Session 2

1.Write a C program to read from standard input and display on standard output.

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<fcntl.h>
#define BUFFSIZE 100
int main()
{
        int n;
        char buf[BUFFSIZE];
        while ((n = read(STDIN_FILENO, buf, BUFFSIZE)) > 0)
                if (write(STDOUT_FILENO, buf, n) != n)
                        printf("write error");
        if (n < 0)
                printf("read error");
      exit(0);

}
```

Included Headers:

- #include <stdio.h>: This header file is essential for standard input/output operations in C. It provides functions like printf for formatted output, scanf for formatted input, getchar and putchar for single character input/output, fgets and fputs for string input/output, and many others related to file streams.

- #include <stdlib.h>: This header offers various utility functions in the C standard library. While not directly used in this particular code, it might contain functions like exit (used in some implementations) or memory allocation functions (like malloc and free) that could be used in a more complex version of this program.

- #include <sys/types.h>: This header defines fundamental system data types like pid_t, uid_t, and off_t that might be needed for system calls used in more advanced file or process management scenarios. However, in this specific code, it's not strictly necessary for the core functionality. Open, read, write

- #include <fcntl.h>: This header deals with file control operations. While not used in this code, it might be relevant if you wanted to open files with specific flags (e.g., read-only, append mode) in a more advanced version.

Below, we can see the syntax for the `read()` function.

```
int read(int fileDescriptor, void *buffer, size_t bytesToRead)
```

read() function syntax

It takes in three arguments which are described below:

1. **`fileDescriptor`**: We need to provide the function with an integer file descriptor for the opened file, which the `open()` function returns when opening a file.
2. **`buffer`**: This pointer points to a buffer where data that is read will be stored.
3. **`bytesToRead`**: Here, we provide an unsigned integer variable that specifies the maximum number of bytes we want to read from the file.

- read(STDIN_FILENO, buf, BUFFSIZE): This function attempts to read up to BUFFSIZE bytes from standard input (represented by STDIN_FILENO) and stores the data in the buf array. It returns the number of bytes actually read.

- STDIN_FILENO: This is a symbolic constant defined in <unistd.h> that represents the file descriptor for standard input (usually 0).

Below we can see the syntax for the `write()` function.

```
int write(int fileDescriptor, void *buffer, size_t bytesToWrite)
```

We need to provide the function with three arguments which are discussed below:

1. `fileDescriptor`: It is an integer file descriptor for the opened file, which the `open()` function returns when opening a file.
2. `buffer`: This pointer points to a buffer containing the data we want to write into the file.
3. `bytesToWrite`: Here, we provide an unsigned integer variable that specifies the maximum number of bytes we want to write from the buffer to the file.

Writing Read Data:

- if (write(STDOUT_FILENO, buf, n) != n) printf("write error");: This conditional statement checks if the number of bytes written to standard output (STDOUT_FILENO) using write matches the number of bytes read (n).

- write(STDOUT_FILENO, buf, n): This function attempts to write n bytes from the buf array to standard output (represented by STDOUT_FILENO). It returns the number of bytes actually written.

- If the number of bytes written (write's return value) doesn't equal n, an error message ("write error") is printed using printf (from <stdio.h>). This indicates a potential problem with writing to standard output.

2.Write a program to read n characters from a file and append them back to the same file using dup2 function.

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
int main()
{
        int fd1=0,fd2=0;
        char buf[50];
        if((fd1=open("t12.txt",O_RDWR,0))<0)
                printf("file open error");

        fd2=dup(fd1);
        printf("%d %d \n",fd1, fd2);
        read(fd1,buf,10);
        lseek(fd2, 0L, SEEK_END);
        write(fd2, buf, 10);
        printf("%s\n",outbuf);
        return 0;
}
```

Syntax of open() in C

```
int open (const char* Path, int flags);
```

- **Path:** Specify the path to the file you want to open. Use an absolute path (starting with "/") when not working in the same directory as the C source file. Use a relative path (just the file name with extension) when working in the same directory.
- **Flags:** Flags are used to specify how you want to open the file. You can choose from various flags based on your requirements.

