#include <iostream>

using namespace std;

// STACK VƏ HEAP

// Part 1 | Task 1

// 1. Dinamik yaddaş ayıran funskiya yazın.

int\* reserveMemory(const int\* size)

{

int\* arr = new int[\*size] {};

return arr;

}

// Part 1 | Task 2

// 2. Dinamik massivi random ədədlərlə dolduran funksiya yazın

void initArray(int\* arr,const int\* size)

{

int min = 1;

int max = 100;

int random = 0;

for (int x = 0; x < \*size; x++)

{

random = min + rand() % (max - min);

arr[x] = random;

}

}

// Part 2 Task 1 üçün

void initArray2(int\* arr, const int\* size)

{

srand(time(0));

int min = -50;

int max = 100;

int random = 0;

for (int x = 0; x < \*size; x++)

{

random = min + rand() % (max - min);

arr[x] = random;

}

}

// Part 1 | Task 3

// 3. Dinamik massivi ekrana çıxaran funksiya yazın.

void printArray(int\* arr,const int\* size)

{

if (arr != nullptr)

{

cout << "\n My Dynamic Array : ";

for (int y = 0; y < \*size; y++)

{

cout << \*(arr + y) << ", ";

}

cout << "\b\b." << endl;

}

else

cout << "\n There is no such a pointer and array." << endl;

}

// Part 1 | Task 4

// 4. Dinamik massivi silən funksiya yazın.

int\* deleteArray(int\* arr)

{

delete[]arr;

arr = nullptr;

cout << "\n Dynamic array has been successully deleted!" << endl;

return arr;

}

// Part 1 | Task 5

// 5. Dinamik massivin sonuna element əlavə edən funksiya yazın.

int\* addNumberToEnd(int\* arr, int\* size, int number)

{

int\* newArray = new int[\*size + 1]{};

for (int x = 0; x < \*size; x++)

{

\*(newArray + x) = \*(arr + x);

}

newArray[\*size] = number;

delete[]arr;

arr = newArray;

newArray = nullptr;

(\*size)++;

return arr;

}

// Part 1 | Task 6

// 6. Dinamik massivin göstərilən indexinə element əlavə edən funksiya

// yazın.

int\* addNumberIndex(int\* arr, int\* size, int index, int number)

{

int\* newArray = new int[\*size + 1]{};

for (int y = 0; y < index; y++)

{

newArray[y] = arr[y];

}

newArray[index] = number;

for (int z = index; z < \*size; z++)

{

newArray[z+1] = arr[z];

}

delete[]arr;

arr = newArray;

newArray = nullptr;

(\*size)++;

return arr;

}

// Part 1 | Task 7

// 7. Dinamik massivdən göstərilən indexdəki elementin silinməsi

int\* deleteNumberIndex(int\* arr, int\* size, int index)

{

int\* newArray = new int[\*size - 1]{};

for (int x = 0; x < index; x++)

{

newArray[x] = arr[x];

}

for (int y = index+1; y < \*size; y++)

{

newArray[y-1] = arr[y];

}

delete[]arr;

arr = newArray;

newArray = nullptr;

(\*size)--;

return arr;

}

// Part 1 | Task 8

// 8. Dinamik massivin sonuna bir blok element əlavə edən funksiya yazın.

int\* addElements(int\* arr, int\* size, int\* arrOfElements, const int size2)

{

int\* newArray = new int[\*size + size2]{};

for (int x = 0; x < \*size; x++)

{

newArray[x] = arr[x];

}

for (int y = \*size; y < \*size+size2; y++)

{

newArray[y] = arrOfElements[y-\*size];

}

delete[]arr;

arr = newArray;

newArray = nullptr;

\*size += size2;

return arr;

}

// Part 1 | Task 9

// 9. Dinamik massivin göstərilən indexindən sonra bir blok element əlavə

// edən funksiya yazın.

int\* addElementsIndex(int\* arr, int\* size, int\* arrOfElements, const int size2, int index)

