1.Print the following Pattern
A 1 a B 2 b C 3 c ... Y 25 y Z 26 z
(Multithreads (Hint: using 3 Threads))

Code:

```
#include <iostream>
#include <thread>
#include <mutex>
#include <semaphore.h>
#include <unistd.h>
#define THREAD_NUM 3
using namespace std;

sem_t SmallL;
sem_t CapitalL;
sem_t numerical;

void function()
{

   char c;
   for (c = 'A'; c <= 'Z'; ++c)
   {
      sem_wait(&CapitalL);
      std::cout << c << "";
      sem_post(&numerical);
   }
}

void function1()
{
   for (int i = 1; i < 27; i++)
   {
      sem_wait(&numerical);
      std::cout << " " << i << " ";
      sem_post(&SmallL);
   }
}

void function2()
{

   char c;
   for (c = 'a'; c <= 'z'; ++c)
   {
      sem_wait(&SmallL);
      std::cout << c << " ";
      sem_post(&CapitalL);
```

```
    }
}

int main()
{
    sem_init(&SmallL, 0, 0);
    sem_init(&CapitalL, 0, 1);
    sem_init(&numerical, 0, 0);
    std::thread small, capital, numeric;

    small = std::thread(function2);
    capital = std::thread(function);
    numeric = std::thread(function1);

    capital.join();
    numeric.join();
    small.join();
}
```

Output:

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

Code:

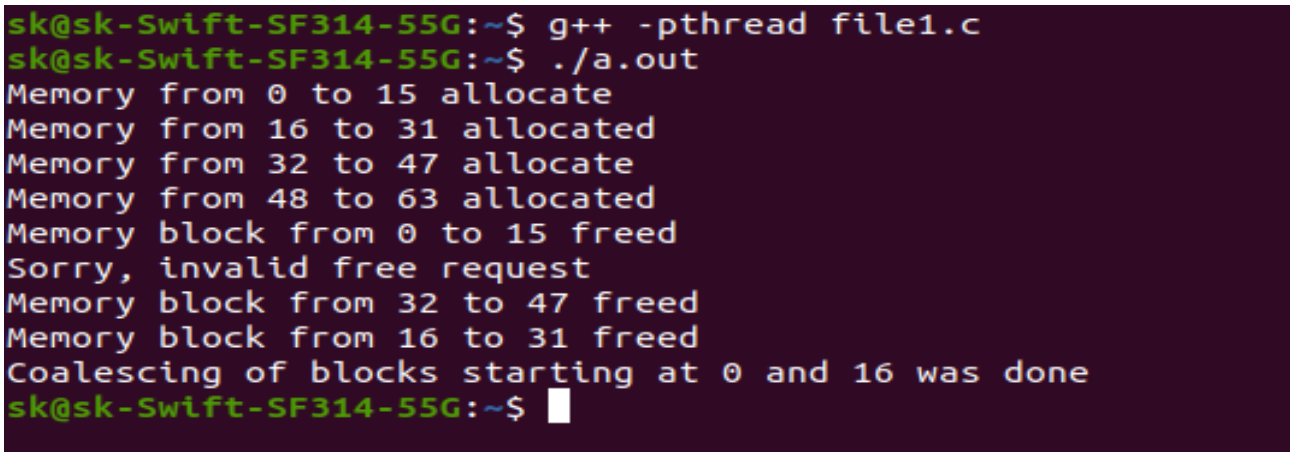
```
#include<bits/stdc++.h>
using namespace std;

// declaration of size of vector of pairs
int size;

// Global vector of pairs will track all the free nodes of various sizes
vector<pair<int, int>> Array1[1000000];

// Map used as hash map to store the starting address as key and size of allocated segment key as value
map<int, int> mp;

void Buddy(int k)
{

   // Maximum number of powers of 2 possible
   int n = ceil(log(k) / log(2));

   size = n + 1;
   for(int i = 0; i <= n; i++)
      Array1[i].clear();

   // Initially whole block of specified size will be available
   Array1[n].push_back(make_pair(0, k - 1));
}

void allocation(int s)
{

   // Calculate index in free list to search for block if available
   int x = ceil(log(s) / log(2));

   // Block available
   if (Array1[x].size() > 0)
   {
      pair<int, int> temp = Array1[x][0];

      // Remove block from free list
      Array1[x].erase(Array1[x].begin());
```

```
    cout << "Memory from " << temp.first
         << " to " << temp.second
         << " allocated" << "\n";

    // Map starting address with
    // size to make deallocating easy
    mp[temp.first] = temp.second -
                     temp.first + 1;
}
else
{
    int i;

    // If not, search for a larger block
    for(i = x + 1; i < size; i++)
    {

        // Find block size greater
        // than request
        if (Array1[i].size() != 0)
            break;
    }

    // If no such block is found
    if (i == size)
    {
        cout << "Failed to allocate memory\n";
    }

    // If found
    else
    {
        pair<int, int> temp;
        temp = Array1[i][0];

        // Remove first block to split it into halves
        Array1[i].erase(Array1[i].begin());
        i--;

        for(;i >= x; i--)
        {

            // Now divide block into two halves
            pair<int, int> pair1, pair2;
            pair1 = make_pair(temp.first,
                              temp.first +
                             (temp.second -
                              temp.first) / 2);
            pair2 = make_pair(temp.first +
                             (temp.second -
```

```
                        temp.first + 1) / 2,
                        temp.second);

            Array1[i].push_back(pair1);

            // Push them in free list
            Array1[i].push_back(pair2);
            temp = Array1[i][0];

            // Remove first free block for further spliting
            Array1[i].erase(Array1[i].begin());
        }

        cout << "Memory from " << temp.first
             << " to " << temp.second
             << " allocate" << "\n";

        mp[temp.first] = temp.second -
                    temp.first + 1;
      }
   }
}

void deallocation(int id)
{

   // If no such starting address available
   if(mp.find(id) == mp.end())
   {
      cout << "Invalid free request\n";
      return;
   }

   // Size of block to be searched
   int n = ceil(log(mp[id]) / log(2));

   int i, buddyNumber, buddyAddress;

   // Add the block in free list
   Array1[n].push_back(make_pair(id,
                       id + pow(2, n) - 1));
   cout << "Memory block from " << id
        << " to "<< id + pow(2, n) - 1
        << " freed\n";

   // Calculate the buddy number
   buddyNumber = id / mp[id];

   if (buddyNumber % 2 != 0)
      buddyAddress = id - pow(2, n);
```

```
    else
        buddyAddress = id + pow(2, n);

    // Search in free list to find it's buddy
    for(i = 0; i < Array1[n].size(); i++)
    {

        // If buddy found and is also free
        if (Array1[n][i].first == buddyAddress)
        {

            // Now merge the buddies to make them one large free memory block
            if (buddyNumber % 2 == 0)
            {
                Array1[n + 1].push_back(make_pair(id,
                  id + 2 * (pow(2, n) - 1)));

                cout << "Coalescing of blocks starting at "
                     << id << " and " << buddyAddress
                     << " was done" << "\n";
            }
            else
            {
                Array1[n + 1].push_back(make_pair(
                    buddyAddress, buddyAddress +
                    2 * (pow(2, n))));

                cout << "Coalescing of blocks starting at "
                     << buddyAddress << " and "
                     << id << " was done" << "\n";
            }
            Array1[n].erase(Array1[n].begin() + i);
            Array1[n].erase(Array1[n].begin() +
            Array1[n].size() - 1);
            break;
        }
    }

    // Will remove the key existence from map
    mp.erase(id);
}

// Driver code
int main()
{
    Buddy(128);
    allocation(16);
    allocation(16);
    allocation(16);
    allocation(16);
```

```
    deallocation(0);
    deallocation(9);
    deallocation(32);
    deallocation(16);

    return 0;
}
```



