21. Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty). Implement the MyQueue class:

1. void push(int x) Pushes element x to the back of the queue.
2. int pop() Removes the element from the front of the queue and returns it.
3. int peek() Returns the element at the front of the queue.
4. boolean empty() Returns true if the queue is empty, false otherwise.

Input

["MyQueue", "push", "push", "peek", "pop", "empty"]

[[], [1], [2], [], [], []]

Output

[null, null, null, 1, 1, false] Explanation

MyQueue myQueue = new MyQueue(); myQueue.push(1); // queue is: [1]

myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue) myQueue.peek(); // return 1

myQueue.pop(); // return 1, queue is [2] myQueue.empty(); // return false

code:

```
#include <stdio.h>
#include <stdlib.h>
#include
<stdbool.h>
#define MAX_SIZE
100 typedef struct {
 int
```

```
 data[MAX_SIZE];
 int top;
} Stack;
typedef
struct
{ Stack
stack1;

 Stack stack2;
} MyQueue;
void stack_init(Stack
 *stack) { stack->top = -1;
}
bool stack_is_empty(Stack
*stack) { return stack->top ==
-1;
}
void stack_push(Stack *stack, int
 value) { if (stack->top < MAX_SIZE
 - 1) {
 stack->data[++stack->top] = value;
 } else {
 printf("Stack overflow\n");
 }
}
int stack_pop(Stack
 *stack) { if (!
 stack_is_empty(stack))
 {
 return stack->data[stack->top--];
 } else {
 printf("Stack
```

```
 underflow\n"); return -1;
 }
}
int stack_peek(Stack
 *stack) { if (!
 stack_is_empty(stack)) {
 return stack->data[stack->top];
 } else {
 printf("Stack is
 empty\n"); return -1;
 }
}

void myQueue_init(MyQueue
 *queue) { stack_init(&queue-
 >stack1); stack_init(&queue-
 >stack2);
}
void myQueue_push(MyQueue *queue,
 int x) { stack_push(&queue->stack1, x);
}
int myQueue_pop(MyQueue
 *queue) { if
 (stack_is_empty(&queue-
 >stack2)) {
 while (!stack_is_empty(&queue->stack1))
 { stack_push(&queue->stack2, stack_pop(&queue-
 >stack1));
 }
 }
 return stack_pop(&queue->stack2);
}
```

```
int myQueue_peek(MyQueue
 *queue) { if
 (stack_is_empty(&queue-
 >stack2)) {
 while (!stack_is_empty(&queue->stack1))
 { stack_push(&queue->stack2, stack_pop(&queue-
 >stack1));
 }
 }
 return stack_peek(&queue->stack2);
}
bool myQueue_empty(MyQueue *queue) {
 return stack_is_empty(&queue->stack1) && stack_is_empty(&queue->stack2);
}
int main()
 { MyQueue
 queue;
 myQueue_init(&queue);
 myQueue_push(&queue,
  1);
 myQueue_push(&queue, 2);
 myQueue_push(&queue, 3);
 printf("Front element: %d\n",
 myQueue_peek(&queue)); printf("Popped element:
 %d\n", myQueue_pop(&queue));
 printf("Is queue empty? %s\n", myQueue_empty(&queue) ? "Yes" :
 "No"); printf("Front element: %d\n", myQueue_peek(&queue));
 printf("Popped element: %d\n", myQueue_pop(&queue));
 printf("Popped element: %d\n", myQueue_pop(&queue));
 printf("Is queue empty? %s\n", myQueue_empty(&queue) ? "Yes" :
 "No"); return 0;
}
```

Input:

Output: Front element:

1 Popped element: 1

Is queue empty?

No Front

element: 2
Popped element: 2

Popped element:

3 Is queue

empty? Yes

22. **Given an array arr, sort the elements in descending order using bubblesort. Arr=[9,10,-9,23,67,-90]**

Code:

```
#include <stdio.h>

void bubbleSortDescending(int arr[], int n) {

for (int i = 0; i < n-1; i++) {

for (int j = 0; j < n-i-1; j++) {

if (arr[j] < arr[j+1]) {

int temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

}

int main() {

int arr[] = {9, 10, -9, 23, 67, -90};

int n = sizeof(arr)/sizeof(arr[0]);
```

```
bubbleSortDescending(arr, n);

printf("Output: [");

for (int i = 0; i < n; i++) {

printf("%d", arr[i]);

if (i < n - 1) {

printf(", ");

}

} printf("]\n");

return 0;

}
```

Input: [9,10,-9,23,67,-90]

Output: [67, 23, 10, 9, -9, -90]

23. you have been given a positive integer N. You need to find and print the Factorial of this number without using recursion. The Factorial of a positive integer N refers to the product of all number in the range from 1 to N.

