Assignment 2, ENGG1111

If you have any questions regarding this assignment, please post to the Moodle discussion forum.

Introduction

In this assignment you will solve six tasks and, as in Assignment 1, you will use a tester – *test.cpp* – to test your code. The tester employs a small number of test cases. Note that we will use additional test cases when marking your assignment submission.

Each task requires you to write a number of functions to create a solution to the problem. Those functions are called from the *main()* function supplied for each task. **You must not modify any of the *main()* functions provided. If you do so you will get 0 marks for that task.**

Except for the *main()* function supplied for each task, there are no other function declarations or definitions provided. As part of each task you must first examine the problem and its corresponding *main()* function. Then work out the required form of the function headers for your function definitions. The skeleton programs provided will not compile until you supply the missing functions.

You are welcome, if you wish, to break down problems further and write helper functions to be called from the bodies of your other functions. **You must not, however, use any part of the C++ standard library other than *iostream*.**

In all test case examples shown in this worksheet, user input is shown in *blue italics* .

Files you will *edit, compile and run*:

Filename	Description
1.cpp	Task 1
2.cpp	Task 2
3.cpp	Task 3
4.cpp	Task 4
5.cpp	Task 5
6.cpp	Task 6

Files you will *compile* and *run*:

Filename	Description
test.cpp	Tester

Folders for auto grading

There are three folders. Do not edit any files in these folders.

Folder name	Description
expectedOutput	Expected output files
input	Input files for test cases
yourOutput	Output files produced by your program

Submission

Zip the files *1.cpp, 2.cpp, …, 6.cpp* to the file *submit.zip* and upload the zip file to Moodle before the deadline. Make sure to upload only those source files. Do not submit any other files. There should only be the 6 files named above in the zip archive. Please check carefully that you have submitted the correct files. We suggest you download your submitted zip file, extract the programs, and check again for correctness. Do not submit any other files from *a2.zip*. You will receive 0 marks for this assignment if you submit incorrect files. Resubmission after the deadline is not allowed.

Deadline

The deadline is November 13, 23:55. No late submissions can be accepted.

Task 1

Available marks: 15/100

In this task you will complete a program to shift the elements of an array to the left or to the right by a given number of indexed locations. The shift will be done with wrap-around, meaning that when a value is shifted off one end of the array it wraps around to the other end. An example will make this clear:

Consider an *int* array, *my_array*, of length 3 initialized with values {10, 20, 30}.

my_array[0]	*my_array[1]*	*my_array[2]*
10	20	30

Shifting the elements of *my_array* one location to the left with wrap-around gives the following result:

my_array[0]	*my_array[1]*	*my_array[2]*
20	30	10

Further shifting the resulting array contents two locations to the right with wrap-around gives:

my_array[0]	*my_array[1]*	*my_array[2]*
30	10	20

The number of locations to shift will be indicated by means of a signed integer using the convention that a negative number indicates a left shift and a positive number indicates a right shift.

In *1.cpp*, write definitions of functions *read_array()*, *read_number_of_shifts()* and *shift_array()* to satisfy the following specifications:

- *read_array()*: reads a list of integers supplied by the user. The list is read into the array passed as an argument. The list will contain exactly SIZE integers, where SIZE is a global constant and thus is available for use in your function definitions. SIZE is guaranteed to be initialized with a value greater than 0. In the given skeleton program it has a value of 8.
- *read_number_of_shifts()*: reads and returns an integer supplied by the user as the number of locations through which the array is to be shifted.
- *shift_array()*: shifts the elements of the array passed as the first argument by the number of locations passed as the second argument.

The completed program will read a list of 8 integers and a number of locations to shift it. It will then display the shifted list. You may assume that every integer supplied by the user can be represented as an *int*.

Test Cases

1_1

```
0 1 2 3 4 5 6 7
5
3 4 5 6 7 0 1 2
```

1_2

```
0 1 2 3 4 5 6 7
-5
5 6 7 0 1 2 3 4
```

1_3

```
0 1 2 3 4 5 6 7
8
0 1 2 3 4 5 6 7
```

1_4

```
0 1 2 3 4 5 6 7
-16
0 1 2 3 4 5 6 7
```

1_5

```
0 1 2 3 4 5 6 7
23
1 2 3 4 5 6 7 0
```

1_6

```
0 1 2 3 4 5 6 7
-42
2 3 4 5 6 7 0 1
```

Task 2

Available marks: 15/100

In *2.cpp*, write definitions of functions *read_word(), read_target()* and *delete_target()* to satisfy the following specifications:

- *read_word()*: reads an integer supplied by the user into the variable passed as the second argument. The integer is guaranteed to be in {1, 2,...,10} and represents *word_length*, the length of a word in characters. It then reads a word supplied as *word_length* characters, each separated by whitespace. The characters are read into the array passed as the first argument.
- *read_target()*: reads and returns a single character supplied by the user. The character represents a target for deletion from the word supplied previously. The target character may or may not be present in the word and may be present multiple times.
- *delete_target()*: deletes all occurrences of the target character from the array passed as the first argument. To satisfy the needs of *main()* your function must change the contents of the array to reflect the deletion and modify the value of *word_length,* passed as the second argument, accordingly. The target character is passed as the third argument.

The completed program will read the characters of a word and a target for deletion, and then display the word with all occurrences of the target character removed.

Test Cases

2_1

```
8
E N G G 1 1 1 1
1
ENGG
```

2_2

```
7
s t u d e n t
t
suden
```

2_3

```
10
p r o g r a m m e r
b
programmer
```

2_4

```
3
w w w
w
```

Task 3

Available marks: 20/100

In *3.cpp*, write definitions of functions *read_lists()* and *display_common_integers()* to satisfy the following specifications:

- *read_lists()*: reads three lists of integers supplied by the user into the arrays passed as the first three arguments. An integer may appear multiple times in a list and the integers in each list will be supplied in non-decreasing order. The lists will be of known equal size. That size is passed as the fourth argument.
- *display_common_integers()*: displays all integers that appear in every one of the three arrays passed as arguments. They are displayed in order of occurrence, with each on a new line. If an integer appears in all three arrays multiple times, then it will be displayed multiple times. If no integer appears in all three arrays, then the text "*None*" will be displayed. The size of the arrays is passed as the fourth argument.

The completed program will read three lists of 8 integers and will display all integers that are common to all three lists. You may assume that every integer supplied by the user can be represented as an *int*.

Test Cases

3_1

```
0 1 2 3 5 7 11 13
0 5 6 7 8 9 10 11
0 4 8 9 10 11 23 42
0
11
```

3_2

```
15 20 20 29 33 33 42 50
-1 0 15 29 30 31 33 33
4 6 12 15 15 33 33 50
15
33
33
```

3_3

```
-8 -7 -6 -5 -4 -3 -2 -1
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
None
```

3_4

```
-8 -7 -6 -5 -4 -3 -2 -1
-1 0 1 2 3 4 5 6
-1 0 1 2 3 4 5 6
-1
```

Task 4

Available marks: 10/100

In *4.cpp*, write definitions of functions *read_matrix()* and *display_inner_cw_rotated()* to satisfy the following specifications:

- *read_matrix()*: reads *dim*, an integer supplied by the user, into the variable passed as the second argument, where *dim* is guaranteed to be in {3, 4, ...,10}. Then reads values of elements of a *dim* x *dim* square matrix of integers, each separated by whitespace. The values are supplied by the user row by row and are read into the 2D array passed as the first argument. The size of the array is *MAXDIM* x *MAXDIM*, where *MAXDIM* is a global constant and thus is available for use in your function definitions. *MAXDIM* is initialized with a value of 10 and, therefore, the array is large enough to hold the maximum number of values that the user will supply.
- *display_inner_cw_rotated()*: takes, as a first argument, a 2D array containing a square matrix and, as a second, the dimension, *dim*, of that matrix. It then displays the values of a *dim-2* x *dim-2* matrix formed by excluding the outermost rows and columns of the original matrix (that is, those that form its outer perimeter) and then rotating the reduced matrix clockwise through ninety degrees. The elements of the resulting matrix are displayed row by row with a space after each element.

The completed program will read the dimension, *dim*, and the elements of a *dim* x *dim* matrix and then display the elements of its inner *dim-2* x *dim-2* matrix rotated clockwise through 90 degrees.

