21.Develop a C program to implement worst fit algorithm of memory management.

#include <stdio.h>

#define MAX 25

void worstFit(int blockSize[], int m, int processSize[], int n) {

int allocation[n];

// Initially no block is assigned to any process

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

allocation[i] = -1;

// Pick each process and find the worst fit block for it

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

int worstIdx = -1;

for (int j = 0; j < m; j++) {

if (blockSize[j] >= processSize[i]) {

if (worstIdx == -1 || blockSize[j] > blockSize[worstIdx])

worstIdx = j;

}

}

// If we found a block for the process

if (worstIdx != -1) {

// Allocate block j to p[i] process

allocation[i] = worstIdx;

// Reduce available memory in this block

blockSize[worstIdx] -= processSize[i];

}

}

printf("\nProcess No.\tProcess Size\tBlock No.\n");

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

printf(" %d\t\t%d\t\t", i + 1, processSize[i]);

if (allocation[i] != -1)

printf("%d\n", allocation[i] + 1); // Block numbers start from 1

else

printf("Not Allocated\n");

}

}

int main() {

int blockSize[MAX], processSize[MAX], m, n;

printf("Enter number of memory blocks: ");

scanf("%d", &m);

printf("Enter sizes of %d memory blocks:\n", m);

for (int i = 0; i < m; i++)

scanf("%d", &blockSize[i]);

printf("Enter number of processes: ");

scanf("%d", &n);

printf("Enter sizes of %d processes:\n", n);

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

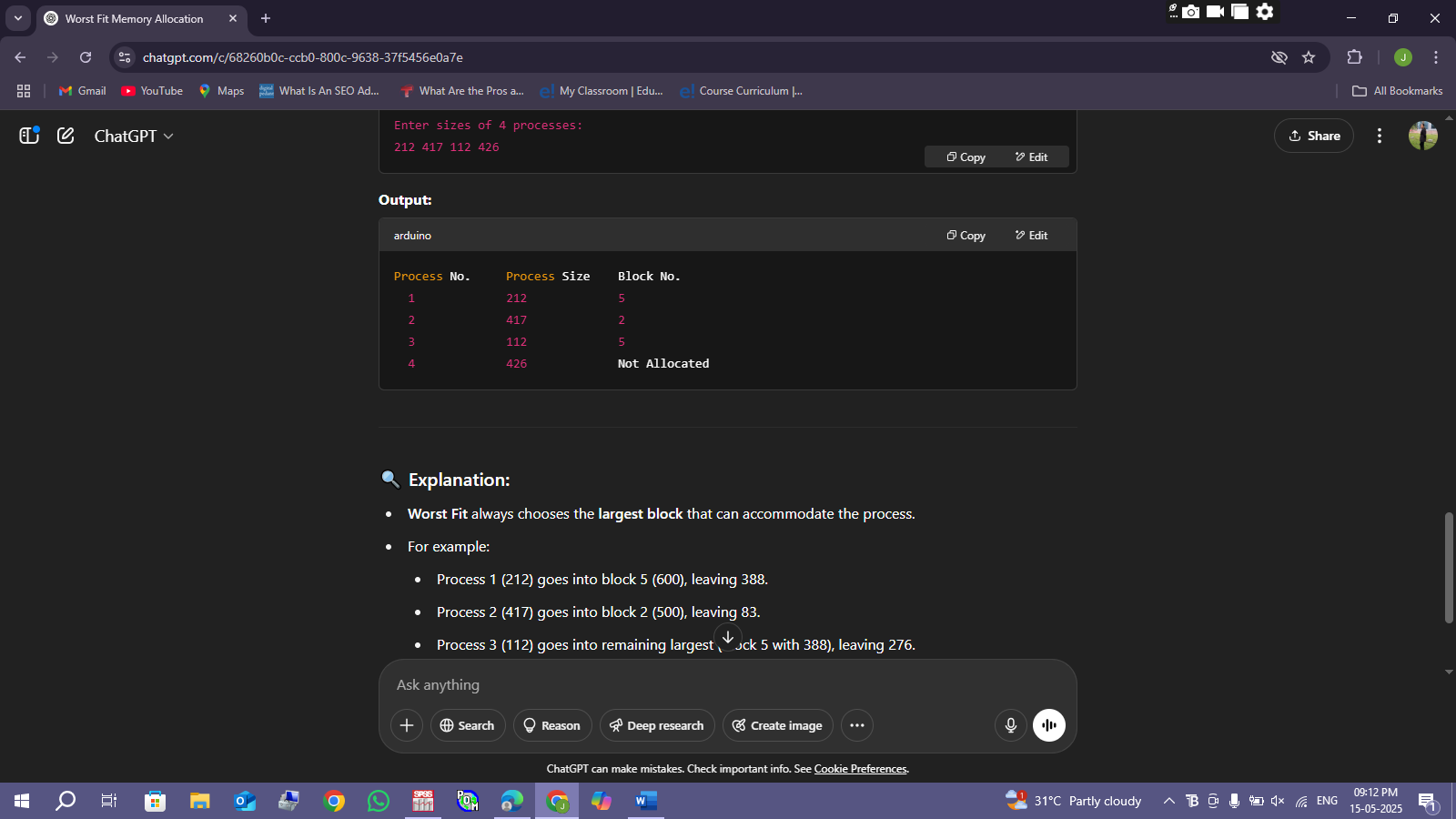
scanf("%d", &processSize[i]);

worstFit(blockSize, m, processSize, n);

return 0;

}

OUTPUT:



22.Construct a C program to implement best fit algorithm of memory management.