{

int\* newArray = new int[\*size + size2]{};

for (int x = 0; x < index+1; x++)

{

newArray[x] = arr[x];

}

for (int y = index+1, y2 = 0; y < index + size2+1; y++, y2++)

{

newArray[y] = arrOfElements[y2];

}

for (int z = index+size2+1; z < \*size+size2; z++)

{

newArray[z] = arr[z-size2];

}

delete[]arr;

arr = newArray;

newArray = nullptr;

\*size += size2;

return arr;

}

// Part 1 | Task 10

// 10. Dinamik massivdən bir blok elementi silən funksiya yazın.

// Bir index-dən digər index-ə kimi başa düşdüm

int\* deleteElementsIndex(int\* arr, int\* size, int start, int end)

{

int mySize = \*size - (end - start + 1);

int\* newArray = new int[mySize]{};

for (int x = 0; x < start; x++)

{

newArray[x] = arr[x];

}

for (int y = end+1,y2=start; y < \*size; y++,y2++)

{

newArray[y2] = arr[y];

}

delete[]arr;

arr = newArray;

newArray = nullptr;

\*size -= (end - start)+1;

return arr;

}

// STACK VƏ HEAP 2

// Part 2 | Task 1

// 1. Funksiya, parametr olaraq, massiv və onun ölçüsünü qəbul

// edir.Massivin içərsindəki bütün mənfi ədədləri silin və yeni

// massivin addressini qaytarın.

int\* onlyPositives(int\* arr, int\* size)

{

int\* newArray = new int[\*size]{};

int index = 0;

for (int y = 0; y < \*size; y++)

{

if (arr[y] >= 0)

{

newArray[index] = arr[y];

index++;

}

}

\*size = index;

delete[]arr;

arr = newArray;

newArray = nullptr;

return arr;

}

// Part 2 | Task 2

// 2. İki massiv verilib: А[M] və B[N] (M və N klaviaturadan daxil

// edilir).Minimal ölçüdə üçüncü massiv yaradıb bu iki

// massivi üçüncü masivə yazın.

int\* uniteMassives(int\* arr1, int\* size1, int\* arr2, const int\* size2)

{

int totalSize = \*size1 + \*size2;

int\* newArray = new int[totalSize] {};

for (int x = 0; x < \*size1; x++)

{

newArray[x] = arr1[x];

}

for (int y = 0,y2 = \*size1; y < \*size2; y++,y2++)

{

newArray[y2] = arr2[y];

}

delete[]arr1;

arr1 = newArray;

newArray = nullptr;

\*size1 = totalSize;

return arr1;

}

// Part 2 | Task 3

// 3. İki massiv verilib: А[M] və B[N] (M və N klaviaturadan daxil

// edilir).Minimal ölçüdə üçüncü massiv yaradıb bu iki

// massivin ortaq elementlərini üçüncü masivə yazın.

int\* commonElements(int\* arr1, int\* size1, int\* arr2, int\* size2)

{

int\* newArray = new int[\*size1]{};

int index = 0;

for (int y = 0; y < \*size1; y++)

{

for (int x = 0; x < \*size2; x++)

{

if (arr1[y] == arr2[x])

{

newArray[index] = arr1[y];

index++;

}

}

}

\*size1 = index;

delete[]arr1;

arr1 = newArray;

newArray = nullptr;

return arr1;

}

// Part 2 | Task 4

// 4. İki massiv verilib: А[M] və B[N] (M və N klaviaturadan daxil

// edilir).Minimal ölçüdə üçüncü massiv yaradıb, A - da olub

// B - də olmayan elementləri üçüncü massivə yazın.

