File handle Assignment 06
Author: ThanhTH10

4. Write a program to hide all occurrence of the substr in a file

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

void hideOccurrences(char *line, const char *substr)
{
    char *pos;
    int index = 0;
    int substrLen = strlen(substr);

    char *result = malloc(strlen(line) + 1);
    if (!result)
    {
        perror("malloc");
        exit(EXIT_FAILURE);
    }
    result[0] = '\0';

    while ((pos = strstr(line + index, substr)) != NULL)
    {
        strncat(result, line + index, pos - (line + index));
        strcat(result, "");
        index = (pos - line) + substrLen;
    }
    strcat(result, line + index);
    strcpy(line, result);
    free(result);
}

int main(int argc, char *argv[])
{
    if (argc != 4)
    {
        fprintf(stderr, "Usage: %s <filename> <substr> <newfile>\n", argv[0]);
        return 1;
    }

    const char *filename = argv[1];
    const char *substr = argv[2];
    const char *newfile = argv[3];

    FILE *file = fopen(filename, "r");
    if (file == NULL)
    {
        perror("Error opening file");
        return 1;
    }

    FILE *outputFile = fopen(newfile, "w");
    if (outputFile == NULL)
    {
        perror("Error opening output file");
        fclose(file);
        return 1;
    }
```

```
    char *line = NULL;
    int len = 0;
    int read;
```

```
    while ((read = getline(&line, &len, file)) != -1)
    {
        hideOccurrences(line, substr);
        fputs(line, outputFile);
    }
```

```
    free(line);
    fclose(file);
    fclose(outputFile);
```

```
    printf("All occurrences of '%s' have been hidden in %s and saved to %s.\n", substr,
filename, newfile);
    return 0;
}
// ./program input.txt substr output.txt
```

6. Write a program to Convert the first and last letter of Every Word to Upper Case in a file.

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

#define BUFFER_SIZE 1000
```

```
void convertFirstLastToUpper(FILE *inputFile, FILE *outputFile)
{
    char buffer[BUFFER_SIZE];
    while (fgets(buffer, BUFFER_SIZE, inputFile))
    {
        int len = strlen(buffer);
        int start = -1;
        for (int i = 0; i < len; i++)
        {
            if (isalpha(buffer[i]))
            {
                if (start == -1)
                {
                    start = i;
                }
            }
            else
            {
                if (start != -1)
                {
                    buffer[start] = toupper(buffer[start]);
                    buffer[i - 1] = toupper(buffer[i - 1]);
                    start = -1;
                }
```

```c
            }
        }
        if (start != -1) // Case where the last word might end at the end of the line
        {
            buffer[start] = toupper(buffer[start]);
            buffer[len - 1] = toupper(buffer[len - 1]);
        }
        fputs(buffer, outputFile);
    }
}
```

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

```c
    const char *filename = argv[1];
```

```c
    FILE *inputFile = fopen(filename, "r");
    if (!inputFile)
    {
        perror("Error opening input file");
        return 1;
    }
```

```c
    FILE *outputFile = fopen("output.txt", "w");
    if (!outputFile)
    {
        perror("Error opening output file");
        fclose(inputFile);
        return 1;
    }
```

```c
    convertFirstLastToUpper(inputFile, outputFile);
    printf("Converted !\n");
    fclose(inputFile);
    fclose(outputFile);
```

```c
    return 0;
}
```

7. Write a program to Sort the lines of a file according to the alphabetically increasing order and update back to same file.

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

#define MAX_LINE_LENGTH 1000
```

```c
char **readLines(const char *filename, int *lineCount)
{
    FILE *file = fopen(filename, "r");
    if (!file)
    {
        perror("Error opening file for reading");
        return NULL;
    }
```

```c
    char **lines = NULL;
    int count = 0;
    int capacity = 10;
    lines = malloc(capacity * sizeof(char *));
    if (!lines)
    {
        perror("Memory allocation failed");
        fclose(file);
        return NULL;
    }
```

```c
    char buffer[MAX_LINE_LENGTH];
    while (fgets(buffer, sizeof(buffer), file))
    {
        if (count >= capacity)
        {
            capacity *= 2;
            lines = realloc(lines, capacity * sizeof(char *));
            if (!lines)
            {
                perror("Memory allocation failed");
                fclose(file);
                return NULL;
            }
        }
        lines[count] = strdup(buffer);
        if (!lines[count])
        {
            perror("Memory allocation failed");
            fclose(file);
            return NULL;
        }
        count++;
    }
    fclose(file);
```

