Quiz 1

> Section A

Question 1 - 5 marks

Problem Description

Write a program to find the factorial of a number. Here `n` can be taken to be an input from the user.

Solution

```c
#include <stdio.h>

int main(){
  int n;
  scanf("%d",&n);
  if(n<0){
    printf("Invalid Input!");
  }else{
    int ans=1;

    for(int i=1;i<=n;i++){
      ans*=i;
    }

    printf("%d\n",ans);
  }

  return 0;
}
```

Grading Scheme

1 mark: For writing

```
#include <stdio.h>
```

1 mark: For writing

```
int main(){
    return 0
}
```

3 marks: For correct logic, this includes correct initialization of `ans=1` , writing the correct for-loop and correct formula. Marks have been cut for compilation errors(Between 0.5 to 1 marks have been cut)

> Note: For approaches other than the standard for-loop approach, marks have been awarded according to correctness of the approach.
> For eg, in case of a recursion based approach, marks have been awarded if the base-case is correct and the recursion works properly.

Question 2 - 5 marks

Problem Description

What is type-casting? Explain the types of type casting in C.

Solution

The idea of type-casting is to convert one data type into another.

There are two types of type conversions/castings:

1) Implicit conversions: The idea here is to convert the data type of one variable without altering the actual value of the variable. The datatype can be both promoted and demoted by compiler. There can be various implicit conversions:

a) Arithmetic conversions: Happens when operands in arithmetic or logical expressions don't have same type. Example: Addition of integer and float.

```
int a = 10;
```

```
float b = 20.0;

float c = a + b;
// type of (a+b) is float, so it is stored back in `float c`
```

b) Conversion during assignment: Happens when type of expression on the right side of an assignment operator doesn't match the type of the variable on the left side. Example: assigning an int to char.

```
char pi1 = 3;
int  pi2 = 'a';
int  pi3 = 3.14; // Results in loss of information
float pi4 = 3;  // 3 is an int, but assigned to float
```

c) Others: In function argument; in return statement.

2. Explicit type-casting: In this case, the datatype conversion is user-defined according to the program's needs. It can be a lower or higher end data type. Example:

```
float pi  = (float)21/7;
```

Grading Scheme

Definition of typecasting **(1 mark)**.

Definition of implicit conversion **(1 mark)** and its example **(1 mark)**. (Sub-types of implicit conversion are not required.) Definition of explicit typecasting **(1 mark)** and its example **(1 mark)**.
Writing extra details relevant to the topic can also result in extra marks. For example: Writing hierarchical order of promotion in arithmetic conversion. Marks have been deducted for conceptual or syntax errors.

Question 3 - 5 marks

Problem Description

Write `-35` in 2's complement form (assume `8-bit` representation).

Solution & Grading Scheme

The number 35 in binary is given by: `00100011` **(2 marks)**
In 2 complement it is given by: `11011101` . **(3 marks)**

Question 4 - 5 marks

Problem Description

State True/False with justification: Given two positive integers `x` and `y` , it is not possible to calculate the minimum and maximum of the two numbers without using any conditions or loops.

Solution

False **(1 mark)**.

```
#include<stdio.h>

int main()
{
int x,y, min, max;
min = y^((x^y)&-(x<y));
max = x^((x^y)&-(x<y));
printf("min = %d,max = %d\n", min,max);
return  0;
}
```

(4 marks).

Grading Scheme

- **1 mark** for writing False or even if have have wrote True but your Explanation points out to be correct marks have been awarded giving benefit of reading the question wrong . Such cases have been marked Lucky and are instructed to not do it from next time.
- **1/2 mark** if there's a contradictory statements for the explanation and answer
- **3/2** for writing the formula for minimum number expression here the formula using Absolute value concept and other correct equivalent bitwise expressions have also been taken into account.

- Again, **3/2** for writing the correct way to find out the maximum value .

> Note:- Using the terms like maximum can be found out logically, naturally etc. from minimum have been given 0 marks for the same

- **1 mark** for writing the above expressions with the whole of code structure.

Question 5 - 10 marks

Problem Description

Write a program to print the following pattern of the Pascal's triangle.

```
    1

   1 1

  1 2 1

 1 3 3 1
```

Solution

```c
#include<stdio.h>

int main() {
    int rows=4, num= 1, space, i, j;
    for(i=0; i < rows; i++) {
        for(space=1; space <= rows-i; space++)
            printf(" ");
            for(j=0; j <= i; j++) {
            if (j==0 || i==0)
                num= 1;
            else
                num = num*(i-j+1)/j;
            printf(" %d", num);
        }
        printf("\n\n");
    }
```

```
    return 0;
}
```

Grading Scheme

- Header files: (**1 mark** - awarded if the syntax is strictly correct)
- Initialising variables using the right syntax: (**1 mark** - awarded if the syntax is strictly correct)
- Using the variables that were declared correctly: (**1 mark** - awarded with *some* leniency depending on the mistake)
- Using `printf` correctly for printing spaces:
  - **2 marks are** awarded if spaces are printed properly with correct ***generalised*** logic.
  - **1 mark is** awarded if just the syntax is correct.
- Using `for` or `while` loop:
  - **2 marks are** awarded if numbers are printed properly with correct ***generalised*** logic.
  - **1 mark is** awarded if just the syntax is correct
- Correct logic for ***generalised*** output - **2 marks**
- Correct final output (brute or general) - **1 mark**

Question 6 - 20 marks

Problem Description

What is output of the following programs with proper justification ?

i) 5 marks

```c
#include<stdio.h>
int main()
{
    int x = 5, y, z;
    y = x++;
    z = x--;
    printf("%d %d %d\n", x, y, z);
    return 0;
}
```

Solution & Grading Scheme

Output: `5 5 6`

Justification:

```
y = x++; // y = 5, x = 6
z = x--; // z = 6, x = 5
```

`x = 5` (post-increment operator) **(2 marks)**
`y = 5` (post-increment operator) **(2 marks)**
`z = 6` (post-decrement operator) **(1 mark)**

ii) 5 marks

```c
#include<stdio.h>
int main()
{
    int a = 3;
    a = (a++) + ~ (++a);
    printf("%d", a);
    return 0;
}
```

Solution & Grading Scheme

Output: `-3` (**2 marks**)

Justification:

```
a = 3;
```

```
a = (a++) + ~ (++a)
```

```
a = 3 + ~(5)
```

[**1 marks** for post-increment, **1 marks** for pre-increment]

`a = 3 + (-6)` [1 marks for writing `~(5) = -6`]

`a = -3`

iii) 5 marks

```c
#include<stdio.h>
int main()
{
    int i = 21 > 5 > 3 < 4;
    printf("%d", i);
    return 0;
}
```

Solution & Grading Scheme

Output: 1 (**2 marks**, only **1 mark** if justification is incorrect)

Justification: The expression gets evaluated from left to right

```c
int i = (((21 > 5) > 3) > 4)
```

`21 > 5` is true, so it gets evaluated to 1. **(1 mark)**
`1 > 3` is false, so it gets evaluated to 0. **(1 mark)**
`0 < 4` is true, so it gets evaluated to 1. **(1 mark)**

iv) 5 marks

```c
#include<stdio.h>
int main()
{
    int y;
    printf("%d", scanf("%d", &y));
    /* Suppose that input value given for above scanf is 2023*/
    return 0;
}
```

Solution & Grading Scheme

`Output: 1`

Justification: Scanf returns the number of objects that are input, in this case the number of object is 1 (an integer `y`) so **Scanf will return 1** and it will be printed.

2 marks for output and **3 marks** for justification