CS204 – Fall 2021 - Sabancı University
Homework #2 – Add/Drop Courses
Due: October 30th, Saturday, 23:55

1. Introduction

In this homework, you are asked to implement a program that stores the courses taken by students organized (sorted in ascending order) according to the name of the courses during the add/drop period. This program must use a Linked List structure for storage. Initially, course names, course codes and IDs of the students who have taken a certain course(s) are going to be read from a text file. Afterwards the user will be able to add or drop students from the courses. The program details will be explained in the subsequent sections.

2. The Data Structure to be used

In this homework, you **must** use a linked list (regular one-way non-circular linked list) as your main data structure and each of the nodes of the linked list is dynamically allocated. The node structure of this list must have the following data members (if you want, you can add constructors to the structure).

```
struct courseNode
{
  string courseCode, courseName;
  vector<int> studentsAttendingIDs;
  courseNode * next;
}
```

In this node structure, the `StudentsAttendingIDs` integer vector is the list of the students' IDs who are attending the related course. A visualization of a single `courseNode` structure together with its info (data) fields is given in Fig. 1. Fig.3 shows a singly linked list of `courseNode`s with the corresponding data (info) for each node.

Please pay attention that, **you are not allowed to use arrays, extra vectors and similar containers (including extra files) in this homework; all data must be stored and processed within the linked list.** The only vector that you can use is the one inside the `courseNode` structure.

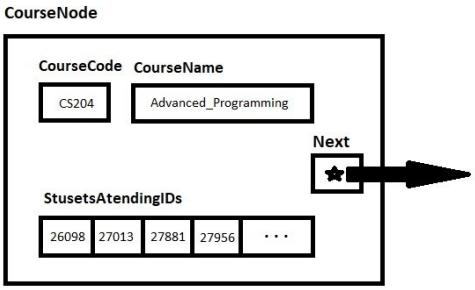


Fig. 1: A visualization of the `courseNode` structure

3. Program Initialization

Your program is going to start by getting an input from the user regarding the name of the .txt file that contains course enrollments information. You assume that the user always enters the correct file name, so no checks need to be done in this regard. After successfully opening the file, your program is going to start storing course information by reading the entered file line by line. Each line of the file contains three pieces of information regarding a course. The first entry of each line is a course code, which is a string. The second one is the course name (also a string). If the course name has more than one word, the course name is given by merging all of the words with underline so as to form a single string, e.g. Advanced_Programming. Thus, you can assume that there are no empty spaces in between the course names. And, the third entry of each line is an integer which is the ID of a student that has been enrolled to that particular course. You can assume the file contains correct inputs, so no input checks are required for the content of the .txt file either. An example input file is shown in Fig. 2.

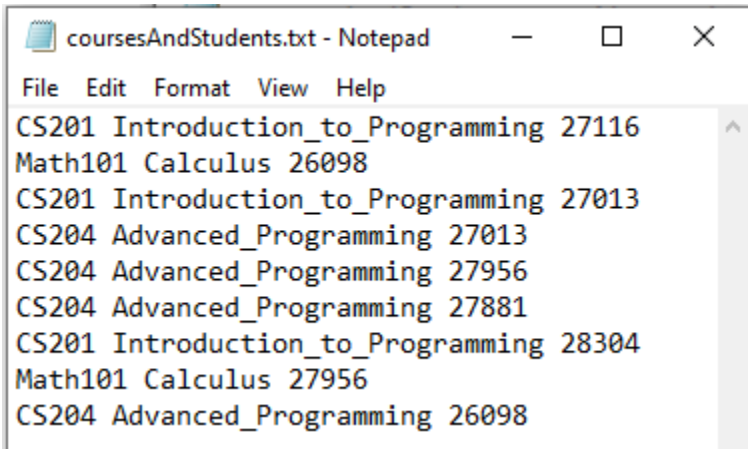


Fig. 2: Sample Input File (coursesAndStudents.txt)