Code:

```
#include <stdio.h>

int main() {

 int N;

 long long factorial = 1;

 printf("Enter a positive integer: ");

 scanf("%d", &N);l

 for(int i = 1; i <= N; i++) {

 factorial *= i;

 }
```

```
 printf("Factorial of %d = %lld\n", N, factorial);

 return 0;

}
```

Input:4

Output:24

24. Given an array arr, sort the elements in ascending order using Bubble sort. Arr=[9,10,-9,23,67,- 90] Output:[-90,-9,9,10,23,67]

Code:

```
#include <stdio.h>

void bubbleSort(int arr[], int n) {

 int i, j, temp;

 for (i = 0; i < n-1; i++) {

 for (j = 0; j < n-i-1; j++) {

 if (arr[j] > arr[j+1]) {

 temp = arr[j];

 arr[j] = arr[j+1];

 arr[j+1] = temp;

 }

 }

 }

}

int main() {

 int arr[] = {9, 10, -9, 23, 67, -90};

 int n = sizeof(arr)/sizeof(arr[0]);

 bubbleSort(arr, n);
```

```
printf("Output: [");
for (int i = 0; i < n; i++) {
printf("%d", arr[i]);
if (i < n - 1) {
printf(", ");
}
}
printf("]\n");
return 0;
}
```

Input: Arr=[9,10,-9,23,67,-90]

Output: [-90,-9,9,10,23,67]

25. Design a stack that supports push, pop, top, and retrieving the minimum element in constant time. Implement the MinStack class:

1. MinStack() initializes the stack object.
2. void push(int val) pushes the element val onto the stack.
3. void pop() removes the element on the top of the stack.
4. int top() gets the top element of the stack.
5. int getMin() retrieves the minimum element in the stack. Input ["MinStack","push","push","push","getMin","pop","top","g etMin"] [[],[-2],[0],[-3],[],[],[],[]]

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
typedef struct {
```

```
    int *stack;
    int *minStack;
    int topIndex;
    int minIndex;
    int capacity;
} MinStack;

MinStack* minStackCreate() {
    MinStack *minStack = (MinStack *)malloc(sizeof(MinStack));
    minStack->capacity = 1000;
    minStack->stack = (int *)malloc(minStack->capacity * sizeof(int));
    minStack->minStack = (int *)malloc(minStack->capacity * sizeof(int));
    minStack->topIndex = -1;
    minStack->minIndex = -1;
    return minStack;
}

void minStackPush(MinStack* obj, int val) {
    obj->stack[++(obj->topIndex)] = val;
    if (obj->minIndex == -1 || val <= obj->minStack[obj->minIndex]) {
        obj->minStack[++(obj->minIndex)] = val;
    }
}

void minStackPop(MinStack* obj) {
    if (obj->stack[obj->topIndex] == obj->minStack[obj->minIndex]) {
        obj->minIndex--;
    }
```

```
 obj->topIndex--;
}
int minStackTop(MinStack* obj) {
 return obj->stack[obj->topIndex];
}
int minStackGetMin(MinStack* obj) {
 return obj->minStack[obj->minIndex];
}
void minStackFree(MinStack* obj) {
 free(obj->stack);
 free(obj->minStack);
 free(obj);
}
int main() {
 MinStack* minStack = minStackCreate();
 minStackPush(minStack, 3);
 minStackPush(minStack, 5);
 printf("Current Min: %d\n", minStackGetMin(minStack));
 minStackPush(minStack, 2);
 minStackPush(minStack, 1);
 printf("Current Min: %d\n", minStackGetMin(minStack));
 minStackPop(minStack);
 printf("Current Min: %d\n", minStackGetMin(minStack));
 printf("Top Element: %d\n", minStackTop(minStack));
```

