Name: Jap Purohit
Enrollment Number: AU1940109

Q1.Print the following Pattern : A 1 a B 2 b C 3 c ... Y 25 y Z 26 z

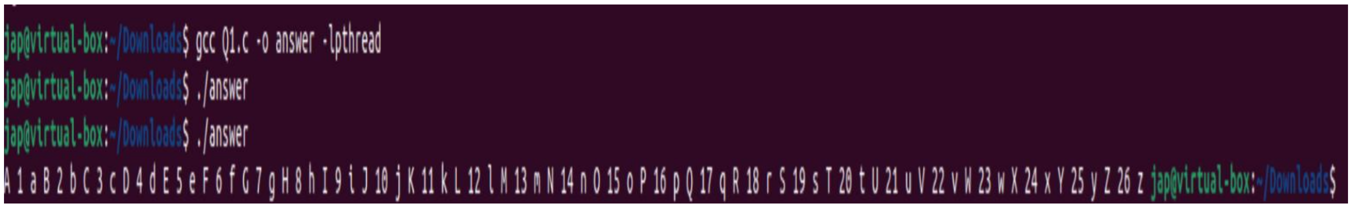
Using any one of the following concepts

a. Multiprocesses (Hint: using 3 child processes)

b. Multithreads (Hint: using 3 Threads)

Ans: For this program, I have taken one common data known as a flag which will help to determine which thing to print for the pattern. And took another three variables to maintain the record of upper case, lower case, and number pattern. For the execution of this program, I created three separate threads to print the character

Output:



```
jap@virtual-box:~/Downloads$ gcc Q1.c -o answer -lpthread
jap@virtual-box:~/Downloads$ ./answer
jap@virtual-box:~/Downloads$ ./answer
A 1 a B 2 b C 3 c D 4 d E 5 e F 6 f G 7 g H 8 h I 9 i J 10 j K 11 k L 12 l M 13 m N 14 n O 15 o P 16 p Q 17 q R 18 r S 19 s T 20 t U 21 u V 22 v W 23 w X 24 x Y 25 y Z 26 z jap@virtual-box:~/Downloads$
```

Q2. Describe and implement any one of the following

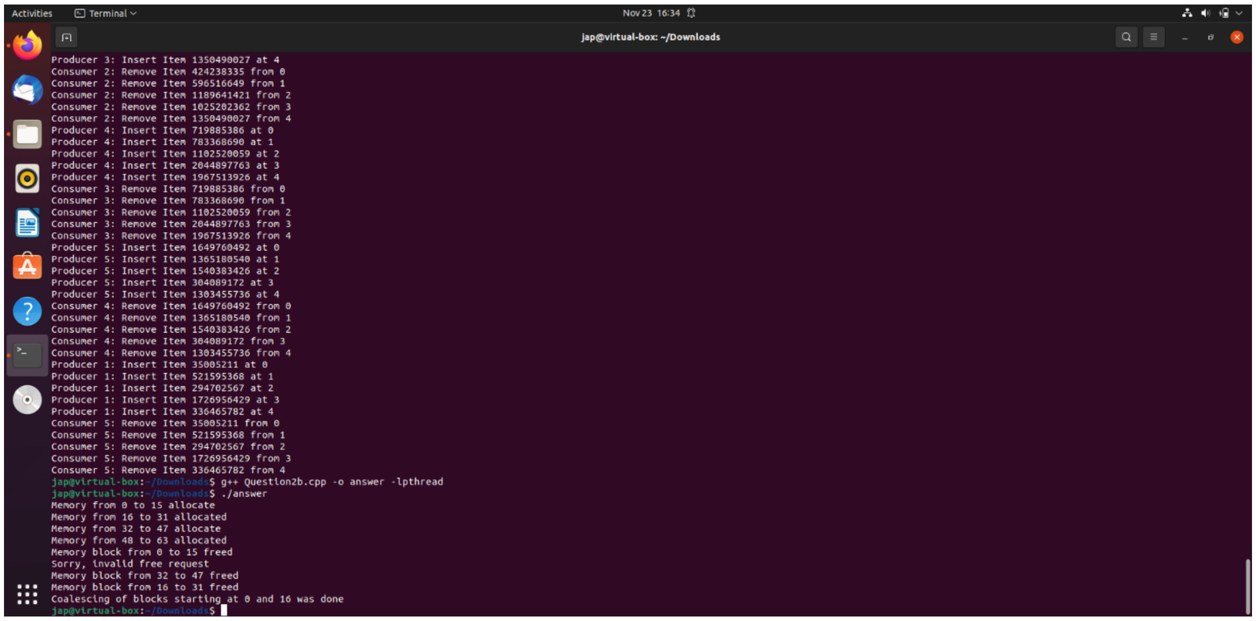
a. **Describe Buddy's Algorithm for Memory Allocation and Deallocation along with an example and implement it in C or C++.**

