```c
/*Write a C program that declares a static variable and a const variable
within a function.
The program should increment the static variable each time the function is
called and use a switch case to check the value of the const variable.
 The function should handle at least three different cases for the const
variable and demonstrate the persistence of the static variable across
  multiple calls.*/
  #include<stdio.h>
  void func(int );
  void main()
  {
    int choice;
    do
    {
    printf("1.Option 1\n");
    printf("1.Option 2\n");
    printf("1.Option 3\n");
    printf("Enter choice : ");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1:func(1);
        break;
        case 2:func(2);
        break;
        case 3:func(3);
        break;
    }
    }while(choice!=4);

  }
  void func(int n)
  {
    static int count =0;
    const int c=n;
    count++;
    printf("Static value %d\n",count);
    printf("Constant value %d\n",c);
  }
```

```c
/*: Create a C program where a static variable is used to keep track of
the number of times a function has been called.
Implement a switch case to print a different message based on the number
of times the function has been invoked (e.g., first call, second call,
more
than two calls).Ensure that a const variable is used to define a maximum
call limit and terminate further calls once the limit is reached.*/
#include<stdio.h>
void func()
{
    static int n=0;
    const int max=5;
    n++;
    if(n==max)
    {
        printf("Exceeds maximum number of calls\n");
        return;
    }
    switch(n)
    {
        case 1:printf("First call\n");
        break;
        case 2:printf("Second call\n");
        break;
        case 3:printf("Third call\n");
        break;
        case 4:printf("Fourth call\n");
    }
}
void main()
{
    int choice;
    do
    {
        printf("1.Option1\n");
        printf("2.Option2\n");
        printf("3.Option3\n");
        printf("4.Option4\n");
        printf("5.Exit\n");
        printf("Enter option\n");
```

```c
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:func();
            break;
            case 2:func();
            break;
            case 3:func();
            break;
            case 4:func();
            break;
            case 5:
            break;
        }
    } while (choice!=5);

}
/*Develop a C program that utilizes a static array inside a function to
store values across multiple calls.
Use a const variable to define the size of the array. Implement a switch
case to perform different operations on the array elements
 (e.g., add, subtract, multiply) based on user input.
Ensure the array values persist between function calls.*/
#include<stdio.h>
void add(int ar[],int s)
{
    for(int i=0;i<s;i++)
    {
        ar[i]=ar[i]+3;
    }
    for(int i=0;i<s;i++)
    printf("%d|",ar[i]);
    printf("\n");
}
void subtract(int ar[],int s)
{
    for(int i=0;i<s;i++)
    {
        ar[i]=ar[i]-2;
    }
```

```
    for(int i=0;i<s;i++)
    printf("%d|",ar[i]);
    printf("\n");
}
void multiply(int ar[],int s)
{
    for(int i=0;i<s;i++)
    ar[i]=ar[i]*2;
    for(int i=0;i<s;i++)
    printf("%d|",ar[i]);
    printf("\n");
}
void main()
{
    const int size=5;
    static int ar[5]={1,2,3,4,5};
    int choice;
    do
    {
        printf("1.Add\n");
        printf("2.Subtract\n");
        printf("3.Multiply\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:add(ar,size);
            break;
            case 2:subtract(ar,size);
            break;
            case 3:multiply(ar,size);
            break;
        }
    } while (choice!=4);

}
#include<stdio.h>
void func(int n)
{
```

```
    switch(n)
    {
        case  0:printf("count is %d\n",n);
        break;
        case 1:printf("Sum of count and 5 is %d\n",n+5);
        break;
        case 2:printf("Difference between 5 and count is %d\n",5-n);
        break;
        case 3:printf("Product of 5 and count is %d\n",5*n);
        break;
        case 4:printf("quotient of 10 and count is %d\n",10/n);
    }
}
void main()
{
    static int count=0;
    const int max=5;
    do
    {
        switch(count)
        {
            case 0:func(count);
            count++;
            break;
            case 1:func(count);
             count++;
            break;
            case 2:func(count);
             count++;
            break;
            case 3:func(count);
             count++;
            break;
            case 4:func(count);
             count++;
            break;
        }
    } while (count<max);

}
```

```c
/*Create a C program with a static counter and a const limit.
The program should include a switch case to print different messages based
on the value of the counter.
After every 5 calls, reset the counter using the const limit.
The program should also demonstrate the immutability of the const variable
by attempting to modify it and showing the compilation error.*/
#include<stdio.h>
void func(int n , const int m)
{
    switch(n)
    {
        case  0:printf("count is %d\n",n);
        break;
        case 1:printf("Sum of count and 5 is %d\n",n+5);
        break;
        case 2:printf("Difference between 5 and count is %d\n",5-n);
        break;
        case 3:printf("Product of 5 and count is %d\n",5*n);
        break;
        case 4:m++;
        printf("%d",m);
        break;
    }
}
void main()
{
    static int count=0;
    const int max=5;
    do
    {
        if(count == max)
        count=0;
        switch(count)
        {
            case 0:func(count,max);
            count++;
            break;
            case 1:func(count,max);
             count++;
            break;
```

```
            case 2:func(count,max);
             count++;
            break;
            case 3:func(count,max);
             count++;
            break;
            case 4:func(count,max);
             count++;
            break;
        }
    } while (count<max);

}
/*Write a C program that demonstrates the use of both single and double
pointers.
Implement a function that uses a for loop to initialize an array and a
second function that modifies the array elements using a double pointer.
Use the const keyword to prevent modification of the array elements in one
of the functions.*/
#include<stdio.h>
void func(const int *p)
{
    for(int i=0;i<5;i++)
    printf("%d ",*(p+i));

}
void func1(int **p)
{
    printf("\n");
    for(int i=0;i<5;i++)
    {
        (*p)[i]=(*p)[i]+1;
    }

      for(int i=0;i<5;i++)
    printf("%d ",(*p)[i]);
}
void main()
{
    int ar[5]={0};
```

```c
   int *ptr=ar;
    func(ptr);

    func1(&ptr);
}
/*Develop a program that reads a matrix from the user and uses a function
to transpose the matrix.
The function should use a double pointer to manipulate the matrix.
Demonstrate both call by value and call by reference in the program.
