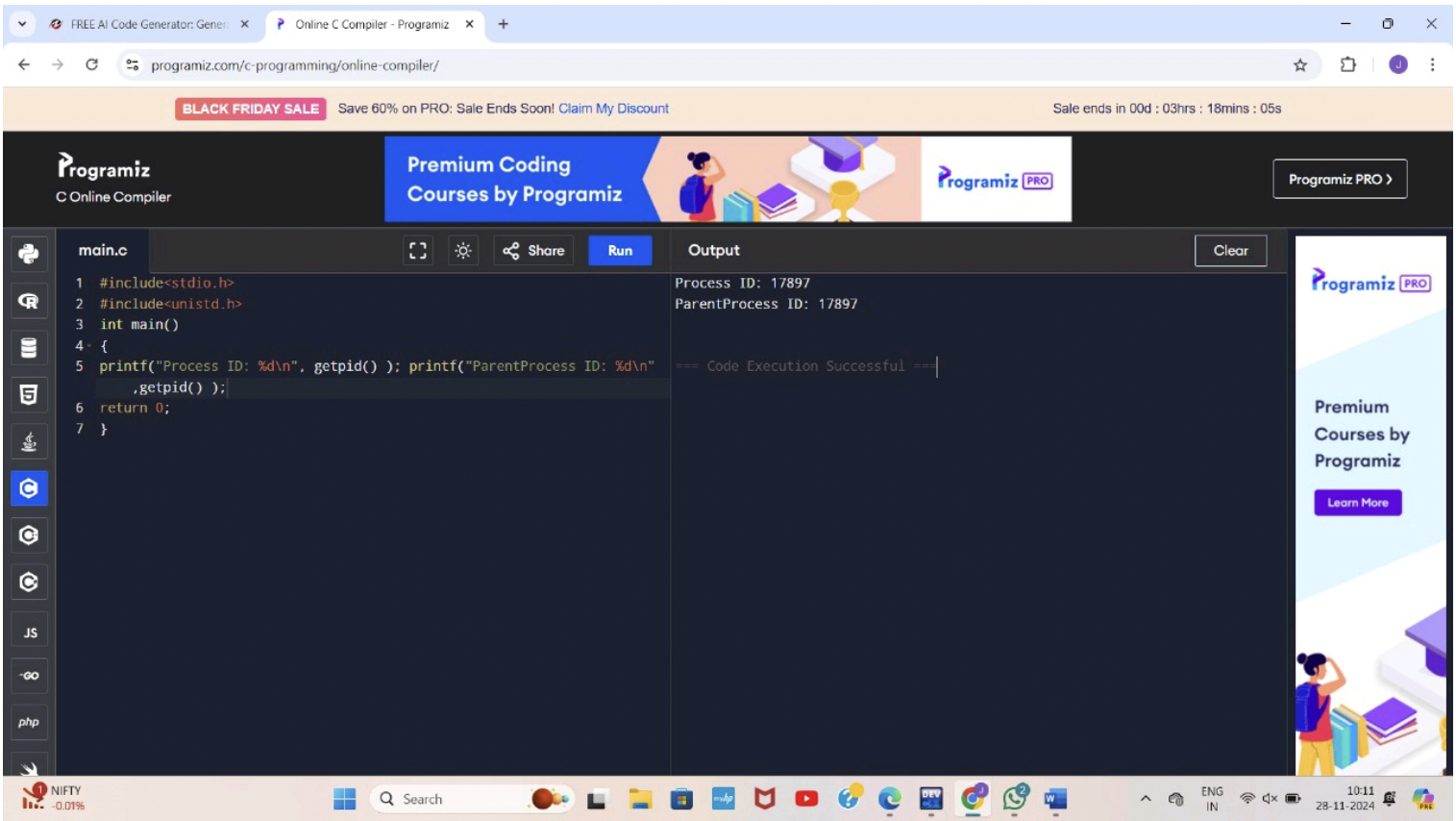
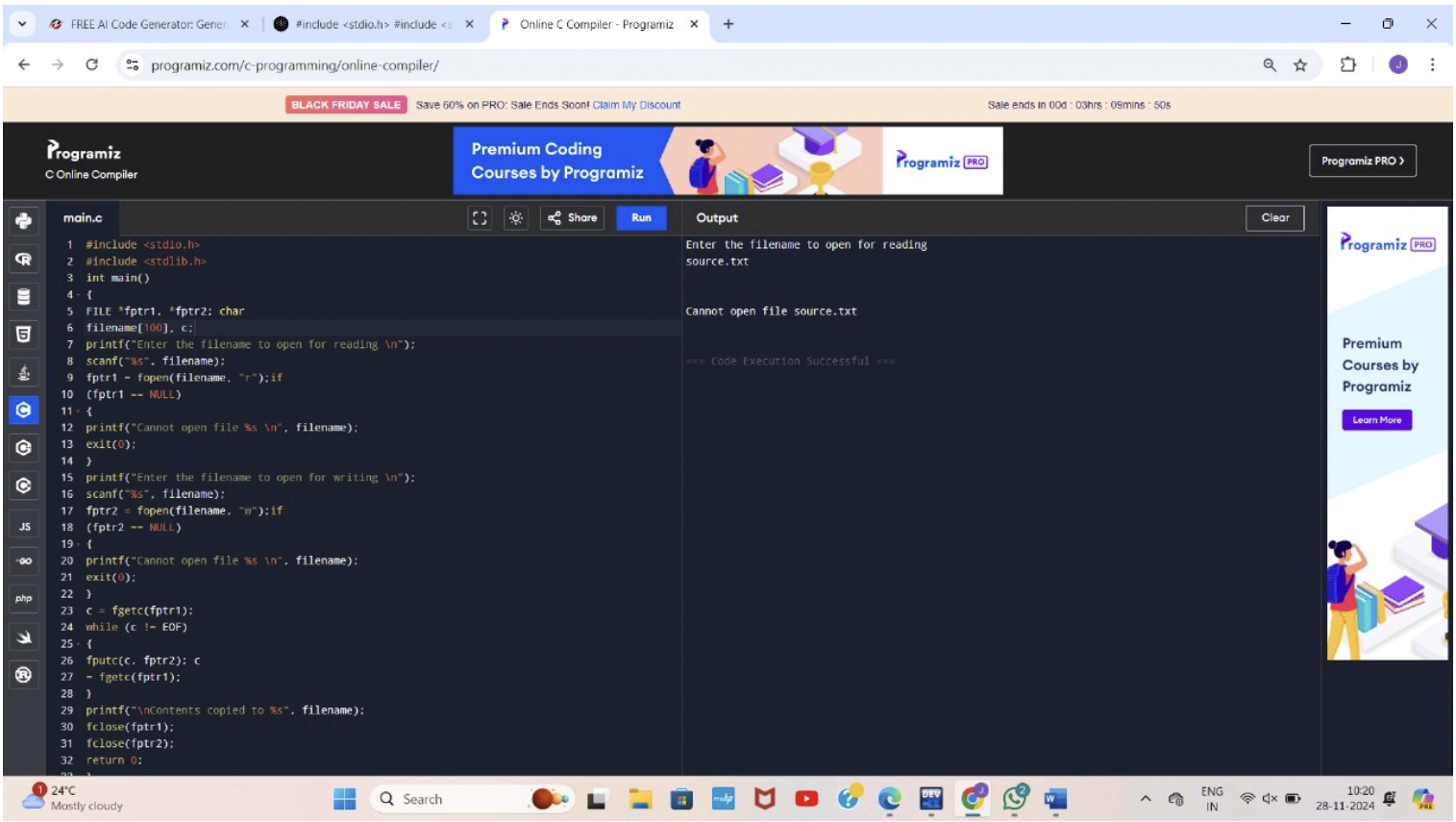
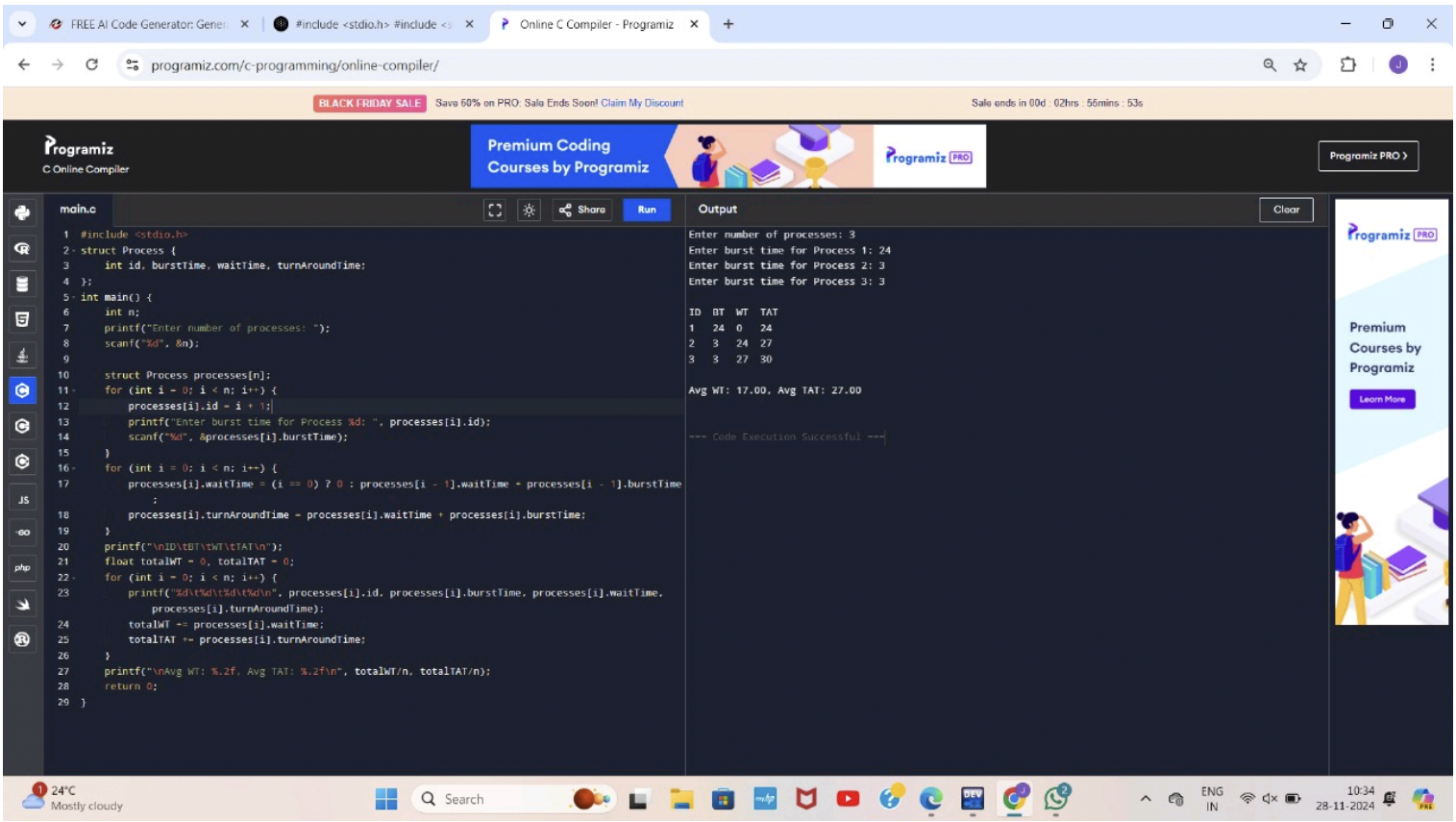
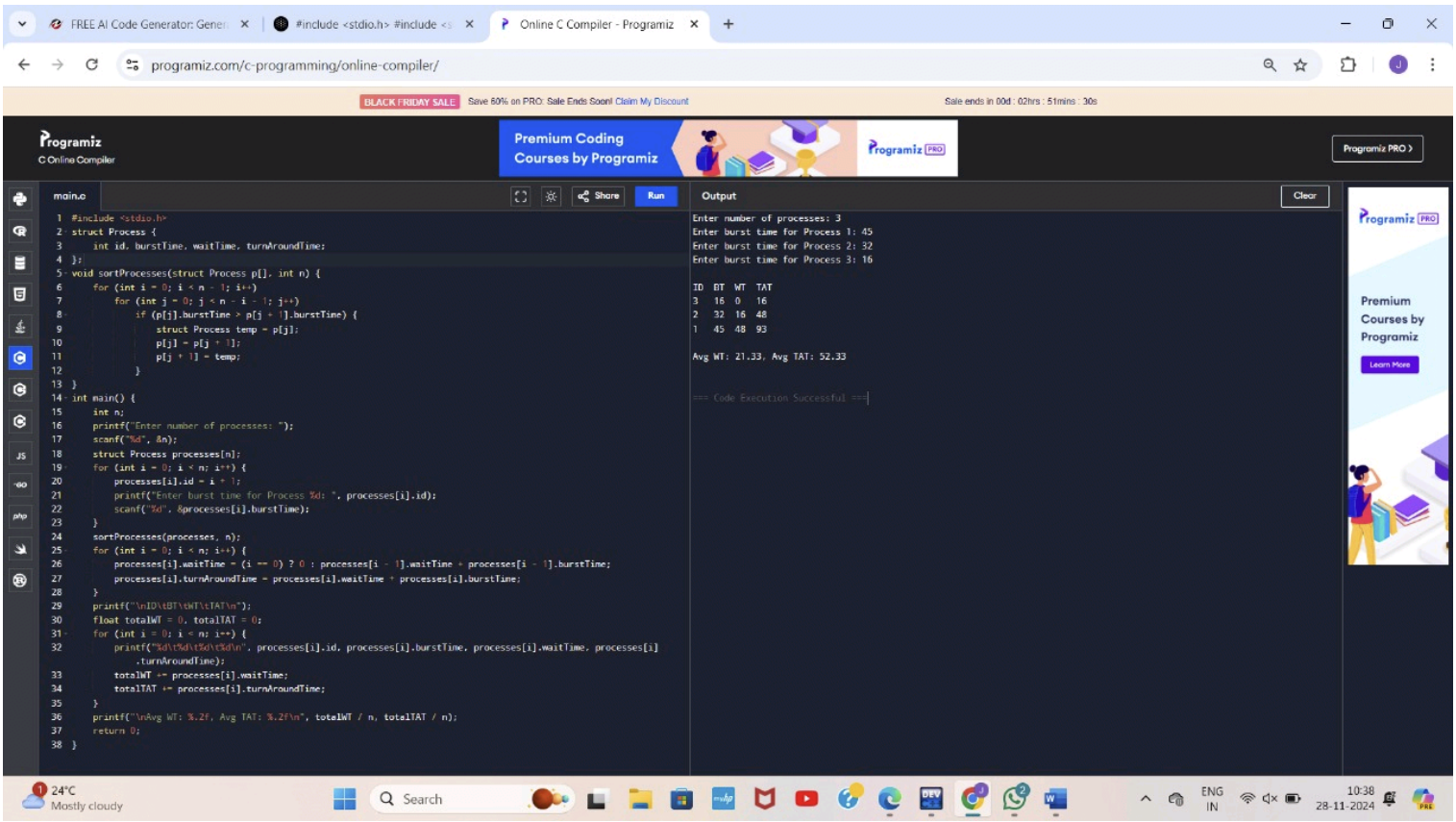
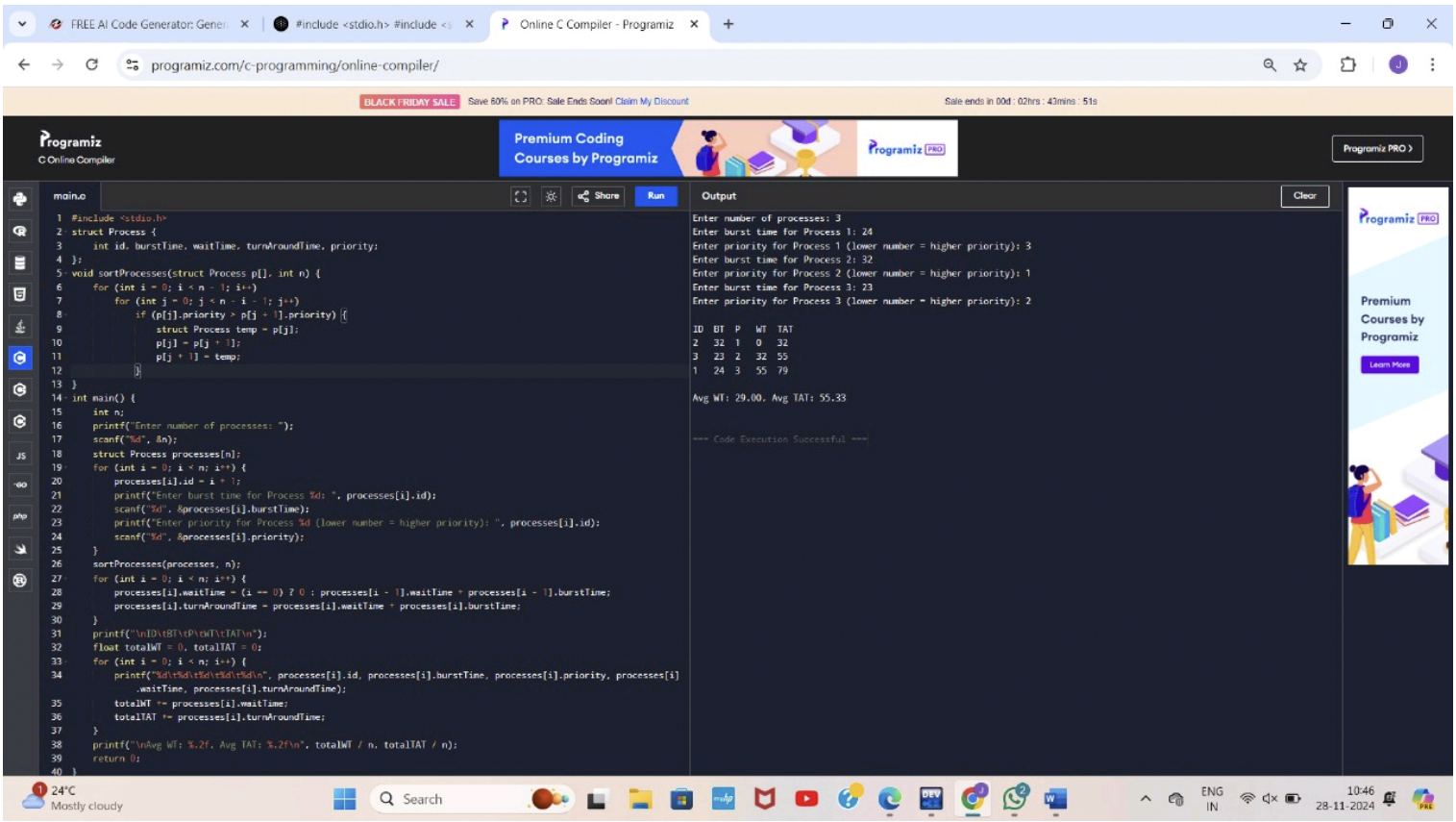
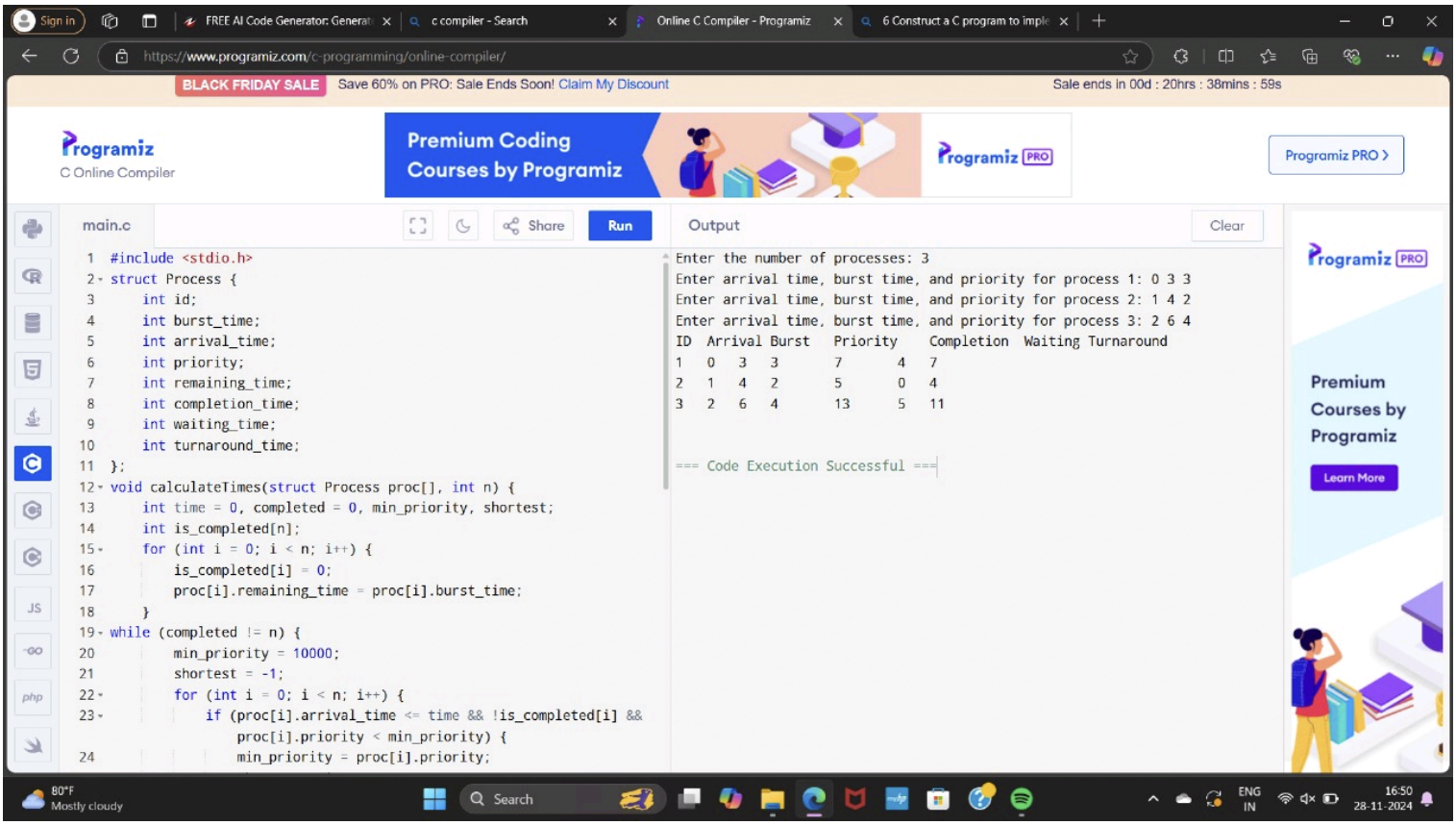

