PART1:

Input checks for my program

```
if(argc != 3 && argc != 4){
    fprintf(stderr,"Only 2 or 3 arguments allowed.\n");
    exit(1);
}
if(numOfBytes <= 0){
    fprintf(stderr,"Number of bytes should be bigger than zero.\n");
    exit(1);
}
```

The part on deciding how to write to the file, if 2 arguments given O_APPEND will be added otherwise lseek() will be used.

```
if(argc == 3){
    fd = open(filename,  O_WRONLY | O_CREAT | O_APPEND,0666);
    while (numOfBytes > 0)
    {
        write(fd,byte,1);
        numOfBytes--;
    }
}else{
    fd = open(filename,  O_WRONLY | O_CREAT,0666);
    while (numOfBytes > 0)
    {
        lseek(fd,0,SEEK_END);
        write(fd,byte,1);
        numOfBytes--;
    }
}
```

Outputs:

```
ry@ry-Nitro-AN515-54:~/Desktop/Systems_HW1/part1$ ./output file2.txt 1000000 x & ./output file2.txt 1000000 x
[1] 5254
Successful
Successful
[1]+  Done                    ./output file2.txt 1000000 x
ry@ry-Nitro-AN515-54:~/Desktop/Systems_HW1/part1$ ./output file1.txt 1000000 & ./output file1.txt 1000000
[1] 5276
Successful
Successful
[1]+  Done                    ./output file1.txt 1000000
ry@ry-Nitro-AN515-54:~/Desktop/Systems_HW1/part1$ ls -l file1.txt
-rw-rw-r-- 1 ry ry 2000000 Mar 27 17:20 file1.txt
ry@ry-Nitro-AN515-54:~/Desktop/Systems_HW1/part1$ ls -l file2.txt
-rw-rw-r-- 1 ry ry 1949962 Mar 27 17:20 file2.txt
```

2 millions of bytes have been written to file1.txt as expected, but 1.949.962 bytes have been written to file2.txt this is because when bytes are written to file2.txt each time because of O_APPEND the newly added byte is added to the end, but when we use lseek we set the position for the byte and this causes overwrites sometimes.

PART2:
newdup function is implemented as below, this way it duplicates old file descriptor using the lowest numbered available descriptor.

```
int newdup(int oldfd){
    return fcntl (oldfd, F_DUPFD, 0);
}
```

newdup2 function is implemented as below, if oldfile does not exist errno is set and function returns -1. If old and new file descriptors are the same then returns one of them, if old file descriptor exists and old and new file descriptors are different then new file descriptor is closed and set to old file descriptor as in dup2 function.

```
int newdup2(int oldfd, int newfd){
    // if old file descriptor does not exist, se
    if(fcntl(oldfd,F_GETFL) == -1){
        errno = EBADF;
        return -1;
    }
    // if old file descriptor and new file descr
    if (oldfd == newfd) {
        return newfd;
    }
    // otherwise close the new file descriptor a
    else {
        close(newfd);
        return fcntl(oldfd, F_DUPFD, newfd);
    }
```

Main:
Firstly open two files. And find the lowest numbered available descriptor for fd5.

```
int fd1 = open("file1.txt", O_RDONLY | O_WRONLY | O_APPEND | O_CREAT,0666);
int fd2 = open("file2.txt", O_RDONLY | O_WRONLY | O_APPEND | O_CREAT,0666);
int fd5 = newdup(0); // sets fd5 to the lowest available descriptor
int fd3;
```

After execution output is like this.

```
fd after newdup(0):3
fd1 created:3
fd2 created:4
fd5 after newdup:5
```

Execute these lines: first write should write to file2.txt after newdup2(fd1,fd2) function file2.txt is closed and fd2 becomes a duplicate of fd1, and second write function will write to file1.txt which is descripted by fd1 (also fd2 implicitly).

```
fd3 after newdup2(fd1:3,fd2:4):4
error check: Success
```

And lastly, newdup2(fd,15) is tested where 15 is invalid descriptor. So an error should be returned.

```
int fd4 = newdup2(fd2, 15); // invalid new file descr
printf("fd4 after newdup2(fd2:%d,15):%d\n",fd2, fd4);
perror("error check");
```

And we get bad file descriptor error as expected.

```
fd4 after newdup2(fd2:4,15):15
error check: Bad file descriptor
```

PART3:

Firstly, open the two files.

```
int fd1 = open("file1.txt", O_RDONLY | O_WRONLY | O_APPEND | O_CREAT,0666);
int fd2 = open("file2.txt", O_RDONLY | O_WRONLY | O_APPEND | O_CREAT,0666);
```

After that, set the current offsets differently and print it to see.

```
lseek(fd1,5,SEEK_SET);
lseek(fd2,3,SEEK_SET);
off_t fd1_off = lseek(fd1, 0, SEEK_CUR);
off_t fd2_off = lseek(fd2, 0, SEEK_CUR);

printf("fd1_offset:%ld,  fd2_offset:%ld\n",fd1_off,fd2_off);
```

```
fd1:3, fd2:4
fd1_offset:5,  fd2_offset:3
```

And write to files, “1” will be written to file1.txt, “0” will be written to file2.txt

```
write(fd1,(const char*)"1",1); //written to file1.txt
write(fd2,(const char*)"0",1); // written to file2.txt
```

After that, use dup2 to duplicate fd1 to fd2 and check offsets

```
dup2(fd1,fd2);

printf("After dup2(fd1,fd2)\n");
fd1_off = lseek(fd1, 0, SEEK_CUR);
fd2_off = lseek(fd2, 0, SEEK_CUR);
printf("fd1_offset:%ld,  fd2_offset:%ld\n",fd1_off,fd2_off);
```

```
After dup2(fd1,fd2)
fd1_offset:1,  fd2_offset:1
```

Write to fd1 and fd2 again.

```
write(fd1,(const char*)"2",1); // written to file1.txt
write(fd2,(const char*)"3",1); // written to file1.txt
```

This time both write operations should be made to file1.txt, check offsets again.

```
fd1_off = lseek(fd1, 0, SEEK_CUR);
fd2_off = lseek(fd2, 0, SEEK_CUR);
printf("fd1_offset:%ld,  fd2_offset:%ld\n",fd1_off,fd2_off);
```

```
fd1_offset:3,  fd2_offset:3
```

As seen offsets are the same. Let’s check the files:

file1.txt

```
123
```

file2.txt

```
0
```

As expected, 1,2 and 3 are written to file1.txt 0 is written to file2.txt. Notice this 3 was written by using fd2 which opened file2.txt at the beginning.