1)practical 1 : Experiment 1: Write a C Program that Calculates Sum of Natural Numbers

```
#include <stdio.h>

int main() {
   int n, i;
   int sum = 0;

   printf("Enter a positive integer: ");
   scanf("%d", &n);

   // Validate input
   if (n <= 0) {
      printf("Please enter a positive number.\n");
      return 1;
   }

   for (i = 1; i <= n; ++i) {
      sum += i;
   }

   printf("The sum of the first %d natural numbers is: %d\n", n, sum);

   return 0;
}
```

nano sum_natural.c

gcc sum_natural.c -o sum_natural

./sum_natural

Paste the code and press Ctrl + O, then Enter, and Ctrl + X to exit.

2) Experiment 2: Python Code that Calculates Sum of Natural Numbers

```
# Program to calculate sum of first N natural numbers

# Take input from user
n = int(input("Enter a positive integer: "))

# Initialize sum
total = 0

# Validate input
if n <= 0:
   print("Please enter a positive number.")
else:
   for i in range(1, n + 1):
      total += i
   print(f"The sum of the first {n} natural numbers is: {total}")
```

```
nano sum_natural.py
python3 sum_natural.py
```

Experiment 3: Create Three C Files Containing Different Arithmetic Operations and Execute Multiple C Programs on Linux

main.c file:

```
#include <stdio.h>

void add(int a, int b);
void sub(int a, int b);
void mul(int a, int b);

int main() {
   int a, b;
   printf("Enter two numbers: ");
   scanf("%d %d", &a, &b);

   add(a, b);
   sub(a, b);
   mul(a, b);

   return 0;
}
```

add.c file :

```
#include <stdio.h>
void add(int a, int b) {
   printf("Addition: %d\n", a + b);
}
```

sub.c file :

```
#include <stdio.h>
void sub(int a, int b) {
   printf("Subtraction: %d\n", a - b);
}
```

mul.c file :

```
#include <stdio.h>
void mul(int a, int b) {
   printf("Multiplication: %d\n", a * b);
}
```

```
gcc main.c add.c sub.c mul.c -o result
./result
```

Experiment 4: Write a C Program for a Simple Calculator and Execute on Linux (Ubuntu)

```
#include <stdio.h>

int main() {
    int a, b, choice;
    float div_result;

    while (1) {
        printf("\n=== SIMPLE CALCULATOR ===\n");
        printf("1. Addition\n");
        printf("2. Subtraction\n");
        printf("3. Multiplication\n");
        printf("4. Division\n");
        printf("5. Modulus\n");
        printf("6. Exit\n");
        printf("Enter your choice (1–6): ");
        scanf("%d", &choice);

        // Exit condition
        if (choice == 6) {
            printf("Exiting Calculator... Goodbye!\n");
            break;
        }

        // Take input for numbers
        printf("Enter two integers: ");
        scanf("%d %d", &a, &b);

        switch (choice) {
```

```
        case 1:
            printf("Result: %d + %d = %d\n", a, b, a + b);
            break;
        case 2:
            printf("Result: %d - %d = %d\n", a, b, a - b);
            break;
        case 3:
            printf("Result: %d * %d = %d\n", a, b, a * b);
            break;
        case 4:
            if (b == 0)
                printf("Error: Division by zero not allowed!\n");
            else {
                div_result = (float)a / b;
                printf("Result: %d / %d = %.2f\n", a, b, div_result);
            }
            break;
        case 5:
            if (b == 0)
                printf("Error: Modulus by zero not allowed!\n");
            else
                printf("Result: %d %% %d = %d\n", a, b, a % b);
            break;
        default:
            printf("Invalid choice! Please select between 1 and 6.\n");
    }
}

return 0;
```

```
}
```

```
gcc calculator.c -o calculator
```

```
./calculator
```

Experiment 5: Create and Execute a Shell Script of All Arithmetic Operations

```
#!/bin/bash
# Arithmetic Operations Script

echo "=== Simple Arithmetic Operations ==="
echo -n "Enter first number: "
read a
echo -n "Enter second number: "
read b

# Addition
sum=$((a + b))
echo "Addition: $a + $b = $sum"

# Subtraction
sub=$((a - b))
echo "Subtraction: $a - $b = $sub"

# Multiplication
mul=$((a * b))
echo "Multiplication: $a * $b = $mul"

# Division (check divide-by-zero)
if [ $b -eq 0 ]; then
   echo "Division: Error! Division by zero is not allowed."
else
   div=$((a / b))
   echo "Division: $a / $b = $div"
fi
```

```
echo "=== End of Program ==="
```

```
nano arithmetic.sh
```

```
chmod +x arithmetic.sh
```

```
./arithmetic.sh
```

