Sabancı University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Spring 2023

Homework 4 – Operator Overloading for a 2D Dynamic Array Weekly Schedule

Due: 25/04/2023, Tuesday, 21:00

PLEASE NOTE:

Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; you are expected to take actions accordingly!

You can NOT collaborate with your friends and discuss solutions. You have to write down the code on your own. Plagiarism will not be tolerated!

Introduction

In this homework, you are asked to implement a class for a weekly `Schedule` with lots of operator overloading in it. The details of these operators are given below. Our focus in this homework is on the class design, implementation and operator overloading technicalities. The usage of this class in the main program is given to you in the homework pack. You are going to use it without any change.

Class Design for the Schedule

You will implement the `Schedule` class basically as 2D dynamic array (i.e. dynamic matrix) of strings. Each element of this dynamic matrix is a string that can have the value "free" or "busy". The number of rows in the dynamic matrix will always be 7, one per day of week. The `Schedule` class will have two private data members: The first one is an integer *`time_slots`*, that represents the number of time slots in the schedule (i.e. number of columns of the matrix) in the schedule. The second one is *`data`*, which is a pointer to pointer to string (i.e. `string**`). Please see Figure 1 for a depiction of this structure.

For dynamic matrix processing, you will use the *dynamic matrix* techniques that we covered in Week 4 (see lecture notes); you will **not** use vector of vector and you will **not** use any type of linked lists.

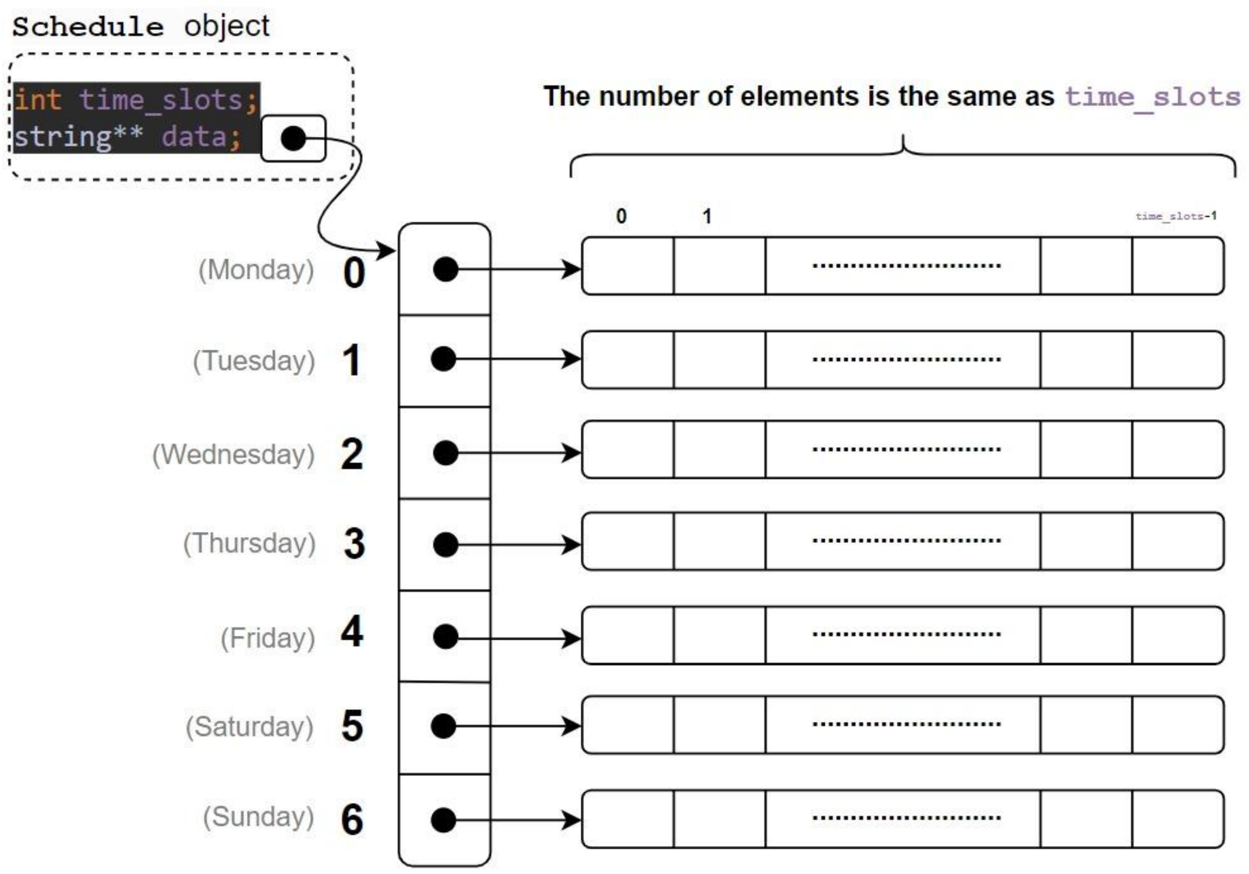

Figure 1. `Schedule` class structure

Your `Schedule` class implementation must include the following basic class functions:

- Default constructor: creates an empty `Schedule` object with 7 rows and zero columns. Thus it should create a dynamic array of string pointer of size 7, of which all elements are set to `NULL`.
- Parametric constructor: creates a `Schedule` object with 7 rows (one per day) and *c* columns, where *c* is the only parameters (integer) of the constructor. This constructor first allocates a dynamic array of string pointers of length 7, in which each element points to a dynamically allocated array (again to be allocated within the constructor) of strings of length *c*. Then, all the elements will be initialized to "free".
- Deep copy constructor: takes a `Schedule` object as const-reference parameter and creates a new `Schedule` object as a deep copy of the parameter.
- Destructor: By definition, destructor deletes all dynamically allocated memory and returns them back to the heap. You should implement this.

In addition to the basic class functions that are explained above, you will overload several operators for the `Schedule` class. These operators and the details of overloading are given below. You can overload operators as member or free functions. However, to test your competence in both mechanisms, we require you to have at least three functions implemented as free functions and at least three functions as member functions. Details are given below. However, before that let us explain an enumerated type that you will need to use.

In your homework, you should use an enumerated type that represents the days of the week as follows.

```
enum Days {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday};
```

You should add this line at the beginning of your `schedule.h` header file (after include lines, before the class definition). This `Days` type has already been used in main, and you will need to use in some operator function implementation as described below. If you do not know what `enum` type is, I have provided some notes under Week 7 notes (also briefly explained in class).

In the matrix that we use throughout the homework, our convention is that 0th row corresponds to Monday, 1st row to Tuesday, 2nd row to Wednesday, ..., 6th row to Sunday. However, the matrix indices are, of course, integers. Slot numbering starts from 0 as well. These are depicted in Figure 1.

Now, let us continue with the details of the operators that you will implement.

= This operator will be overloaded such that it will assign the `Schedule` object on the right hand-side (*rhs*) of the operator to the `Schedule` object on the left-hand side (*lhs*). Since the number of time slots of both sides might be different, the matrix sizes may not be the same. Thus, we recommend you to deallocate the matrix of *lhs* first and then make new memory allocation before copying the values. Moreover, take precaution for self-assignment. This function must be implemented in a way to allow cascaded assignments. Due to C++ language rules, this operator must be a member function (cannot be free function).

Note that + operator will be overloaded in three different ways as (i) addition of a `Days` value to a `Schedule` object (ii) addition of an integer value to a `Schedule` object (iii) addition of a `Schedule` object to another `Schedule` object.

+ This operator will be used to set all the time slots for a particular day of a schedule to "busy". It will be overloaded such that it takes a `Days` parameter on the *rhs* and a `Schedule` object on the *lhs*. Since this operator must return value (not reference), you must work on a brand-new `Schedule` object in the implementation of this function and return it at the end. The content of this brand-new `Schedule` object to be returned will exactly be the same as *lhs*, with *rhs* day row set to "busy". You will do that by changing the values in all the elements of *rhs* day (row) to the string value "busy". Please remark that this operator should **not** change the content of *lhs* object. This operator can be implemented as member or free function.

+ This operator will be used to set a particular time slot of all days of a schedule to "busy". It will be overloaded such that it takes an integer parameter on the *rhs* and

a `Schedule` object on the *lhs*. Since this operator must return value (not reference), you must work on a brand-new `Schedule` object in the implementation of this function and return it at the end. The content of this brand-new `Schedule` object to be returned will exactly be the same as *lhs*, with *rhs*th column (time slot) set to "busy". You will do that by changing the values in all of the elements of *rhs*th column to the string value “busy”. Assume that column (time slot) indexing starts with 0 (i.e. leftmost time slot is 0th slot). Please remark that this operator should **not** change the content of *lhs* object. This operator can be implemented as member or free function.
While implementing the function, you do not need to deal with the range issues of *rhs*.

+ This operator will be used to create and return a `Schedule` object that contains common free time slots of both sides of the operator. Thus, this operator will be overloaded such that it takes Schedule objects as both *rhs* and *lhs*. Since this operator must return value (not reference), you must work on a brand-new `Schedule` object in the implementation of this function and return it at the end. Each element of this brand-new `Schedule` object to be returned will be "free", if the corresponding elements in *lhs* and *rhs* are both "free"; otherwise, "busy". In this operator, you may assume that there are same number of time slots (i.e. columns) in both *lhs* and *rhs*. This operator can be implemented as member or free function.