#include <stdio.h>

#define MAX 25

void bestFit(int blockSize[], int m, int processSize[], int n) {

int allocation[n];

// Initially no block is assigned to any process

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

allocation[i] = -1;

// Pick each process and find the best fit block for it

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

int bestIdx = -1;

for (int j = 0; j < m; j++) {

if (blockSize[j] >= processSize[i]) {

if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx])

bestIdx = j;

}

}

// If we found a block for the process

if (bestIdx != -1) {

allocation[i] = bestIdx;

blockSize[bestIdx] -= processSize[i];

}

}

// Display allocation result

printf("\nProcess No.\tProcess Size\tBlock No.\n");

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

printf(" %d\t\t%d\t\t", i + 1, processSize[i]);

if (allocation[i] != -1)

printf("%d\n", allocation[i] + 1); // Block numbers start from 1

else

printf("Not Allocated\n");

}

}

int main() {

int blockSize[MAX], processSize[MAX], m, n;

printf("Enter number of memory blocks: ");

scanf("%d", &m);

printf("Enter sizes of %d memory blocks:\n", m);

for (int i = 0; i < m; i++)

scanf("%d", &blockSize[i]);

printf("Enter number of processes: ");

scanf("%d", &n);

printf("Enter sizes of %d processes:\n", n);

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

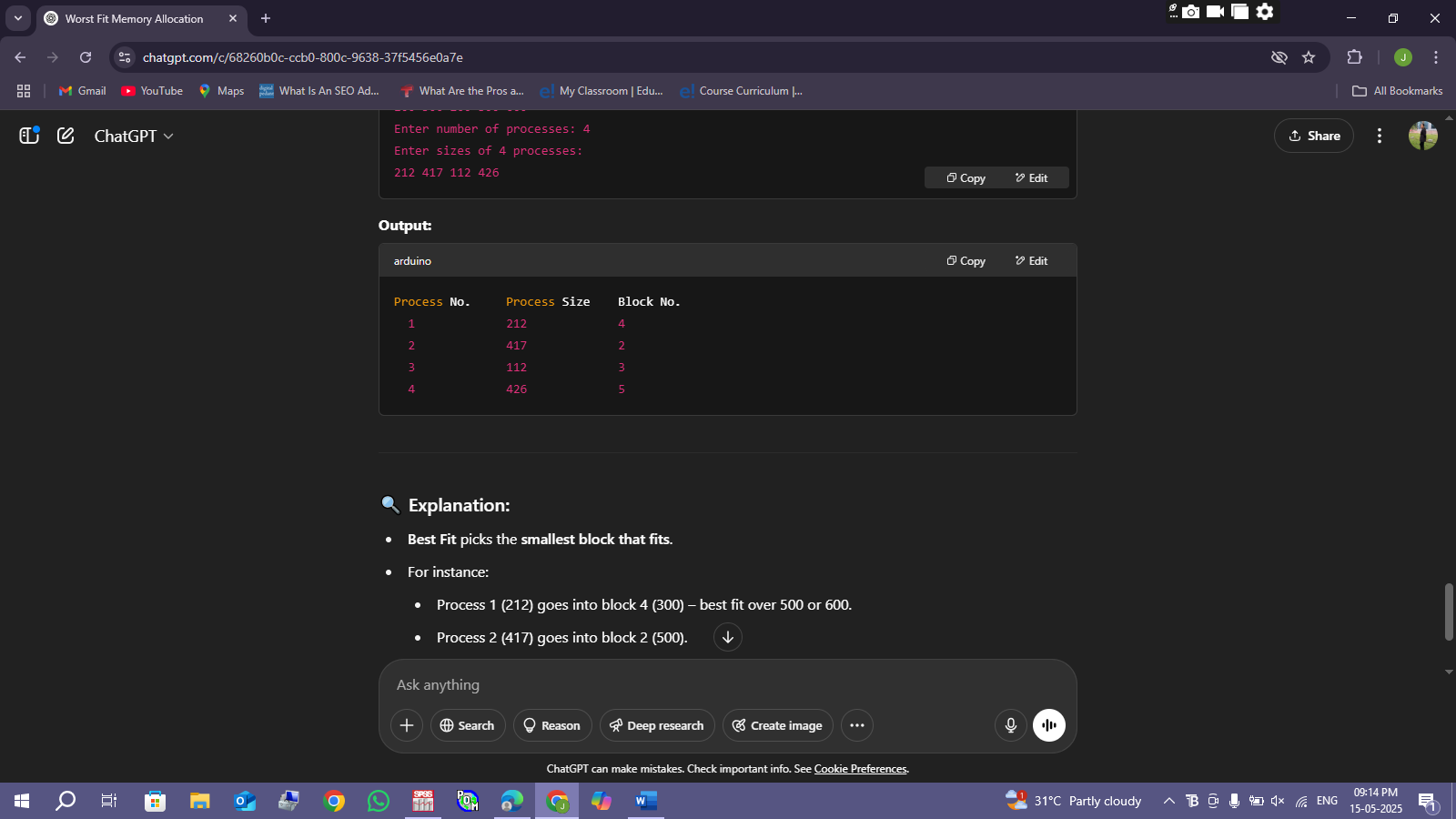
scanf("%d", &processSize[i]);

bestFit(blockSize, m, processSize, n);

return 0;

}

OUTPUT:



23.Construct a C program to implement first fit algorithm of memory management.

#include <stdio.h>

#define MAX 25

void firstFit(int blockSize[], int m, int processSize[], int n) {

int allocation[n];

// Initially no block is assigned to any process

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

allocation[i] = -1;

// Pick each process and find the first suitable block

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

for (int j = 0; j < m; j++) {

if (blockSize[j] >= processSize[i]) {

// Allocate block j to process i

allocation[i] = j;

blockSize[j] -= processSize[i];

break;

}

}

}

// Display allocation result

printf("\nProcess No.\tProcess Size\tBlock No.\n");

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

printf(" %d\t\t%d\t\t", i + 1, processSize[i]);

if (allocation[i] != -1)

printf("%d\n", allocation[i] + 1); // Block numbers start from 1

else

printf("Not Allocated\n");

}

}

int main() {

int blockSize[MAX], processSize[MAX], m, n;

printf("Enter number of memory blocks: ");

scanf("%d", &m);

printf("Enter sizes of %d memory blocks:\n", m);

for (int i = 0; i < m; i++)

scanf("%d", &blockSize[i]);

printf("Enter number of processes: ");

scanf("%d", &n);

printf("Enter sizes of %d processes:\n", n);

