Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

New York University
Tandon School of Engineering

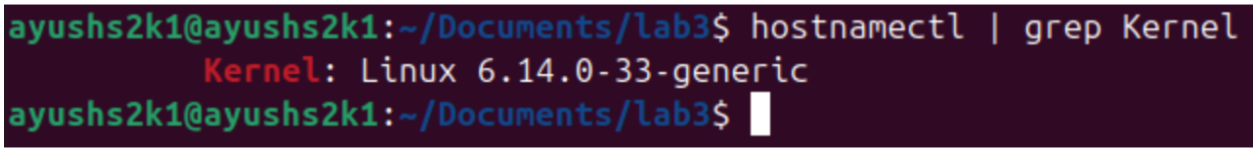
Department of Computer Science and Engineering

Introduction to Operating Systems

Fall 2025

Assignment 3

Kernel Version: Linux 6.14.0-33-generic

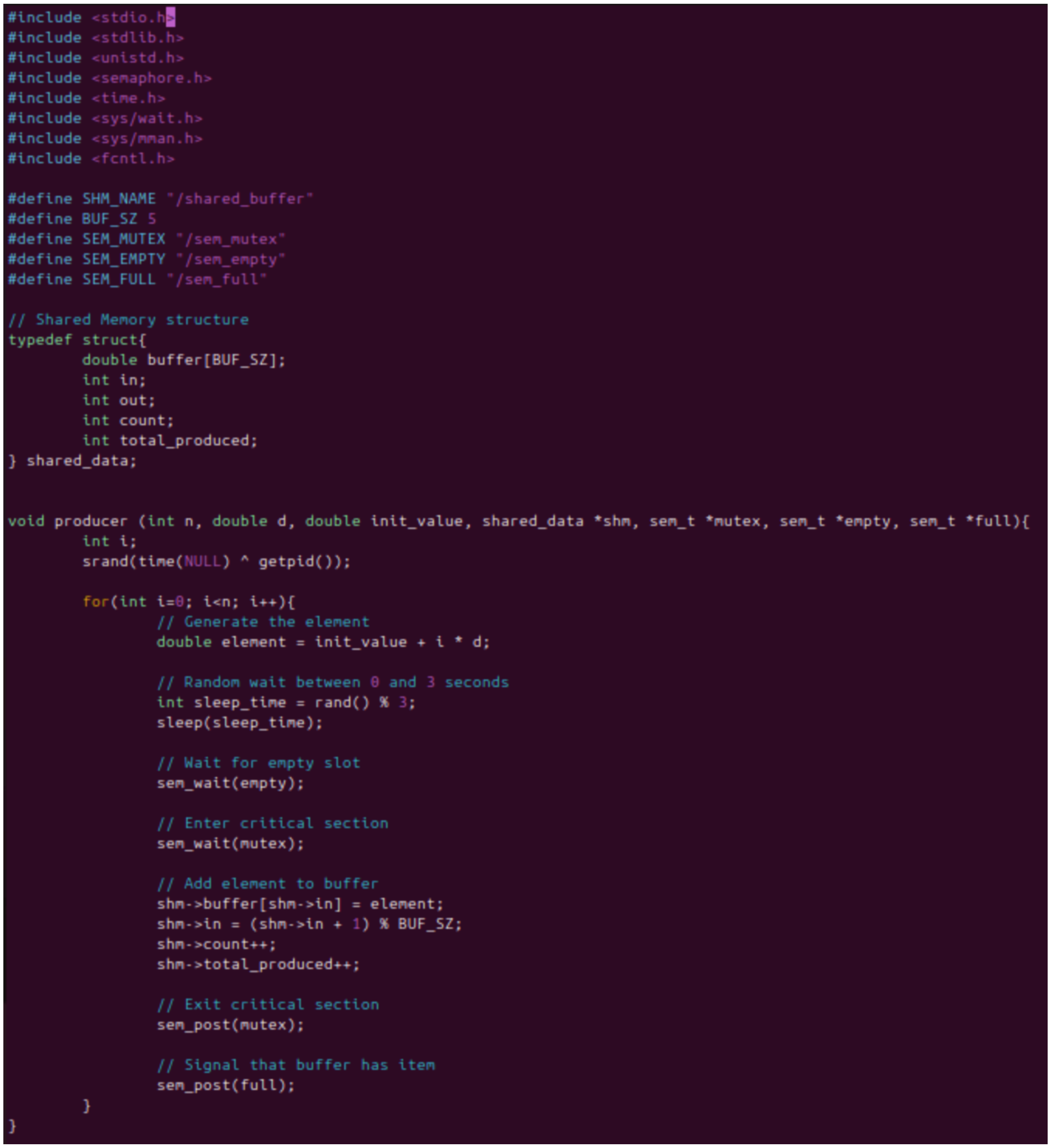


```
ayushs2k1@ayushs2k1:~/Documents/lab3$ hostnamectl | grep Kernel
          Kernel: Linux 6.14.0-33-generic
ayushs2k1@ayushs2k1:~/Documents/lab3$ 
```

Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

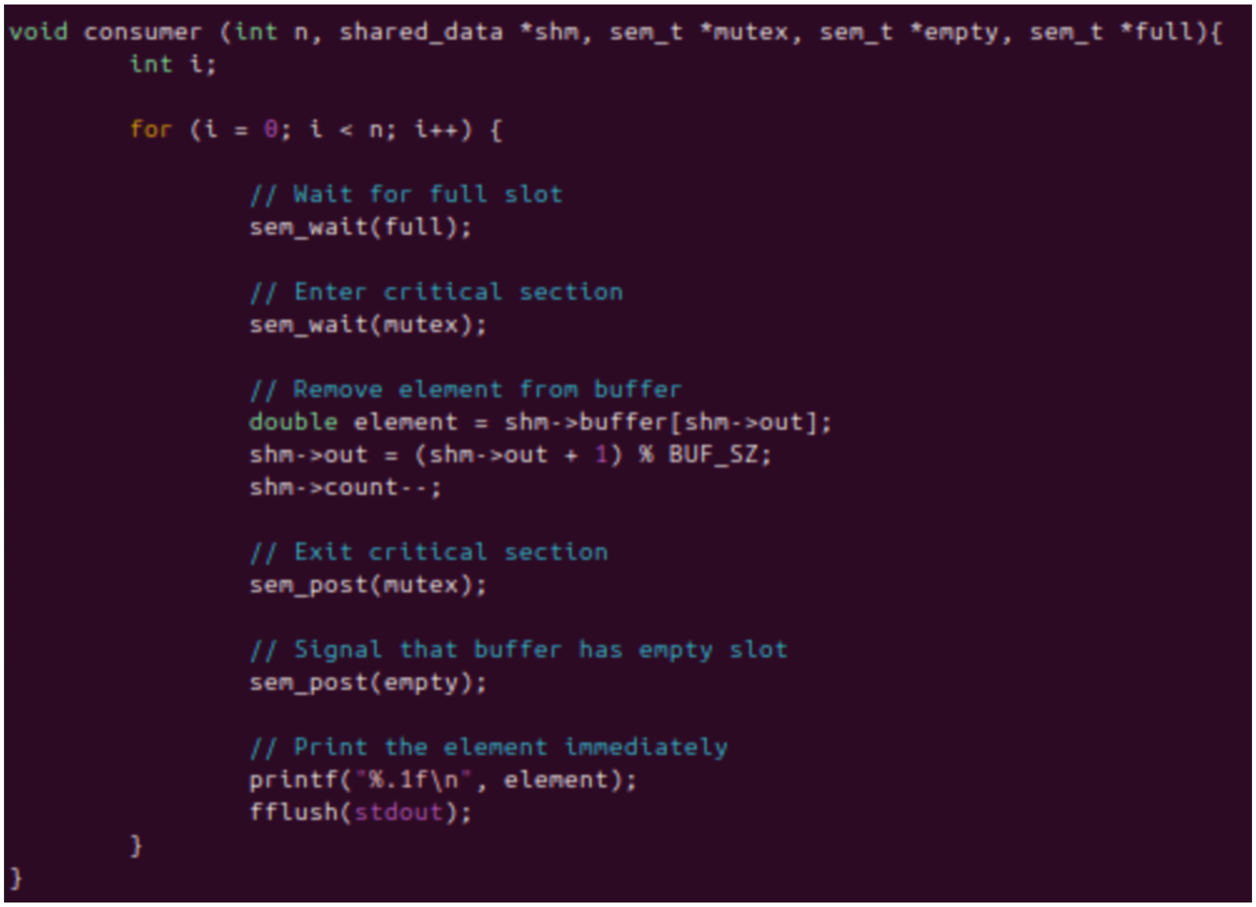
PART A

Code:



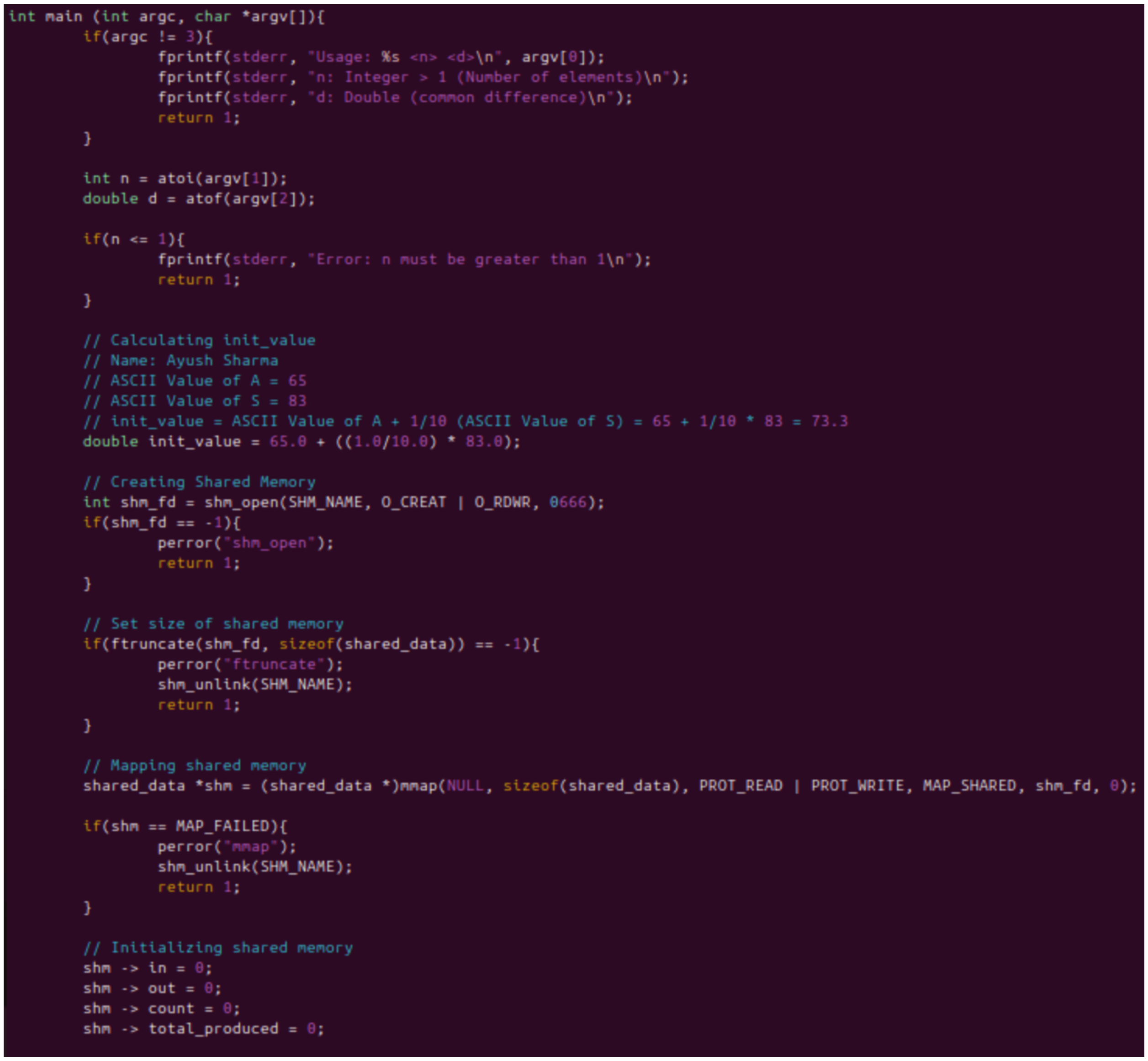
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <semaphore.h>
#include <time.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <fcntl.h>

