Practical Assignment using C Programming

Q1) Create a file with hole in it.
Ans.

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

char    buf1[] = "abcdefghij";
char    buf2[] = "ABCDEFGHIJ";

int
main(void)
{
    int     fd;

    if ((fd = creat("Myfile.txt",DEFFILEMODE)) < 0)
        printf("creat error");

    if (write(fd, buf1, 10) != 10)
        printf("buf1 write error");
    /* offset now = 10 */

    if (lseek(fd, 16384, SEEK_SET) == -1)
        printf("lseek error");
    /* offset now = 16384 */

    if (write(fd, buf2, 10) != 10)
        printf("buf2 write error");
    /* offset now = 16394 */

    exit(0);
}
```

Q2) Take multiple files as Command Line Arguments and print their inode number.
Ans.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
int get_inode (int fd)
{
 struct stat buf;
int ret;
ret = fstat (fd, &buf);
if (ret < 0)
{
perror ("fstat");
//return −1;
}
return buf.st_ino;
}
int main (int argc, char *argv[])
{
int fd, inode;
if (argc < 2)
{
fprintf (stderr, "usage: %s <file>\n", argv[0]);
//return 1;
}
 fd = open (argv[1], O_RDONLY);
if (fd < 0)
{
perror ("open");
//return 1;
```

```
}
inode = get_inode(fd);
printf ("%d\n", inode);
return 0;
}
```

Output.

```
adi@adi:~/Documents$ gcc assigno02.c -0 assigno02
adi@adi:~/Documents$ ./assigno02 myfile.txt
799055
```

Q3) Write a C program to find file properties such as inode number, number of hard link,File permissions, File size, File access and modification time and so on of a given file using stat() system call.

Ans.

```
#include<stdio.h>
#include<unistd.h>
#include<dirent.h>
#include<string.h>
#include<time.h>
#include<stdlib.h>
#include<sys/stat.h>
#include<sys/types.h>
int main(int argc,char* argv[])
{
struct stat info;
if(argc!=2)
{
printf("Enter a filename");
```

```
scanf("%s",argv[1]);
}
if(stat(argv[1],&info)==-1)
{
printf("stat error/n");
exit(0);
}
printf("inode number=%d\n",info.st_ino);
printf("size = %d",(long)info.st_size);
printf("last file access = %s\n",ctime(&info.st_atime));
printf("notification time = %s\n",ctime(&info.st_mtime));
printf("No of Hardlink = %d\n",info.st_nlink);
printf("File Permissions : \n");
printf((info.st_mode && S_IRUSR)?"r":"-");
printf((info.st_mode && S_IWUSR)?"w":"-");
printf((info.st_mode && S_IXUSR)?"x":"-");
return 0;

}
```

Output.

```
adi@adi:~/Documents$ cc assigno03.c -0 assigno03
adi@adi:~/Documents$ ./assigno03 myfile.txt
inode number=799055
size = 16394last file access = Sun Dec  3 21:09:16 2023
notification time = Sun Dec  3 21:09:12 2023
No of Hardlink = 1
File Permissions :
rwx
```

Q4) Print the type of file where file name accepted through Command Line.
Ans.

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

int main(int argc, char *argv[]) {
    struct stat fileStat;
    char fnm[30];
    int fd = 0;

    printf("Enter file name= ");
    scanf("%s", fnm);

    if ((fd = open(fnm, O_RDONLY)) == -1) {
        perror("open ");
        exit(1);
    }

    if (fstat(fd, &fileStat) < 0)
        return 1;

    printf("Information for %s\n", fnm);

    if (S_ISREG(fileStat.st_mode)) {
        printf("File Type: Regular File\n");
    } else if (S_ISDIR(fileStat.st_mode)) {
        printf("File Type: Directory\n");
    } else if (S_ISCHR(fileStat.st_mode)) {
```

```
        printf("File Type: Character Device\n");
    } else if (S_ISBLK(fileStat.st_mode)) {
        printf("File Type: Block Device\n");
    } else if (S_ISFIFO(fileStat.st_mode)) {
        printf("File Type: FIFO/Named Pipe\n");
    } else if (S_ISSOCK(fileStat.st_mode)) {
        printf("File Type: Socket\n");
    } else if (S_ISLNK(fileStat.st_mode)) {
        printf("File Type: Symbolic Link\n");
    } else {
        printf("File Type: Unknown\n");
    }

    system("pause");
    return 0;
}
```

Output.

