CS204 – Fall 2021 - Sabancı University
Homework #3 – Bus Lines
Due: November 11th, Thursday, 23:55

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

You CANNOT collaborate with your friends and discuss solutions. You have to write down the code on your own. Plagiarism will not

1. Introduction

The aim of this homework is to use a hybrid data structure consisted of singly linked lists and doubly linked lists. You will employ these structures to implement a bus line/stop management system. Since the number of bus lines and stops cannot be known in advance and changes from case to case, you will use dynamic memory allocation, i.e., new/delete operations to dynamically allocate and delete the corresponding nodes of the structures. Don't forget, we don't want memory leaks in the software so each memory region allocated by a new operation should be freed by a delete operation once it is not needed anymore.

2. A sample input file

İETT wants you to implement a bus line management and navigation system. Before, they were keeping the track of their data in a notepad text file. A sample of a such file named busLines.txt is given in Fig.1:

```
133U: Sabanci Nakliyeciler Viaport Pendik Kartal
132B: Akfirat Kimyaciler Birkent Viaport Pendik
15K: Emniyet Beyazit Vergi_Dairesi Sabiha_Gokcen Eminonu
166A: Seyhli Birkent Cam_Evleri Benzinlik Nakliyeciler
500T: Guzelyali Pendik_Koprusu Kartal_Koprusu Bostanci_Koprusu
```

Fig.1 The content of the busLines.txt input file

Each bus-line of the input file is given in a separate line which ends in a newline i.e. '\n' character. The first item (word) of each line is the bus line name which ends with ':', proceeded by the bus stop names through which that bus line passes by. The bus line and bus stops of each bus line are separated from each other by an empty space (one or more). Each line will always contain the bus line name and at least one bus stop, in addition, bus lines and bus stop names will always be a single word (e.g. "Sabanci", "Cam_Evleri", "Kartal_Koprusu", "Sabiha_Gokcen" etc.).

3. The data structure and its initialization

You will use a 2D bus line linked list data structure. The lines will be stored in a singly linked list and the bus stops of each bus line will be stored in a doubly linked list. This done so since the route of the bus lines is the same in both directions, i.e they start from their first bus stop, go to the last one and come back through the same route. Fig.2 gives a visual description of the data structure that you should construct having in mind the input file of Fig.1. Thus, the first thing that you should do when the program starts is to open the input file (which was described in Section 2 and Fig.1) and from it construct the data structure illustrated in Fig.2.

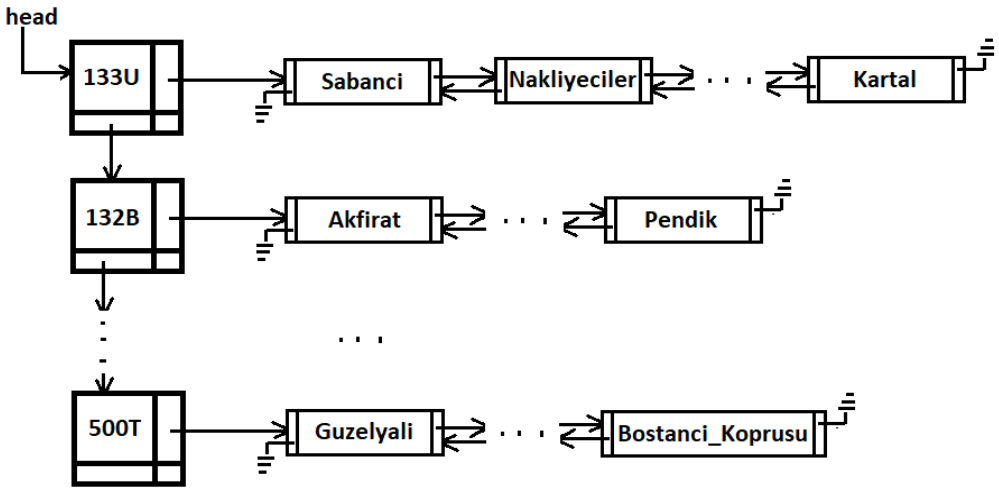


Fig.2 Illustration of the data structure that should be built if the input file is busLines.txt given in Fig.1

There are a few things that you should take care of in your program. An empty bus line cannot exist, i.e. each bus line needs to have at least one bus stop. So when the last bus stop of a bus line is deleted (see the rest of the assignment), the bus line also needs to be deleted. Also, there cannot be 2 bus lines with the same name, neither two bus stops within the same bus line with the same name. You can assume that everything about the input file is correct.