```
 minStackFree(minStack);

 return 0;

}
```

Input:

Output: Current Min: 3

Current Min: 1

Current Min: 2

Top Element: 2

26.find the factorial of a number using iterative procedure Input : 3

```
#include <stdio.h>

int main() {

 int number = 3;

 int factorial = 1;

for(int i = 1; i <= number; i++) {

 factorial *= i;

 }

 printf("Factorial of %d is %d\n", number, factorial);

 return 0;

}
```

Output: Factorial of 3 is 6

27.Given the head of a linked list, insert the node in nth place and return its head.
Input: head = [1,3,2,3,4,5], p=3 n = 2 Output: [1,3,2,3,4,5

Code:

```
#include

<stdio.h>
```

```
#include
<stdlib.h>
struct ListNode
{

 int val;
 struct ListNode *next;
};
struct ListNode* insertNode(struct ListNode* head, int p, int n) {
 struct ListNode* newNode = (struct
 ListNode*)malloc(sizeof(struct ListNode)); newNode->val = p;
 newNode->next =
NULL; if (n == 0) {
 newNode->next =
 head; return
 newNode;
 }
struct ListNode* current = head;
 for (int i = 0; i < n - 1 && current != NULL; i++) {
  current = current->next;
}
 if (current != NULL) {
 newNode->next = current-
 >next; current->next =
 newNode;
 }
return head;
}
 void printList(struct ListNode*
 head) { struct ListNode* current
 = head; while (current != NULL) {
 printf("%d ", current-
```

```
>val); current = current-
>next;
}
printf("\n");
}
int main() {
struct ListNode* head = (struct ListNode*)malloc(sizeof(struct ListNode));
head->val = 1;
head->next = (struct ListNode*)malloc(sizeof(struct
ListNode)); head->next->val = 3;
head->next->next = (struct ListNode*)malloc(sizeof(struct
ListNode)); head->next->next->val = 2;
head->next->next->next = (struct ListNode*)malloc(sizeof(struct
ListNode)); head->next->next->next->val = 3;
head->next->next->next->next = (struct ListNode*)malloc(sizeof(struct
ListNode)); head->next->next->next->next->val = 4;
head->next->next->next->next->next = (struct
ListNode*)malloc(sizeof(struct ListNode)); head->next->next->next->next-
>next->val = 5;
head->next->next->next->next->next->next =
NULL; head = insertNode(head, 3, 2);
printList(hea
d); return 0;
}
```

28. Given the head of a singly linked list and two integers left and right where left <= right, reverse the nodes of the list from position left to position right, and return the reversed list. Input: head = [1, 2, 3, 4, 5], left = 2, right = 4 Output: [1, 4, 3, 2, 5]

Code:

```
struct ListNode {
    int val;
    struct ListNode *next;

};
```

```
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
    if (!head || left == right) return head;

    struct ListNode dummy;
    dummy.next = head;
    struct ListNode* prev = &dummy;
    for (int i = 1; i < left; i++) {
 prev = prev->next;
    }

    struct ListNode* current = prev->next;
    struct ListNode* next = current->next;
    for (int i = 0; i < right - left; i++) {
 current->next = next->next;
 next->next = prev->next;
 prev->next = next;
 next = current->next;
    }

    return dummy.next;
}
```

29.you are given with the following linked list The digits are stored in the above order, you are asked to print the list in reverse order.

Code:

```
#include <stdio.h>
#include <stdlib.h>
struct ListNode {
 int val;
```

```
 struct ListNode *next;
};
struct ListNode* createNode(int val) {
 struct ListNode* newNode = (struct ListNode*)malloc(sizeof(struct ListNode));
 newNode->val = val;
 newNode->next = NULL;
 return newNode;
}
void printReverse(struct ListNode* head) {
 if (head == NULL) {
 return;
 }
 printReverse(head->next);
 printf("%d -> ", head->val);
}
void freeList(struct ListNode* head) {
 while (head != NULL) {
 struct ListNode* temp = head;
 head = head->next;
 free(temp);
 }
}
int main() {
 struct ListNode* head = createNode(1);
```