// Təkrarlanmasın.

int\* notCommon(int\* arr1, int\* size1, int\* arr2, int\* size2)

{

int\* newArray = new int[\*size1]{};

int index = 0;

for (int x = 0; x < \*size1; x++)

{

int hasSimilar = 0;

for (int y = 0; y < \*size2; y++)

{

if (arr1[x] == arr2[y])

{

hasSimilar++;

}

}

if (hasSimilar == 0)

{

newArray[index] = arr1[x];

index++;

}

}

//Zipping array

int\* zippedNewArray = new int[index]{};

int zippedArrayindex = 0;

for (int a = 0; a < \*size1; a++)

{

int hasSimilar = 0;

for (int b = a + 1; b < \*size1; b++)

{

if (newArray[a] == newArray[b])

{

hasSimilar++;

}

}

zippedNewArray[zippedArrayindex] = newArray[a];

zippedArrayindex++;

a += hasSimilar;

}

\*size1 = index;

delete[]arr1;

arr1 = zippedNewArray;

zippedNewArray = nullptr;

newArray = nullptr;

return arr1;

}

// Part 2 | Task 5

// 5. İki massiv verilib: А[M] və B[N] (M və N klaviaturadan daxil

// edilir).Minimal ölçüdə üçüncü massiv yaradıb, A və B - nin

// ortaq olmayan elemntlərini üçüncü massivə yazın.

// Təkrarlanmasın.

int\* notCommonBoth(int\* arr1, int\* size1, int\* arr2, int\* size2)

{

int\* newArray = new int[\*size1+\*size2]{};

int index = 0;

for (int x = 0; x < \*size1; x++)

{

int hasSimilar = 0;

for (int y = 0; y < \*size2; y++)

{

if (arr1[x] == arr2[y])

{

hasSimilar++;

}

}

if (hasSimilar == 0)

{

newArray[index] = arr1[x];

index++;

}

}

for (int x = 0; x < \*size2; x++)

{

int hasSimilar = 0;

for (int y = 0; y < \*size1; y++)

{

if (arr2[x] == arr1[y])

{

hasSimilar++;

}

}

if (hasSimilar == 0)

{

newArray[index] = arr2[x];

index++;

}

}

//Zipping array

int\* zippedNewArray = new int[index] {};

int zippedArrayindex = 0;

for (int a = 0; a < \*size1; a++)

{

int hasSimilar = 0;

for (int b = a + 1; b < \*size1; b++)

{

if (newArray[a] == newArray[b])

{

hasSimilar++;

}

}

zippedNewArray[zippedArrayindex] = newArray[a];

zippedArrayindex++;

a += hasSimilar;

}

\*size1 = index;

delete[]arr1;

arr1 = newArray;

zippedNewArray = nullptr;

newArray = nullptr;

return arr1;

}

// Part 2 | Task 6

// 6. Funksiya, parametr olaraq, massiv və onun ölçüsünü qəbul

// edir.Massivin içərsindəki bütün sadə ədədləri silin və yeni

// massivin addressini qaytarın.

int numberOfComposites(int arr[],const int size)

{

int compositeCounter = 0;

for (int y = 0; y < size; y++)

{

int counter = 0;

for (int x = 1; x <= arr[y]; x++)

{

if (arr[y] % x == 0) {

counter++;

}

}

if (counter > 2)

compositeCounter++;

}

return compositeCounter;

}

int\* onlyComposite(int\* arr, int\* size)

{

int\* newArray = new int[\*size]{};

int index = 0;

for (int y = 0; y < \*size; y++)

{

int counter = 0;

for (int x = 1; x <= arr[y]; x++)

{

if (arr[y] % x == 0) {

counter++;

}

}

if (counter > 2)

{

newArray[index] = arr[y];

index++;

}

}

int composites = numberOfComposites(arr, \*size);

int\* ptr = &composites;

\*size = \*ptr;

delete[]arr;

arr = newArray;

newArray = nullptr;

return arr;

}

void main()

{

// PART 1x

int size = 10;

// Part 1 | Task 1

auto arr = reserveMemory(&size);

// Part 1 | Task 2

initArray(arr, &size);

cout << " \n Part 1 | Task 3";

printArray(arr, &size);

