

CC-112L

Programming Fundamentals

Laboratory 12

File Handling

Version: 1.0.0

Release Date: 14-10-2022

Department of Information Technology

University of the Punjab

Lahore, Pakistan

Contents:

- Learning Objectives
- Required Resources
- General Instructions
- Background and Overview
 - Bit Fields
 - Enumeration Constants
 - Sequential Access
 - Random Access
 - File Handling in C
- Activities
 - Pre-Lab Activity
 - Bitwise Operators
 - Bit Fields
 - Defining Bit Fields
 - Unnamed Bit Fields
 - Enumeration Constants
 - Task 01: Bitwise Operations
 - Task 02: Enums
 - In-Lab Activity
 - Creating a Sequential-Access File
 - Pointer to File
 - Open a File
 - End-of-file Indicator
 - Close a File
 - File Open Modes
 - Read from a Sequential-Access File
 - Random-Access File
 - Creating a Random-Access File
 - Writing Data Randomly
 - fseek() function
 - Reading Data Randomly
 - Task 01: Maximum Digit
 - Task 02: Second Largest Word
 - Task 03: Word Occurrence
 - Task 04: Batsman Average
 - Post-Lab Activity
 - Task 01: Vigenère Cipher using File Handling
- Submissions
- Evaluations Metric
- References and Additional Material
- Lab Time and Activity Simulation Log

Learning Objectives:

- Bitwise Operators
- Bit Fields
- Enumeration Constants
- File Handling in C

Resources Required:

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

General Instructions:

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, even not allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

Teachers:		
Course Instructor	Prof. Dr. Syed Waqar ul Qounain	swjaffry@pucit.edu.pk
Teacher Assistants	Usman Ali	bitf19m007@pucit.edu.pk
	Saad Rahman	bsef19m021@pucit.edu.pk

Background and Overview:

Bit Fields:

In C, we can specify the size (in bits) of the structure and union members. The idea of bit-field is to use memory efficiently when we know that the value of a field or group of fields will never exceed a limit or is within a small range. Bit fields are used when the storage of our program is limited.

Enumeration Constants:

In C programming, an enumeration type (also called enum) is a data type that consists of integral constants. To define enums, the “**enum**” keyword is used.

Sequential Access:

It is the simplest access method. Information in the file is processed in order, one record after the other. This mode of access is by far the most common.

Random Access:

A random-access file behaves like a large array of bytes stored in the file system. There is a kind of cursor, or index into the implied array, called the file pointer; input operations read bytes starting at the file pointer and advance the file pointer past the bytes read.

File Handling in C:

File handling refers to the method of storing data in the C program in the form of an output or input that might have been generated while running a C program in a data file, i.e., a binary file or a text file.

Activities:

Pre-Lab Activities:

Bitwise Operators:

The bitwise operators are used to manipulate the bits of integral operands. Bitwise data manipulations are machine dependent. The following table summarizes the bitwise operators.

Operator	Description
"&" bitwise AND	Compare its two operands bit by bit. The bits in the result are set to 1 if the corresponding bits in the two operands are both 1.
" " bitwise OR	Compare its two operands bit by bit. The bits in the result are set to 1 if at least one of the corresponding bits in the two operands is 1.
"^" bitwise XOR	Compare its two operands bit by bit. The bits in the result are set to 1 if the corresponding bits in the two operands are different.
"<<" left shift	Shifts the bits of the first operand left by the number of bits specified by the second operand; fill from the right with 0 bits.
">>" right shift	Shifts the bits of the first operand right by the number of bits specified by the second operand; the method of filling from the left is machine-dependent when the left operand is negative.
"~" complement	All 0 bits are set to 1, and all 1 bit are set to 0. This is often called toggling the bits.

Suppose two numbers 12 and 25. Their bitwise AND operation will be as follows

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bit Operation of 12 and 25

00001100

& 00011001

00001000 = 8 (In decimal)

Now using above numbers in C compiler to perform bitwise AND operation.

```

1  #include <stdio.h>
2  int main() {
3      int a = 12, b = 25;
4      printf("Output = %d", a & b);
5      return 0;
6  }
```

Fig 01. (Bitwise AND Operator)

The output of the above program is following:



Fig 02. (Bitwise AND Operator)

Bit Fields:

You can specify the number of bits in which to store an unsigned or signed integral member of a struct or union. Known as bit fields, these enable better memory utilization by storing data in the minimum number of bits required. Bit field members typically are declared as int or unsigned int.