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

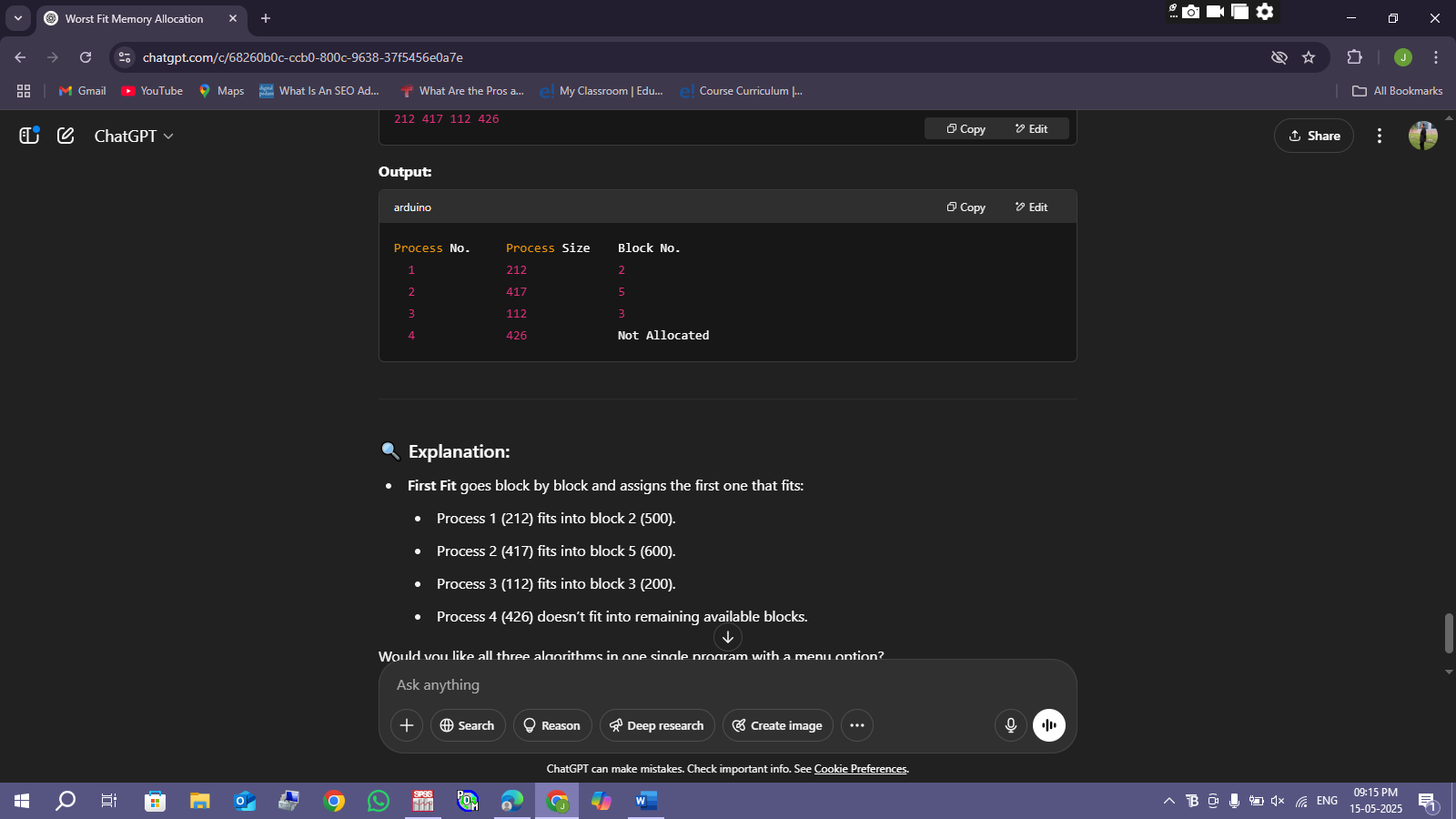
scanf("%d", &processSize[i]);

firstFit(blockSize, m, processSize, n);

return 0;

}

OUTPUT:



24.Design a C program to demonstrate UNIX system calls for file management.

#include <stdio.h>

#include <fcntl.h> // For open() and O\_\* constants

#include <unistd.h> // For read(), write(), lseek(), close()

#include <string.h>

int main() {

int fd;

char buffer[100];

// 1. Create and open a file (with write permissions)

fd = open("demo.txt", O\_CREAT | O\_WRONLY, 0644);

if (fd < 0) {

perror("Error creating file");

return 1;

}

// 2. Write data to the file

char data[] = "Hello, this is a demo of UNIX file system calls.\n";

write(fd, data, strlen(data));

close(fd); // Close after writing

// 3. Reopen file in read-only mode

fd = open("demo.txt", O\_RDONLY);

if (fd < 0) {

perror("Error opening file for reading");

return 1;

}

// 4. Read from the file

int bytesRead = read(fd, buffer, sizeof(buffer) - 1);

if (bytesRead < 0) {

perror("Error reading file");

return 1;

}

buffer[bytesRead] = '\0'; // Null terminate

printf("Content of 'demo.txt':\n%s", buffer);

// 5. Seek to beginning and read again (demonstrate lseek)

lseek(fd, 0, SEEK\_SET); // Move file pointer to beginning

bytesRead = read(fd, buffer, sizeof(buffer) - 1);

buffer[bytesRead] = '\0';

printf("\nRe-read content using lseek:\n%s", buffer);