#define SHM_NAME "/shared_buffer"
#define BUF_SZ 5
#define SEM_MUTEX "/sem_mutex"
#define SEM_EMPTY "/sem_empty"
#define SEM_FULL "/sem_full"

// Shared Memory structure
typedef struct{
        double buffer[BUF_SZ];
        int in;
        int out;
        int count;
        int total_produced;
} shared_data;


void producer (int n, double d, double init_value, shared_data *shm, sem_t *mutex, sem_t *empty, sem_t *full){
        int i;
        srand(time(NULL) ^ getpid());

        for(int i=0; i<n; i++){
                // Generate the element
                double element = init_value + i * d;

                // Random wait between 0 and 3 seconds
                int sleep_time = rand() % 3;
                sleep(sleep_time);

                // Wait for empty slot
                sem_wait(empty);

                // Enter critical section
                sem_wait(mutex);

                // Add element to buffer
                shm->buffer[shm->in] = element;
                shm->in = (shm->in + 1) % BUF_SZ;
                shm->count++;
                shm->total_produced++;

                // Exit critical section
                sem_post(mutex);

                // Signal that buffer has item
                sem_post(full);
        }
}
```

Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108



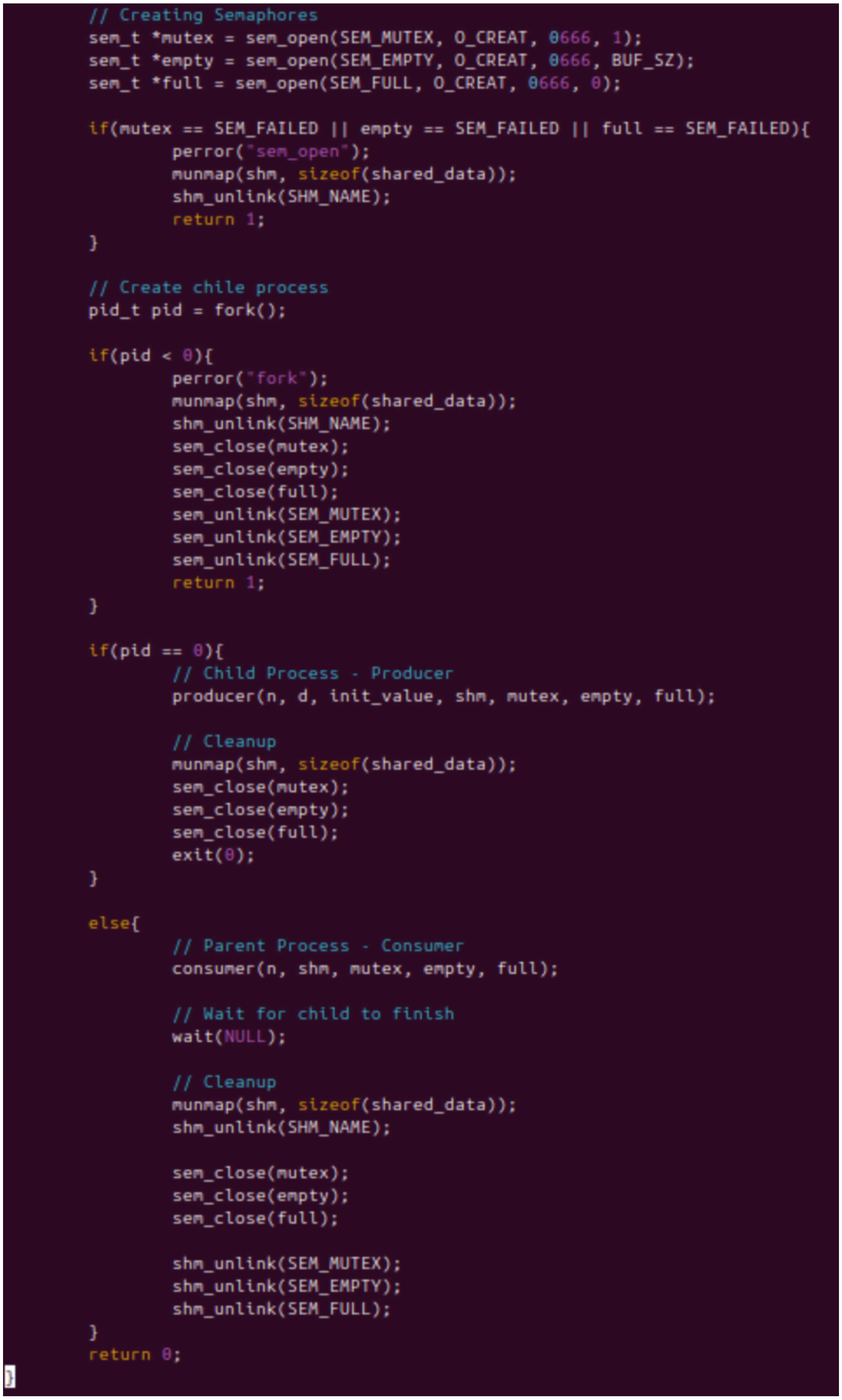
```c
void consumer (int n, shared_data *shm, sem_t *mutex, sem_t *empty, sem_t *full){
        int i;