```
head->next = createNode(2);

head->next->next = createNode(3);

head->next->next->next = createNode(4);

head->next->next->next->next = createNode(5);

printf("Linked list in reverse order: ");

printReverse(head);

printf("NULL\n");

freeList(head);

return 0;

}
```

Output: Linked list in reverse order: 5 -> 4 -> 3 -> 2 -> 1 -> NULL

30.Given two sorted arrays nums1 and nums2 of size m and n respectively, return the sum of these two arrays

Code:

```
#include <stdio.h>

int sumSortedArrays(int* nums1, int m, int* nums2, int n) {

 int sum = 0;

 for (int i = 0; i < m; i++) {

 sum += nums1[i];

 }

 for (int j = 0; j < n; j++) {

 sum += nums2[j];

 }

 return sum;

}
```

```
int main() {
 int nums1[] = {1, 2, 3};
 int nums2[] = {4, 5, 6};
 int m = sizeof(nums1) / sizeof(nums1[0]);
 int n = sizeof(nums2) / sizeof(nums2[0]);
 int result = sumSortedArrays(nums1, m, nums2, n);
 printf("The sum of the two arrays is: %d\n", result);
 return 0;
}
```

Output: The sum of the two arrays is: 21

10. **Given a string s, find the frequency of characters Code:**

```
#include <stdio.h>
#include <string.h>
#define MAX_CHAR
256
void findFrequency(char
  *s) { int
  freq[MAX_CHAR] = {0};
  for (int i = 0; s[i] != '\0'; i+
  +) {
    freq[(unsigned char)s[i]]++;
  }
  for (int i = 0; i < MAX_CHAR; i++) {
    if (freq[i] > 0) {
      printf("%c -> %d\n", i, freq[i]);
    }
  }
```

```
}
int main() {
   char s[] =
   "tree";
   findFrequency(
   s); return 0;
}
```

Input: s = "tree"

Output: e ->

2 r -> 1

t -> 1

11. Given an unsorted array arr[] with both positive and negative elements,the task is to find the smallest positive number missing from the array.

Code:

```
#include <stdio.h>
int findMissingPositive(int arr[], int
  size) { int i;
 for (i = 0; i < size; i++) {
    while (arr[i] > 0 && arr[i] <= size && arr[arr[i] - 1] !=
      arr[i]) { int temp = arr[i];
      arr[i] = arr[temp -
      1]; arr[temp - 1] =
      temp;
    }
 }
 for (i = 0; i < size; i+
    +) { if (arr[i] != i +
    1) {
      return i + 1;
    }
 }
 return size + 1;
```

```
}
int main() {
   int arr[] = {2, 3, 7, 6, 8, -1, -10, 15};
   int size = sizeof(arr) / sizeof(arr[0]);
   int missing = findMissingPositive(arr, size);
   printf("The smallest positive number missing from the array is: %d\n",
   missing); return 0;
}
```

Input: 2,3,7,6,-1,-10,15

Output:1

12. Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return the binary tree. Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7] Output: [3,9,20,null,null,15,7]

Code:

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

struct
   TreeNode
   { int val;
   struct TreeNode
   *left; struct
   TreeNode *right;
};

struct TreeNode* buildTree(int* preorder, int preorderSize, int* inorder, int
   inorderSize) { if (preorderSize == 0 || inorderSize == 0) {
      return NULL;
   }

   struct TreeNode* root = (struct TreeNode*)malloc(sizeof(struct
   TreeNode)); root->val = preorder[0];
```

```
    int rootIndex;
    for (rootIndex = 0; rootIndex < inorderSize;
      rootIndex++) { if (inorder[rootIndex] == root->val)
      {
        break;
      }
    }

    root->left = buildTree(preorder + 1, rootIndex, inorder, rootIndex);
    root->right = buildTree(preorder + rootIndex + 1, preorderSize - rootIndex - 1, inorder +
rootIndex + 1, inorderSize - rootIndex - 1);

    return root;
}
```

```
void printTree(struct TreeNode*
    root) { if (root == NULL) {
      printf("null
      "); return;
    }
    printf("%d ", root-
    >val); printTree(root-
    >left); printTree(root-
    >right);
}

int main() {
    int preorder[] = {3, 9, 20, 15, 7};
    int inorder[] = {9, 3, 15, 20, 7};
    struct TreeNode* root = buildTree(preorder, 5, inorder, 5);
    printTree(root);
    return 0;
```