* This operator will be used to create and return a `Schedule` object that contains common busy time slots of both sides of the operator. Thus, this operator will be overloaded such that it takes Schedule objects as both *rhs* and *lhs*. Since this operator must return value (not reference), you must work on a brand-new `Schedule` object in the implementation of this function and return it at the end. Each element of this brand-new `Schedule` object to be returned will be "busy", if the corresponding elements in *lhs* and *rhs* are both "busy"; otherwise, "free". In this operator, you may assume that there are same number of time slots (i.e. columns) in both *lhs* and *rhs*. This operator can be implemented as member or free function.

[] This operator takes a `Schedule` object as a left-hand side parameter (*lhs*), and `Days` type value as a right-hand side parameter (*rhs*). It will return a string pointer which is actually the pointer that points to the *rhs* row of *lhs*. In other words, this operator returns the row array of the *lhs* `Schedule` object that corresponds to *rhs* day so that we access the schedule of a particular day. In main.cpp file, we use it to access a particular row array by writing the row index (of type `Days`) within square brackets. While implementing the function, you do not need to deal with the index range issues.

This operator is implemented using the operator function `operator[]`. This operator can be implemented as member or free function. There is an example of this operator in Lab 7.

< This operator is used to compare the number of busy slots in *lhs* and *rhs* operands. Thus, it is overloaded such that it takes `Schedule` objects as both *lhs* and *rhs*. It returns true if the number of "busy" elements in *lhs* is less than the ones in the *rhs*. Otherwise, returns false.

Note that the operator << will be overloaded in two different ways as (i) for a `Schedule` object, and (ii) for a `Days` type value.

<< This operator takes an output stream (of type `ostream`) as *lhs* parameter and a `Schedule` object as *rhs* parameter. This operator is overloaded such that it puts the content of *rhs* `Schedule` object on the *lhs* output stream. We use this operator in main to display the content of the schedules. See the sample runs for the required output format. Since *lhs* is an output stream, this function must be a free function.

<< This operator takes an output stream (of type `ostream`) as *lhs* parameter and a `Days` value as *rhs* parameter. This operator is overloaded such that it puts the name of the day (as "Monday", "Tuesday", ..., "Sunday"), corresponding to the value of *rhs*, on the *lhs* output stream. We use this operator in main to display the `Days` type variables. Since *lhs* is an output stream, this function must be a free function.

Remark that you cannot simply put an enumerated type variable to the output stream hoping that its corresponding identifier will be displayed; if you do this, its code is displayed.

In order to see the usage and the effects of these operators, please see **sample runs** and the provided **main.cpp**.

In this homework, you are allowed to implement some other helper member functions such as accessors (getters) and mutators (setters), if needed. However, you are **<u>not allowed to use friend functions and friend classes</u>**.

A life-saving advice: Any member function that does not change the private data members needs to be **`const` member function** so that it works fine with const-reference parameters. If you do not remember what "`const` member function" is, please refer to CS201 notes.

Another important advice: First, please learn the return type and parameter structures of the operators (whether they are reference or value, whether they are const or not) and the tricks of developing them by checking out the lecture notes. Then, start designing and coding the homework. Any wrong design decision or wrong approach taken would cause severe bugs that you cannot resolve easily. Do not Google/ChatGPT these stuff since online resources may mislead you; please refer to lecture notes.

Provided main cpp file, and the requested class implementation

In this homework, **main.cpp** file is given to you within this homework package and we will test your codes with this main function with different inputs. You are not allowed to make any modifications in the main function. All class related functions, definitions and declarations (including free operator functions) must be implemented in class header and implementation files named as `schedule.h` and `schedule.cpp`. Actually, this is your task in this homework.

In main.cpp, inputs and outputs are explained with appropriate prompts so that you do not get confused. Also you can assume that there won't be any wrong inputs in test cases; in other words, you do not need to do input checks in the functions that you are going to implement.

We strongly recommend you to examine main.cpp file and sample runs before starting your homework.

Some Important Rules

In order to get a full credit, your programs must be efficient and well presented, presence of any redundant computation or unnecessary memory usage or bad indentation, or missing, irrelevant comments are going to decrease your grades. You also have to use understandable identifier names, informative introduction and prompts. Modularity is also important; you have to use functions wherever needed and appropriate. Since using classes is mandated in this homework, a proper object-oriented design and implementation will also be considered in grading.

Since you will use dynamic memory allocation in this homework, it is very crucial to properly manage the allocated area and return the deleted parts to the heap whenever appropriate. Inefficient use of memory may reduce your grade.