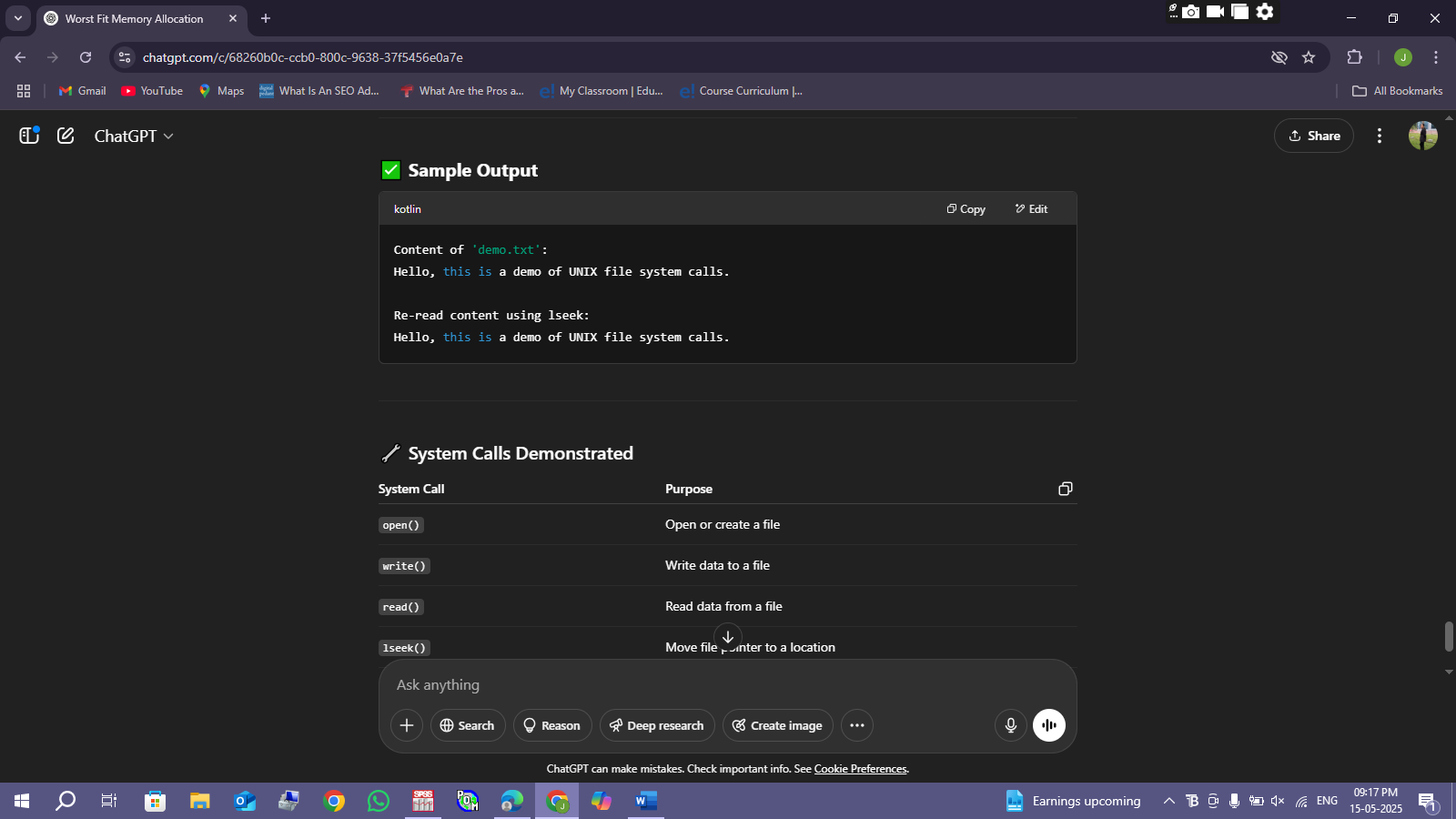
// 6. Close the file

close(fd);

return 0;

}

OUTPUT:



25.Construct a C program to implement the I/O system calls of UNIX (fcntl, seek, stat, opendir, readdir)

#include <stdio.h>

#include <fcntl.h> // for open(), fcntl()

#include <unistd.h> // for lseek(), close()

#include <sys/stat.h> // for stat()

#include <dirent.h> // for opendir(), readdir()

#include <string.h>

int main() {

int fd;

char \*filename = "sample.txt";

char buffer[50];

// 1. Create and open a file

fd = open(filename, O\_CREAT | O\_RDWR, 0644);

if (fd < 0) {

perror("open");

return 1;

}

// 2. Write data to the file

write(fd, "UNIX system call demo\n", 23);

// 3. Use lseek to move file pointer

lseek(fd, 0, SEEK\_SET); // Move to beginning

read(fd, buffer, 23);

buffer[23] = '\0';

printf("Data read using lseek: %s\n", buffer);

// 4. Use fcntl to get file descriptor flags

int flags = fcntl(fd, F\_GETFL);

if (flags < 0) {

perror("fcntl");

return 1;

} else {

printf("File descriptor flags: %d\n", flags);

}

// 5. Use stat to get file info

struct stat st;

if (stat(filename, &st) == 0) {

printf("File Size: %ld bytes\n", st.st\_size);

printf("Inode Number: %ld\n", st.st\_ino);

printf("Permissions: %o\n", st.st\_mode & 0777);

} else {

perror("stat");

}

close(fd); // Close the file

// 6. Use opendir and readdir to list current directory

DIR \*dir = opendir(".");

if (dir == NULL) {

perror("opendir");

return 1;

}

printf("\nFiles in current directory:\n");

struct dirent \*entry;

while ((entry = readdir(dir)) != NULL) {

printf(" %s\n", entry->d\_name);