Use a const pointer to ensure the original matrix is not modified during
the transpose operation.*/
#include<stdio.h>
#include<stdlib.h>
int **allocate(int row,int col)
{
    int **matrix=(int **)malloc(row*sizeof(int *));
    for(int i=0;i<row;i++)
    {
        matrix[i]=(int *)malloc(col*sizeof(int));
    }
return matrix;
}
int **transpose1(int **matrix,int row,int col)
{
    int **transpose=allocate(row,col);
    for(int i=0;i<row;i++)
    for(int j=0;j<col;j++)
    transpose[j][i]=matrix[i][j];
    return transpose;

}
void transpose2(int **matrix,int row,int col)
{
    int temp;
    for(int i=0;i<row;i++)
    for(int j=0;j<col;j++)
    {
        if(i<j)
        {
        temp=matrix[i][j];
```

```
        matrix[i][j]=matrix[j][i];
        matrix[j][i]=temp;

        }

    }
}

void insert(int **matrix,int row,int col)
{
    printf("Input matrix values\n");
    for(int i=0;i<row;i++)
    for(int j=0;j<col;j++)
    scanf("%d",&matrix[i][j]);
}
void display(int **matrix,int row,int col)
{
    for(int i=0;i<row;i++)
    {
        for(int j=0;j<col;j++)
        printf("%d ",matrix[i][j]);
        printf("\n");
    }
}

void main()
{
    int **matrix,**transpose;
    int row,col;
    printf("Enter size of matrix  :  ");
    scanf("%d %d",&row,&col);
    matrix=allocate(row,col);
    insert(matrix,row,col);
    printf("Matrix is \n");
    display(matrix,row,col);
    transpose=transpose1(matrix,col,row);
    printf("Transposed matrix\n");
    display(transpose,row,col);
     printf("Transposed matrix\n");
    transpose2(matrix,row,col);
```

```c
    display(matrix,row,col);

}
/*Create a C program that uses a single pointer to dynamically allocate
memory for an array.
Write a function to initialize the array using a while loop, and another
function to print the array.
Use a const pointer to ensure the printing function does not modify the
array.*/
#include<stdio.h>
#include<stdlib.h>
int *allocate(int n)
{
    int *array=(int *)malloc(n*sizeof(int));
    return array;
}
void initialize(int *array,int n)
{
    for (int i = 0; i < n; i++)
    {
        array[i]=i+1;
    }
}
void print(const int *array,int n)
{
    for(int i=0;i<n;i++)
    printf("%d ",array[i]);
}
void main()
{
    int n=5;
    int *array;
    array=allocate(n);
    initialize(array,n);
    print(array,n);
}
/*Write a program that demonstrates the use of double pointers to swap two
arrays.
 Implement functions using both call by value and call by reference.
```

```c
/* Use a for loop to print the swapped arrays and apply the const keyword
appropriately to ensure no modification occurs in certain operations.*/
#include<stdio.h>
#include<stdlib.h>
void swap1(int *a1,int *a2,int **newar1,int **newar2)
{
   *newar1=(int *)malloc(5*sizeof(int ));
   *newar2=(int *)malloc(5*sizeof(int ));
   for(int i=0;i<5;i++)
   {
       (*newar1)[i]=a1[i];
       (*newar2)[i]=a2[i];
   }
}
void swap2(int **a1,int **a2)
{
   int t;
   for(int i=0;i<5;i++)
   {
       t=(*a1)[i];
       (*a1)[i]=(*a2)[i];
       (*a2)[i]=t;
   }
}

void print(int *arr)
{
   for(int i=0;i<5;i++)
   printf("%d ",arr[i]);
   printf("\n");
}
void main()
{
   int *arr1;
   int *arr2;
   int *newar1,*newar2;
   arr1=(int *)malloc(5*sizeof(int));
   arr2=(int *)malloc(5*sizeof(int));
   for(int i=0;i<5;i++)
   scanf("%d",&*(arr1+i));
```

```c
     for(int i=0;i<5;i++)
    scanf("%d",&*(arr2+i));
    swap1(arr1,arr2,&newar1,&newar2);
    printf("swapped : \n");
    print(newar1);
    print(newar2);
    swap2(&arr1,&arr2);
    printf("swapped : \n");
    print(arr1);
    print(arr2);

 }

/*Develop a C program that demonstrates the application of const with
pointers.
Create a function to read a string from the user and another function to
count the frequency of each character using a do-while loop.
Use a const pointer to ensure the original string is not modified during
character frequency calculation.*/
#include<stdio.h>
#include<stdlib.h>
void findfreq(const char *str)
{
    int freq[26]={0};
    int i=0;
    do
    {
        if(str[i]>='a' &&str[i]<='z')
        {
            freq[str[i]-'a']++;

        }
        i++;
    }while(str[i]!='\0');
    printf("Frequency\n");
    for(int j=0;j<26;j++)
    {
        if(freq[j]!=0)
        printf("%c -> %d\n",j+'a',freq[j]);
    }
```

```c
}
void main()
{
    char *str;
    str=(char *)malloc(40*sizeof(char));
    printf("Enter string \n");
   fgets(str,40,stdin);
    findfreq(str);
}

/*Write a C program that uses an array of structures to store information
about employees.
Each structure should contain a nested structure for the address. Use
typedef to simplify the structure definitions.
The program should allow the user to enter and display employee
information.*/
#include<stdio.h>
#include<stdlib.h>
typedef struct
{
    char address[30];
}Details;
struct employee
{
    char name[20];
    int id;
    float salary;
    Details d;
};
static int count =0;
struct employee e[5];
void add()
{
    printf("Enter employee name : ");
    scanf("%s",e[count].name);
    printf("Enter employee id : ");
    scanf("%d",&e[count].id);
    printf("Enter salary : ");
    scanf("%f",&e[count].salary);
    printf("enter address : ");
```

```
    scanf("%s",e[count].d.address);
    count++;
}
void display()
{
    for(int i=0;i<count;i++)
    {
        printf("Employee name : %s\n",e[i].name);
        printf("Employee id : %d\n",e[i].id);
       printf("Salary : %.2f\n",e[i].salary);
       printf("Address : %s\n",e[i].d.address);
       printf("\n");

    }
}
void main()
{

    int choice;
    do
    {
    printf("1.Enter detatils\n");
    printf("2.Display details\n");
    printf("Enter choice\n");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1:add();
        break;
        case 2:display();
        break;
        case 3:
        break;
    }
    } while (choice !=3);

}
/*Create a program that demonstrates the use of a union to store different
types of data.
```

```c
/*Implement a nested union within a structure and use a typedef to define
the structure.
Use an array of this structure to store and display information about
different data types (e.g., integer, float, string).*/
#include <stdio.h>
#include <string.h>

typedef struct {
    char type;
    union {
        int intValue;
        float floatValue;
        char strValue[20];
    } data;
} DataStorage;
void display(DataStorage arr[], int size) {
    printf("\nStored Data:\n");
    for (int i = 0; i < size; i++) {
        printf("Entry %d: ", i + 1);
        switch (arr[i].type) {
            case 'i':
                printf("Integer = %d\n", arr[i].data.intValue);
                break;
            case 'f':
                printf("Float = %.2f\n", arr[i].data.floatValue);
                break;
            case 's':
                printf("String = %s\n", arr[i].data.strValue);
                break;
            default:
                printf("Unknown Type\n");
        }
    }
}

int main() {
    DataStorage arr[3];
    arr[0].type = 'i';
    arr[0].data.intValue = 42;
    arr[1].type = 'f';
```

```c
    arr[1].data.floatValue = 3.14;
    arr[2].type = 's';
    strcpy(arr[2].data.strValue, "Hello");
    display(arr, 3);
    return 0;
}

/*Student Admission Queue: Write a program to simulate a student admission
process.