When we grade your homework, we pay attention to these issues. Moreover, in order to observe the real performance of your codes, we may run your programs in *Release* mode and **we may test your programs with very large test cases**. Of course, your program should work in *Debug* mode as well.

Please do not use any non-ASCII characters (Turkish or other) in your code (not even as comments). And also do not use non-ASCII characters in your file and folder names. We really mean it; otherwise, you may encounter some errors.

You are not allowed to use codes found somewhere online or other than course material. You can use only the material that are covered in lectures (CS201 and CS204 so far). However, if you use a non-standard C++ library/function/class (such as strutils) covered in the courses, either (i) you have to submit your code together with these extra source and header files so that CodeRunner runs your code properly, or (ii) copy necessary functions from them into your submitted code file by specifying where you copied them. In some assignments, we allow option (i) above in CodeRunner settings, but in some others we may not allow, so you can follow option (ii). In this homework, you may follow (ii).

You are allowed to use sample codes shared with the class by the instructor and TAs. However, you cannot start with an existing .cpp or .h file directly and update it; you have to start with an empty file. Only the necessary parts of the shared code files can be used and these parts must be clearly marked in your homework by putting comments like the following. Even if you take a piece of code and update it slightly, you have to put a similar marking (by adding "`and updated`" to the comments below.

```
/* Begin: code taken from ptrfunc.cpp */
…
/* End: code taken from ptrfunc.cpp */
```

Submission Rules (PLEASE READ)

It'd be a good idea to write your name and last name in the files that you will submit (as a comment line of course). Do not use any Turkish characters anywhere in your code (not even in comment parts). For example, if your full name is "Satılmış Özbugsızkodyazaroğlu", then you must type it as follows:

// Satilmis Ozbugsizkodyazaroglu

We use CodeRunner in SUCourse for submission. No other way of submission is possible. Since the functionality part of the grading process will be automatic, you have to strictly follow these guidelines; otherwise, we cannot grade your homework.

You will see two different versions of the homework at SUCourse. One of them (**testing ground version**) is for you to test your code. However, you will **not** submit there. You will submit to the other version (**submission version**). You cannot test your code in the submission version. **We will grade what you submit to the submission version.** Please make sure that the submitted version is the final one that you tested and you submitted both `schedule.cpp` (by copying to the answer area) and `schedule.h` (as attachment on the user interface).

The advantage CodeRunner it is that you will be able to test your code against sample test cases. However, the output should be exact, but the textual differences between the correct output and yours can be highlighted (by pressing "show differences" button) on the interface.

The submission mechanism is a bit different than the other assignments. You will submit two files: `schedule.cpp` and `schedule.h` files. **`schedule.cpp` file's content will be copied and pasted into the "Answer" area** as in other assignments. **However, you will upload `schedule.h` file as attachment** to your submission in the relevant assignment submission page on SUCourse. Then you can test your code via CodeRunner against the sample runs (by pressing the "Check" button). You can check as much as you can; there is no penalty for multiple checks. However, as mentioned above, you can test only in the testing ground version of the homework.

The file name that you will upload must definitely be `schedule.h`; otherwise it does not work. And you will not upload the provided main and stack files; we have already put them there.

Even any tiny change in the output format will result in your grade being zero (0) for that particular test case, so please test your programs yourself, and against the sample runs that are available at the relevant assignment submission page on SUCourse (CodeRunner).

In the CodeRunner, there are some visible test cases. However, they may not be used in grading. The grading test cases might be different and it is always a possibility that your program works with these given test cases, but does not work with one or more grading test cases. Thus you have to test your program thoroughly.

You don't have to complete your task in one time, you can continue from where you left last time, but you should not proceed with submission before finalizing it. Therefore, you should make sure that it's your final solution version before you submit it. Also, we still do not suggest that you develop your solution on CodeRunner but rather on your IDE on your computer.

So far the operations that you can perform on the **testing ground version** of the homework has been explained. Once you decide to submit, you should switch to **submission version** of the homework and complete the submission process there (do not submit to the testing ground version). In the submission version, you will not be able to test your homework. In the submission version, you should just copy the content of the `schedule.cpp` into the answer area, and upload `schedule.h` file to the attachments part, and then complete the submission process by finishing the attempt. Of course, you have to use the final version of your code during submission. Moreover, remark that you have to manually "Submit" (there is no automatic submission). There is no re-submission. That means, after you submit, you cannot take it back.

Last, even if you cannot completely finish your homework, you can still submit.

Sample Runs

Sample runs are given below, but these are not comprehensive, therefore you have to consider **all possible cases** to get full mark. Inputs are shown in ***bold and italic***.