Experiment 6: Create and Execute Shell Script of Loops

loops.sh

```
#!/bin/bash
# Shell Script Demonstrating For, While, and Until Loops

echo "=== Loop Demonstration ==="

# -------- For Loop --------
echo -e "\n1 For Loop: Print numbers from 1 to 5"
for i in {1..5}
do
   echo "Number: $i"
done

# -------- While Loop --------
echo -e "\n2 While Loop: Print numbers from 1 to 5"
count=1
while [ $count -le 5 ]
do
   echo "Count: $count"
   ((count++))
done

# -------- Until Loop --------
echo -e "\n3 Until Loop: Print numbers from 1 to 5"
num=1
until [ $num -gt 5 ]
```

```
do
    echo "Value: $num"
    ((num++))
done


echo -e "\n=== End of Program ==="


nano loops.sh
chmod +x loops.sh
./loops.sh
```

**Experiment 7: Create Shell Script of Conditional Statements and Switch Case in Linux and Execute**

condition.sh

```
#!/bin/bash
# Script to check whether a number is positive, negative, or zero

echo -n "Enter a number: "
read num

if [ $num -gt 0 ]; then
   echo "$num is Positive"
elif [ $num -lt 0 ]; then
   echo "$num is Negative"
else
   echo "$num is Zero"
fi
```

```
nano condition.sh
chmod +x condition.sh
./condition.sh
```

case_menu.sh

```
#!/bin/bash
# Menu-driven calculator using case statement
```

```
echo "=== Simple Calculator ==="
echo "1. Addition"
echo "2. Subtraction"
echo "3. Multiplication"
echo "4. Division"
echo -n "Enter your choice (1-4): "
read choice

echo -n "Enter first number: "
read a
echo -n "Enter second number: "
read b

case $choice in
  1)
    echo "Result: $((a + b))"
    ;;
  2)
    echo "Result: $((a - b))"
    ;;
  3)
    echo "Result: $((a * b))"
    ;;
  4)
    if [ $b -eq 0 ]; then
      echo "Error: Division by zero not allowed."
    else
      echo "Result: $((a / b))"
    fi
```

```
        ;;
    *)
        echo "Invalid choice! Please select between 1–4."
        ;;
Esac
```

```
nano case_menu.sh
chmod +x case_menu.sh
./case_menu.sh
```

Experiment 8: Create a User “eln”, Give Credentials, Add it to Group “finalyear”, and Delete it

| Task | Command | Description |
|---|---|---|
| Create group | `sudo groupadd finalyear` | Makes new group |
| Create user | `sudo useradd -m -s /bin/bash eln` | Makes user “eln” |
| Set password | `sudo passwd eln` | Gives password |
| Add user to group | `sudo usermod -aG finalyear eln` | Adds user “eln” to “finalyear” |
| Verify groups | `groups eln` | Lists group memberships |
| Switch user | `su - eln` | Login as that user |
| Delete user | `sudo userdel -r eln` | Deletes user and home |
| Delete group | `sudo groupdel finalyear` | Removes the group |

```
# 1. Create group and user
sudo groupadd finalyear
sudo useradd -m -s /bin/bash eln
sudo passwd eln

# 2. Add user to group
sudo usermod -aG finalyear eln

# 3. Verify
id eln
groups eln

# 4. Delete
sudo userdel -r eln
sudo groupdel finalyear
```

# Experiment 9: Create and Execute Python Program with Conditional Statements and Loop Operations; Execute Multiple Files

condition.py

```
# condition.py
# Program to check whether a number is positive, negative, or zero

num = int(input("Enter a number: "))

if num > 0:
    print(f"{num} is Positive")
elif num < 0:
    print(f"{num} is Negative")
else:
    print("Number is Zero")
```

loop.py

```
# loop.py
# Program to print numbers from 1 to N using a loop

n = int(input("Enter the range: "))

print(f"Numbers from 1 to {n}:")
for i in range(1, n + 1):
    print(i)
```

```
python3 condition.py
python3 loop.py
```