Defining Bit Fields:

Bit fields are variables that are defined using a pre-defined width or size.

```
struct{
    data_type member_name: width of bit field;
}
```

Element	Description
data_type	It is an integer type that determines the bit-field value which is to be interpreted.
member_name	The member name is the name of the bit field.
width	The number of bits in the bit-field.

Following code is an example of a Bit Field.

```
1  #include <stdio.h>
2  struct date {
3      // 5 bits width for 31 days
4      int day : 5;
5      // 4 bits width for 12 months
6      int month : 4;
7
8      int year;
9  };
10 int main() {
11     struct date dt = { 10,9,2022 };
12     printf("Date is %d/%d/%d", dt.day,dt.month,dt.year);
13     return 0;
14 }
```

Fig 03. (Bit Field)

The output of the above program is as follows:



Fig 04. (Bit Field)

The output of month comes out to be negative. What happened behind is that the value of 9 was stored in 4 bits as 1001. The MSB is a 1, so it's a negative number and you need to calculate the 2's complement of the binary number. By calculating 2's complement you will arrive at value 0111 which equivalent to decimal number 7 and since it's a negative number you will get -7.

Unnamed Bit Fields:

An unnamed bit field is used as padding in a struct. For example, the definition

```
struct example {
    unsigned int a : 13;
    unsigned int : 19;
    unsigned int b : 4;
};
```

uses an unnamed 19-bit field as padding. Nothing can be stored in those 19 bits. Member b (assuming a four-byte-word computer) is stored in a separate word of memory.

Enumeration Constants:

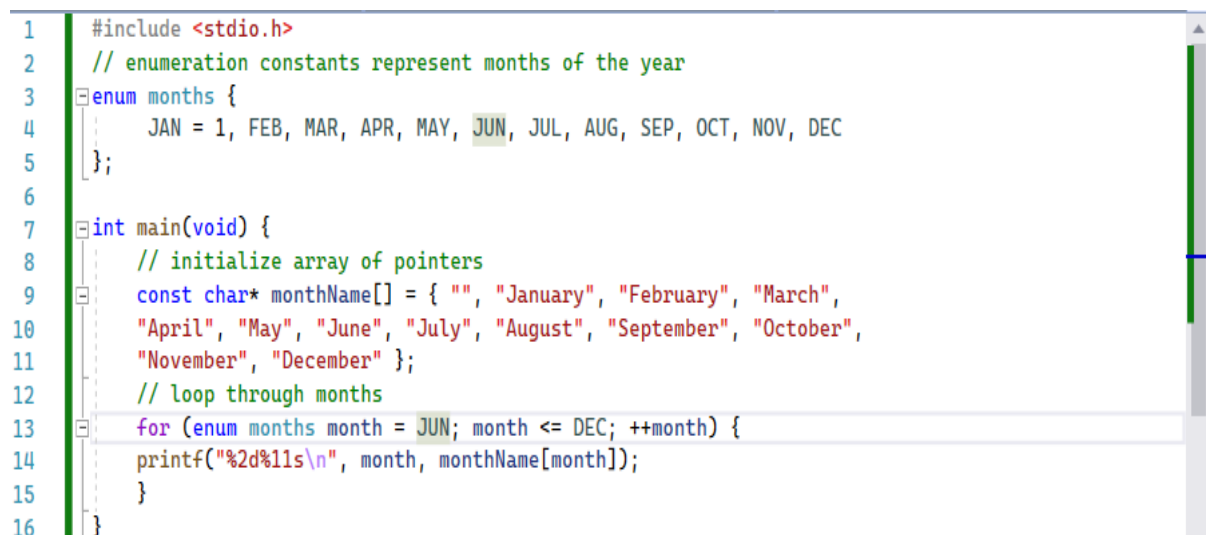
“**enum**” keyword for defining a set of integer enumeration constants represented by identifiers. Values in an enum start with 0, unless specified otherwise, and increment by 1.

For example,

```
enum months {JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC};
```

creates the new type enum months in which the identifiers are set to the integers 0 through 11. . To number the months 1 to 12, use:

```
enum months {JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC};
```



```
1  #include <stdio.h>
2  // enumeration constants represent months of the year
3  enum months {
4      JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
5  };
6
7  int main(void) {
8      // initialize array of pointers
9      const char* monthName[] = { "", "January", "February", "March",
10     "April", "May", "June", "July", "August", "September", "October",
11     "November", "December" };
12     // loop through months
13     for (enum months month = JUN; month <= DEC; ++month) {
14         printf("%2d%11s\n", month, monthName[month]);
15     }
16 }
```

Fig 05. (Enumeration Constants)

In the above program, months from June to December will be printed on the Console.



