CS201 – Spring 2015-2016 - Sabancı University

Homework #5: Boulder Pushing Puzzle

Due April 16, Saturday, 22:00 (Sharp Deadline)

Motivation

The aim of this homework is to practice on member functions and classes using the already known `Robot` class. Of course, you will need to use the topics that we have seen before such as if statements, functions and loops as required.

Description

In this homework, you will write a game program, implementing an interactive puzzle game using the `Robot` class.

`Robot` class: In the zip package that you obtained this document, you can find the modified versions of the `Robot` class files (Robots_Modified.cpp, Robots_Modified.h, MiniFW_Modified.cpp and MiniFW_Modified.h). You have to use these files in your project and program. Besides writing the program for the game in a separate cpp file, you will also make some further modifications to the `Robot` class, so you are going to update Robots_Modified.cpp and Robots_Modified.h files. These modifications consist of adding some more member functions to the `Robot` class as will be explained later in this document. The game will be implemented in another cpp file that you will write from scratch, not in `Robot` class files. `Robot` class files are to be updated only to add new member functions.

Game: In the game that you will develop, there is a player robot that is manually controlled by the user of your program. There are also 3 additional robots present that will be called *boulders* from now on. These boulders can be pushed by the player, that is, whenever a player robot and a boulder robot collide, the boulder will roll (move in one direction) until it hits a wall. The world is an 11-by-11 area that is filled with walls, and the aim of the player is to bring any of the boulders to the (10, 10) cell (northeast corner of the 11-by-11 area), within a movement restriction. The movement restriction is defined by the number of things that the player robot currently has in its bag, and the player can pick up things inside the area in order to increase this restriction. Please see the following Game Rules section for more details.

Please read the following explanations very carefully, because they are crucial for developing your program and modifying the header and source files of the robot class.

IMPORTANT!

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

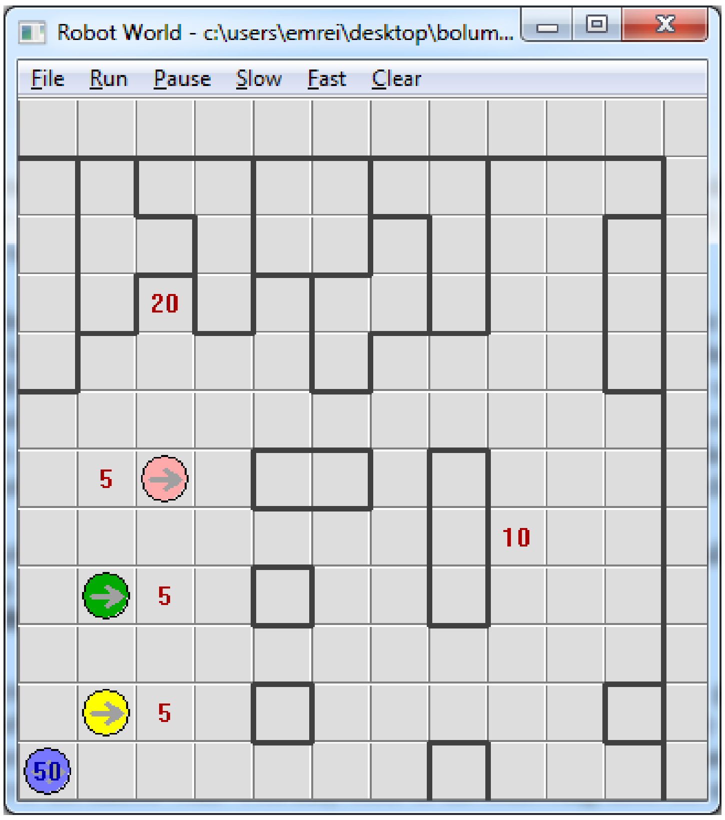
Game Rules and Details

Environment

Your program will use an environment that consists of a rectangular area and several walls. The corner points of the environment are (0, 0), (0, 10), (10, 0) and (10, 10). The boundaries of this environment are all walls (south and west are world boundaries; north and east are manually set walls).

Both walls and the things will be placed by the *.rw file you will load for your program. As the developer you are **not** responsible for putting any of them.

The player robot will start in the (0, 0) cell. For the boulder robots, your program will ask the player to enter X and Y coordinates for all 3 boulders in order to create them. **You may assume that none of the robots will be created on top of each other, in the starting cell of the player robot or in the exit cell. You don't need to implement any checks regarding their coordinates while creating them.**



The Player Robot

The player robot is actually an ordinary robot (an object from **`Robot`** class). As shown in the figure above, the player robot should be created at position (0, 0) facing **east**. Initially its bag content must be set to **50** and its color must be set to **blue**.