Experiment 10: Create and Execute a Single Python File for Arithmetic Operations in Linux, then Create Another Python File and Execute Both Files Using a Shell Script

arithmetic.py

```
# arithmetic.py

# Program to perform basic arithmetic operations


a = int(input("Enter first number: "))

b = int(input("Enter second number: "))


print("\n--- Arithmetic Operations ---")


print(f"Addition: {a} + {b} = {a + b}")

print(f"Subtraction: {a} - {b} = {a - b}")

print(f"Multiplication: {a} * {b} = {a * b}")


if b == 0:

    print("Division: Error! Division by zero not allowed.")

    print("Modulus: Error! Division by zero not allowed.")

else:

    print(f"Division: {a} / {b} = {a / b:.2f}")

    print(f"Modulus: {a} % {b} = {a % b}")
```

summary.py

```
print("\n==============================")
print("     Arithmetic Program Done!  ")
print("  Executed Successfully in Linux ")
print("==============================\n")
```

run_both.sh

```
#!/bin/bash
# Script to execute both Python files sequentially

echo "Starting Arithmetic Operation..."
python3 arithmetic.py

echo "Running Summary Program..."
python3 summary.py

echo "All Python programs executed successfully!"
```

```
nano arithmetic.py
nano summary.py
nano run_both.sh
chmod +x run_both.sh
./run_both.sh
```

Experiment 11: Create 3 Different C Files and Execute Them in Linux; Execute the Same Three Files Using a Makefile

RTOS_Arithmetic/

├── add.c

├── sub.c

├── mul.c

├── main.c

├── operations.h

└── **Makefile**

# Step 1: Header File — `operations.h`

```
// operations.h
// Header file for arithmetic operations

#ifndef OPERATIONS_H
#define OPERATIONS_H

// Function prototypes
int add(int a, int b);
int sub(int a, int b);
int mul(int a, int b);

#endif
```

# Step 2: Source Files

add.c

```
#include "operations.h"

int add(int a, int b) {
    return a + b;
}
```

---

sub.c

```
#include "operations.h"

int sub(int a, int b) {
    return a - b;
}
```

---

mul.c

```
#include "operations.h"
```

```
int mul(int a, int b) {
    return a * b;
}
```

---

## 💻 Step 3: Main File — `main.c`

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

int main(void) {
    int a, b;

    printf("Enter two numbers: ");
    if (scanf("%d %d", &a, &b) != 2) {
        printf("Invalid input!\n");
        return 1;
    }

    printf("Addition: %d\n", add(a, b));
    printf("Subtraction: %d\n", sub(a, b));
    printf("Multiplication: %d\n", mul(a, b));

    return 0;
}
```

Step 4: Makefile

CC = gcc

TARGET = final

SRC = main.c add.c sub.c mul.c

HEADERS = operations.h

$(TARGET): $(SRC) $(HEADERS)

	$(CC) $(SRC) -o $(TARGET)

clean:

	rm -f *.o $(TARGET)

to run :

make

./final

12)experiment 12 : 2 leds

Minimal Two-Task LED Blinking (uC/OS-II)

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

#define STK 64

OS_STK T0[STK];
OS_STK T1[STK];

void Task0(void *pdata);
void Task1(void *pdata);

int main(void)
{
   LED_init();
   TargetInit();
   OSInit();

   OSTaskCreate(Task0, 0, &T0[STK - 1], 6);
   OSTaskCreate(Task1, 0, &T1[STK - 1], 7);

   OSStart();
   return 0;
}

/************ Task 0: Blink LED 0 ************/
void Task0(void *pdata)
```

```
{
   while (1)
   {
      LED_on(0);
      OSTimeDly(5);
      LED_off(0);
      OSTimeDly(5);
   }
}

/************ Task 1: Blink LED 1 ************/
void Task1(void *pdata)
{
   while (1)
   {
      LED_on(1);
      OSTimeDly(5);
      LED_off(1);
      OSTimeDly(5);
   }
}
```

13) Experiment 3 : 3 leds

**Short & Simple 3-Task LED Blinking (uC/OS-II)**

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

#define STK 64   // Task stack length

OS_STK T0[STK];
OS_STK T1[STK];
OS_STK T2[STK];

void Task0(void *pdata);
void Task1(void *pdata);
void Task2(void *pdata);

int main(void)
{
   LED_init();
   TargetInit();
   OSInit();

   OSTaskCreate(Task0, 0, &T0[STK - 1], 6);
   OSTaskCreate(Task1, 0, &T1[STK - 1], 7);
   OSTaskCreate(Task2, 0, &T2[STK - 1], 8);

   OSStart();
   return 0;
}
```

```
/************ Task 0: Blink LED0 ************/
void Task0(void *pdata)
{
   while (1)
   {
      LED_on(0);
      OSTimeDly(15);
      LED_off(0);
      OSTimeDly(15);
   }
}