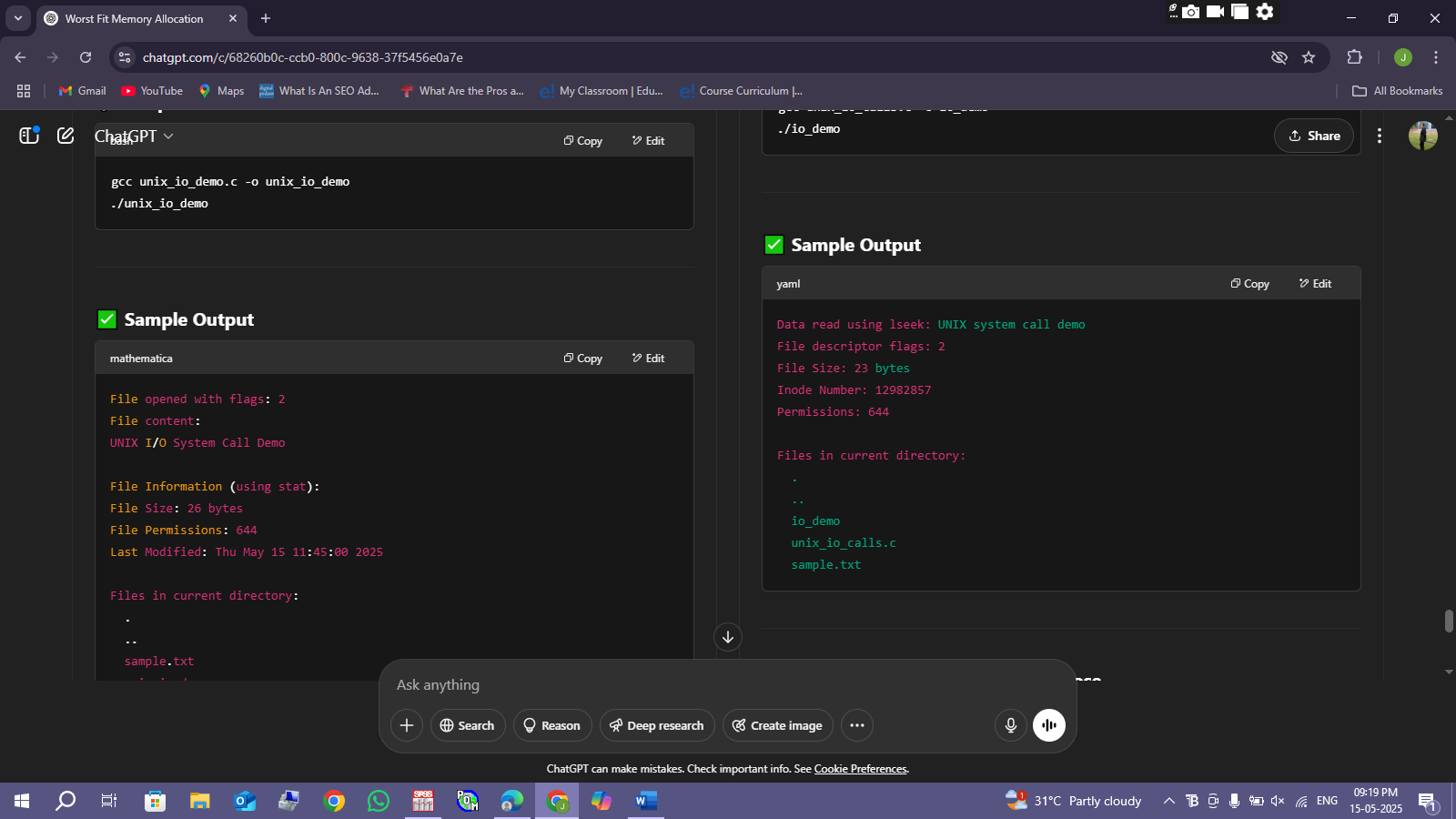
}

closedir(dir);

return 0;

}

OUTPUT:



26. Construct a C program to implement the file management operations.

#include <stdio.h>

#include <stdlib.h>

int main() {

FILE \*fp;

char ch;

char data[100];

// 1. Create and Write to a File

fp = fopen("file\_demo.txt", "w");

if (fp == NULL) {

perror("File creation failed");

return 1;

}

fprintf(fp, "This is a file management demo.\n");

fclose(fp);

printf("File created and data written.\n");

// 2. Read from the File

fp = fopen("file\_demo.txt", "r");

if (fp == NULL) {

perror("File read failed");

return 1;

}

printf("\nReading file content:\n");

while ((ch = fgetc(fp)) != EOF) {

putchar(ch);

}

fclose(fp);

// 3. Append Data

fp = fopen("file\_demo.txt", "a");

if (fp == NULL) {

perror("File append failed");

return 1;

}

fprintf(fp, "Appending more content to the file.\n");

fclose(fp);

printf("\nData appended to file.\n");

// 4. Rename the File

if (rename("file\_demo.txt", "renamed\_file.txt") == 0)

printf("File successfully renamed to 'renamed\_file.txt'.\n");

else {

perror("File rename failed");

return 1;

}

// 5. Delete the File

if (remove("renamed\_file.txt") == 0)

printf("File 'renamed\_file.txt' successfully deleted.\n");

else {

perror("File deletion failed");