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

Code:

```
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include <stdio.h>

/*
I have used 5 producers and 5 consumers
*/

#define MaximumItems 5 // Maximum items that will be poduced by producer
#define Buffer_Size 5 // Size of the buffer

sem_t Empty;
sem_t full;
int in = 0;
int out = 0;
int buffer[Buffer_Size];
pthread_mutex_t mutex;

void *producer(void *pno)
{
   int item;
   for(int i = 0; i < MaximumItems; i++) {
      item = rand(); // Produce an random item
      sem_wait(&Empty);
      pthread_mutex_lock(&mutex);
      buffer[in] = item;
      printf("Producer %d: Insert Item %d at %d\n", *((int *)pno),buffer[in],in);
      in = (in+1)%Buffer_Size;
      pthread_mutex_unlock(&mutex);
      sem_post(&full);
   }
}
void *consumer(void *cno)
{
   for(int i = 0; i < MaximumItems; i++) {
```

```
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int item = buffer[out];
        printf("Consumer %d: Remove Item %d from %d\n",*((int *)cno),item, out);
        out = (out+1)%Buffer_Size;
        pthread_mutex_unlock(&mutex);
        sem_post(&Empty);
    }
}

int main()
{

    pthread_t pro[5],con[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&Empty,0,Buffer_Size);
    sem_init(&full,0,0);

    int k[5] = {1,2,3,4,5}; //Will give numbering the producer and consumer

    for(int i = 0; i < 5; i++) {
        pthread_create(&pro[i], NULL, (void *)producer, (void *)&k[i]);
    }
    for(int i = 0; i < 5; i++) {
        pthread_create(&con[i], NULL, (void *)consumer, (void *)&k[i]);
    }

    for(int i = 0; i < 5; i++) {
        pthread_join(pro[i], NULL);
    }
    for(int i = 0; i < 5; i++) {
        pthread_join(con[i], NULL);
    }

    pthread_mutex_destroy(&mutex);
    sem_destroy(&Empty);
    sem_destroy(&full);

    return 0;

}
```

Output:

```
sk@sk-Swift-SF314-55G:~$ gcc -pthread file.c
sk@sk-Swift-SF314-55G:~$ ./a.out
Producer 1: Insert Item 1804289383 at 0
Producer 1: Insert Item 1714636915 at 1
Producer 1: Insert Item 1957747793 at 2
Producer 2: Insert Item 1681692777 at 3
Producer 3: Insert Item 846930886 at 4
Consumer 1: Remove Item 1804289383 from 0
Consumer 2: Remove Item 1714636915 from 1
Producer 1: Insert Item 424238335 at 0
Producer 5: Insert Item 719885386 at 1
Consumer 1: Remove Item 1957747793 from 2
Consumer 1: Remove Item 1681692777 from 3
Consumer 1: Remove Item 846930886 from 4
Producer 2: Insert Item 596516649 at 2
Consumer 3: Remove Item 424238335 from 0
Consumer 4: Remove Item 719885386 from 1
Producer 3: Insert Item 1189641421 at 3
Producer 5: Insert Item 1350490027 at 4
Consumer 1: Remove Item 596516649 from 2
Producer 4: Insert Item 1649760492 at 0
Consumer 3: Remove Item 1189641421 from 3
Consumer 5: Remove Item 1350490027 from 4
Producer 3: Insert Item 1102520059 at 1
Producer 5: Insert Item 2044897763 at 2
Producer 1: Insert Item 1025202362 at 3
Consumer 5: Remove Item 1649760492 from 0
Producer 4: Insert Item 1967513926 at 4
Producer 2: Insert Item 783368690 at 0
Consumer 2: Remove Item 1102520059 from 1
Consumer 4: Remove Item 2044897763 from 2
Consumer 4: Remove Item 1025202362 from 3
Consumer 2: Remove Item 1967513926 from 4
Producer 3: Insert Item 1365180540 at 1
Consumer 3: Remove Item 783368690 from 0
Producer 5: Insert Item 1540383426 at 2
Producer 4: Insert Item 304089172 at 3
Consumer 4: Remove Item 1365180540 from 1
Producer 2: Insert Item 1303455736 at 4
Consumer 3: Remove Item 1540383426 from 2
Producer 5: Insert Item 521595368 at 0
Consumer 5: Remove Item 304089172 from 3
Producer 3: Insert Item 35005211 at 1
Consumer 3: Remove Item 1303455736 from 4
Producer 4: Insert Item 294702567 at 2
Consumer 4: Remove Item 521595368 from 0
Consumer 2: Remove Item 35005211 from 1
Producer 2: Insert Item 1726956429 at 3
Producer 4: Insert Item 336465782 at 4
Consumer 5: Remove Item 294702567 from 2
Consumer 5: Remove Item 1726956429 from 3
Consumer 2: Remove Item 336465782 from 4
sk@sk-Swift-SF314-55G:~$
```