FREE AI Code Generator: Gener
Online C Compiler - Programiz
programiz.com/c-programming/online-compiler/
BLACK FRIDAY SALE
Save 60% on PRO: Sale Ends Soon! Claim My Discount
Sale ends in 00d : 03hrs : 18mins : 05s
Programiz
C Online Compiler
Premium Coding Courses by Programiz
Programiz PRO
Programiz PRO ›
main.c
Share
Run
Output
Clear
#include<stdio.h>
#include<unistd.h>
int main()
{
printf("Process ID: %d\n", getpid()); printf("ParentProcess ID: %d\n" ,getpid());
return 0;
}
Process ID: 17897
ParentProcess ID: 17897
=== Code Execution Successful ===
Premium Courses by Programiz
Learn More
JS
php
NIFTY -0.01%
Search
ENG IN
10:11
28-11-2024

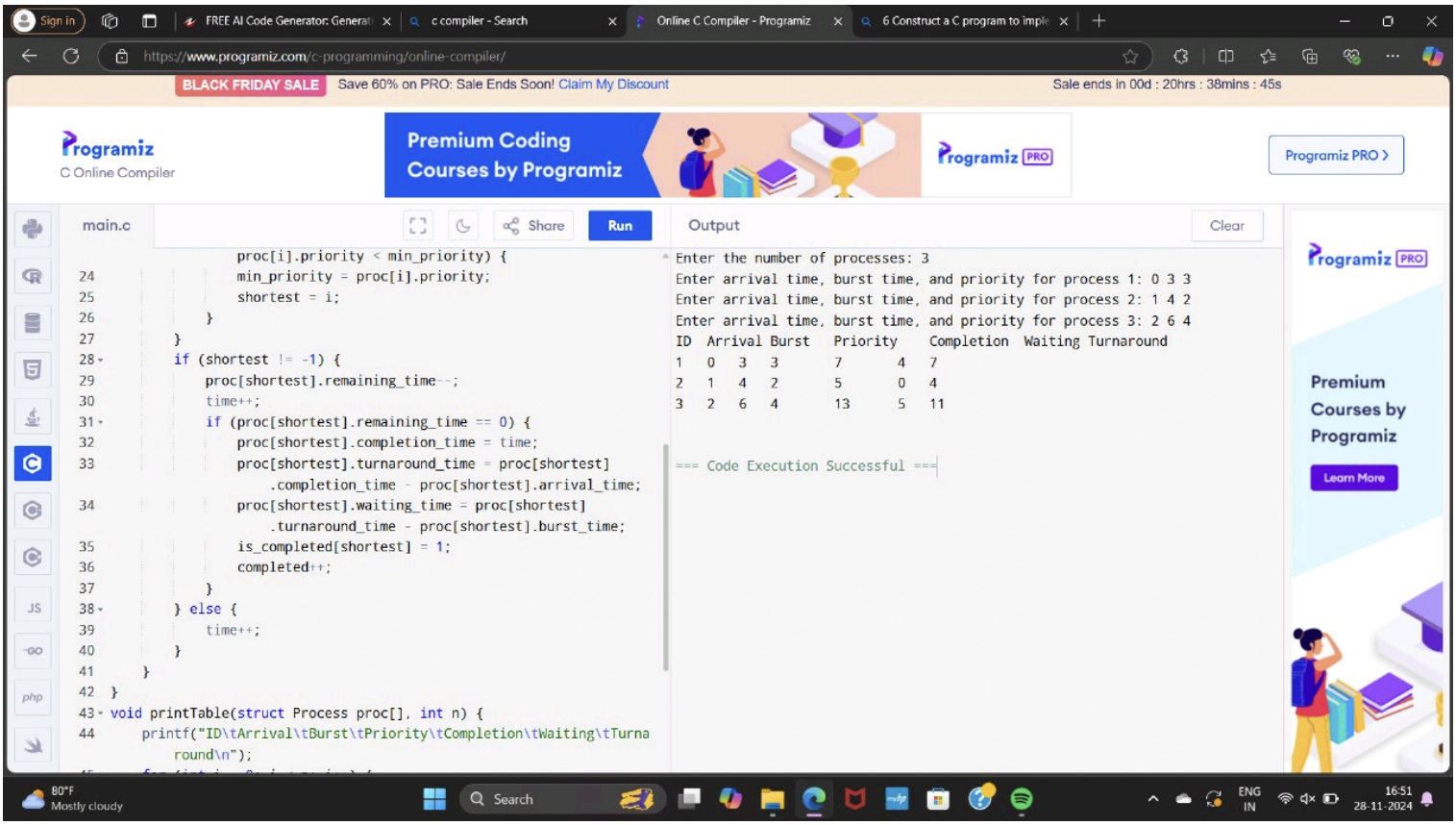

FREE AI Code Generator: Gener
#include <stdio.h> #include <s
Online C Compiler - Programiz
programiz.com/c-programming/online-compiler/
BLACK FRIDAY SALE
Save 60% on PRO: Sale Ends Soon! Claim My Discount
Sale ends in 00d : 03hrs : 09mins : 50s
Programiz
C Online Compiler
Premium Coding Courses by Programiz
Programiz PRO
Programiz PRO ›
main.c
Share
Run
Output
Clear
#include <stdio.h>
#include <stdlib.h>
int main()
{
FILE *fptr1, *fptr2; char
filename[100], c;
printf("Enter the filename to open for reading \n");
scanf("%s", filename);
fptr1 = fopen(filename, "r");if
(fptr1 == NULL)
{
printf("Cannot open file %s \n", filename);
exit(0);
}
printf("Enter the filename to open for writing \n");
scanf("%s", filename);
fptr2 = fopen(filename, "w");if
(fptr2 == NULL)
{
printf("Cannot open file %s \n", filename);
exit(0);
}
c = fgetc(fptr1);
while (c != EOF)
{
fputc(c, fptr2); c
= fgetc(fptr1);
}
printf("\nContents copied to %s", filename);
fclose(fptr1);
fclose(fptr2);
return 0;
Enter the filename to open for reading
source.txt
Cannot open file source.txt
=== Code Execution Successful ===
Programiz PRO
Premium Courses by Programiz
Learn More
24°C
Mostly cloudy
Search
ENG
IN
10:20
28-11-2024