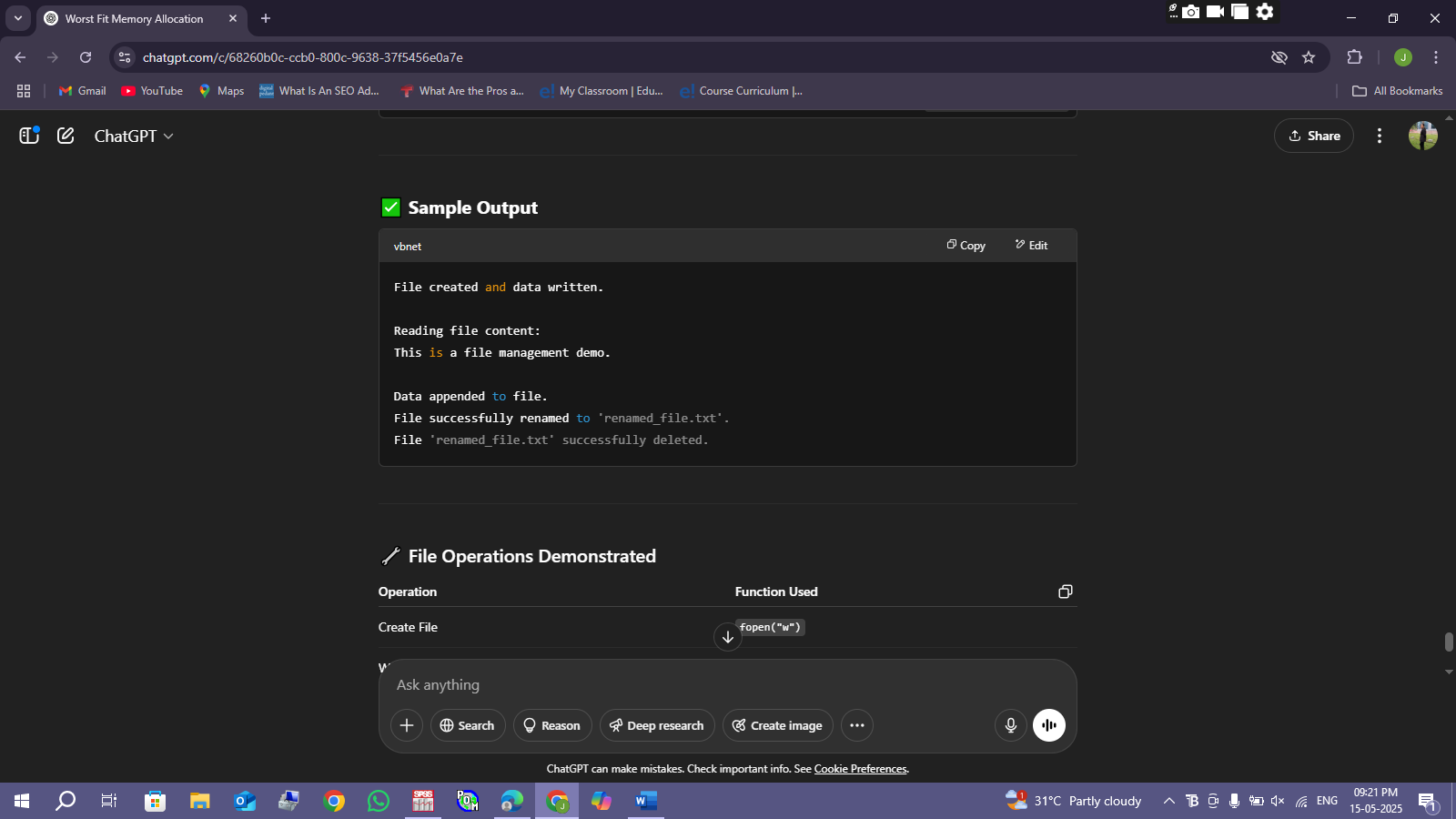
return 1;

}

return 0;

}

OUTPUT:



27. Develop a C program for simulating the function of ls UNIX Command.

#include <stdio.h>

#include <dirent.h>

#include <string.h>

#include <sys/types.h>

int main(int argc, char \*argv[]) {

struct dirent \*dp;

DIR \*dir;

// Use current directory if no argument is passed

char \*path = ".";

if (argc > 1) {

path = argv[1]; // Use directory given in argument

}

// Open directory

dir = opendir(path);

if (dir == NULL) {

perror("Unable to open directory");

return 1;

}

printf("Listing files in: %s\n\n", path);

// Read and display entries

while ((dp = readdir(dir)) != NULL) {

// Skip hidden files (like . and ..) unless you want to show them

if (dp->d\_name[0] != '.') {

printf("%s ", dp->d\_name);

}

}

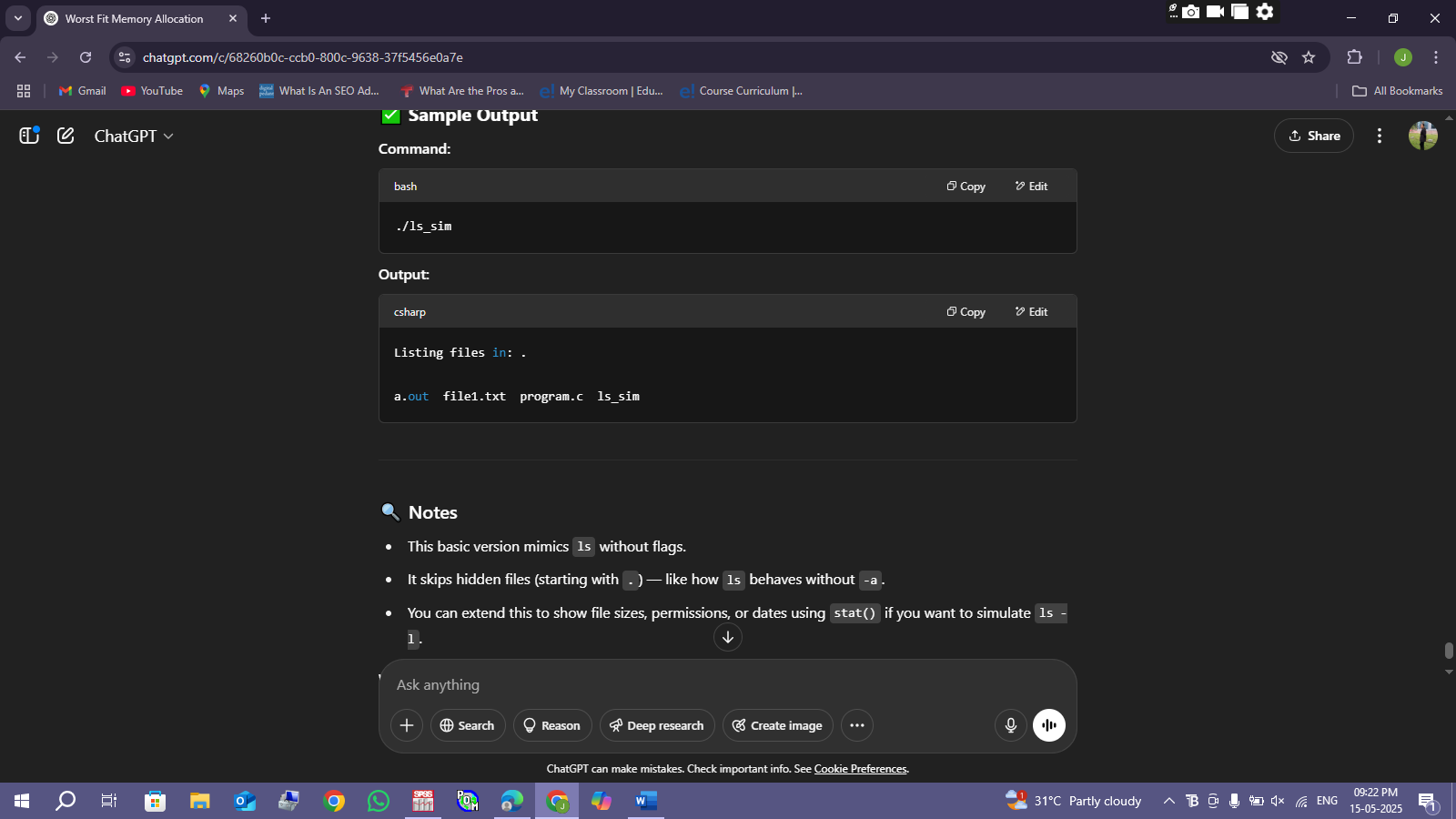
printf("\n");

closedir(dir);

return 0;