In the `courseNode` structure, all of the students IDs with the same course code and course name must be stored in the same node. That is why we use a vector for attending students' IDs there. Thus, do not create a new node for a course if there already exists a node with that course's code and name; instead put it to the `AttendingStudentsIDs` vector of that node while preserving the order of IDs in asceding (increasing) order. Note that you should maintain the vector in each node in an ascending sorted fashion all the time, till the end of the program. No course has the same student ID repeated, i.e a certain student ID number can appear at most once inside the `AttendingStudentsIDs` vector of each `courseNode`. However, if there is not a node with new course's code and name, then create a new `courseNode` with the new name and code and push the student ID as the first entry of that node's `AttendingStudentsIDs` vector. This mechanism in the initialization phase of the program guarantees that (i) there will not be a node with an empty vector, (ii) there will not be two nodes with the same course name and course code in the data structure, and (iii) the same student cannot be enrolled twice in the same course; Moreover, your program should maintain the linked list in a sorted fashion with respect to courses' names all the time, till the end of the program. While inserting a new node, you have to preserve the order of the list all the time also This is going to be checked in grading, so follow these rules strictly. For example, if the input file is the one in Fig. 2, after reading and processing it by the program, output messages will be displayed as it is shown below (the bold texts are the user inputs).

```
Please enter file name:
coursesAndStudents.txt
Successfully opened file coursesAndStudents.txt
The linked list is created.
The initial list is:
CS204 Advanced_Programming: 26098 27013 27881 27956
Math101 Calculus: 26098 27956
CS201 Introduction_to_Programming: 27013 27116 28304
```

At the end of the initialization phase, the created linked list should look like it is illustrated in Fig.3:

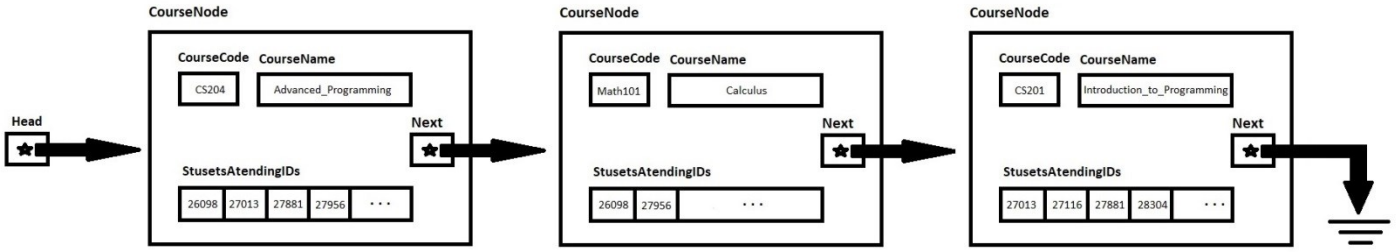


Fig. 3: A sample linked-list which is created after reading the coursesAndStudents.txt input file given in Fig.2.

4. Program Flow

After storing the data read from .txt file into your linked list, your program should constantly display a menu with four different options, as given below.

```
1. Add to List
2. Drop from List
3. Display List
4. Finish Add/Drop and Exit
```

According to the option value (1, 2, 3, or 4) that the user selects, the system performs the associated operation. Your program reads the option from the standard input (keyboard). You can assume that the user always chooses an existing option from the menu, i.e. when prompted so, the user gives an integer from 1 to 4 as an input. So you don't have to check for wrong input choices from the menu. All of the 4 menu choices are explained below.

4.1. Add to List

When the user selects this option, your program should ask for all students IDs with related courses that are supposed to be added to the list. Note that, the user can enter more than one course and for each course there might be more than one student to be added. After reading a course and its related IDs from the input, you have basically two options; either that course exists in the linked list or not. If currently there is no node with a course's name in the list, then a new node needs to be created and added to the linked list in a proper position to keep it sorted. If currently the given course exists in the link lists, then you need to check if the given student ID is already enrolled to the given course or not. If it is enrolled to the related course before, your program should print appropriate message to the user and no addition will be performed. If not, the student ID needs to be added to the related `AttendingStudentsIDs` vector in a proper position to keep it sorted in ascending order. For example, having in mind the linked list created by the program initialization above in Fig.3, after choosing option 1 from the menu, the behavior of the program (i.e. the updates on the

linked list and the displayed output messages) is shown below (user input are given in bold, the updates on link list of Fig.3 are shown in Fig.4):

```
Please select one of the choices:
1. Add to List
2. Drop from List
3. Display List
4. Finish Add/Drop and Exit
1
Give the student ID and the course names and course codes that he/she wants to add:
CS204 Advanced_Programming 27881 CS201 Introduction_to_Programming 27881 27116 CS301
Algorithms 27881
Student with id 27881 already is enrolled to CS204. No action taken.
Student with id 27881 is enrolled to CS201.
Student with id 27116 already is enrolled to CS201. No action taken.
CS301 does not exist in the list of Courses. It is added up.
Student with id 27881 is enrolled to CS301.
```