Ans. The buddy memory allocation approach is a memory allocation mechanism that splits memory into divisions in order to provide the best feasible response to a memory request. This system divides memory in half in order to find the optimal match. There are several types of buddy systems; the simplest and most prevalent are those in which each block is split into two smaller blocks. In this system, every memory block has an order, which is a number ranging from 0 to a set upper limit. A block of order n is proportionate to 2n in size, therefore the blocks are precise twice the size of blocks of lower order. Because all buddies are aligned on memory address boundaries that are powers of two, address calculation is straightforward with power-of-two block sizes. When a bigger block is split, it generates two smaller blocks, each of which becomes a unique friend for the other. A split block can only be combined with its one-of-a-kind buddy block, which reassembles the bigger block from which it was separated.

To begin, the smallest feasible block, i.e. the smallest memory block that may be allocated, is determined. There would be a lot of memory and computational overhead for the system to keep track of which sections of the memory are allocated and unallocated if there was no lower limit at all (e.g., bit-sized allocations were feasible). However, a low limit may be desired in order to reduce average memory waste per allocation (for allocations that are not multiples of the smallest block in size). The lower limit is often minimal enough to reduce average wasted space per allocation while being large enough to avoid undue overhead. All higher orders are represented as power-of-two multiples of the lowest block size, which is subsequently used as the size of an order-0 block.

The programmer must then choose, or develop code to achieve, the greatest possible order that will fit in the remaining memory space. The biggest block size may not cover the whole memory of a computer system if the total accessible memory is not a power-of-two multiple of the minimum block size. The top restriction on the order would be 8 if the system had 2000 K of physical memory and the order-0 block size was 4 K, because an order-8 block (256 order-0 blocks, 1024 K) is the largest block that will fit in memory. As a result, allocating the whole physical memory in a single chunk is impracticable; the remaining 976 K of memory must be allocated in smaller chunks.

