fifth;

head = malloc(sizeof(struct node));

printf("Enter data for first node: ");

scanf("%d", &head->data);

head->next = NULL;

second = malloc(sizeof(struct node));

printf("Enter data for second node: ");

scanf("%d", &second->data);

second->next = NULL;

head->next = second;

third = malloc(sizeof(struct node));

printf("Enter data for third node: ");

scanf("%d", &third->data);

third->next = NULL;

second->next = third;

fourth = malloc(sizeof(struct node));

printf("Enter data for fourth node: ");

scanf("%d", &fourth->data);

fourth->next = NULL;

third->next = fourth;

fifth = malloc(sizeof(struct node));

printf("Enter data for fifth node: ");

scanf("%d", &fifth->data);

fifth->next = NULL;

fourth->next = fifth;

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

printf("Enter data for new node in the middle: ");

scanf("%d", &middle->data);

int position, i = 1;

printf("Enter the position to insert: ");

scanf("%d", &position);

struct node \*temp = head;

while (i < position - 1)

{

temp = temp->next;

i++;

}

middle->next = temp->next;

temp->next = middle;

temp = head;

while (temp != NULL)

{

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

temp = temp->next;

}

printf("NULL\n");

return 0;

}

1. Singly Linked List Insertion at End.

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

int main()

{

struct node \*head, \*second, \*third, \*fourth, \*fifth;

head = malloc(sizeof(struct node));

printf("Enter data for first node: ");

scanf("%d", &head->data);

head->next = NULL;

second = malloc(sizeof(struct node));

printf("Enter data for second node: ");

scanf("%d", &second->data);

second->next = NULL;

head->next = second;

third = malloc(sizeof(struct node));

printf("Enter data for third node: ");

scanf("%d", &third->data);

third->next = NULL;

second->next = third;

fourth = malloc(sizeof(struct node));

printf("Enter data for fourth node: ");

scanf("%d", &fourth->data);

fourth->next = NULL;

third->next = fourth;

fifth = malloc(sizeof(struct node));

printf("Enter data for fifth node: ");

scanf("%d", &fifth->data);

fifth->next = NULL;

fourth->next = fifth;

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

printf("Enter data for new node at the end: ");

scanf("%d", &end->data);

fifth->next = end;

end->next = NULL;

struct node \*temp = head;

while (temp != NULL)

{

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

temp = temp->next;

}

printf("NULL\n");

return 0;

}

1. **Singly Linked List Deletion at Beginning.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

int main()

{

struct node \*head, \*second, \*third, \*fourth, \*fifth;

head = malloc(sizeof(struct node));

printf("Enter data for first node: ");

scanf("%d", &head->data);

head->next = NULL;

second = malloc(sizeof(struct node));

printf("Enter data for second node: ");

scanf("%d", &second->data);

second->next = NULL;

head->next = second;

third = malloc(sizeof(struct node));

printf("Enter data for third node: ");

scanf("%d", &third->data);

third->next = NULL;

second->next = third;

fourth = malloc(sizeof(struct node));

printf("Enter data for fourth node: ");

scanf("%d", &fourth->data);

fourth->next = NULL;

third->next = fourth;

fifth = malloc(sizeof(struct node));

printf("Enter data for fifth node: ");

scanf("%d", &fifth->data);

fifth->next = NULL;

fourth->next = fifth;

// Deleting the first node (beginning)

struct node \*temp = head;

head = head->next;

free(temp);

temp = head;

while (temp != NULL)

{

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

temp = temp->next;

}

printf("NULL\n");

return 0;

}

1. **Singly Linked List Deletion at Middle.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

int main()

{

struct node \*head, \*second, \*third, \*fourth, \*fifth;

head = malloc(sizeof(struct node));

printf("Enter data for first node: ");

scanf("%d", &head->data);

head->next = NULL;

second = malloc(sizeof(struct node));

printf("Enter data for second node: ");

scanf("%d", &second->data);

second->next = NULL;

head->next = second;

third = malloc(sizeof(struct node));

printf("Enter data for third node: ");

scanf("%d", &third->data);

third->next = NULL;

second->next = third;

fourth = malloc(sizeof(struct node));

printf("Enter data for fourth node: ");

scanf("%d", &fourth->data);

fourth->next = NULL;

third->next = fourth;

fifth = malloc(sizeof(struct node));

printf("Enter data for fifth node: ");

scanf("%d", &fifth->data);

fifth->next = NULL;

fourth->next = fifth;

// Deleting a node in the middle (third node)

struct node \*temp = head;

struct node \*prev;

int position = 3, i = 1;

while (i < position - 1)