Flags	Description
O_RDONLY	It will Open the file in read-only mode.
O_WRONLY	It will Open the file in write-only mode.
O_RDWR	It will Open the file in read and write mode.
O_CREAT	It can Create a file if it doesn't exist.
O_EXCL	It can Prevent creation if it already exists.
O_APPEND	It will Open the file and places the cursor at the end of the contents.
O_ASYNC	It will Enable input and output control by signal.
O_CLOEXEC	It will Enable close-on-exec mode on the open file.
O_NONBLOCK	It will Disable the blocking of the file opened.
O_TMPFILE	Create an unnamed temporary file at the specified path.

Format

```
#define _POSIX_SOURCE
#include <unistd.h>

int dup(int fildes);
```

General description

Returns a new file descriptor that is the lowest numbered available descriptor. The new file descriptor refers to the same open file as *fildes* and shares any locks that may be associated with *fildes*.

Returned value

If successful, dup() returns a new file descriptor.

If unsuccessful, dup() returns -1 and sets errno to one of the following values:

Error Code

Description

EBADF

fildes is not a valid open file descriptor.

EMFILE

The process has already reached its maximum number of open file descriptors.

lseek( )

- Every open file has an associated "current file offset," normally a non-negative integer that measures the number of bytes from the beginning of the file.

```
#include <unistd.h>

off_t lseek(int fd, off_t offset, int whence);
```

Returns: new file offset if OK, -1 on error

- If *whence is SEEK_SET, the file's offset is set to offset bytes from the beginning of* the file.

- If *whence is SEEK_CUR, the file's offset is set to its current value plus the offset.*

- If *whence is SEEK_END, the file's offset is set to the size of the file plus the offset.*

Included Headers:

- <stdio.h>: This standard C header file provides functions for standard input/output operations, including printf for formatted output.
- <unistd.h>: This header offers various POSIX-compliant system calls, including open, read, write, close, and dup (used in this code).
- <sys/types.h>: This header defines fundamental system data types like pid_t, uid_t, and off_t that might be needed for system calls. In this case, it might be necessary for open, read, and write.
- <sys/stat.h>: This header typically provides functions for file status operations (stat, lstat, etc.
- <fcntl.h>: This header defines file control operations (open, fcntl, etc.). While open is included from <unistd.h>, including both headers might be a practice to ensure all necessary file control constants are available. However, in this specific code, it's not strictly required for the core functionality.

Opening a File:

- if ((fd1 = open("t12.txt", O_RDWR, 0)) < 0):
- open("t12.txt", O_RDWR, 0): Attempts to open the file "t12.txt" using the open system call (from <unistd.h>).
- "t12.txt": The name of the file to open.
- O_RDWR: Opening mode flags for both reading and writing.
- 0: Optional third argument sometimes used for setting permissions (not used here).

Reading from the File (fd1):

- read(fd1, buf, 10);: Reads up to 10 bytes from the file referred to by fd1 (which is also fd2) and stores the data in the buf character array.

Seeking to End of File (fd2):

- lseek(fd2, 0L, SEEK_END);: This line uses lseek (from <unistd.h>) to reposition the file offset associated with fd2 (which is also fd1) to the end of the file.
- 0L: The offset value (0 in this case).
- SEEK_END: This value indicates seeking relative to the end of the file.

Writing to the File (fd2):

- write(fd2, buf, 10);: Attempts to write the 10 bytes stored in buf to the file referred to by fd2 (which is also fd1). However, since lseek positioned the file offset to the end, this write operation will likely append the data to the end of the existing file content.

