S. No.	Date	Title	~~Page No.~~ Marks	Teacher's Sign / Remarks
1)	14TH DECEMBER, 2023	LABORATORY-1	10	7/12
2)	21ST DECEMBER 2023	LABORATORY-2	10	21/12
3)	29TH DECEMBER 2023	LABORATORY-3	10	28/12
4)	11TH JANUARY, 2024	LABORATORY-4	10	16/1/24
5)	18TH JANUARY 2024	LABORATORY-5	10	18/1/24.
6)	25TH JANUARY, 2024	LABORATORY-6	10	25/1/24
7)	1ST FEBRUARY, 2024	LABORATORY-7	10	
8)	15TH FEBRUARY, 2024	LABORATORY-8	10	15/2/24
9)	22ND February, 2024	LABORATORY-9	10	29/2/24
10)	29TH FEBRUARY, 2024	LABORATORY-10	10	
			10	29/2/24

LABORATORY-1 14TH DECEMBER, 2023 THURSDAY

1) Develop a C program that simulates a banking system in basic with functionalities like account creation, withdrawal, deposit, and balance enquiry. Write different user defined functions for each.

Output:

Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. Check Balance
0. Exit
Enter your choice: 1
Enter account number: 1234
Account created successfully!

Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. Check Balance
0. Exit
Enter your choice: 2
Enter the amount to deposit: $33
Deposit successful! New Balance: $33.00

Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. Check Balance
0. Exit

Date
Page

Enter your choice: 3
Enter the amount to withdraw: $1
Withdrawal Successful! New balance: $32.00

2) Implement a C program that sorts strings lexicographically, considering uppercase and lowercase letters without using the standard library sorting functions

Output:
Enter the number of strings (up to 10): 4
Enter 4 strings:
String 1: Ash
String 2: dike
String 3: eel
String 4: ball

Sorted Strings:
Ash
ball
eel
like

3) Implement a C program to check if a given element is present in a 2D array with a user defined function

Output:
Enter the number of rows and columns (up to 10 each): 2 2
Enter the elements of the 2D array:
Element at position [0][0]: 1
Element at position [0][1]: 2
Element at position [1][0]: 3
Element at position [1][1] = 4
Enter the element to search: 3
Element 3 is present in the 2D array

4) Create a program in C to search for a substring within a large string using a user defined function.

Output:

Enter the main string: Halltrick

Enter the substring to search for: trick

Substring found at index 3 in the main string.

5) Write a C program to find the index of the last occurence of a number in an array with a user defined function.

Output:

Enter the size of the array: 10

Enter 10 elements for the array:

Element 1: 1

Element 2: 2

Element 3: 3

Element 4: 2

Element 5: 3

Element 6: 2

Element 7: 3

Element 8: 4

Element 9: 5

Element 10: 2

Enter the number to find its last accurence: 2

Last occurence of 2 is at index 9.

6) Write a C program to search for a specific element in an array using linear search with a user defined function.

Output:

Enter the number of elements in the array: 5

Enter the array elements: 3 13 4 5 2

Enter the value to be searched: 4.

4 found at location 3.

7) Implement a C program to perform a binary search on a ... array with a user defined function.

Output:

Enter number of elements in the array: 5

Enter the array elements: 6 7 8 9 11

Enter the value to be searched : 2

2 isn't present in the array.

8) Create a program in C to search for the maximum and minimum elements in an array with a user defined function.

Output:

Enter the size of the array: 6

Enter 6 elements for the array:

Element 1: 1

Element 2: 2

Element 3: 34

Element 4: 5

Element 5: 34

Element 6: -6

Minimum Element: -6

Maximum Element: 34

7/12

LABORATORY - 2 21ST DECEMBER, 2023 THURSDAY

1) Write a program in C to swap two numbers using pointers.

Code:

```
#include<stdio.h>
void swap1(int *a, int *b);
void swap2(int *a, int *b);
int main()
  {
    int a, b;
    printf("Enter the values of a and b: \t");
    scanf("%d %d", &a, &b);
    printf("\n The values of a\t and b\t before swapping: %d \t%d,",
        a, b);
    swap1(&a, &b);
    printf("\n The values of a\t and b\t after swapping using
        temporary variable: %d \t %d", a, b);
    swap2(&a, &b);
    printf("\n The values of a\t and b\t after swapping without
        using temporary variable: %d \t %d", a, b);
  }

void swap1(int *a, int *b)
  {
    int t;
    t = *a;
    *a = *b;
    *b = t;
  }

void swap2(int *a, int *b)
  {
    *a = *a + *b;
    *b = *a - *b;
```

```
*a = *a - *b;
}
```

Output:

Enter the values of a and b = 9 -7

The values of a and b before swapping : 9 -7

The values of a and b after swapping using temporary variable: -7 9

The values of a and b after swapping without using temporary variable : -7 9

2) Write a program in C to show dynamic memory allocation using malloc, calloc and realloc.

