5) EFFICIENTLY IMPLEMENTING K STACKS IN SINGLE ARRAY

Method 1 (Divide the array in slots of size n/k)
A simple way to implement k stacks is to divide the array in k slots of size n/k each, and fix the slots for different stacks, i.e., use arr[0] to arr[n/k-1] for first stack, and arr[n/k] to arr[2n/k-1] for stack2 where arr[] is the array to be used to implement two stacks and size of array be n.

The problem with this method is inefficient use of array space. A stack push operation may result in stack overflow even if there is space available in arr[]. For example, say the k is 2 and array size (n) is 6 and we push 3 elements to the first and do not push anything to the second second stack. When we push the 4th element to first, there will be overflow even if we have space for 3 more elements in array.

Method 2 (A space efficient implementation)
The idea is to use two extra arrays for efficient implementation of k stacks in an array. This may not make much sense for integer stacks, but stack items can be large for example stacks of employees, students, etc where every item is hundreds of bytes. For such large stacks, the extra space used is comparatively very less as we use two integer arrays as extra space.

Following are the two extra arrays are used:

1) top[]: This is of size k and stores indexes of top elements in all stacks.

2) previous[]: This is of size n and stores indexes of next item for the items in array arr[]. Here arr[] is an actual array that stores k stacks.

3) empty_positions[]: This array is used to store the empty indexes because on popping the elements it will create many empty spaces in between,so to keep the track of empty positions we use this array to store all those indexes.

CODE OF METHOD 2:

```
#include<iostream>
using namespace std;

class kstacks{
   private:
   int array[100];
   int empty_positions[100];
   int k;
   int *top;
   int previous[100];
   int j=0;

   public:
   kstacks(int k){
      this->k=k;
      top=new int[k];
      for(int i=0;i<k;i++){
```

```
        top[i]=-1;
      }
      for(int i=0;i<100;i++){
        empty_positions[i]=i;
      }
   }
   void push(int k,int data){
      if(j==100){
        cout<<"\n Stack Overflow.";
      }
      else{
        array[empty_positions[j]]=data;
        previous[empty_positions[j]]=top[k];
        top[k]=empty_positions[j];
        j++;
      }
   }
   void peek(int k){
      int start_index=top[k];
      if(start_index!=-1){
        cout<<"\n The stack elements of stack k : ";
        while(start_index!=-1){
          cout<<array[start_index]<<"("<<start_index<<")"<<"  ";
          start_index=previous[start_index];
        }
      }
      else{
        cout<<"\n The stack k is empty.";
      }
   }
   void pop(int k){
      if(top[k]==-1){
        cout<<"\n Stack Underflow.";
      }
      else{
        j--;
        empty_positions[j]=top[k];
        top[k]=previous[top[k]];
      }
   }
};


int main(){
   kstacks k(10);
   k.push(0,10);
   k.push(3,20);
   k.push(2,30);
   k.push(1,40);
   k.push(0,50);
```

```
    k.push(1,60);
    k.push(0,70);
    k.push(2,80);
    k.push(3,90);
    k.push(2,100);
    k.push(0,110);

    k.peek(0);
    k.peek(1);
    k.peek(2);
    k.peek(3);
    k.peek(4);

    cout<<endl;

    k.pop(0);
    k.push(3,200);
    k.peek(0);
    k.peek(1);
    k.peek(2);
    k.peek(3);

    cout<<endl;

    k.pop(1);
    k.push(2,300);
    k.peek(0);
    k.peek(1);
    k.peek(2);
    k.peek(3);

    return 0;
}
```