Output



```
Producer 3: Insert Item 1350490027 at 4
Consumer 2: Remove Item 424238335 from 0
Consumer 2: Remove Item 596516649 from 1
Consumer 2: Remove Item 1189641421 from 2
Consumer 2: Remove Item 1025202362 from 3
Consumer 2: Remove Item 1350490027 from 4
Producer 4: Insert Item 719885386 at 0
Producer 4: Insert Item 783368690 at 1
Producer 4: Insert Item 1102520059 at 2
Producer 4: Insert Item 2044897763 at 3
Producer 4: Insert Item 1967513926 at 4
Consumer 3: Remove Item 719885386 from 0
Consumer 3: Remove Item 783368690 from 1
Consumer 3: Remove Item 1102520059 from 2
Consumer 3: Remove Item 2044897763 from 3
Consumer 3: Remove Item 1967513926 from 4
Producer 5: Insert Item 1649760492 at 0
Producer 5: Insert Item 1365180540 at 1
Producer 5: Insert Item 1540383426 at 2
Producer 5: Insert Item 304089172 at 3
Producer 5: Insert Item 1303455736 at 4
Consumer 4: Remove Item 1649760492 from 0
Consumer 4: Remove Item 1365180540 from 1
Consumer 4: Remove Item 1540383426 from 2
Consumer 4: Remove Item 304089172 from 3
Consumer 4: Remove Item 1303455736 from 4
Producer 1: Insert Item 35005211 at 0
Producer 1: Insert Item 521595368 at 1
Producer 1: Insert Item 294702567 at 2
Producer 1: Insert Item 1726956429 at 3
Producer 1: Insert Item 336465782 at 4
Consumer 5: Remove Item 35005211 from 0
Consumer 5: Remove Item 521595368 from 1
Consumer 5: Remove Item 294702567 from 2
Consumer 5: Remove Item 1726956429 from 3
Consumer 5: Remove Item 336465782 from 4
jap@virtual-box:~/Downloads$ g++ Question2b.cpp -o answer -lpthread
jap@virtual-box:~/Downloads$ ./answer
Memory from 0 to 15 allocate
Memory from 16 to 31 allocated
Memory from 32 to 47 allocate
Memory from 48 to 63 allocated
Memory block from 0 to 15 freed
Sorry, invalid free request
Memory block from 32 to 47 freed
Memory block from 16 to 31 freed
Coalescing of blocks starting at 0 and 16 was done
jap@virtual-box:~/Downloads$
```

Q3. Describe what is Producer-Consumer Problem and its solution in detail using Semaphores and Mutex and implement it in C.

Ans. A fixed-size buffer serves as a queue for both the producer and the consumer. It is the producer's responsibility to generate data and store it in the buffer. The consumer's role is to consume this buffer's data one by one.

Pseudocode Solution using Semaphore and Mutex
Initialization

```
Mutex mutex; // Used to provide mutual exclusion for critical section
Semaphore empty = N; // Number of empty slots in buffer
Semaphore full = 0 // Number of slots filled
int in = 0; //index at which producer will put the next data
int out = 0; // index from which the consumer will consume next data
int buffer[N];
```

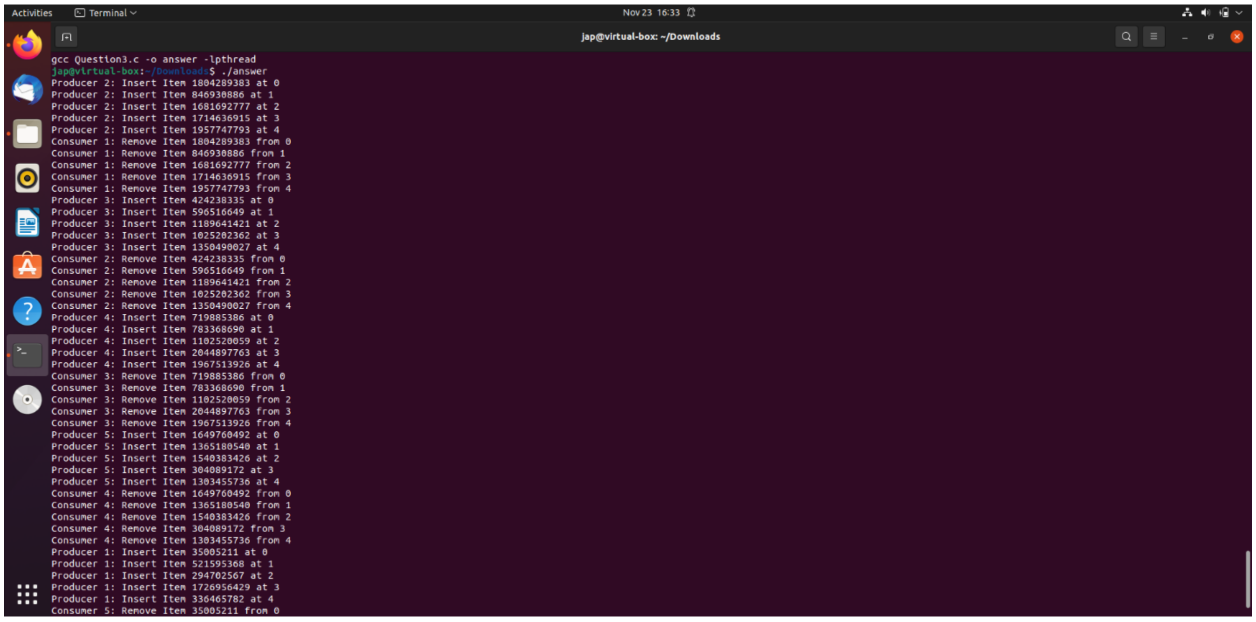
Producer Code

```
while(True) {
   // produce an item
   wait(empty); // wait/sleep when there are no empty slots
   wait(mutex);
   buffer[in] = item
   in = (in+1)%buffersize;
   signal(mutex);
   signal(full); // Signal/wake to consumer that buffer has some   data and they can consume now
}
```