}

OUTPUT:



28. Write a C program for simulation of GREP UNIX command

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_LINE 1024

int main(int argc, char \*argv[]) {

FILE \*fp;

char line[MAX\_LINE];

char \*filename, \*pattern;

// Check for correct number of arguments

if (argc != 3) {

printf("Usage: %s <pattern> <filename>\n", argv[0]);

return 1;

}

pattern = argv[1];

filename = argv[2];

// Open the file for reading

fp = fopen(filename, "r");

if (fp == NULL) {

perror("Error opening file");

return 1;

}

printf("Lines matching pattern \"%s\" in file \"%s\":\n\n", pattern, filename);

// Read and search each line

while (fgets(line, MAX\_LINE, fp) != NULL) {

if (strstr(line, pattern) != NULL) {

printf("%s", line);

}

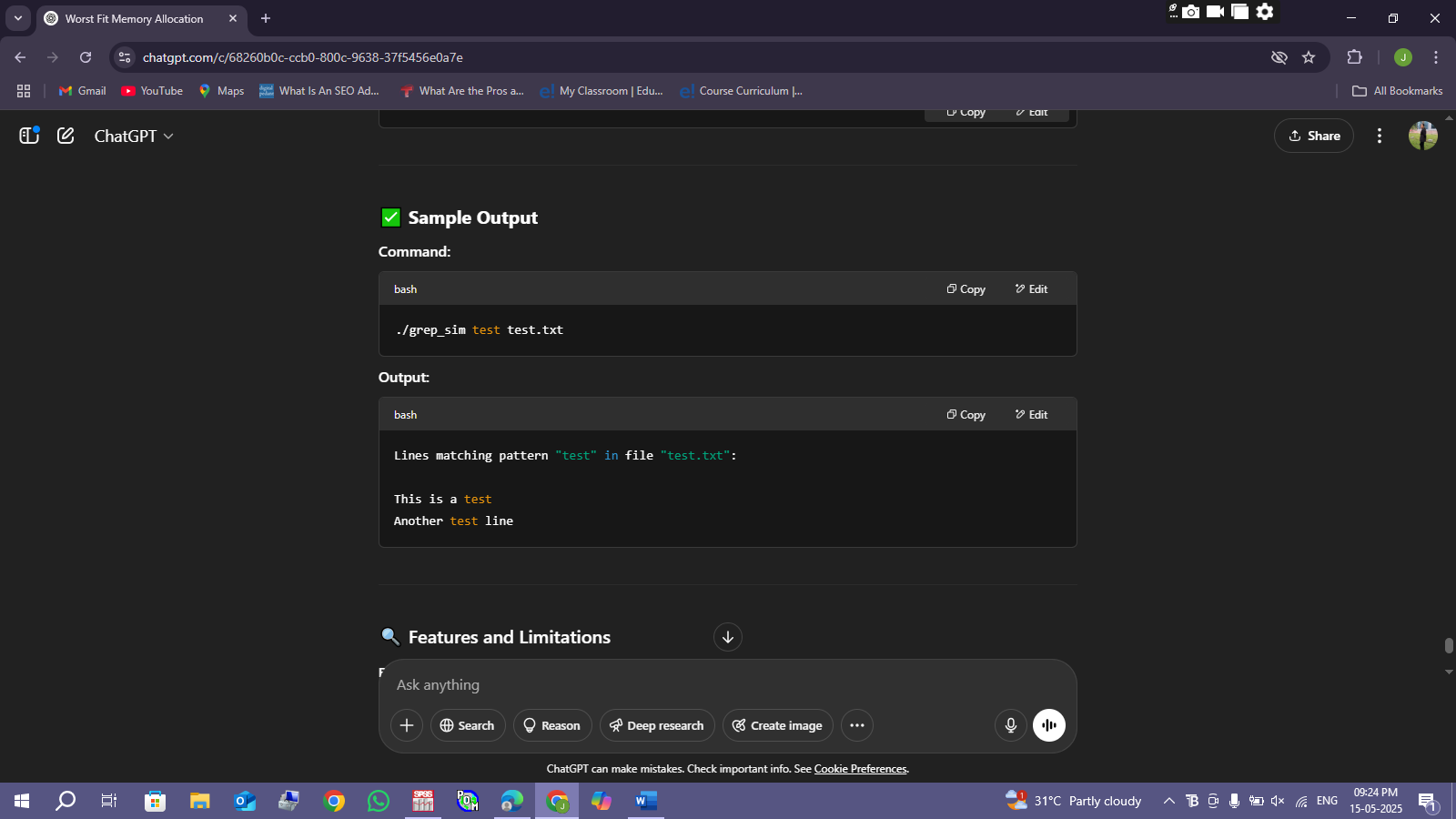
}

fclose(fp);

return 0;

}

OUTPUT:



29. Write a C program to simulate the solution of Classical Process Synchronization Problem

#include <stdio.h>

#include <stdlib.h>

#define SIZE 5 // Buffer size

int mutex = 1; // Binary semaphore for critical section

int full = 0; // Semaphore for number of full slots

int empty = SIZE; // Semaphore for number of empty slots

int buffer[SIZE]; // Shared buffer

int in = 0, out = 0; // Buffer pointers

// Wait operation (P)

int wait(int s) {

return --s;

}

// Signal operation (V)

int signal(int s) {

return ++s;

