Ketaki Chhayendra Mahajan

**Batch:** P1-2

**Roll Number:** 16010622050

**Experiment / assignment / tutorial No. 2**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE:**  Write a program to accept 3 numbers from the user and find the largest of the 3 numbers using -   1. If-else, if-else 2. Ternary operator |

**Aim:**

Write a program to accept 3 numbers from the user and find the largest of the 3 numbers using-

1. If-else, if-else
2. Ternary operator

**Expected OUTCOME of experiment:**

The objective of the program is to output the largest number out of the 3 numbers given by the user. In case if the numbers entered by the user are the same, the program should still be able to output the greatest number instead of displaying any garbage values.

This program is to be solved using two different algorithms. One of the methods is to use to an if-else ladder and the other method is through the use of ternary operators. Although both algorithms are different, both programs should be able to display the greatest number without any errors.

**Books/ Journals/ Websites referred:**

1. Programming in ANSI C, E. Balagurusamy, 7 th Edition, 2016, McGraw-Hill Education, India.
2. Structured Programming Approach, Pradeep Dey and Manas Ghosh, 1 st Edition, 2016, Oxford University Press, India.
3. Let Us C, Yashwant Kanetkar, 15th Edition, 2016, BPB Publications, India.

**Problem definition:**

Ask user to input three numbers. Compare these three numbers to find the largest of them using -

1. Nested if else statement
2. Using ternary operator

**Algorithm:**

1. If-else, if-else -

Start

Declare variables num1, num2 and num3

Read num1, num2 and num3 from user

If number 1 is greater than equal to number 2 AND if number 1 is greater than equal to number 3

Display number 1 is greatest

Else if number 2 is greater than equal to number 1 AND if number 2 is greater than equal to number 3

Display number 2 is greatest

Else

Display number 3 is greatest

End if

Stop

1. Ternary operator -

Start

Declare variables num1, num2, num3 and greatest

Read num1, num2 and num3 from user

Checks the ternary nested ternary conditional operator

Assign the greatest number to the variable greatest

Display greatest

Stop

**Implementation details:**

1. If-else, if-else -

//this program calculates the greatest of 3 numbers given by the user using the if/else ladder.

#include<stdio.h>

int main()

{

printf("\n This program calculates the greatest of 3 numbers using if/else ladder.");

int num1, num2, num3;

printf("\n\n Enter the first number: ");

scanf("%d", &num1); //input number 1

printf(" Enter the second number: ");

scanf("%d", &num2); //input number 2

printf(" Enter the third number: ");

scanf("%d", &num3); //input number 3

if((num1>=num2)&&(num1>=num3)) //Checks if number 1 is greater than number 2 AND if number 1 is greater than number 3. If both conditions are satisfied, number 1 is printed as the greatest number.

printf("\n %d is the greatest number!", num1);

else if ((num2>=num1)&&(num2>=num3))

printf("\n %d is the greatest number!", num2);

else

printf("\n %d is the greatest number!", num3);

printf("\n\n End of program.");

return 0;

}

1. Ternary operator -

//this program calculates the greatest of 3 numbers given by the user using ternary operators.

#include<stdio.h>

int main()

{

printf(" This program finds the greatest number using ternary operators.");

int num1, num2, num3, greatest;

printf("\n\n Enter the first number: ");

scanf("%d", &num1); //input number 1

printf(" Enter the second number: ");

scanf("%d", &num2); //input number 2

printf(" Enter the third number: ");

scanf("%d", &num3); //input number 3

greatest = (num1>num2) ? ((num1>num3) ? num1 : num3) : ((num2>num3) ? num2 : num3);