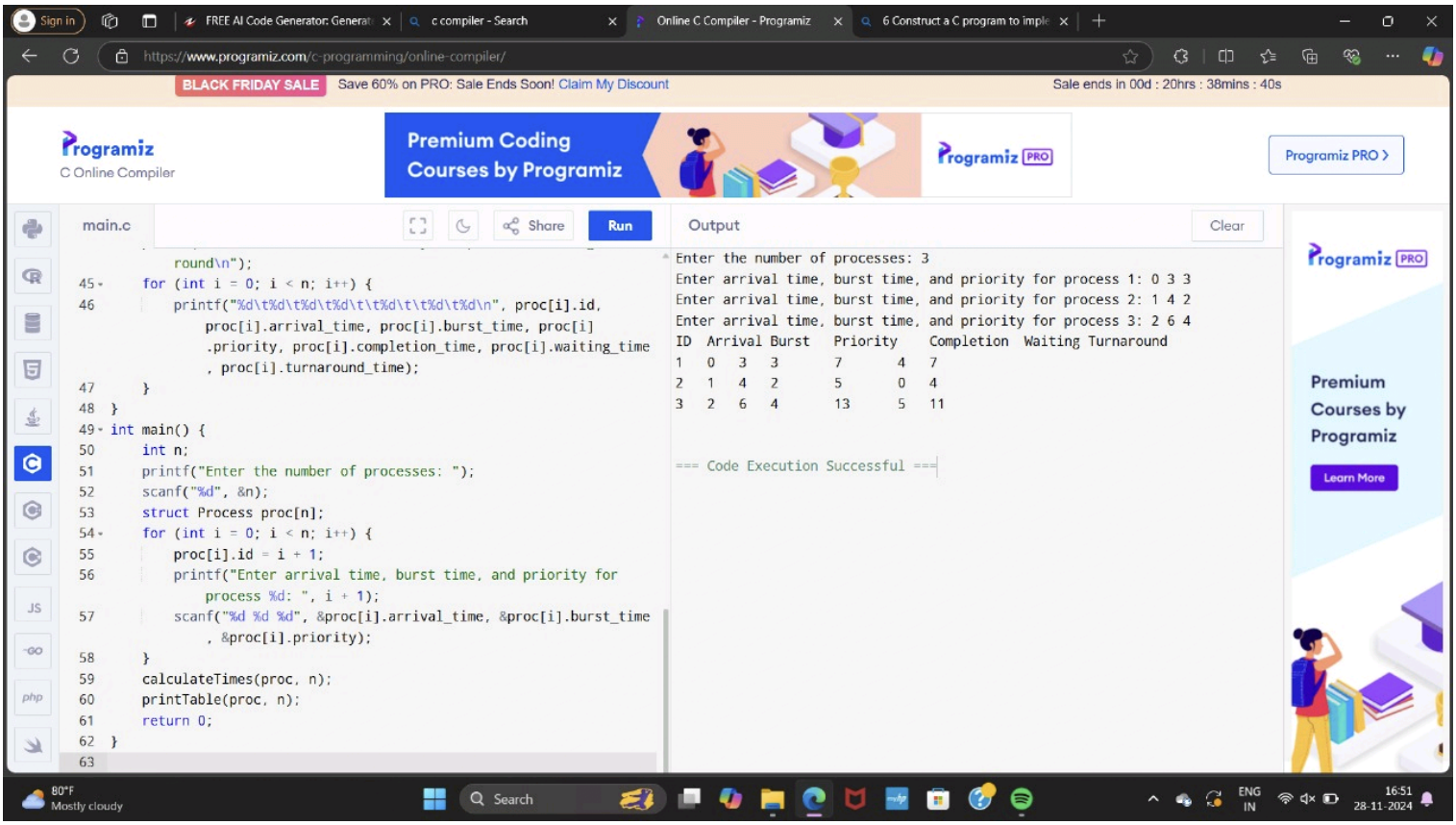

FREE AI Code Generator: Gener
#include <stdio.h> #include <
Online C Compiler - Programiz
programiz.com/c-programming/online-compiler/
BLACK FRIDAY SALE
Save 60% on PRO. Sale Ends Soon! Claim My Discount
Sale ends in 00d : 02hrs : 56mins : 53s
Programiz
C Online Compiler
Premium Coding Courses by Programiz
Programiz PRO
Programiz PRO ›
main.c
Share
Run
Output
Clear
#include <stdio.h>
struct Process {
int id, burstTime, waitTime, turnAroundTime;
};
int main() {
int n;
printf("Enter number of processes: ");
scanf("%d", &n);
struct Process processes[n];
for (int i = 0; i < n; i++) {
processes[i].id = i + 1;
printf("Enter burst time for Process %d: ", processes[i].id);
scanf("%d", &processes[i].burstTime);
}
for (int i = 0; i < n; i++) {
processes[i].waitTime = (i == 0) ? 0 : processes[i - 1].waitTime + processes[i - 1].burstTime
;
processes[i].turnAroundTime = processes[i].waitTime + processes[i].burstTime;
}
printf("\nID\tBT\tWT\tTAT\n");
float totalWT = 0, totalTAT = 0;
for (int i = 0; i < n; i++) {
printf("%d\t%d\t%d\t%d\n", processes[i].id, processes[i].burstTime, processes[i].waitTime,
processes[i].turnAroundTime);
totalWT += processes[i].waitTime;
totalTAT += processes[i].turnAroundTime;
}
printf("\nAvg WT: %.2f, Avg TAT: %.2f\n", totalWT/n, totalTAT/n);
return 0;
}
Enter number of processes: 3
Enter burst time for Process 1: 24
Enter burst time for Process 2: 3
Enter burst time for Process 3: 3
ID BT WT TAT
1 24 0 24
2 3 24 27
3 3 27 30
Avg WT: 17.00, Avg TAT: 27.00
--- Code Execution Successful ---
Programiz PRO
Premium Courses by Programiz
Learn More
24°C
Mostly cloudy
Search
ENG
IN
10:34
28-11-2024

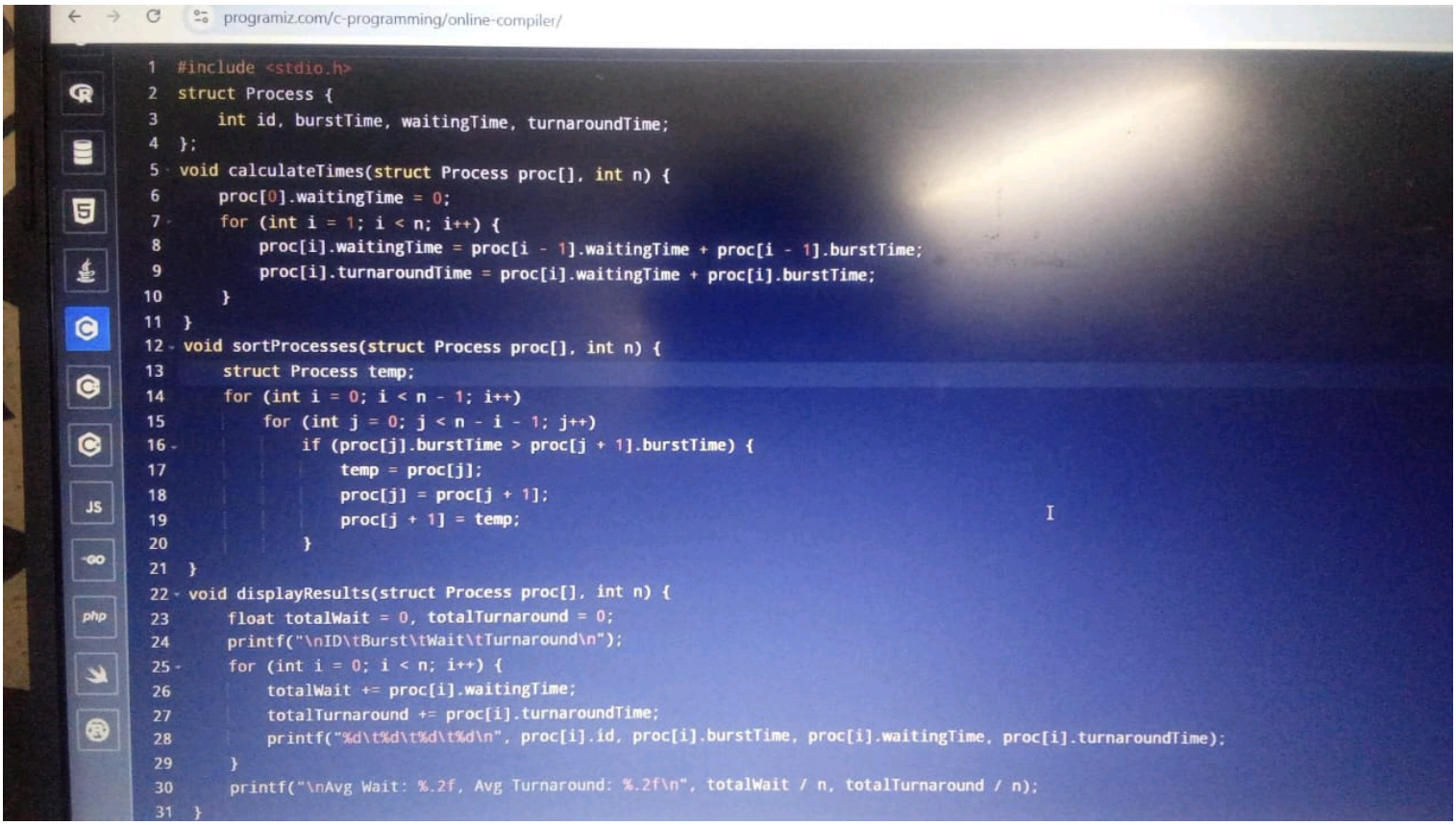

FREE AI Code Generator: Gener
#include <stdio.h> #include <
Online C Compiler - Programiz
programiz.com/c-programming/online-compiler/
BLACK FRIDAY SALE
Save 60% on PRO: Sale Ends Soon! Claim My Discount
Sale ends in 00d : 02hrs : 51mins : 30s
Programiz
C Online Compiler
Premium Coding Courses by Programiz
Programiz PRO
Programiz PRO >
main.c
Share
Run
Output
Clear
#include <stdio.h>
struct Process {
int id, burstTime, waitTime, turnAroundTime;
};
void sortProcesses(struct Process p[], int n) {
for (int i = 0; i < n - 1; i++)
for (int j = 0; j < n - i - 1; j++)
if (p[j].burstTime > p[j + 1].burstTime) {
struct Process temp = p[j];
p[j] = p[j + 1];
p[j + 1] = temp;
}
}
int main() {
int n;
printf("Enter number of processes: ");
scanf("%d", &n);
struct Process processes[n];
for (int i = 0; i < n; i++) {
processes[i].id = i + 1;
printf("Enter burst time for Process %d: ", processes[i].id);
scanf("%d", &processes[i].burstTime);
}
sortProcesses(processes, n);
for (int i = 0; i < n; i++) {
processes[i].waitTime = (i == 0) ? 0 : processes[i - 1].waitTime + processes[i - 1].burstTime;
processes[i].turnAroundTime = processes[i].waitTime + processes[i].burstTime;
}
printf("\nID\tBT\tWT\tTAT\n");
float totalWT = 0, totalTAT = 0;
for (int i = 0; i < n; i++) {
printf("%d\t%d\t%d\t%d\n", processes[i].id, processes[i].burstTime, processes[i].waitTime, processes[i]
.turnAroundTime);
totalWT += processes[i].waitTime;
totalTAT += processes[i].turnAroundTime;
}
printf("\nAvg WT: %.2f, Avg TAT: %.2f\n", totalWT / n, totalTAT / n);
return 0;
}
Enter number of processes: 3
Enter burst time for Process 1: 45
Enter burst time for Process 2: 32
Enter burst time for Process 3: 16
ID BT WT TAT
3 16 0 16
2 32 16 48
1 45 48 93
Avg WT: 21.33, Avg TAT: 52.33
=== Code Execution Successful ===
Programiz PRO
Premium Courses by Programiz
Learn More
24°C
Mostly cloudy
Search
ENG
IN
10:38
28-11-2024


FREE AI Code Generator: Gener
#include <stdio.h> #include <s
Online C Compiler - Programiz
programiz.com/c-programming/online-compiler/
BLACK FRIDAY SALE
Save 60% on PRO: Sale Ends Soon! Claim My Discount
Sale ends in 00d : 02hrs : 43mins : 51s
Programiz
C Online Compiler
Premium Coding Courses by Programiz
Programiz PRO
Programiz PRO >
main.c
Share
Run
Output
Clear
#include <stdio.h>
struct Process {
int id, burstTime, waitTime, turnAroundTime, priority;
};
void sortProcesses(struct Process p[], int n) {
for (int i = 0; i < n - 1; i++)
for (int j = 0; j < n - i - 1; j++)
if (p[j].priority > p[j + 1].priority) {
struct Process temp = p[j];
p[j] = p[j + 1];
p[j + 1] = temp;
}
}
int main() {
int n;
printf("Enter number of processes: ");
scanf("%d", &n);
struct Process processes[n];
for (int i = 0; i < n; i++) {
processes[i].id = i + 1;
printf("Enter burst time for Process %d: ", processes[i].id);
scanf("%d", &processes[i].burstTime);
printf("Enter priority for Process %d (lower number = higher priority): ", processes[i].id);
scanf("%d", &processes[i].priority);
}
sortProcesses(processes, n);
for (int i = 0; i < n; i++) {
processes[i].waitTime = (i == 0) ? 0 : processes[i - 1].waitTime + processes[i - 1].burstTime;
processes[i].turnAroundTime = processes[i].waitTime + processes[i].burstTime;
}
printf("\nID\tBT\tP\tWT\tTAT\n");
float totalWT = 0, totalTAT = 0;
for (int i = 0; i < n; i++) {
printf("%d\t%d\t%d\t%d\t%d\n", processes[i].id, processes[i].burstTime, processes[i].priority, processes[i]
.waitTime, processes[i].turnAroundTime);
totalWT += processes[i].waitTime;
totalTAT += processes[i].turnAroundTime;
}
printf("\nAvg WT: %.2f, Avg TAT: %.2f\n", totalWT / n, totalTAT / n);
return 0;
}
Enter number of processes: 3
Enter burst time for Process 1: 24
Enter priority for Process 1 (lower number = higher priority): 3
Enter burst time for Process 2: 32
Enter priority for Process 2 (lower number = higher priority): 1
Enter burst time for Process 3: 23
Enter priority for Process 3 (lower number = higher priority): 2
ID BT P WT TAT
2 32 1 0 32
3 23 2 32 55
1 24 3 55 79
Avg WT: 29.00, Avg TAT: 55.33
=== Code Execution Successful ===
Programiz PRO
Premium Courses by Programiz
Learn More
24°C
Mostly cloudy
Search
ENG
IN
10:46
28-11-2024


Sign in
FREE AI Code Generator: Generat
c compiler - Search
Online C Compiler - Programiz
6 Construct a C program to imple
https://www.programiz.com/c-programming/online-compiler/
BLACK FRIDAY SALE
Save 60% on PRO: Sale Ends Soon! Claim My Discount
Sale ends in 00d : 20hrs : 38mins : 59s
Programiz
C Online Compiler
Premium Coding Courses by Programiz
Programiz PRO
Programiz PRO >
main.c
Share
Run
Output
Clear
#include <stdio.h>
struct Process {
int id;
int burst_time;
int arrival_time;
int priority;
int remaining_time;
int completion_time;
int waiting_time;
int turnaround_time;
};
void calculateTimes(struct Process proc[], int n) {
int time = 0, completed = 0, min_priority, shortest;
int is_completed[n];
for (int i = 0; i < n; i++) {
is_completed[i] = 0;
proc[i].remaining_time = proc[i].burst_time;
}
while (completed != n) {
min_priority = 10000;
shortest = -1;
for (int i = 0; i < n; i++) {
if (proc[i].arrival_time <= time && !is_completed[i] &&
proc[i].priority < min_priority) {
min_priority = proc[i].priority;
Enter the number of processes: 3
Enter arrival time, burst time, and priority for process 1: 0 3 3
Enter arrival time, burst time, and priority for process 2: 1 4 2
Enter arrival time, burst time, and priority for process 3: 2 6 4
ID Arrival Burst Priority Completion Waiting Turnaround
1 0 3 3 7 4 7
2 1 4 2 5 0 4
3 2 6 4 13 5 11
=== Code Execution Successful ===
Programiz PRO
Premium Courses by Programiz
Learn More
80°F
Mostly cloudy
Search
ENG
IN
16:50
28-11-2024


Sign in
FREE AI Code Generator: Generat
c compiler - Search
Online C Compiler - Programiz
6 Construct a C program to imple
https://www.programiz.com/c-programming/online-compiler/
BLACK FRIDAY SALE
Save 60% on PRO: Sale Ends Soon! Claim My Discount
Sale ends in 00d : 20hrs : 38mins : 45s
Programiz
C Online Compiler
Premium Coding Courses by Programiz
Programiz PRO
Programiz PRO >
main.c
Share
Run
Output
Clear
proc[i].priority < min_priority) {
min_priority = proc[i].priority;
shortest = i;
}
}
if (shortest != -1) {
proc[shortest].remaining_time--;
time++;
if (proc[shortest].remaining_time == 0) {
proc[shortest].completion_time = time;
proc[shortest].turnaround_time = proc[shortest]
.completion_time - proc[shortest].arrival_time;
proc[shortest].waiting_time = proc[shortest]
.turnaround_time - proc[shortest].burst_time;
is_completed[shortest] = 1;
completed++;
}
} else {
time++;
}
}
}
void printTable(struct Process proc[], int n) {
printf("ID\tArrival\tBurst\tPriority\tCompletion\tWaiting\tTurna
round\n");
Enter the number of processes: 3
Enter arrival time, burst time, and priority for process 1: 0 3 3
Enter arrival time, burst time, and priority for process 2: 1 4 2
Enter arrival time, burst time, and priority for process 3: 2 6 4
ID Arrival Burst Priority Completion Waiting Turnaround
1 0 3 3 7 4 7
2 1 4 2 5 0 4
3 2 6 4 13 5 11
=== Code Execution Successful ===
Programiz PRO
Premium Courses by Programiz
Learn More
80°F
Mostly cloudy
Search
ENG
IN
16:51
28-11-2024

