Operators

26-10-25

Aim - Implement a program to demonstrate unary arithmetic logical relational bitwise and ternary operators

Theory –

Unary operators work on one operand to increment, decrement, or negate a value, changing its state.

Arithmetic operators perform basic mathematical calculations like addition, subtraction, multiplication, division, and modulus on operands.

Relational operators compare two values and return true or false based on their relationship or condition.

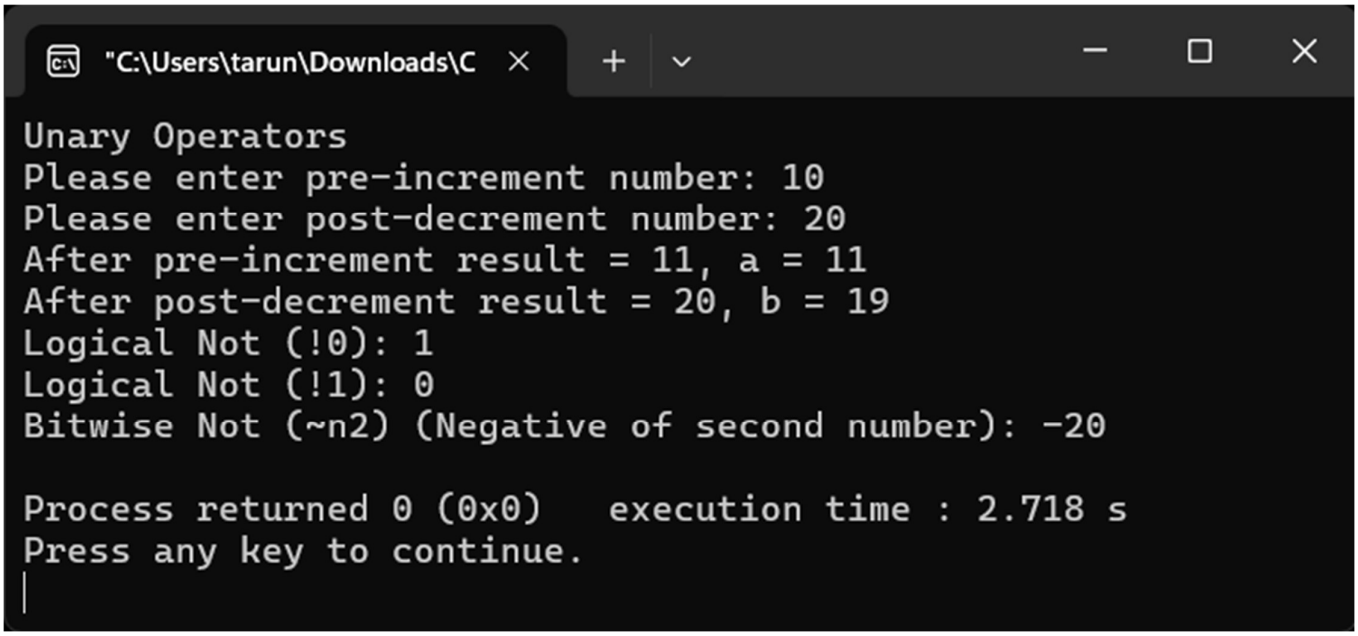
Logical operators combine multiple conditions and determine overall truth value using AND, OR, and NOT logic.

Bitwise operators manipulate data at the bit level using AND, OR, XOR, NOT, shift operations.

The ternary operator uses three operands to perform conditional evaluation like a compact if-else statement.