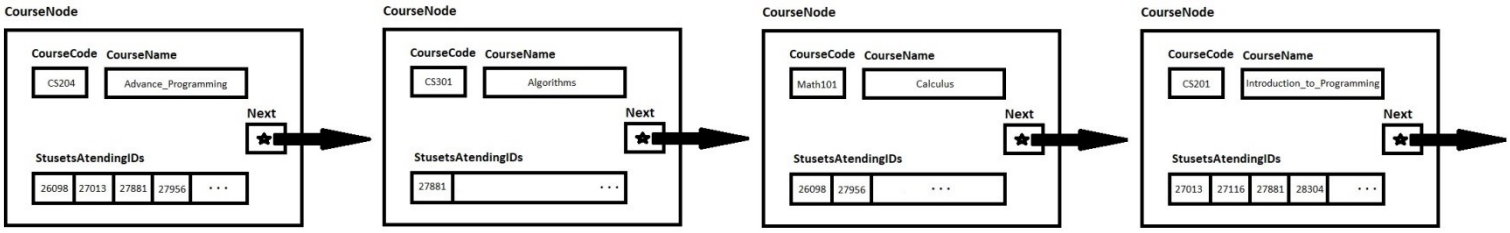


Fig. 4: Updating the linked list of Fig.3 according to the user input

4.2. Drop from List

When the user selects this option, your program should ask for all students IDs with related courses that are supposed to be dropped from the list. Note that the user can enter more than one course and for each course there might be more than one student to be dropped. The input is done in a way that, for each course, the course code is given first, proceeded by the course name and then by the student IDs. For each course you can assume that at least a single student ID is entered. After taking these inputs, first you need to check if the given student ID is already enrolled to the given course or not. If it is not enrolled in related course, your program should print appropriate message to the user and no deletion will be performed. If it is enrolled before, your program should delete the student ID from its related course's vector and inform the user by an appropriate message. After the dropping, your list and vectors inside it must remain sorted. Also, if an `AttendingStudentsIDs` vector is empty after a drop action, you should not delete that `courseNode` since at the end, all course nodes that have less than three attending students, will be closed. For example, having in mind the linked list above in Fig.4, after choosing option 2 from the menu, the behavior of the program (i.e. the updates on the linked list and the displayed output messages) is shown below (user input are given in bold, the updates on link list of Fig.4 are shown in Fig.5):

```
Please select one of the choices:
1. Add to List
2. Drop from List
3. Display List
4. Finish Add/Drop and Exit
2
```

```
Give the student ID and the course names and course codes that he/she wants to drop:
CS204 Advanced_Programming 27013 33333 CS201 Introduction_to_Programming 27013 CS301
Algorithms 27013 NS101 Natural_Sciences 25784
Student with id 27013 has dropped CS204.
Student with id 33333 is not enrolled to CS204, thus he can't drop that course.
Student with id 27013 has dropped CS201.
Student with id 27013 is not enrolled to CS301, thus he can't drop that course.
The NS101 course is not in the list, thus student with id 25784 can't be dropped
```