Implement a queue using arrays to manage students waiting for admission.
Include operations to enqueue (add a student), dequeue (admit a student),
and display the current queue of students.*/
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
struct queue
{
    int size;
    int front;
    int rear;
    char **name;
};
void create(struct queue *q,int n)
{
    q->size=n;
   q->name=(char **)malloc(n*sizeof(char *));
   for(int i=0;i<n;i++)
   q->name[i]=(char *)malloc(20*sizeof(char));
   q->front=-1;
   q->rear=-1;
}
void enqueue(struct queue *q)
{
    if(q->rear==q->size-1)
    printf("Queue is full\n");
    else
    {
        q->rear++;
        printf("Enter name of student : ");
        scanf("%s",q->name[q->rear]);
```

```
    }

}
char dequeue(struct queue *q)
{
    char str[20];
    if(q->front==q->rear)
    printf("The queue is empty\n");
    else
    {
        q->front++;
        strcpy(str,q->name[q->front]);
    }
}
void display(struct queue *q)
{
    for(int i=q->front+1;i<=q->rear;i++)
    printf("%s ->",q->name[i]);
}
void main()
{
    int s,choice;
    struct queue q;
    printf("Enter size of queue : ");
    scanf("%d",&s);
    do
    {
        printf("1.create\n");
        printf("2.Enqueue\n");
        printf("3.Dequeue\n");
        printf("4.Display\n");
        printf("5.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            create(&q,s);
            break;
        case 2:
```

```
            enqueue(&q);
            break;
        case 3:
            dequeue(&q);
            break;
        case 4:
            display(&q);
            break;
case 5:
printf("Exiting....");
break;
        default: printf("Enter valid option\n");
            break;
        }
    } while (choice !=5);

}
/*Library Book Borrowing Queue: Develop a program that simulates a
library's book borrowing system. Use a queue to manage students waiting
to borrow books.Include functions to add a student to the queue, remove a
student after borrowing a book, and display the queue status*/
#include<stdio.h>
#include<stdlib.h>
struct queue
{
    int size;
    int front;
    int rear;
    char **name;
};
void create(struct queue *q,int n)
{
 q->size=n;
 q->front=q->rear=-1;
 q->name=(char **)malloc(n*sizeof(char *));
 for(int i=0;i<n;i++)
 q->name[i]=(char *)malloc(20*sizeof(char));
}
void enqueue(struct queue *q)
{
```

```
    if(q->rear==q->size-1)
    printf("Queue is full\n");
    else
    {
        q->rear++;
        printf("Enter name :");
        scanf("%s",q->name[q->rear]);
    }
}
void dequeue(struct queue *q)
{
    if(q->front==q->rear)
    printf("Queue is empty\n");
    else
    {
        q->front++;
        printf("Removed student %s\n",q->name[q->front]);

    }
}
void display(struct queue *q)
{
    for(int i=q->front+1;i<=q->rear;i++)
    printf("%s -> ",q->name[i]);
    printf("\n");
}
void main()
{
    struct queue q;
    int choice,n;
    printf("Enter size of queue : ");
    scanf("%d",&n);
    do
    {
         printf("1.create\n");
        printf("2.Enqueue\n");
        printf("3.Dequeue\n");
        printf("4.Display\n");
        printf("5.Exit\n");
        printf("Enter choice\n");
```

```
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            create(&q,n);
            break;
        case 2:
            enqueue(&q);
            break;
        case 3:
            dequeue(&q);
            break;
        case 4:
            display(&q);
            break;
case 5:
printf("Exiting....");
        }
    } while (choice !=5);

}
/*Create a program that simulates a cafeteria token system for students.
Implement a queue using arrays to manage students waiting for their turn.
Provide operations to issue tokens (enqueue), serve students (dequeue),
and display the queue of students.*/
#include<stdio.h>
#include<stdlib.h>
struct queue
{
    int size;
    int front;
    int rear;
    char **name;
};
void create(struct queue *q,int n)
{
 q->size=n;
 q->front=q->rear=-1;
 q->name=(char **)malloc(n*sizeof(char *));
 for(int i=0;i<n;i++)
```

```
 q->name[i]=(char *)malloc(20*sizeof(char));
}
void enqueue(struct queue *q)
{
    if(q->rear==q->size-1)
    printf("Queue is full\n");
    else
    {
        q->rear++;
        printf("Enter name :");
        scanf("%s",q->name[q->rear]);
    }
}
void dequeue(struct queue *q)
{
    if(q->front==q->rear)
    printf("Queue is empty\n");
    else
    {
        q->front++;
        printf("Removed student %s\n",q->name[q->front]);

    }
}
void display(struct queue *q)
{
    for(int i=q->front+1;i<=q->rear;i++)
    printf("%s -> ",q->name[i]);
    printf("\n");
}
void main()
{
    struct queue q;
    int choice,n;
    printf("Enter size of queue : ");
    scanf("%d",&n);
    do
    {
         printf("1.create\n");
        printf("2.issue tokens(Enqueue)\n");
```

```
        printf("3.serve students(Dequeue)\n");
        printf("4.Display\n");
        printf("5.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            create(&q,n);
            break;
        case 2:
            enqueue(&q);
            break;
        case 3:
            dequeue(&q);
            break;
        case 4:
            display(&q);
            break;
case 5:
printf("Exiting....");
        }
    } while (choice !=5);

}
/*Write a program to manage a help desk queue in a classroom. Use a queue
to track students waiting for assistance.
Include functions to add students to the queue, remove them once helped,
and view the current queue.*/
#include<stdio.h>
#include<stdlib.h>
struct queue
{
    int size;
    int front;
    int rear;
    char **name;
};
void create(struct queue *q,int n)
{
```

```
 q->size=n;
 q->front=q->rear=-1;
 q->name=(char **)malloc(n*sizeof(char *));
 for(int i=0;i<n;i++)
 q->name[i]=(char *)malloc(20*sizeof(char));
}
void enqueue(struct queue *q)
{
    if(q->rear==q->size-1)
    printf("Queue is full\n");
    else
    {
        q->rear++;
        printf("Enter name :");
        scanf("%s",q->name[q->rear]);
    }
}
void dequeue(struct queue *q)
{
    if(q->front==q->rear)
    printf("Queue is empty\n");
    else
    {
        q->front++;
        printf("Removed student %s\n",q->name[q->front]);

    }
}
void display(struct queue *q)
{
    for(int i=q->front+1;i<=q->rear;i++)
    printf("%s -> ",q->name[i]);
    printf("\n");
}
void main()
{
    struct queue q;
    int choice,n;
    printf("Enter size of queue : ");
    scanf("%d",&n);
```

```
    do
    {
         printf("1.create\n");
        printf("2.add students(Enqueue)\n");
        printf("3.remove students(Dequeue)\n");
        printf("4.Display\n");
        printf("5.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            create(&q,n);
            break;
        case 2:
            enqueue(&q);
            break;
        case 3:
            dequeue(&q);
            break;
        case 4:
            display(&q);
            break;
case 5:
printf("Exiting....");
        }
    } while (choice !=5);

}
/*Develop a program to simulate the exam registration process. Use a queue
to manage the order of student registrations.