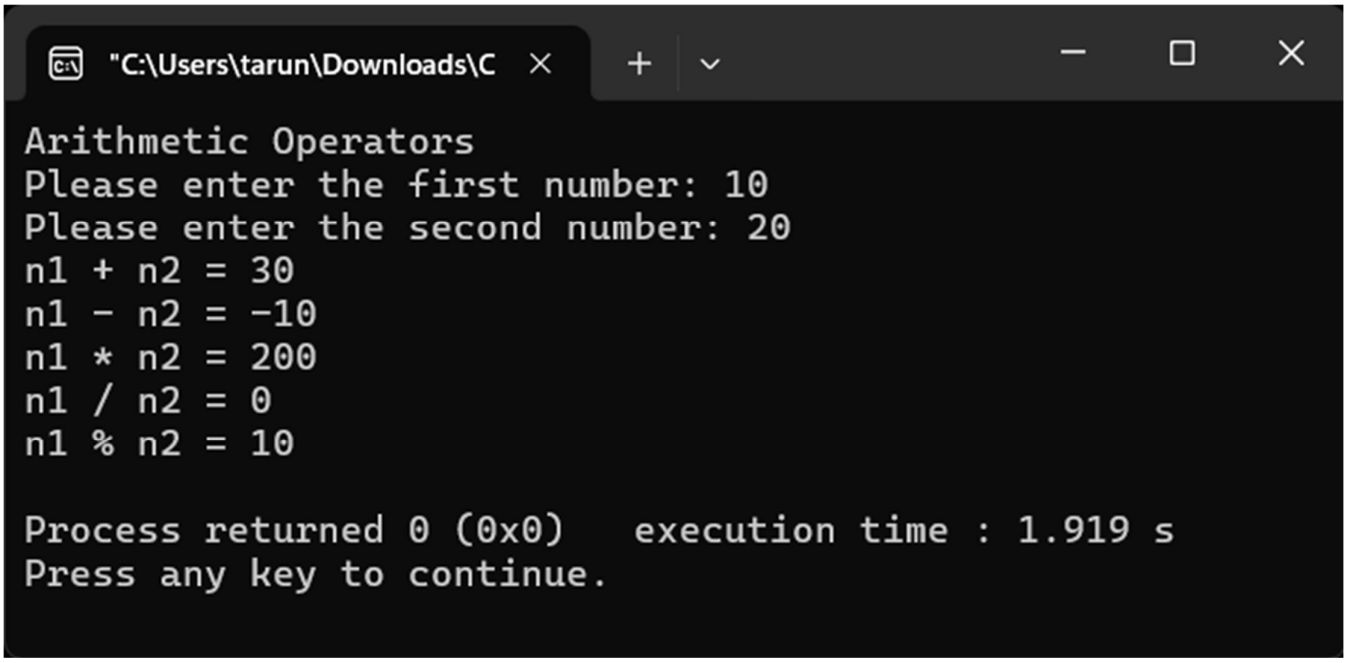
A1.

```
#include <stdio.h>
int main() {
    int n1, n2, result;
    printf("Unary Operators \n");
    printf("Please enter pre-increment number: ");
    scanf("%d", &n1);
    printf("Please enter post-decrement number: ");
    scanf("%d", &n2);
    result = ++n1;
    printf("After pre-increment result = %d, a = %d\n", result, n1);
    result = n2--;
    printf("After post-decrement result = %d, b = %d\n", result, n2);
    printf("Logical Not (!0): %d\n", !0);
    printf("Logical Not (!1): %d\n", !1);
    printf("Bitwise Not (~n2) (Negative of second number): %d\n", ~n2);
}
```