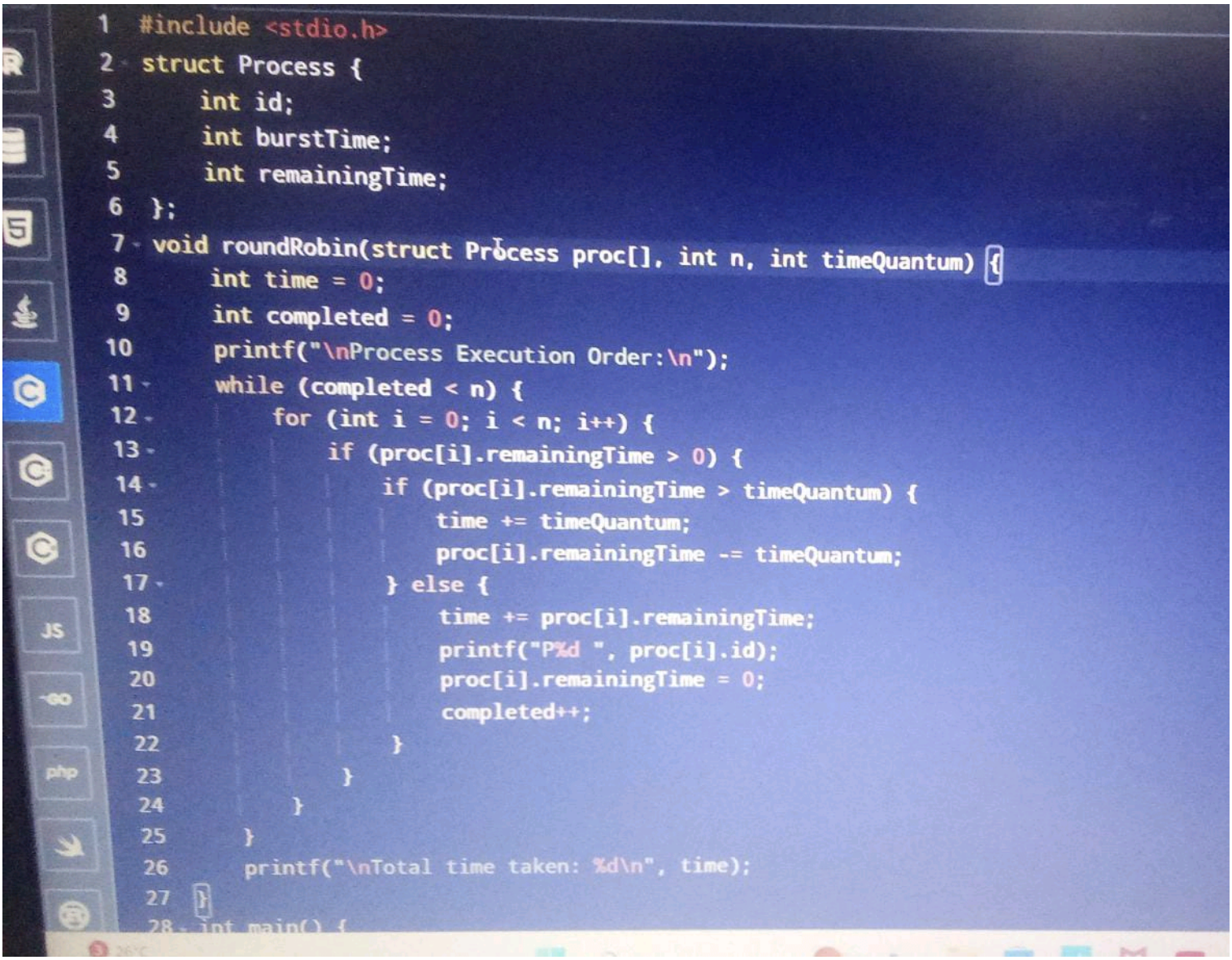

FREE AI Code Generator: Generat
c compiler - Search
Online C Compiler - Programiz
6 Construct a C program to imple
https://www.programiz.com/c-programming/online-compiler/
BLACK FRIDAY SALE
Save 60% on PRO: Sale Ends Soon! Claim My Discount
Sale ends in 00d : 20hrs : 38mins : 40s
Programiz
C Online Compiler
Premium Coding Courses by Programiz
Programiz PRO
Programiz PRO >
main.c
Share
Run
Output
Clear
round\n");
for (int i = 0; i < n; i++) {
printf("%d\t%d\t%d\t%d\t\t%d\t\t%d\t%d\n", proc[i].id,
proc[i].arrival_time, proc[i].burst_time, proc[i]
.priority, proc[i].completion_time, proc[i].waiting_time
, proc[i].turnaround_time);
}
}
int main() {
int n;
printf("Enter the number of processes: ");
scanf("%d", &n);
struct Process proc[n];
for (int i = 0; i < n; i++) {
proc[i].id = i + 1;
printf("Enter arrival time, burst time, and priority for
process %d: ", i + 1);
scanf("%d %d %d", &proc[i].arrival_time, &proc[i].burst_time
, &proc[i].priority);
}
calculateTimes(proc, n);
printTable(proc, n);
return 0;
}
Enter the number of processes: 3
Enter arrival time, burst time, and priority for process 1: 0 3 3
Enter arrival time, burst time, and priority for process 2: 1 4 2
Enter arrival time, burst time, and priority for process 3: 2 6 4
ID Arrival Burst Priority Completion Waiting Turnaround
1 0 3 3 7 4 7
2 1 4 2 5 0 4
3 2 6 4 13 5 11
=== Code Execution Successful ===
Programiz PRO
Premium Courses by Programiz
Learn More
80°F
Mostly cloudy
Search
ENG
IN
16:51
28-11-2024


programiz.com/c-programming/online-compiler/
#include <stdio.h>
struct Process {
int id, burstTime, waitingTime, turnaroundTime;
};
void calculateTimes(struct Process proc[], int n) {
proc[0].waitingTime = 0;
for (int i = 1; i < n; i++) {
proc[i].waitingTime = proc[i - 1].waitingTime + proc[i - 1].burstTime;
proc[i].turnaroundTime = proc[i].waitingTime + proc[i].burstTime;
}
}
void sortProcesses(struct Process proc[], int n) {
struct Process temp;
for (int i = 0; i < n - 1; i++)
for (int j = 0; j < n - i - 1; j++)
if (proc[j].burstTime > proc[j + 1].burstTime) {
temp = proc[j];
proc[j] = proc[j + 1];
proc[j + 1] = temp;
}
}
void displayResults(struct Process proc[], int n) {
float totalWait = 0, totalTurnaround = 0;
printf("\nID\tBurst\tWait\tTurnaround\n");
for (int i = 0; i < n; i++) {
totalWait += proc[i].waitingTime;
totalTurnaround += proc[i].turnaroundTime;
printf("%d\t%d\t%d\t%d\n", proc[i].id, proc[i].burstTime, proc[i].waitingTime, proc[i].turnaroundTime);
}
printf("\nAvg Wait: %.2f, Avg Turnaround: %.2f\n", totalWait / n, totalTurnaround / n);
}

```
            }
}
void displayResults(struct Process proc[], int n) {
    float totalWait = 0, totalTurnaround = 0;
    printf("\nID\tBurst\tWait\tTurnaround\n");
    for (int i = 0; i < n; i++) {
        totalWait += proc[i].waitingTime;
        totalTurnaround += proc[i].turnaroundTime;
        printf("%d\t%d\t%d\t%d\n", proc[i].id, proc[i].burstTime, proc[i].waitingTime, proc[i].turnaroundTime);
    }
    printf("\nAvg Wait: %.2f, Avg Turnaround: %.2f\n", totalWait / n, totalTurnaround / n);
}
int main() {
    struct Process proc[10];
    int n;

    printf("Enter number of processes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        proc[i].id = i + 1;
        printf("Enter Burst Time for Process %d: ", proc[i].id);
        scanf("%d", &proc[i].burstTime);
    }
    sortProcesses(proc, n);
    calculateTimes(proc, n);
    displayResults(proc, n);
    return 0;
}
```

Output

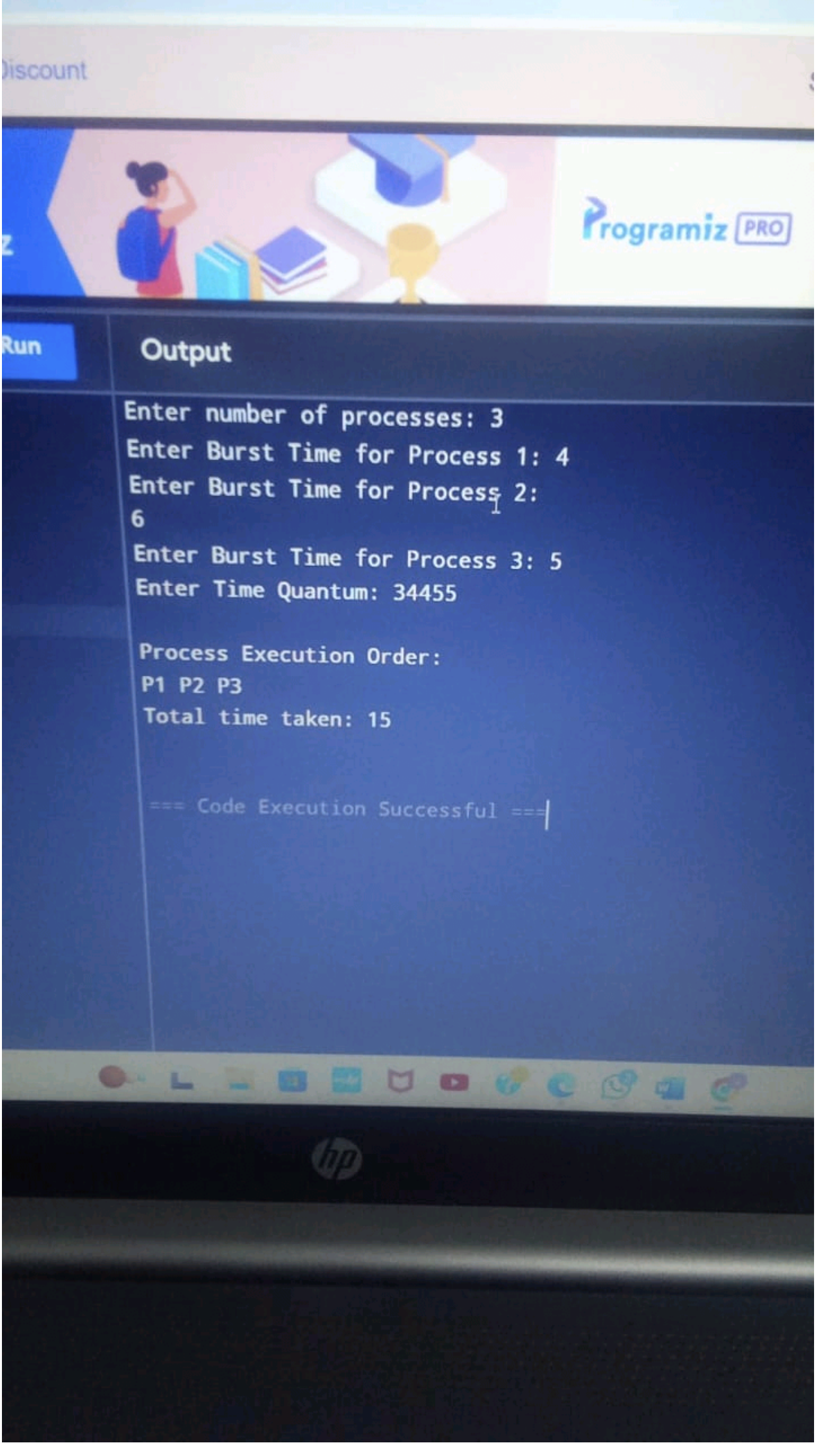
```
Enter number of processes: 3
Enter Burst Time for Process 1: 24
Enter Burst Time for Process 2: 6
Enter Burst Time for Process 3: 3

ID  Burst   Wait    Turnaround
3   3   0   0
2   6   3   9
1   24  9   33

Avg Wait: 4.00, Avg Turnaround: 14.00


=== Code Execution Successful ===
=== Session Ended. Please Run the code again ===
```

