ASSIGNMENT 3

1. The precedence of arithmetic operators is (from highest to lowest)
   a) %, *, /, +, –
   b) %, +, /, *, –
   c) +, -, %, *, /
   d) %, +, -, *, /

Solution: (a) The precedence order follows the first option (a)

2. What is the output of the following program? (% indicates modulo operation, which results the reminder of a division operation)

```
#include <stdio.h>
int main()
 {
        float i = -3.0;
        int k = i % 2;
        printf("%d", k);
        return 0;
}
```

   a) -1
   b) 1
   c) 0
   d) Compilation error

Solution: (d) 'int to binary ' operator '%' cannot be operated on floating variable. Thus i%2 is not a valid operation in C. The compiler will show error at this step.

3. Find the output of the following C code.

```
#include<stdio.h>
int main()
{
        int a=10, b=3, c=2, d=4, result;
        result=a+a*-b/c%d+c*d;
        printf("%d", result);
        return 0;
}
```

   a) -42
   b) 24
   c) 15
   d) -34

Solution: (c) Following the precedence rule, we can conclude that the operation steps are

- ➔ Result=10+10*- 3/2%4+2*4
- ➔ Result=10-30/2%4+2*4
- ➔ Result=10-15%4+2*4
- ➔ Result=10-3+2*4
- ➔ Result=10-3+8
- ➔ Result=7+8
- ➔ Result=15

4. What is the output of the following C code?

```
#include <stdio.h>
int main()
{
int h = 8;
int b = 4 * 6 + 3 * 4 < h*5 ?4 : 3;
printf("%d\n", b);
return 0;

}
```

a) 0
b) 3
c) 4
d) Compilation error

Solution: (c) '?:' is Conditional Expression. If Condition is true ? then value X : otherwise value Y. After simplifying the expression, we get 36<40?4:3. The condition in LHS of ? is true. Thus 4 will be stored in b.

5. What will be the output?

```
#include <stdio.h>
int main ()
{
   int a = 1, b = 2, c = 3;
   if (c > b > a)
           printf("TRUE");
   else
           printf("FALSE");
   return 0;
}
```

a) TRUE
b) FALSE
c) Syntax Error
d) Compilation Error

Solution: (b) FALSE
(c > b > a) is treated as ((c > b) > a), associativity of '>' is left to right. Therefore, the value becomes ((3 > 2) > 1) which becomes (1 > 1), thus FALSE

6. Find the output of the following C code

```
#include <stdio.h>
int main()
{
int x=1;
   if ((3>5) || (2!=3))
printf("IITKGP\n");
   else if (x&=0)
printf("IITD\n");
   else
printf("IITM\n");
return 0;
}
```

a) IITKGP
b) IITD and IITM
c) IITKGP and IITM
d) IITM

Solution: (a) Only the first if condition will be executed as the condition inside the if statement is true. Thus IITKGP will be printed.

7. What will be the output?

```
#include <stdio.h>
int main()
{
int x=2;
   if(x=1)
printf("TRUE");
   else
printf("FALSE");
```

```
    return 0;
}
```

a) TRUE
b) FALSE
c) Compilation Error
d) Compiler Dependent

Solution: (a) TRUE
if(x=1)... "=" is an assignment operator, so 1 will be assigned to x and condition will be true due to if(1).

8. Which of the following statement is correct?
    a) Operator precedence determines which operator is performed first in an expression with more than one operator with different precedence. Associativity is used when two operators of same precedence appear in an expression
    b) Operator associativity determines which operator is performed first in an expression with more than one operator with different associativity. Precedence is used when two operators of same precedence appear in an expression
    c) Operator precedence and associativity are same.
    d) None of the above

Solution: (a) Operator precedence determines which operator is performed first in an expression with more than one operator with different precedence, whereas associativity is used when two operators of same precedence appear in an expression

9. Which of the following method are accepted for assignment?
    a) 8=x=y=z
    b) x=8=y=z
    c) x=y=z=8
    d) None

Solution: (c)

10. What will be the output?

```
#include<stdio.h>
int main()
{
int x;
 x= 9<5+3 && 7;
printf("%d", x);
return 0;
}
```

a. 0
b. 1
c. 7
d. Compilation error

Solution: (a) 0

This expression is equivalent to:
((9< (5 + 3)) && 7)
i.e., (5 + 3) executes first resulting into 8
then, first part of the expression (9< 8) executes resulting into 0 (FALSE)
Then, (0 && 7) executes resulting into 0 (FALSE)