3. Write a program

a. to read first 20 characters from a file

b. seek to 10th byte from the beginning and display 20 characters from there

c. seek 10 bytes ahead from the current file offset and display 20 characters

d. display the file size

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/types.h>
int main()
{
        int file=0, n;
        char buffer[25];
        if((file=open("testfile.txt",O_RDONLY)) < -1)
                printf(“file open error\n”);
        if(read(file,buffer,20) != 20)
                printf(“file read operation failed\n”);
        else
                write(STDOUT_FILENO, buffer, 20);
        printf("\n");
```

```
if(lseek(file,10,SEEK_SET) < 0)
        printf(“lseek operation to beginning of file failed\n”);
if(read(file,buffer,20) != 20)
                printf(“file read operation failed\n”);
        else
                write(STDOUT_FILENO, buffer, 20);
        printf("\n");
```

```
if(lseek(file,10,SEEK_CUR) < 0)
printf(“lseek operation to beginning of file failed\n”);
if(read(file,buffer,20) != 20)
                printf(“file read operation failed\n”);
else
                write(STDOUT_FILENO, buffer, 20);
printf("\n");
if((n = lseek(file,0,SEEK_END)) <0)
                printf(“lseek operation to end of file failed\n”);
        printf("size of file is %d bytes\n",n);
        close(file);
        return 0;
}
```

4.Write a program to display the file content in reverse order using lseek system call.

```
#include<stdlib.h>
#include<stdio.h>
#include<fcntl.h>
#include<string.h>
#include<sys/stat.h>
#include<unistd.h>

int main(int argc, char *argv[])
{
   int source, dest, n;
   char buf;
   int filesize;
   int i;
```

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
        printf("argc = %d\n\n", argc);
        printf("Everything inside of argv[]\n");

        int i;

        for (i = 0; i < argc; i++)
        {
                printf("argv[%d] = %s\n", i, argv[i]);
        }
        return (0);
}
```

```
gcc argument.c -o argument
```

./argument
./argument source dest

```
if (argc != 3) {
      fprintf(stderr, "usage %s <source> <dest>", argv[0]);
      exit(-1);
   }
   if ((source = open(argv[1], O_RDONLY)) < 0)
   {   fprintf(stderr, "can't open source\n");
      exit(-1);
   }
   if ((dest = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC)) < 0)
   {   fprintf(stderr, "can't create dest\n");
      exit(-1);
   }
```

Checking Arguments:

- if (argc != 3) { ... }: This if statement checks if the number of command-line arguments (argc) is not equal to 3 (program name, source file, destination file). If not, it prints an error message using fprintf and exits the program with exit(-1).

Opening Source File:

- if ((source = open(argv[1], O_RDONLY)) < 0) { ... }: This block attempts to open the source file specified in argv[1] (second argument) in read-only mode (O_RDONLY) using open. If opening fails (indicated by a negative return value), an error message is printed, and the program exits with exit(-1).

Opening Destination File:

- if ((dest = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC)) < 0) { ... }: This block attempts to open the destination file specified in argv[2] (third argument) with the following flags:
- O_WRONLY: Opens the file for writing only.
- O_CREAT: Creates the file if it doesn't exist.
- O_TRUNC: If the file exists, truncates it to zero length (effectively deleting its content). If opening fails, an error message is printed, and the program exits.

```
filesize = lseek(source, (off_t) 0, SEEK_END);
   printf("Source file size is %d\n", filesize);

   for (i = filesize - 1; i >= 0; i--)
   {
      lseek(source, (off_t) i, SEEK_SET);
      if ((n = read(source, &buf, 1)) != 1)
     {
         fprintf(stderr, "can't read 1 byte");
         exit(-1);
      }
      if ((n = write(dest, &buf, 1)) != 1)
     {
         fprintf(stderr, "can't write 1 byte");
         exit(-1);
      }
   }
 write(STDOUT_FILENO, "DONE\n", 5);
    close(source);
    close(dest);
return 0;
}
```