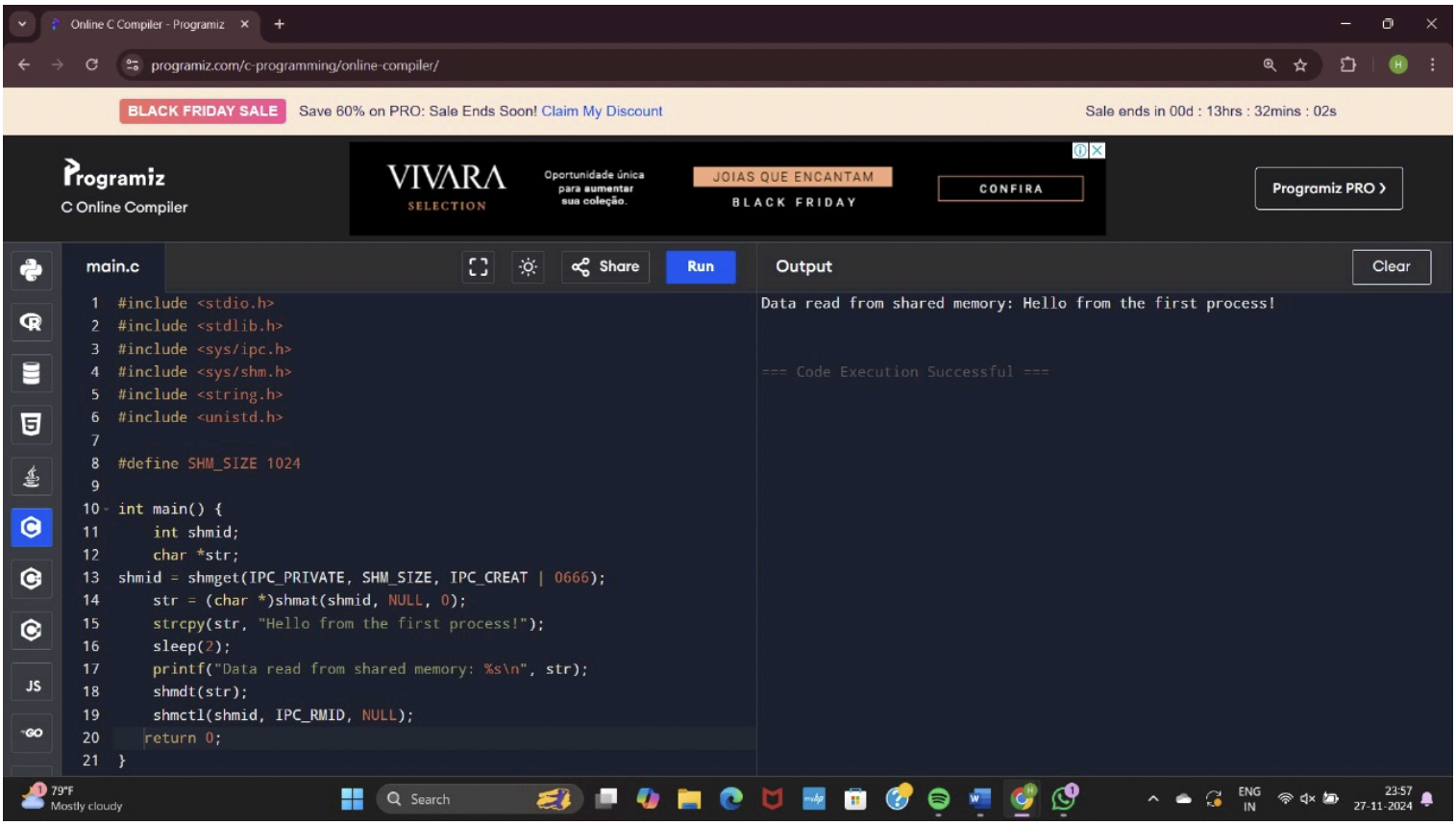


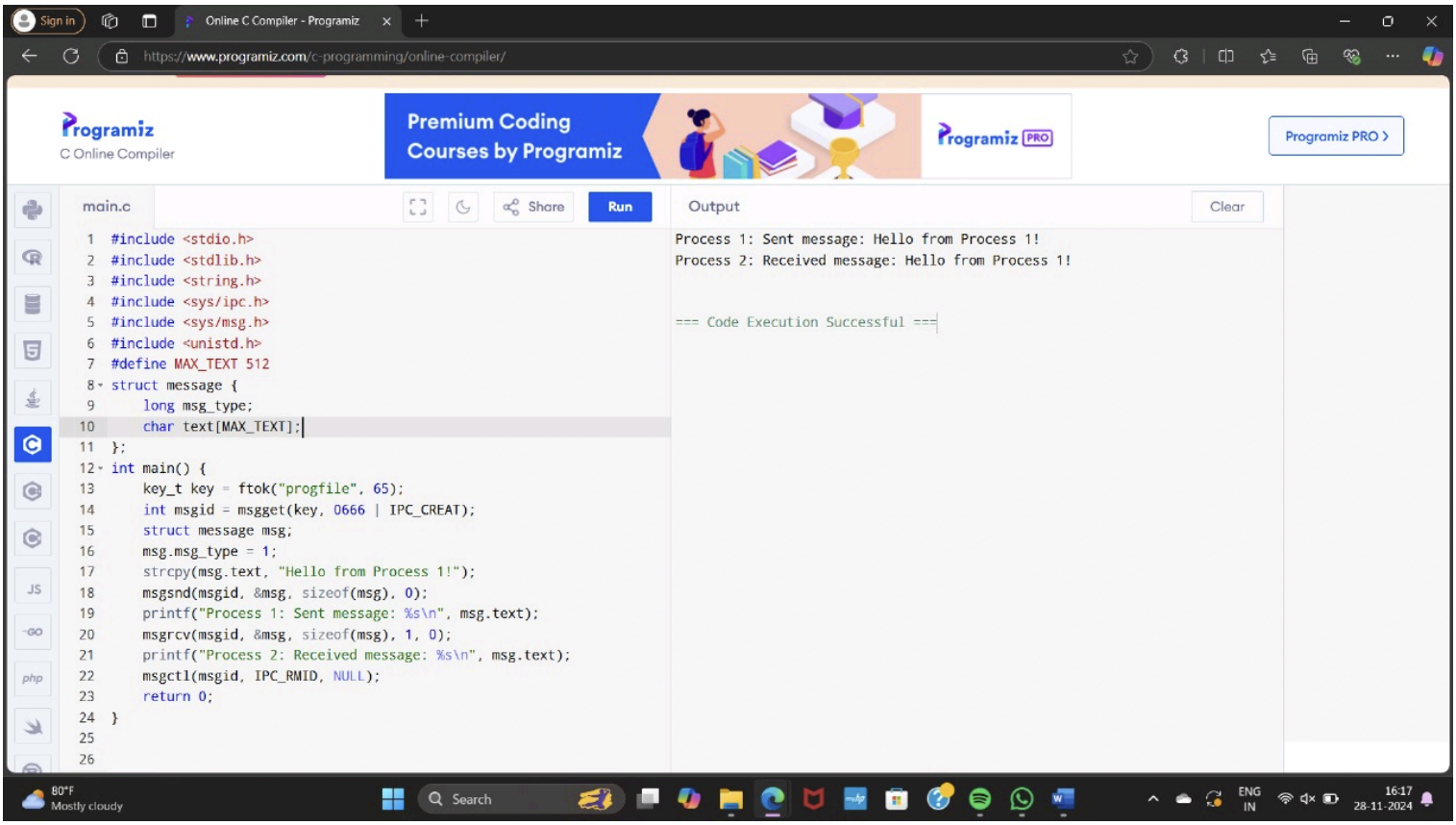
```c
#include <stdio.h>
struct Process {
    int id;
    int burstTime;
    int remainingTime;
};
void roundRobin(struct Process proc[], int n, int timeQuantum) {
    int time = 0;
    int completed = 0;
    printf("\nProcess Execution Order:\n");
    while (completed < n) {
        for (int i = 0; i < n; i++) {
            if (proc[i].remainingTime > 0) {
                if (proc[i].remainingTime > timeQuantum) {
                    time += timeQuantum;
                    proc[i].remainingTime -= timeQuantum;
                } else {
                    time += proc[i].remainingTime;
                    printf("P%d ", proc[i].id);
                    proc[i].remainingTime = 0;
                    completed++;
                }
            }
        }
    }
    printf("\nTotal time taken: %d\n", time);
}
int main() {
```

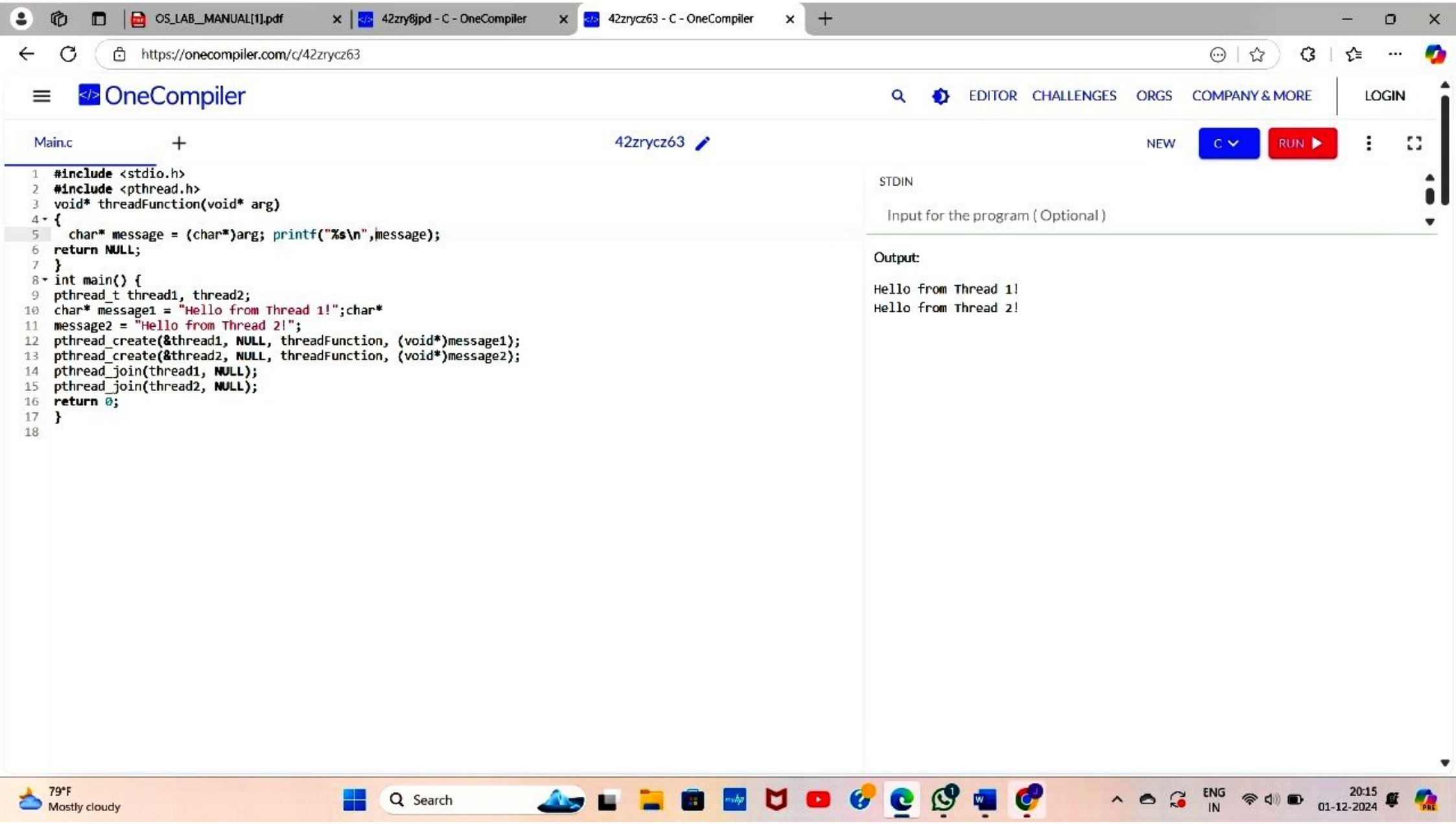
programiz.com/c-programming/online-compiler/

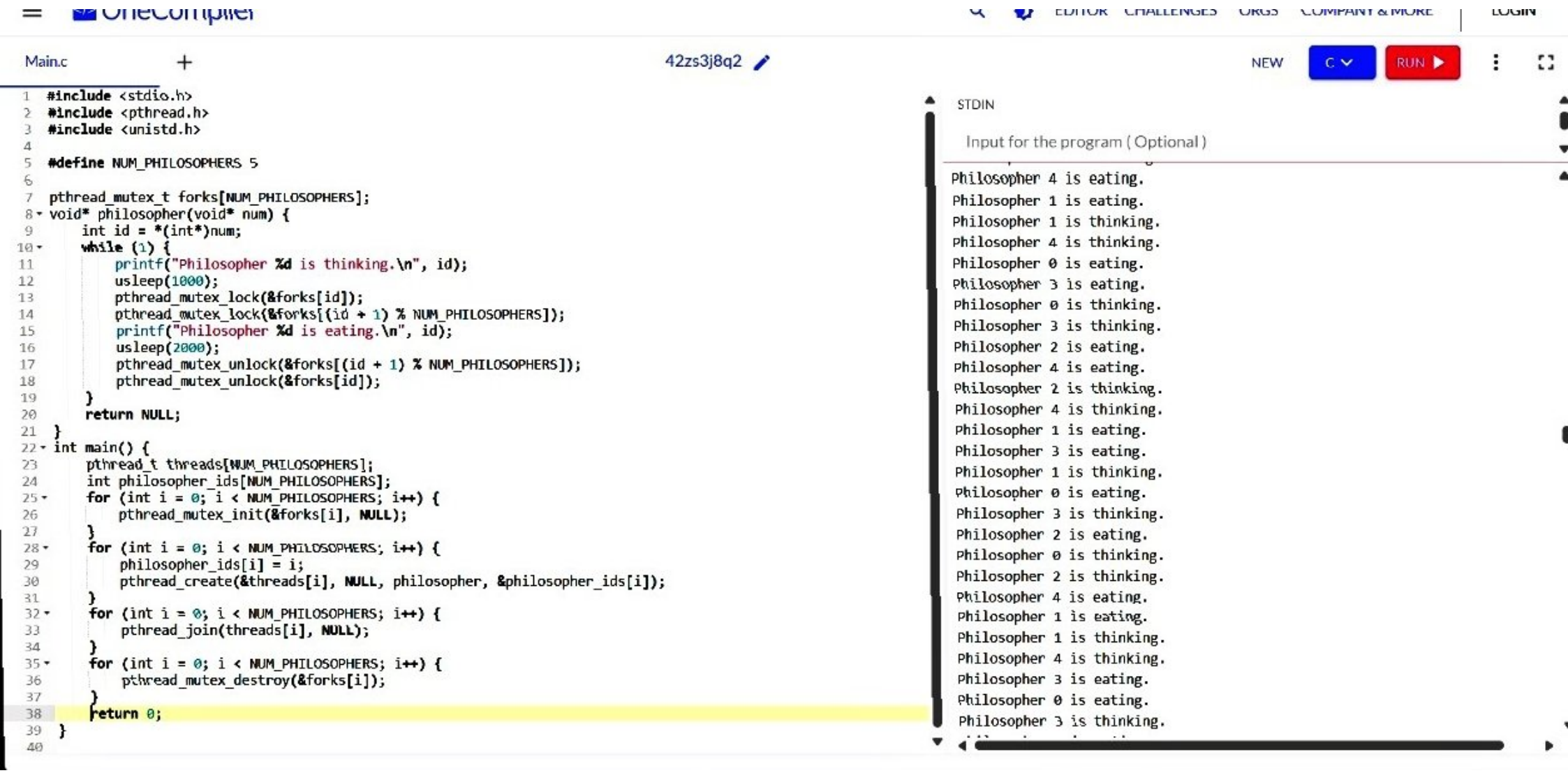
```
                printf("P%d ", proc[i].id);
                proc[i].remainingTime = 0;
                completed++;
            }
        }
    }
    printf("\nTotal time taken: %d\n", time);
}
int main() {
    struct Process proc[10];
    int n, timeQuantum;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        proc[i].id = i + 1;
        printf("Enter Burst Time for Process %d: ", proc[i].id);
        scanf("%d", &proc[i].burstTime);
        proc[i].remainingTime = proc[i].burstTime;
    }
    printf("Enter Time Quantum: ");
    scanf("%d", &timeQuantum);
    roundRobin(proc, n, timeQuantum);
    return 0;
}
```


Discount
Programiz PRO
Run
Output
Enter number of processes: 3
Enter Burst Time for Process 1: 4
Enter Burst Time for Process 2:
6
Enter Burst Time for Process 3: 5
Enter Time Quantum: 34455
Process Execution Order:
P1 P2 P3
Total time taken: 15
=== Code Execution Successful ===


Online C Compiler - Programiz
programiz.com/c-programming/online-compiler/
BLACK FRIDAY SALE
Save 60% on PRO: Sale Ends Soon! Claim My Discount
Sale ends in 00d : 13hrs : 32mins : 02s
Programiz
C Online Compiler
VIVARA
SELECTION
Oportunidade única para aumentar sua coleção.
JOIAS QUE ENCANTAM
BLACK FRIDAY
CONFIRA
Programiz PRO
main.c
Share
Run
Output
Clear
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>
#include <unistd.h>
#define SHM_SIZE 1024
int main() {
int shmid;
char *str;
shmid = shmget(IPC_PRIVATE, SHM_SIZE, IPC_CREAT | 0666);
str = (char *)shmat(shmid, NULL, 0);
strcpy(str, "Hello from the first process!");
sleep(2);
printf("Data read from shared memory: %s\n", str);
shmdt(str);
shmctl(shmid, IPC_RMID, NULL);
return 0;
}
Data read from shared memory: Hello from the first process!
=== Code Execution Successful ===
79°F
Mostly cloudy
Search
ENG
IN
23:57
27-11-2024


Online C Compiler - Programiz
https://www.programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
Premium Coding Courses by Programiz
Programiz PRO
Programiz PRO
main.c
Share
Run
Output
Clear
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <unistd.h>
#define MAX_TEXT 512
struct message {
long msg_type;
char text[MAX_TEXT];
};
int main() {
key_t key = ftok("progfile", 65);
int msgid = msgget(key, 0666 | IPC_CREAT);
struct message msg;
msg.msg_type = 1;
strcpy(msg.text, "Hello from Process 1!");
msgsnd(msgid, &msg, sizeof(msg), 0);
printf("Process 1: Sent message: %s\n", msg.text);
msgrcv(msgid, &msg, sizeof(msg), 1, 0);
printf("Process 2: Received message: %s\n", msg.text);
msgctl(msgid, IPC_RMID, NULL);
return 0;
}
Process 1: Sent message: Hello from Process 1!
Process 2: Received message: Hello from Process 1!
=== Code Execution Successful ===
80°F
Mostly cloudy
Search
ENG
IN
16:17
28-11-2024