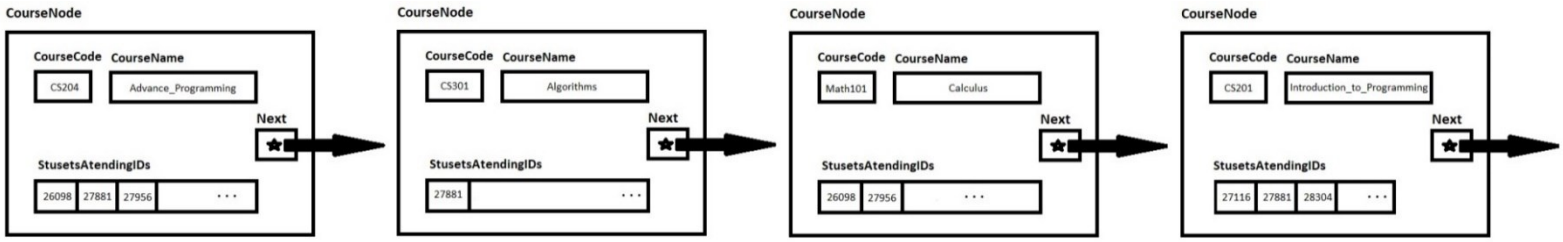


Fig. 5: Dropping some IDs from the list

4.3. Display List

When user selects this option, your program should display all the attending students IDs grouped by their courses code and name. This display operation must be sorted with respect the course name (as stored in the linked list). The IDs with the same course code must be displayed in the order that they are pushed onto the corresponding `AttendingStudentsIDs` vector. Also if there is a node with empty vector, its code and name will be shown like the others but without any students IDs after it. For example, if the user chooses option 3 while the linked list is as it's illustrated in Fig.5, the output will be as shown below:

```
Please select one of the choices:
1. Add to List
2. Drop from List
3. Display List
4. Finish Add/Drop and Exit
3
The current list of course and the students attending them:
CS204  Advanced_Programming: 26098 27881 27956
CS301 Algorithms: 27881
Math101 Calculus: 26098 27956
CS201 Introduction_to_Programming: 27116 27881 28304
```

4.4. Finish Add/Drop and Exit

When this option is selected, your program is terminated after showing the final list of the courses and the enrolled students at the end of the add/drop period. All courses with less than three enrolled students should be closed, thus you'll print a proper message indicating that such a particular course is being closed for the current semester. In order to make sure that you make no memory leak, your program must return all the dynamically allocated memory to the heap before the termination. Having in mind the linked list in Fig.5, if the user chooses option 4, the output (displayed messages) are shown below

```
Please select one of the choices:
1. Add to List
2. Drop from List
3. Display List
4. Finish Add/Drop and Exit
```

```
4
The add/drop period is finished. Printing the final list of courses and students
attending them.
NOTE: Courses with less than 3 students will be closed this semester.
CS204 Advanced_Programming: 26098 27881 27956
CS301 Algorithms 27881 -> This course will be closed
Math101 Calculus 26098 27956 -> This course will be closed
CS201 Introduction_to_Programming: 27116 27881 28304
```

5. Other remarks

You don't have to make any input check in any place of the program. You assume that the input file is correctly entered, its information is also correct. Furthermore, while choosing option 1 (`1. Add to List`) and 2 (`2. Drop from List`) you also assume that the user gives all of his inputs correctly, e.g. all of his course related info is in this format: courseCode course_name1 id1 id2 … idn, he doesn't give a wrong code for a wrong course, etc. Furthermore, we don't check details like having an extra space or lines when you display your outputs. Our aim here is to make you familiar with simple operations over singly linked lists, such as adding a node, deleting a node, searching the linked list, displaying (printing) the linked list, etc. Besides, by making you to have the linked list nodes being sorted by the course names, and in each `courseNode` having the `AttendingStudentsIDs` vector sorted by the student IDs all the time during the run-time, we want you to develop a feeling on how maintaining sorted data works on linked lists and how it differs in vectors. Besides, since you know that the linked list and each `AttendingStudentsIDs` vector are sorted, we expect you to utilize this fact while doing search operations on the linked list and on the `AttendingStudentsIDs` vector (hint: you can use binary search for sorted vectors. What about linked lists???).

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. 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.

VERY IMPORTANT!

Your programs will be compiled, executed and evaluated automatically; therefore you should definitely follow the rules for prompts, inputs and outputs. See **Sample Run** section for an examples.

- **Order of inputs and outputs** must be in the abovementioned format.

Following these rules is crucial for grading, otherwise our software will not be able to process your outputs and you will lose some grades in the best scenario.

IMPORTANT!

