16010122828

Queue (all queue type)- Basic operations and applications (Static and Dynamic implementation)

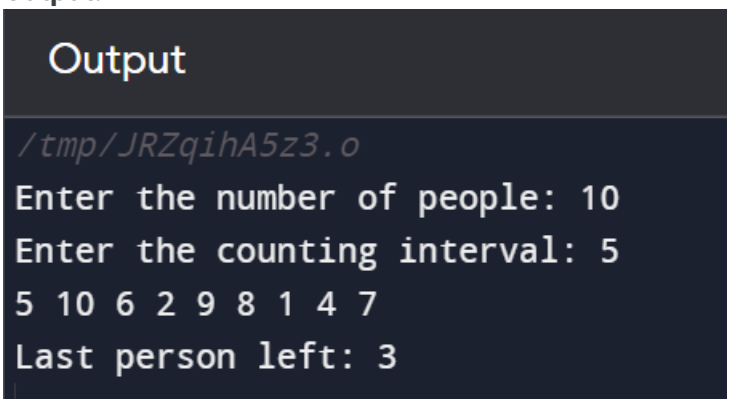
1. Joseph's problem using circular queue

Static:-

```c
#include <stdio.h>
#define MAX_SIZE 10
int queue[MAX_SIZE];
int front = -1, rear = -1;
int n, k;
void enqueue(int data) {
    if ((rear + 1) % MAX_SIZE == front) {
        printf("Error: Queue overflow\n");
        return;
    }
    if (front == -1) {
        front = 0;
    }
    rear = (rear + 1) % MAX_SIZE;
    queue[rear] = data;
}
int dequeue() {
    if (front == -1) {
        printf("Error: Queue underflow\n");
        return -1;
    }
    int data = queue[front];
    if (front == rear) {
        front = -1;
        rear = -1;
    } else {
        front = (front + 1) % MAX_SIZE;
    }
    return data;
}
void josephus(int n, int k) {
    for (int i = 1; i <= n; i++) {
        enqueue(i);
    }
    while (front != rear) {
```

```
        for (int i = 0; i < k - 1; i++) {
            enqueue(dequeue());
        }
        printf("%d ", dequeue());
    }
    printf("\nLast person left: %d\n", dequeue());
}
int main() {
    printf("Enter the number of people: ");
    scanf("%d", &n);
    printf("Enter the counting interval: ");
    scanf("%d", &k);
    josephus(n, k);
    return 0;
}
```

Output:-



Dynamic:-

```c
#include <stdio.h>
int josephus(int n, int k)
{
if (n == 1)
return 1;
else
return (josephus(n - 1, k) + k - 1) % n + 1;
}
int main()
{
int n,k;
printf("Enter the number of people in the circle: ");
scanf("%d",&n);
printf("Enter the position of person to be killed: ");
scanf("%d",&k);
printf("The chosen place is %d", josephus(n, k));
return 0;
}
```

Output

```
/tmp/GeygX1pY4G.o
Enter the number of people in the circle: 10
Enter the position of person to be killed: 3
The chosen place is 4
```

2. priority queue for senior citizens and general public(we can give some scenario)

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
void insert_by_priority(int);
void delete_by_priority(int);
void create();
void check(int);
void display_pqueue();
int pri_que[MAX];
int front, rear;
void main()
{
    int n, ch;
    printf("\n1 - Insert an element into queue");
    printf("\n2 - Delete an element from queue");
    printf("\n3 - Display queue elements");
    printf("\n4 - Exit");
    create();
    while (1)
    {
        printf("\nEnter your choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("\nEnter value to be inserted : ");
            scanf("%d",&n);
            insert_by_priority(n);
            break;
        case 2:
            printf("\nEnter value to delete : ");
            scanf("%d",&n);
            delete_by_priority(n);
            break;
        case 3:
            display_pqueue();
            break;
        case 4:
            exit(0);
        default:
            printf("\nChoice is incorrect, Enter a correct choice");
        }
    }
}
```

```
void create()
{
    front = rear = -1;
}
void insert_by_priority(int data)
{
    if (rear >= MAX - 1)
    {
        printf("\nQueue overflow no more elements can be inserted");
        return;
    }
    if ((front == -1) && (rear == -1))
    {
        front++;
        rear++;
        pri_que[rear] = data;
        return;
    }
    else
        check(data);
    rear++;
}
void check(int data)
{
    int i,j;

    for (i = 0; i <= rear; i++)
    {
        if (data >= pri_que[i])
        {
            for (j = rear + 1; j > i; j--)
            {
                pri_que[j] = pri_que[j - 1];
            }
            pri_que[i] = data;
            return;
        }
    }
    pri_que[i] = data;
}
void delete_by_priority(int data)
{
    int i;
    if ((front==-1) && (rear==-1))
    {
        printf("\nQueue is empty no elements to delete");
        return;
    }
```

```
    for (i = 0; i <= rear; i++)
    {
        if (data == pri_que[i])
        {
            for (; i < rear; i++)
            {
                pri_que[i] = pri_que[i + 1];
            }
        pri_que[i] = -99;
        rear--;
        if (rear == -1)
            front = -1;
        return;
        }
    }
    printf("\n%d not found in queue to delete", data);
}
void display_pqueue()
{
    if ((front == -1) && (rear == -1))
    {
        printf("\nQueue is empty");
        return;
    }
    for (; front <= rear; front++)
    {
        printf(" %d ", pri_que[front]);
    }
    front = 0;
}
```

Output