Code:

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

int main()
{
int *arr, i, n, m;
printf("Enter the values of the size: \t");
scanf("%d", &n);
arr = (int*) malloc (n * sizeof(n));
printf("Input the elements: \n");
for(i = 0; i<n; i++)
    scanf("%d", &arr[i]);
free(arr);
arr = (int *) calloc(n * sizeof(n));
printf("Input the elements: \n");
for(i=0; i<n; i++)
    scanf("%d", &arr[i]);
printf("Enter the new size: \t");
scanf("%d", &m);
```

```
arr = (int*) realloc(arr, m);
printf("Input the elements: \t");
for (i=0; i<m; i++)
  scanf("%d", &arr[i]);
free(arr);
}
```

Output:

Input the values of the size

3

Input the elements:

1

9

5

Input the elements:

7

6

5

Enter new size:

4

Input the elements:

3

2

1

0

Execution time: 22.272 s.

3) Write a program in C to simulate the working of stack using array with the following:
(a) PUSH (b) POP (c) DISPLAY.
The program should print appropriate messages for stack overflow and stack underflow.

```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 5
int top=-1;
int stack[SIZE];

void push(int element);
int pop();
void display();

int main()
{
int choice, element;
do
{
printf("\n Stack Operations:\n");
printf("1. PUSH\n");
printf("2. POP\n");
printf("3. DISPLAY\n");
printf("4. EXIT\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch(choice)
{
case 1: printf("Enter the element to push: ");
scanf("%d", &element);
push(element);
break;
```

```
Case 2:
    element = pop();
    if (element != -1)
        printf(" Popped Element : %d \n", element);
    break;
case 3:
    display();
    break;
Case 4:
    printf(" Exiting program. \n");
    break;
default:
    printf(" Invalid choice. Please enter a valid option. \n");
}

}
while (choice != 4);
return 0;
}

void push (int element)
{
if (top == SIZE - 1)
{
    printf(" Stack overflow. Cannot push element %d. \n", element);
}
else
{
    top ++;
    stack [top] = element;
    print(" Element %d pushed onto the stack. \n", element);
}
}
```

```
int pop()
  {
   if(top == -1)
   {
    printf("Stack underflow, Cannot pop any empty stack
    return-1;
   }
  else
   {
    int element = stack[top];
    top--;
    return element;
   }
}


void display()
  {
   if(top==-1)
    printf("Stack is empty.\n");
  else
   {
    printf("Stack elements:");
    for(int i=0; i<=top; i++)
     printf("%d", stack[i]);
    printf("\n");
   }
}
```

Output:

- Stack operations:

1. PUSH

2. POP

3. DISPLAY

4. EXIT

Enter your choice: 1

Enter element to push: 7

Element 7 pushed onto the stack.

- Stack Operations:

1. PUSH

2. POP

3. DISPLAY

4. EXIT

Enter your choice: 2

Popped element 7.

- Stack Operations:

1. PUSH

2. POP

3. DISPLAY

4. EXIT

Enter your choice: 3

Stack is empty

- Stack underflow. Cannot pop the element.
- Stack overflow: Cannot push the element 69.

24/12

LABORATORY-3 28TH DECEMBER, 2023 THURSDAY

1) Write a program in C to convert an infix expression to postfix.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100

int isOperator(char ch)
  {
    return (ch=='+' || ch=='-' || ch=='*' || ch=='/' || ch=='%')
  }

int precedence(char operator)
  {
    if (operator=='+' || operator=='-')
      return 1;
    if (operator=='*' || operator=='/' || operator=='%')
      return 2;
    return 0;
  }

void infixToPostfix(char infix[], char postfix[])
  {
    char stack[MAX_SIZE];
    int top = -1;
    int i, j;

    for (i=0; j=0; infix[i]!='\0'; i++)
      {
        if (infix[i]>='0' && infix[i]<='9')
          {
            postfix[j++] = infix[i];
          }
```

```
else if (isOperator(infix[i]))
  {
    while (top >= 0 && precedence(stack[top]) >=
            precedence(infix[i]))
      {
        postfix[j++] = stack[top--];
      }
    stack[++top] = infix[i];
  }
else if (infix[i] == '(')
  {
    stack[++top] = infix[i];
  }
else if (infix[i] == ')')
  {
    while (top >= 0 && stack[top] != '(')
      {
        postfix[j++] = stack[top--];
      }
    if (top >= 0 && stack[top] == '(')
      {
        top--;
      }
  }
}

while (top >= 0)
  {
    postfix[j++] = stack[top--];
  }
postfix[j] = '\0';
}

int main()
{
```

```
char infix[MAX_SIZE], postfix[MAX_SIZE];
printf("Enter infix expression: ");
scanf("%s", infix);

infixToPostfix(infix, postfix);
printf("Postfix expression: %s\n", postfix);
return 0;
}
```

