Experiment No. 2

CPL (C Programming Lab)

Aim : WAP to find the sum of odd numbers between two numbers entered by the user.

Software : Codeblocks & MingW

Theory : ***Control Structures***

1. if-else Statements

The if-else construct is used for decision-making in C programming. It allows the program to execute certain code blocks conditionally based on whether a specified condition is true or false.

- **if statement**: Executes a block of code if a given condition is true.

```
if (condition) {
   // Code to execute if condition is true
}
```

- **else statement**: Executes a block of code if the condition in the if statement is false.

```
if (condition) {
   // Code if condition is true
} else {
   // Code if condition is false
}
```

- **else if ladder**: Used when multiple conditions need to be checked.

```
if (condition1) {
   // Code if condition1 is true
} else if (condition2) {
   // Code if condition2 is true
} else {
   // Code if none of the conditions are true
}
```

Example:

```
int num = 5;
if (num > 0) {
    printf("Positive number");
} else if (num < 0) {
    printf("Negative number");
} else {
    printf("Zero");
}
```

2. Loops

Loops allow the repetition of a block of code multiple times, based on a condition. C has three types of loops:

a. for Loop:

Used when the number of iterations is known. It contains three parts: initialization, condition, and increment/decrement.

```
for (initialization; condition; increment/decrement) {
    // Code to be repeated
}
```

Example:

```
for (int i = 0; i < 5; i++) {
    printf("%d ", i);
}
```

b. while Loop:

Executes the code block as long as the condition remains true. It checks the condition before each iteration.

```
while (condition) {
    // Code to be repeated
}
```

Example:

```
int i = 0;

while (i < 5) {

    printf("%d ", i);

    i++;

}
```

c. do-while Loop:

Similar to the while loop but the condition is checked after executing the loop body, so it runs at least once.

```
do {

    // Code to be repeated

} while (condition);
```

Example:

```
int i = 0;

do {

    printf("%d ", i);

    i++;

} while (i < 5);
```

Post-Lab Questions:

1. **What changes would you make if the program needed to find the sum of even numbers instead of odd?**
   How can you modify the condition inside the loop to sum even numbers?

2. **How would you handle cases where the starting number is larger than the ending number?**
   What changes can be made to ensure the program works regardless of the order of input?

3. **What was the output when the starting and ending numbers were the same?**
   How does the program behave in this case, and is it expected?

4. **How can you modify the program to avoid counting negative numbers if they are entered?**
   If the user enters negative numbers, how can you ensure that only positive odd numbers are considered in the sum?

Task 1: **WAP to find if entered number is even or odd. (Draw flowchart also)**

Program with Output:

Task 2 : **WAP to find the sum of all the odd numbers between numbers entered by the user. (Draw flowchart also)**

Program with Output:

Conclusion:

……………………………………………………………………………………………………..........................

……..

CO's Covered:

……………………………………………………………………………………………………..........................

……..