Fig 06. (Enumeration Constants)

Task 01: Bitwise Operations**[Estimated 20 minutes / 20 marks]**

Write a C program which:

- Take two integers as an input from the user
- Perform bitwise operations on the given numbers
- Display the output on the Console

Input	Output
First Number: 9 Second Number: 8	AND: 8 OR: 9 XOR: 1 Complement of first number: -10 Complement of second number: -9

Task 02: Enums**[Estimated 20 minutes / 20 marks]**

Write a C program which:

- Define three enumeration members
- Perform bitwise OR operation i.e. first enum | second enum | third enum
- Perform left shift operation on each of the enum member
- Display the output on the Console

In-Lab Activities:**Creating a Sequential-Access File:**

The following example shows how you can impose your own record structure on a file.

```

1  #include <stdio.h>
2  int main(void) {
3      FILE * cfPtr = NULL; // cfPtr = clients.txt file pointer
4      // fopen opens the file. Exit the program if unable to create the file
5      if ((cfPtr = fopen("clients.txt", "w")) == NULL) {
6          puts("File could not be opened");
7      }
8      else {
9          puts("Enter the account, name, and balance.");
10         puts("Enter EOF to end input.");
11         printf("%s", "? ");
12
13         int account = 0; // account number
14         char name[30] = ""; // account name
15         double balance = 0.0; // account balance
16
17         scanf("%d%29s%lf", &account, name, &balance);
18
19         // write account, name and balance into file with fprintf
20         while (!feof(stdin)) {
21             fprintf(cfPtr, "%d %s %.2f\n", account, name, balance);
22             printf("%s", "? ");
23             scanf("%d%29s%lf", &account, name, &balance);
24         }
25         fclose(cfPtr); // fclose closes file
26     }
27 }
28

```

Fig 07. (Sequential Access File)

Pointer to File:

Line 3 defines cfPtr as a pointer to a FILE structure. A program refers to each open file with a separate FILE pointer.

Open a File:

Line 5 calls “**fopen**” to create the file “clients.txt”. The file pointer that fopen returns is assigned to cfPtr.

Function fopen takes two arguments:

- a filename (which can include path information leading to the file’s location)
- a file open mode.

The file open mode “w” indicates fopen should open the file for writing. If the file does not exist and the file open mode is “w”, fopen creates the file. If you open an existing file, fopen discards the file’s contents without warning. This is a logical error.

End-of-File Indicator:

Line 21 calls “**feof**” to determine whether the end-of-file indicator is set for stdin. The end-of-file indicator informs the program that there’s no more data to process.

Write to a File:

Line 22 writes a record as a line of text to the file clients.txt. The “**fprintf**” function is equivalent to printf, but fprintf also receives a FILE pointer argument specifying the file to which the data will be

written. Function `fprintf` can output data to the standard output by using `stdout` as the FILE pointer argument.

Close a File:

After the user enters end-of-file, the program closes the `clients.txt` file by calling “**fclose**”, then terminates. Function `fclose` receives the FILE pointer as an argument. If you do not call `fclose` explicitly, the file closes when program execution terminates.

The execution of above program is:



```

Microsoft Visual Studio Debug Console
Enter the account, name, and balance.
Enter EOF to end input.
? 12 Saad 120000
? 07 Usman 1023930
? ^Z

```

Fig 08. (Sequential Access File)

A file named “**clients.txt**” will be created at your project path containing the entered data.



```

clients.txt - Notepad
File Edit Format View Help
12 Saad 120000.00
7 Usman 1023930.00

```

Fig 09. (Sequential Access File)

File Open Modes:

The following table summarizes the file open modes:

Mode	Description
r	Open an existing file for reading.
w	Create a file for writing. If the file already exists, discard the current contents.
a	Open or create a file for writing at the end of a file—this is for write operations that append data to a file.
r+	Open an existing file for update (reading and writing).
w+	Create a file for reading and writing. If the file already exists, discard the current contents.
a+	Open or create a file for reading and updating where all writing is done at the end of the file—that is, write operations append data to the file.
rb	Open an existing binary file for reading.
wb	Create a binary file for writing. If the file already exists, discard the current contents.
ab	Open or create a binary file for writing at the end of the file (appending).
rb+	Open an existing binary file for update (reading and writing).
wb+	Create a binary file for update. If the file already exists, discard the current contents.
ab+	Open or create a binary file for update. Writing is done at the end of the file.

