Project 2 Unix Utilities

In project 2 we were tasked on creating 4 Unix utilities. These were my-cat.c, my-grep.c, my-zip.c and my-unzip.c. I will go through how I implemented all these utilities in this document. The general process for all of these utilities was the same. I got the basic idea of how it should work from the task description, did some reading on the man pages, which was the only resource needed for this project and then programmed the functionality and finally did the error handling. As a side note I wanted to learn a bit about using the Copilot addon on Visual Studio Code, so I used it to generate the comments of the code I wrote. The comments are based on the code in context that I myself wrote and the comments were checked and edited by me.

my-cat.c

The functionality of my-cat.c is quite simple. The program takes one or more files as arguments and prints the contents to stdout.

```c
// Function to read a file line by line and print each line
void printFile(char *fileName) {
    FILE *file = fopen(fileName, "r");
    char buffer[128];

    // Check if the file was opened successfully
    if (file == NULL) {
        fprintf(stdout, "my-cat: cannot open file '%s'\n", fileName);
        exit(1);
    }

    // Read and print each line from the file
    // If the file is empty, print an error message
    while (fgets(buffer, sizeof(buffer), file) != NULL) {
        fprintf(stdout, "%s", buffer);
    }
    if (ferror(file)) {
        fprintf(stdout, "my-cat: error reading file '%s'\n", fileName);
        fclose(file);
        exit(1);
    }

    fclose(file);
    return;
}
```

Picture 1. printFile function in my-cat.c

How I implemented this is by calling the printFile function shown in picture 1 once for each file name provided as arguments. The function has a buffer of 128 characters for each line. It was said to use the fgets function to implement this but no maximum size for each line was provided so I came up with a limit of 128. This can be easily changed but if we wanted to make the utility be more dynamic by allowing any length of lines to be used we should use for example getline instead of fgets. After the buffer is read we print the line to stdout and then finally close the file.

my-grep.c

The utility my-grep.c works in the following way. You call the utility with a search term and either no files or any number of files. Then the utility tries to find the line

```c
// Function to search for a term in a string and print the string if found
void grep(char *string, char *searchTerm) {
    int searchChar = 0;

    for (int i = 0; i < strlen(string); i++) {
        if (string[i] == searchTerm[searchChar]) {
            searchChar++;
            if (searchChar == strlen(searchTerm)) {
                fprintf(stdout, "%s", string);
                break;
            }
        } else {
            searchChar = 0;
        }
    }
    return;
}
```

Picture 2. grep function of my-grep.c

The utility I programmed works iin the following way. We check how many arguments we have if there is only a search term, we will call the function grep shown in picture 2 once for each line read from stdin using the getline function and stop reading if we reach an empty line. If we have been given a number of file names, we will call the function for each line read from these files. The lines are again read using getline to make it possible to have arbitrarily long lines. If an empty string is provided as a search term no lines are matched. The actual grep function works in the following way. We will try to match the first character in the search term to a character in the line read in order from first to last. If it is found, we will try to match the next character in the search term to the line read starting from the last match. If no match is found, we go back to the first character. If all characters were matched in order the line is printed to stdout.

my-zip.c and my-unzip.c

The utilities my-zip.c and my-unzip.c are to well zip and unzip a file. The zip uses RLE (Run Length Encoding) to compress the file. This makes inputs such as aaaabbbcc into 4a3b2c foor example. both of these can take any number of files but no files is an error. They also both write too stdout. The utility my-zip.c would then use shell redirection.

```c
/* Use RLE (Run Length Encoding) to compress the file
 and write the compressed data to stdout*/
void compress(char *fileName) {
    FILE *fp = fopen(fileName, "r");
    char *buffer = NULL;
    size_t size = 0;
    __ssize_t read;

    // Check if the file was opened successfully
    if (fp == NULL) {
        fprintf(stdout, "my-zip: cannot open file '%s'\n", fileName);
        exit(1);
    }

    while ((read = getline(&buffer, &size, fp)) != -1) {
        char currentChar = buffer[0];
        int count = 1;

        for (int i = 1; i <= strlen(buffer); i++) {
            if(buffer[i] == currentChar) {
                count++;
            } else {
                fwrite(&count, sizeof(int), 1, stdout);
                fwrite(&currentChar, sizeof(char), 1, stdout);
                count = 1;
                currentChar = buffer[i];
            }
        }
    }

    free(buffer);
    fclose(fp);
    return;
}
```

Picture 3. compress function of my-zip.c

The utility my-zip.c takes a number larger than one of files as arguments and calls the function compress on each of them shown in picture 3. The compress function then opens the file and reads each line one at a time. Each line will then be compressed. The compression checks the first character and if the next character is the same the count goes up. If it is a different character, it will then be assigned as currentCharacter and any checks will be done against it and the previous character as well as the count will be printed to stdout using fwrite. This continues until the entire file is checked.

```c
// decompress the file using RLE (Run Length Encoding)
// and print the decompressed data to stdout
void deCompress(char *fileName) {
    FILE *fp = fopen(fileName, "r");
    char currentChar;
    int count;

    // Check if the file was opened successfully
    if (fp == NULL) {
        fprintf(stdout, "my-unzip: cannot open file '%s'\n", fileName);
        exit(1);
    }

    while (1) {
        if ((fread(&count, sizeof(int), 1, fp) != 1) || (fread(&currentChar, sizeof(char), 1, fp) != 1)) {
            if (feof(fp)) {
                // End of file reached
                break;
            } else {
                // Error reading the file
                printf("my-unzip: error reading file '%s'\n", fileName);
                fclose(fp);
                exit(1);
            }
        }

        // Print the current character 'count' times
        for (int i = 0; i < count; i++){
            printf("%c", currentChar);
        }
    }
    printf("\n");
    fclose(fp);
    return;
}
```

Picture 4. decompress function of my-unzip.c

The utility my-unzip.c basically does the exact opposite of my-zip.c. It will take a number of files that is larger than one, no files is an error. Then for each file the decompress function is called. The decompress function uses fread and reads one integer value and one character at a time until the end of file is reached. Then the current character is printed count times to stdout.