}

Output: 3 9 null null 20 15 null null 7 null null

13. **Write a program to create and display a linked list Example 1: Nodes : 6,7,8,9**
Output: 6->7->8-
>9

Code:

```
#include
<stdio.h>
#include
<stdlib.h>
struct Node {
 int data;
 struct Node* next;
};
void displayList(struct Node*
node) { while (node != NULL) {
printf("%d", node-
>data); if (node-
>next != NULL) { printf
("->");
}
node = node->next;
}
printf("\n");
}
int main() {
 struct Node* head = (struct Node*)malloc(sizeof(struct
 Node)); struct Node* second = (struct
 Node*)malloc(sizeof(struct Node)); struct Node* third =
 (struct Node*)malloc(sizeof(struct Node)); struct Node*
 fourth = (struct Node*)malloc(sizeof(struct Node));
 head->data = 6;
```

```
head->next =
second; second-
>data = 7; second-
>next = third; third-
>data = 8;

third->next =
fourth; fourth-
>data = 9; fourth-
>next = NULL;
displayList(head);
 free(head);
free(second);
free(third);
free(fourth);
 return 0;
}
```

Output: 6->7->8->9

14. **Write a program to sort the below numbers in descending order using bubble sort Input 4,7,9,1,2 Output:9,7,4,2,1**

Code:

```
#include <stdio.h>

void bubbleSort(int arr[], int
n) { int i, j, temp;
for (i = 0; i < n-1; i++) {
for (j = 0; j < n-i-1; j++) {
if (arr[j] <
arr[j+1]) { temp =
arr[j];
arr[j] = arr[j+1];
arr[j+1] = temp;
}
}
```

```
}
}
int main() {
 int arr[] = {4, 7, 9, 1, 2};
 int n = sizeof(arr)/sizeof(arr[0]);
 bubbleSort(arr, n);

 printf("Sorted array in descending
 order: "); for (int i = 0; i < n; i++) {
 printf("%d",
 arr[i]); if (i < n -
 1) { printf(",");
 }
 }
 return 0;
}
```

Output: Sorted array in descending order: 9,7,4,2,1

15. **Given an array of size N-1 such that it only contains distinct integers in the range of 1 to N. Find the missing element. Input: N = 5 A[] = {1,2,3,5} Output:4 Input N = 10 A[] = {6,1,2,8,3,4,7,10,5} Output: 9**

Code:

```
#include <stdio.h>

int findMissing(int A[], int
 N) { int total = (N * (N +
 1)) / 2; int sum = 0;
 for (int i = 0; i < N - 1; i+
 +) { sum += A[i];
 }
 return total - sum;
}
int main() {
 int A1[] = {1, 2, 3, 5};
 int N1 = 5;
```

```
printf("%d\n", findMissing(A1,
N1)); int A2[] = {6, 1, 2, 8, 3, 4,
7, 10, 5};
int N2 = 10;
printf("%d\n", findMissing(A2,
N2)); return 0;
}
```

Output:

4

9

16. **Write a program to find odd number present in the data part of a node Example Linked List 1- >2->3->7 Output: 1,3,7**

Code:

```
#include
<stdio.h>
#include
<stdlib.h>
struct Node {
 int data;
 struct Node* next;
};
void findOddNumbers(struct Node*
 head) { struct Node* current = head;
 while (current != NULL) {
 if (current->data % 2 !=
 0) { printf("%d ", current-
 >data);
 }
 current = current->next;
 }
}
int main() {
```

```
struct Node* head = (struct Node*)malloc(sizeof(struct
Node)); struct Node* second = (struct
Node*)malloc(sizeof(struct Node)); struct Node* third =
(struct Node*)malloc(sizeof(struct Node)); struct Node*
fourth = (struct Node*)malloc(sizeof(struct Node));
head->data = 1;
head->next =
second; second-
>data = 2; second-
>next = third; third-
>data = 3;

third->next =
fourth; fourth-
>data = 7; fourth-
>next = NULL;
printf("Odd numbers in the linked
list: "); findOddNumbers(head);
free(head);
free(secon
d);
free(third);
free(fourth)
; return 0;
}
```