Output:

Enter infix: 3 + ((4 * 5) / 6) % 9

Postfix: 345 * 6 / 9 % +

2) Write a C program to evaluate an expression in Postfix.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define MAX_STACK_SIZE 100

int stack[MAX_STACK_SIZE];
int top = -1;

void push (int item)
{
  if (top == MAX_STACK_SIZE - 1)
  {
    printf("Stack Overflow\n");
    exit(EXITFAILURE);
  }
  stack[++top] = item;
}
```

```
int pop()
{
  if (top == -1)
  {
    printf("Stack Underflow \n");
    exit(EXIT_FAILURE);
  }
  return stack[top--];
}

int isOperator(char ch)
{
  return (ch == '+' || ch == '-' || ch == '*' || ch == '/' ||
    ch == '%');
}

int evaluatePostfix(char postfix[])
{
  int i = 0;
  while (postfix[i] != '\0')
  {
    char currentSymbol = postfix[i];
    if (isdigit(currentSymbol))
    {
      push(currentSymbol - '0');
    }
    else if (isOperator(currentSymbol))
    {
      int operand2 = pop();
      int operand1 = pop();
      switch(currentSymbol)
      {
        case '+': push(operand1 + operand2);
                  break;
```

```
        case '-':
            push (operand1 - operand2);
            break;
        case '*':
            push (operand1 * operand2); break
        case '/':
            push (operand1 / operand2); break
        case '%':
            push (operand1 % operand2);
            break;
    }
}
i++;
}
return pop();
}

int main()
{
    char postfixExpression[100];
    printf("Enter postfix expression: ");
    scanf("%s", postfixExpression);
    int result = evaluatePostfix(postfixExpression);
    printf("Result: %d\n", result);
    return 0;
}
```

Output:

Enter: 345*6/9%+

Result: 6

28/12

3) a) Write a program in C to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display. The program should print appropriate messages for queue and queue overflow conditions

b) Write a program in C to simulate the working of a circular queue of integers using an array: Provide the following operations: Insert, Delete, Display. The program should print appropriate messages for queue empty and queue overflow conditions.

A)

```
#include <stdlib.h>
#include <stdio.h>
#define MAX. 3

int front=-1, rear=-1;
int queue[MAX];

void insert (int item)
{
 if (rear==MAX-1)
  {
   printf("overflow\n");
   exit (EXIT_FAILURE);
  }
else
  {
   if (rear==-1 && front==-1)
    {
     front==rear=0;
     queue[rear] =item;
    }
   else
    {
```

```
        rear += 1;
        queue[rear] = item;
    }
  }
}
int delete()
  {
    if (front == -1 || front > rear)
      {
        return -1;
      }
    else{
        return queue[front++];
      }
}
int main()
  {
    int n, ele, d;
    do
     {
       printf("1. Insert element\n 2. Delete element\n
              3. Exit\n");
       scanf("%d", &n);

       switch(n)
        {
         Case 1:
           printf("Enter the element : ");
           scanf("%d", &ele);
           insert(ele);
           break;

         case 2:
           d = delete();
```

```
if (d == -1)
  {
  printf("Underflow \n");
  exit(EXIT_FAILURE);
  }
  printf("The element deleted is: %d \n", d);
  break;

case 3:
    printf("Exiting the program \n");
    break;
default:
    printf("Please enter the right choice \n");
  }
} while (n != 3)
  return 0;
}
```

Output:

```
1. Insert element
2. Delete element
3. Exit
1
Enter the element: 34
1. Insert element
2. Delete element
3. Exit
2
The element deleted is: 34
1. Insert element
2. Delete element
3. Exit
2
Underflow
```

```
1. Insert element
2. Delete element
3. Exit
1
Enter the element: 44
1. Insert element
2. Delete element
3. Exit
1
Enter the element: 54
1. Insert Element
2. Delete element
3. Exit
1.
Overflow
```

6)

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#define MAX 6
int cq[MAX];
int front=-1, rear=-1;

bool is_full()
{ return (rear+1)% MAX == front;
 }

bool is_empty()
{

return front==-1 && rear==-1;
}
void insert(int item)
{
    if (is_full())
    {
      printf("Overflow: Circular queue is full.\n");
    }

    if (is_empty())
    {
      front = rear = 0;
    }
    else
    {
      rear = (rear+1) % MAX;
    }
  cq[rear] = item;
  printf("Enqueued : %d\n", item);
}
```

```
int dequeue()
{
if (isEmpty())
{
printf("Underflow: Circular queue is empty.");
return -1;
}

int deletedItem = cq[front];

if(front == rear)
    front = rear = -1;
else
    front = (front+1) % MAX;
printf("Dequeued: %d\n", deletedItem);
return deletedItem;

}
```

Output:

1. Insert
2. Delete
3. Exit

1

Enter the element to be inserted: 23

Enqueued: 23

1. Insert
2. Delete
3. Exit

2

Dequeued: 23

The element 23 is removed

1. Insert
2. Delete
3. Exit 2. Underflow: Circular queue is empty.

LABORATORY-4 11TH JANUARY, 2024 THURSDAY

1) Write a program in C to implement Singly Linked List with:
(a) Create a linked list (b) Insertion of a node at first position, at any position and at end of list. (c) Display the contents of the linked list.

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

struct Node
  {
    int data;
    struct Node *next;
  };

struct Node *insertstart (struct Node *start, int data)
  {
    struct Node *ptr = (struct Node *) malloc ( sizeof ( Struct No
    ptr -> data = data;
    ptr -> next = start;
    return ptr;
  }

struct Node * insertIndex (struct Node *start, int data, int indx
  {
  Struct Node *ptr = (struct Node *) malloc (sizeof ( struct Node));
  ptr -> data = data;

  struct Node *p = start;
  int i = 0;
  while (i < index - 1 && p != NULL)
   {
    p = p -> next;
    i++;
```

```
}
if (p==NULL)
{
  printf("Index out of bounds:\n");
  free(ptr);
  return start;
}
}
ptr->next=p->next;
p->next =ptr;
return start;
}

struct Node * InsertEnd (struct Node * start, int data)
{
  struct Node * ptr = (struct Node *) malloc (sizeof(struct Node));
  ptr -> data =data;
  ptr->next = NULL;
  if (start ==NULL)
  {
    return ptr;
  }
  struct Node *p = start;
  while (p->next!=NULL)
  {p=p->next;
  }
  p->next=ptr;
  return start;
}

void display (struct Node *ptr)
{
  while (ptr!=NULL)
  {
    printf("Element: %d \n", ptr->data);
```

```
        ptr = ptr->next;
    }
}
int main()
{
    struct Node *first = NULL;
    char choice;
    int newData, newIndex;

    do{
    printf("\n Choose an option:\n");
    printf(" 1. Insert at the beginning \n");
    printf(" 2. Insert at a specific index\n");
    printf(" 3. Insert at end\n");
    printf("4. Display the list\n");
    printf("5. Quit\n");
    printf("Enter your choice: ");
    scanf("%c", &choice);

    switch (choice)
    {
    Case '1':
        printf("Enter the new element to insert at the beginning:")
        scanf("%d", &newData);
        first = insertatfirst(first, newData);
        break;
    Case '2':
        printf("Enter the new element to insert.");
        scanf("%d", &newData);
        printf("Enter the index to start at:");
        scanf("%d", &newIndex);
        first = insertIndex(first, newData, newIndex);
        break;
```