}

void producer() {

int item;

if (mutex == 1 && empty != 0) {

mutex = wait(mutex);

full = signal(full);

empty = wait(empty);

item = rand() % 100; // Produce a random item

buffer[in] = item;

printf("Producer produced: %d\n", item);

in = (in + 1) % SIZE;

mutex = signal(mutex);

} else {

printf("Buffer full! Producer waits.\n");

}

}

void consumer() {

if (mutex == 1 && full != 0) {

mutex = wait(mutex);

full = wait(full);

empty = signal(empty);

int item = buffer[out];

printf("Consumer consumed: %d\n", item);

out = (out + 1) % SIZE;

mutex = signal(mutex);

} else {

printf("Buffer empty! Consumer waits.\n");

}

}

int main() {

int choice;

printf("Producer-Consumer Problem Simulation\n");

while (1) {

printf("\n1. Produce\n2. Consume\n3. Exit\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

producer();

break;

case 2:

consumer();

break;

case 3:

exit(0);

default:

printf("Invalid choice!\n");

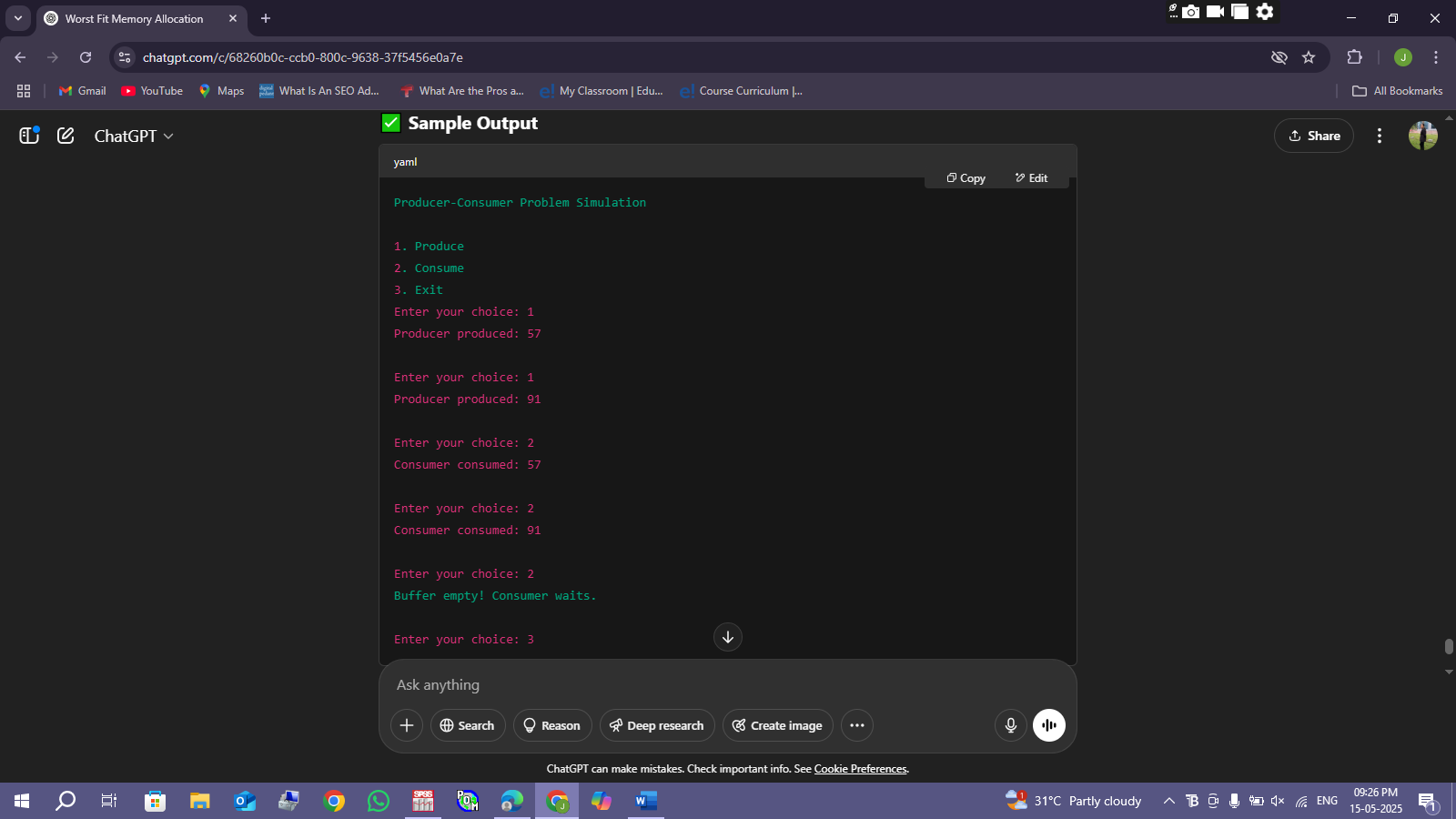
}

}

return 0;

}

OUTPUT:



30. Write C programs to demonstrate the following thread related concepts.

(i) create (ii) join (iii) equal (iv) exit#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

void\* threadFunc1(void\* arg) {

printf("Thread 1: Started\n");

sleep(1);

printf("Thread 1: Exiting\n");

pthread\_exit(NULL); // Explicit exit

}

void\* threadFunc2(void\* arg) {

printf("Thread 2: Started\n");

sleep(2);

printf("Thread 2: Exiting\n");

return NULL; // Implicit exit

}

int main() {

pthread\_t t1, t2;

// (i) CREATE threads

if (pthread\_create(&t1, NULL, threadFunc1, NULL) != 0) {

perror("Failed to create Thread 1");

return 1;

}

if (pthread\_create(&t2, NULL, threadFunc2, NULL) != 0) {

perror("Failed to create Thread 2");

return 1;

}

// (iii) EQUAL - Compare thread IDs

if (pthread\_equal(t1, t2)) {

printf("Main: Threads are equal\n");

} else {

printf("Main: Threads are not equal\n");

}

// (ii) JOIN threads (wait for them to finish)

pthread\_join(t1, NULL);

pthread\_join(t2, NULL);

printf("Main: All threads have exited\n");

return 0;

}

OUTPUT:

