

Q1 b) Code to print the given pattern using multi threads in C++

```
#include <bits/stdc++.h>

void print_pattern(std::vector<char> &v, std::mutex &m, int &thread_id)
{
    std::lock_guard lg{m};

    for (auto &i : v)
    {
        std::cout << i << " ";
    }
}

int main()
{
    /* std::vector<char> capitals;
    std::vector<char> numbers;
    std::vector<char> lowerCase; */

    std::vector<char> shared_resource;

    std::mutex m;

    char temp = 'A';
    char temp2 = 'a';

    for (int i = 0; i < 26; i++)
    {
        char temp3 = '0' + (i + 1);

        shared_resource.push_back(temp++);
        shared_resource.push_back(temp3);
        shared_resource.push_back(temp2++);
    }

    int thread_id_0 = 0;
    int thread_id_1 = 1;
    int thread_id_2 = 2;

    std::thread t1{print_pattern, std::ref(shared_resource), std::ref(m), std::ref(thread_id_0)};
    std::thread t2{print_pattern, std::ref(shared_resource), std::ref(m), std::ref(thread_id_1)};
    std::thread t3{print_pattern, std::ref(shared_resource), std::ref(m), std::ref(thread_id_2)};

    if (t1.joinable())
        t1.join();

    if (t2.joinable())
        t2.join();

    if (t3.joinable())
        t3.join();

    std::cout << "\n";

    return 0;
}
```

**Output:**

A 1 a B 2 b C 3 c D 4 d E 5 e F 6 f G 7 g ..... X 24 x Y 25 y Z 26 z

**Screenshot:**



```
> ./a.out
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
```

## Q2 b) Code to showcase Buddy's Algorithm for Memory Allocation and Deallocation in c++

```

#include <bits/stdc++.h>

using namespace std;

int size;
vector <pair<int, int>> arr[100000];
map <int, int> mp;

void buddyAlgo(int s){
    int n = ceil(log(s) / log(2));
    size = n + 1;
    for (int i = 0; i <= n; i++) arr[i].clear();
    arr[n].push_back(make_pair(0, s - 1));
}

void memory_allocation(int s){
    int x = ceil(log(s) / log(2));
    if (arr[x].size() > 0){
        pair <int, int> temp = arr[x][0];
        arr[x].erase(arr[x].begin());
        cout << "Memory allocated successfully from " << temp.first << " to " << temp.second << endl;
        mp[temp.first] = temp.second - temp.first + 1;
    }
    else{
        int i;
        for (i = x + 1; i < size; i++){
            if (arr[i].size() != 0) break;
        }
        if (i == size){
            cout << "Memory allocation was unsuccessful!" << endl;
        }
        else{
            pair<int, int> temp;
            temp = arr[i][0];
            arr[i].erase(arr[i].begin());
            i--;

            for (; i >= x; i--){
                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);
                arr[i].push_back(pair1);
                arr[i].push_back(pair2);
                temp = arr[i][0];
                arr[i].erase(arr[i].begin());
            }

            cout << "Memory allocation from " << temp.first << " to " << temp.second << endl;

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

```

```
void memory_deallocation(int id){
    if (mp.find(id) == mp.end()){
        cout << "The free request was invalid" << endl;
        return;
    }
    int n = ceil(log(mp[id]) / log(2));
    int i, buddyNumber, buddyAddress;
    arr[n].push_back(make_pair(id, id + pow(2, n) - 1));

    cout << "Memory block from " << id << " to " << id + pow(2, n) - 1 << " has been now freed" << endl;
    buddyNumber = id / mp[id];

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

    for (i = 0; i < arr[n].size(); i++){
        if (arr[n][i].first == buddyAddress){
            if (buddyNumber % 2 == 0){
                arr[n + 1].push_back(make_pair(id, id + 2 * (pow(2, n) - 1)));

                cout << "Blocks starting at " << id << " and " << buddyAddress << " were merged." <<
endl;
            }
            else{
                arr[n + 1].push_back(make_pair(buddyAddress, buddyAddress + 2*(pow(2, n))));

                cout << "Blocks starting at " << buddyAddress << " and " << id << " were merged." <<
endl;
            }
            arr[n].erase(arr[n].begin() + i);
            arr[n].erase(arr[n].begin() +
                arr[n].size() - 1);
            break;
        }
    }
    mp.erase(id);
}

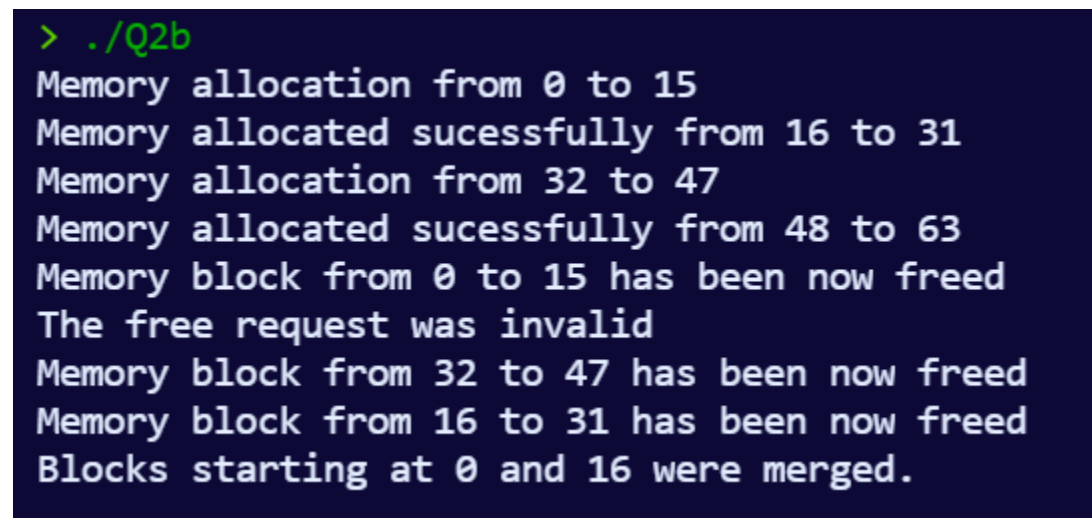
int main(){
    buddyAlgo(128);
    memory_allocation(16);
    memory_allocation(16);
    memory_allocation(16);
    memory_allocation(16);
    memory_deallocation(0);
    memory_deallocation(9);
    memory_deallocation(32);
    memory_deallocation(16);

    return 0;
}
```

**Output:**

Memory allocation from 0 to 15  
Memory allocated successfully from 16 to 31  
Memory allocation from 32 to 47  
Memory allocated successfully from 48 to 63  
Memory block from 0 to 15 has been now freed  
The free request was invalid  
Memory block from 32 to 47 has been now freed  
Memory block from 16 to 31 has been now freed  
Blocks starting at 0 and 16 were merged.

**Screenshot:**



```
> ./Q2b
Memory allocation from 0 to 15
Memory allocated sucessfully from 16 to 31
Memory allocation from 32 to 47
Memory allocated sucessfully from 48 to 63
Memory block from 0 to 15 has been now freed
The free request was invalid
Memory block from 32 to 47 has been now freed
Memory block from 16 to 31 has been now freed
Blocks starting at 0 and 16 were merged.
```

### Q3 Code to showcase the solution of Producer-Consumer Problem in C

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

int mutex = 1;
int full = 0;
int empty = 10, x = 0;

void producer(){
    --mutex;
    ++full;
    --empty;
    x++;

    printf("Producer produces %d\n", x);

    ++mutex;
}

void consumer(){
    --mutex;
    --full;
    ++empty;

    printf("\nConsumer consumes %d", x);

    x--;
    ++mutex;
}

int main(){
    int n;
    printf("\n1. Enter 1 for Producer"
           "\n2. Enter 2 for Consumer"
           "\n3. Press any key to Exit\n");

    for (int i = 1; i > 0; i++)
    {
        printf("\nEnter your choice:");
        scanf("%d", &n);
        switch (n){
            case 1:
                if ((mutex == 1) && (empty != 0)) producer();
                else printf("Buffer is full!");

                break;

            case 2:
                if ((mutex == 1) && (full != 0)) consumer();
                else printf("Buffer is empty!");

                break;
            default:
                exit(0);
                break;
        }
    }
}
```

**Output:**

1. Enter 1 for Producer
2. Enter 2 for Consumer
3. Press any key to Exit

Enter your choice:2

Buffer is empty!

Enter your choice:2

Buffer is empty!

Enter your choice:1

Producer produces 1

Enter your choice:2

Consumer consumes 1

Enter your choice:1

Producer produces 1

Enter your choice:2

Consumer consumes 1

Enter your choice:1

Producer produces 1

Enter your choice:1

Producer produces 2

Enter your choice:1

Producer produces 3

Enter your choice:1

Producer produces 4

Enter your choice:1

Producer produces 5

**Screenshot:**

```
> ./a.out  
1. Enter 1 for Producer  
2. Enter 2 for Consumer  
3. Press any key to Exit
```

```
Enter your choice:2  
Buffer is empty!  
Enter your choice:2  
Buffer is empty!  
Enter your choice:1  
Producer produces 1
```

```
Enter your choice:2
```

```
Consumer consumes 1  
Enter your choice:1  
Producer produces 1
```

```
Enter your choice:2
```

```
Consumer consumes 1  
Enter your choice:1  
Producer produces 1
```

```
Enter your choice:1  
Producer produces 2
```

```
Enter your choice:1  
Producer produces 3
```

```
Enter your choice:1  
Producer produces 4
```

```
Enter your choice:1  
Producer produces 5
```

```
Enter your choice:█
```