```
case '3':
    printf("Enter the new element to insert: ");
    scanf("%d", &newData);
    first = InsertEnd(first, newData);
    break;

case '4':
    printf("Linked List:\n");
    display(first);
    break;

case '5':
    printf("Quitting the program.\n");
    break;

default:
    printf("Invalid choice. Please enter a valid choice.\n");
}

} while (choice != '5');
return 0;
}
```

Output:

```
Choose an options:
1. Insert at the beginning
2. Insert at a a specific index
3. Insert at end
4. Display the list
5. Quit.
Enter your choice: 1
Enter the new element to insert at the beginning: 1
```

Choose an option:
1. Insert at the beginning
2. Insert at a specific index
3. Insert at end
4. Display the list
5. Quit
Enter your choice: 3
Enter the new element to insert: 2

Choose an option:
1. Insert at the beginning
2. Insert at a specific index
3. Insert at end
4. Display the list
5. Quit
Enter your choice: 2
Enter the new element to insert: 3
Enter the index to insert at: 1

Choose an option:
1. Insert at the beginning
2. Insert at a specific index
3. Insert at end
4. Display the list
5. Quit
Enter your choice: 4
Linked List:
Element: 1
Element: 3
Element: 2

Choose an option:
1. Insert at the beginning
2. Insert at a specific index

3. Insert at end

4. Display the list

5. Quit

Enter your choice: 5

Quitting the program.

16/1/24

Leet code

LABORATORY - 5 18TH JANUARY, 2024 THURSDAY

1) Write a program in C to implement a Singly Linked List with following operations.

(a) Create a Singly Linked List.

(b) Deletion of first element, specified element and last element in the list.

(c) Display the contents of the linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node
  {
  int data;
  struct Node *next;
  };

void addAtIndex (struct Node** head, int index, int data)
  {
  struct Node* newNode = (struct Node*) malloc (sizeof(struct
  newNode->data = data;

  if (index == 0)
    {
    newNode->next = *head;
    *head = newNode;
    }
  else
    {
    struct Node* temp = *head;
    for (int i=0; i<index-1 && temp != NULL; i++)
      {
      temp = temp->next;
      }
```

```
if (temp == NULL)
  {
  printf("Invalid index!\n");
  free(newNode);
  return;
  }
}
newNode->next = temp->next;
temp->next = newNode;
}
printf("Element added at index %d\n", index);
}

void deleteAtStart(struct Node ** head)
{
  if (*head == NULL)
  {
    printf("list is empty, cannot delete.\n");
    return;
  }
  struct Node* temp = *head;
  *head = temp->next;
  free(temp);
  printf("Element deleted at start\n");
}

void deleteAtIndex(struct Node ** head, int index)
{
  if (*head == NULL)
  {
    printf("List is empty. cannot delete\n");
    return;
  }
  struct Node* temp = *head;
  if (index == 0)
```

```
{
    *head = temp->next;
    free(temp);
    printf("Element deleted at index 0\n");
}
}
else
{
    for(int i=0; i<index-1 && temp!=NULL; i++)
    {
        temp=temp->next;
    }
    if(temp==NULL || temp->next==NULL)
    {
        printf("Invalid index\n");
        return;
    }
    struct Node* toDelete = temp->next;
    temp->next = toDelete->next;
    free(toDelete);
    printf("Element deleted at index %d\n", index);
}
}

void deleteAtEnd(struct Node** head)
{
    if(*head==NULL)
    {
        printf("List is empty, cannot delete\n");
        return;
    }

    struct Node* temp = *head;
    struct Node* prev = NULL;
```

```
while (temp->next != NULL)
    {
        prev = temp;
        temp = temp->next;
    }
if (prev == NULL)
    {
        *head = NULL;
    }
else
    {
        prev -> next = NULL;
    }
free (temp);
printf ("Element deleted at the end \n");
}

void displayList (struct Node *head)
{
    printf ("Linked list : ");
    while (head != NULL)
    {
        printf ("%d ", head->data);
        head = head->next;
    }
printf ("\n");
}

int main()
{
    struct Node *head = NULL;
    int choice, index, data;

while (1)
```

```
{
printf("\n1. Add element at a given index\n");
printf("2. Delete at start\n");
printf("3. Delete at index\n");
printf("4. Delete at end\n");
printf("5. Display\n");
printf("6. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch(choice)
{
case 1: printf("Enter index and data to be added");
        scanf("%d %d", &index, &data);
        addatIndex(&head, index, data);
        break;
case 2: deleteAtStart(&head);
        break;
case 3: printf("Enter index to delete: ");
        scanf("%d", &index);
        deleteAtIndex(&head, index);
        break;
case 4: deleteAtEnd(&head);
        break;
case 5: displayList(head);
        break;
case 6: exit(0);
default: printf("Invalid choice\n");
}
}

return 0;
}
```