```
/tmp/nW9wxlaYNl.o
1 - Insert an element into queue
2 - Delete an element from queue
3 - Display queue elements
4 - Exit
Enter your choice : 1
Enter value to be inserted : 45
Enter your choice : 1
Enter value to be inserted : 89
Enter your choice : 1
Enter value to be inserted : 3
Enter your choice : 3
89  45  3
Enter your choice : 1
Enter value to be inserted : 98
Enter your choice : 3
98  89  45  3
Enter your choice : 1
Enter value to be inserted : 50
Enter your choice : 3
98  89  50  45  3
Enter your choice : 2
Enter value to delete : 3
Enter your choice : 3
98  89  50  45
Enter your choice : 2
Enter value to delete : 50
Enter your choice : 3
98  89  45
Enter your choice : 4
```

3. OS FIFO quere for processes considering some scenario

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

#define MAX_PROCESSES 10

struct process {
    int pid;
    char *name;
};

struct queue {
    int front;
    int rear;
    struct process *processes[MAX_PROCESSES];
};

void init(struct queue *q) {
    q->front = 0;
    q->rear = -1;
}

void enqueue(struct queue *q, int pid, char *name) {
    if (q->rear == MAX_PROCESSES - 1) {
        printf("Error: queue is full\n");
        return;
    }
    struct process *p = malloc(sizeof(struct process));
    p->pid = pid;
    p->name = malloc(strlen(name) + 1);
    strcpy(p->name, name);
    q->rear++;
    q->processes[q->rear] = p;
}

struct process *dequeue(struct queue *q) {
    if (q->front > q->rear) {
        printf("Error: queue is empty\n");
        return NULL;
    }
    struct process *p = q->processes[q->front];
    q->front++;
    return p;
}
```

```c
int main() {
    struct queue q;
    init(&q);

    int pid;
    char name[100];

    printf("Enter the process name: ");
    scanf("%s", name);
    printf("Enter the process pid: ");
    scanf("%d", &pid);
    enqueue(&q, pid, name);

    printf("Enter the process name: ");
    scanf("%s", name);
    printf("Enter the process pid: ");
    scanf("%d", &pid);
    enqueue(&q, pid, name);

    printf("Enter the process name: ");
    scanf("%s", name);
    printf("Enter the process pid: ");
    scanf("%d", &pid);
    enqueue(&q, pid, name);

    struct process *p;
    while ((p = dequeue(&q)) != NULL) {
        printf("Executing process %d (%s)\n", p->pid, p->name);
        free(p->name);
        free(p);
    }

    return 0;
}
```

Output

```
/tmp/HLNGjYHEJN.o
Enter the process name: task1
Enter the process pid: 1
Enter the process name: task2
Enter the process pid: 2
Enter the process name: task3
Enter the process pid: 3
Executing process 1 (task1)
Executing process 2 (task2)
Executing process 3 (task3)
Error: queue is empty
```

4. To-do list using FIFO queue or priority queue

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

#define MAX_ITEMS 10

struct item {
    int priority;
    char *description;
};

struct priority_queue {
    int size;
    struct item *items[MAX_ITEMS];
};

void init(struct priority_queue *q) {
    q->size = 0;
}

void push(struct priority_queue *q, int priority, char *description) {
    struct item *i = malloc(sizeof(struct item));
    i->priority = priority;
    i->description = malloc(strlen(description) + 1);
    strcpy(i->description, description);

    int j;
    for (j = q->size; j > 0; j--) {
        if (q->items[j - 1]->priority <= i->priority) break;
        q->items[j] = q->items[j - 1];
    }
    q->items[j] = i;
    q->size++;
}

struct item *pop(struct priority_queue *q) {
    if (q->size == 0) return NULL;
    return q->items[--q->size];
}

int main() {
    struct priority_queue q;
    init(&q);

    int priority;
```

```
    char description[100];

    printf("Enter the task description: ");
    scanf("%s", description);
    printf("Enter the task priority (1-10): ");
    scanf("%d", &priority);
    push(&q, priority, description);

    printf("Enter the task description: ");
    scanf("%s", description);
    printf("Enter the task priority (1-10): ");
    scanf("%d", &priority);
    push(&q, priority, description);

    printf("Enter the task description: ");
    scanf("%s", description);
    printf("Enter the task priority (1-10): ");
    scanf("%d", &priority);
    push(&q, priority, description);

    struct item *i;
    while ((i = pop(&q)) != NULL) {
        printf("%s (priority: %d)\n", i->description, i->priority);
        free(i->description);
        free(i);
    }

    return 0;
}
```

Output

```
/tmp/HLNGjYHEJN.o
Enter the task description: run-code
Enter the task priority (1-10): 2
Enter the task description: display-code
Enter the task priority (1-10): 3
Enter the task description: debugg-code
Enter the task priority (1-10): 1
display-code (priority: 3)
run-code (priority: 2)
debugg-code (priority: 1)
```