**Batch: A2 Roll No.: 16010121045**

**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                    If - else if-else                    Ternary operator |

**AIM:** Write a program to accept 3 numbers from the user and find the largest of the 3 numbers using

                  If - else if-else

                  Ternary operator

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

**CO2**: Applying basic concepts of C programming for problem solving

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Programming in ANSI C, E. Balagurusamy, 7 th Edition, 2016, McGraw-Hill

Education, India.

1. Structured Programming Approach, Pradeep Dey and Manas Ghosh, 1 st Edition,

2016, Oxford University Press, India.

1. Let Us C, Yashwant Kanetkar, 15th Edition, 2016, BPB Publications, India.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Problem Definition:**

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

1. Nested if else statement
2. Using ternary operator

**Algorithm:**

1. Start

2. Declare a , b , c .

3. Ask the user to enter three integer values.

4. Read the three integer values in a, b and c (integer variables).

5. Declare max.

6. Check if a is greater than b.

7. **If true**, then check if a is greater than c.

1. **If true**, then store a as max.

2. **If false**, then store b as max.

8. **If false**, then check if b is greater than c.

1. **If true**, then store b as max.

2. **If false**, then store c as max.

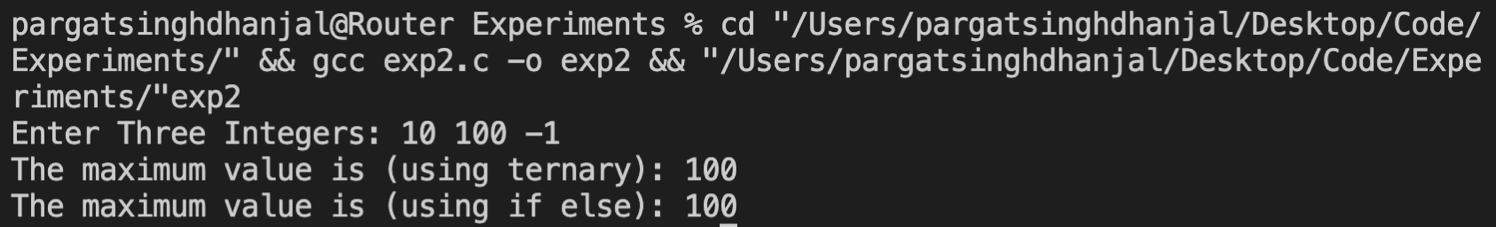
9. Print max.

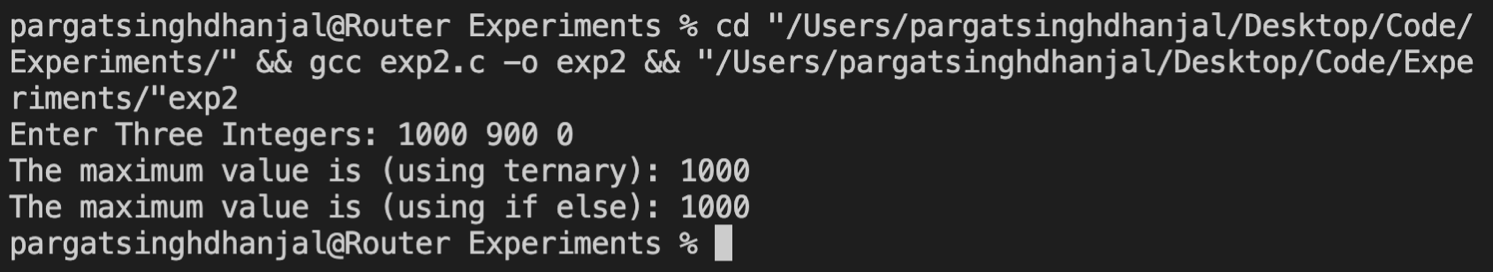
10. End.

