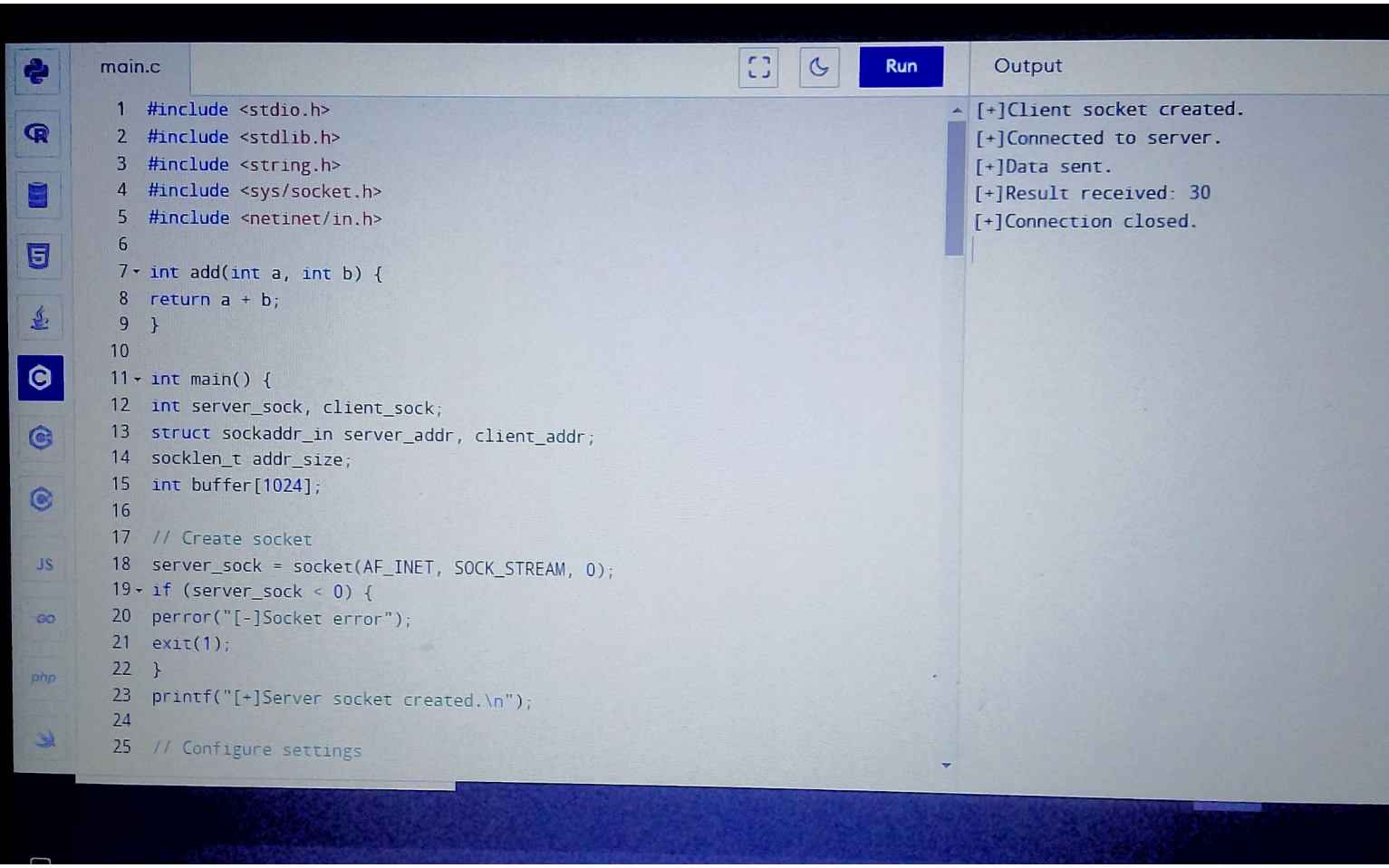
Main.java

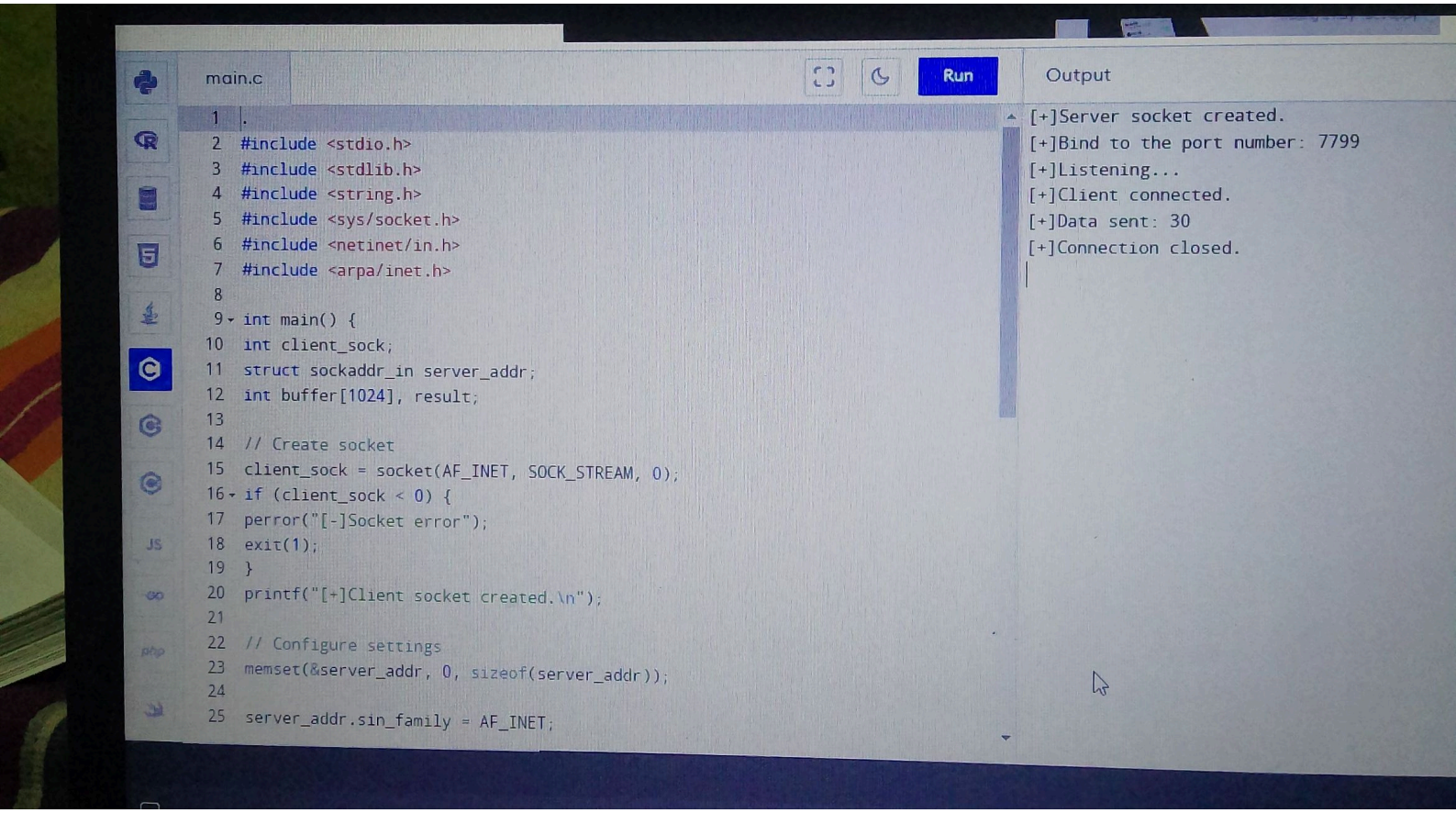
Run

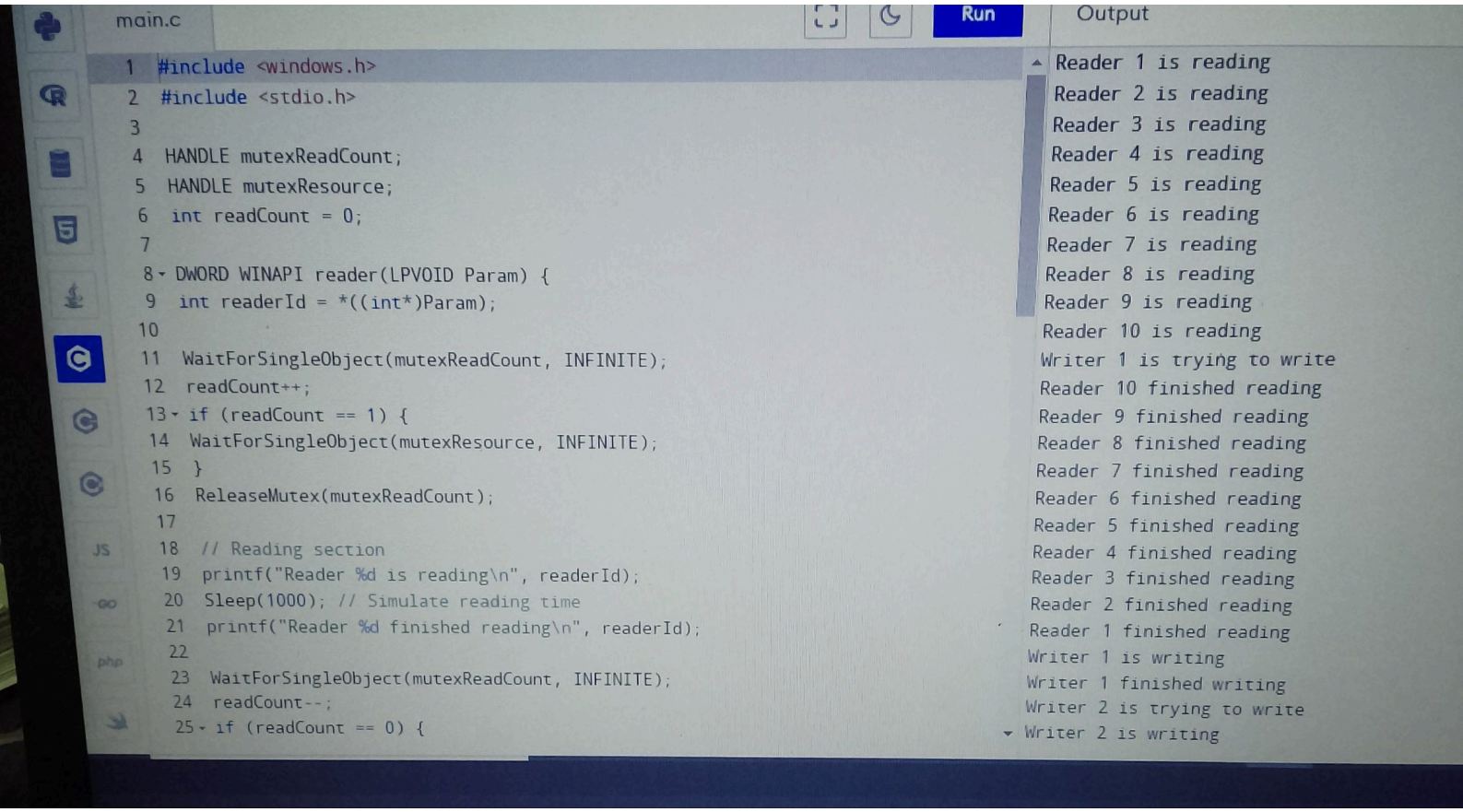
Output

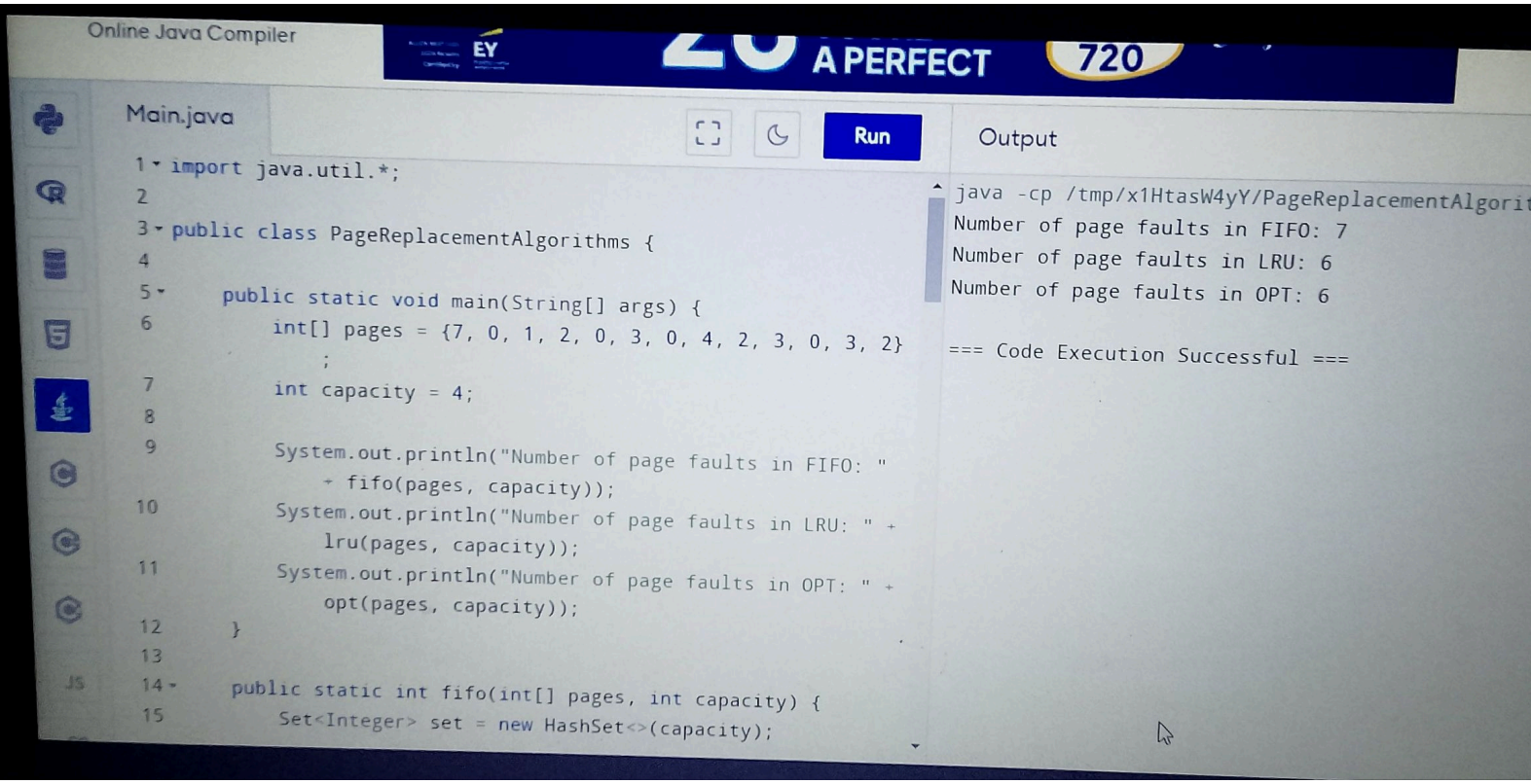
```
1  import java.util.LinkedList;
2  import java.util.Queue;
3  import java.util.concurrent.Semaphore;
4
5  class ProducerConsumer {
6      public static void main(String[] args) {
7          // Shared buffer
8          Buffer buffer = new Buffer(5);
9
10         // Create producer and consumer threads
11         Thread producerThread = new Thread(new Producer
               (buffer));
12         Thread consumerThread = new Thread(new Consumer
               (buffer));
13
14         // Start the threads
15         producerThread.start();
16         consumerThread.start();
17     }
```

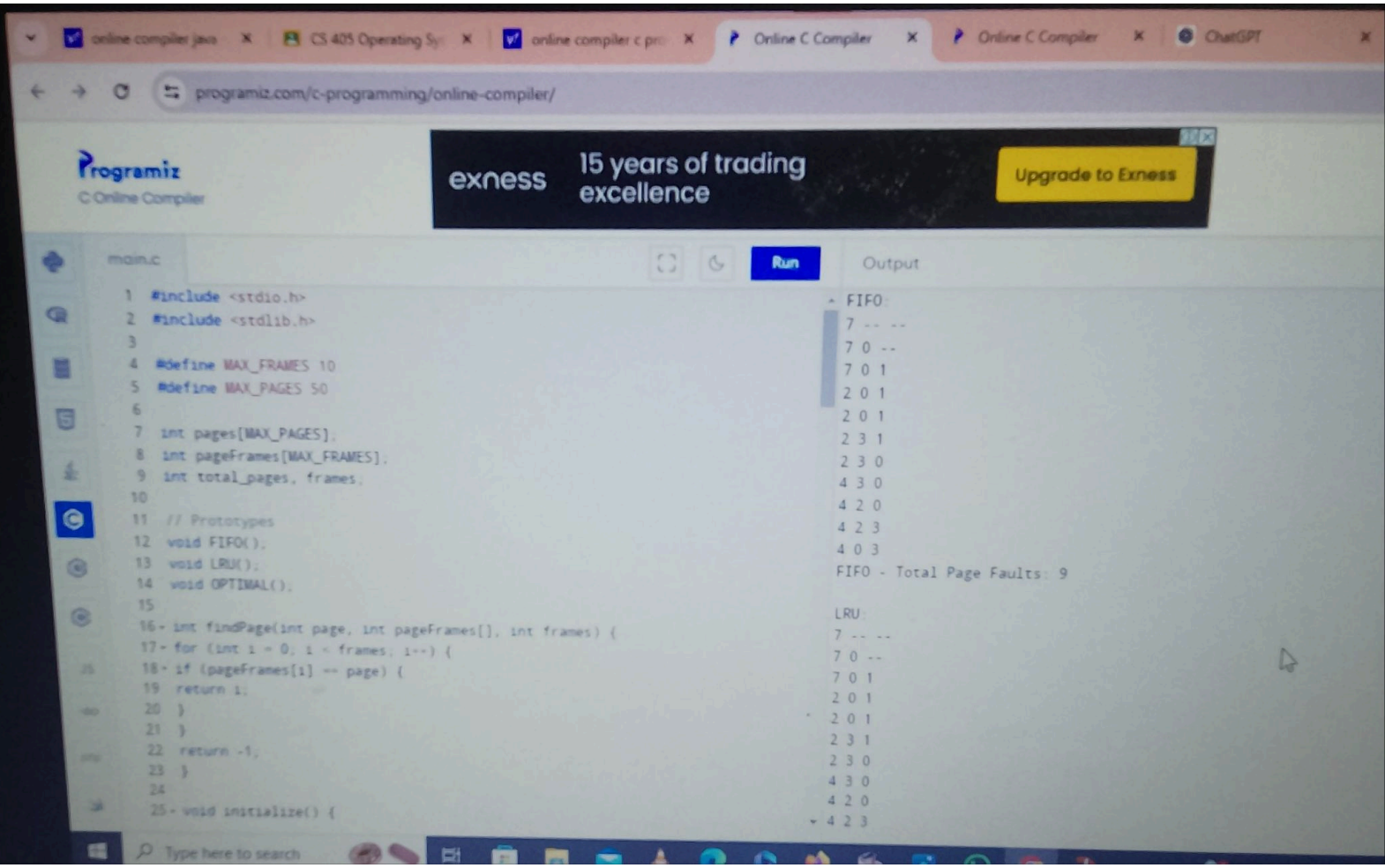
```
java -cp /tmp/zdoxqc4U9h/ProducerConsumer
Produced: 0
Consumed: 0
Produced: 1