Output: Odd numbers in the linked list: 1 3 7

17. **Write a program to perform insert and delete operations in a queue Example : 12,34,56,78 After insertion of 60 content of the queue is 12,34,56,78,60 After deletion of 12 , the contents of the queue : 34,56,78,60**

Code:

```
#include
<stdio.h>
#include
```

```
<stdlib.h>
#define MAX
100 struct
Queue {
 int
 items[MAX];
 int front;
 int rear;
};
struct Queue* createQueue() {
 struct Queue* q = (struct Queue*)malloc(sizeof(struct
 Queue)); q->front = -1;
 q->rear =
 -1; return q;
}
int isFull(struct Queue*
 q) { return q->rear ==
 MAX - 1;

}
int isEmpty(struct Queue* q) {
 return q->front == -1 || q->front > q->rear;
}
void enqueue(struct Queue* q, int
 value) { if (isFull(q)) {
 printf("Queue is
 full\n"); return;
 }
 if
 (isEmpty(q))
 { q->front = 0;
 }
```

```
 q->rear++;
 q->items[q->rear] = value;
}
int dequeue(struct Queue*
 q) { if (isEmpty(q)) {
 printf("Queue is
 empty\n"); return -1;
 }
 int item = q->items[q-
 >front]; q->front++;
 return item;
}
void display(struct
 Queue* q) { if
 (isEmpty(q)) {
 printf("Queue is
 empty\n"); return;
 }
 for (int i = q->front; i <= q->rear; i+
 +) { printf("%d ", q->items[i]);

 }
 printf("\n");
}
int main() {
 struct Queue* q =
 createQueue(); enqueue(q,
 12);
 enqueue(q, 34);
 enqueue(q, 56);
 enqueue(q, 78);
 printf("After insertion of 60, contents of the
```

```
queue: "); enqueue(q, 60);
display(q);

printf("After deletion of %d, contents of the queue: ",

dequeue(q)); display(q);

free(q);

return 0;
}
```

Output: After insertion of 60, contents of the queue: 12 34 56 78 60

After deletion of 12, contents of the queue: 34 56 78 60

18. **Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.**

Code:

```
#include

<stdio.h>

#include

<stdlib.h>

#include

<string.h>

#define MAX

100 typedef

struct {

  char

  items[MAX];

  int top;
} Stack;

void initStack(Stack* s) {

  s->top = -1;


}

int isFull(Stack* s) {

  return s->top == MAX - 1;
```

```
}
int
   isEmpty(Stack*
   s) { return s->top
   == -1;
}
void push(Stack* s, char
   item) { if (!isFull(s)) {
      s->items[++(s->top)] = item;

   }
}
char pop(Stack*
   s) { if (!
   isEmpty(s)) {
      return s->items[(s->top)--];
   }
   return '\0';

}
int isValid(char*
   s) { Stack
   stack;
   initStack(&sta
   ck);
   for (int i = 0; s[i] != '\0'; i++) {
      if (s[i] == '(' || s[i] == '{' || s[i] ==
         '[') { push(&stack, s[i]);
      } else {
         if (isEmpty(&stack)) return 0;

         char top = pop(&stack);
         if ((s[i] == ')' && top != '(') ||
```

```
            (s[i] == '}' && top != '{') ||
            (s[i] == ']' && top !=
            '[')) { return 0;
        }
      }
  }
  return isEmpty(&stack);
}
int main()
  { char
  s[MAX];
  printf("Enter a string of parentheses:
  "); scanf("%s", s);
  if (isValid(s)) {
     printf("The string is valid.\n");
  } else {
     printf("The string is not valid.\n");

  }
  return 0;

}
```

Output: Enter a string of

parentheses: ({}) The string is valid.