OS_LAB_MANUAL[1].pdf
42zry8jpd - C - OneCompiler
42zrycz63 - C - OneCompiler
https://onecompiler.com/c/42zrycz63
OneCompiler
EDITOR
CHALLENGES
ORGS
COMPANY & MORE
LOGIN
Main.c
42zrycz63
NEW
C
RUN
#include <stdio.h>
#include <pthread.h>
void* threadFunction(void* arg)
{
char* message = (char*)arg; printf("%s\n",message);
return NULL;
}
int main() {
pthread_t thread1, thread2;
char* message1 = "Hello from Thread 1!";char*
message2 = "Hello from Thread 2!";
pthread_create(&thread1, NULL, threadFunction, (void*)message1);
pthread_create(&thread2, NULL, threadFunction, (void*)message2);
pthread_join(thread1, NULL);
pthread_join(thread2, NULL);
return 0;
}
STDIN
Input for the program (Optional)
Output:
Hello from Thread 1!
Hello from Thread 2!
79°F
Mostly cloudy
Search
ENG
IN
20:15
01-12-2024


Main.c
42zs3j8q2
NEW
C
RUN
STDIN
Input for the program (Optional)
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_PHILOSOPHERS 5
pthread_mutex_t forks[NUM_PHILOSOPHERS];
void* philosopher(void* num) {
int id = *(int*)num;
while (1) {
printf("Philosopher %d is thinking.\n", id);
usleep(1000);
pthread_mutex_lock(&forks[id]);
pthread_mutex_lock(&forks[(id + 1) % NUM_PHILOSOPHERS]);
printf("Philosopher %d is eating.\n", id);
usleep(2000);
pthread_mutex_unlock(&forks[(id + 1) % NUM_PHILOSOPHERS]);
pthread_mutex_unlock(&forks[id]);
}
return NULL;
}
int main() {
pthread_t threads[NUM_PHILOSOPHERS];
int philosopher_ids[NUM_PHILOSOPHERS];
for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
pthread_mutex_init(&forks[i], NULL);
}
for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
philosopher_ids[i] = i;
pthread_create(&threads[i], NULL, philosopher, &philosopher_ids[i]);
}
for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
pthread_join(threads[i], NULL);
}
for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
pthread_mutex_destroy(&forks[i]);
}
return 0;
}
Philosopher 4 is eating.
Philosopher 1 is eating.
Philosopher 1 is thinking.
Philosopher 4 is thinking.
Philosopher 0 is eating.
Philosopher 3 is eating.
Philosopher 0 is thinking.
Philosopher 3 is thinking.
Philosopher 2 is eating.
Philosopher 4 is eating.
Philosopher 2 is thinking.
Philosopher 4 is thinking.
Philosopher 1 is eating.
Philosopher 3 is eating.
Philosopher 1 is thinking.
Philosopher 0 is eating.
Philosopher 3 is thinking.
Philosopher 2 is eating.
Philosopher 0 is thinking.
Philosopher 2 is thinking.
Philosopher 4 is eating.
Philosopher 1 is eating.
Philosopher 1 is thinking.
Philosopher 4 is thinking.
Philosopher 3 is eating.
Philosopher 0 is eating.
Philosopher 3 is thinking.

Programiz C Online Compiler

Programiz PRO >

main.c

Share Run

```c
#include <stdio.h>
#include <stdlib.h>
void firstFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];
    for (int i = 0; i < n; i++) {
        allocation[i] = -1;
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
                allocation[i] = j;
                blockSize[j] -= processSize[i];
                break;
            }
        }
    }
    printf("First Fit Allocation:\n");
    for (int i = 0; i < n; i++) {
        if (allocation[i] != -1) {
            printf("Process %d allocated to Block %d\n", i + 1,
                allocation[i] + 1);
        } else {
            printf("Process %d not allocated\n", i + 1);
        }
```

Output

Clear

```
First Fit Allocation:
Process 1 allocated to Block 2
Process 2 allocated to Block 5
Process 3 allocated to Block 2
Process 4 not allocated
Best Fit Allocation:
Process 1 allocated to Block 4
Process 2 not allocated
Process 3 allocated to Block 2
Process 4 not allocated
Worst Fit Allocation:
Process 1 not allocated
Process 2 not allocated
Process 3 allocated to Block 3
Process 4 not allocated


=== Code Execution Successful ===
```

Programiz C Online Compiler

Programiz PRO >

main.c

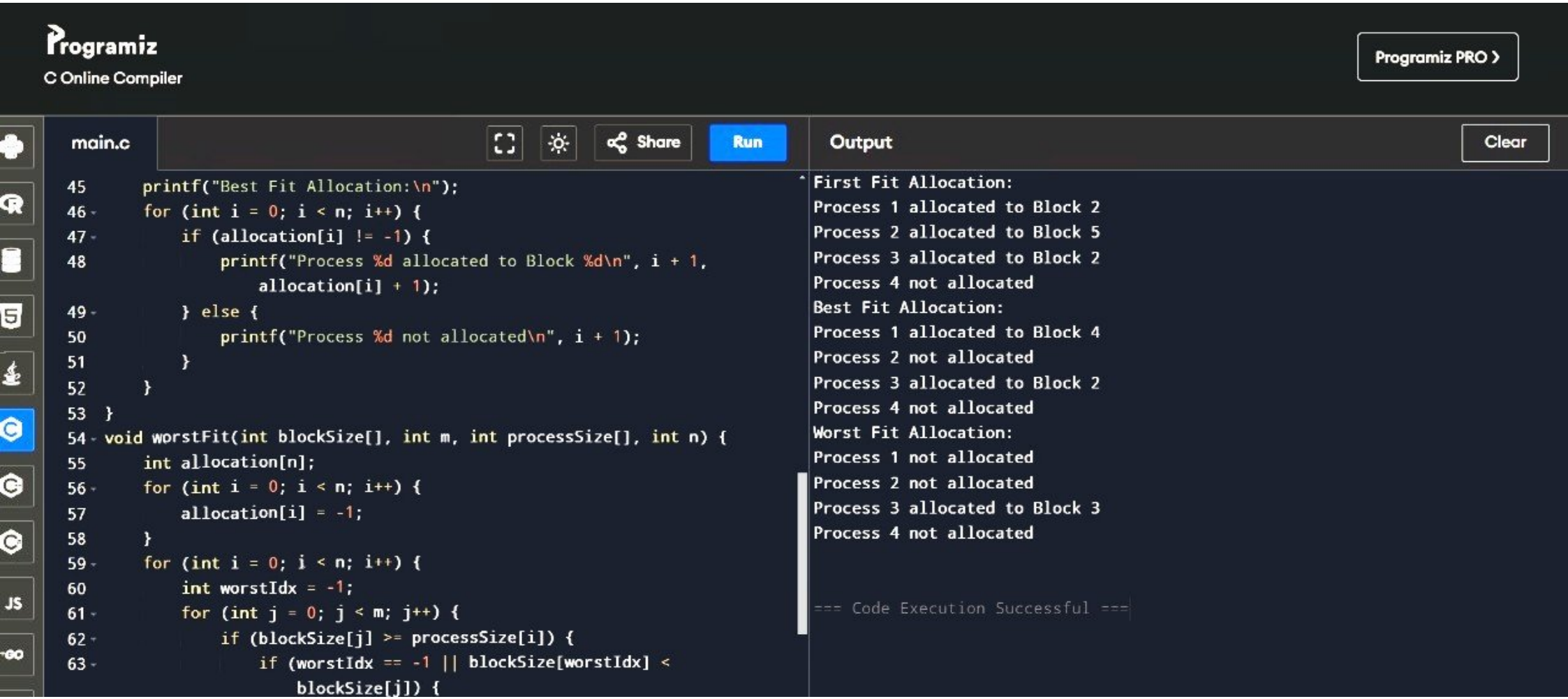
Share Run

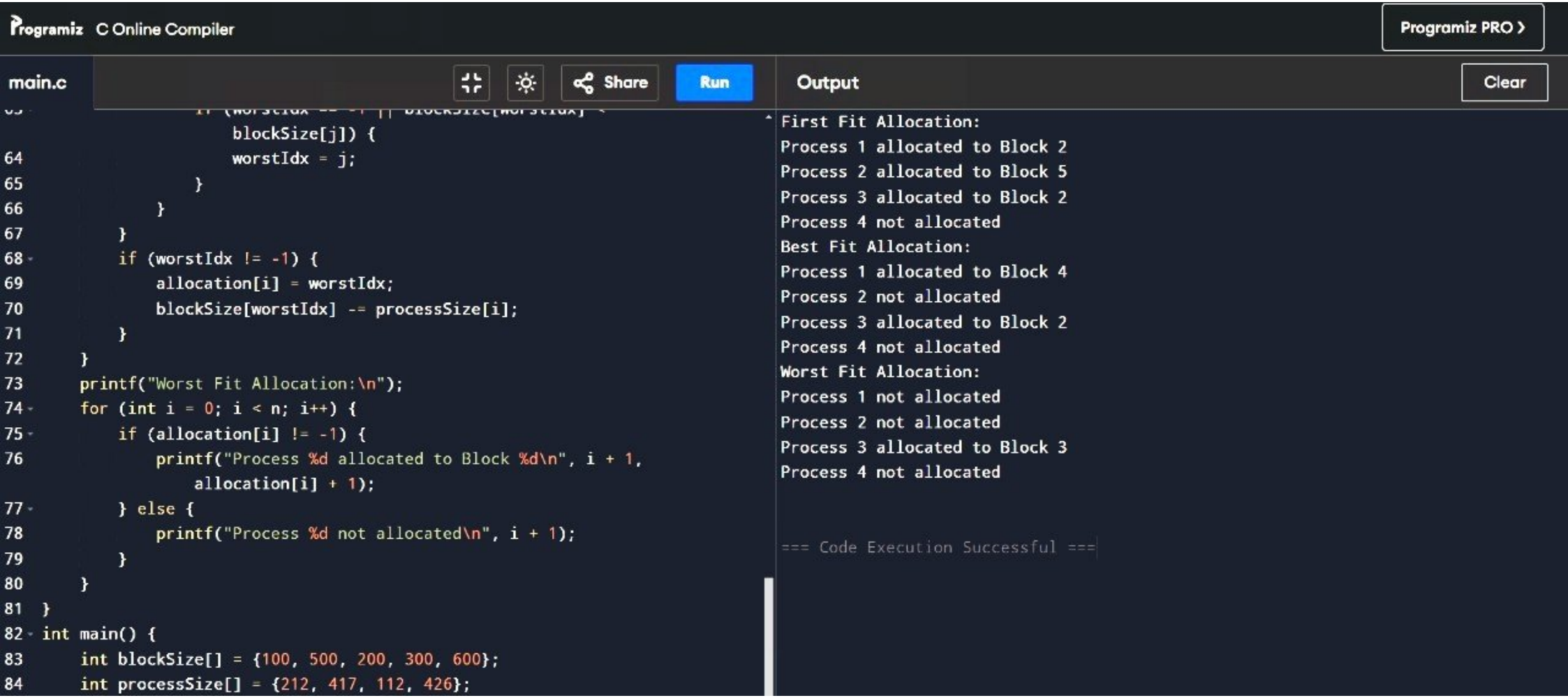
```
        }
    }
}
void bestFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];
    for (int i = 0; i < n; i++) {
        allocation[i] = -1;
    }
    for (int i = 0; i < n; i++) {
        int bestIdx = -1;
        for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
                if (bestIdx == -1 || blockSize[bestIdx] > blockSize[j]
                    ) {
                    bestIdx = j;
                }
            }
        }
        if (bestIdx != -1) {
            allocation[i] = bestIdx;
            blockSize[bestIdx] -= processSize[i];
        }
    }
```

Output

Clear

```
First Fit Allocation:
Process 1 allocated to Block 2
Process 2 allocated to Block 5
Process 3 allocated to Block 2
Process 4 not allocated
Best Fit Allocation:
Process 1 allocated to Block 4
Process 2 not allocated
Process 3 allocated to Block 2
Process 4 not allocated
Worst Fit Allocation:
Process 1 not allocated
Process 2 not allocated
Process 3 allocated to Block 3
Process 4 not allocated


=== Code Execution Successful ===
```


Programiz
C Online Compiler
Programiz PRO ›
main.c
Share
Run
Output
Clear
printf("Best Fit Allocation:\n");
for (int i = 0; i < n; i++) {
if (allocation[i] != -1) {
printf("Process %d allocated to Block %d\n", i + 1,
allocation[i] + 1);
} else {
printf("Process %d not allocated\n", i + 1);
}
}
}
void worstFit(int blockSize[], int m, int processSize[], int n) {
int allocation[n];
for (int i = 0; i < n; i++) {
allocation[i] = -1;
}
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[worstIdx] <
blockSize[j]) {
First Fit Allocation:
Process 1 allocated to Block 2
Process 2 allocated to Block 5
Process 3 allocated to Block 2
Process 4 not allocated
Best Fit Allocation:
Process 1 allocated to Block 4
Process 2 not allocated
Process 3 allocated to Block 2
Process 4 not allocated
Worst Fit Allocation:
Process 1 not allocated
Process 2 not allocated
Process 3 allocated to Block 3
Process 4 not allocated
=== Code Execution Successful ===


Programiz C Online Compiler
Programiz PRO
main.c
Share
Run
Output
Clear
blockSize[j]) {
worstIdx = j;
}
}
}
if (worstIdx != -1) {
allocation[i] = worstIdx;
blockSize[worstIdx] -= processSize[i];
}
}
printf("Worst Fit Allocation:\n");
for (int i = 0; i < n; i++) {
if (allocation[i] != -1) {
printf("Process %d allocated to Block %d\n", i + 1,
allocation[i] + 1);
} else {
printf("Process %d not allocated\n", i + 1);
}
}
}
int main() {
int blockSize[] = {100, 500, 200, 300, 600};
int processSize[] = {212, 417, 112, 426};
First Fit Allocation:
Process 1 allocated to Block 2
Process 2 allocated to Block 5
Process 3 allocated to Block 2
Process 4 not allocated
Best Fit Allocation:
Process 1 allocated to Block 4
Process 2 not allocated
Process 3 allocated to Block 2
Process 4 not allocated
Worst Fit Allocation:
Process 1 not allocated
Process 2 not allocated
Process 3 allocated to Block 3
Process 4 not allocated
=== Code Execution Successful ===

main.c

Share Run

```
        }
    }
    printf("Worst Fit Allocation:\n");
    for (int i = 0; i < n; i++) {
        if (allocation[i] != -1) {
            printf("Process %d allocated to Block %d\n", i + 1,
                allocation[i] + 1);
        } else {
            printf("Process %d not allocated\n", i + 1);
        }
    }
}
int main() {
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);
    firstFit(blockSize, m, processSize, n);
    bestFit(blockSize, m, processSize, n);
    worstFit(blockSize, m, processSize, n);
    return 0;
}

```

Output

Clear

```
First Fit Allocation:
Process 1 allocated to Block 2
Process 2 allocated to Block 5
Process 3 allocated to Block 2
Process 4 not allocated
Best Fit Allocation:
Process 1 allocated to Block 4
Process 2 not allocated
Process 3 allocated to Block 2
Process 4 not allocated
Worst Fit Allocation:
Process 1 not allocated
Process 2 not allocated
Process 3 allocated to Block 3
Process 4 not allocated


=== Code Execution Successful ===
```

main.c

Share | Run

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#define BUFFER_SIZE 4096
void copy(){
const char *sourcefile= "C:/Users/itssk/OneDrive/Desktop/sasi.txt";
const char *destination_file="C:/Users/itssk/OneDrive/Desktop/sk.txt";int
source_fd = open(sourcefile, O_RDONLY);
int dest_fd = open(destination_file, O_WRONLY | O_CREAT |O_TRUNC,0666);
char buffer[BUFFER_SIZE]; ssize_t
bytesRead, bytesWritten;
while ((bytesRead = read(source_fd, buffer, BUFFER_SIZE)) > 0) {
bytesWritten = write(dest_fd, buffer, bytesRead);
}
close(source_fd);
close(dest_fd);
printf("File copied successfully.\n");
}
void create()
{
char path[100];
FILE *fp; fp=fopen("C:/Users/itssk/OneDrive/Desktop/sasi.txt","w");
printf("file created successfully");
}
int main(){
int n;
printf("1. Create \t2. Copy \t3. Delete\nEnter your choice: " );
scanf("%d",&n);
switch(n){
case 1:
create(); break;
case 2:
copy();
break;
case 3:
remove("C:/Users/itssk/OneDrive/Desktop/sasi.txt");
printf("Deleted successfully");
}}
```

Output

```
1. Create   2. Copy     3. Delete
Enter your choice: 2
File copied successfully.