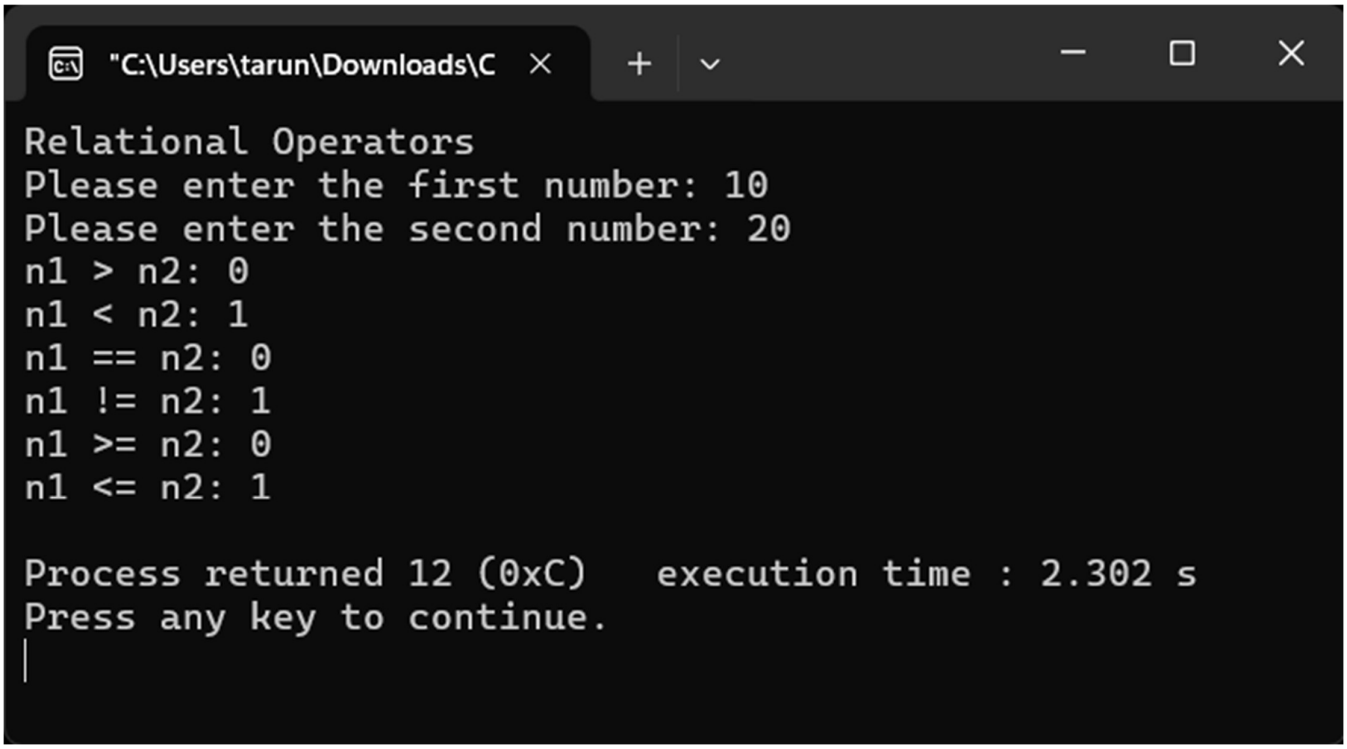
A2.

```
#include <stdio.h>
int main() {
    int n1, n2, result;
    printf("Arithmetic Operators\n");
    printf("Please enter the first number: ");
    scanf("%d", &n1);
    printf("Please enter the second number: ");
    scanf("%d", &n2);
    result = n1 + n2;
    printf("n1 + n2 = %d\n", result);
    result = n1 - n2;
    printf("n1 - n2 = %d\n", result);
    result = n1 * n2;
    printf("n1 * n2 = %d\n", result);
    result = n1 / n2;
    printf("n1 / n2 = %d\n", result);
    result = n1 % n2;
    printf("n1 %% n2 = %d\n", result);
}
```



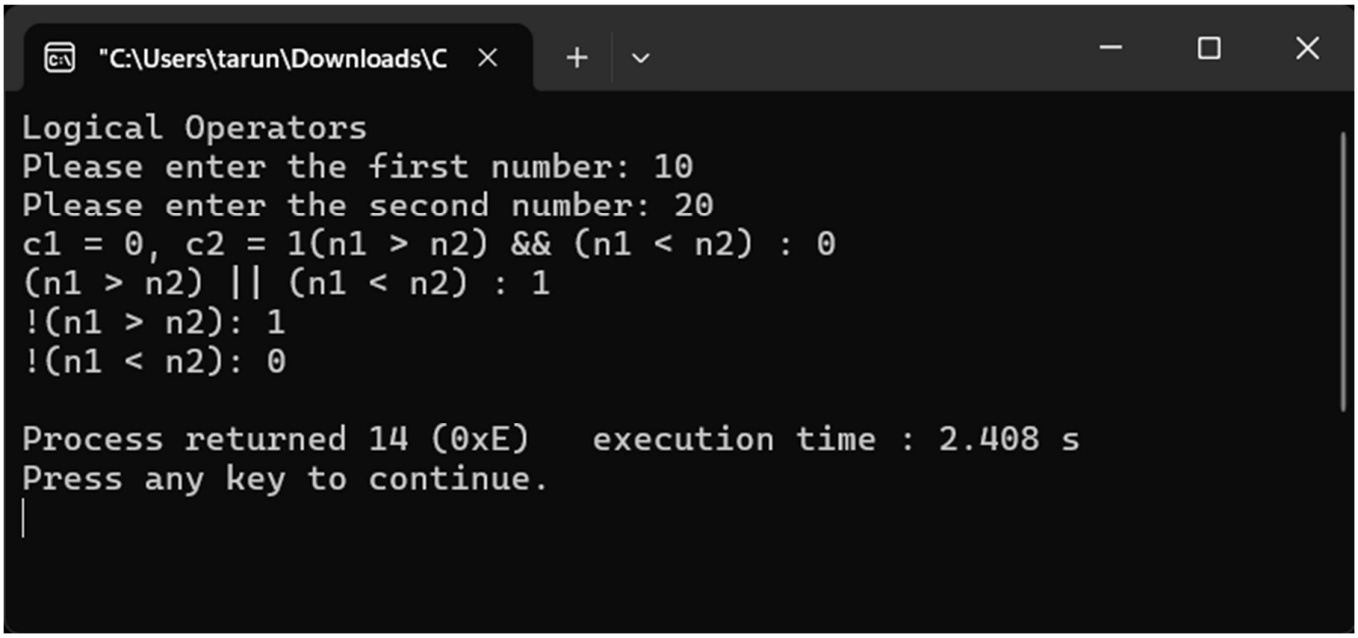
A3.

```
#include <stdio.h>
void main() {
    int n1, n2;
    printf("Relational Operators\n");
    printf("Please enter the first number: ");
    scanf("%d", &n1);
    printf("Please enter the second number: ");
    scanf("%d", &n2);
    printf("n1 > n2: %d\n", n1 > n2);
    printf("n1 < n2: %d\n", n1 < n2);
    printf("n1 == n2: %d\n", n1 == n2);
    printf("n1 != n2: %d\n", n1 != n2);
    printf("n1 >= n2: %d\n", n1 >= n2);
    printf("n1 <= n2: %d\n", n1 <= n2);
}
```



