Lab assignment 6

210011 Dodajon Xusnitdinov, 210033 Muhammadjon Islomov, 210072 Suxrob Sotiboldiyev

Task 1

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

void process_option(const char *option) {
        if (strcmp(option, "-help") == 0) {
        printf("Help message: You need to provide at least 11 arguments.\n");
        };
        } else {
        printf("Unknown option: %s\n", option);
        }
}

int main(int argc, char *argv[]) {
        if (argc != 11) {
        printf("Usage: %s <arg1> <arg2> ... <arg10>\n", argv[0]);
        return 1;
        }

        for (int i = 1; i < argc; i++) {
        if (argv[i][0] == '-') {
        printf("Option: %s\n", argv[i] + 1); // Print option without the '-'
        process_option(argv[i]);
        } else {
        printf("Argument: %s\n", argv[i]);
        }
        }

        return 0;
}
```

Task 2

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

int main(int argc, char *argv[]) {
        int opt;
        int n_flag = 0, m_flag = 0;
        int N = 0, M = 0;

        // Parse command line options using getopt
        while ((opt = getopt(argc, argv, "n:m:")) != -1) {
        switch (opt) {
```

```
            case 'n':
            n_flag = 1;
            N = atoi(optarg);
            break;
            case 'm':
            m_flag = 1;
            M = atoi(optarg);
            break;
            default: /* '?' */
            fprintf(stderr, "Usage: %s -n <N> -m <M>\n", argv[0]);
            exit(EXIT_FAILURE);
            }
            }

            // Check if both -n and -m flags are provided
            if (!n_flag || !m_flag) {
            fprintf(stderr, "Both flags -n and -m are required.\n");
            exit(EXIT_FAILURE);
            }

            // Calculate the sum of the first N natural numbers
            int sum = 0;
            for (int i = 1; i <= N; i++) {
            sum += i;
            }

            // Calculate the product of the first M natural numbers
            int product = 1;
            for (int i = 1; i <= M; i++) {
            product *= i;
            }

            // Print the sum and product
            printf("Sum of the first %d natural numbers: %d\n", N, sum);
            printf("Product of the first %d natural numbers: %d\n", M, product);

            return 0;
}
```

Task 3

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

void addition(int N) {
```

```
        int sum = 0;
        for (int i = 1; i <= N; i++) {
        sum += i;
        }
        printf("Sum of first %d natural numbers: %d\n", N, sum);
}

void product(int M) {
        int prod = 1;
        for (int i = 1; i <= M; i++) {
        prod *= i;
        }
        printf("Product of first %d natural numbers: %d\n", M, prod);
}

int main(int argc, char *argv[]) {
        int opt;
        int n_flag = 0, m_flag = 0;
        int N = 0, M = 0;

        // Parse command line options using getopt
        while ((opt = getopt(argc, argv, "n:m:")) != -1) {
        switch (opt) {
        case 'n':
        n_flag = 1;
        N = atoi(optarg);
        break;
        case 'm':
        m_flag = 1;
        M = atoi(optarg);

        break;
        default: /* '?' */
        fprintf(stderr, "Usage: %s -n <N> -m <M>\n", argv[0]);
        exit(EXIT_FAILURE);
        }
        }

        // Check if both -n and -m flags are provided
        if (!n_flag || !m_flag) {
        fprintf(stderr, "Both -n and -m flags are required.\n");
        exit(EXIT_FAILURE);
        }

        // Fork two child processes
        pid_t pid1 = fork();
        if (pid1 == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
        } else if (pid1 == 0) {
        // Child process for addition
        addition(N);
```

```
            exit(EXIT_SUCCESS);
        }

        // Fork the second child process
        pid_t pid2 = fork();
        if (pid2 == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
        } else if (pid2 == 0) {
        // Child process for product
        product(M);
        exit(EXIT_SUCCESS);
        }

        // Wait for both child processes to finish
        int status;
        waitpid(pid1, &status, 0);
        waitpid(pid2, &status, 0);

        return 0;
}
```