**Batch: D2 Roll No.: 16010221038**

**Name : Eshan Kalp Trivedi**

**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:**

The expected outcome for experiment of if else operator is mentioned in the code below,

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

**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 three numbers to find the largest of them using

1. Nested if else statement
2. Using ternary operator

**Algorithm:**

**1. Nested if else statement**

1. Start

2. Read three numbers as n1, n2, n3 (integer variables).

3. Check if n1 is greater than n2.

4. If true, then check if n1 is greater than n3.

5. If true, then print ‘n1’ as the greatest number.

6. If false, then print ‘n3’ as the greatest number.

7. If false, then check if n2 is greater than n3.

8. If true, then print ‘n2’ as the greatest number.

9. If false, then print ‘n3’ as the greatest number.

10. END

1. **Using ternary operator**

1. Start

2. Read three numbers as n1, n2, n3, max (integer variables)

3. Enter three numbers

4. In ‘max’ check if n1 is greater than n2

5. if true print “biggest number is n1”

6. If false, then print “biggest number is n3”.

7. If false, then check if n2 is greater than n3.

8. If true, then print biggest number is n2

9. If false, then print ‘n3’ as the greatest number.

10. Save these all-in max

11. END

**Flowchart:** (for both)

Read n1, n2, n3

A,B,C

IF n1>n2

**False True**

If n2>n3

If n1>n3

**True False False True**

Print n3 is the greatest number

Print n1 is the greatest number

Print n2 is the greatest number

**Implementation details:**

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

**Nested if else statement**

**Code:**

int main()

{

float n1, n2, n3;

printf("Enter three numbers: ");

scanf("%f %f %f",&n1,&n2,&n3);

if(n1 >= n2)

{

if(n1 >= n3)

printf("Largest Number=%.2f\n",n1);

else

printf("Largest Number=%.2f\n",n3);

}

else

{

if(n2>=n3)

printf("Largest Number=%.2f\n",n2);

else

printf("Largest Number=%.2f\n",n3);

}

return 0;

}

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

**Using ternary operator**

**Code:**

#include <stdio.h>

void main()

{

int n1, n2, n3, max ;

printf(" Enter the three numbers : ") ;

scanf("%d %d %d", &n1, &n2, &n3) ;

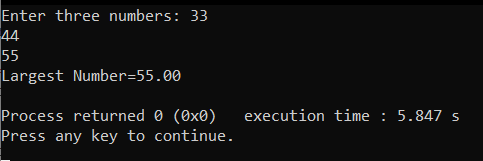
max = n1 > n2 ? (n1 > n3 ? n1 : n3) : (n2 > n3 ? n2 : n3) ;

printf("\n Biggest number is : %d", max) ;

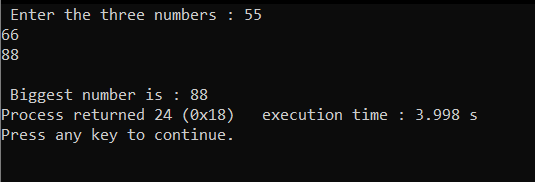
}

**Output(s):**

* + - 1. **Nested if else statement**



* + - 1. **Using ternary operator**

****

**Conclusion:**

From the following experiment related to input three numbers and compare three numbers to find the largest of them using two methods one as ‘Nested if else statement’ and other as ‘Using ternary operator’.

From the experiment we learn how we can compare and find out largest among the three numbers using two different ways and also by using ‘switch case’.

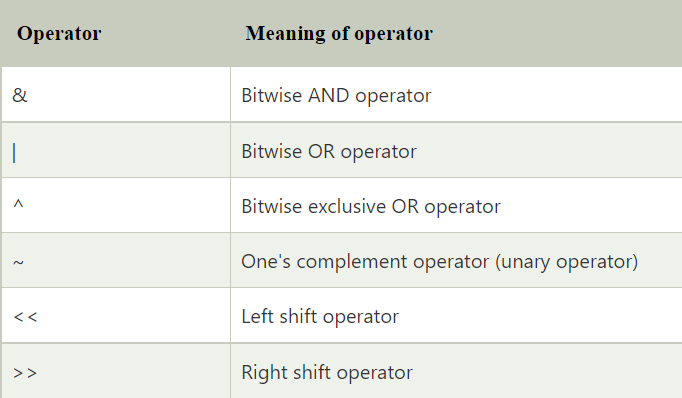
**Post Lab Descriptive Questions**

1. **Explain bitwise operators with examples**

In the C programming language, operations can be performed on a bit level using bitwise operators.

Bitwise operations are contrasted by byte-level operations which characterize the bitwise operators' logical counterparts, the AND, OR, NOT operators. Instead of performing on individual bits, byte-level operators perform on strings of eight bits (known as bytes) at a time. The reason for this is that a byte is normally the smallest unit of addressable memory (i.e. data with a unique memory address).

This applies to bitwise operators as well, which means that even though they operate on only one bit at a time they cannot accept anything smaller than a byte as their input.



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

**Code:**

#include<stdio.h>

int main()

{

char a = 5, b = 9;

printf("a<<1 = %d\n", a<<1);

printf("b<<1 = %d\n", b<<1);

return 0;

}

Here, ‘char’, stands for character.

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Precedence** | **Operator** | **Description** | **Associativity** |
| **1** | ++ -- | Suffix/postfix increment and decrement | Left-to-right |
| () | Function call |
| [] | Array subscripting |
| . | Structure and union member access |
| -> | Structure and union member access through pointer |
| (*type*){*list*} | Compound literal |
| **2** | ++ -- | Prefix increment and decrement | Right-to-left |
| + - | Unary plus and minus |
| ! ~ | Logical NOT and bitwise NOT |
| (*type*) | Cast |
| \* | Indirection (dereference) |
| & | Address-of |
| sizeof | Size-of |
| \_Alignof | Alignment requirement(C11) |
| **3** | \* / % | Multiplication, division, and remainder | Left-to-right |
| **4** | + - | Addition and subtraction |
| **5** | << >> | Bitwise left shift and right shift |
| **6** | < <= | For relational operators < and ≤ respectively |
| > >= | For relational operators > and ≥ respectively |
| **7** | == != | For relational = and ≠ respectively |
| **8** | & | Bitwise AND |
| **9** | ^ | Bitwise XOR (exclusive or) |
| **10** | | | Bitwise OR (inclusive or) |
| **11** | && | Logical AND |
| **12** | || | Logical OR |
| **13** | ?: | Ternary conditional | Right-to-left |
| **14** | = | Simple assignment |
| += -= | Assignment by sum and difference |
| \*= /= %= | Assignment by product, quotient, and remainder |
| <<= >>= | Assignment by bitwise left shift and right shift |
| &= ^= |= | Assignment by bitwise AND, XOR, and OR |
| **15** | , | Comma | Left-to-right |

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

Storage Classes are used to describe the features of a variable/function. These features basically include the scope, visibility and life-time which help us to trace the existence of a particular variable during the runtime of a program.

C language uses 4 storage classes, namely:

* 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.
* Extern: Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used. Basically, the value is assigned to it in a different block and this can be overwritten/changed in a different block as well. So 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
* Static: This storage class is used to declare static variables which are popularly used while writing programs in C language. 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. An important and interesting point to be noted here is that we cannot obtain the address of a register variable using pointers.

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