Consumed: 1
Produced: 2
Consumed: 2
Produced: 3
Produced: 4
Consumed: 3
Produced: 5
Consumed: 4
Produced: 6
Produced: 7
Consumed: 5
Produced: 8
Consumed: 6
Produced: 9
Produced: 10
```

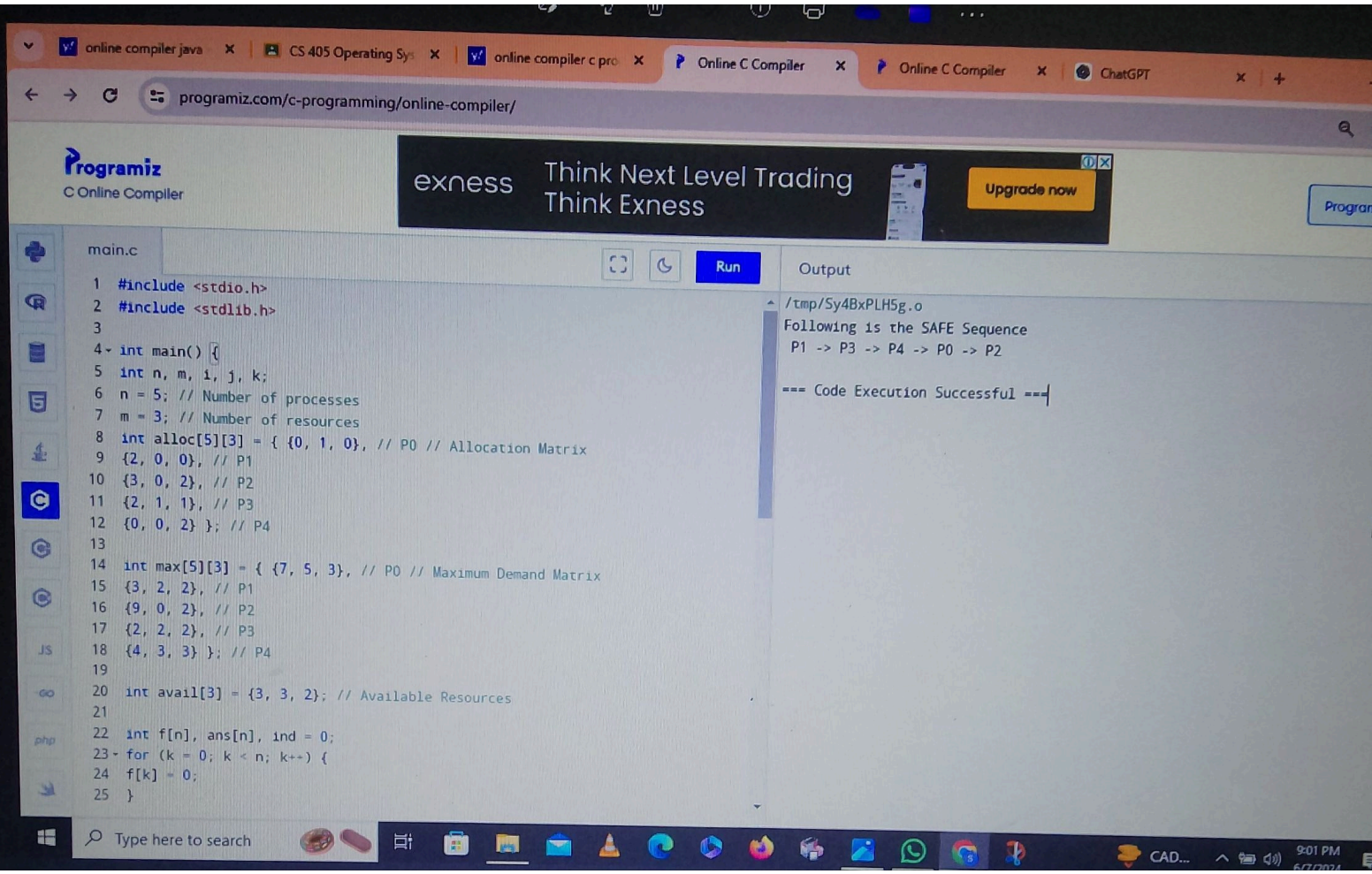

main.c
Run
Output
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
int add(int a, int b) {
return a + b;
}
int main() {
int server_sock, client_sock;
struct sockaddr_in server_addr, client_addr;
socklen_t addr_size;
int buffer[1024];
// Create socket
server_sock = socket(AF_INET, SOCK_STREAM, 0);
if (server_sock < 0) {
perror("[-]Socket error");
exit(1);
}
printf("[+]Server socket created.\n");
// Configure settings
[+]Client socket created.
[+]Connected to server.
[+]Data sent.
[+]Result received: 30
[+]Connection closed.

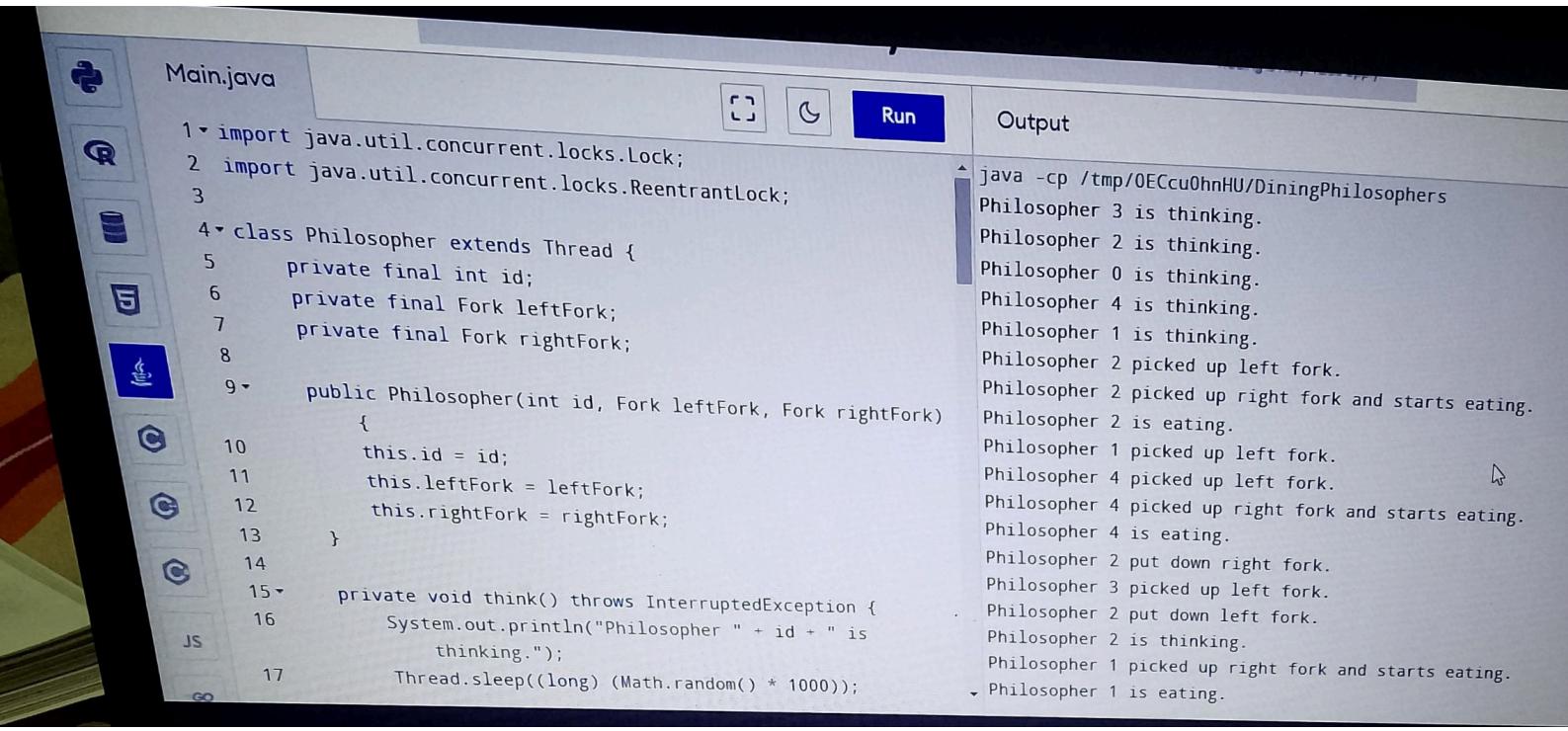

main.c
Run
Output
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
int main() {