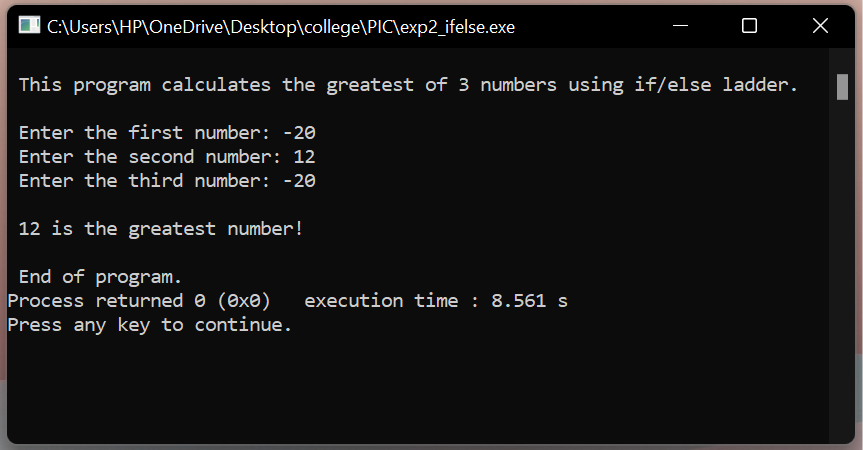
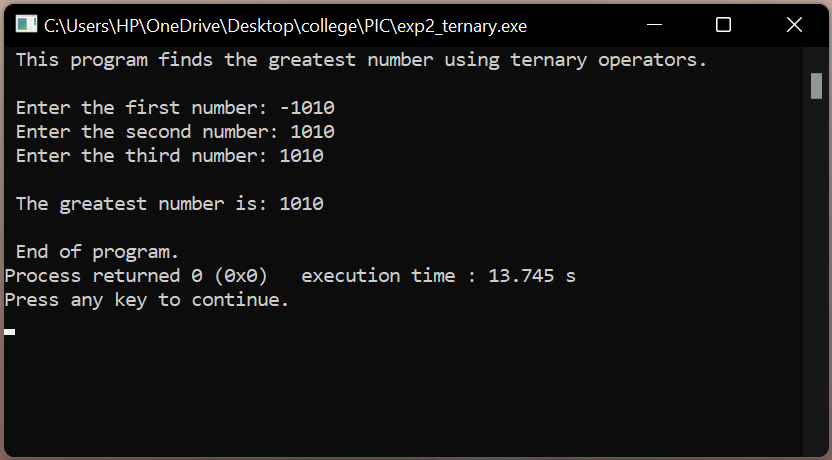
printf("\n The greatest number is: %d", greatest);

printf("\n\n End of program.");

return 0;

}

**Output(s):**

1. If-else, if-else –
2. ****Ternary operator -

**Conclusion:**

To conclude, carrying out experiment 2 has resulted in learning various new concepts such as-

* if-else ladder
* ternary conditional operator and its syntax
* use of relational and logical operators

e.g. ((num2>=num1) && (num2>=num3))

Hence, I was able to successfully carry out experiment 2 and fulfil the objectives that the problem defined as seen in the output.

**Post Lab Descriptive Questions:**

**Explain relational, logical and bitwise operators with examples.**

1. Relational operators –

We often compare need to compare two quantities, and depending on their relation, take certain decisions. For example, we many compare the age of two people, or price of two items, and so on. This comparison can be carried out with the help of relational operators. Relational operators allow us to compare two values either numeric, character string, or logical data and check whether they are equal or unequal to each other or whether they are greater or less than. The value of a relational expression is either one or zero, one being when specified relation is true and zero being when the specified relation is false.

|  |  |  |
| --- | --- | --- |
| **Operators** | **Meaning** | **Examples** |
| < | is less than | checks if the first operand is lesser than the second one. If so, it returns true, else it returns false. e.g. 5<4 will return false and 6<8 will return true |
| < = | is less than or equal to | checks if the first operand is equal to or less than the second one. If so, it returns true, else it returns false. e.g. 9<=9 and 6<=9 will return true and 9<=8 will return false |
| > | is greater than | checks if the first operand is greater than the second one. If so, it returns true, else it returns false. e.g. 5>4 will return true and 7>8 will return false |
| > = | is greater than or equal to | Checks if the first operand is equal to or less than the second one. If so, it returns true, else it returns false. e.g. 5>=5 and 5>=4 will return to be true |
| = = | is equal to | checks if two of the available operands are equal to each other or not. e.g.  4 == 5 gets evaluated to 0  4 == 4 gets evaluated to 1 |
| ! = | is not equal to | checks if two operands are equal to each other or not. e.g.  4 != 5 gets evaluated to 1  4 != 4 gets evaluated to 0 |

1. Logical operators –

C allows the usage of the following three logical operators:

|  |  |
| --- | --- |
| **Operators** | **Meaning** |
| & & | logical AND |
| | | | logical OR |
| ! | logical NOT |