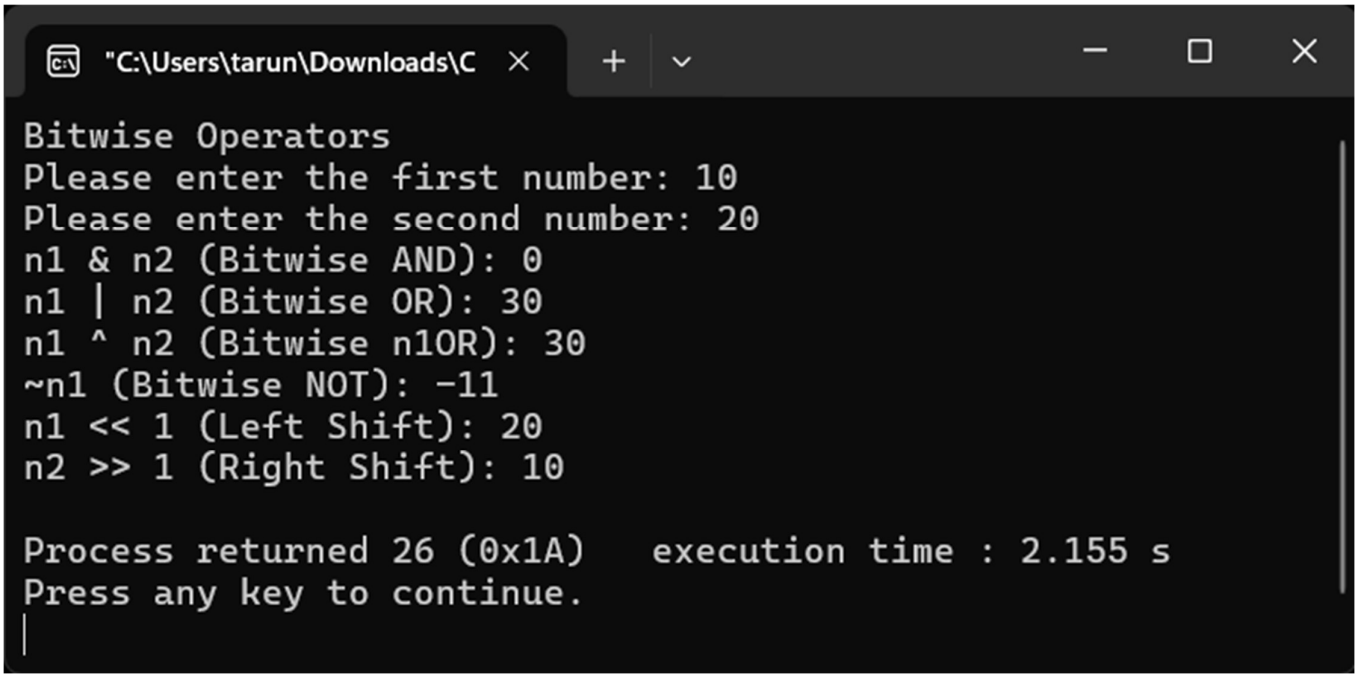
A4.

```
#include <stdio.h>
void main() {
    int n1, n2;
    printf("Logical Operators\n");
    printf("Please enter the first number: ");
    scanf("%d", &n1);
    printf("Please enter the second number: ");
    scanf("%d", &n2);
    int c1 = (n1 > n2);
    int c2 = (n1 < n2);
    printf("c1 = %d, c2 = %d", c1, c2);
    printf("(n1 > n2) && (n1 < n2) : %d\n", c1 && c2);
    printf("(n1 > n2) || (n1 < n2) : %d\n", c1 || c2);
    printf("!(n1 > n2): %d\n", !c1);
    printf("!(n1 < n2): %d\n", !c2);
}
```



