1. **Activity: Check whether string is palindrome or not using a Queue implemented with a Linked List**

**Algorithm:**

1. Get string from user and enqueue its characters to the queue one-by-one.
2. Dequeue elements and store characters in 2 strings-
   1. First-to-last in one stirng
   2. Last-to-first in other string
3. Check is both strings are same.
   1. If yes, string is a palindrome

**Test Case Table:**

|  |  |  |
| --- | --- | --- |
| Input | Palindrome | Pass/Fail |
| dad | True | Pass |
| abcde | False | Pass |
| madam | True | Pass |
| aniruddh |  |  |

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct Queue{

    char c;

    struct Queue \*next;

} \*head=NULL, \*ptr=NULL;

//Insert Beginning

void enqueue(char x){

    struct Queue \*temp = NULL;

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

    if (temp == NULL)

    {

        printf("Memory full.");

        return;

    }

    temp->c = x;

    if(head == NULL){

        temp->next = NULL;

        head = temp;

        return;

    }

    temp->next = head;

    head = temp;

}

char findFirst(){

    for(ptr = head; ptr->next != NULL; ptr = ptr->next);

    return ptr->c;

}

//Delete End

void dequeue(){

    if (head == NULL)

    {

        return;

    }

    if (head->next == NULL)

    {

        head = NULL;

        return;

    }

    for(ptr = head; ptr->next->next != NULL; ptr = ptr->next);

    struct Queue \*temp = ptr->next;

    ptr->next = NULL;

    free(temp);

}

int main(){

    printf("Enter string size: ");

    int n;

    scanf("%d", &n);

    while((getchar())!='\n');

    printf("Enter input string: ");

    for (int i = 0; i < n; i++)

    {

        char t;\

        scanf("%c", &t);

        enqueue(t);

    }

    char str1[n], str2[n];

    for (int i = 0; i < n; i++)

    {

        str1[i] = findFirst();

        \*(str2 + n - i - 1) = findFirst();

        dequeue();

    }

    if(!(strcmp(str1, str2))){

        printf("Given word is a Pallindrome.");

    }

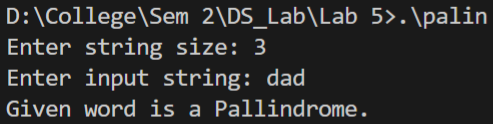
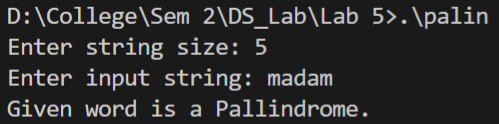
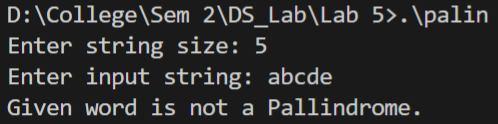
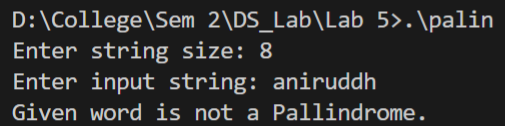
    else{

        printf("Given word is not a Pallindrome.");

    }

}

**Screenshot of compilation and execution:**

**  **

1. **Activity:** Using array to implement Priority Queue and its operations

**Algorithm:**

1. Create array of struct (Struct has two variables, data and priority) of desired size
2. Priority queue has 2 operations:-
   1. **Enqueue**
      1. Increment top element by 1
      2. Set the passed variables to the topth element of the array
   2. **Dequeue**
      1. Find the index with lowest number in priority by traversing through array
      2. Remove the element at that index
      3. Move next elements in the array to the left
      4. Decrement Top

**Test Case Table:**

|  |  |  |
| --- | --- | --- |
| Input | Output | Pass/Fail |
| (1/(2-3)\*4+5) | 123-/4\*5+ | Pass |
| (1+2\*3-4)/(5\*6) | 123\*+4-56\*/ | Pass |

**Program:**

#include <stdio.h>

#include <stdlib.h>

struct pr{

    int n;

    int priority;