=== Code Execution Successful ===
```

Programiz C Online Compiler

Programiz PRO >

main.c

Share Run

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#define MAX_DIRS 10
#define MAX_FILES 10
#define DIR_NAME_LENGTH 50
#define FILE_NAME_LENGTH 50
typedef struct {
    char files[MAX_FILES][FILE_NAME_LENGTH];
    int file_count;
} SubDirectory;
typedef struct {
    char name[DIR_NAME_LENGTH];
    SubDirectory subdirs[MAX_DIRS];
    int subdir_count;
} MainDirectory;
void create_directory(MainDirectory *main_dir, const char *dir_name) {
    if (main_dir->subdir_count < MAX_DIRS) {
        strcpy(main_dir->subdirs[main_dir->subdir_count].files[0], dir_name);
        main_dir->subdirs[main_dir->subdir_count].file_count = 0;
        main_dir->subdir_count++;
        printf("Directory '%s' created successfully.\n", dir_name);
    } else {
        printf("Maximum number of directories reached.\n");
    }
}
void add_file(MainDirectory *main_dir, const char *dir_name, const char *file_name) {
    for (int i = 0; i < main_dir->subdir_count; i++) {
        if (strcmp(main_dir->subdirs[i].files[0], dir_name) == 0) {
            if (main_dir->subdirs[i].file_count < MAX_FILES) {
                strcpy(main_dir->subdirs[i].files[main_dir->subdirs[i].file_count + 1], file_name);
                main_dir->subdirs[i].file_count++;
                printf("File '%s' added to directory '%s'.\n", file_name, dir_name);
            } else {
                printf("Maximum number of files in directory '%s' reached.\n", dir_name);
            }
            return;
        }
    }
    printf("Directory '%s' not found.\n", dir_name);
}
void display_structure(MainDirectory *main_dir) {
    printf("Directory Structure:\n");
    for (int i = 0; i < main_dir->subdir_count; i++) {
        printf("Directory: %s\n", main_dir->subdirs[i].files[0]);
        for (int j = 1; j <= main_dir->subdirs[i].file_count; j++) {
            printf("  File: %s\n", main_dir->subdirs[i].files[j]);
        }
    }
}
int main() {
    MainDirectory main_dir = { .subdir_count = 0 };
    create_directory(&main_dir, "Documents");
```

Output

Clear

```
Directory 'Documents' created successfully.
Directory 'Pictures' created successfully.
File 'Resume.docx' added to directory 'Documents'.
File 'CoverLetter.docx' added to directory 'Documents'.
File 'Vacation.jpg' added to directory 'Pictures'.
Directory Structure:
Directory: Documents
  File: Resume.docx
  File: CoverLetter.docx
Directory: Pictures
  File: Vacation.jpg


=== Code Execution Successful ===
```

main.c

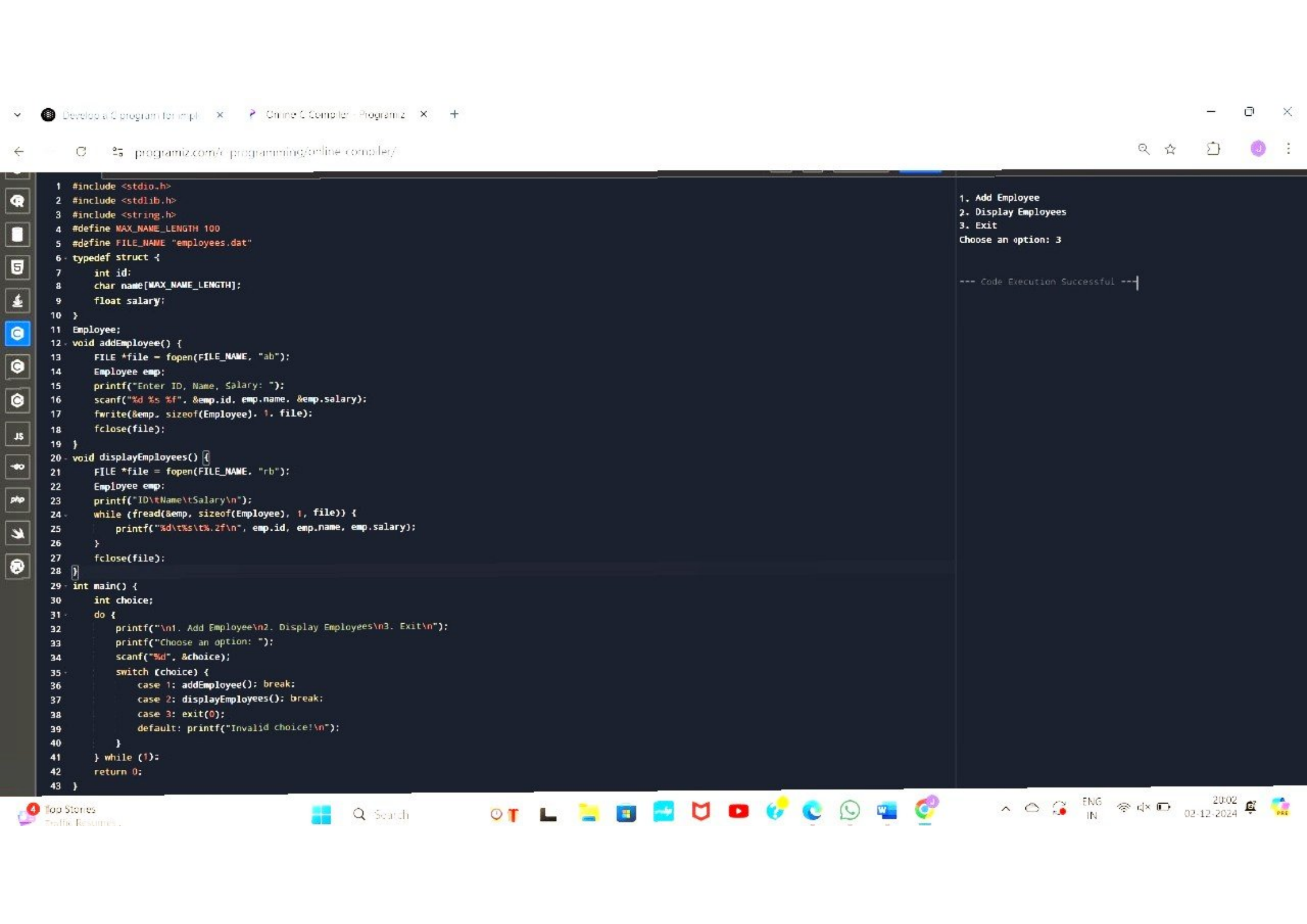
Share Run

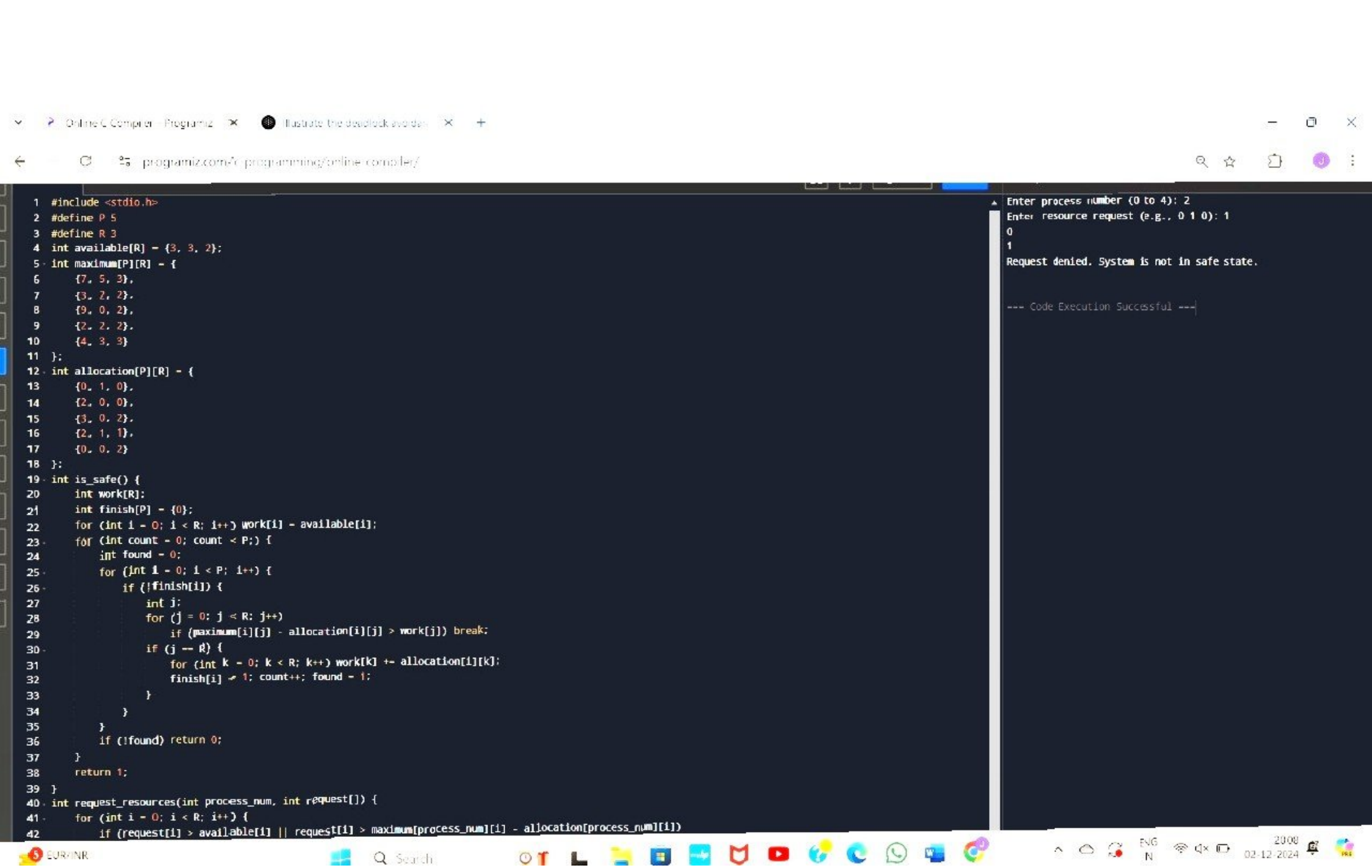
```c
    char files[MAX_FILES][FILE_NAME_LENGTH];
    int file_count;
} SubDirectory;
typedef struct {
    char name[DIR_NAME_LENGTH];
    SubDirectory subdirs[MAX_DIRS];
    int subdir_count;
} MainDirectory;
void create_directory(MainDirectory *main_dir, const char *dir_name) {
    if (main_dir->subdir_count < MAX_DIRS) {
        strcpy(main_dir->subdirs[main_dir->subdir_count].files[0], dir_name);
        main_dir->subdirs[main_dir->subdir_count].file_count = 0;
        main_dir->subdir_count++;
        printf("Directory '%s' created successfully.\n", dir_name);
    } else {
        printf("Maximum number of directories reached.\n");
    }
}
void add_file(MainDirectory *main_dir, const char *dir_name, const char *file_name) {
    for (int i = 0; i < main_dir->subdir_count; i++) {
        if (strcmp(main_dir->subdirs[i].files[0], dir_name) == 0) {
            if (main_dir->subdirs[i].file_count < MAX_FILES) {
                strcpy(main_dir->subdirs[i].files[main_dir->subdirs[i].file_count + 1], file_name);
                main_dir->subdirs[i].file_count++;
                printf("File '%s' added to directory '%s'.\n", file_name, dir_name);
            } else {
                printf("Maximum number of files in directory '%s' reached.\n", dir_name);
            }
            return;
        }
    }
    printf("Directory '%s' not found.\n", dir_name);
}
void display_structure(MainDirectory *main_dir) {
    printf("Directory Structure:\n");
    for (int i = 0; i < main_dir->subdir_count; i++) {
        printf("Directory: %s\n", main_dir->subdirs[i].files[0]);
        for (int j = 1; j <= main_dir->subdirs[i].file_count; j++) {
            printf("  File: %s\n", main_dir->subdirs[i].files[j]);
        }
    }
}
int main() {
    MainDirectory main_dir = { .subdir_count = 0 };
    create_directory(&main_dir, "Documents");
    create_directory(&main_dir, "Pictures");
    add_file(&main_dir, "Documents", "Resume.docx");
    add_file(&main_dir, "Documents", "CoverLetter.docx");
    add_file(&main_dir, "Pictures", "Vacation.jpg");
    display_structure(&main_dir);
    return 0;
}
```