int client_sock;
struct sockaddr_in server_addr;
int buffer[1024], result;
// Create socket
client_sock = socket(AF_INET, SOCK_STREAM, 0);
if (client_sock < 0) {
perror("[-]Socket error");
exit(1);
}
printf("[+]Client socket created.\n");
// Configure settings
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
[+]Server socket created.
[+]Bind to the port number: 7799
[+]Listening...
[+]Client connected.
[+]Data sent: 30
[+]Connection closed.


main.c
Run
Output
1 #include <windows.h>
2 #include <stdio.h>
3
4 HANDLE mutexReadCount;
5 HANDLE mutexResource;
6 int readCount = 0;
7
8 DWORD WINAPI reader(LPVOID Param) {
9 int readerId = *((int*)Param);
10
11 WaitForSingleObject(mutexReadCount, INFINITE);
12 readCount++;
13 if (readCount == 1) {
14 WaitForSingleObject(mutexResource, INFINITE);
15 }
16 ReleaseMutex(mutexReadCount);
17
18 // Reading section
19 printf("Reader %d is reading\n", readerId);
20 Sleep(1000); // Simulate reading time
21 printf("Reader %d finished reading\n", readerId);
22
23 WaitForSingleObject(mutexReadCount, INFINITE);
24 readCount--;
25 if (readCount == 0) {
Reader 1 is reading
Reader 2 is reading
Reader 3 is reading
Reader 4 is reading
Reader 5 is reading
Reader 6 is reading
Reader 7 is reading
Reader 8 is reading
Reader 9 is reading
Reader 10 is reading
Writer 1 is trying to write
Reader 10 finished reading
Reader 9 finished reading
Reader 8 finished reading
Reader 7 finished reading
Reader 6 finished reading
Reader 5 finished reading
Reader 4 finished reading
Reader 3 finished reading
Reader 2 finished reading
Reader 1 finished reading
Writer 1 is writing
Writer 1 finished writing
Writer 2 is trying to write
Writer 2 is writing


Online Java Compiler
EY
A PERFECT
720
Main.java
Run
Output
1 import java.util.*;
2
3 public class PageReplacementAlgorithms {