The logical operators && and | | are used when we want to test more than one condition and make a decision. An example of this is:

(a > b) && (x = = 10)

Similar to the relational operator, the logical operators also yield a value of one or zero, as shown in the truth table below. However, the logical expression will be true only if *both* the conditions (a > b) and (x = = 10) are true. If *only one* of the conditions is true, then the expression is false.

|  |  |  |  |
| --- | --- | --- | --- |
| condition 1 (cond1) | condition 2 (cond2) | value of expression | |
| cond 1 && cond 2 | cond 1 | | cond 2 |
| true | true | 1 | 1 |
| true | false | 0 | 1 |
| false | true | 0 | 1 |
| false | false | 0 | 0 |

Some examples of the usage of logical operators are:

* if ((age > 60) && (salary < 20000))

Here, the expression will be true only when both (age>60) and (salary<20000) are true. Any case other than when both the conditions are true will result in a false expression.

* If ((number > 10) | | (number < 100))

Here, the expression will be true when both the conditions are true or *either one* of the conditions is true. Both conditions being false results in value of expression being false.

1. Bitwise operators –

Bitwise operators are used for manipulating data at the bit level, which is also known as bit level programming. These operators operate on one or more bit patterns or binary numbers at their individual bit level. It consists of two digits, either 0 or 1. Bitwise operators are used in numerical computations to make the calculation process faster.

In C language there are 6 bitwise operators including –

|  |  |
| --- | --- |
| **Operators** | **Meaning** |
| & | bitwise AND |
| | | bitwise OR |
| << | left shift |
| >> | right shift |
| ~ | bitwise NOT |
| ^ | bitwise XOR |

& (bitwise AND) - two numbers are taken as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.

e.g. a = 5 (00000101), b = 9 (00001001)

a & b = 00000001 = 1

| (bitwise OR) - two numbers are taken as operands and does OR on every bit of two numbers. The result of OR is 1 if any of the two bits is 1.

e.g. a = 5 (00000101), b = 9 (00001001)

a | b = 00001101 = 13

^ (bitwise XOR) - two numbers are taken as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.

e.g. a = 5 (00000101), b = 9 (00001001)

a ^ b = 00001100 = 12

<< (left shift) - The bits are shifted to the left by the number of positions specified. The bit positions that have been vacated by the shift operation are zero-filled.

e.g. b = 9 (00001001)

b << 2 = 00100100 = 39

>> (right shift) - The bits are shifted to the right by the number of positions specified. The bit positions that have been vacated by the shift operation are zero-filled.

e.g. b = 9 (00001001)

b >> 2 = 00000010 = 2

~ (bitwise NOT) - one number is taken and inverts all of its bits.

e.g. a = 5 (00000101)

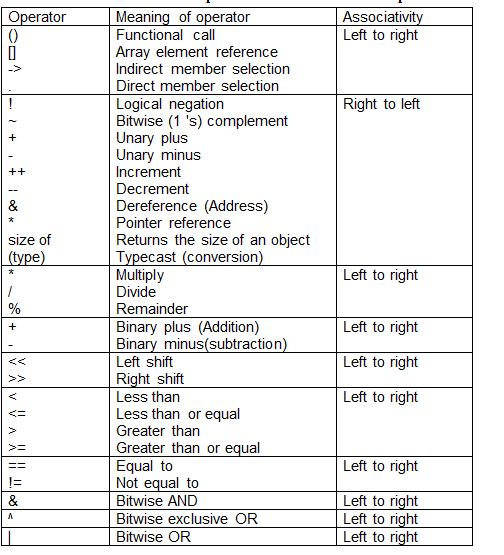
~ a = 11111010 = -6

**Write associative rules and precedence table of various operators.**

Operators precedence and associativity are two characteristics of operators that determine the evaluation order of sub-expressions in absence of brackets. Operator precedence determines which operation is performed first in an expression with more than one operators with different precedence. However, operator’s associativity is used when two operators of same precedence appear in an expression. Associativity can be either left to right or right to left.

Associative rules include -

1. Associativity is only used when there are two or more operators of same precedence.
2. All operators with the same precedence have same associativity
3. Precedence and associativity of postfix ++ and prefix ++ are different
4. Comma has the least precedence among all operators and should be used carefully
5. There is no chaining of comparison operators in C



**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**