Implement operations to add students to the queue, process their
registration, and display the queue status.*/
#include<stdio.h>
#include<stdlib.h>
struct queue
{
    int size;
    int front;
    int rear;
```

```
    char **name;
};
void create(struct queue *q,int n)
{
 q->size=n;
 q->front=q->rear=-1;
 q->name=(char **)malloc(n*sizeof(char *));
 for(int i=0;i<n;i++)
 q->name[i]=(char *)malloc(20*sizeof(char));
}
void enqueue(struct queue *q)
{
    if(q->rear==q->size-1)
    printf("Queue is full\n");
    else
    {
        q->rear++;
        printf("Enter name :");
        scanf("%s",q->name[q->rear]);
    }
}
void dequeue(struct queue *q)
{
    if(q->front==q->rear)
    printf("Queue is empty\n");
    else
    {
        q->front++;
        printf("Removed student %s\n",q->name[q->front]);

    }
}
void display(struct queue *q)
{
    for(int i=q->front+1;i<=q->rear;i++)
    printf("%s -> ",q->name[i]);
    printf("\n");
}
void main()
{
```

```
    struct queue q;
    int choice,n;
    printf("Enter size of queue : ");
    scanf("%d",&n);
    do
    {
         printf("1.create\n");
        printf("2.add students(Enqueue)\n");
        printf("3.remove students(Dequeue)\n");
        printf("4.Display\n");
        printf("5.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            create(&q,n);
            break;
        case 2:
            enqueue(&q);
            break;
        case 3:
            dequeue(&q);
            break;
        case 4:
            display(&q);
            break;
case 5:
printf("Exiting....");
        }
    } while (choice !=5);

}
/*Create a program that simulates the boarding process of a school bus.
Implement a queue to manage the order in which students board the bus.
Include functions to enqueue students as they arrive and dequeue them as
they board.*/
#include<stdio.h>
#include<stdlib.h>
struct queue
```

```
{
    int size;
    int front;
    int rear;
    char **name;
};
void create(struct queue *q,int n)
{
 q->size=n;
 q->front=q->rear=-1;
 q->name=(char **)malloc(n*sizeof(char *));
 for(int i=0;i<n;i++)
 q->name[i]=(char *)malloc(20*sizeof(char));
}
void enqueue(struct queue *q)
{
    if(q->rear==q->size-1)
    printf("Queue is full\n");
    else
    {
        q->rear++;
        printf("Enter name :");
        scanf("%s",q->name[q->rear]);
    }
}
void dequeue(struct queue *q)
{
    if(q->front==q->rear)
    printf("Queue is empty\n");
    else
    {
        q->front++;
        printf("Removed student %s\n",q->name[q->front]);

    }
}
void display(struct queue *q)
{
    for(int i=q->front+1;i<=q->rear;i++)
    printf("%s -> ",q->name[i]);
```

```
    printf("\n");
}
void main()
{
    struct queue q;
    int choice,n;
    printf("Enter size of queue : ");
    scanf("%d",&n);
    do
    {
         printf("1.create\n");
        printf("2.add students(Enqueue)\n");
        printf("3.remove students(Dequeue)\n");
        printf("4.Display\n");
        printf("5.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            create(&q,n);
            break;
        case 2:
            enqueue(&q);
            break;
        case 3:
            dequeue(&q);
            break;
        case 4:
            display(&q);
            break;
case 5:
printf("Exiting....");
        }
    } while (choice !=5);

}
/*Write a program to manage a queue for students waiting for a counseling
session.
```

```c
/*Use an array-based queue to keep track of the students, with operations to
add (enqueue) and serve (dequeue) students, and display the queue.*/
#include<stdio.h>
#include<stdlib.h>
struct queue
{
    int size;
    int front;
    int rear;
    char **name;
};
void create(struct queue *q,int n)
{
 q->size=n;
 q->front=q->rear=-1;
 q->name=(char **)malloc(n*sizeof(char *));
 for(int i=0;i<n;i++)
 q->name[i]=(char *)malloc(20*sizeof(char));
}
void enqueue(struct queue *q)
{
    if(q->rear==q->size-1)
    printf("Queue is full\n");
    else
    {
        q->rear++;
        printf("Enter name :");
        scanf("%s",q->name[q->rear]);
    }
}
void dequeue(struct queue *q)
{
    if(q->front==q->rear)
    printf("Queue is empty\n");
    else
    {
        q->front++;
        printf("Removed student %s\n",q->name[q->front]);

    }
```

```
}
void display(struct queue *q)
{
    for(int i=q->front+1;i<=q->rear;i++)
    printf("%s -> ",q->name[i]);
    printf("\n");
}
void main()
{
    struct queue q;
    int choice,n;
    printf("Enter size of queue : ");
    scanf("%d",&n);
    do
    {
         printf("1.create\n");
        printf("2.add students(Enqueue)\n");
        printf("3.remove students(Dequeue)\n");
        printf("4.Display\n");
        printf("5.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            create(&q,n);
            break;
        case 2:
            enqueue(&q);
            break;
        case 3:
            dequeue(&q);
            break;
        case 4:
            display(&q);
            break;
case 5:
printf("Exiting....");
        }
    } while (choice !=5);
```

```c
}
/*Develop a program that manages the registration queue for a school
sports event.
Use a queue to handle the order of student registrations, with functions
to add, process, and display the queue of registered students.*/
#include<stdio.h>
#include<stdlib.h>
struct queue
{
    int size;
    int front;
    int rear;
    char **name;
};
void create(struct queue *q,int n)
{
 q->size=n;
 q->front=q->rear=-1;
 q->name=(char **)malloc(n*sizeof(char *));
 for(int i=0;i<n;i++)
 q->name[i]=(char *)malloc(20*sizeof(char));
}
void enqueue(struct queue *q)
{
    if(q->rear==q->size-1)
    printf("Queue is full\n");
    else
    {
        q->rear++;
        printf("Enter name :");
        scanf("%s",q->name[q->rear]);
    }
}
void dequeue(struct queue *q)
{
    if(q->front==q->rear)
    printf("Queue is empty\n");
    else
    {
```

```
        q->front++;
        printf("Removed student %s\n",q->name[q->front]);

    }
}
void display(struct queue *q)
{
    for(int i=q->front+1;i<=q->rear;i++)
    printf("%s -> ",q->name[i]);
    printf("\n");
}
void main()
{
    struct queue q;
    int choice,n;
    printf("Enter size of queue : ");
    scanf("%d",&n);
    do
    {
         printf("1.create\n");
        printf("2.add students(Enqueue)\n");
        printf("3.remove students(Dequeue)\n");
        printf("4.Display\n");
        printf("5.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            create(&q,n);
            break;
        case 2:
            enqueue(&q);
            break;
        case 3:
            dequeue(&q);
            break;
        case 4:
            display(&q);
            break;
```

```c
case 5:
printf("Exiting....");
        }
    } while (choice !=5);

}
/*Create a program to simulate a queue for students waiting to check out
laboratory equipment.