} queue[1000000];

int top = -1;

void enqueue(int n, int p){

    top++;

    queue[top].n = n;

    queue[top].priority = p;

}

int findHighest(){

    int min = INT\_MAX;

    int index = -1;

    for (int i = 0; i <= top; i++)

    {

        if (min > queue[i].priority)

        {

            min = queue[i].priority;

            index = i;

        }

    }

    return index;

}

void dequeue(){

    int index = findHighest();

    for (int i = index; i < top; i++)

    {

        queue[i] = queue[i+1];

    }

    top--;

}

int main(){

    int arr[6] = {1,4,15,3,17,21};

    enqueue(1, 1);

    enqueue(4, 2);

    enqueue(15, 10);

    enqueue(3, 3);

    enqueue(17, 3);

    enqueue(21, 4);

    printf("Normal Queue: ");

    for (int i = 0; i < 6; i++)

    {

        printf("%d ", \*(arr+i));

    }

    printf("\n");

    printf("Priority Queue: ");

    int size = top+1;

    for (int i = 0; i < size; i++)

    {

        printf("%d ", queue[findHighest()].n);

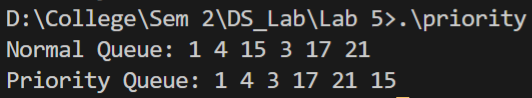
        dequeue();

    }

    return 0;

}

**Screenshot of compilation and execution:**

****

1. **Activity:** Using linked list to implement Double Ended Queue and its operations.

**Algorithm:**

1. **Input – Restricted:**
   1. For input:
      1. Ask user for data
      2. Create new node.
      3. Assign data to the new node
      4. Traverse to the end of list
      5. Last->next = new
   2. Ask user whether to dequeue from front or rear
   3. If rear:
      1. Traverse to 2nd last element of list
      2. Store its next element as temp
      3. Int ret = temp->data
      4. Free(temp)
      5. Return ret
   4. If front
      1. Temp = head
      2. Head = temp->next
      3. Int ret = temp->data
      4. Free(temp)
      5. Return ret
2. **Output – Restricted:**
   1. For input Ask user whether to enqueue from front or rear
   2. If rear:
      1. Ask user for data
      2. Create new node.
      3. Assign data to the new node
      4. Traverse to the end of list
      5. Last->next = new
   3. If front:
      1. Create new node
      2. Assign data to new node
      3. New->next = head
      4. Head = new
   4. For output:
      1. Temp = head
      2. Head = temp->next
      3. Int ret = temp->data
      4. Free(temp)
      5. Return ret

**Program:**

#include <stdio.h>

#include <stdlib.h>

int size, top = -1;

#include <stdio.h>

#include <stdlib.h>

struct queue{

    int n;

    struct queue \*next;

} \*in\_r=NULL, \*out\_r=NULL;

void clear\_buff(){

    while((getchar())!='\n');

}

//Insert Beginning

void ins\_beg(int n, struct queue \*\*p){

    struct queue \*temp = NULL;

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

    if (temp == NULL)

    {

        printf("Memory full...");

        return;

    }

    temp->n = n;

    temp->next = \*p;

    \*p = temp;

}

//Insert End

void ins\_end(int n, struct queue \*\*p){

    struct queue \*temp = NULL, \*ptr;

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

    if (temp == NULL)

    {

        printf("Memory full...");

        return;

    }

    temp->n = n;

    temp->next = NULL;

    if (\*p == NULL)

    {

        \*p = temp;

        return;

    }

    for (ptr = \*p; ptr->next != NULL ; ptr = ptr->next);

    ptr->next = temp;

}

//Display

void display(struct queue \*h){

    printf("Queue: ");

    struct queue \*ptr;

    for (ptr = h; ptr != NULL ; ptr = ptr->next)

    {

        printf("%d ", ptr->n);

    }

}

//Delete beginning

void delete\_beg(struct queue \*\*p){

    if (\*p == NULL)

    {

        printf("List Empty.");

        return;

    }

    struct queue \*temp;

    temp = \*p;

    if (temp->next == NULL)

    {

        free(temp);