**Implementation details:**



**Output(s):**





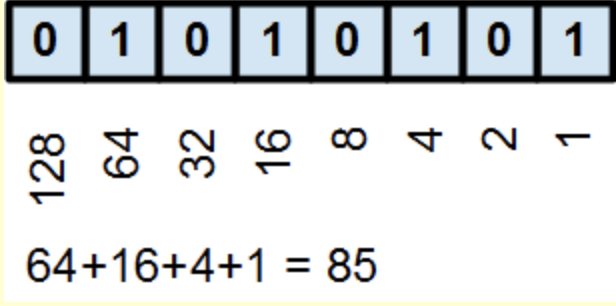
**Conclusion:**

Successfully executed Exp 2.

Learnt to use conditional statements i.e. if-else and ternary operators. Also learnt to write algorithms for programs.

**Post Lab Descriptive Questions**

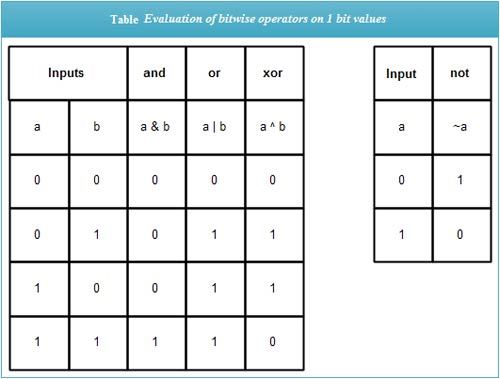
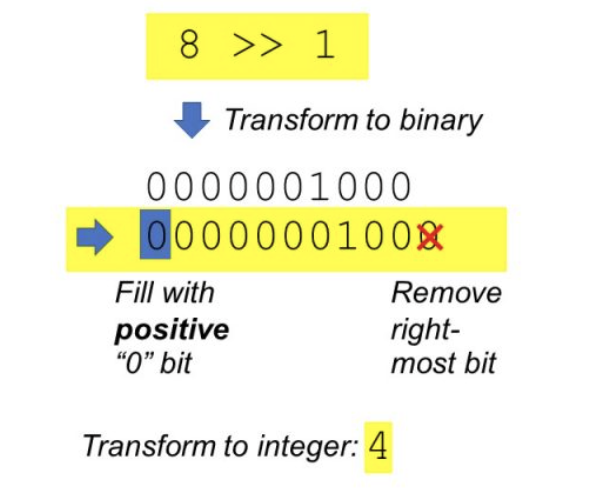
1. **Explain bitwise operators with examples.**

To understand bitwise we need to first know what is a *Bit*. A *bit* is most basic unit of information in computing. It is short for binary digit, which means that it can only have one of two values, **0** or **1**.

So we can represent the number **85** as **01010101** in binary number system.

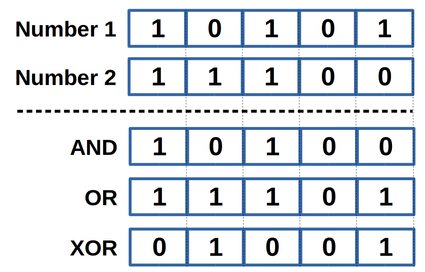
**Bitwise Operators**

* **& (bitwise AND)** in C takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.
* **| (bitwise OR)** in C takes two numbers 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.
* **^ (bitwise XOR)** in C takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.
* **<< (left shift)** in C takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.
* **>> (right shift)** in C takes two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.
* **~ (bitwise NOT)** in C takes one number and inverts all bits of it.



The above following operators apply the operation to each and every bit of a digit.

So as of a result we get something like this.

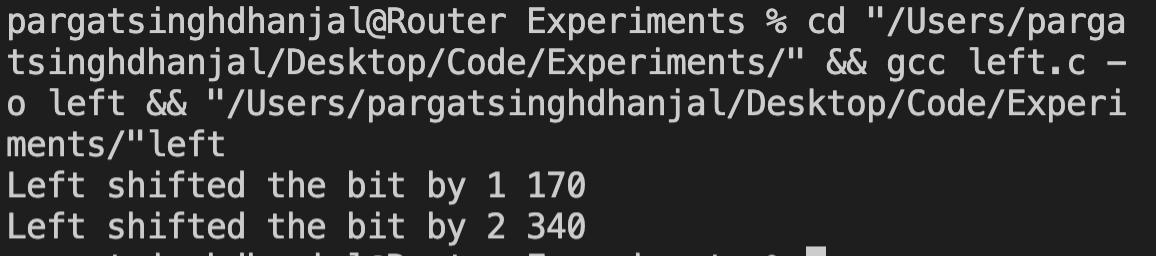


1. **Write a code snippet to perform left shifting of bits by some positions**

Code:



Output:



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

The precedence of operators determines which operator is executed first if there is more than one operator in an expression. Let us consider an example:



In C, the precedence of **/** is higher than **–** and **=.** Hence, **100/2** is evaluated first. Then the expression involving **-** is evaluated as the precedence of **-** is higher than that of **=** .

So **ans= 100 – (100/2). , i.e. 100 – 50 = 50**

Here's a table of operators precedence from higher to lower.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning of operator | Associativity |
| () [] -> . | Functional call  Array element reference Indirect member selection Direct member selection | Left to right |
| ! ~ + - ++ -- & \* sizeof (type) | Logical negation Bitwise(1 's) complement Unary plus Unary minus Increment Decrement Dereference (Address) Pointer reference Returns the size of an object Typecast (conversion) | Right to left |
| \* / % | Multiply Divide Remainder | Left to right |
| + - | Binary plus(Addition) Binary minus(subtraction) | Left to right |
| << >> | Left shift Right shift | Left to right |
| < <= > >= | Less than Less than or equal Greater than Greater than or equal | Left to right |
| == != | Equal to Not equal to | Left to right |
| & | Bitwise AND | Left to right |
| ^ | Bitwise exclusive OR | Left to right |
| | | Bitwise OR | Left to right |
| && | Logical AND | Left to right |
| || | Logical OR | Left to right |
| ?: | Conditional Operator | Right to left |
| = \*= /= %= += -= &= ^= |= <<= >>= | Simple assignment Assign product Assign quotient Assign remainder Assign sum Assign difference Assign bitwise AND Assign bitwise XOR Assign bitwise OR Assign left shift Assign right shift | Right to left |
| , | Separator of expressions | Left to right |

1. **What are different storage class specifiers in C?**

There are 4 Storage class specifiers in C they are:

**Auto:** This is the ***default storage class*** for all the variables declared inside a function or a block. Hence, the keyword auto is rarely used while writing programs in C language. Auto variables can be only accessed within the block/function they have been declared and not outside them. Of course, these can be accessed within nested blocks within the parent block/function in which the auto variable was declared. However, they can be accessed outside their scope as well using the concept of pointers. They are assigned a garbage value by default whenever they are declared.

**Extern:** Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used. Basically, an extern variable is nothing but a ***global variable*** initialized with a legal value where it is declared in order to be used elsewhere. It can be accessed within any function/block. Also, a normal global variable can be made extern as well by placing the *‘extern’* keyword before its declaration/definition in any function/block. This basically signifies that we are not initializing a new variable but instead we are using/accessing the global variable only. *The main purpose of using extern variables is that they can be accessed between two different files which are part of a large program.*

**Static:** This storage class is used to declare static variables which are popularly used while writing programs in C. Static variables have a property of ***preserving their value*** even after they are *out of their scope!* Hence, static variables preserve the value of their last use in their scope. So we can say that they are initialized only once and exist till the termination of the program. Thus, no new memory is allocated because they are not re-declared. Their scope is local to the function to which they were defined. Global static variables can be accessed anywhere in the program. By default, they are assigned the value 0 by the compiler.

**Register:** This storage class declares register variables which have the same functionality as that of the auto variables. The only difference is that the ***compiler tries to store these variables in the register of the microprocessor*** if a free register is available. This makes the use of register variables to be much faster than that of the variables stored in the memory during the runtime of the program. If a free register is not available, these are then stored in the memory only. Usually few variables which are to be accessed very frequently in a program are declared with the register keyword which improves the running time of the program.

**Date: 20/11/21 Signature of faculty in-charge**