```
adi@adi:~/Documents$ cc assigno04.c -0 assigno04
adi@adi:~/Documents$ ./assigno04
Enter file name= myfile.txt
Information for myfile.txt
File Type: Regular File
sh: 1: pause: not found
```

Q5) Write a C program to find whether a given file is present in current directory or not.

Ans.

```
#include<stdio.h>
#include<unistd.h>
int main(int argc,char *argv[])
{
```

```
if(access(argv[1],F_OK)==0)
printf("File %s exists.",argv[1]);
else
printf("File not exists.");
return 0;
}
```

Output.

```
adi@adi:~/Documents$ gcc assigno05.c -o assigno05
adi@adi:~/Documents$ ./assigno05 myfile.txt
File myfile.txt exists.
```

Q6) Write a C program that a string as an argument and return all the files that begins with that name in the current directory. For example > ./a.out foo will return all file names that begins with foo

Ans.

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <string.h>

void listFilesWithPrefix(const char *prefix) {
    DIR *dir;
    struct dirent *entry;

    // Open the current directory
    dir = opendir(".");

    if (dir == NULL) {
        perror("opendir");
        exit(EXIT_FAILURE);
```

```
    }

    // Iterate over the directory entries
    while ((entry = readdir(dir)) != NULL) {
        // Check if the entry is a regular file and starts with the
specified prefix
        if (entry->d_type == DT_REG && strncmp(entry-
>d_name, prefix, strlen(prefix)) == 0) {
            printf("%s\n", entry->d_name);
        }
    }

    // Close the directory
    closedir(dir);
}

int main(int argc, char *argv[]) {
    // Check if the command-line argument is provided
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <prefix>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    // List files with the specified prefix
    listFilesWithPrefix(argv[1]);

    return 0;
}
```

Output.

```
adi@adi:~/Documents$ gcc assigno06.c -o assigno06
adi@adi:~/Documents$ ./assigno06 foo
foo123.txt
foobar.txt
```

Q7) Read the current directory and display the name of the files, no of files in current directory
Ans.

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

int main() {
    DIR *dir;
    struct dirent *entry;

    // Open the current directory
    dir = opendir(".");

    if (dir == NULL) {
        perror("opendir");
        exit(EXIT_FAILURE);
    }

    // Count the number of files
    int fileCount = 0;

    // Iterate over the directory entries
    while ((entry = readdir(dir)) != NULL) {
        // Check if the entry is a regular file
        if (entry->d_type == DT_REG) {
            printf("%s\n", entry->d_name);
            fileCount++;
        }
    }

    // Close the directory
    closedir(dir);
```

```
    // Display the total number of files
    printf("\nTotal number of files: %d\n", fileCount);

    return 0;
}
```

Output.

```
adi@adi:~/Documents$ ^C
adi@adi:~/Documents$ gcc assigno07.c -o assigno07
adi@adi:~/Documents$ ./assigno07
assigno05
foo123.txt
foobar.txt
assigno01.c
assigno02.c
assigno07.c
assigno03
assigno06
assigno05.c
myfile.txt
assigno02
```

Q8) Write a C program which receives file names as command line arguments and display those filenames in ascending order according to their sizes. I) (e.g $ a.out a.txt b.txt c.txt, …)
Ans.

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

typedef struct {
    char *name;
    off_t size;
```

```
} FileEntry;

// Function to compare two FileEntry structures based on size
int compareFileSize(const void *a, const void *b) {
    return ((FileEntry *)a)->size - ((FileEntry *)b)->size;
}

int main(int argc, char *argv[]) {
    // Check if command-line arguments are provided
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <file1> <file2> ... <fileN>\n",
argv[0]);
        return 1;
    }

    // Allocate an array of FileEntry structures
    FileEntry *files = malloc((argc - 1) * sizeof(FileEntry));

    // Populate the array with filenames and their sizes
    for (int i = 1; i < argc; i++) {
        files[i - 1].name = argv[i];

        struct stat st;
        if (stat(argv[i], &st) == 0) {
            files[i - 1].size = st.st_size;
        } else {
            perror("stat");
            return 1;
        }
    }

    // Sort the array based on file sizes
    qsort(files, argc - 1, sizeof(FileEntry), compareFileSize);

    // Display the sorted filenames
```

```
    printf("Filenames in ascending order according to their
sizes:\n");
    for (int i = 0; i < argc - 1; i++) {
        printf("%s\n", files[i].name);
    }

    // Free allocated memory
    free(files);

    return 0;
}
```