        for (i = 0; i < n; i++) {

                // Wait for full slot
                sem_wait(full);

                // Enter critical section
                sem_wait(mutex);

                // Remove element from buffer
                double element = shm->buffer[shm->out];
                shm->out = (shm->out + 1) % BUF_SZ;
                shm->count--;

                // Exit critical section
                sem_post(mutex);

                // Signal that buffer has empty slot
                sem_post(empty);

                // Print the element immediately
                printf("%.1f\n", element);
                fflush(stdout);
        }
}
```

Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108



```
int main (int argc, char *argv[]){
        if(argc != 3){
                fprintf(stderr, "Usage: %s <n> <d>\n", argv[0]);
                fprintf(stderr, "n: Integer > 1 (Number of elements)\n");
                fprintf(stderr, "d: Double (common difference)\n");
                return 1;
        }

        int n = atoi(argv[1]);
        double d = atof(argv[2]);

        if(n <= 1){
                fprintf(stderr, "Error: n must be greater than 1\n");
                return 1;
        }

        // Calculating init_value
        // Name: Ayush Sharma
        // ASCII Value of A = 65
        // ASCII Value of S = 83
        // init_value = ASCII Value of A + 1/10 (ASCII Value of S) = 65 + 1/10 * 83 = 73.3
        double init_value = 65.0 + ((1.0/10.0) * 83.0);

        // Creating Shared Memory
        int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
        if(shm_fd == -1){
                perror("shm_open");
                return 1;
        }

        // Set size of shared memory
        if(ftruncate(shm_fd, sizeof(shared_data)) == -1){
                perror("ftruncate");
                shm_unlink(SHM_NAME);
                return 1;
        }

        // Mapping shared memory
        shared_data *shm = (shared_data *)mmap(NULL, sizeof(shared_data), PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);

        if(shm == MAP_FAILED){
                perror("mmap");
                shm_unlink(SHM_NAME);
                return 1;
        }

        // Initializing shared memory
        shm -> in = 0;
        shm -> out = 0;
        shm -> count = 0;
        shm -> total_produced = 0;
```

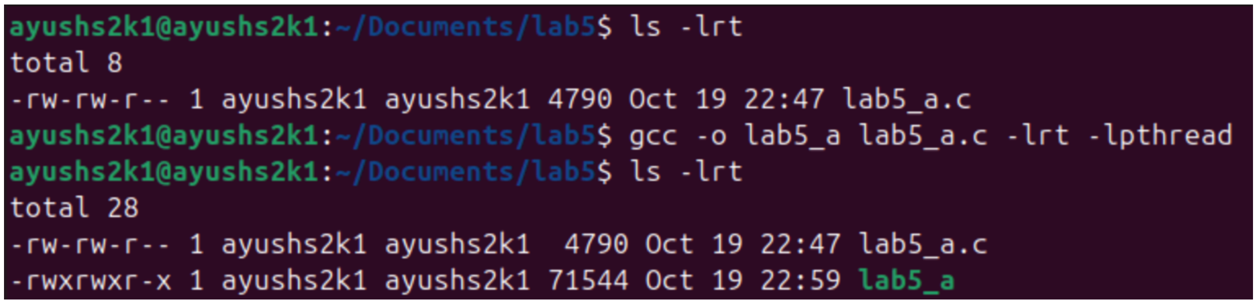
Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108



```
        // Creating Semaphores
        sem_t *mutex = sem_open(SEM_MUTEX, O_CREAT, 0666, 1);
        sem_t *empty = sem_open(SEM_EMPTY, O_CREAT, 0666, BUF_SZ);
        sem_t *full = sem_open(SEM_FULL, O_CREAT, 0666, 0);

        if(mutex == SEM_FAILED || empty == SEM_FAILED || full == SEM_FAILED){
                perror("sem_open");
                munmap(shm, sizeof(shared_data));
                shm_unlink(SHM_NAME);
                return 1;
        }