/************ Task 1: Blink LED1 ************/
void Task1(void *pdata)
{
   while (1)
   {
      LED_on(1);
      OSTimeDly(25);
      LED_off(1);
      OSTimeDly(25);
   }
}

/************ Task 2: Blink LED2 ************/
void Task2(void *pdata)
{
   while (1)
   {
      LED_on(2);
      OSTimeDly(35);
```

```
        LED_off(2);
        OSTimeDly(35);
    }
}
```

14)Experiment 14 : semaphore

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

#define STK 64

OS_STK T0[STK];
OS_STK T1[STK];

OS_EVENT *sem;
unsigned char err;

void Task0(void *pdata);
void Task1(void *pdata);

int main(void)
{
   UART0_Init();
   TargetInit();
   OSInit();

   sem = OSSemCreate(1);        // Binary semaphore (mutex)

   OSTaskCreate(Task0, 0, &T0[STK-1], 6);
   OSTaskCreate(Task1, 0, &T1[STK-1], 7);

   OSStart();
   return 0;
}
```

```
void Task0(void *pdata)
{
   while(1)
   {
      UART0_SendData("\nTask0 waiting...");
      OSSemPend(sem, 0, &err);

      UART0_SendData(" Task0 got semaphore");
      OSTimeDly(100);          // Simulated work

      UART0_SendData(" Task0 released\n");
      OSSemPost(sem);
   }
}

void Task1(void *pdata)
{
   while(1)
   {
      UART0_SendData("\nTask1 waiting...");
      OSSemPend(sem, 0, &err);

      UART0_SendData(" Task1 got semaphore");
      OSTimeDly(100);          // Simulated work

      UART0_SendData(" Task1 released\n");
      OSSemPost(sem);
   }
}
```

15) experiment 15: without semaphore

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

#define STK 64

OS_STK T0[STK];
OS_STK T1[STK];

//OS_EVENT *sem;
//unsigned char err;

void Task0(void *pdata);
void Task1(void *pdata);

int main(void)
{
   UART0_Init();
   TargetInit();
   OSInit();

   //sem = OSSemCreate(1);        // Binary semaphore (mutex)

   OSTaskCreate(Task0, 0, &T0[STK-1], 6);
   OSTaskCreate(Task1, 0, &T1[STK-1], 7);

   OSStart();
   return 0;
}
```

```
void Task0(void *pdata)
{
   while(1)
   {
      UART0_SendData("\nTask0 waiting...");
     // OSSemPend(sem, 0, &err);
                              OSTimeDly(100);

      UART0_SendData(" Task0 got semaphore");
      OSTimeDly(100);          // Simulated work

      UART0_SendData(" Task0 released\n");
      //OSSemPost(sem);
   }
}

void Task1(void *pdata)
{
   while(1)
   {
      UART0_SendData("\nTask1 waiting...");
      //OSSemPend(sem, 0, &err);
                              OSTimeDly(100);

      UART0_SendData(" Task1 got semaphore");
      OSTimeDly(100);          // Simulated work

      UART0_SendData(" Task1 released\n");
     // OSSemPost(sem);
   }
}
```

16) experiment 16 : Mail box

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

#define STK 64

OS_STK T0[STK];
OS_STK T1[STK];

OS_EVENT *mbox;
uint8 err;

void Task0(void *pdata);
void Task1(void *pdata);

int main(void)
{
   LED_init();
   TargetInit();
   OSInit();

   mbox = OSMboxCreate(0);

   OSTaskCreate(Task0, 0, &T0[STK-1], 6);
   OSTaskCreate(Task1, 0, &T1[STK-1], 7);

   OSStart();
   return 0;
}

/**********************
```

```
 * Task0: Send message
 ***********************/
void Task0(void *pdata)
{
   unsigned int msg = 12;

   while(1)
   {
      LED_on(0);
      OSTimeDly(40);
      LED_off(0);
      OSTimeDly(40);

      OSMboxPost(mbox, &msg);   // send number 12
   }
}

/************************
 * Task1: Receive & blink
 ************************/
void Task1(void *pdata)
{
   unsigned int *ptr;
   int i;

   while(1)
   {
      ptr = OSMboxPend(mbox, 0, &err); // wait for message

      if(err == OS_NO_ERR)
      {
```

```
            for(i = 0; i < (*ptr - 5); i++)
            {
               LED_on(1);
               OSTimeDly(1);
               LED_off(1);
               OSTimeDly(1);
            }
         }
      }
   }
```