Lesson 2

If you find any open questions in some of the presented exercises then feel free to make your own decisions. However, your decisions should be reasonable and must not contradict any of the conditions explicitly written for the exercise. Please, write comments in the programs that clarify your assumptions/decisions, if any.

Exercise 1

Start by reviewing class **`Clock`** discussed in Fö 7, 8, and 9 and class **`Matrice`** presented in Fö 8. Then, solve the exercise described below.

Define a class **`Set`** to represent sets of non-negative integer values belonging to the interval $[0, n]$, for a given $n > 0$. Sets of non-negative integer values belonging to the interval $[0, n]$ should be represented with an array $(n + 1)$ slots storing Booleans (**`bool`**). For instance, let $n = 10$ and consider a set $S = \{2, 5, 8\}$. Then, this set would be represented by the array below.

false	false	**true**	false	false	**true**	false	false	**true**	false	false
0	1	2	3	4	5	6	7	8	9	10

Notice that $5 \in S$, thus the 6th slot of the array stores the **`true`**, while 1 does not belong to set S and consequently the 2nd slot of the array stores **`false`**.

Add your code to the files **`set.h`** and **`set.cpp`**. The file **`test_set.cpp`** already contains a **`main`** function ready to test incrementally your code. Thus, you can comment away those tests corresponding to functions not yet implemented. The file **`ex1_out.txt`** contains the expected output of the program (disregard by now the tests in phase 5 and 6).

i. Define a class **`Set`** to represent sets implemented as described above and equip it with the following operations.

- A constructor **`Set::Set(int i);`** that creates a set with element **`i`**, i.e. **`{i}`**, if **`i`** is a non-negative integer. In this case, start with an array with $(i + 1)$ slots to represent the set. If **`i`** is a negative integer then an empty set is created represented with an array of 11 slots.
- A copy constructor.
- An assignment operator.
- An **`operator<<`** that displays all elements of a set.

Test your code with tests phases 0 to 2 (comment the other tests away).

ii. Add to your class the following operators.

- An **`operator+`** such that **`S1+S2`** returns a new set corresponding to the set union **`S1`** $\cup$ **`S2`**.

- An **`operator-`** such that **`S1-S2`** returns a new set with the elements in **`S1`** that do not belong to **`S2`**.

When testing your code, add tests phases 3 and 4.

iii. Investigate whether your class supports the operations **`S+i`** and **`i+S`**, where **`i`** is an integer. To this end use test phases 5 and 6.

Note that **`S+i`** (**`i+S`**) should return a new set with all elements belonging to **`S`** plus integer **`i`** (i.e. **`S`** ∪ **`{i}`**). If **`i`** is a negative number then **`S+i`** should return a new set with all elements that belong to **`S`**.

If one of the expressions **`S+i`** or **`i+S`** is not supported by the implementation you provided then make the necessary changes in your code such that expressions of these forms compile and execute as described.

iv. Read section 8.4.6 of the course book about the subscript **`operator[]`**. Add to the class **`Set`** a subscript operator, **`operator[]`**, with the functionality described below. Assume that **`S`** is a **`Set`** and **`i`** is an integer.

- **`S[i] = true;`** adds **`i`** to set **`S`**. Assume that **`i`** is a non-negative integer smaller than the size of the array used for set **`S`**.
- **`S[i] = false`**; removes **`i`** from set **`S`**. Assume that **`i`** is a non-negative integer smaller than the size of the array used for set **`S`**.
- **`S[i]`** should return **`true`** or **`false`**, depending whether **`i`** belongs to set **`S`**. For instance, it should be possible to write statements such as **`if (S[i])`** …**`;`**.

When testing your code, add test phase 7.

v. Finally, modify your class such that statements as the ones below compile and execute as expected.

```
Set S(100);
S = S + 7 + 8 + 10;
string w = s;
//w ← "7 8 10 100"
```

```
Set S(100);
S = S + 7 + 8 + 10;
string w;
w = s;
//w ← "7 8 10 100"
```

When testing your code, add test phase 8.

Exercise 2

Review the concept pointers to functions introduced in Fö 6. Then, solve the exercise described below.

Define a function, named **`selected_average`**, that calculates the average of all integers satisfying a given test condition **`T`**, like being a prime number or being an odd number. The

integers are stored in an array passed as parameter to the function. Note that the function should not do any keyboard input or screen output.

Start by adding the implementation of function **`selected_average`** to the file **`ex2.cpp`**. This file already contains a **`main`** that declares an array **`V`** of integers. Then,

- add to the given **`main`** the necessary function call to **`selected_average`** such that the average of the prime numbers in **`V`** is calculated and displayed; then,
- add another function call to **`selected_average`** such that the average of odd numbers in **`V`** is calculated and displayed.

Lycka till!!!