Output:

MENU

Enter your choice: 1

Enter index and data to add: 0 1

Element added at index 0

MENU

Enter your choice: 1

Enter index and data to add: 1 2

Element added at index 1

MENU

Enter your choice: 4

Element deleted at the end

MENU

Enter your choice: 5

Linked list: 1

MENU

Enter your choice: 2

Element deleted at the start

MENU

Enter your choice: 6

Process Returned 0(0x0) execution time: 184.258 s

Press any key to continue;

Sahil
28/1/24

LABORATORY - 6 25TH JANUARY, 2024 THURSDAY

1) Write a program in C to implement Singly linked list with following operations: Sort the linked list, Reverse the linked list, concatenation of two linked lists.

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

struct Node {
  int data;
  struct Node * next;
};

struct Node * insert (struct Node * first, int data)
{
  struct Node * newNode = (struct Node *) malloc (sizeof(struct Node))
  newNode->data = data;
  newNode->next = NULL;

  if (first == NULL)
  {
    return newNode;
  }

  struct Node * ptr = first;
  while (ptr-> next != NULL)
  {
    ptr = ptr->next;
  }
  ptr -> next = newNode;
  return first;
}
```

```
void sort(struct Node *first)
{
    if(first==NULL)
    {
        printf("List is empty.\n");
        return;
    }

struct Node* ptr;
struct Node * p;

for(ptr = first; ptr!=NULL; ptr = ptr->next)
    {
        bool swapped = false;
        for(p = first; p->next != NULL; p = p->next)
        {
            if(p->data > (p->next)->data)
            {
                int temp = p->data;
                p->data = (p->next)->data;
                (p->next)->data = temp;
                swapped = true;
            }
        }
    }
    if(!swapped)
    {
        break;
    }
}
struct Node* p1 = first;
while(p1 != NULL)
{
    printf("Element: %d", p1->data);
    p1 = p1->next;
```

```
    }
    printf("\n");
}

void display (struct Node* first)
{
    struct Node* p1 = first;
    while(p1 != NULL)
    {
        printf("Element: %d", p1->data);
        p1 = p1->next;
    }
    printf("\n");
}

struct Node* reverse(struct Node* first)
{
    struct Node* prev = NULL;
    struct Node* current = first;
    struct Node* next = NULL;

    while (current != NULL)
    {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}

int main()
{
    struct Node* first = NULL;
```

```
int n, ele;
do
{
  printf("\n1. Add Element\n2. Sort\n3. Reverse\n4. Display
  \n5. Exit\n");
  scanf("%d", &n);
  switch (n)
  {
    Case 1:  printf("Enter the data: \n");
             scanf("%d", &ele);
             first = insert(first, ele);
             break;
    Case 2:  sort(first);
             break;
    Case 3:  first = reverse(first);
             printf("List reversed\n");
             break;
    Case 4:  display(first);
             break;
    Case 5:  exit(0);
    default: printf("Enter correct choice\n");
  }
} while(5);

return(0);
}
```

<u>Output:</u>

```
1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
1
```

Enter Student data:
-10

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
1

Enter the data:
2

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
1

Enter the data:
0

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
2

Element: -10 Element: 0 Element: 2

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
4

Element: 2 Element: 0 Element: -10

1. Add Element
2. Sort
3. Reverse
4. Display
5. Exit
5

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

struct Node {
    int data;
    struct Node* next;
};

struct Node* createNode(int data)
{
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
```

```
newNode->data = data;
newNode->next = NULL;
return newNode;
}

void printList(struct Node* head)
{
  while(head != NULL)
  {
    printf("%d->", head->data);
    head = head->next;
  }
  printf("NULL\n");
}

struct Node* concatenateLists(struct Node* list1, struct Node
  list2)
{
  if (list1 == NULL)
  {
    return list2;
  }

  struct Node* current = list1;
  while(current->next != NULL)
  {
    current = current->next;
  }
  current->next = list2;
  return list1;
}
```

```
int main()
{
  struct Node * list1 = CreateNode(1);
  list1 -> next = CreateNode(2);
  list1 -> next -> next = CreateNode(3);

  struct Node * list2 = CreateNode (4);
  list2 -> next = CreateNode(5);
  list2 -> next -> next = Create Node(6);

  printf("Original List 1: ");
  printList (list1);
  printf("Original List 2: ");
  printList(list2);

  list1 = concatenateLists (list1, list2);
  printf("Concatenated List: ");
  printList (list1);
  free(list1);
  free(list2);
  return 0;
}
```