Output

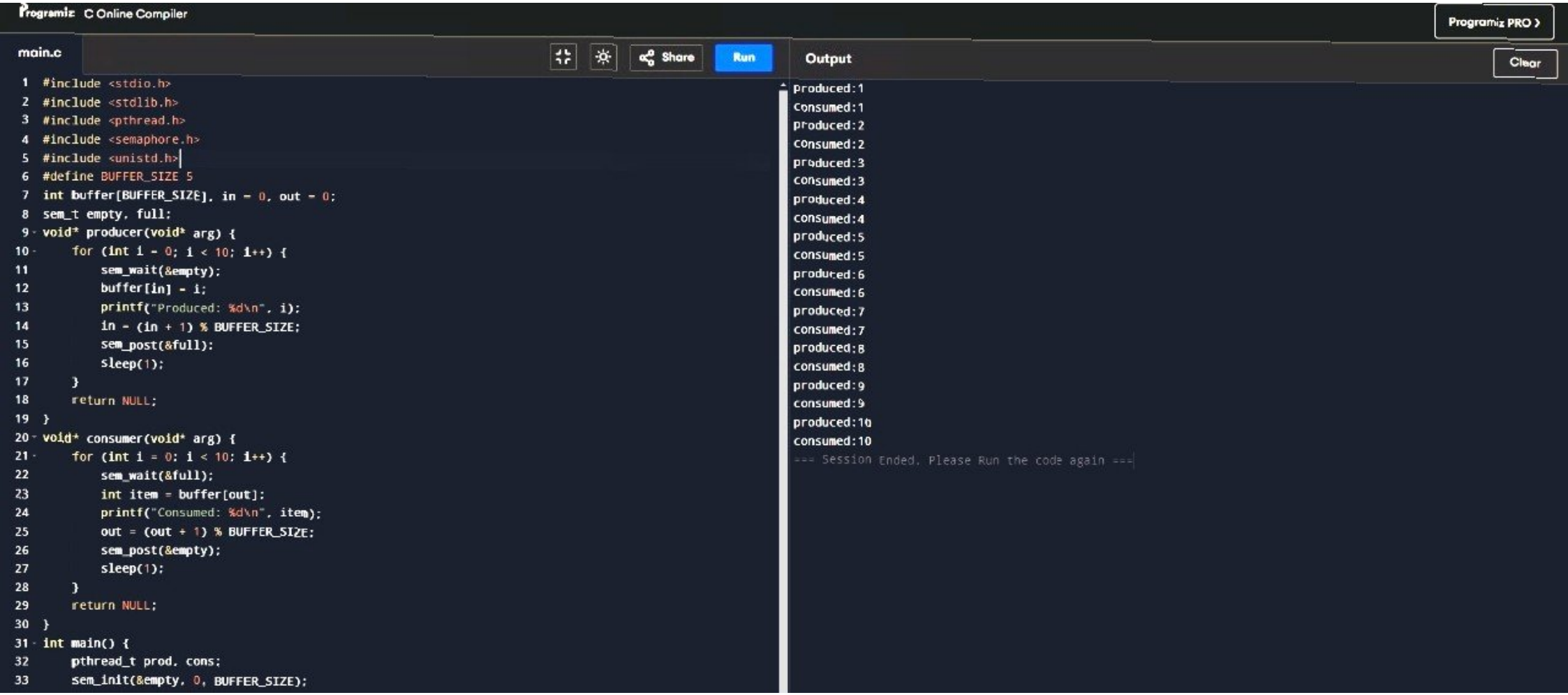
Clear

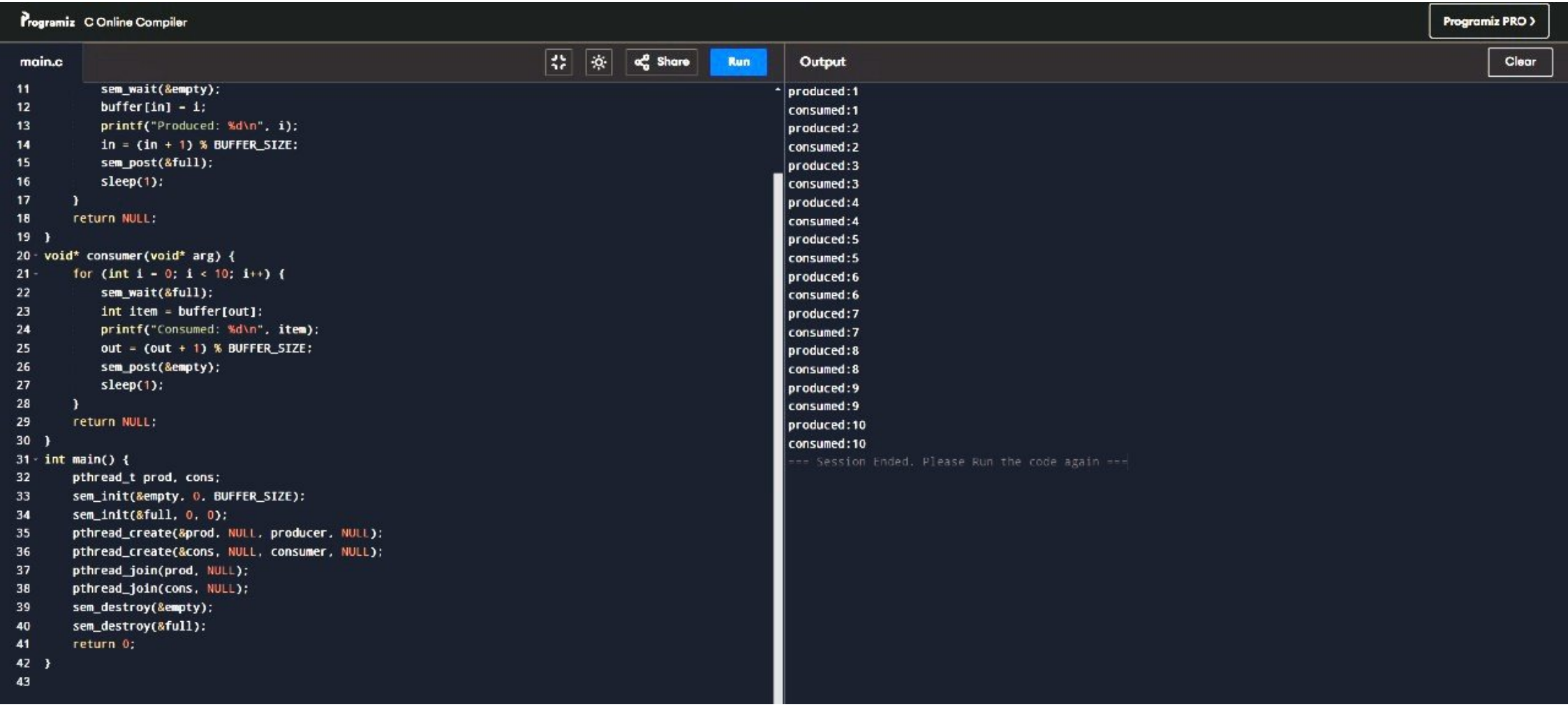
```
Directory 'Documents' created successfully.
Directory 'Pictures' created successfully.
File 'Resume.docx' added to directory 'Documents'.
File 'CoverLetter.docx' added to directory 'Documents'.
File 'Vacation.jpg' added to directory 'Pictures'.
Directory Structure:
Directory: Documents
  File: Resume.docx
  File: CoverLetter.docx
Directory: Pictures
  File: Vacation.jpg


=== Code Execution Successful ===
```


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_NAME_LENGTH 100
#define FILE_NAME "employees.dat"
typedef struct {
int id;
char name[MAX_NAME_LENGTH];
float salary;
}
Employee;
void addEmployee() {
FILE *file = fopen(FILE_NAME, "ab");
Employee emp;
printf("Enter ID, Name, Salary: ");
scanf("%d %s %f", &emp.id, emp.name, &emp.salary);
fwrite(&emp, sizeof(Employee), 1, file);
fclose(file);
}
void displayEmployees() {
FILE *file = fopen(FILE_NAME, "rb");
Employee emp;
printf("ID\tName\tSalary\n");
while (fread(&emp, sizeof(Employee), 1, file)) {
printf("%d\t%s\t%.2f\n", emp.id, emp.name, emp.salary);
}
fclose(file);
}
int main() {
int choice;
do {
printf("\n1. Add Employee\n2. Display Employees\n3. Exit\n");
printf("Choose an option: ");
scanf("%d", &choice);
switch (choice) {
case 1: addEmployee(); break;
case 2: displayEmployees(); break;
case 3: exit(0);
default: printf("Invalid choice!\n");
}
} while (1);
return 0;
}
1. Add Employee
2. Display Employees
3. Exit
Choose an option: 3
--- Code Execution Successful ---


#include <stdio.h>
#define P 5
#define R 3
int available[R] = {3, 3, 2};
int maximum[P][R] = {
{7, 5, 3},
{3, 2, 2},
{9, 0, 2},
{2, 2, 2},
{4, 3, 3}
};
int allocation[P][R] = {
{0, 1, 0},
{2, 0, 0},
{3, 0, 2},
{2, 1, 1},
{0, 0, 2}
};
int is_safe() {
int work[R];
int finish[P] = {0};
for (int i = 0; i < R; i++) work[i] = available[i];
for (int count = 0; count < P;) {
int found = 0;
for (int i = 0; i < P; i++) {
if (!finish[i]) {
int j;
for (j = 0; j < R; j++)
if (maximum[i][j] - allocation[i][j] > work[j]) break;
if (j == R) {
for (int k = 0; k < R; k++) work[k] += allocation[i][k];
finish[i] = 1; count++; found = 1;
}
}
}
if (!found) return 0;
}
return 1;
}
int request_resources(int process_num, int request[]) {
for (int i = 0; i < R; i++) {
if (request[i] > available[i] || request[i] > maximum[process_num][i] - allocation[process_num][i])
Enter process number (0 to 4): 2
Enter resource request (e.g., 0 1 0): 1
0
1
Request denied. System is not in safe state.
--- Code Execution Successful ---

```c
        if (!found) return 0;
    }
    return 1;
}
int request_resources(int process_num, int request[]) {
    for (int i = 0; i < R; i++) {
        if (request[i] > available[i] || request[i] > maximum[process_num][i] - allocation[process_num][i])
            return 0;
    }
    for (int i = 0; i < R; i++) {
        available[i] -= request[i];
        allocation[process_num][i] += request[i];
    }
    if (is_safe()) return 1;
    for (int i = 0; i < R; i++) {
        available[i] += request[i];
        allocation[process_num][i] -= request[i];
    }
    return 0;
}
int main() {
    int process_num, request[R];
    printf("Enter process number (0 to 4): ");
    scanf("%d", &process_num);
    printf("Enter resource request (e.g., 0 1 0): ");
    for (int i = 0; i < R; i++) scanf("%d", &request[i]);
    if (request_resources(process_num, request))
        printf("Request granted.\n");
    else
        printf("Request denied. System is not in safe state.\n");

    return 0;
}
```


Programiz C Online Compiler
Programiz PRO >
main.c
Share
Run
Output
Clear
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define BUFFER_SIZE 5
int buffer[BUFFER_SIZE], in = 0, out = 0;
sem_t empty, full;
void* producer(void* arg) {
for (int i = 0; i < 10; i++) {
sem_wait(&empty);
buffer[in] = i;
printf("Produced: %d\n", i);
in = (in + 1) % BUFFER_SIZE;
sem_post(&full);
sleep(1);
}
return NULL;
}
void* consumer(void* arg) {
for (int i = 0; i < 10; i++) {
sem_wait(&full);
int item = buffer[out];
printf("Consumed: %d\n", item);
out = (out + 1) % BUFFER_SIZE;
sem_post(&empty);
sleep(1);
}
return NULL;
}
int main() {
pthread_t prod, cons;
sem_init(&empty, 0, BUFFER_SIZE);
produced:1
consumed:1
produced:2
consumed:2
produced:3
consumed:3
produced:4
consumed:4
produced:5
consumed:5
produced:6
consumed:6
produced:7
consumed:7
produced:8
consumed:8
produced:9
consumed:9
produced:10
consumed:10
=== Session Ended. Please Run the code again ===


Programiz C Online Compiler
Programiz PRO
main.c
Share
Run
Output
Clear
sem_wait(&empty);
buffer[in] = i;
printf("Produced: %d\n", i);
in = (in + 1) % BUFFER_SIZE;
sem_post(&full);
sleep(1);
}
return NULL;
}
void* consumer(void* arg) {
for (int i = 0; i < 10; i++) {
sem_wait(&full);
int item = buffer[out];
printf("Consumed: %d\n", item);
out = (out + 1) % BUFFER_SIZE;
sem_post(&empty);
sleep(1);
}
return NULL;
}
int main() {
pthread_t prod, cons;
sem_init(&empty, 0, BUFFER_SIZE);
sem_init(&full, 0, 0);
pthread_create(&prod, NULL, producer, NULL);
pthread_create(&cons, NULL, consumer, NULL);
pthread_join(prod, NULL);
pthread_join(cons, NULL);
sem_destroy(&empty);
sem_destroy(&full);
return 0;
}
produced:1
consumed:1
produced:2
consumed:2
produced:3
consumed:3
produced:4
consumed:4
produced:5
consumed:5
produced:6
consumed:6
produced:7
consumed:7
produced:8
consumed:8
produced:9
consumed:9
produced:10
consumed:10
--- Session Ended. Please Run the code again ---

OS_LAB_MANUAL[1].pdf | 42zuxvb3u - C - OneCompiler

https://onecompiler.com/c/42zuxvb3u

OneCompiler

EDITOR CHALLENGES ORGS COMPANY & MORE LOGIN

Main.c

42zuxvb3u

NEW C RUN

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t mutex, writeBlock;
int data = 0, readersCount = 0;
void *reader(void *arg) {
    for (int i = 0; i < 5; i++) {
        sem_wait(&mutex);
        readersCount++;
        if (readersCount == 1) sem_wait(&writeBlock);
        sem_post(&mutex);
        printf("Reader reads data: %d\n", data);
        usleep(100);
        sem_wait(&mutex);
        readersCount--;
        if (readersCount == 0) sem_post(&writeBlock);
        sem_post(&mutex);
        usleep(100);
    }
    return NULL;
}
void *writer(void *arg) {
    for (int i = 0; i < 5; i++) {
        sem_wait(&writeBlock);
        data++;
        printf("Writer writes data: %d\n", data);
        sem_post(&writeBlock);
        usleep(100);
    }
    return NULL;
}
int main() {
    pthread_t readers[3], writerThread;
    sem_init(&mutex, 0, 1);
    sem_init(&writeBlock, 0, 1);
    for (int i = 0; i < 3; i++) pthread_create(&readers[i], NULL, reader, NULL);
    pthread_create(&writerThread, NULL, writer, NULL);
    for (int i = 0; i < 3; i++) pthread_join(readers[i], NULL);
    pthread_join(writerThread, NULL);
    sem_destroy(&mutex);
    sem_destroy(&writeBlock);
    return 0;
}
```

STDIN

Input for the program (Optional)

Output:

```
Reader reads data: 0
Reader reads data: 0
Reader reads data: 0
Writer writes data: 1
Reader reads data: 1
Reader reads data: 1
Reader reads data: 1
Writer writes data: 2
Reader reads data: 2
Reader reads data: 2
Reader reads data: 2
Writer writes data: 3
Reader reads data: 3
Reader reads data: 3
Reader reads data: 3
Writer writes data: 4
Reader reads data: 4
Reader reads data: 4
Reader reads data: 4
Writer writes data: 5
```

28°C Mostly cloudy

Search

ENG IN 20:19 02-12-2024

OS_LAB_MANUAL[1].pdf | 42zuxvb3u - C - OneCompiler

https://onecompiler.com/c/42zuxvb3u

OneCompiler

EDITOR C

Main.c

42zuxvb3u

```c
#include <stdio.h>
#include <pthread.h>
int counter = 0;
pthread_mutex_t mutex;
void *threadFunction(void *arg)
{
int i;
for (i = 0; i < 1000000; ++i)
{ }
return NULL;
}
int main()
{
pthread_mutex_init(&mutex, NULL);
pthread_t thread1, thread2;
pthread_create(&thread1, NULL, threadFunction, NULL);
pthread_create(&thread2, NULL, threadFunction, NULL);
pthread_join(thread1, NULL);
pthread_join(thread2, NULL);
pthread_mutex_destroy(&mutex);
printf("Final counter value: %d\n", counter);
return 0;
}

```

STDIN

20000000

Output:

```
Final counter value: 0
```

28°C Mostly cloudy

Search

ENG IN

20:26 02-12-2024

OS_LAB_MANUAL[1].pdf | 42zuxvb3u - C - OneCompiler

https://onecompiler.com/c/42zuxvb3u

OneCompiler

EDITOR C

Main.c

42zuxvb3u

```c
#include <stdio.h>
#include <pthread.h>
int counter = 0;
pthread_mutex_t mutex;
void *threadFunction(void *arg)
{
int i;
for (i = 0; i < 1000000; ++i)
{ }
return NULL;
}
int main()
{
pthread_mutex_init(&mutex, NULL);
pthread_t thread1, thread2;
pthread_create(&thread1, NULL, threadFunction, NULL);
pthread_create(&thread2, NULL, threadFunction, NULL);
pthread_join(thread1, NULL);
pthread_join(thread2, NULL);
pthread_mutex_destroy(&mutex);
printf("Final counter value: %d\n", counter);
return 0;
}

```

STDIN

20000000

Output:

```
Final counter value: 0
```

28°C Mostly cloudy

Search

ENG IN

20:26 02-12-2024