Output.

```
adi@adi:~/Documents$ gcc assigno8.c -o assigno8
adi@adi:~/Documents$ ./assigno8 a.txt b.txt c.txt
Filenames in ascending order according to their sizes:
c.txt
b.txt
a.txt
```

Q9) Display all the files from current directory which are created in particular month

Ans.

```
#include <stdio.h>
#include <dirent.h>
#include <string.h>
#include <sys/stat.h>
#include <time.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    char mon[100];
    DIR *dp;
```

```
    struct dirent *ep;
    struct stat sb;

    // Check if the target month is provided as a command-line
argument
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <target_month>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    dp = opendir("./");

    if (dp != NULL) {
        while ((ep = readdir(dp)) != NULL) {
            if (stat(ep->d_name, &sb) == -1) {
                perror("stat");
                exit(EXIT_SUCCESS);
            }

            // Convert the creation time to a formatted string
            strftime(mon, sizeof(mon), "%b",
localtime(&sb.st_ctime));

            if (strcmp(mon, argv[1]) == 0) {
                printf("%s\t\t%s", ep->d_name,
ctime(&sb.st_ctime));
            }
        }

        (void)closedir(dp);
    }

    return 0;
}
```

Output.

```
adi@adi:~/Documents$ gcc assigno9.c -o assigno9
adi@adi:~/Documents$ ./assigno9 Dec
fille2.txt        Sun Dec  3 22:48:03 2023
..           Sun Dec  3 22:23:45 2023
assigno8.c           Sun Dec  3 22:55:36 2023
assigno05            Sun Dec  3 22:22:56 2023
foo123.txt           Sun Dec  3 22:34:46 2023
file3.txt         Sun Dec  3 22:48:21 2023
foobar.txt           Sun Dec  3 22:34:30 2023
assigno01.c          Sun Dec  3 21:08:59 2023
assigno02.c          Sun Dec  3 21:18:57 2023
```

Q10) Display all the files from current directory whose size is greater that n Bytes Where n is accept from user.
Ans.

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <sys/stat.h>
#include <string.h>
void displayFilesGreaterThanSize(const char *directoryPath, long sizeThreshold) {
    DIR *dir;
    struct dirent *entry;

    // Open the directory
    dir = opendir(directoryPath);

    if (dir == NULL) {
        perror("opendir");
        exit(EXIT_FAILURE);
    }
```

```
    // Iterate over the directory entries
    while ((entry = readdir(dir)) != NULL) {
        // Skip "." and ".." entries
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry-
>d_name, "..") == 0) {
            continue;
        }

        // Get file information
        struct stat fileStat;
        char filePath[256];
        snprintf(filePath, sizeof(filePath), "%s/%s",
directoryPath, entry->d_name);

        if (stat(filePath, &fileStat) == -1) {
            perror("stat");
            continue;
        }

        // Check if the file size is greater than the specified
threshold
        if (fileStat.st_size > sizeThreshold) {
            printf("%s\t%ld bytes\n", entry->d_name,
(long)fileStat.st_size);
        }
    }

    // Close the directory
    closedir(dir);
}

int main() {
    char directoryPath[256];
    long sizeThreshold;
```

```
    // Get the directory path from the user
    printf("Enter directory path: ");
    scanf("%255s", directoryPath);

    // Get the size threshold from the user
    printf("Enter size threshold (in bytes): ");
    scanf("%ld", &sizeThreshold);

    // Display files greater than the specified size
    displayFilesGreaterThanSize(directoryPath,
sizeThreshold);

    return 0;
}
```

Output.

```
adi@adi:~/Documents$ gcc assigno10.c -o assigno10
adi@adi:~/Documents$ ./assigno10
Enter directory path: /home/adi
Enter size threshold (in bytes): 1000
.local      4096 bytes
.bashrc   3771 bytes
.cache     4096 bytes
Music     4096 bytes
Pictures  4096 bytes
Videos    4096 bytes
Documents     4096 bytes
.thunderbird  4096 bytes
Public     4096 bytes
Desktop  4096 bytes
.config    4096 bytes
Downloads     4096 bytes
.bash_history 1008 bytes
```

Templates 4096 bytes
snap 4096 bytes
.mozilla 4096 bytes

Q11) Write a C Program that demonstrates redirection of standard output to a file.
Ans.

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
main(int argc, char *argv[])
{
char d[50];
if(argc==2)
{
bzero(d,sizeof(d));
strcat(d,"ls ");
strcat(d,"> ");
strcat(d,argv[1]);
system(d);
}
else
printf("\nInvalid No. of inputs");
}
```

Output.

```
adi@adi:~/Documents$ gcc  -o assigno11.out assigno11.c
di@adi:~/Documents$ ls
assigno01.c  assigno04.c  assigno07.c
assigno8     c.txt        foobar.txt
assigno02    assigno05    assigno10
assigno8.c   f1           myfile.txt
```

```
assigno02.c  assigno05.c  assigno10.c
assigno9    file1.txt   test
assigno03    assigno06    assigno11
assigno9.c  file3.txt
assigno03.c  assigno06.c  assigno11.c
a.txt       fille2.txt
assigno04    assigno07    assigno11.out
b.txt       foo123.txt
adi@adi:~/Documents$ cat > f1
^Z
[2]+  Stopped                 cat > f1
adi@adi:~/Documents$ ./assigno11.out f1
adi@adi:~/Documents$ cat f1
assigno01.c
assigno02
assigno02.c
assigno03
assigno11.c
assigno11.out
a.txt
b.txt
c.txt
f1
file1.txt
file3.txt
fille2.txt
foo123.txt
foobar.txt
myfile.txt
test
```

Q12) Write a C program that will only list all subdirectories in alphabetical order from current directory.
Ans.

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
int
main(void)
{
struct dirent **namelist;
int n;
int i=0;
n = scandir(".", &namelist, 0, alphasort);
if (n < 0)
perror("scandir");
else {
while (i<n) {
printf("%s\n", namelist[i]->d_name);
free(namelist[i]);
i++;
}
free(namelist);
}
}
```

Output.

```
adi@adi:~/Documents$ gcc assigno12.c -o assigno12
adi@adi:~/Documents$ ./assigno12
.
..
a.txt
assigno01.c
assigno02
assigno02.c
assigno03
assigno03.c
assigno04
```

assigno04.c
assigno05
assigno05.c

Q13) Write a C program that redirects standard output to a file output.txt. (use of dup and open system call).
Ans.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
int main(void) {
int number1, number2, sum;
int input_fds = open("./input.txt", O_RDONLY);
if(dup2(input_fds, STDIN_FILENO) < 0) {
printf("Unable to duplicate file descriptor.");
exit(EXIT_FAILURE);
}
scanf("%d %d", &number1, &number2);
sum = number1 + number2;
printf("%d + %d = %d\n", number1, number2, sum);
return EXIT_SUCCESS;
}
```

Output.

```
adi@adi:~/Documents$ gcc assigno13.c -o assigno13
adi@adi:~/Documents$ ./assigno13 input.txt
10 + 20 = 30
```

Q14) Write a C program to Identify the type (Directory, character device, Block device,Regular file, FIFO or pipe, symbolic link or socket) of given file using stat() system call.
Ans.

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

void identifyFileType(const char *path) {
    struct stat fileStat;

    // Use stat() to get information about the file
    if (stat(path, &fileStat) == -1) {
        perror("Error in stat");
        return;
    }

    // Check the file type using st_mode
    if (S_ISDIR(fileStat.st_mode)) {
        printf("%s is a directory\n", path);
    } else if (S_ISCHR(fileStat.st_mode)) {
        printf("%s is a character device\n", path);
    } else if (S_ISBLK(fileStat.st_mode)) {
        printf("%s is a block device\n", path);
    } else if (S_ISREG(fileStat.st_mode)) {
        printf("%s is a regular file\n", path);
    } else if (S_ISFIFO(fileStat.st_mode)) {
        printf("%s is a FIFO or pipe\n", path);
    } else if (S_ISLNK(fileStat.st_mode)) {
        printf("%s is a symbolic link\n", path);
    } else if (S_ISSOCK(fileStat.st_mode)) {
        printf("%s is a socket\n", path);
    } else {
```

```
        printf("%s is of unknown type\n", path);
    }
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <file_path>\n", argv[0]);
        return 1;
    }

    identifyFileType(argv[1]);

    return 0;
}
```

Output.

```
adi@adi-:Documents$ gcc assigno14.c  -o assigno14
adi@adi:~/Documents$ ./assigno14 /home/adi
/home/adi is a directory
```

Q15)Generate parent process to write unnamed pipe and will read from it.

Ans

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

#define BUFFER_SIZE 1024

int main() {
    int pipe_fd[2]; // Pipe file descriptors

    // Create the pipe
    if (pipe(pipe_fd) == -1) {
```

```
        perror("Pipe creation failed");
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork(); // Fork a child process

    if (pid == -1) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    }

    if (pid > 0) {
        // Parent process
        close(pipe_fd[0]); // Close unused read end

        char message[] = "Hello, child process!";

        // Write to the pipe
        if (write(pipe_fd[1], message, sizeof(message)) == -1) {
            perror("Write to pipe failed");
            exit(EXIT_FAILURE);
        }

        close(pipe_fd[1]); // Close write end

        printf("Parent process wrote to the pipe: %s\n",
message);
    } else {
        // Child process
        close(pipe_fd[1]); // Close unused write end

        char buffer[BUFFER_SIZE];

        // Read from the pipe
```

```
        ssize_t bytesRead = read(pipe_fd[0], buffer,
sizeof(buffer));

        if (bytesRead == -1) {
            perror("Read from pipe failed");
            exit(EXIT_FAILURE);
        }

        close(pipe_fd[0]); // Close read end

        printf("Child process read from the pipe: %.*s\n",
(int)bytesRead, buffer);
    }

    return 0;
}
```

Output.

```
adi@adi:~/Documents$ gcc assigno15.c  -o assigno15
adi@adi:~/Documents$ ./assigno15
Parent process wrote to the pipe: Hello, child process!
Child process read from the pipe: Hello, child process!
```

Q16) Handle the two-way communication between parent and child processes using pipe.

Ans.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
```

```
#define BUFFER_SIZE 256

int main() {
    int parent_to_child[2]; // Pipe for parent to child
    int child_to_parent[2]; // Pipe for child to parent
    pid_t pid;
    char parent_msg[BUFFER_SIZE];
    char child_msg[BUFFER_SIZE];

    if (pipe(parent_to_child) == -1 || pipe(child_to_parent) == -
1) {
        perror("Pipe creation failed");
        exit(EXIT_FAILURE);
    }

    pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    }

    if (pid > 0) { // Parent process
        close(parent_to_child[0]); // Close reading end in the
parent to child pipe
        close(child_to_parent[1]); // Close writing end in the child
to parent pipe

        printf("Enter message for child: ");
        fgets(parent_msg, BUFFER_SIZE, stdin);
        write(parent_to_child[1], parent_msg, strlen(parent_msg)
+ 1); // Write to the parent to child pipe

        read(child_to_parent[0], child_msg, BUFFER_SIZE); //
Read from the child to parent pipe
```

```
        printf("Message received from child: %s", child_msg);

        close(parent_to_child[1]); // Close writing end in the
parent to child pipe
        close(child_to_parent[0]); // Close reading end in the child
to parent pipe
    } else { // Child process
        close(parent_to_child[1]); // Close writing end in the
parent to child pipe
        close(child_to_parent[0]); // Close reading end in the child
to parent pipe

        read(parent_to_child[0], child_msg, BUFFER_SIZE); //
Read from the parent to child pipe
        printf("Message received from parent: %s", child_msg);

        printf("Enter message for parent: ");
        fgets(child_msg, BUFFER_SIZE, stdin);
        write(child_to_parent[1], child_msg, strlen(child_msg) +
1); // Write to the child to parent pipe

        close(parent_to_child[0]); // Close reading end in the
parent to child pipe
        close(child_to_parent[1]); // Close writing end in the child
to parent pipe
    }

    return 0;
}
```

Output.

```
adi@adi:~/Documents$ gcc assigno16.c  -o assigno16
adi@adiDocuments$ ./assigno16
Enter message for child: hi
```

Message received from parent: hi
Enter message for parent: hey
Message received from child: hey

Q17)Demonstrate the use of atexit() function.
Ans.

```
#include <stdio.h>
#include <stdlib.h>
void functionA () {
printf("This is functionA\n");
}
int main () {
/* register the termination function */
atexit(functionA );
printf("Starting main program...\n");
printf("Exiting main program...\n");
return(0);
}
```

Output.