        \*p = NULL;

        printf("Deleted list completely.");

        return;

    }

    \*p = temp->next;

    free(temp);

    printf("Deletion from front successful.");

}

//Delete End

void delete\_end(struct queue \*\*p){

    if (\*p == NULL)

    {

        printf("List Empty.");

        return;

    }

    struct queue \*temp, \*ptr;

    temp = \*p;

    if (temp->next == NULL)

    {

        free(temp);

        \*p = NULL;

        printf("Deleted list completely.");

        return;

    }

    for(;temp->next->next != NULL; temp = temp->next);

    ptr = temp->next;

    temp->next = NULL;

    free(ptr);

    printf("Deletion from end successful.");

}

int main(){

    int n;

    Main\_Menu: system("cls");

    printf("\tMain Menu\n");

    printf("-------------------------\n");

    printf("2 Queues available.\n\n");

    printf("1. Input restricted\n");

    printf("2. Output restriced\n");

    printf("3. Exit");

    printf("\nSelect Choice: ");

    scanf("%d", &n);

    switch (n)

    {

    case 1:

        In\_Menu: system("cls");

        printf("\tQueue Menu\n");

        printf("-------------------------\n");

        printf("The queue is input restricted\n");

        printf("1. Enqueue (rear)\n");

        printf("2. Display queue\n");

        printf("3. Dequeue (front)\n");

        printf("4. Dequeue (rear)\n");

        printf("5. Back\n");

        printf("\nSelect Choice: ");

        clear\_buff();

        scanf("%d", &n);

        int in;

        switch (n)

        {

        case 1:

            printf("Enter number u want to insert: ");

            scanf("%d", &in);

            ins\_end(in, &in\_r);

            goto Main\_Menu;

            break;

        case 2:

            display(in\_r);

            printf("\nPress Enter to Continue..");

            clear\_buff();

            getchar();

            goto Main\_Menu;

            break;

        case 3:

            delete\_beg(&in\_r);

            printf("\nPress Enter to Continue..");

            clear\_buff();

            getchar();

            goto Main\_Menu;

            break;

        case 4:

            delete\_end(&in\_r);

            printf("\nPress Enter to Continue..");

            clear\_buff();

            getchar();

            goto Main\_Menu;

            break;

        case 5:

            goto Main\_Menu;

        default:

            printf("Invalid Selection. Try again..");

            clear\_buff();

            getchar();

            goto In\_Menu;

            break;

        }

        break;

    case 2:

        In\_Menu: system("cls");

        printf("\tQueue Menu\n");

        printf("-------------------------\n");

        printf("The queue is Output restricted\n");

        printf("1. Enqueue (rear)\n");

        printf("2. Enqueue (front)\n");

        printf("3. Display Queue\n");

        printf("4. Dequeue (front)\n");

        printf("5. Back\n");

        printf("\nSelect Choice: ");

        clear\_buff();

        scanf("%d", &n);

        int in;

        switch (n)

        {

        case 1:

            printf("Enter number u want to insert: ");

            scanf("%d", &in);

            ins\_end(in, &out\_r);

            goto Main\_Menu;

            break;

        case 2:

            printf("Enter number u want to insert: ");

            scanf("%d", &in);

            ins\_beg(in, &out\_r);

            goto Main\_Menu;

            break;

        case 3:

            display(out\_r);

            printf("\nPress Enter to Continue..");

            clear\_buff();

            getchar();

            goto Main\_Menu;

            break;

        case 4:

            delete\_beg(&out\_r);

            printf("\nPress Enter to Continue..");

            clear\_buff();

            getchar();

            goto Main\_Menu;

            break;

        case 5:

            goto Main\_Menu;

        default:

            printf("Invalid Selection. Try again..");

            clear\_buff();

            getchar();

            goto In\_Menu;

            break;

        }

        break;

    case 3:

        free(in\_r);

        free(out\_r);

        break;

    default:

        printf("Invalid Selection. Try again..");

        clear\_buff();

        getchar();

        goto Main\_Menu;

        break;

    }

    return 0;

}

**Screenshot of compilation and execution:**