4
5 public static void main(String[] args) {
6 int[] pages = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2}
;
7 int capacity = 4;
8
9 System.out.println("Number of page faults in FIFO: "
+ fifo(pages, capacity));
10 System.out.println("Number of page faults in LRU: " +
lru(pages, capacity));
11 System.out.println("Number of page faults in OPT: " +
opt(pages, capacity));
12 }
13
14 public static int fifo(int[] pages, int capacity) {
15 Set<Integer> set = new HashSet<>(capacity);
java -cp /tmp/x1HtasW4yY/PageReplacementAlgorit
Number of page faults in FIFO: 7
Number of page faults in LRU: 6
Number of page faults in OPT: 6
=== Code Execution Successful ===


online compiler java
CS 405 Operating Sy
online compiler c pro
Online C Compiler
Online C Compiler
ChatGPT
programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
exness
15 years of trading excellence
Upgrade to Exness
main.c
Run
Output
#include <stdio.h>
#include <stdlib.h>
#define MAX_FRAMES 10
#define MAX_PAGES 50
int pages[MAX_PAGES];
int pageFrames[MAX_FRAMES];
int total_pages, frames;
// Prototypes
void FIFO();
void LRU();
void OPTIMAL();
int findPage(int page, int pageFrames[], int frames) {
for (int i = 0; i < frames; i++) {
if (pageFrames[i] == page) {
return i;
}
}
return -1;
}
void initialize() {
FIFO:
7 -- --
7 0 --
7 0 1
2 0 1
2 0 1
2 3 1
2 3 0
4 3 0
4 2 0
4 2 3
4 0 3
FIFO - Total Page Faults: 9
LRU:
7 -- --
7 0 --
7 0 1
2 0 1
2 0 1
2 3 1
2 3 0
4 3 0
4 2 0
4 2 3
Type here to search


online compiler java
CS 405 Operating Sys
online compiler c pro
Online C Compiler
Online C Compiler
ChatGPT
programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
exness
Think Next Level Trading
Think Exness
Upgrade now
Programiz
main.c
Run
Output
#include <stdio.h>
#include <stdlib.h>
int main() {