        // Create chile process
        pid_t pid = fork();

        if(pid < 0){
                perror("fork");
                munmap(shm, sizeof(shared_data));
                shm_unlink(SHM_NAME);
                sem_close(mutex);
                sem_close(empty);
                sem_close(full);
                sem_unlink(SEM_MUTEX);
                sem_unlink(SEM_EMPTY);
                sem_unlink(SEM_FULL);
                return 1;
        }

        if(pid == 0){
                // Child Process - Producer
                producer(n, d, init_value, shm, mutex, empty, full);

                // Cleanup
                munmap(shm, sizeof(shared_data));
                sem_close(mutex);
                sem_close(empty);
                sem_close(full);
                exit(0);
        }

        else{
                // Parent Process - Consumer
                consumer(n, shm, mutex, empty, full);

                // Wait for child to finish
                wait(NULL);

                // Cleanup
                munmap(shm, sizeof(shared_data));
                shm_unlink(SHM_NAME);

                sem_close(mutex);
                sem_close(empty);
                sem_close(full);

                shm_unlink(SEM_MUTEX);
                shm_unlink(SEM_EMPTY);
                shm_unlink(SEM_FULL);
        }
        return 0;
}
```

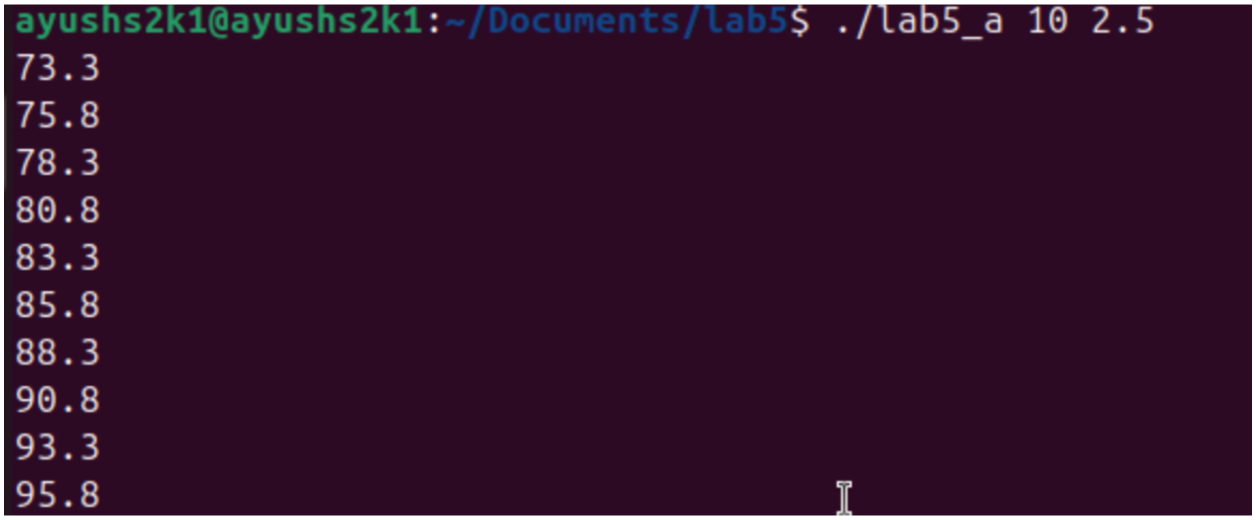
Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

Command used to compile the program: gcc -o lab5_a lab5_a.c -lrt -lpthread
After compiling, it creates an executable file, lab5_a, as shown below.



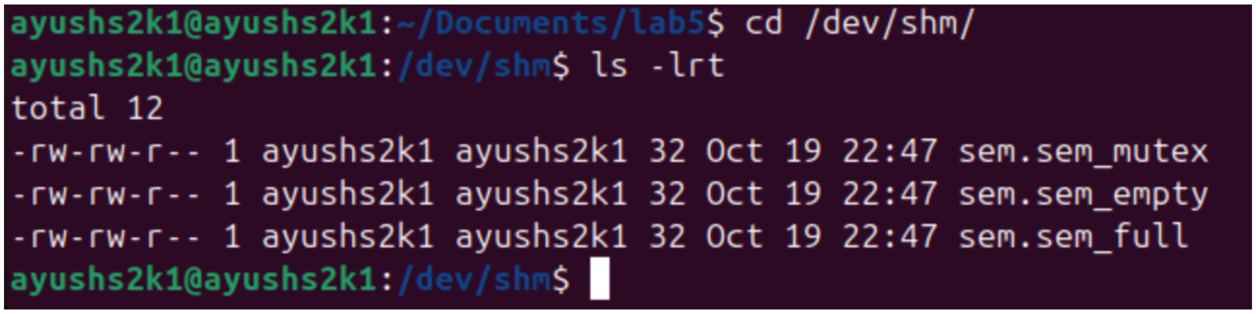
```
ayushs2k1@ayushs2k1:~/Documents/lab5$ ls -lrt
total 8
-rw-rw-r-- 1 ayushs2k1 ayushs2k1 4790 Oct 19 22:47 lab5_a.c
ayushs2k1@ayushs2k1:~/Documents/lab5$ gcc -o lab5_a lab5_a.c -lrt -lpthread
ayushs2k1@ayushs2k1:~/Documents/lab5$ ls -lrt
total 28
-rw-rw-r-- 1 ayushs2k1 ayushs2k1  4790 Oct 19 22:47 lab5_a.c
-rwxrwxr-x 1 ayushs2k1 ayushs2k1 71544 Oct 19 22:59 lab5_a
```

Running the executable:



```
ayushs2k1@ayushs2k1:~/Documents/lab5$ ./lab5_a 10 2.5
73.3
75.8
78.3
80.8
83.3
85.8
88.3
90.8
93.3
95.8
```

Named semaphores appear under /dev/shm as sem.*

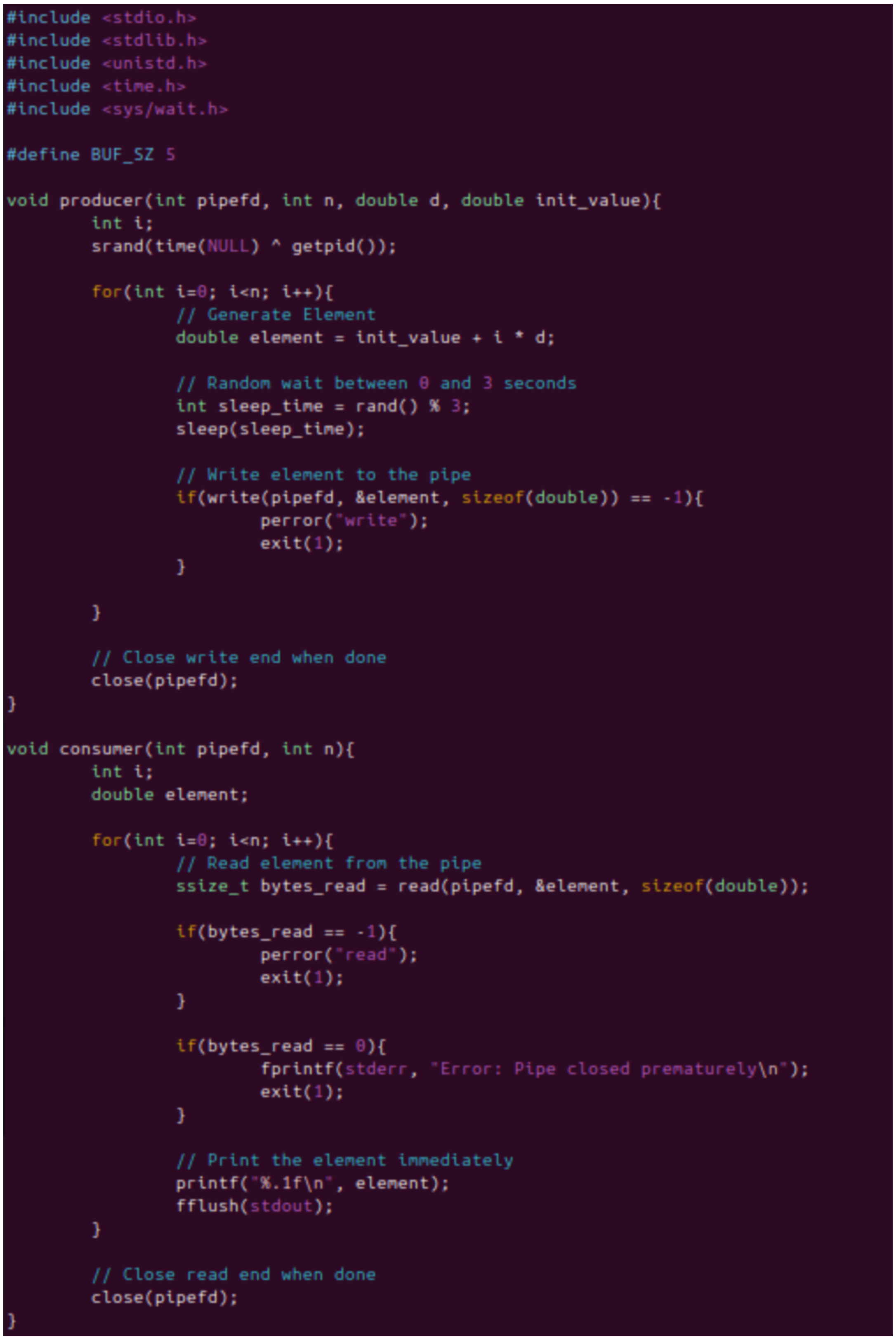