Output:

Original List 1: 1 -> 2 -> 3 -> NULL
Original List 2: 4 -> 10 -> 6 -> NULL
Concatenated List: 1 -> 2 -> 3 -> 4 -> 10 -> 6 -> NULL

2) Write a program in C to implement Single linked list, which simulate Stack operations.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
```

```
int data;
struct Node* next;
};

struct Node* push(struct Node* top, int data)
{
struct Node* newNode = (struct Node*) malloc(sizeof(struct Node
newNode->data =data;
if (top == NULL)
{
newNode->next = NULL;
top = newNode;
return top;
}
newNode->next = top;
top = newNode;
return top;
}

struct Node* pop(struct Node* top)
{
if (top == NULL)
{
printf("Under-flow");
exit(1);
}
struct Node *ptr = top;
top = ptr->next;
free(ptr);
return top;
}

void display(struct Node* top)
{
```

```
if(top == NULL)
{
printf("Stack is empty \n");
return;
}
printf("Stack elements: ");
Struct Node * current = top;
while(current != NULL)
{
printf("%d ", current->data);
current = current->next;
}
printf("\n");
}
int main()
{
Struct Node * top = NULL;
int choice, element;
do
{
printf("\n Stack Menu: \n");
printf("1. Push \n");
printf("2. Pop \n");
printf("3. Display \n");
printf("4. Exit \n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch(choice)
{
case 1:
printf("Enter the element to push:");
scanf("%d", &element);
top = push(top, element);
break;
```

```
Case 2:
    top = pop(top);
    break;
Case 3:
    display(top);
    break;
Case 4:
    printf("Exiting the program.\n");
    break;
default:
    printf("Invalid choice. Please enter a valid option.\n");
}
} while(choice != 4);

while(top != NULL)
{
  Struct Node * temp = top;
  top = top -> Next;
  free(temp);
}
return 0;
```

Output:

Stack Menu:
1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 1
Enter the element to push: 12

Stack Menu:
1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 1
Enter the element to push: -3

Stack Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack Elements: -3 12

Stack Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Enter your choice: 2
Enter your choice: 2

Stack Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Stack Underflow

Stack Menu:
1. Push
2. Pop
3. Display
4. Exit
Exiting the program

3) Write a program in C to implement Single Linked list to simulate Stack operations.

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

struct Node
{
  int data;
  struct Node * next;
};

struct Node* enqueue(struct Node* front, struct Node* rear, int data)
{
  struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
  newNode->data = data;
  newNode->next = NULL;
```

```
if (front == NULL)
{
// Queue is empty
front = rear = newNode;
}
else
{
rear->next = newNode;
rear = newNode;
}
return rear; // Return the updated rear
}

void display(struct Node* front, struct Node* rear)
{
if (rear = NULL)
{
printf("queue is empty.\n");
return;
}
printf("queue elements: ");
struct Node*ptr = front;
while (ptr != NULL)
{
printf("%d", ptr->data);
ptr = ptr->next;
}
printf("\n");
}

int main()
{
Struct Node* front = NULL;
Struct Node* rear = NULL;
```

```
int choice, element;
do {
    printf("\n Queue Menu:\n");
    printf("1. Enqueue\n");
    printf("2. Dequeue\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
    Case 1:
        printf("Enter the element to enqueue:");
        scanf("%d", &element);
        rear = enqueue(front, rear, element);
        if (front==NULL)
        {
            //Update front when queue was empty
            front=rear;
        }
    break;
    Case 2:
        front = dequeue(front, rear);
        break;
    Case 3:
        display(front, rear);
        break;
    Case 4:
        printf("Exiting the program.\n");
        break;
    default:
        printf("Invalid choice. Please enter a valid option\n");
    }
} while (choice!=4);
```

```
//Free memory before exiting the program
while (front != NULL)
  {
    struct Node * temp = front;
    front = front->next;
    free(temp);
  }
return 0;
}
```

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice : 1
Enter the element to enqueue : 12

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice : 1
Enter the element to enqueue 3

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice : 2

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice : 2

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice : 3
Queue is empty

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice : 2
Queue underflow

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice : 4
Exiting the program

LABORATORY - 9 29TH FEBRUARY, 2024 THURSDAY

Write a program in C to traverse a graph using Breadth First Search method and to check whether given graph is connected or not using Depth First Search method.