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

Another Sample Run:

//Assume that the input file is the same as file in previous example above.

```
Please enter file name: coursesAndStudents.txt
Successfully opened file CoursesAndStudents.txt
The linked list is created.
The initial list is:
Advance_Programming CS204: 26098 27013 27881 27956
Calculus Math101: 26098 27956
Introduction_to_Programming CS201: 27013 27116 28304

Please select one of the choices:
1. Add to List
2. Drop from List
3. Display List
4. Finish Add/Drop and Exit
1
Give the student ID and the course names and course codes that he/she wants to add:
CS204 Advanced_Programming 28304 27881 27281 NS102 Science_of_Nature 27881 26370 27116
27956 CS201 Inroduction_to_Programming 27881 27013
Student with id 28304 is enrolled to CS204.
Student with id 27881 already is enrolled to CS204. No action taken.
Student with id 27281 is enrolled to CS204.
NS102 does not exist in the list of Courses. It is added up.
Student with id 27881 is enrolled to NS102.
Student with id 26370 is enrolled to NS102.
Student with id 27116 is enrolled to NS102.
Student with id 27956 is enrolled to NS102.
Student with id 27881 is enrolled to CS201.
Student with id 27013 already is enrolled to CS201. No action taken.


Please select one of the choices:
1. Add to List
2. Drop from List
3. Display List
4. Finish Add/Drop and Exit
3
The current list of course and the students attending them:
CS204 Advanced_Programming: 26098 27013 27281 27881   27956 28304
Math101 Calculus: 26098 27956
CS201 Introduction_to_Programming: 27013 27116 27881 28304
NS102 Science_of_Nature: 26370 27116 27881 27956
```

```
Please select one of the choices:
1. Add to List
2. Drop from List
3. Display List
4. Finish Add/Drop and Exit
2
Give the student ID and the course names and course codes that he/she wants to drop:
Math101 Calculus 26098 27013 27956
Student with id 26098 has dropped Math101
Student with id 27013 is not enrolled to Math101, thus he can't drop that course.
Student with id 27956 has dropped Math101.

Please select one of the choices:
1. Add to List
2. Drop from List
3. Display List
4. Finish Add/Drop and Exit
3
The current list of course and the students attending them:
CS204 Advanced_Programming: 26098 27013 27281 27881 27956 28304
Math101 Calculus:
CS201 Introduction_to_Programming: 27013 27116 27881 28304
NS102 Science_of_Nature: 26370 27116 27881 27956

Please select one of the choices:
1. Add to List
2. Drop from List
3. Display List
4. Finish Add/Drop and Exit
4
The add/drop period is finished. Printing the final list of courses and students
attending them.
NOTE: Courses with less than 3 students will be closed this semester.
CS204 Advanced_Programming: 26098 27013 27281 27881 27956 28304
Math101 Calculus  -> This course will be closed
CS201 Introduction_to_Programming: 27013 27116 27881 28304
NS102 Science_of_Nature: 26370 27116 27881 27956
```

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% 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)

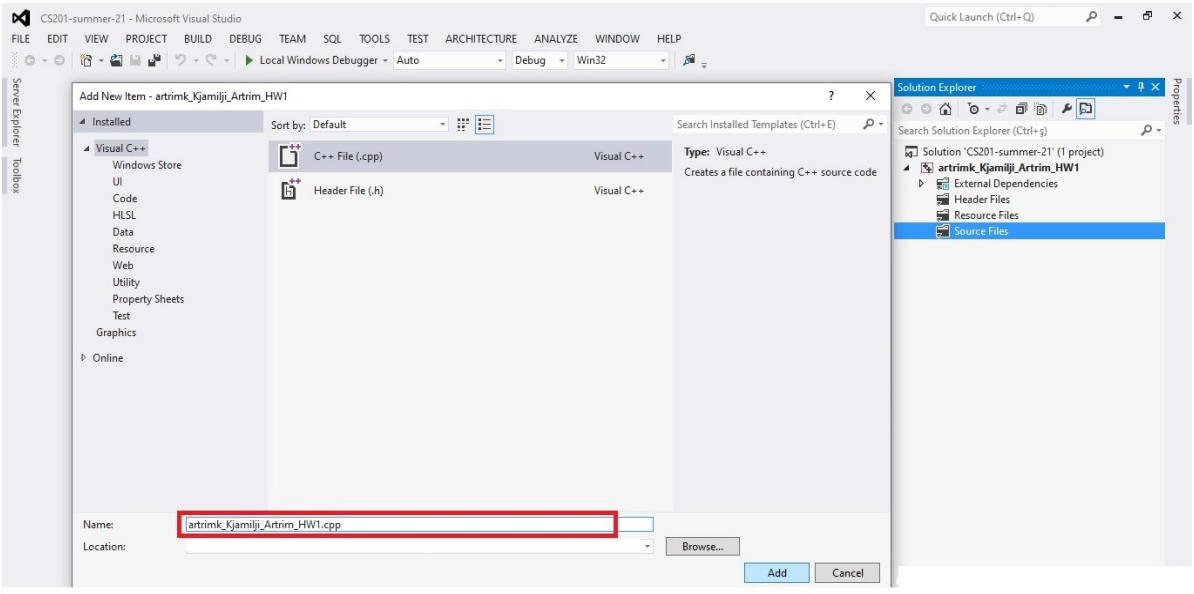
Submissions guidelines are below. Students are expected to strictly follow these guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the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