{

temp = temp->next;

i++;

}

prev = temp->next; // node to be deleted

temp->next = temp->next->next;

free(prev);

temp = head;

while (temp != NULL)

{

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

temp = temp->next;

}

printf("NULL\n");

return 0;

}

1. **Singly Linked List Deletion at End.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

int main()

{

struct node \*head, \*second, \*third, \*fourth, \*fifth;

head = malloc(sizeof(struct node));

printf("Enter data for first node: ");

scanf("%d", &head->data);

head->next = NULL;

second = malloc(sizeof(struct node));

printf("Enter data for second node: ");

scanf("%d", &second->data);

second->next = NULL;

head->next = second;

third = malloc(sizeof(struct node));

printf("Enter data for third node: ");

scanf("%d", &third->data);

third->next = NULL;

second->next = third;

fourth = malloc(sizeof(struct node));

printf("Enter data for fourth node: ");

scanf("%d", &fourth->data);

fourth->next = NULL;

third->next = fourth;

fifth = malloc(sizeof(struct node));

printf("Enter data for fifth node: ");

scanf("%d", &fifth->data);

fifth->next = NULL;

fourth->next = fifth;

// Deleting the last node (fifth node)

struct node \*temp = head;

struct node \*prev;

while (temp->next != NULL)

{

prev = temp;

temp = temp->next;

}

prev->next = NULL;

free(temp);

temp = head;

while (temp != NULL)

{

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

temp = temp->next;

}

printf("NULL\n");

return 0;

}

1. **Doubly Linked List Creation.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

struct node \*prev;

};

int main()

{

struct node \*head, \*second, \*third, \*fourth, \*fifth;

// Creating 5 initial nodes

head = malloc(sizeof(struct node));

printf("Enter data for first node: ");

scanf("%d", &head->data);

head->next = NULL;

head->prev = NULL;

second = malloc(sizeof(struct node));

printf("Enter data for second node: ");

scanf("%d", &second->data);

second->next = NULL;

head->next = second;

second->prev = head;

third = malloc(sizeof(struct node));

printf("Enter data for third node: ");

scanf("%d", &third->data);

third->next = NULL;

second->next = third;

third->prev = second;

fourth = malloc(sizeof(struct node));

printf("Enter data for fourth node: ");

scanf("%d", &fourth->data);

fourth->next = NULL;

third->next = fourth;

fourth->prev = third;

fifth = malloc(sizeof(struct node));

printf("Enter data for fifth node: ");

scanf("%d", &fifth->data);

fifth->next = NULL;

fourth->next = fifth;

fifth->prev = fourth;

// Forward Traversal

printf("Forward Traversal: \n");

struct node \*temp1 = head;

while (temp1 != NULL)

{

printf("%d -> ", temp1->data);

temp1 = temp1->next;

}

printf("NULL\n");

// Backward Traversal

printf("Backward Traversal: \n");

struct node \*temp2 = fifth;

while (temp2 != NULL)

{

printf("%d -> ", temp2->data);

temp2 = temp2->prev;

}

printf("NULL\n");

return 0;

}

1. **Doubly Linked List Insertion at Beginning.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

struct node \*prev;

};

int main()

{

struct node \*head, \*second, \*third, \*fourth, \*fifth;

// Creating 5 initial nodes (same as before)

head = malloc(sizeof(struct node));

printf("Enter data for first node: ");

scanf("%d", &head->data);

head->next = NULL;

head->prev = NULL;

second = malloc(sizeof(struct node));

printf("Enter data for second node: ");

scanf("%d", &second->data);

second->next = NULL;

head->next = second;

second->prev = head;

third = malloc(sizeof(struct node));

printf("Enter data for third node: ");

scanf("%d", &third->data);

third->next = NULL;

second->next = third;

third->prev = second;

fourth = malloc(sizeof(struct node));

printf("Enter data for fourth node: ");

scanf("%d", &fourth->data);

fourth->next = NULL;

third->next = fourth;

fourth->prev = third;

fifth = malloc(sizeof(struct node));

printf("Enter data for fifth node: ");

scanf("%d", &fifth->data);

fifth->next = NULL;

fourth->next = fifth;

fifth->prev = fourth;

// Forward Traversal before insertion

printf("Forward Traversal (before insertion): \n");

struct node \*temp1 = head;

while (temp1 != NULL)

{

printf("%d -> ", temp1->data);

temp1 = temp1->next;

}

printf("NULL\n");