```
adi@adi:~/Documents$ gcc assigno17.c  -o assigno17
adi@adi:~/Documents$ ./assigno17
Starting main program...
Exiting main program...
This is functionA
```

Q18) Write a C program to demonstrates the different behaviour that can be seen with automatic, global, register, static and volatile variables (Use setjmp() and longjmp() system call)
Ans.

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

```
#include <setjmp.h>
jmp_buf buffer;
// Global variable
int globalVar = 1;
// Static variable
static int staticVar = 2;
void demonstrateVariables() {
// Automatic variable
int autoVar = 3;
// Register variable
register int regVar = 4;
// Volatile variable
volatile int volVar = 5;
printf("Global Variable: %d\n", globalVar++);
printf("Static Variable: %d\n", staticVar++);
printf("Automatic Variable: %d\n", autoVar++);
printf("Register Variable: %d\n", regVar++);
printf("Volatile Variable: %d\n", volVar++);
}
int main() {
int jumpResult = setjmp(buffer);
if (jumpResult == 0) {
// Initial execution
printf("Initial Execution:\n");
demonstrateVariables();
// Jump to setjmp point
longjmp(buffer, 1);
} else {
// After longjmp
printf("\nAfter longjmp:\n");
demonstrateVariables();
}
return 0;
}
```

Output.

```
adi@adi:~/Documents$ gcc assigno18.c  -o assigno18
adi@adi:~/Documents$ ./assigno18
Initial Execution:
Global Variable: 1
Static Variable: 2
Automatic Variable: 3
Register Variable: 4
Volatile Variable: 5

After longjmp:
Global Variable: 2
Static Variable: 3
Automatic Variable: 3
Register Variable: 4
Volatile Variable: 5
```

Q19).Implement the following unix/linux command (use fork, pipe and exec system call) ls –l | wc.
Ans.

```
#include <stdio.h>
#include <setjmp.h>
jmp_buf buff;
int globalv=1;
void demov()
{
int autov=2;
register int regv=3;
static int staticv=4;
volatile int volatilev=5;
printf("global variable:%d\n",globalv);
printf("automatic variable:%d\n",autov);
```

```
printf("register variable:%d\n",regv);
printf("static variable:%d\n",staticv);
printf("volatile variable:%d\n",volatilev);
longjmp(buff,1);
}
int main(){
if(setjmp(buff)==0)
{
printf("first time through setjmp()\n");
demov();
}
else
{
printf("returning through longjmp()\n");
}
return 0;
}
```

Output.

```
adi@adi:~/Documents$ gcc assigno19.c  -o assigno19
adi@adi:~/Documents$ ./assigno19
first time through setjmp()
global variable:1
automatic variable:2
register variable:3
static variable:4
volatile variable:5
returning through longjmp()
```

Q20)Write a C program that print the exit status of a terminated child process
Ans.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void) {
    pid_t pid = fork();

    if (pid == 0) {
        // Child process
        // The pathname of the file passed to execl()
        // is intentionally incorrect to generate an error
        execl("/bin/sh", "bin/sh", "-c", "./nopath", NULL);

        // If execl() fails
        perror("execl");
        exit(EXIT_FAILURE);
    }

    int status;
    waitpid(pid, &status, 0);

    if (WIFEXITED(status)) {
        // Child process terminated normally
        int exit_status = WEXITSTATUS(status);
        printf("Exit status of the child was %d\n", exit_status);
    } else {
        // Child process terminated abnormally
```

```
        printf("Child process terminated abnormally\n");
    }

    return 0;
}
```

Output.

```
adi@adi:~/Documents$ gcc assigno24.c  -o assigno24
adi@adi:~/Documents$ ./assigno24
NULL: 1: ./nopath: not found
Exit status of the child was 127
```

Q21)Write a C program to create an unnamed pipe. The child process will write following three messages to pipe and parent process display it.
Message1 = “Hello World”
Message2 = “Hello SPPU”
Message3 = “Linux is Funny”

Ans.

```
#include<stdio.h>
#include<unistd.h>
int main() {
int pipefds[2];
int returnstatus;
char writemessages[3][20]={"Hello World", "Hello SPPU","Linux is Funny"};
char readmessage[20];
returnstatus = pipe(pipefds);
if (returnstatus == -1) {
printf("Unable to create pipe\n");
return 1;
}
```

```
int child = fork();
if(child==0){
printf("Child is Writing to pipe - Message 1 is %s\n",
writemessages[0]);
write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
printf("Child is Writing to pipe - Message 2 is %s\n",
writemessages[1]);
write(pipefds[1], writemessages[1], sizeof(writemessages[1]));
printf("Child is Writing to pipe - Message 3 is %s\n",
writemessages[2]);
write(pipefds[1], writemessages[2], sizeof(writemessages[2]));
}
else
{
read(pipefds[0], readmessage, sizeof(readmessage));
printf("Parent Process is Reading from pipe – Message 1 is
%s\n",
readmessage);
read(pipefds[0], readmessage, sizeof(readmessage));
printf("Parent Process is Reading from pipe – Message 2 is
%s\n",
readmessage);
read(pipefds[0], readmessage, sizeof(readmessage));
printf("Parent Process is Reading from pipe – Message 3 is
%s\n",
readmessage);
}
return 0;
}
```