4. The Main Menu

The node/list structures and functions to print the main menu and to process the input are given below to make your job easier. If you wish, you can use them as is, but you may want to (we prefer that you do) add appropriate constructors to the struct definitions. The consistencyCheck() function is checking if your whole structure of doubly and singly linked lists is consistent (i.e. no broken links, etc.).

```
struct busStop {
	string busStopName;
	busStop *left;
	busStop *right;
};
struct busLine {
	string busLineName;
	busLine *next;
	busStop *busStops;
};
busLine* head = nullptr;
```

```
void printMainMenu() {
	cout << endl;
	cout <<"I*********************************************I"<<endl
	<<"I 0 - EXIT PROGRAM I"<<endl
	<<"I 1 - PRINT LINES I"<<endl
	<<"I 2 - ADD BUS LINE I"<<endl
	<<"I 3 - ADD BUS STOP I"<<endl
	<<"I 4 - DELETE BUS LINE I"<<endl
	<<"I 5 - DELETE BUS STOP I"<<endl
	<<"I 6 - PATH FINDER I"<<endl
	<<"I*********************************************I"<<endl
	<<">>";
cout << endl;
}


bool consistencyCheck() {
	busLine* currBL = head; //a global variable
	while(currBL) {
		busStop* currBS = currBL->busStops;
		while(currBS) {
			busStop* rightBS = currBS->right;
			if(rightBS && rightBS->left != currBS) {
				cout << "Inconsistency for " << currBL->busLineName
				     << " " << currBS->busStopName << endl;
				return false;
			}
			currBS = currBS->right;
		}
		currBL = currBL->next;
	}
	return true;
}



void processMainMenu() {
	char input;
	do
	{
		If(!consistencyCheck()) {
			cout << "There are inconsistencies. Exit." << endl;
			return;
		}
		printMainMenu();
		cout << "Please enter your option " << endl;
		cin >> input;
		switch (input) {
			case '0':
				cout << "Thanks for using our program" << endl;
				return;
			case '1':
				printBusLines();
				break;
			case '2':
				addBusLine();
				break;
```

```
                        case '3':
                                addBusStop();
                                break;
                        case '4':
                                deleteBusLine();
                                break;
                        case '5':
                                deleteBusStop();
                                break;
                        case '6':
                                pathfinder();
                                break;
                        default:
                                cout << "Invalid option: please enter again" << endl;
                        } //switch
        } while(true);
}//processMainMenu
```

Below we describe the main menu functions with some examples to better understand them: User input are given in bold.

4.1) **Print Bus Lines**

This menu item will print the bus line table. Each bus line will be printed in a line. The bus line name will be the first item in the line followed by a ':'. Then the "<->" symbol will be used to separate the bus stop names.

Execution:
Please enter your option
1

133U: Sabanci <-> Nakliyeciler <-> Viaport <-> Pendik <-> Kartal
132B: Akfirat <-> Kimyaciler <-> Birkent <-> Viaport <-> Pendik
15K: Emniyet <-> Beyazit <-> Vergi_Dairesi <-> Sabiha_Gokcen <-> Eminonu
166A: Seyhli <-> Birkent <-> Cam_Evleri <-> Benzinlik <-> Nakliyeciler
500T: Guzelyali <-> Pendik_Koprusu <-> Kartal_Koprusu <-> Bostanci_Koprusu

4.2) **Add Bus Line**

This menu item will be used to add a new bus line. Firstly, the name of the bus line will be asked (0 is reserved to go back to main menu) and if the bus line already exists in the bus line table it will be asked again (Execution 2). If the name is not there, the bus stop names will be asked (again duplicate bus stop names are not allowed, see busStop3 in Execution 1). 0 will be used to complete the operation. Empty bus lines with no bus stops are not allowed (Execution 2). If everything is OK, the entered bus line information will be shown (Execution 1) and the user will be asked to confirm. If s/he enters y/Y then the operation will be completed by adding the new bus line to the first row of the bus line table and the whole bus line information will be printed. Otherwise, if s/he doesn't want to go with it she enters n/N. After the confirmation step, the main menu will appear (see the while(true) loop in processMainMenu() function in the codes above).

Execution 1:
Please enter your option
2
Enter the name of the new bus line (0 for exit to main menu).
myBusLine
Enter the name of the next bus stop (enter 0 to complete)
busStop1
Enter the name of the next bus stop (enter 0 to complete)
busStop2
Enter the name of the next bus stop (enter 0 to complete)
busStop3
Enter the name of the next bus stop (enter 0 to complete)
busStop3
Bus stop already exists in the line
Enter the name of the next bus stop (enter 0 to complete)
0
The bus line information is shown below
myBusLine: busStop1 <-> busStop2 <-> busStop3
Are you sure? Enter (y/Y) for yes (n/N) for no?
Y

myBusLine: busStop1 <-> busStop2 <-> busStop3
133U: Sabanci <-> Nakliyeciler <-> Viaport <-> Pendik <-> Kartal
132B: Akfirat <-> Kimyaciler <-> Birkent <-> Viaport <-> Pendik
15K: Emniyet <-> Beyazit <-> Vergi_Dairesi <-> Sabiha_Gokcen <-> Eminonu
166A: Seyhli <-> Birkent <-> Cam_Evleri <-> Benzinlik <-> Nakliyeciler
500T: Guzelyali <-> Pendik_Koprusu <-> Kartal_Koprusu <-> Bostanci_Koprusu

Execution 2:
Please enter your option
2
Enter the name of the new bus line (0 for exit to main menu).
myBusLine
Bus line already exists: enter a new one (0 for exit)
myBusLine2
Enter the name of the next bus stop (enter 0 to complete)
0
You are not allowed to add an empty bus line

4.3) **Add Bus Stop**

This menu item will be used to add a new bus stop to an existing bus line. First an existing bus line name will be asked. If it does not exist in the list, then main the menu will appear (Execution 1). If it exists, the bus line's information will be shown (Execution 2/3) and the new bus stop name will be requested. If the bus stop already exists in the bus line, the main menu will appear (Execution 2). Otherwise, the previous bus stop name for the new bus stop will be asked. 0 is reserved to add the new bus stop to the doubly linked list as the first one (Execution 3). If the name does not exist, then it will be asked again (Execution 3). When an existing bus stop is given the program inserts the new bus stop just after it and show the updated bus line data structure.

Execution 1**:**
Please enter your option
3
Enter the name of the bus line to insert a new bus stop (0 for main menu)
myBusLine2
Bus line cannot be found. Going back to previous menu.

Execution 2:
Please enter your option
3
Enter the name of the bus line to insert a new bus stop (0 for main menu)
myBusLine
The bus line information is shown below
myBusLine: busStop1 <-> busStop2 <-> busStop3
Enter the name of the new bus stop
busStop1
Bus stop already exists. Going back to previous menu.

Execution 3:
Please enter your option
3
Enter the name of the bus line to insert a new bus stop (0 for main menu)
myBusLine
The bus line information is shown below
myBusLine: busStop1 <-> busStop2 <-> busStop3
Enter the name of the new bus stop
busStop1.5
Enter the name of the previous bus stop to put the new one after it (0 to put the new one as the first bus stop)
busStop4
Bus stop does not exist. Typo? Enter again (0 for main menu)
busStop1

myBusLine: busStop1 <-> busStop1.5 <-> busStop2 <-> busStop3
133U: Sabanci <-> Nakliyeciler <-> Viaport <-> Pendik <-> Kartal
132B: Akfirat <-> Kimyaciler <-> Birkent <-> Viaport <-> Pendik
15K: Emniyet <-> Beyazit <-> Vergi_Dairesi <-> Sabiha_Gokcen <-> Eminonu
166A: Seyhli <-> Birkent <-> Cam_Evleri <-> Benzinlik <-> Nakliyeciler
500T: Guzelyali <-> Pendik_Koprusu <-> Kartal_Koprusu <-> Bostanci_Koprusu

4.4) **Delete a Bus Line**

This menu item will be used to delete an existing bus line. The bus line name will be asked from the user. If it does not exist main menu will appear (Execution 1). Otherwise, the bus line will be deleted and the updated bus line table will be shown before the main menu (Execution 2).

Execution 1:
Please enter your option
4
Enter the name of the bus line to delete
133
Bus line cannot be found. Going back to the previous (main) menu.

Execution 2:
Please enter your option
4
Enter the name of the bus line to delete
133U

myBusLine: busStop1 <-> busStop1.5 <-> busStop2 <-> busStop3
132B: Akfirat <-> Kimyaciler <-> Birkent <-> Viaport <-> Pendik
15K: Emniyet <-> Beyazit <-> Vergi_Dairesi <-> Sabiha_Gokcen <-> Eminonu
166A: Seyhli <-> Birkent <-> Cam_Evleri <-> Benzinlik <-> Nakliyeciler
500T: Guzelyali <-> Pendik_Koprusu <-> Kartal_Koprusu <-> Bostanci_Koprusu

4.5) **Delete a Bus Stop**

This menu item will be used to delete an existing bus stop. Firstly, a bus line will be requested as an input. If the bus line does not exist the main menu will appear (Execution 1). Otherwise, the bus line information will be shown (Execution 2) and the bus stop name will be requested (0 is reserved to go back to the main menu). If the bus stop does not exist in the bus line the name will be asked again (Execution 2). If the bus stop exists it will be deleted and the updated bus line table will be shown before the appearance of the main menu (Execution 2).

Execution 1:
Please enter your option
5
Enter the name of the bus line to delete a new bus stop (0 for main menu)
myBusLine2
Bus line cannot be found. Going back to previous (main) menu.

Execution 2:
Please enter your option
5
Enter the name of the bus line to delete a new bus stop (0 for main menu)
myBusLine
The bus line information is shown below
myBusLine: busStop1 <-> busStop1.5 <-> busStop2 <-> busStop3
Enter the name of the bus stop to delete (0 for main menu)
busStop5
Bus stop cannot be found. Enter the name of the bus stop to delete (0 for main menu)
busStop2

myBusLine: busStop1 <-> busStop1.5 <-> busStop3
132B: Akfirat <-> Kimyaciler <-> Birkent <-> Viaport <-> Pendik
15K: Emniyet <-> Beyazit <-> Vergi_Dairesi <-> Sabiha_Gokcen <-> Eminonu
166A: Seyhli <-> Birkent <-> Cam_Evleri <-> Benzinlik <-> Nakliyeciler
500T: Guzelyali <-> Pendik_Koprusu <-> Kartal_Koprusu <-> Bostanci_Koprusu

4.6) **Pathfinder**

This menu item will be used to find a path between two bus stops. Two bus stop names will be requested. If one does not exist, the main menu will appear after an appropriate message (Execution 1). Otherwise, the program will check if the two bus stops are both in a bus line and if this is true the corresponding path will be printed (see Execution 2, and note the reverse order). Otherwise, the user will be informed that such bus line does not exist (Execution 3).

BONUS: If no bus line having the two bus stops exists then interchanging bus lines will be investigated and printed if one found (Execution 4). You don't need to look for more than one interchange.

Execution 1:

Please enter your option
6
Where are you now?
busStop1
Where do you want to go?
busStop2
Bus stop does not exist in the table. Going back to previous menu.

Execution 2:

Please enter your option
6
Where are you now?
Benzinlik
Where do you want to go?
Birkent
You can go there by 166A: Benzinlik->Cam_Evleri->Birkent

Execution 3:

Please enter your option
6
Where are you now?
Benzinlik
Where do you want to go?
Pendik
Sorry no path from Birkent to Pendik could be found.

Execution 4 (BONUS):

Please enter your option
6
Where are you now?
Benzinlik
Where do you want to go?
Pendik
You can go there by 166A: Benzinlik->Cam_Evleri->Birkent
132B: Birkent->Viaport->Pendik

5. Other remarks

You can assume that the input file is correctly entered and that its information is also correct. You should just read the input file and construct the data structure of Fig.2. Furthermore, we don't check details like whether the input file is opened correctly, or having an extra space or lines when you display your outputs, etc. Our aim here is to make you familiar with simple operations over a more complex structure of doubly and singly linked lists, such as adding a node, deleting a node, searching the linked lists, displaying (printing) the linked lists, etc. In order to make it easier to you, we gave you the codes for the definitions of node structures for both the doubly and singly linked lists. Also, we gave you the `processMainMenu()` functions in which, based on the user choice, a corresponding function is called. It is up to you to write those function to behave in the way that it is described above. While writing your assignment, your program can easily have some bugs due to the improper way of adding or deleting nodes in the data structure illustrated in Fig.2. In order to make sure that your data structure is consistent and has no broken links, you can call the `consistencyCheck()` function given above whenever you encounter a bug. Although you might want to have other (and better) approaches, for this assignment you can declare the main head of the data structure in Fig.2 as a global variable.

Note!!! Perhaps the best approach would be to have an object oriented approach where everything is organized as a class. This means that you can have the head of the data structure of Fig.2 as a private variable of the class and the functions that are called inside the `processMainMenu()` are the public member functions of that class. It is up to you to decide on the approach anyway.

You can use any library that you find useful. In that case, besides your main cpp file, you should include those libraries in your zipped file that you are going to submit to SUCourse+. You can use functions/constructors, etc. as needed (it is up to you to decide how many of them and what they do). In order to avoid any inconsistencies, for any inquiries/questions related to the workflow and the general concept of the assignment you should contact your TA Seyedpouya Seyedkazemi by his email (seyedpouya@sabanciuniv.edu) or you can ask your questions through the course's WhatsApp group, where either the instructor or Seyedpouya will answer your inquiries to the best of their abilities. Of course, for technical help while coding your assignment, you are free to ask any of the TAs/LAs or the instructor during the online office hours.

IMPORTANT!

If your code does not compile, you will get **zero**. Please be careful about this and double check your code before submission.

General Rules and Guidelines about Homeworks

The following rules and guidelines will be applicable to all homeworks, unless otherwise noted.

How to get help?

You may ask questions to TAs (Teaching Assistants) and LAs (Learning assistants) of CS204. Office hours of TAs are at SUCourse+. Lectures and/or recitations will be partially dedicated to clarify the issues related to homework, so it is to your benefit to attend the lectures/recitations or watch the corresponding video lecture. The style of submitting the assignments is similar to CS201.

What and Where to Submit

Please see the detailed instructions below/in the next page. The submission steps will get natural/easy for later homeworks.

Grading and Objections

Careful about the semi-automatic grading: Your programs might be graded using a semi-automated system. Therefore you should follow the guidelines about input and output order; moreover you should also use same prompts as given in the Sample Runs. Otherwise semi-automated grading process will fail for your homework, and you may get a zero, or in the best scenario you will lose points.

Grading:

- ❑ Late penalty is 10 points off of the full grade and only one late day is allowed.
- ❑ **Having a correct program is necessary, but not sufficient to get the full grade. Comments, indentation, meaningful and understandable identifier names, informative introduction and prompts, and especially proper use of required functions, unnecessarily long program (which is bad) and unnecessary code duplications (which is also bad) will also affect your grade.**
- ❑ Please submit your own work only (even if it is not working). It is really easy to find out "similar" programs!
- ❑ For detailed rules and course policy on plagiarism, please check out http://people.sabanciuniv.edu/levi/cs204/policy_plagiarism.html. and keep in mind that

Plagiarism will not be tolerated!