We will configure CodeRunner to test these sample runs for you. However, grading test cases might be totally different.

Sample Run 1

```
Creating an empty schedule and displaying it.
Mo:
Tu:
We:
Th:
Fr:
Sa:
Su:

Schedule s after calling the full_schedule() function
Mo: busy
Tu: busy
We: busy
Th: busy
Fr: busy
Sa: busy
Su: busy
```

```
Please enter the less slots per day:
6
Throughout the program, whenever you are asked to enter a day, enter the day name with initial letter capitalized.

Initial schedule 1 (all slots are free by default):
Mo: free free free free free free
Tu: free free free free free free
We: free free free free free free
Th: free free free free free free
Fr: free free free free free free
Sa: free free free free free free
Su: free free free free free free

Please enter the day of the week that you want to make busy in schedule 1:
Monday
Please enter the slot number that you want to change to busy in all days of the week in schedule 1:
The valid range is [0,5]:
4
Schedule 1 after changing all the slots of Monday and slot number 4 to busy:
Mo: busy busy busy busy busy busy
Tu: free free free free busy free
We: free free free free busy free
Th: free free free free busy free
Fr: free free free free busy free
Sa: free free free free busy free
Su: free free free free busy free

Please enter the day of the week that you want to make busy in schedule 2:
Monday
Schedule 2 after changing all the slots of Monday to busy:
Mo: busy busy busy busy busy busy
Tu: free free free free free free
We: free free free free free free
Th: free free free free free free
Fr: free free free free free free
Sa: free free free free free free
Su: free free free free free free

Please enter the slot number that you want to change to busy in all days of the week in schedule 2:
The valid range is [0,5]:
4
The output of    cout << schedule2 + 4 << endl;    is:
Mo: busy busy busy busy busy busy
Tu: free free free free busy free
We: free free free free busy free
Th: free free free free busy free
Fr: free free free free busy free
Sa: free free free free busy free
```

```
Su: free free free free busy free

Schedule 1 after self-assignment:
Mo: busy busy busy busy busy busy
Tu: free free free free busy free
We: free free free free busy free
Th: free free free free busy free
Fr: free free free free busy free
Sa: free free free free busy free
Su: free free free free busy free

Schedule 3 created as multiplication (common busy hours) of schedule 1 and schedule 2:
Printing using display_sch() function:
    Monday busy busy busy busy busy busy
   Tuesday free free free free free free
 Wednesday free free free free free free
  Thursday free free free free free free
    Friday free free free free free free
  Saturday free free free free free free
    Sunday free free free free free free

schedule 2 and schedule 3 have the same number of busy slots.

Schedule 1 after   schedule1 = schedule1 + schedule2;
Mo: busy busy busy busy busy busy
Tu: free free free free busy free
We: free free free free busy free
Th: free free free free busy free
Fr: free free free free busy free
Sa: free free free free busy free
Su: free free free free busy free

Schedule 2 after toggling one element:
Mo: busy busy busy busy busy busy
Tu: free free free free free free
We: free free free free free free
Th: free free free free free free
Fr: free free free free free free
Sa: free free free busy free free
Su: free free free free free free

schedule 1 after   schedule1 = schedule1 + schedule2 + schedule3;
Mo: busy busy busy busy busy busy
Tu: free free free free busy free
We: free free free free busy free
Th: free free free free busy free
Fr: free free free free busy free
```

```
Sa: free free free busy busy free
Su: free free free free busy free

Schedule 4 has been created with 10 slots.
Mo: free free free free free free free free free free
Tu: free free free free free free free free free free
We: free free free free free free free free free free
Th: free free free free free free free free free free
Fr: free free free free free free free free free free
Sa: free free free free free free free free free free
Su: free free free free free free free free free free

schedule 1, 2 and 4 after   schedule1 = schedule2 = schedule4;
schedule 1:
Mo: free free free free free free free free free free
Tu: free free free free free free free free free free
We: free free free free free free free free free free
Th: free free free free free free free free free free
Fr: free free free free free free free free free free
Sa: free free free free free free free free free free
Su: free free free free free free free free free free

schedule 2:
Mo: free free free free free free free free free free
Tu: free free free free free free free free free free
We: free free free free free free free free free free
Th: free free free free free free free free free free
Fr: free free free free free free free free free free
Sa: free free free free free free free free free free
Su: free free free free free free free free free free

schedule 4:
Mo: free free free free free free free free free free
Tu: free free free free free free free free free free
We: free free free free free free free free free free
Th: free free free free free free free free free free
Fr: free free free free free free free free free free
Sa: free free free free free free free free free free
Su: free free free free free free free free free free
```