```
ayushs2k1@ayushs2k1:~/Documents/lab5$ cd /dev/shm/
ayushs2k1@ayushs2k1:/dev/shm$ ls -lrt
total 12
-rw-rw-r-- 1 ayushs2k1 ayushs2k1 32 Oct 19 22:47 sem.sem_mutex
-rw-rw-r-- 1 ayushs2k1 ayushs2k1 32 Oct 19 22:47 sem.sem_empty
-rw-rw-r-- 1 ayushs2k1 ayushs2k1 32 Oct 19 22:47 sem.sem_full
ayushs2k1@ayushs2k1:/dev/shm$
```

Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

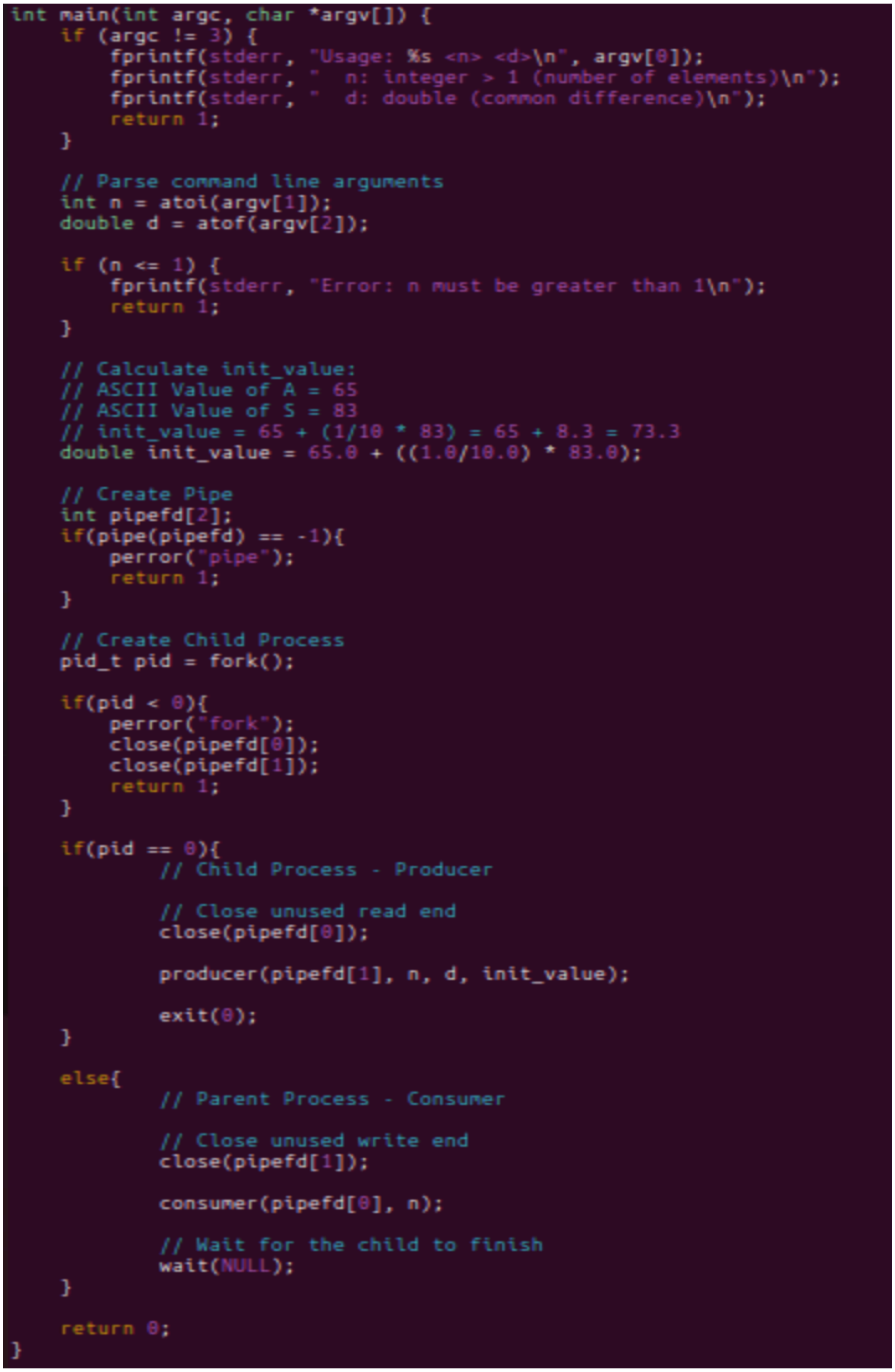
PART B

Code:



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

#define BUF_SZ 5

void producer(int pipefd, int n, double d, double init_value){
        int i;
        srand(time(NULL) ^ getpid());

        for(int i=0; i<n; i++){
                // Generate Element
                double element = init_value + i * d;

                // Random wait between 0 and 3 seconds
                int sleep_time = rand() % 3;
                sleep(sleep_time);

                // Write element to the pipe
                if(write(pipefd, &element, sizeof(double)) == -1){
                        perror("write");
                        exit(1);
                }

        }

        // Close write end when done
        close(pipefd);
}

void consumer(int pipefd, int n){
        int i;
        double element;

        for(int i=0; i<n; i++){
                // Read element from the pipe
                ssize_t bytes_read = read(pipefd, &element, sizeof(double));

                if(bytes_read == -1){
                        perror("read");
                        exit(1);
                }

                if(bytes_read == 0){
                        fprintf(stderr, "Error: Pipe closed prematurely\n");
                        exit(1);
                }

                // Print the element immediately
                printf("%.1f\n", element);
                fflush(stdout);
        }

        // Close read end when done
        close(pipefd);
}
```