Output.

```
di@adi:~/Documents$ gcc assigno21.c  -o assigno21
adi@adi:~/Documents$ ./assigno21
Child is Writing to pipe - Message 1 is Hello World
```

Child is Writing to pipe - Message 2 is Hello SPPU
Child is Writing to pipe - Message 3 is Linux is Funny
Parent Process is Reading from pipe – Message 1 is Hello World
Parent Process is Reading from pipe – Message 2 is Hello SPPU
Parent Process is Reading from pipe – Message 3 is Linux is Funny

Q22)Write a C program to get and set the resource limits such as files, memory associated with a process.
Ans.

```
#include <stdio.h>
#include <sys/resource.h>

void print_resource_limits() {
    struct rlimit rlim;

    // Get the current resource limit for the maximum number of open files
    if (getrlimit(RLIMIT_NOFILE, &rlim) == 0) {
        printf("Current maximum number of open files: %lu\n", (unsigned long)rlim.rlim_cur);
        printf("Maximum allowed number of open files: %lu\n", (unsigned long)rlim.rlim_max);
    } else {
        perror("getrlimit");
    }
}

void set_resource_limit(unsigned long new_limit) {
    struct rlimit rlim;

    // Get the current resource limit
```

```
    if (getrlimit(RLIMIT_NOFILE, &rlim) == 0) {
        // Set the new resource limit
        rlim.rlim_cur = new_limit;
        if (setrlimit(RLIMIT_NOFILE, &rlim) == 0) {
            printf("New maximum number of open files set
successfully.\n");
        } else {
            perror("setrlimit");
        }
    } else {
        perror("getrlimit");
    }
}

int main() {
    printf("Before setting resource limits:\n");
    print_resource_limits();

    // Set a new resource limit (change this value as needed)
    unsigned long new_limit = 1000;
    set_resource_limit(new_limit);

    printf("\nAfter setting resource limits:\n");
    print_resource_limits();

    return 0;
}
```

Output.

```
adi@adi:~/Documents$ gcc assigno22.c  -o assigno22
adi@adi:~/Documents$ ./assigno22
Before setting resource limits:
Current maximum number of open files: 1024
Maximum allowed number of open files: 1048576
```

New maximum number of open files set successfully.

After setting resource limits:
Current maximum number of open files: 1000
Maximum allowed number of open files: 1048576

Q23)Write a C program to send SIGALRM signal by child process to parent process and parent process make a provision to catch the signal and display alarm is fired.(Use Kill, fork, signal and sleep system call)
Ans.

```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>

void Dingdong(int signum)
{
   printf("Ding!\n");
   exit(1);
}

int main(int argc, char *argv[])
{
   if (argc != 2)
   {
      printf("Usage: %s <pause_seconds>\n", argv[0]);
      return 1;
   }

   int PauseSecond = atoi(argv[1]);
```

```
    if (PauseSecond <= 0)
    {
        printf("Invalid pause seconds\n");
        return 1;
    }

    if (fork() == 0)
    {
        printf("Child process waiting for alarm to go off\n");
        printf("%d second pause\n", PauseSecond);
        sleep(PauseSecond);
        kill(getpid(), SIGALRM);
    }
    else
    {
        printf("Parent process starting alarm\n");

        // Registering the signal handler for SIGALRM
        signal(SIGALRM, Dingdong);

        // Setting an alarm to trigger after PauseSecond seconds
        alarm(PauseSecond);

        // Waiting for the child process to finish
        wait(NULL);

        printf("Parent process done\n");
    }

    return 0;
}
```

Output.

```
adi@adi:~/Documents$ gcc assigno25.c  -o assigno25
```

adi@adi:~/Documents$ ./assigno25 3
Parent process starting alarm
Child process waiting for alarm to go off
3 second pause
Ding!