Sample Run 2

```
Creating an empty schedule and displaying it.
Mo:
Tu:
We:
Th:
Fr:
Sa:
Su:

Schedule s after calling the full_schedule() function
Mo: busy
Tu: busy
We: busy
Th: busy
Fr: busy
Sa: busy
Su: busy

Please enter the number of slots per day:
4
Throughout the program, whenever you are asked to enter a day, enter the day name with initial letter capitalized.

Initial schedule 1 (all slots are free by default):
Mo: free free free free
Tu: free free free free
We: free free free free
Th: free free free free
Fr: free free free free
Sa: free free free free
Su: free free free free

Please enter the day of the week that you want to make busy in schedule 1:
Friday
Please enter the slot number that you want to change to busy in all days of the week in schedule 1:
The valid range is [0,3]:
0
Schedule 1 after changing all the slots of Friday and slot number 0 to busy:
Mo: busy free free free
Tu: busy free free free
We: busy free free free
Th: busy free free free
Fr: busy busy busy busy
Sa: busy free free free
Su: busy free free free

Please enter the day of the week that you want to make busy in schedule 2:
```

Friday

```
Schedule 2 after changing all the slots of Friday to busy:
Mo: free free free free
Tu: free free free free
We: free free free free
Th: free free free free
Fr: busy busy busy busy
Sa: free free free free
Su: free free free free

Please enter the slot number that you want to change to busy in all days of the week in schedule 2:
The valid range is [0,3]:
```

1

```
The output of    cout << schedule2 + 1 << endl;    is:
Mo: free busy free free
Tu: free busy free free
We: free busy free free
Th: free busy free free
Fr: busy busy busy busy
Sa: free busy free free
Su: free busy free free

Schedule 1 after self-assignment:
Mo: busy free free free
Tu: busy free free free
We: busy free free free
Th: busy free free free
Fr: busy busy busy busy
Sa: busy free free free
Su: busy free free free

Schedule 3 created as multiplication (common busy hours) of schedule 1 and schedule 2:
Printing using display_sch() function:
    Monday free free free free
   Tuesday free free free free
 Wednesday free free free free
  Thursday free free free free
    Friday busy busy busy busy
  Saturday free free free free
    Sunday free free free free

schedule 2 and schedule 3 have the same number of busy slots.

Schedule 1 after   schedule1 = schedule1 + schedule2;
Mo: busy free free free
Tu: busy free free free
We: busy free free free
Th: busy free free free
```

```
Fr: busy busy busy busy
Sa: busy free free free
Su: busy free free free

Schedule 2 after toggling one element:
Mo: free free free free
Tu: free free free free
We: free free free free
Th: free free free free
Fr: busy busy busy busy
Sa: free free busy free
Su: free free free free

schedule 1 after   schedule1 = schedule1 + schedule2 + schedule3;
Mo: busy free free free
Tu: busy free free free
We: busy free free free
Th: busy free free free
Fr: busy busy busy busy
Sa: busy free busy free
Su: busy free free free

Schedule 4 has been created with 8 slots.
Mo: free free free free free free free free
Tu: free free free free free free free free
We: free free free free free free free free
Th: free free free free free free free free
Fr: free free free free free free free free
Sa: free free free free free free free free
Su: free free free free free free free free

schedule 1, 2 and 4 after   schedule1 = schedule2 = schedule4;
schedule 1:
Mo: free free free free free free free free
Tu: free free free free free free free free
We: free free free free free free free free
Th: free free free free free free free free
Fr: free free free free free free free free
Sa: free free free free free free free free
Su: free free free free free free free free

schedule 2:
Mo: free free free free free free free free
Tu: free free free free free free free free
We: free free free free free free free free
Th: free free free free free free free free
Fr: free free free free free free free free
```

```
Sa: free free free free free free free free
Su: free free free free free free free free

schedule 4:
Mo: free free free free free free free free
Tu: free free free free free free free free
We: free free free free free free free free
Th: free free free free free free free free
Fr: free free free free free free free free
Sa: free free free free free free free free
Su: free free free free free free free free
```