19. **Given a number n, the task is to print the Fibonacci series and the sum of the series using Iterative procedure.**

Code:

```
#include
<stdio.h> int
main() {
  int n = 10;
```

```
int first = 0, second = 1, next, sum = 0;

printf("Fibonacci series: ");

for (int i = 0; i < n; i+
  +) { if (i <= 1) {
    next = i;
  } else {
    next = first +
    second; first =
    second; second =
    next;
  }
  printf("%d",
  next); sum +=
    next;
    if (i < n - 1)
      { printf(", ");
    }
  }

  printf("\nSum: %d\n", sum);

  return 0;
}
```

Output: Fibonacci series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

Sum: 88

3. **Given the head of a singly linked list, return number of nodes present in a linked Example 1:**
1->2->3->5->8

Output

5 Code:

```
#include <stdio.h>

#include
<stdlib.h>
struct Node {
  int data;
  struct Node* next;
};
int countNodes(struct Node*
  head) { int count = 0;
  struct Node* current = head;

  while (current !=
    NULL) { count++;
    current = current->next;
  }
  return count;
}
struct Node* createNode(int data) {
  struct Node* newNode = (struct
  Node*)malloc(sizeof(struct Node)); newNode->data =
  data;
  newNode->next =
  NULL; return
  newNode;
}
int main() {
  struct Node* head =
  createNode(1); head->next =
  createNode(2);
  head->next->next = createNode(3);
  head->next->next->next = createNode(5);
```

```
    head->next->next->next->next =
    createNode(8); int nodeCount =
    countNodes(head);

    printf("Number of nodes: %d\n", nodeCount);

    return 0;
}
```

Output: Number of nodes: 5

4. **Given a number n. the task is to print the Fibonacci series and the sum of the series using recursion.**

input: n=10

output: Fibonacci series

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

Sum:

88

Code:

```
#include
<stdio.h> int
fibonacci(int n) {
   if (n <= 1)
      return n;
   return fibonacci(n - 1) + fibonacci(n - 2);
}
int
   sumFibonacci(int
   n) { if (n == 0)
      return 0;
   return fibonacci(n - 1) + sumFibonacci(n - 1);
}
void
   printFibonacciSeries(int
```

```
n) { for (int i = 0; i < n; i++) {
      printf("%d",
      fibonacci(i)); if (i < n -
      1) {
        printf(", ");
      }
   }
   printf("\n");
}

int main()
   { int n =
   10;
   printf("Fibonacci
   series: ");
   printFibonacciSeries(n
   );

   int sum =
   sumFibonacci(n);
   printf("Sum: %d\n",
   sum);

   return 0;
}
```

Output: Fibonacci series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

Sum: 88

5. **You are given an array arr in increasing order. Find the element x from arr using binary search.**
Example 1: arr={ 1,5,6,7,9,10},X=6

Output : Element found at location 2 Example 2:

arr={ 1,5,6,7,9,10},X=11

Output : Element not found at

location 2 Code:

```
#include <stdio.h>

int binarySearch(int arr[], int size,
  int x) { int left = 0;
  int right = size - 1;


  while (left <= right) {

    int mid = left + (right - left)
     / 2; if (arr[mid] == x)
      return mid;
    if (arr[mid] <
    x)
      left = mid +
     1; else
      right = mid - 1;

  }
  return -1;

}


int main() {
  int arr1[] = {1, 5, 6, 7, 9, 10};
  int size = sizeof(arr1) /
  sizeof(arr1[0]); int x1 = 6;
  int result = binarySearch(arr1,
  size, x1); if (result != -1)
    printf("Element %d found at location %d\n", x1,
  result); else
    printf("Element %d not found\n",
```

```
    x1); int x2 = 11;
  result = binarySearch(arr1,
  size, x2); if (result != -1)
    printf("Element %d found at location %d\n", x2,
  result); else
    printf("Element %d not found\n", x2);

  return 0;
}
```

Output: Element 6 found at location 2 Element 11 not found

7. **Given a string s, sort it in ascending order and find the starting index of repeated character Input: s = "tree"**

Output:"eert", starting index 0 Input: s = "kkj"

Output:"jkk", starting index : 1 Example 2:

Input: s = "cccaaa"

Output:"aaaccc", starting index 0,3 Example 3:

Input: s = "Aabb"