// Backward Traversal before insertion

printf("Backward Traversal (before insertion): \n");

struct node \*temp2 = fifth;

while (temp2 != NULL)

{

printf("%d -> ", temp2->data);

temp2 = temp2->prev;

}

printf("NULL\n");

// Inserting a new node at the beginning

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

printf("Enter data for the new beginning node: ");

scanf("%d", &begin->data);

begin->next = head;

head->prev = begin;

begin->prev = NULL;

head = begin; // Update head to new node

// Forward Traversal after insertion

printf("Forward Traversal (after insertion): \n");

temp1 = head;

while (temp1 != NULL)

{

printf("%d -> ", temp1->data);

temp1 = temp1->next;

}

printf("NULL\n");

// Backward Traversal after insertion

printf("Backward Traversal (after insertion): \n");

temp2 = fifth;

while (temp2 != NULL)

{

printf("%d -> ", temp2->data);

temp2 = temp2->prev;

}

printf("NULL\n");

return 0;

}

1. **Doubly Linked List Insertion at Middle.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

struct node \*prev;

};

int main()

{

struct node \*head, \*second, \*third, \*fourth, \*fifth;

// Creating 5 initial nodes

head = malloc(sizeof(struct node));

printf("Enter data for first node: ");

scanf("%d", &head->data);

head->next = NULL;

head->prev = NULL;

second = malloc(sizeof(struct node));

printf("Enter data for second node: ");

scanf("%d", &second->data);

second->next = NULL;

head->next = second;

second->prev = head;

third = malloc(sizeof(struct node));

printf("Enter data for third node: ");

scanf("%d", &third->data);

third->next = NULL;

second->next = third;

third->prev = second;

fourth = malloc(sizeof(struct node));

printf("Enter data for fourth node: ");

scanf("%d", &fourth->data);

fourth->next = NULL;

third->next = fourth;

fourth->prev = third;

fifth = malloc(sizeof(struct node));

printf("Enter data for fifth node: ");

scanf("%d", &fifth->data);

fifth->next = NULL;

fourth->next = fifth;

fifth->prev = fourth;

// Forward Traversal

printf("Forward Traversal: \n");

struct node \*temp1 = head;

while (temp1 != NULL)

{

printf("%d -> ", temp1->data);

temp1 = temp1->next;

}

printf("NULL\n");

// Backward Traversal

printf("Backward Traversal: \n");

struct node \*temp2 = fifth;

while (temp2 != NULL)

{

printf("%d -> ", temp2->data);

temp2 = temp2->prev;

}

printf("NULL\n");

// Insertion in the middle (between second and third)

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

printf("Enter data for middle node: ");

scanf("%d", &middle->data);

middle->next = third;

middle->prev = second;

second->next = middle;

third->prev = middle;

// Forward Traversal after insertion

printf("Forward Traversal: \n");

temp1 = head;

while (temp1 != NULL)

{

printf("%d -> ", temp1->data);

temp1 = temp1->next;

}

printf("NULL\n");

// Backward Traversal after insertion

printf("Backward Traversal: \n");

temp2 = fifth;

while (temp2 != NULL)

{

printf("%d -> ", temp2->data);

temp2 = temp2->prev;

}

printf("NULL\n");

return 0;

}

1. **Doubly Linked List Insertion at End.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

struct node \*prev;

};

int main()

{

struct node \*head, \*second, \*third, \*fourth, \*fifth;

// Creating 5 initial nodes

head = malloc(sizeof(struct node));

printf("Enter data for first node: ");

scanf("%d", &head->data);

head->next = NULL;

head->prev = NULL;

second = malloc(sizeof(struct node));

printf("Enter data for second node: ");

scanf("%d", &second->data);

second->next = NULL;

head->next = second;

second->prev = head;

third = malloc(sizeof(struct node));

printf("Enter data for third node: ");

scanf("%d", &third->data);

third->next = NULL;

second->next = third;

third->prev = second;

fourth = malloc(sizeof(struct node));

printf("Enter data for fourth node: ");

scanf("%d", &fourth->data);

fourth->next = NULL;

third->next = fourth;

fourth->prev = third;

fifth = malloc(sizeof(struct node));

printf("Enter data for fifth node: ");

scanf("%d", &fifth->data);

fifth->next = NULL;

fourth->next = fifth;

fifth->prev = fourth;

// Forward Traversal

printf("Forward Traversal: \n");

struct node \*temp1 = head;

while (temp1 != NULL)

{

printf("%d -> ", temp1->data);

temp1 = temp1->next;

}

printf("NULL\n");