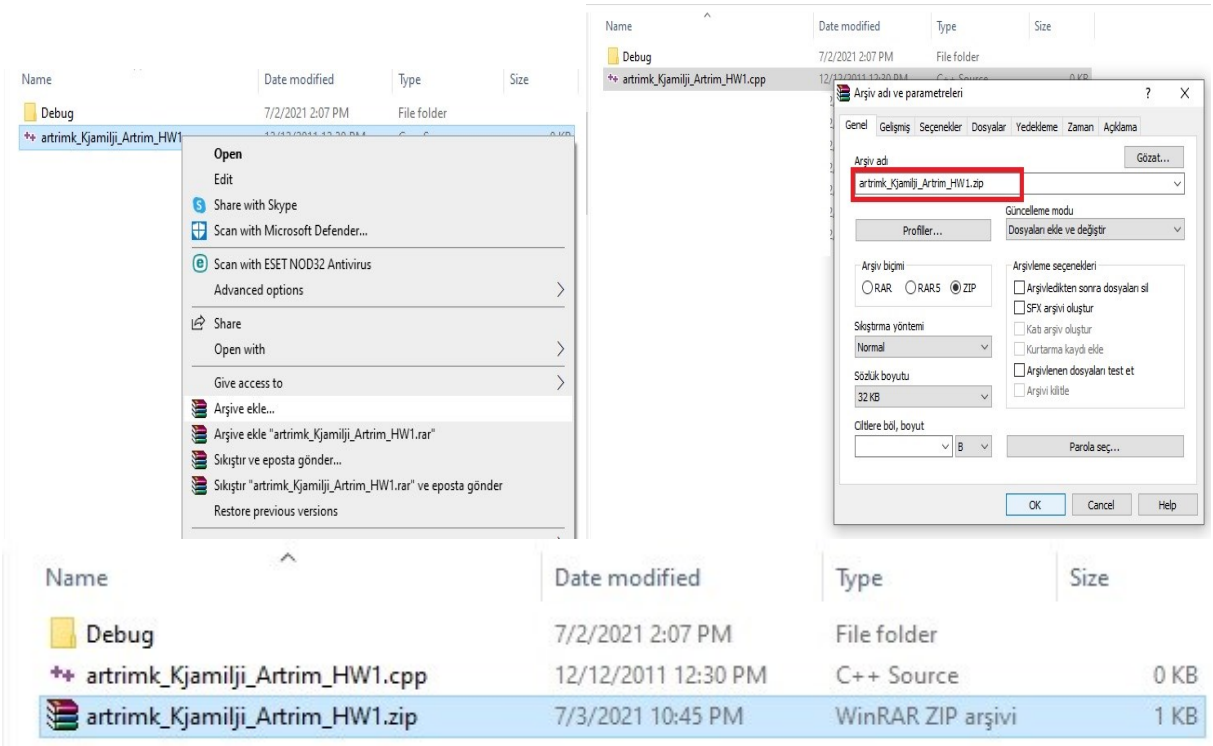
- ❑ Your SUCourse+ user name is actually your SUNet user name which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse+ user name is artrimk, your name is Artrim, your last name is Kjamilji, and you are uploading 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