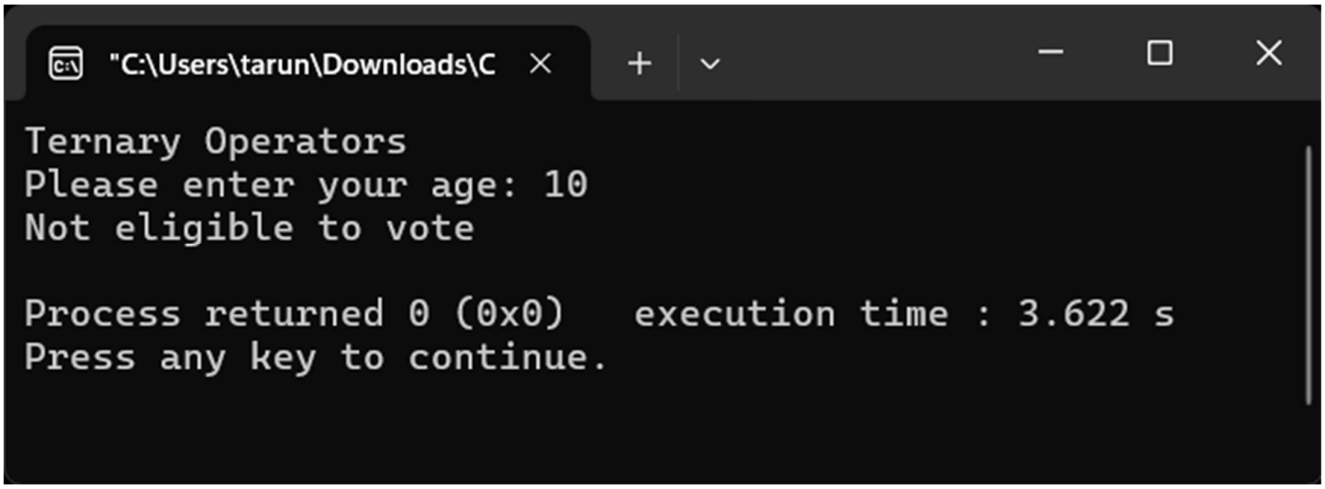
A5.

```
#include <stdio.h>
void main() {
    int n1, n2;
    printf("Bitwise Operators\n");
    printf("Please enter the first number: ");
    scanf("%d", &n1);
    printf("Please enter the second number: ");
    scanf("%d", &n2);
    printf("n1 & n2 (Bitwise AND): %d\n", n1 & n2);
    printf("n1 | n2 (Bitwise OR): %d\n", n1 | n2);
    printf("n1 ^ n2 (Bitwise n1OR): %d\n", n1 ^ n2);
    printf("~n1 (Bitwise NOT): %d\n", ~n1);
    printf("n1 << 1 (Left Shift): %d\n", n1 << 1);
    printf("n2 >> 1 (Right Shift): %d\n", n2 >> 1);
}
```



A6.

```
#include <stdio.h>
void main() {
    int age;
    printf("Ternary Operators\n");
    printf("Please enter your age: ");
    scanf("%d", &age);
    char* voterStatus = (age >= 18) ? "Eligible to vote" : "Not eligible to vote";
    printf("%s\n", voterStatus);
}
```



Conclusion

This program illustrates how various C operators work to carry out necessary tasks. Programs can be made more logical, efficient, and understandable by using unary, arithmetic, relational, ternary, and bitwise operators. They efficiently act as the fundamental basis of all C programming concepts and logic development by streamlining computation, comparisons, and decision-making.