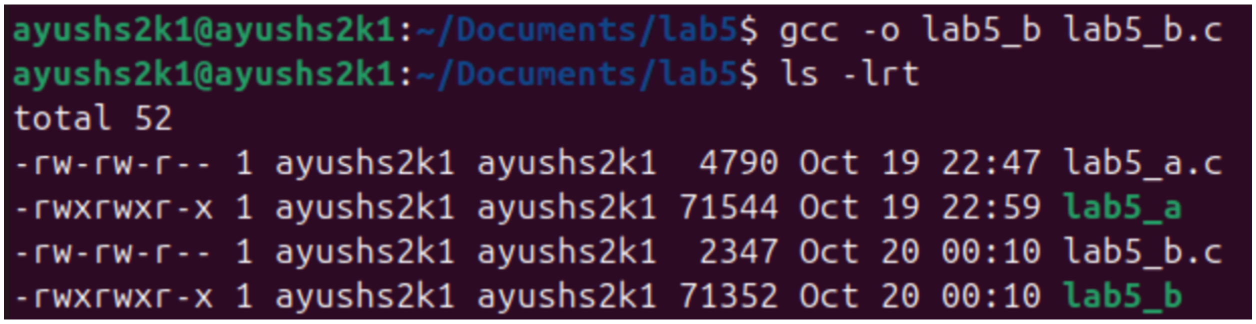
Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108



```
int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <n> <d>\n", argv[0]);
        fprintf(stderr, "  n: integer > 1 (number of elements)\n");
        fprintf(stderr, "  d: double (common difference)\n");
        return 1;
    }

    // Parse command line arguments
    int n = atoi(argv[1]);
    double d = atof(argv[2]);

    if (n <= 1) {
        fprintf(stderr, "Error: n must be greater than 1\n");
        return 1;
    }

    // Calculate init_value:
    // ASCII Value of A = 65
    // ASCII Value of S = 83
    // init_value = 65 + (1/10 * 83) = 65 + 8.3 = 73.3
    double init_value = 65.0 + ((1.0/10.0) * 83.0);

    // Create Pipe
    int pipefd[2];
    if(pipe(pipefd) == -1){
        perror("pipe");
        return 1;
    }

    // Create Child Process
    pid_t pid = fork();

    if(pid < 0){
        perror("fork");
        close(pipefd[0]);
        close(pipefd[1]);
        return 1;
    }

    if(pid == 0){
            // Child Process - Producer

            // Close unused read end
            close(pipefd[0]);

            producer(pipefd[1], n, d, init_value);

            exit(0);
    }

    else{
            // Parent Process - Consumer

            // Close unused write end
            close(pipefd[1]);

            consumer(pipefd[0], n);

            // Wait for the child to finish
            wait(NULL);
    }

    return 0;
}
```

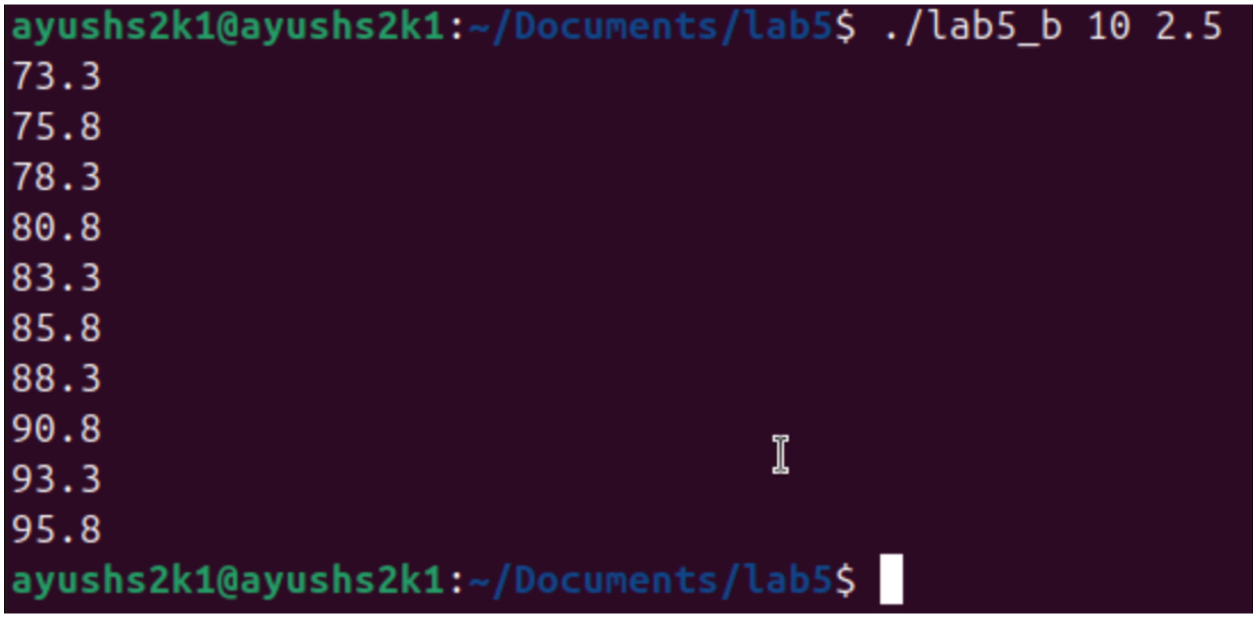
Name: Ayush Sharma NYU ID Number: N15532582 Net ID: as21108

Command used to compile the program: gcc -o lab5_b lab5_b.c
After compiling, it creates an executable file, lab5_b, as shown below.



```
ayushs2k1@ayushs2k1:~/Documents/lab5$ gcc -o lab5_b lab5_b.c
ayushs2k1@ayushs2k1:~/Documents/lab5$ ls -lrt
total 52
-rw-rw-r-- 1 ayushs2k1 ayushs2k1  4790 Oct 19 22:47 lab5_a.c
-rwxrwxr-x 1 ayushs2k1 ayushs2k1 71544 Oct 19 22:59 lab5_a
-rw-rw-r-- 1 ayushs2k1 ayushs2k1  2347 Oct 20 00:10 lab5_b.c
-rwxrwxr-x 1 ayushs2k1 ayushs2k1 71352 Oct 20 00:10 lab5_b
```

Running the executable:



```
ayushs2k1@ayushs2k1:~/Documents/lab5$ ./lab5_b 10 2.5
73.3
75.8
78.3
80.8
83.3
85.8
88.3
90.8
93.3
95.8
ayushs2k1@ayushs2k1:~/Documents/lab5$
```

-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-**END**-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x