Read Data from a Sequential Access File:

Now we read the contents of the above written file using “r” file mode as shown in the code below:

```

2  #include <stdio.h>
3  int main(void) {
4      FILE * cfPtr = NULL; // cfPtr = clients.txt file pointer
5      // fopen opens the file. Exit the program if unable to create the file
6      if ((cfPtr = fopen("clients.txt", "r")) == NULL) {
7          puts("File could not be opened");
8      }
9      else {
10         int account = 0; // account number
11         char name[30] = ""; // account name
12         double balance = 0.0; // account balance
13
14         printf("%-10s%-13s%\n", "Account", "Name", "Balance");
15         fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);
16         // while not end of file
17         while (!feof(cfPtr)) {
18             printf("%-10d%-13s%7.2f\n", account, name, balance);
19             fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);
20         }
21         fclose(cfPtr); // fclose closes the file
22     }
23 }

```

Fig 10. (Reading Data)

The output of the above program is:



Account	Name	Balance
12	Saad	120000.00
7	Usman	1023930.00

Fig 11. (Reading Data)

Random Access File:

Creating a Random-Access File:

“fopen” function is used to create a file as shown in the example code.

```

2  #include<stdio.h>
3  int main()
4  {
5      char ch;
6      // file pointer
7      FILE* fptr;
8      // open and creates file in write mode if it does not exist.
9      fptr = fopen("char", "w");
10     if (fptr != NULL)
11     {
12         printf("File created successfully!\n");
13     }
14     else
15     {
16         printf("Failed to create the file.\n");
17         return 0;
18     }
19     fclose(fptr);
20     return 0;
21 }

```

Fig 12. (Creating a File)

Writing Data Randomly:

The data is stored randomly in a file using `fseek()` and `fwrite()` functions.

`fseek()` Function:

If you have many records inside a file and need to access a record at a specific position, you need to loop through all the records before it to get the record.

This will waste a lot of memory and operation time. An easier way to get to the required data can be achieved using `fseek()` function.

As the name suggests, `fseek()` seeks the cursor to the given record in the file. Following is the syntax of `fseek()`:

`fseek(FILE * stream, long int offset, int whence);`

The first parameter `stream` is the pointer to the file. The second parameter is the position of the record to be found, and the third parameter specifies the location where the offset starts.

Whence	Description
SEEK_SET	Starts the offset from the beginning of the file.
SEEK_END	Starts the offset from the end of the file.
SEEK_CUR	Starts the offset from the current location in the file.

The following program writes data to the file "student.txt". It stores data at precise points in the file using `fseek()` and `fwrite()`. The file position pointer is set to a given place in the file by `fseek()`, and then the data is written by `fwrite()`.

```

2  #include <stdio.h>
3  // Student structure definition
4  struct Student {
5      char name[20]; // student name
6      int roll_number; // roll number
7  };
8  int main(){
9      FILE* fp; // file pointer
10     // The below line creates a student object with default values
11     struct Student s = { "", 0 };
12     // fopen opens the file, and exits if file cannot be opened
13     if (!(fp = fopen("student.txt", "r+"))){
14         printf("File cannot be opened.");
15         return 0;
16     }
17     // The user will enter information which will be copied to the file
18     while (1){
19         // require the user to specify roll number
20         printf("Enter roll number from (1 to 100) , -1 to end input : ");
21         scanf("%d", &s.roll_number);
22         if (s.roll_number == -1)
23             break;
24         // require the user to specify name
25         printf("Enter name : ");
26         scanf("%s", s.name);
27         fseek(fp, (s.roll_number - 1) * sizeof(s), 0);
28         fwrite(&s, sizeof(s), 1, fp);
29     }
30     fclose(fp); // fclose closes the file
31     return 0;
32 }

```

Fig 13. (Writing Data randomly)

Randomly Reading from a file:

fseek() function can be used to find a specific record in a file provided we already know where the record starts in the file and its size.

To fetch any specified record from the data file, the knowledge of two things are essential:

- Where the data starts in the file.
- Size of the data

Operation	Description
fseek(fp, 0, 0)	This takes us to the beginning of the file.
fseek(fp, 0, 2)	This takes us to the end of the file.
fseek(fp, N, 0)	This takes us to (N + 1)th bytes in the file.
fseek(fp, N, 1)	This takes us N bytes forward from the current position in the file.
fseek(fp, -N, 1)	This takes us N bytes backward from the current position in the file.
fseek(fp, -N, 2)	This takes us N bytes backward from the end position in the file.