Implement operations to add students to the queue, remove them once they
receive equipment, and view the current queue.*/
#include<stdio.h>
#include<stdlib.h>
struct queue
{
    int size;
    int front;
    int rear;
    char **name;
};
void create(struct queue *q,int n)
{
 q->size=n;
 q->front=q->rear=-1;
 q->name=(char **)malloc(n*sizeof(char *));
 for(int i=0;i<n;i++)
 q->name[i]=(char *)malloc(20*sizeof(char));
}
void enqueue(struct queue *q)
{
    if(q->rear==q->size-1)
    printf("Queue is full\n");
    else
    {
        q->rear++;
        printf("Enter name :");
        scanf("%s",q->name[q->rear]);
    }
}
void dequeue(struct queue *q)
{
```

```
    if(q->front==q->rear)
    printf("Queue is empty\n");
    else
    {
        q->front++;
        printf("Removed student %s\n",q->name[q->front]);

    }
}
void display(struct queue *q)
{
    for(int i=q->front+1;i<=q->rear;i++)
    printf("%s -> ",q->name[i]);
    printf("\n");
}
void main()
{
    struct queue q;
    int choice,n;
    printf("Enter size of queue : ");
    scanf("%d",&n);
    do
    {
         printf("1.create\n");
        printf("2.add students(Enqueue)\n");
        printf("3.remove students(Dequeue)\n");
        printf("4.Display\n");
        printf("5.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            create(&q,n);
            break;
        case 2:
            enqueue(&q);
            break;
        case 3:
            dequeue(&q);
```

```
            break;
        case 4:
            display(&q);
            break;
case 5:
printf("Exiting....");
        }
    } while (choice !=5);

}
/*Write a program to manage a queue for a parent-teacher meeting. Use a
queue to organize the order in which parents meet the teacher.
 Include functions to enqueue parents, dequeue them after the meeting, and
display the queue status.*/
 #include<stdio.h>
#include<stdlib.h>
struct queue
{
    int size;
    int front;
    int rear;
    char **name;
};
void create(struct queue *q,int n)
{
 q->size=n;
 q->front=q->rear=-1;
 q->name=(char **)malloc(n*sizeof(char *));
 for(int i=0;i<n;i++)
 q->name[i]=(char *)malloc(20*sizeof(char));
}
void enqueue(struct queue *q)
{
    if(q->rear==q->size-1)
    printf("Queue is full\n");
    else
    {
        q->rear++;
        printf("Enter name :");
        scanf("%s",q->name[q->rear]);
```

```
    }
}
void dequeue(struct queue *q)
{
    if(q->front==q->rear)
    printf("Queue is empty\n");
    else
    {
        q->front++;
        printf("Removed student %s\n",q->name[q->front]);

    }
}
void display(struct queue *q)
{
    for(int i=q->front+1;i<=q->rear;i++)
    printf("%s -> ",q->name[i]);
    printf("\n");
}
void main()
{
    struct queue q;
    int choice,n;
    printf("Enter size of queue : ");
    scanf("%d",&n);
    do
    {
         printf("1.create\n");
        printf("2.add parents(Enqueue)\n");
        printf("3.remove parents(Dequeue)\n");
        printf("4.Display\n");
        printf("5.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            create(&q,n);
            break;
        case 2:
```

```
            enqueue(&q);
            break;
        case 3:
            dequeue(&q);
            break;
        case 4:
            display(&q);
            break;
case 5:
printf("Exiting....");
break;
        }
    } while (choice !=5);

}
/*Implement a queue using a linked list to store real-time data from
various sensors (e.g., temperature, pressure).
The system should enqueue sensor readings, process and dequeue the oldest
data when a new reading arrives, and search for
specific readings based on timestamps.*/
#include<stdio.h>
#include<stdlib.h>
struct data
{
    float temp;
    float pressure;
    struct data*next;
}*front=NULL,*rear=NULL;
void enqueue()
{
    struct data *new=(struct data *)malloc(sizeof(struct data));
    if(new==NULL)
    printf("Queue is full");
    else
    {
        printf("Enter temperature : ");
        scanf("%f",&new->temp);
        printf("Enter pressure : ");
        scanf("%f",&new->pressure);
            new->next=NULL;
```

```
        if(front==NULL)
        {
            front=rear=new;

        }
        else
        {
            rear->next=new;
            rear=new;
        }
    }
}
void dequeue()
{
    struct data *t;
    if(front ==NULL)
    printf("Queue is empty");
    else
    {
        t=front;
        front=front->next;
        printf("Pressure : %.2f\n",t->pressure);
        printf("Temperature : %.2f\n",t->temp);
        free(t);
    }
}
void display()
{
    struct data *ptr=front;
    while(ptr!=NULL)
    {
        printf("Pressure : %.2f | Temperature : %.2f
\n",ptr->pressure,ptr->temp);
        ptr=ptr->next;
    }
}
void main()
{
    int choice;
    do
```

```
    {
        printf("1.add data(Enqueue)\n");
        printf("2.remove data(Dequeue)\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting....");
            break;
        }
    } while (choice !=4);

}
/*Design a queue using a linked list to manage task scheduling in an RTOS.
 Each task should have a unique identifier, priority level, and execution
time.
 Implement enqueue to add tasks, dequeue to remove the next task for
execution, and search to find tasks by priority.*/
 #include<stdio.h>
 #include<stdlib.h>
 struct task
 {
    int id;
    int priority;
    float time;
    struct task*next;
 }*front=NULL,*rear=NULL;
 void enqueue()
```

```
{
   struct task *new =(struct task *)malloc(sizeof(struct task));
   if(new==NULL)
   printf("Queue is full\n");
   else
   {
       new->next=NULL;
       printf("Enter task id : ");
       scanf("%d",&new->id);
       printf("Enter priority : ");
       scanf("%d",&new->priority);
       printf("Enter time : ");
       scanf("%f",&new->time);
       if(front==NULL)
       {
           front=rear=new;
       }
       else
       {
           rear->next=new;
           rear=rear->next;
       }
   }
}
void dequeue()
{
   struct task *t;
   if(front==NULL)
   printf("Queue is empty\n");
   else
   {
       t=front;
       front=front->next;
       free(t);
   }
}
void display()
{
   struct task *ptr=front;
   while(ptr!=NULL)
```

```
    {
        printf("Id : %d | Prioirity : %d | Time : %.2f
\n",ptr->id,ptr->priority,ptr->time);
        ptr=ptr->next;
    }
}
 void main()
{
    int choice;
    do
    {
        printf("1.add task(Enqueue)\n");
        printf("2.remove task(Dequeue)\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting....");
            break;
        }
    } while (choice !=4);

}
/*Create a queue using a linked list to manage interrupt requests (IRQs)
in an embedded system.