Grade announcements: Grades will be posted in SUCourse+, and you will get an Announcement at the same time. You will find the grading policy and/or test cases in that announcement.

Grade objections: It is your right to object to your grade if you think there is a problem, but before making an objection please try the steps below and if you still think there is a problem, contact the TA that graded your homework from the email address provided in the comment section of your announced homework grade or attend the specified objection hour in your grade announcement.

- Check the comment section in the homework tab to see the problem with your homework.
- Download the .zip file you submitted to SUCourse+ and try to compile it.
- Check the test cases in the announcement and try them with your code.
- Compare your results with the given results in the announcement.

What and where to submit (IMPORTANT)

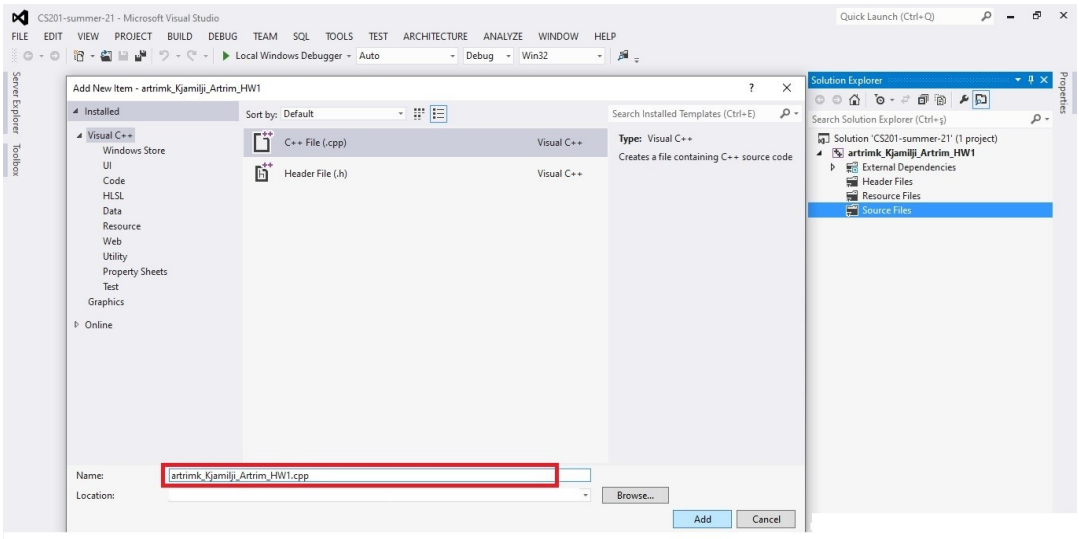
Submissions guidelines are below. Students are expected to strictly follow these guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade.

Add your name to the program: It is a good practice to write your name and last name somewhere in the beginning program (as a comment line of course).
Name your submission file:

- ❑ Use only English alphabet letters, digits or underscore in the file names. Do not use blank, Turkish characters or any other special symbols or characters.
- ❑ Name your cpp file that contains your program as follows:

"**SUCourseUserName_YourLastname_YourName_HWnumber.cpp**"