Sample Run 3

```
Creating an empty schedule and displaying it.
Mo:
Tu:
We:
Th:
Fr:
Sa:
Su:

Schedule s after calling the full_schedule() function
Mo: busy
Tu: busy
We: busy
Th: busy
Fr: busy
Sa: busy
Su: busy

Please enter the number of slots per day:
7
Throughout the program, whenever you are asked to enter a day, enter the day name with initial letter capitalized.

Initial schedule 1 (all slots are free by default):
Mo: free free free free free free free
Tu: free free free free free free free
We: free free free free free free free
Th: free free free free free free free
Fr: free free free free free free free
Sa: free free free free free free free
Su: free free free free free free free

Please enter the day of the week that you want to make busy in schedule 1:
Sunday
```

```
Please enter the slot number that you want to change to busy in all days of the week in schedule 1:
The valid range is [0,6]:
```
6
```
Schedule 1 after changing all the slots of Sunday and slot number 6 to busy:
Mo: free free free free free free busy
Tu: free free free free free free busy
We: free free free free free free busy
Th: free free free free free free busy
Fr: free free free free free free busy
Sa: free free free free free free busy
Su: busy busy busy busy busy busy busy

Please enter the day of the week that you want to make busy in schedule 2:
```
Wednesday
```
Schedule 2 after changing all the slots of Wednesday to busy:
Mo: free free free free free free free
Tu: free free free free free free free
We: busy busy busy busy busy busy busy
Th: free free free free free free free
Fr: free free free free free free free
Sa: free free free free free free free
Su: free free free free free free free

Please enter the slot number that you want to change to busy in all days of the week in schedule 2:
The valid range is [0,6]:
```
6
```
The output of    cout << schedule2 + 6 << endl;    is:
Mo: free free free free free free busy
Tu: free free free free free free busy
We: busy busy busy busy busy busy busy
Th: free free free free free free busy
Fr: free free free free free free busy
Sa: free free free free free free busy
Su: free free free free free free busy

Schedule 1 after self-assignment:
Mo: free free free free free free busy
Tu: free free free free free free busy
We: free free free free free free busy
Th: free free free free free free busy
Fr: free free free free free free busy
Sa: free free free free free free busy
Su: busy busy busy busy busy busy busy

Schedule 3 created as multiplication (common busy hours) of schedule 1 and schedule 2:
Printing using display_sch() function:
    Monday free free free free free free free
   Tuesday free free free free free free free
```

```
 Wednesday free free free free free free busy
  Thursday free free free free free free free
    Friday free free free free free free free
  Saturday free free free free free free free
    Sunday free free free free free free free

schedule 3 is less busy than schedule 2.

Schedule 1 after   schedule1 = schedule1 + schedule2;
Mo: free free free free free free busy
Tu: free free free free free free busy
We: busy busy busy busy busy busy busy
Th: free free free free free free busy
Fr: free free free free free free busy
Sa: free free free free free free busy
Su: busy busy busy busy busy busy busy

Schedule 2 after toggling one element:
Mo: free free free free free free free
Tu: free free free free free free free
We: busy busy busy busy busy busy busy
Th: free free free free free free free
Fr: free free free free free free free
Sa: free free free busy free free free
Su: free free free free free free free

schedule 1 after   schedule1 = schedule1 + schedule2 + schedule3;
Mo: free free free free free free busy
Tu: free free free free free free busy
We: busy busy busy busy busy busy busy
Th: free free free free free free busy
Fr: free free free free free free busy
Sa: free free free busy free free busy
Su: busy busy busy busy busy busy busy

Schedule 4 has been created with 11 slots.
Mo: free free free free free free free free free free free
Tu: free free free free free free free free free free free
We: free free free free free free free free free free free
Th: free free free free free free free free free free free
Fr: free free free free free free free free free free free
Sa: free free free free free free free free free free free
Su: free free free free free free free free free free free

schedule 1, 2 and 4 after   schedule1 = schedule2 = schedule4;
schedule 1:
Mo: free free free free free free free free free free free
```

```
Tu: free free free free free free free free free free free
We: free free free free free free free free free free free
Th: free free free free free free free free free free free
Fr: free free free free free free free free free free free
Sa: free free free free free free free free free free free
Su: free free free free free free free free free free free

schedule 2:
Mo: free free free free free free free free free free free
Tu: free free free free free free free free free free free
We: free free free free free free free free free free free
Th: free free free free free free free free free free free
Fr: free free free free free free free free free free free
Sa: free free free free free free free free free free free
Su: free free free free free free free free free free free

schedule 4:
Mo: free free free free free free free free free free free
Tu: free free free free free free free free free free free
We: free free free free free free free free free free free
Th: free free free free free free free free free free free
Fr: free free free free free free free free free free free
Sa: free free free free free free free free free free free
Su: free free free free free free free free free free free
```

