Tasks

1. **Typecasting**
   - **a.** Copy **i.** a **`short`** variable to **`long`** variable, **ii.** a **`float`** variable to **`double`** variable
   - **b.** Copy **i.** a **`long`** variable to **`short`** variable. **ii.** a **`double`** variable to **`float`** variable
   - **c.** Perform division of following; **i.** 3/5 **ii.** 3.0/5

2. **Arrays**

   Make an array **`double real_line[7]`** .
   - **a.** Initialize the whole array with zero.
   - **b.** Initialize first 2 elements with 3.
   - **c.** Initialize every element with its index position without using loop.

3. **Loops (for)**
   - **a.** Use **`for`** loop to do task-2c.
   - **b.** Calculate the sum of all the members of **`real_line`** array.
   - **c.** For array declared in task 2, set elements at even position to 2 and elements at odd position to 3.

4. **Loops (while)**
   - **a.** Declare variable **`double fnum = 0.0`** and **`double fval = 0.179`**;
   - **b.** Use **`while`** loop to calculate how many loop iterations (at minimum) it would take to reach the value of **`100.537`**.

5. **Array of Structures**
   - **a.** Make structure to define complex data type, having members **`re`** and **`im`** of **`double`** type. Use **`typedef`** to name it **`complex`**.
   - **b.** A line **`line1`** is defined by 5 points having co-ordinates (0,0), (1,2), (2,3), (3,4) and (4,5). Use array of complex structure to store these points.
   - **c.** Another line **`line2`** is defined by 5 points having co-ordinates (0,0), (1,-2), (2,-3), (3,-4) and (4,-5). Use array of complex structure to store these points.
   - **d.** Define following complex functions;
     - **i.** `complex cadd( complex c1, complex c2);`
     - **ii.** `complex csub( complex c1, complex c2);`
     - **iii.** `complex cmul( complex c1, complex c2);`
     - **iv.** `complex cdiv( complex c1, complex c2);`
     - **v.** `double creal(complex c);`
     - **vi.** `double cimag(complex c);`
     - **vii.** `complex comp( double re, double im);`
     - **viii.** `double cmag(complex c);`
     - **ix.** `double cangle(complex c);`
     - **x.** `comp cadd_real( complex c, double re);`
     - **xi.** `comp cmul_real( complex c, double re);`
     - **xii.** `comp cdiv_real( complex c, double re);`

e. Perform each of the above operation on the given 2 lines. e.g.

i. `cnum = cmul( line1[3], line2[3] );`
ii. `real_part = creal( line2[4]);`
iii. `cnum = cdiv_real( line[0], 3.5);`

6. Logical, Shift and Bitwise Operations

a. Write a function **`byte_unpacker()`** which unpacks a byte to an **`unsigned char`** array of 8 elements, so that every element contains one bit in its least significant bit position. Use this function to unpack a byte having value equal to your last 2 digits of your registration number. e.g. if you have reg. id = 1234 then you are going to have 34 in your byte. Function prototype would look as follows;

```
void byte_unpacker(unsigned char packed_byte, unsigned char unpacked_bytes[])
```

Here **`unsigned char packed_byte`** is your input argument, while **`unsigned char unpacked_bytes[]`** is your output argument.

b. Write a function **`byte_packer().`** It would pack the unpacked array of task-a to a byte. Function prototype would look as follows;

```
void byte_packer(unsigned char unpacked_bytes[],unsigned char packed_byte)
```

Here **`unsigned char unpacked_bytes[]`** is your input argument, while **`unsigned char packed_byte`** is your output argument.

7. Number from ASCII representation to Binary representation

Conversion of a number represented in ASCII to an integer number. Write a function **`my_atoi()`** whose prototype would look as follows;

```
int my_atoi(const char * const num_str);
```

a. It should be able to differentiate between positive and negative numbers.
b. It should be able to convert a string og maximum 12 decimal digits to an integer.
c. Use properties of ASCII table to do this job.
d. Typical call would look like as follows;

```
num = my_atoi("123456789"); // would give num = 123456789
num = my_atoi("-456789");
num = my_atoi("+123");
```

8. Hamming Weight

It is the number of ones (bits having value of one) in a given number. So the hamming weight of 37 is 3, while that of 4659 is 6. Write a function **`ham_weight()`** which can have the prototype as follows;

```
int ham_weight( unsigned long num);
```

9. Parity Calculator

Use the function **`ham_weight()`** of previous task to calculate the parity of the given number. It would return 1 if parity is odd, otherwise 0. Function can have the prototype as follows;

```
int parity_calc( unsigned long num);
```