Test Cases

4_1

```
3
11 12 13
21 22 23
31 32 33
22
```

4_2

```
4
11 12 13 14
21 22 23 24
31 32 33 34
41 42 43 44
32 22
33 23
```

4_3

```
5
11 12 13 14 15
21 22 23 24 25
31 32 33 34 35
41 42 43 44 45
51 52 53 54 55
42 32 22
43 33 23
44 34 24
```

Task 5

Available marks: 15/100

In *5.cpp*, write definitions of functions *read_matrix()* and *display_with_fill()* to satisfy the following specifications:

- *read_matrix()*: reads *dim*, an integer supplied by the user, into the variable passed as the second argument, where *dim* is guaranteed to be in {2, 3, ...,10}. Then reads values of elements of a *dim* x *dim* square matrix of characters, each separated by whitespace. The values are supplied by the user row by row and are read into the 2D array passed as the first argument. As in Task 4, the size of the array is *MAXDIM* x *MAXDIM*, where *MAXDIM* is a global constant initialized with a value of 10.
- *display_with_fill()*: takes as arguments a 2D array containing a square matrix of characters and the dimension, *dim*, of that matrix. It then displays a modified *dim* x *dim* matrix in which all elements in the same row or column as an element that originally contained the character *'x'* now also contain an *'x'*. The elements of the modified matrix will be displayed row by row with a trailing space after each element.

The completed program will read the dimension *dim* and the elements of a *dim* x *dim* matrix, then display the corresponding matrix in which rows and columns that originally contained at least one *'x'* are now filled with *'x'*.

Test Cases

5_1

```
2
a b
d x
a x
x x
```

5_2

```
3
a b c
d x f
g h i
a x c
x x x
g x i
```

5_3

```
5
a b c d e
x l m n o
k l m x o
x l m n o
a b c d e
x b c x e
x x x x x
x x x x x
x x x x x
x b c x e
```

Task 6

Available marks: 25/100

In Task 7 of Assignment 1 your program played the first four moves of Tic-Tac-Toe (Noughts and Crosses). Here you will complete a program that, given a board of a game in progress, will determine if either player has won.

Background of gameplay

The 9 squares of the board are numbered as in Assignment 1:

```
1 2 3
4 5 6
7 8 9
```

In this Task, the board is represented by a 1D *char* array of size 9. The first element of the array (subscript 0) represents Square 1, the second element (subscript 1) represents Square 2, and so on.

At the start of a game, each element of the array is initialized with a *char* identifying its square. Thus the first element is initialized with *'1'*, the second with *'2'*, and so on until the last element is initialized with *'9'*. When a player makes a valid move, the value of the corresponding element is changed to either *'x'* or *'o'* depending on which player made the move.

Your task

Given a game board to evaluate, there are four possible outcomes:

- player *x* has won by occupying 3 squares in a straight line.
- player *o* has won by occupying 3 squares in a straight line.
- neither player has won and there are no more unoccupied squares – the game is a draw.
- neither player has won and there are still unoccupied squares – the game is not finished yet.

In *6.cpp*, write definitions of functions *read_board()* and *evaluate_board()* to satisfy the following specifications:

- *read_board()*: reads a game board supplied by the user into the array passed as the first argument. Each supplied value is either the square's number if the square is unoccupied, or is *'x'* or *'o'* indicating which player occupied it. The values are supplied three per line and are separated by whitespace.
- *evaluate_board()*: takes an array representing a game board as its first argument and returns the winner (*'x'* or *'o'*) or, if there is no winner, returns *'d'* if the game is a draw or *'u'* if the game is unfinished. [Hint: There are only eight winning combinations of squares. That number is sufficiently small for those combinations to be hard-coded into your function definition.]

You may assume that the board supplied by the user represents a valid, reachable game position and, as a consequence, there will never be more than one player with a winning combination of squares.

The completed program will read a board and display an evaluation of the board's current state.

Test Cases

6_1

```
o 2 x
o x 6
x 8 9
Player x wins!!
```

6_2

```
x x o
x x o
7 o o
Player o wins!!
```

6_3

```
x o o
o x x
x x o
No winner. It’s a draw!!
```

6_4

```
o x o
4 x 6
7 o x
No winner, but game is not over yet. Please continue.
```

After submitting your assignment, check it by downloading it yourself. Unzip the archive in our development environment and run the tester to make sure all test cases pass:

```
Task number (1 to 6, or 0 to run all): 0
1_1: PASSED
1_2: PASSED
1_3: PASSED
1_4: PASSED
1_5: PASSED
1_6: PASSED

2_1: PASSED
2_2: PASSED
2_3: PASSED
2_4: PASSED

3_1: PASSED
3_2: PASSED
3_3: PASSED
3_4: PASSED

4_1: PASSED
4_2: PASSED
4_3: PASSED

5_1: PASSED
5_2: PASSED
5_3: PASSED

6_1: PASSED
6_2: PASSED
6_3: PASSED
6_4: PASSED
```