Each interrupt should have a priority level and a handler function.
Implement operations to enqueue new interrupts, dequeue the
highest-priority interrupt, and search for interrupts by their source.*/
```

```c
#include<stdio.h>
#include<stdlib.h>
struct IRQ
{
    int lvl;
    char fun[10];
    struct IRQ*next;
}*front=NULL,*rear=NULL;
void enqueue()
{
    struct IRQ *new=(struct IRQ *)malloc(sizeof(struct IRQ));
    if(new==NULL)
    printf("Queue is full\n");
    else
    {
        new->next=NULL;
        printf("Enter priority lvl : ");
        scanf("%d",&new->lvl);
        printf("Enter handler function : ");
        scanf("%s",new->fun);
        if(front==NULL)
        {
            front=rear=new;
        }
        else
        {
            rear->next=new;
            rear=rear->next;
        }
    }
}
void dequeue()
{
    struct IRQ *t;
    if(front==NULL)
    {
        printf("Queue is empty\n");
    }
    else
    {
```

```
        t=front;
        front=front->next;
        free(t);
    }

}
void display()
{
    struct IRQ *ptr=front;
    while(ptr!=NULL)
    {
        printf("priority lvl : %d | handler function :
%s\n",ptr->lvl,ptr->fun);
        ptr=ptr->next;
    }
    printf("\n");
}
void main()
{
    int choice;
    do
    {
        printf("1.add request(Enqueue)\n");
        printf("2.remove request(Dequeue)\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
```

```
            printf("Exiting....");
            break;
        }
    } while (choice !=4);

}
/*Implement a message queue using a linked list to handle inter-process
communication in embedded systems.
 Each message should include a sender ID, receiver ID, and payload.
 Enqueue messages as they arrive, dequeue messages for processing, and
search for messages from a specific sender.*/
 #include<stdio.h>
 #include<stdlib.h>
 struct msg
 {
    int sid;
    int rid;
    char payload[20];
    struct msg *next;
 }*front =NULL,*rear=NULL;
void enqueue()
{
    struct msg *new=(struct msg *)malloc(sizeof(struct msg));
    if(new==NULL)
    printf("Queeue is full\n");
    else
    {
        new->next=NULL;
        printf("Enter senter id : ");
        scanf("%d",&new->sid);
        printf("Enter receiver id : ");
        scanf("%d",&new->rid);
        printf("Enter payload : ");
        scanf("%s",new->payload);
        if(front==NULL)
        front=rear=new;
        else
        {
            rear->next=new;
            rear=rear->next;
```

```
        }
    }
}
void dequeue()
{
    struct msg *t;
    if(front ==NULL)
    printf("Emmpty queue\n");
    else
    {
        t=front;
        front=front->next;
        free(t);
    }
}
void display()
{
    struct msg *ptr=front;
    while(ptr!=NULL)
    {
        printf("Sender id : %d | Receiver id : %d | Payload :
%s\n",ptr->sid,ptr->rid,ptr->payload);
        ptr=ptr->next;
    }
}
 void main()
{
    int choice;
    do
    {
        printf("1.add message(Enqueue)\n");
        printf("2.remove message(Dequeue)\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            enqueue();
```

```
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting....");
            break;
        }
    } while (choice !=4);

}
/*Design a queue using a linked list to log data in an embedded system.
Each log entry should contain a timestamp, event type, and description.
 Implement enqueue to add new logs, dequeue old logs when memory is low,
and search for logs by event type.*/
  #include<stdio.h>
 #include<stdlib.h>
 #include<time.h>
 struct log
 {
    time_t now;
    char type[10];
    char des[20];
    struct log *next;
 }*front =NULL,*rear=NULL;
void enqueue()
{
    struct log *new=(struct log *)malloc(sizeof(struct log));
    if(new==NULL)
    printf("Queeue is full\n");
    else
    {
        new->next=NULL;
        new->now=time(NULL);
        printf("Enter event type : ");
        scanf("%s",new->type);
        getchar();
```

```
        printf("Enter description : ");
        scanf("%s",new->des);
        if(front==NULL)
        front=rear=new;
        else
        {
            rear->next=new;
            rear=rear->next;
        }
    }
}
void dequeue()
{
    struct log *t;
    if(front ==NULL)
    printf("Emmpty queue\n");
    else
    {
        t=front;
        front=front->next;
        free(t);
    }
}
void display()
{
    struct log *ptr=front;
    while(ptr!=NULL)
    {
        printf("Time stamp : %ld | Event type : %s | Description :
%s\n",ptr->now,ptr->type,ptr->des);
        ptr=ptr->next;
    }
}
 void main()
{
    int choice;
    do
    {
        printf("1.add log(Enqueue)\n");
        printf("2.remove log(Dequeue)\n");
```

```
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting....");
            break;
        }
    } while (choice !=4);

}
/*Create a queue using a linked list to manage network packets in an
embedded router.
Each packet should have a source IP, destination IP, and payload.
Implement enqueue for incoming packets, dequeue for packets ready for
transmission, and search for packets by IP address.*/
#include<stdio.h>
 #include<stdlib.h>
 #include<time.h>
 struct packet
 {
    char sip[10];
    char rip[10];
    char payload[20];
    struct packet *next;
 }*front =NULL,*rear=NULL;
void enqueue()
{
    struct packet *new=(struct packet *)malloc(sizeof(struct packet));
```

```
    if(new==NULL)
    printf("Queeue is full\n");
    else
    {
        new->next=NULL;
        printf("Enter sender ip : ");
        scanf("%s",new->sip);
        printf("Enter receiver ip : ");
        scanf("%s",new->rip);
        printf("Enter payload : ");
        scanf("%s",new->payload);
        if(front==NULL)
        front=rear=new;
        else
        {
            rear->next=new;
            rear=rear->next;
        }
    }
}
void dequeue()
{
    struct packet *t;
    if(front ==NULL)
    printf("Emmpty queue\n");
    else
    {
        t=front;
        front=front->next;
        free(t);
    }
}
void display()
{
    struct packet *ptr=front;
    while(ptr!=NULL)
    {
        printf("Sender ip : %s | Receiver ip : %s | Payload :
%s\n",ptr->sip,ptr->rip,ptr->payload);
        ptr=ptr->next;
```

```
    }
}
 void main()
{
    int choice;
    do
    {
        printf("1.add packet(Enqueue)\n");
        printf("2.remove packet(Dequeue)\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting....");
            break;
        }
    } while (choice !=4);

}
/*Implement a queue using a linked list to manage firmware updates in an
embedded system.
Each update should include a version number, release notes, and file path.