// Backward Traversal

printf("Backward Traversal: \n");

struct node \*temp2 = fifth;

while (temp2 != NULL)

{

printf("%d -> ", temp2->data);

temp2 = temp2->prev;

}

printf("NULL\n");

// Insertion at the end

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

printf("Enter data for end node: ");

scanf("%d", &end->data);

fifth->next = end;

end->prev = fifth;

end->next = NULL;

fifth = end;

// Forward Traversal after insertion

printf("Forward Traversal: \n");

temp1 = head;

while (temp1 != NULL)

{

printf("%d -> ", temp1->data);

temp1 = temp1->next;

}

printf("NULL\n");

// Backward Traversal after insertion

printf("Backward Traversal: \n");

temp2 = fifth;

while (temp2 != NULL)

{

printf("%d -> ", temp2->data);

temp2 = temp2->prev;

}

printf("NULL\n");

return 0;

}

1. **Doubly Linked List Deletion at Beginning.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

struct node \*prev;

};

int main()

{

struct node \*head, \*second, \*third, \*fourth, \*fifth;

// Creating 5 initial nodes

head = malloc(sizeof(struct node));

printf("Enter data for first node: ");

scanf("%d", &head->data);

head->next = NULL;

head->prev = NULL;

second = malloc(sizeof(struct node));

printf("Enter data for second node: ");

scanf("%d", &second->data);

second->next = NULL;

head->next = second;

second->prev = head;

third = malloc(sizeof(struct node));

printf("Enter data for third node: ");

scanf("%d", &third->data);

third->next = NULL;

second->next = third;

third->prev = second;

fourth = malloc(sizeof(struct node));

printf("Enter data for fourth node: ");

scanf("%d", &fourth->data);

fourth->next = NULL;

third->next = fourth;

fourth->prev = third;

fifth = malloc(sizeof(struct node));

printf("Enter data for fifth node: ");

scanf("%d", &fifth->data);

fifth->next = NULL;

fourth->next = fifth;

fifth->prev = fourth;

// Forward Traversal before deletion

printf("Forward Traversal: \n");

struct node \*temp1 = head;

while (temp1 != NULL)

{

printf("%d -> ", temp1->data);

temp1 = temp1->next;

}

printf("NULL\n");

// Backward Traversal before deletion

printf("Backward Traversal: \n");

struct node \*temp2 = fifth;

while (temp2 != NULL)

{

printf("%d -> ", temp2->data);

temp2 = temp2->prev;

}

printf("NULL\n");

// Deleting the first node (head)

struct node \*to\_delete = head;

head = head->next;

head->prev = NULL;

free(to\_delete);

// Forward Traversal after deletion

printf("Forward Traversal: \n");

temp1 = head;

while (temp1 != NULL)

{

printf("%d -> ", temp1->data);

temp1 = temp1->next;

}

printf("NULL\n");

// Backward Traversal after deletion

printf("Backward Traversal: \n");

temp2 = fifth;

while (temp2 != NULL)

{

printf("%d -> ", temp2->data);

temp2 = temp2->prev;

}

printf("NULL\n");

return 0;

}

1. **Doubly Linked List Deletion at Middle.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

struct node \*prev;

};

int main()

{

struct node \*head, \*second, \*third, \*fourth, \*fifth;

// Creating 5 initial nodes

head = malloc(sizeof(struct node));

printf("Enter data for first node: ");

scanf("%d", &head->data);

head->next = NULL;

head->prev = NULL;

second = malloc(sizeof(struct node));

printf("Enter data for second node: ");

scanf("%d", &second->data);

second->next = NULL;

head->next = second;

second->prev = head;

third = malloc(sizeof(struct node));

printf("Enter data for third node: ");

scanf("%d", &third->data);

third->next = NULL;

second->next = third;

third->prev = second;

fourth = malloc(sizeof(struct node));

printf("Enter data for fourth node: ");

scanf("%d", &fourth->data);

fourth->next = NULL;

third->next = fourth;

fourth->prev = third;

fifth = malloc(sizeof(struct node));

printf("Enter data for fifth node: ");

scanf("%d", &fifth->data);

fifth->next = NULL;

fourth->next = fifth;

fifth->prev = fourth;

// Forward Traversal before deletion

printf("Forward Traversal: \n");

struct node \*temp1 = head;

while (temp1 != NULL)

{

printf("%d -> ", temp1->data);

temp1 = temp1->next;

}

printf("NULL\n");

// Backward Traversal before deletion

printf("Backward Traversal: \n");

struct node \*temp2 = fifth;

while (temp2 != NULL)

