Send an email with the solutions as attachments (.c, .cpp or if you have .h and .hpp files) to the following email address: tofihomework+2024info3en@gmail.com

You shouldn't attach the compiled programs. If you feel like you're sending a lot of files (5+) you can put them in a zip, but you don't need to.

Set (continued)

This task is a continuation of Homework 5. You can hand them in together or just homework 5. Together they're worth twice the points of course.

Write a Set class that can store a given amount of unique integers.

- When constructing a new set we should be able to set the maximum number of integers it can store.
- We should be able to add a new element through a method. If the given number is already in the set, then don't add it and return **false**. Otherwise add the new element and return **true**.
- The class should have a copy constructor.
- It should be possible to add a new integer to the set with the + operator. This should work similarly to the above method, but we shouldn't directly add the element to the existing set. Instead we should create a new set that contains this integer as well and return this new set.
- Make a method that returns the length of the set, in other words how many elements it contains.
- Write a method called **in** that checks whether the given integer is in the set or not. (It should return true/false.)
- Make it so that we can create the union of two sets with the + operator.
- We should also be able to subtract two sets. This should result in a set that contains all elements of the left set except the elements of the right set.
- In the above methods when we're creating a new set from two existing sets the maximum size of the new set should be the same as the larger of the two existing sets.
- We don't need to have any logic to deal with whether a new element fits in an existing set, we can assume that the sets will always be created large enough. (But if you have the time you can do it. A very easy method would be to double the size of the storage array when we run out of space and copy the existing content.)
- It should be possible to get the elements of the set through **operator[]**. The order doesn't matter. However we should make it so that the elements cannot be changed through this. (In other words it shouldn't be possible to use this as a left value.)
- We should be able to print the set using **cout**. This is how it should look like (the order doesn't matter):

```
(1, 8, 5)
```

Example main function for testing the earlier methods:

```
int main(void) {
  Set A = Set(10);
  A.add(23);
  Set B = A;
  Set C = A + 12;
  Set D = A - 23;
  Set E = A + 12;
  cout << B.in(23) << endl;
  cout << C.in(12) << endl;
  cout << D.in(23) << endl;
  cout << E.length() << endl;
  return 0;
}
```

Example main function for testing the later methods:

```
int main(void) {
  Set A = Set(100);
  Set B = Set(200);
  for(int i = 0; i < 100; i++) {
    if(i % 2 == 0) { // even numbers in A
      A.add(i);
    }
    if(i % 3 == 0) { // divisible by 3 in B
      B.add(i);
    }
  }
  Set C = A + B;
  Set D = A - B;
  for(int i = 0; i < D.length(); i++) {
    cout << D[i] << endl;
  }
  // A[0] = 10; // This shouldn't work!
  cout << C << endl;
  return 0;
}
```

Hints at the end of the page (if you have the time/perseverance, first try to solve it without the help, or just think about how you would solve it).

The task doesn't really have anything tricky to solve, it's just long. So don't be afraid of it just start working on it and everything will fall into place.

For the storage we can use a dinamically allocated array. Create this in the constructor. We should also store the amount of elements already in the set. Don't forget to write the destructor. Make sure to create a new array in the copy constructor, you shouldn't copy the array's pointer.

One thing to keep in mind is to always correctly increment and decrement the variable that stores the current amount of elements. Also you can use the earlier methods to more easily solve the later ones.

When writing the **operator[]** the only thing needed to make it unable to be used as a left value is to use a regular int as the return type instead of int reference.