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_NODES 100;
#define MAX_EDGES 100;

int graph[MAX NODES][MAX NODES];
int visited[MAX NODES];

void DFS(int start, int n)
{
  visited[start] = 1;
for(int i= 0; i<n; i++)
  {
   if(graph[start][i] == 1 && !visited[i])
    {
     DFS(i,n);
    }
  }
}

int isconnected(int n)
  {
    DFS(0,n);
for(int i=0; i<n; i++)
  {
    if(!visited[i])
     {
      return 0;
     }
  }
```

```
return 1;
}

int main()
{
int n, m;
printf("Enter the number of nodes and edges:");
scanf("%d %d", &n &m);
printf("Enter the edges:\n");
for(int i=0; i<m; i++)
{
int a, b;
scanf("%d %d", &n &m);
printf("Enter the edges:\n");
for(int i=0; i<m; i++)
{
int a, b;
scanf("%d %d", &a, &b);
graph[a][b]=1;
graph[b][a]=1
}

if (isConnected(n)),
{
printf("The graph is connected.\n");
}
else
{
printf("The graph is connected\n");
}
return 0;
}
```

Output:
BFS traversal: 0 1 4 2 3
Graph is connected.

2) Hackerrank:

```
void inOrderTraversal(Node *root, int * result, int *index)
{
    if(root == NULL) return;
    inOrderTraversal(root->left, result, index);
    result[*index++] = root->data;
    inOrderTraversal(root->right, result, index);
}

void swapAtlevel(Node *root, int k, int level)
{
    if (root == NULL)
        return;

    if (level % k == 0)
    {
        Node *temp = root->left;
        root->left = root->right;
        root->right = temp;
    }
```

29/2/22

LABORATORY - 10 29TH FEBRUARY, 2024 THURSDAY.

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

#define size 10

int table[size];

void push(int data);
int pop (int data);
void search (int data);
void display ();

int main ()
{
  for (int i=0; i<size; i++)
    table [i] = -1;
  int choice;
  printf ("1. Insert \n 2. Delete \n 3. Display \n 4. Exit \n Choice ")
  scanf ("%d", &choice);
  int a;
  while(choice!=4)
  {
    switch (choice)
    {
      Case 1:
        printf ("Enter integer to be pushed:");
        scanf ("%d", &a);
        push (a);
        break;
      Case 2:
        printf ("Enter integer to be popped(\n)
```

```
Acanf("%d", &a);
int res = pop(a);
if (res == 0) printf("Integer popped\n");
else printf("Integer not found\n");
break;
case 3:
    display();
    break;
default:
    printf("Enter choice: ");
    Scanf("%d", &choice);
}
printf("Enter choice: ");
scanf("%d", &choice);
}
}


void push(int data)
{
    int hash = data % size;
    while (table[hash] != -1 && hash <= (hash + size - 1))
        hash = (hash + 1) % size;
    if (table[hash] == -1)
        table[hash] = data;
    else
        printf("Table is full");
}

int pop(int data);
{
    int hash = data % size;
    for (int i=0; (table[hash] != data) || (i < size); i++)
        hash = (hash-1) % size);
```

```
if (table[hash] == data)
  {
    table[hash] = -1;
    return 0;
  }
return -1;
}

void display()
  {
    printf("Table: ");
    for (int i = 0; i < size; i++)
      printf("%d ", table[i]);
    printf("\n");
  }
```

1. Insert
2. Search
3. Delete
4. Exit

Choice: 1

Enter integer to pushed: 1

Enter choice: 1

Enter integer to be pushed: 3

Enter choice: 1

Enter integer to be pushed: 90

Enter choice: 1

Enter integer to be pushed: 13

Enter choice: 3

Table: 90 1 -1 3 -1 -1 -1 -1 -1 -1

Enter integer to be pushed: 4

Table: 90 1 -1 3 13 4 -1 -1 -1 -1

29/12/24