int n, m, i, j, k;
n = 5; // Number of processes
m = 3; // Number of resources
int alloc[5][3] = { {0, 1, 0}, // P0 // Allocation Matrix
{2, 0, 0}, // P1
{3, 0, 2}, // P2
{2, 1, 1}, // P3
{0, 0, 2} }; // P4
int max[5][3] = { {7, 5, 3}, // P0 // Maximum Demand Matrix
{3, 2, 2}, // P1
{9, 0, 2}, // P2
{2, 2, 2}, // P3
{4, 3, 3} }; // P4
int avail[3] = {3, 3, 2}; // Available Resources
int f[n], ans[n], ind = 0;
for (k = 0; k < n; k++) {
f[k] = 0;
}
/tmp/Sy4BxPLH5g.o
Following is the SAFE Sequence
P1 -> P3 -> P4 -> P0 -> P2
=== Code Execution Successful ===
Type here to search
CAD...
9:01 PM


Main.java
Run
Output
1 import java.util.concurrent.locks.Lock;
2 import java.util.concurrent.locks.ReentrantLock;
3
4 class Philosopher extends Thread {
5 private final int id;
6 private final Fork leftFork;
7 private final Fork rightFork;
8
9 public Philosopher(int id, Fork leftFork, Fork rightFork)
{
10 this.id = id;
11 this.leftFork = leftFork;
12 this.rightFork = rightFork;
13 }
14
15 private void think() throws InterruptedException {
16 System.out.println("Philosopher " + id + " is
thinking.");
17 Thread.sleep((long) (Math.random() * 1000));
java -cp /tmp/OECcuOhnHU/DiningPhilosophers
Philosopher 3 is thinking.
Philosopher 2 is thinking.
Philosopher 0 is thinking.
Philosopher 4 is thinking.
Philosopher 1 is thinking.
Philosopher 2 picked up left fork.
Philosopher 2 picked up right fork and starts eating.
Philosopher 2 is eating.
Philosopher 1 picked up left fork.
Philosopher 4 picked up left fork.
Philosopher 4 picked up right fork and starts eating.
Philosopher 4 is eating.
Philosopher 2 put down right fork.
Philosopher 3 picked up left fork.
Philosopher 2 put down left fork.
Philosopher 2 is thinking.
Philosopher 1 picked up right fork and starts eating.
Philosopher 1 is eating.