```c
    *lineCount = count;
    return lines;
}
```

```c
void sortLines(char **lines, int count)
{
    for (int i = 0; i < count - 1; i++)
    {
        for (int j = i + 1; j < count; j++)
        {
            if (strcmp(lines[i], lines[j]) > 0)
            {
                char *temp = lines[i];
                lines[i] = lines[j];
                lines[j] = temp;
            }
        }
    }
}
```

```c
void writeLines(const char *filename, char **lines, int count)
{
    FILE *file = fopen(filename, "w");
    if (!file)
    {
        perror("Error opening file for writing");
        for (int i = 0; i < count; i++)
        {
            free(lines[i]);
        }
        free(lines);
        return;
    }
```

```
    for (int i = 0; i < count; i++)
    {
        fputs(lines[i], file);
        free(lines[i]);
    }
    free(lines);
    fclose(file);
}
```

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

```
    int lineCount;
    char **lines = readLines(argv[1], &lineCount);
    if (!lines)
    {
        return 1;
    }
```

```
    sortLines(lines, lineCount);
    writeLines(argv[1], lines, lineCount);
```

```
    printf("Lines sorted and updated in %s.\n", argv[1]);
    return 0;
}
```

10. Write a program to delete the lines between a given range in a file

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

#define MAX_LINE_LENGTH 1000
```

```
int main(int argc, char *argv[])
{
    if (argc != 4)
    {
        printf("Usage: %s <filename> <start_line> <end_line>\n", argv[0]);
```

```c
        return 1;
    }
```

```c
    const char *filename = argv[1];
    int start_line = atoi(argv[2]);
    int end_line = atoi(argv[3]);
```

```c
    FILE *file = fopen(filename, "r");
    if (!file)
    {
        perror("Error opening file");
        return 1;
    }
```

```c
    char **lines = NULL;
    int count = 0;
    int capacity = 10;
    lines = malloc(capacity * sizeof(char *));
    if (!lines)
    {
        perror("Memory allocation failed");
        fclose(file);
        return 1;
    }
```

```c
    char buffer[MAX_LINE_LENGTH];
    while (fgets(buffer, sizeof(buffer), file))
    {
        if (count >= capacity)
        {
            capacity *= 2;
            lines = realloc(lines, capacity * sizeof(char *));
            if (!lines)
            {
                perror("Memory allocation failed");
                fclose(file);
                return 1;
            }
        }
        lines[count] = strdup(buffer);
        if (!lines[count])
        {
            perror("Memory allocation failed");
            fclose(file);
            return 1;
        }
        count++;
    }
    fclose(file);
```

```c
    file = fopen(filename, "w");
    if (!file)
    {
        perror("Error opening file for writing");
        for (int i = 0; i < count; i++)
        {
            free(lines[i]);
        }
        free(lines);
        return 1;
    }
    for (int i = 0; i < count; i++)
    {
        if (i + 1 < start_line || i + 1 > end_line)
        {
            fputs(lines[i], file);
        }
        free(lines[i]);
    }
    free(lines);
    fclose(file);

    printf("Lines between %d and %d deleted from %s.\n", start_line, end_line, filename);
    return 0;
}
```

12. Write a program to copy the contents of one file to N no of Destination files.

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

#define BUFFER_SIZE 1024

int main(int argc, char *argv[])
{
    if (argc < 3)
    {
        printf("Usage: %s <source_file> <dest_file_1> [<dest_file_2> ...]\n", argv[0]);
        return 1;
    }

    const char *source_file = argv[1];

    FILE *src_file = fopen(source_file, "r");
    if (!src_file)
    {
        fprintf(stderr, "Error opening source file: %s\n", source_file);
        return 1;
    }

    // Copy
    for (int i = 2; i < argc; i++)
    {
        const char *dest_file = argv[i];
        FILE *dest_file_ptr = fopen(dest_file, "w");
        if (!dest_file_ptr)
        {
            fprintf(stderr, "Error opening destination file: %s\n", dest_file);
            fclose(src_file);
            return 1;
        }

        char buffer[BUFFER_SIZE];
        size_t bytes_read;
        while ((bytes_read = fread(buffer, 1, BUFFER_SIZE, src_file)) > 0)
        {
            if (fwrite(buffer, 1, bytes_read, dest_file_ptr) != bytes_read)
            {
                fprintf(stderr, "Error writing to destination file: %s\n", dest_file);
                fclose(src_file);
                fclose(dest_file_ptr);
                return 1;
            }
        }
```

```
        // Res the file position
        rewind(src_file);
```

```
        fclose(dest_file_ptr);
        printf("Copied contents of %s to %s\n", source_file, dest_file);
    }
```

```
    fclose(src_file);
    return 0;
}
```

```
// ./program input.txt cop1.txt cop2.txt ....
```