

EXPERIMENTS ON APPEND MODE IN FILE HANDLING



GROUP NO: 04

MEMBERS' NAME & ROLL NO:

- | | |
|----------------------------|---------|
| 1. Sanju Basak - | 1805064 |
| 2. Maneesha Rani Saha - | 1805076 |
| 3. Shehabul Islam Sawraz - | 1805088 |
| 4. Utchchhwas Singha - | 1805100 |
| 5. Md. Asif Haider - | 1805112 |

Question - 1: Is it possible to write into a file opened in append mode (“a”) at a position rather than the last position of the file?

Observation:

a. Trying to write before the end of the file:

```
printf("Beginning position: %d\n", ftell(fp));
if (fseek(fp, 2, SEEK_SET)) {
    printf("Error!\n");
    exit(1);
}
printf("Position after fseek: %d\n", ftell(fp));
fputc(buf[0], fp);
printf("Position after fputc: %d\n", ftell(fp));
```

Console Output:

```
Beginning position: 0
Position after fseek: 2
Position after fputc: 14
```

Here, we tried to overwrite data which was already present in the file. The opened file had 13 characters (i.e. it has a size of 13 bytes). After just opening the file, using `ftell()` we get the position 0 which means that the file was pointing at the beginning. Then using `fseek()` we moved 2 bytes forward. Now, the next write operation was supposed to take place at position 2. But after writing a char using `fputc()`, the position of the file is at 14 which is the position of the end of the file.

b. Trying to write after the end of the file:

```
fseek(fp, 0, SEEK_END);
printf("END position: %d\n", ftell(fp));
if (fseek(fp, 5, SEEK_END)) {
    printf("Error!\n");
    exit(1);
}
printf("Position after fseek: %d\n", ftell(fp));
fputc(buf[0], fp);
printf("Position after fputc: %d\n", ftell(fp));
```

Console Output:

```
END position: 12
Position after fseek: 17
Position after fputc: 13
```

Here, we tried to write after the end of the file. The end position was at 12. Using `fseek()` we moved to position 17. Then writing a character using `fputc()`, the position is at 13.

So, we can conclude that write operation in append mode always takes place at the end of the file.

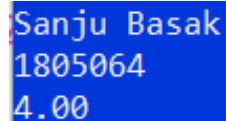
Question - 2: Can append mode be used for binary operation (write)?

Observation:

```
typedef struct {
    char name[20];
    int roll;
    float cgpa;
} S;

int main() {
    FILE *fp;
    S s;
    if ((fp = fopen("file", "a")) == NULL) {
        printf("Cannot open file.\n");
        exit(1);
    }
    gets(s.name);
    scanf("%d %f", &s.roll, &s.cgpa);
    fwrite(&s, sizeof(S), 1, fp);
    fclose(fp);
    return 0;
}
```

Console input:

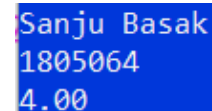


```
Sanju Basak
1805064
4.00
```

```
typedef struct {
    char name[20];
    int roll;
    float cgpa;
} S;

int main() {
    FILE *fp;
    S s;
    if ((fp = fopen("file", "rb")) == NULL) {
        printf("Cannot open file.\n");
        exit(1);
    }
    fread(&s, sizeof(S), 1, fp);
    printf("%s\n%d\n%.2f\n", s.name, s.roll, s.cgpa);
    fclose(fp);
    return 0;
}
```

Console output:



```
Sanju Basak
1805064
4.00
```

We were successful at performing binary operation in append mode. We created a structure containing some information about a student. Then we used `fwrite()` to write that structure into a file while opened in append mode. Since the file did not exist, the file was created. Then to check if that data was stored successfully, we fetched that data using `fread()` in another program while the file being opened in "rb" mode. If it is found that the data was stored successfully. This means that `fwrite()` can be used to write binary data in append mode. But it is not advisable to perform binary operation in append mode as some character translation may occur. So there might not be one to one relation between the actual data and the data written to the file. For example, the newline character is converted into carriage return/line feed pair. So even though binary operation can be performed in append mode, it is not advisable to do so.

Question - 3: Can append mode be used for binary operation (read)?

Observation:

```
int main ()
{
    char s[100], ch;
    FILE *fp = fopen ("file.txt", "a");
    while (feof(fp))
    {
        if (feof(fp))
        {
            break;
        }
        else if (ferror(fp))
        {
            printf("read error occurred!\n");
        }
        fread(s, 100, 1, fp);
    }
    printf("%s", s);

    fclose(fp);

    return 0;
}
```

Console output:



First a file named 'file.txt' was created in the same folder of the code. The file had a string (Hello World!) written in it. Then we opened the file in append mode and tried to read from it with fread() function. However, when we ran the code, we got garbage value on the console. Therefore, we can conclude that binary read operation, fread() does not work in append mode.

Question - 4: Can append mode be used for creating a new file and writing?

Observation:

```
#include<stdio.h>
int main ()
{
    char s[100], ch;
    int x = 10, y = 12, m, n;
    FILE *fp = fopen ("new.txt", "a");

    if (ferror(fp))
    {
        printf("read error occurred!\n");
    }

    fprintf(fp, "%d %d\n", x, y);

    fclose(fp);

    return 0;
}
```

File output:

new.txt - Notepad
File Edit Format View Help
10 12

A file (new.txt) was created and two integers (10 12) were successfully written in it. Therefore, we can conclude that append mode can be operated for both existing file and created file.

Question - 5: Can append mode be used for reading from an existing file?

Observation:

```
int main ()
{
    int m, n;
    FILE *fp = fopen ("new.txt", "a");

    if (ferror(fp))
    {
        printf("read error occurred!\n");
    }
    printf("Cursor position: %ld\n", ftell(fp));
    fscanf(fp, "%d", &m);
    fscanf(fp, "%d", &n);
    printf("%d\n%d\n", m, n);
    printf("Cursor position: %ld\n", ftell(fp));

    fclose(fp);

    return 0;
}
```

Console Output:

```
Cursor position: 0
8
71
Cursor position: 0
```

A file containing two integers was opened for reading from it. However, `fscanf()` failed to fetch the correct data from it. "10 12" was written on the file but the console output printed two garbage values. `ftell()` was used to check the cursor position before and after reading. As for both the case, the cursor was in the same position, reading failed. However, if the same file is opened in "r" mode, `fscanf()` and `ftell()` worked out just fine.