Task 01: Maximum Digit**[20 minutes / 20 mark]**

Write a C program that:

- Take a number as an input from the user
- Find the largest digit of the number
- Store the digit in a file named your Roll No

Input	Output
1893	Maximum Digit: 9
11	Maximum Digit: 1

Task 02: Second Largest**[30 minutes / 30 marks]**

Create a C program that

- Reads four integers from a file
- Find the second largest among the numbers
- Display the Second Largest number in a new file

Input	Output
10 20 30 40	Second Largest Number: 30
10 20 10 13	Second Largest Number: 13

Task 03: Word Occurrence**[30 minutes / 30 marks]**

Create a C Program that:

- Take a word as an input from the user
- Opens a file in reading mode
- Count the occurrence of the word
- Display the result on the Console

Task 04: Batsman Average**[30 minutes / 40 marks]**

Create a C Program that:

- Take five strings (Batsman Scores) as input from the user
- Store the scores in a file
- Calculate the average of the score
- Store the Average on the file

Input	Output
10 20 30 40	Batsman Average: 25
15 20 10* 30	Batsman Average: 25

Post-Lab Activities:**Task 01: Vigenère Cipher using File Handling****[Estimated 60 minutes / 50 marks]**

Vigenère Cipher is a method of encrypting alphabetic text.

Create a C program that:

- Stores the Vigenère Cipher Table in a file
- Inputs a String from the user
- Inputs a keyword from a user
- Read the Vigenère Cipher table from the file
- Encrypt the string using Vigenère Cipher table
- Decrypt the encrypted string by using Vigenère Cipher
- Display the Encrypted and Decrypted string in a file

Vigenère Cipher Table:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Fig 24. (Post-Lab Task)

Input	Output
MY NAME THIS IS A MESSAGE	Keyword is bigger than string
THIS IS A MESSAGE KEYWORD	Encrypted: DLGO WJ D WIQOOXH Decrypted: THIS IS A MESSAGE
TEXT KEY	Encrypted: DIVD Decrypted: TEXT

Submit “.c” files named your “**Roll No**” on Google Classroom.

Submissions:

- For In-Lab Activity:
 - Save the files on your PC.
 - TA's will evaluate the tasks offline.
- For Pre-Lab & Post-Lab Activity:
 - Submit the .c file on Google Classroom and name it to your roll no.

Evaluations Metric:

- All the lab tasks will be evaluated offline by TA's
- **Division of Pre-Lab marks:** **[40 marks]**
 - Task 01: Bitwise Operations [20 marks]
 - Task 02: Enums [20 marks]
- **Division of In-Lab marks:** **[120 marks]**
 - Task 01: Maximum Digit [20 marks]
 - Task 02: Second Largest Number [30 marks]
 - Task 03: Word Occurrence [30 marks]
 - Task 04: Batsman Average [40 marks]
- **Division of Post-Lab marks:** **[50 marks]**
 - Task 01: Vigenère Cipher using File Handling [50 marks]

References and Additional Material:

- C Bitwise Operators
<https://www.programiz.com/c-programming/bitwise-operators>
- Bit Fields
<https://www.geeksforgeeks.org/bit-fields-c/>
- Enumeration Constants
<https://www.programiz.com/c-programming/c-enumeration>
- File Handling in C
<https://www.programiz.com/c-programming/c-file-input-output>

Lab Time Activity Simulation Log:

- Slot – 01 – 00:00 – 00:15: Class Settlement
- Slot – 02 – 00:15 – 00:30: In-Lab Task
- Slot – 03 – 00:30 – 00:45: In-Lab Task
- Slot – 04 – 00:45 – 01:00: In-Lab Task
- Slot – 05 – 01:00 – 01:15: In-Lab Task
- Slot – 06 – 01:15 – 01:30: In-Lab Task
- Slot – 07 – 01:30 – 01:45: In-Lab Task
- Slot – 08 – 01:45 – 02:00: In-Lab Task
- Slot – 09 – 02:00 – 02:15: In-Lab Task
- Slot – 10 – 02:15 – 02:30: In-Lab Task
- Slot – 11 – 02:30 – 02:45: Evaluation of Lab Tasks
- Slot – 12 – 02:45 – 03:00: Discussion on Post-Lab Task