{

printf("%d -> ", temp2->data);

temp2 = temp2->prev;

}

printf("NULL\n");

// Deleting the middle node (third)

struct node \*to\_delete = third;

second->next = fourth;

fourth->prev = second;

free(to\_delete);

// Forward Traversal after deletion

printf("Forward Traversal: \n");

temp1 = head;

while (temp1 != NULL)

{

printf("%d -> ", temp1->data);

temp1 = temp1->next;

}

printf("NULL\n");

// Backward Traversal after deletion

printf("Backward Traversal: \n");

temp2 = fifth;

while (temp2 != NULL)

{

printf("%d -> ", temp2->data);

temp2 = temp2->prev;

}

printf("NULL\n");

return 0;

}

1. **Doubly Linked List Deletion at End.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

struct node \*prev;

};

int main()

{

struct node \*head, \*second, \*third, \*fourth, \*fifth;

// Creating 5 initial nodes

head = malloc(sizeof(struct node));

printf("Enter data for first node: ");

scanf("%d", &head->data);

head->next = NULL;

head->prev = NULL;

second = malloc(sizeof(struct node));

printf("Enter data for second node: ");

scanf("%d", &second->data);

second->next = NULL;

head->next = second;

second->prev = head;

third = malloc(sizeof(struct node));

printf("Enter data for third node: ");

scanf("%d", &third->data);

third->next = NULL;

second->next = third;

third->prev = second;

fourth = malloc(sizeof(struct node));

printf("Enter data for fourth node: ");

scanf("%d", &fourth->data);

fourth->next = NULL;

third->next = fourth;

fourth->prev = third;

fifth = malloc(sizeof(struct node));

printf("Enter data for fifth node: ");

scanf("%d", &fifth->data);

fifth->next = NULL;

fourth->next = fifth;

fifth->prev = fourth;

// Forward Traversal before deletion

printf("Forward Traversal: \n");

struct node \*temp1 = head;

while (temp1 != NULL)

{

printf("%d -> ", temp1->data);

temp1 = temp1->next;

}

printf("NULL\n");

// Backward Traversal before deletion

printf("Backward Traversal: \n");

struct node \*temp2 = fifth;

while (temp2 != NULL)

{

printf("%d -> ", temp2->data);

temp2 = temp2->prev;

}

printf("NULL\n");

// Deleting the last node (fifth)

struct node \*to\_delete = fifth;

fourth->next = NULL;

fifth = fourth;

free(to\_delete);

// Forward Traversal after deletion

printf("Forward Traversal: \n");

temp1 = head;

while (temp1 != NULL)

{

printf("%d -> ", temp1->data);

temp1 = temp1->next;

}

printf("NULL\n");

// Backward Traversal after deletion

printf("Backward Traversal: \n");

temp2 = fifth;

while (temp2 != NULL)

{

printf("%d -> ", temp2->data);

temp2 = temp2->prev;

}

printf("NULL\n");

return 0;

}

**Stack**

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node \*next;

};

typedef struct Node node;

node \*head = NULL;

void push(int data)

{

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

if (temp == NULL)

{

printf("Memory Allocation Failed.\n");

return;

}

temp->data = data;

temp->next = head;

head = temp;

}

void pop()

{

if (head == NULL)

{

printf("Stack Underflow.\n");

return;

}

node \*del = head;

head = head->next;

free(del);

}

int main()

{

push(1);

push(2);

push(3);

push(4);

push(5);

push(6);

pop();

pop();

push(7);

pop();

push(8);

while (head != NULL)

{

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

node \*temp = head;

head = head->next;

free(temp);

}

printf("\n");

return 0;

}

**Queue**

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node \*next;

};

typedef struct Node node;

node \*front = NULL;

node \*rear = NULL;

void enqueue (int data)

{

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

if (temp == NULL)

{

printf ("Memory Allocation Failed. \n");

return;

}

temp -> data = data;

temp -> next = NULL;

if (rear == NULL)

{

front = rear = temp;

}

else

{

rear -> next = temp;

rear = temp;

}

}

void dequeue ()

{

if (front == NULL)

{

printf ("Queue Underflow.\n");

return;

}

node \*del = front;

front = front -> next;

if (front == NULL)

{

rear = NULL;

}

free (del);

}

int main ()

{

enqueue (1);

enqueue (2);

enqueue (3);

enqueue (4);

enqueue (5);

dequeue ();

dequeue ();

dequeue ();

enqueue (6);

while (front != NULL)

{

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

node \*temp = front;

front = front->next;

free(temp);

}

printf("\n");

return 0;

}