 Enqueue updates as they become available, dequeue them for installation,
and search for updates by version number.*/
 #include<stdio.h>
 #include<stdlib.h>
 #include<time.h>
 struct update
```

```
 {
    float ver;
    char notes[10];
    char filepath[20];
    struct update *next;
 }*front =NULL,*rear=NULL;
void enqueue()
{
    struct update *new=(struct update *)malloc(sizeof(struct update));
    if(new==NULL)
    printf("Queeue is full\n");
    else
    {
        new->next=NULL;
        printf("Enter version number : ");
        scanf("%f",&new->ver);
        printf("Enter release notes : ");
        scanf("%s",new->notes);
        printf("Enter filepath : ");
        scanf("%s",new->filepath);
        if(front==NULL)
        front=rear=new;
        else
        {
            rear->next=new;
            rear=rear->next;
        }
    }
}
void dequeue()
{
    struct update *t;
    if(front ==NULL)
    printf("Emmpty queue\n");
    else
    {
        t=front;
        front=front->next;
        free(t);
    }
```

```
}
void display()
{
    struct update *ptr=front;
    while(ptr!=NULL)
    {
        printf("Version : %.2f | Release note ip : %s | filepath :
%s\n",ptr->ver,ptr->notes,ptr->filepath);
        ptr=ptr->next;
    }
}
 void main()
{
    int choice;
    do
    {
        printf("1.add update(Enqueue)\n");
        printf("2.remove update(Dequeue)\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting....");
            break;
        }
    } while (choice !=4);

}
```

```c
/*Design a queue using a linked list to handle power management events in an embedded device.
 Each event should have a type (e.g., power on, sleep), timestamp, and associated action.
 Implement operations to enqueue events, dequeue events as they are handled, and search for events by type.*/
 #include<stdio.h>
 #include<stdlib.h>
 #include<time.h>
 struct event
 {
    time_t now;
    char type[10];
    char action[20];
    struct event *next;
 }*front =NULL,*rear=NULL;
void enqueue()
{
    struct event *new=(struct event *)malloc(sizeof(struct event));
    if(new==NULL)
    printf("Queeue is full\n");
    else
    {
        new->next=NULL;
        new->now=time(NULL);
        printf("Enter type : ");
        scanf("%s",new->type);
        printf("Enter action : ");
        scanf("%s",new->action);
        if(front==NULL)
        front=rear=new;
        else
        {
            rear->next=new;
            rear=rear->next;
        }
    }
}
void dequeue()
{
```

```
    struct event *t;
    if(front ==NULL)
    printf("Emmpty queue\n");
    else
    {
        t=front;
        front=front->next;
        free(t);
    }
}
void display()
{
    struct event *ptr=front;
    while(ptr!=NULL)
    {
        printf("Timestamp : %ld | Type : %s | action :
%s\n",ptr->now,ptr->type,ptr->action);
        ptr=ptr->next;
    }
}
 void main()
{
    int choice;
    do
    {
        printf("1.add event(Enqueue)\n");
        printf("2.remove event(Dequeue)\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
```

```
            display();
            break;
        case 4:
            printf("Exiting....");
            break;
        }
    } while (choice !=4);

}
 /* Create a command queue using a linked list to handle user or system
commands. Each command should have an ID, type, and parameters.
 Implement enqueue for new commands, dequeue for commands ready for
execution, and search for commands by type.*/
 #include<stdio.h>
 #include<stdlib.h>
 #include<time.h>
 struct command
 {
    int id;
    char type[10];
    char parameter[20];
    struct command *next;
 }*front =NULL,*rear=NULL;
void enqueue()
{
    struct command *new=(struct command *)malloc(sizeof(struct command));
    if(new==NULL)
    printf("Queeue is full\n");
    else
    {
        new->next=NULL;
        printf("Enter id : ");
        scanf("%d",new->id);
        printf("Enter type : ");
        scanf("%s",new->type);
        printf("Enter parameter : ");
        scanf("%s",new->parameter);
        if(front==NULL)
        front=rear=new;
        else
```

```
        {
            rear->next=new;
            rear=rear->next;
        }
    }
}
void dequeue()
{
    struct command *t;
    if(front ==NULL)
    printf("Emmpty queue\n");
    else
    {
        t=front;
        front=front->next;
        free(t);
    }
}
void display()
{
    struct command *ptr=front;
    while(ptr!=NULL)
    {
        printf("Command id : %d | Type : %s | parameter :
%s\n",ptr->id,ptr->type,ptr->parameter);
        ptr=ptr->next;
    }
}
 void main()
{
    int choice;
    do
    {
        printf("1.add command(Enqueue)\n");
        printf("2.remove command(Dequeue)\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
```

```
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting....");
            break;
        }
    } while (choice !=4);

}
 /*Implement a queue using a linked list to buffer audio samples in an
embedded audio system.
 Each buffer entry should include a timestamp and audio data.
  Enqueue new audio samples, dequeue samples for playback, and search for
samples by timestamp.*/
 #include<stdio.h>
 #include<stdlib.h>
 #include<time.h>
 struct sample
 {
    time_t now;
    char data[10];
    struct sample *next;
 }*front =NULL,*rear=NULL;
void enqueue()
{
    struct sample *new=(struct sample *)malloc(sizeof(struct sample));
    if(new==NULL)
    printf("Queeue is full\n");
    else
    {
        new->next=NULL;
        new->now=time(NULL);
```

```
        printf("Enter data : ");
        scanf("%s",new->data);
        if(front==NULL)
        front=rear=new;
        else
        {
            rear->next=new;
            rear=rear->next;
        }
    }
}
void dequeue()
{
    struct sample *t;
    if(front ==NULL)
    printf("Emmpty queue\n");
    else
    {
        t=front;
        front=front->next;
        free(t);
    }
}
void display()
{
    struct sample *ptr=front;
    while(ptr!=NULL)
    {
        printf("Time stamp: %ld | data : %s\n",ptr->now,ptr->data);
        ptr=ptr->next;
    }
}
 void main()
{
    int choice;
    do
    {
        printf("1.add sample(Enqueue)\n");
        printf("2.remove sample(Dequeue)\n");
        printf("3.Display\n");
```

```
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting....");
            break;
        }
    } while (choice !=4);

}
/*Design a queue using a linked list to manage events in an event-driven
embedded system. Each event should have an ID, type, and associated data.
Implement enqueue for new events, dequeue for event handling, and search
for events by type or ID.*/
 #include<stdio.h>
 #include<stdlib.h>
 #include<time.h>
 struct event
 {
    int id;
    char type[10];
    char data[10];
    struct event *next;
 }*front =NULL,*rear=NULL;
void enqueue()
{
    struct event *new=(struct event *)malloc(sizeof(struct event));
    if(new==NULL)
    printf("Queeue is full\n");
```

```
    else
    {
        new->next=NULL;
        printf("Enter id : ");
        scanf("%d",&new->id);
        printf("Enter type : ");
        scanf("%s",new->type);
        printf("Enter data : ");
        scanf("%s",new->data);
        if(front==NULL)
        front=rear=new;
        else
        {
            rear->next=new;
            rear=rear->next;
        }
    }
}
void dequeue()
{
    struct event *t;
    if(front ==NULL)
    printf("Emmpty queue\n");
    else
    {
        t=front;
        front=front->next;
        free(t);
    }
}
void display()
{
    struct event *ptr=front;
    while(ptr!=NULL)
    {
        printf("Event id: %ld |Type : %s | data :
%s\n",ptr->id,ptr->type,ptr->data);
        ptr=ptr->next;
    }
}
```

```c
 void main()
{
    int choice;
    do
    {
        printf("1.add event(Enqueue)\n");
        printf("2.remove event(Dequeue)\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting....");
            break;
        }
    } while (choice !=4);

}
/*Create a queue using a linked list to manage GUI events (e.g., button
clicks, screen touches) in an embedded system.