Consumer Code

```
while(True) {
   wait(full); // wait/sleep when there are no full slots
   wait(mutex);
   item = buffer[out];
   out = (out+1)%buffersize;
   signal(mutex);
   signal(empty); // Signal/wake the producer that buffer slots are emptied and they can produce more
   //consumer the item
}
```

Output:



```
gcc Question3.c -o answer -lpthread
jap@virtual-box:~/Downloads$ ./answer
Producer 2: Insert Item 1804289383 at 0
Producer 2: Insert Item 846930886 at 1
Producer 2: Insert Item 1681692777 at 2
Producer 2: Insert Item 1714636915 at 3
Producer 2: Insert Item 1957747793 at 4
Consumer 1: Remove Item 1804289383 from 0
Consumer 1: Remove Item 846930886 from 1
Consumer 1: Remove Item 1681692777 from 2
Consumer 1: Remove Item 1714636915 from 3
Consumer 1: Remove Item 1957747793 from 4
Producer 3: Insert Item 424238335 at 0
Producer 3: Insert Item 596516649 at 1
Producer 3: Insert Item 1189641421 at 2
Producer 3: Insert Item 1025202362 at 3
Producer 3: Insert Item 1350490027 at 4
Consumer 2: Remove Item 424238335 from 0
Consumer 2: Remove Item 596516649 from 1
Consumer 2: Remove Item 1189641421 from 2
Consumer 2: Remove Item 1025202362 from 3
Consumer 2: Remove Item 1350490027 from 4
Producer 4: Insert Item 719885386 at 0
Producer 4: Insert Item 783368690 at 1
Producer 4: Insert Item 1102520059 at 2
Producer 4: Insert Item 2044897763 at 3
Producer 4: Insert Item 1967513926 at 4
Consumer 3: Remove Item 719885386 from 0
Consumer 3: Remove Item 783368690 from 1
Consumer 3: Remove Item 1102520059 from 2
Consumer 3: Remove Item 2044897763 from 3
Consumer 3: Remove Item 1967513926 from 4
Producer 5: Insert Item 1649760492 at 0
Producer 5: Insert Item 1365180540 at 1
Producer 5: Insert Item 1540383426 at 2
Producer 5: Insert Item 304089172 at 3
Producer 5: Insert Item 1303455736 at 4
Consumer 4: Remove Item 1649760492 from 0
Consumer 4: Remove Item 1365180540 from 1
Consumer 4: Remove Item 1540383426 from 2
Consumer 4: Remove Item 304089172 from 3
Consumer 4: Remove Item 1303455736 from 4
Producer 1: Insert Item 35005211 at 0
Producer 1: Insert Item 521595368 at 1
Producer 1: Insert Item 294702567 at 2
Producer 1: Insert Item 1726956429 at 3
Producer 1: Insert Item 336465782 at 4
Consumer 5: Remove Item 35005211 from 0
```