Sample Run 4

```
Creating an empty schedule and displaying it.
Mo:
Tu:
We:
Th:
Fr:
Sa:
Su:

Schedule s after calling the full_schedule() function
Mo: busy
Tu: busy
We: busy
Th: busy
Fr: busy
Sa: busy
Su: busy

Please enter the number of slots per day:
```
8
```
Throughout the program, whenever you are asked to enter a day, enter the day name with initial letter capitalized.

Initial schedule 1 (all slots are free by default):
Mo: free free free free free free free free
Tu: free free free free free free free free
We: free free free free free free free free
Th: free free free free free free free free
Fr: free free free free free free free free
Sa: free free free free free free free free
Su: free free free free free free free free

Please enter the day of the week that you want to make busy in schedule 1:
```
Tuesday
```
Please enter the slot number that you want to change to busy in all days of the week in schedule 1:
The valid range is [0,7]:
```
2
```
Schedule 1 after changing all the slots of Tuesday and slot number 2 to busy:
Mo: free free busy free free free free free
Tu: busy busy busy busy busy busy busy busy
We: free free busy free free free free free
Th: free free busy free free free free free
Fr: free free busy free free free free free
Sa: free free busy free free free free free
Su: free free busy free free free free free

Please enter the day of the week that you want to make busy in schedule 2:
```

Saturday

```
Schedule 2 after changing all the slots of Saturday to busy:
Mo: free free free free free free free free
Tu: free free free free free free free free
We: free free free free free free free free
Th: free free free free free free free free
Fr: free free free free free free free free
Sa: busy busy busy busy busy busy busy busy
Su: free free free free free free free free

Please enter the slot number that you want to change to busy in all days of the week in schedule 2:
The valid range is [0,7]:
```
5

```
The output of    cout << schedule2 + 5 << endl;    is:
Mo: free free free free free busy free free
Tu: free free free free free busy free free
We: free free free free free busy free free
Th: free free free free free busy free free
Fr: free free free free free busy free free
Sa: busy busy busy busy busy busy busy busy
Su: free free free free free busy free free

Schedule 1 after self-assignment:
Mo: free free busy free free free free free
Tu: busy busy busy busy busy busy busy busy
We: free free busy free free free free free
Th: free free busy free free free free free
Fr: free free busy free free free free free
Sa: free free busy free free free free free
Su: free free busy free free free free free

Schedule 3 created as multiplication (common busy hours) of schedule 1 and schedule 2:
Printing using display_sch() function:
    Monday free free free free free free free free
   Tuesday free free free free free free free free
 Wednesday free free free free free free free free
  Thursday free free free free free free free free
    Friday free free free free free free free free
  Saturday free free busy free free free free free
    Sunday free free free free free free free free

schedule 3 is less busy than schedule 2.

Schedule 1 after   schedule1 = schedule1 + schedule2;
Mo: free free busy free free free free free
Tu: busy busy busy busy busy busy busy busy
We: free free busy free free free free free
Th: free free busy free free free free free
```

```
Fr: free free busy free free free free free
Sa: busy busy busy busy busy busy busy busy
Su: free free busy free free free free free

Schedule 2 after toggling one element:
Mo: free free free free free free free free
Tu: free free free free free free free free
We: free free free free free free free free
Th: free free free free free free free free
Fr: free free free free free free free free
Sa: busy busy busy busy free busy busy busy
Su: free free free free free free free free

schedule 1 after   schedule1 = schedule1 + schedule2 + schedule3;
Mo: free free busy free free free free free
Tu: busy busy busy busy busy busy busy busy
We: free free busy free free free free free
Th: free free busy free free free free free
Fr: free free busy free free free free free
Sa: busy busy busy busy busy busy busy busy
Su: free free busy free free free free free

Schedule 4 has been created with 12 slots.
Mo: free free free free free free free free free free free free
Tu: free free free free free free free free free free free free
We: free free free free free free free free free free free free
Th: free free free free free free free free free free free free
Fr: free free free free free free free free free free free free
Sa: free free free free free free free free free free free free
Su: free free free free free free free free free free free free

schedule 1, 2 and 4 after   schedule1 = schedule2 = schedule4;
schedule 1:
Mo: free free free free free free free free free free free free
Tu: free free free free free free free free free free free free
We: free free free free free free free free free free free free
Th: free free free free free free free free free free free free
Fr: free free free free free free free free free free free free
Sa: free free free free free free free free free free free free
Su: free free free free free free free free free free free free

schedule 2:
Mo: free free free free free free free free free free free free
Tu: free free free free free free free free free free free free
We: free free free free free free free free free free free free
Th: free free free free free free free free free free free free
Fr: free free free free free free free free free free free free
```

```
Sa: free free free free free free free free free free free free
Su: free free free free free free free free free free free free

schedule 4:
Mo: free free free free free free free free free free free free
Tu: free free free free free free free free free free free free
We: free free free free free free free free free free free free
Th: free free free free free free free free free free free free
Fr: free free free free free free free free free free free free
Sa: free free free free free free free free free free free free
Su: free free free free free free free free free free free free
```

Good Luck!
Albert Levi, Ahmed Salem