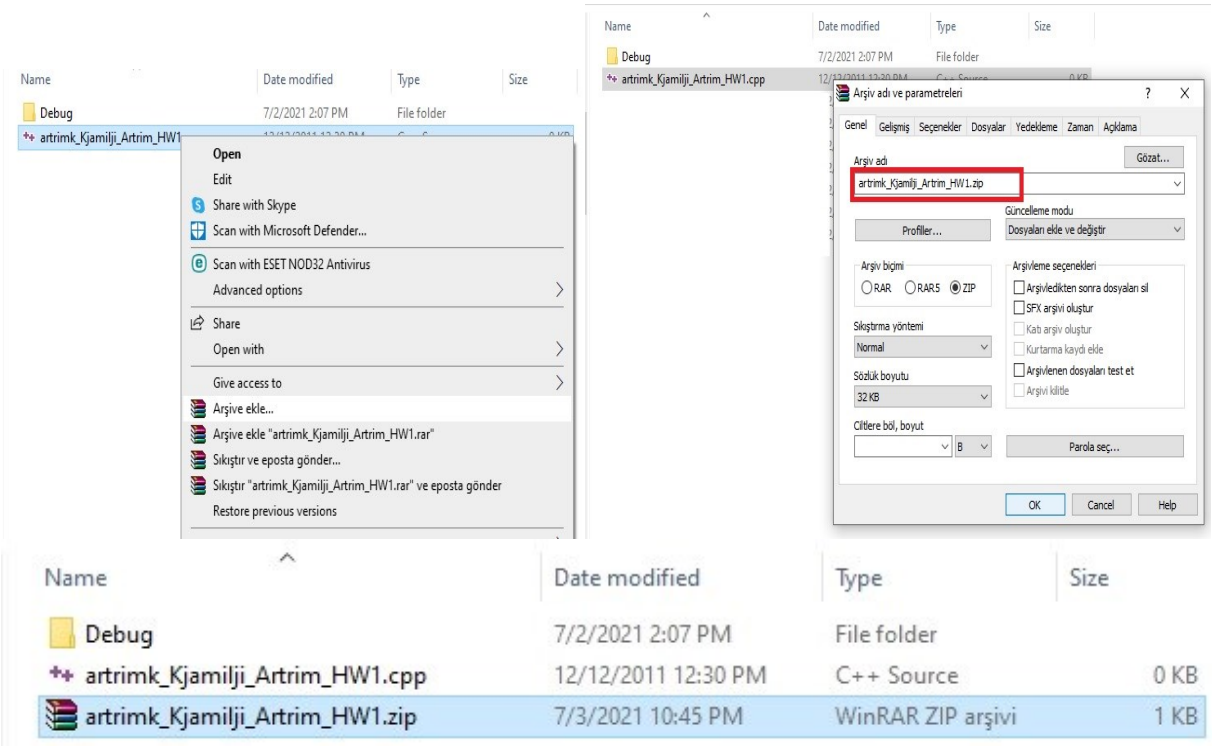
- ❑ Your SUCourse+ user name is actually your SUNet user name which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse+ user name is artrimk, your name is Artrim, your last name is Kjamilji, and you are uploading the first homework (HW1) to the corresponding assignment in SUCourse +, then the file name must be: **artrimk_Kjamilji_Artrim_hw1.cpp**



- ❑ Do not add any other character or phrase to the file name.
- ❑ Make sure that this file is the latest version of your homework program.
- ❑ Compress this cpp and, if necessary, other files such as libraries (both the .h and the corresponding .cpp files) that you might have used in your assignment using WINZIP or WINRAR programs. Please use "**zip**" compression. "rar" or another compression mechanism is NOT allowed. Our homework processing system works only with zip files. Therefore, make sure that the resulting compressed file has a zip extension.
- ❑ Check that your compressed file opens up correctly and it contains all of your **zipped** file(s). You will receive no credits if your compressed zip file does not expand or it does not contain the correct file(s).
- ❑ The naming convention of the zip file is the same as the cpp file (except the extension of the file of course). The name of the zip file should be as follows.

"**SUCourseUserName_YourLastname_YourName_HWnumber.zip**"

For example zubzipler_Zipleroglu_Zubeyir_hw1.zip is a valid name, but hw1_hoz_HasanOz.zip, HasanOzHoz.zip are NOT valid names.



Submission:

- ❑ Submit via SUCourse+ ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.). Click on "ASSIGNMENTS" at CS204 SUCourse+.

Resubmission:

- ❑ If you would like to resubmit your work, you should first remove the existing file(s). This step is very important. If you do not delete the old file(s), we will receive both files and the old one may be graded.

Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!
Artrim Kjamilji and Seyedpouya Seyedkazemi