// Part 1 | Task 4

//arr = deleteArray(arr);

cout << " \n Part 1 | Task 5";

arr = addNumberToEnd(arr, &size, 55);

printArray(arr, &size);

cout << " \n Part 1 | Task 6";

arr = addNumberIndex(arr, &size, 1, 34);

printArray(arr, &size);

cout << " \n Part 1 | Task 7";

arr = deleteNumberIndex(arr, &size, 3);

printArray(arr, &size);

cout << " \n Part 1 | Task 8";

const int size2 = 5;

int array[size2] = {1,2,3,4,5};

arr = addElements(arr, &size, array, size2);

printArray(arr, &size);

cout << " \n Part 1 | Task 9";

int array2[size2] = {11,22,33,44,55};

arr = addElementsIndex(arr, &size, array2, size2, 4);

printArray(arr, &size);

cout << " \n Part 1 | Task 10";

arr = deleteElementsIndex(arr, &size, 3, 7);

printArray(arr, &size);

// PART 2

int mySize = 10;

auto arr2 = reserveMemory(&mySize);

initArray2(arr2, &mySize);

printArray(arr2, &mySize);

cout << "\n Part 2 | Task 1";

arr2 = onlyPositives(arr2, &mySize);

printArray(arr2, &mySize);

cout << "\n Part 2 | Task 2";

int size1 = 0;

cout << "\n Enter size of the array 1 : ";

cin >> size1;

int size2m = 0;

cout << " Enter size of the array 2 : ";

cin >> size2m;

auto arr3 = reserveMemory(&size1);

auto arr4 = reserveMemory(&size2m);

initArray(arr3, &size1);

printArray(arr3, &size1);

initArray(arr4, &size2m);

printArray(arr4, &size2m);

auto resultArray = uniteMassives(arr3, &size1, arr4, &size2m);

printArray(resultArray, &size1);

cout << "\n Part 2 | Task 3";

int size3 = 0;

cout << "\n Enter size of the array 1 : ";

cin >> size3;

int size4 = 0;

cout << " Enter size of the array 2 : ";

cin >> size4;

auto arr5 = reserveMemory(&size3);

auto arr6 = reserveMemory(&size4);

initArray(arr5, &size3);

printArray(arr5, &size3);

initArray(arr6, &size4);

printArray(arr6, &size4);

auto resultArray2 = commonElements(arr5, &size3, arr6, &size4);

printArray(resultArray2, &size3);

cout << "\n Part 2 | Task 4";

int size5 = 0;

cout << "\n Enter size of the array 1 : ";

cin >> size5;

int size6 = 0;

cout << " Enter size of the array 2 : ";

cin >> size6;

auto arr7 = reserveMemory(&size5);

auto arr8 = reserveMemory(&size6);

initArray(arr7, &size5);

printArray(arr7, &size5);

initArray(arr8, &size6);

printArray(arr8, &size6);

auto resultArray3 = notCommon(arr7, &size5, arr8, &size6);

printArray(resultArray3, &size5);

cout << "\n Part 2 | Task 5";

int size7 = 0;

cout << "\n Enter size of the array 1 : ";

cin >> size7;

int size8 = 0;

cout << " Enter size of the array 2 : ";

cin >> size8;

auto arr9 = reserveMemory(&size7);

auto arr10 = reserveMemory(&size8);

initArray(arr9, &size7);

printArray(arr9, &size7);

initArray(arr10, &size8);

printArray(arr10, &size8);

auto resultArray4 = notCommonBoth(arr9, &size7, arr10, &size8);

printArray(resultArray4, &size7);

cout << "\n Part 2 | Task 6";

int size9 = 0;

cout << "\n Enter size of the array : ";

cin >> size9;

auto arr11 = reserveMemory(&size9);

initArray(arr11, &size9);

printArray(arr11, &size9);

auto resultArray5 = onlyComposite(arr11, &size9);

printArray(resultArray5, &size9);

}