Output: "bbAa",starting index 0,2 Code:

```
#include <stdio.h>
#include <string.h>
void swap(char *x, char
  *y) { char temp = *x;
  *x = *y;
  *y = temp;
}
void sortString(char
```

```
s[]) { int n = strlen(s);
  for (int i = 0; i < n - 1; i+
    +) { for (int j = i + 1; j <
    n; j++) {
      if (s[i] > s[j]) {
        swap(&s[i], &s[j]);
      }
    }
  }
}
void
  findRepeatedIndices(char
  s[]) { int n = strlen(s);
  printf("Starting indices of repeated
  characters: "); for (int i = 0; i < n - 1; i++) {
    if (s[i] == s[i + 1]) {
      printf("%d", i);
      while (i < n - 1 && s[i] == s[i +
        1]) { i++;
      }
      if (i < n - 1)
        { printf(",
        ");
      }
    }
  }
  printf("\n");
}

int main() {
  char s1[] =
```

```
"tree";
sortString(s1);
printf("Sorted string: %s\n",
s1);
findRepeatedIndices(s1);

char s2[] =
"kkj";
sortString(s2);
printf("Sorted string: %s\n",
s2);
findRepeatedIndices(s2);

char s3[] =
"cccaaa";
sortString(s3);
printf("Sorted string: %s\n",
s3);
findRepeatedIndices(s3);

char s4[] =
"Aabb";
sortString(s4);
printf("Sorted string: %s\n",
s4);
findRepeatedIndices(s4);

return 0;
}
```

Output: Sorted string: eert

Starting indices of repeated characters:

0, Sorted string: jkk

Starting indices of repeated

characters: 1 Sorted string: aaaccc

Starting indices of repeated characters:

0, 3 Sorted string: Aabb

Starting indices of repeated characters: 2

8. Given the head of a singly linked list, return true if it is a palindrome or false otherwise. Example 1: Input: head = [1,2,2,1] Output: true Example 2: Input: head = [1,2] Output: false Input: R->A->D-

>A->R->NULL Output: Yes Input: C->O->D->E->NULL Output:

No Code:

```
#include

<stdio.h>

#include

<stdlib.h>

#include

<stdbool.h>
struct Node {
   char data;
   struct Node* next;
};
struct Node* createNode(char data) {
   struct Node* newNode = (struct
   Node*)malloc(sizeof(struct Node)); newNode->data =
   data;
   newNode->next =
   NULL; return
   newNode;
}
struct Node* reverseList(struct Node*
   head) { struct Node* prev = NULL;
   struct Node* current =
   head; struct Node* next
```

```
    = NULL;

    while (current !=
      NULL) { next =
      current->next;
      current->next =
      prev; prev =
      current; current =
      next;
    }
    return prev;
}
bool isPalindrome(struct Node* head) {
    if (head == NULL || head->next ==
      NULL) { return true;
    }
    struct Node* slow =
    head; struct Node*
    fast = head;
    while (fast != NULL && fast->next !=
      NULL) { slow = slow->next;
      fast = fast->next->next;
    }
    struct Node* secondHalf =
    reverseList(slow); struct Node*
    firstHalf = head;
    while (secondHalf != NULL) {
      if (firstHalf->data != secondHalf-
        >data) { return false;
      }
      firstHalf = firstHalf->next;
      secondHalf = secondHalf-
```

```
    >next;
  }
  return true;
}
int main() {
  struct Node* head1 = createNode(1);

  head1->next = createNode(2);
  head1->next->next =
  createNode(2);
  head1->next->next->next =
  createNode(1); if
  (isPalindrome(head1)) {
    printf("Output: true\n");
  } else {
    printf("Output: false\n");
  }
  struct Node* head2 =
  createNode(1); head2->next =
  createNode(2);

  if
    (isPalindrome(head2)
    ) { printf("Output:
    false\n");
} else {
    printf("Output: false\n");
}
  struct Node* head3 =
  createNode('R'); head3->next =
  createNode('A');
  head3->next->next = createNode('D');
  head3->next->next->next =
```

```
createNode('A');
head3->next->next->next->next =
createNode('R'); if (isPalindrome(head3)) {
   printf("Output: Yes\n");
} else {
   printf("Output: No\n");
}
struct Node* head4 =
createNode('C'); head4->next =
createNode('O');
head4->next->next = createNode('D');
head4->next->next->next =
createNode('E');

if
   (isPalindrome(head
   4)) { printf("Output:
   No\n");
  } else {
   printf("Output: No\n");
  }

  return 0;
}
```

Output:

Output:

true

Output:

false

Output:

Yes

Output: No