 Each event should have an event type, coordinates, and timestamp.
 Implement enqueue for new GUI events, dequeue for event handling, and
search for events by type.*/
 #include<stdio.h>
 #include<stdlib.h>
 #include<time.h>
 struct event
 {
    time_t now;
```

```
    char type[10];
    char cordinates[5];
    struct event *next;
 }*front =NULL,*rear=NULL;
void enqueue()
{
    struct event *new=(struct event *)malloc(sizeof(struct event));
    if(new==NULL)
    printf("Queeue is full\n");
    else
    {
        new->next=NULL;
        new->now=time(NULL);
        printf("Enter type : ");
        scanf("%s",new->type);
        printf("Enter coordinates : ");
        scanf("%s",new->cordinates);
        if(front==NULL)
        front=rear=new;
        else
        {
            rear->next=new;
            rear=rear->next;
        }
    }
}
void dequeue()
{
    struct event *t;
    if(front ==NULL)
    printf("Emmpty queue\n");
    else
    {
        t=front;
        front=front->next;
        free(t);
    }
}
void display()
{
```

```
    struct event *ptr=front;
    while(ptr!=NULL)
    {
        printf("Time stamp: %ld | Type : %s | Cordinates :
%s\n",ptr->now,ptr->type,ptr->cordinates);
        ptr=ptr->next;
    }
}
 void main()
{
    int choice;
    do
    {
        printf("1.add event(Enqueue)\n");
        printf("2.remove event(Dequeue)\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting....");
            break;
        }
    } while (choice !=4);

}
/*Implement a queue using a linked list to buffer data in a serial
communication system.
Each buffer entry should include data and its length.
```

```c
/*Enqueue new data chunks, dequeue them for transmission, and search for
specific data patterns.*/
#include<stdio.h>
 #include<stdlib.h>
 #include<time.h>
 struct buffer
 {
    int len;
    char data[30];
    struct buffer *next;
 }*front =NULL,*rear=NULL;
void enqueue()
{
    struct buffer *new=(struct buffer *)malloc(sizeof(struct buffer));
    if(new==NULL)
    printf("Queeue is full\n");
    else
    {
        new->next=NULL;
        printf("Enter len : ");
        scanf("%d",&new->len);
        printf("Enter data : ");
        scanf("%s",new->data);
        if(front==NULL)
        front=rear=new;
        else
        {
            rear->next=new;
            rear=rear->next;
        }
    }
}
void dequeue()
{
    struct buffer *t;
    if(front ==NULL)
    printf("Emmpty queue\n");
    else
    {
        t=front;
```

```
        front=front->next;
        free(t);
    }
}
void display()
{
    struct buffer *ptr=front;
    while(ptr!=NULL)
    {
        printf("buffer len: %ld | data : %s\n",ptr->len,ptr->data);
        ptr=ptr->next;
    }
}
 void main()
{
    int choice;
    do
    {
        printf("1.add buffer(Enqueue)\n");
        printf("2.remove buffer(Dequeue)\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting....");
            break;
        }
    } while (choice !=4);
```

```
}
/*Design a queue using a linked list to manage CAN bus messages in an
embedded automotive system.
Each message should have an ID, payload length, and payload.
Implement enqueue for incoming messages, dequeue for processing, and
search for messages by ID*/
 #include<stdio.h>
 #include<stdlib.h>
 #include<time.h>
 struct msg
 {
    int id;
    int len;
    char payload[10];
    struct msg *next;
 }*front =NULL,*rear=NULL;
void enqueue()
{
    struct msg *new=(struct msg *)malloc(sizeof(struct msg));
    if(new==NULL)
    printf("Queeue is full\n");
    else
    {
        new->next=NULL;
        printf("Enter id : ");
        scanf("%d",&new->id);
        printf("Enter length : ");
        scanf("%d",&new->len);
        printf("Enter payload : ");
        scanf("%s",new->payload);
        if(front==NULL)
        front=rear=new;
        else
        {
            rear->next=new;
            rear=rear->next;
        }
    }
}
```

```
void dequeue()
{
    struct msg *t;
    if(front ==NULL)
    printf("Emmpty queue\n");
    else
    {
        t=front;
        front=front->next;
        free(t);
    }
}
void display()
{
    struct msg *ptr=front;
    while(ptr!=NULL)
    {
        printf("msg id: %d |Length : %d | payload :
%s\n",ptr->id,ptr->len,ptr->payload);
        ptr=ptr->next;
    }
}
 void main()
{
    int choice;
    do
    {
        printf("1.add message(Enqueue)\n");
        printf("2.remove message(Dequeue)\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
```

```
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting....");
            break;
        }
    } while (choice !=4);

}
/*Create a queue using a linked list to manage input data for machine
learning inference in an embedded system.
Each entry should contain input features and metadata. Enqueue new data,
dequeue it for inference, and search for specific input data by
metadata.*/
 #include<stdio.h>
 #include<stdlib.h>
 #include<time.h>
 struct data
 {
    char meta[10];
    char feauture[10];
    struct data *next;
 }*front =NULL,*rear=NULL;
void enqueue()
{
    struct data *new=(struct data *)malloc(sizeof(struct data));
    if(new==NULL)
    printf("Queeue is full\n");
    else
    {
        new->next=NULL;
        printf("Enter metadata : ");
        scanf("%s",new->meta);
        printf("Enter feauture : ");
        scanf("%s",new->feauture);
        if(front==NULL)
        front=rear=new;
        else
```

```
        {
            rear->next=new;
            rear=rear->next;
        }
    }
}
void dequeue()
{
    struct data *t;
    if(front ==NULL)
    printf("Emmpty queue\n");
    else
    {
        t=front;
        front=front->next;
        free(t);
    }
}
void display()
{
    struct data *ptr=front;
    while(ptr!=NULL)
    {
        printf("Meta data: %s | feauture : %s\n",ptr->meta,ptr->feauture);
        ptr=ptr->next;
    }
}
 void main()
{
    int choice;
    do
    {
        printf("1.add data(Enqueue)\n");
        printf("2.remove data(Dequeue)\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&choice);
        switch (choice)
        {
```

```
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting....");
            break;
        }
    } while (choice !=4);

}
```