The user will control the player robot using ***up***, ***down***, ***right***, and ***left*** arrow keys on the keyboard. The control for the arrow keys can be done very easily using **`IsPressed`** function that you learned in recitations. Whenever one of the arrow keys is pressed, the player robot will move one step in the direction according to the pressed key. For example, if the user presses ↑ key, then the player robot will move one step (cell) towards north; if the user presses ← key, then the player robot will move one step (cell) towards west. **For each movement the player robot makes, the thing count should decreased by 1.**
Whenever the player robot moves into a cell that contains thing(s), it has to pick all the things in that cell after the movement. The things will be added to the player robot's bag, which will increase the number of movements your player can make in your game logic. **Only the player robot can pick up things.**

If there is a wall blocking player robot's movement, the player robot should **not** move (and hence die) and its thing count should not be decreased.

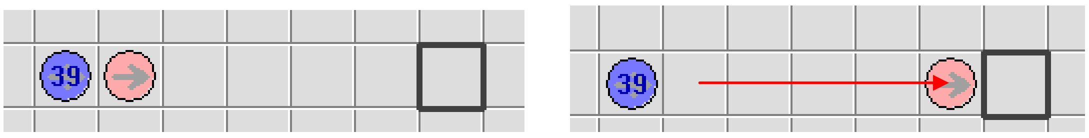
If there's a boulder robot blocking player robot's movement, again it should **not** move and its thing count should not be decreased, **but the boulder robot should start rolling.** This will be explained in the Boulder Robot section.

Please be careful that, in this homework, none of the robots are going to die (get stalled). You should always check that the player robot is controllable.

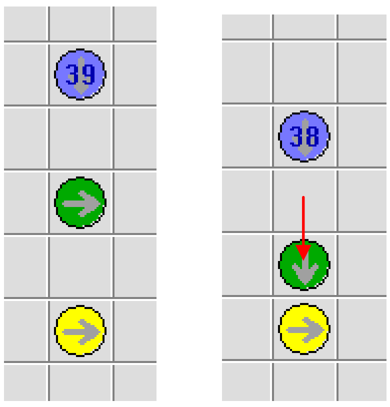
The Boulder Robot(s)

You will create 3 robots that will act as boulders for your game. The initial starting positions of these robots should be prompted to user to enter. An example prompt can be as following: "*Please enter the X coordinate for boulder Z*", in which Z dictates the current boulder that is going to be created. After both X and Y variables are entered for a robot, it should be created facing east. Colors for the boulders should be set as **Yellow**, **Green** and **Pink** in order.

If the player robot collides with a boulder robot, the boulder robot should start rolling till it encounters an obstacle (another robot, wall, or world border). **A boulder will not roll if the collision comes from another boulder robot.** Please look at the following screenshots from 2 examples:



Example 1: The Pink boulder robot is pushed from its left side by the player robot and it started rolling east. Please notice that the Pink boulder robot stops rolling when faced by a wall boundary. Please also notice the bag count of the player robot has not been changed after this roll.



Example 2: The Green boulder robot is pushed from its upper side and started rolling south. Please notice that Green boulder has changed its direction – this should happen before it starts rolling. Please also note that the Green boulder stops rolling when it faces the Yellow robot and Yellow boulder does not roll.

Please be careful that, in this homework, none of the robots are going to die (get stalled). This is mostly important when 2 boulders collide. You should make sure that both boulders are alive.

Reset Key

Since this is a puzzle game, you'll probably get stuck while trying to solve the game. In order not to re-open the game each time an unfavorable movement is made, you are going to implement a Reset key to reset all the robots to their starting positions. You should also set the thing count for the player robot back to 50.

Please note that you are not going to recreate the things provided in the initial load of your world. (To do so, you can use File > Reload button in your Robot program.)

You will use the **Home** key as the reset key. You should use the provided *Teleport* function in order to change the positions of your robots.

End Condition

There's both a win and a lose condition for this game:

- The game will be won if a boulder robot reaches the (10, 10) cell with the player robot still has things in its bag. Please be careful that the player robot in the exit cell does not count as a win condition. When the winning conditions occur, the program will end with prompting an output message displayed as below, with Z being the remaining things in the player robot's bag.

  Congratulations, solved the puzzle with Z energy!

- The game will be lost if the player robot does not have any things remaining in its bag. When the losing condition occurs, the program will end with prompting an output message displayed as below, with X and Y being the coordinates the player robot currently at.

  You couldn't reach to exit in time and blacked out in X, Y

Use of Functions and Classes, etc. (READ THIS. VERY VERY IMPORTANT)

For this homework, you have to add some new member functions to the Robot class. The number of additional functions can be different up to your design, but you have to add at least the following 8 member functions:

1. **`GetXCoordinate`**: A member function that returns the X coordinate of the robot object.
2. **`GetYCoordinate`**: A member function that returns the Y coordinate of the robot object.
3. **`TurnFace`**: A member function that takes a Direction type (this is an enumerated type defined in robot header file) parameter and turns the robot object towards this direction.
4. **`GetBagCount`**: A member function that returns the number of things in the robot's bag. This function will be useful to check for end conditions.
5. **`SetBagCount`**: A member function that will set the number of things in the robot's bag. This function will be useful for reset condition and player robot movement.
6. **`PickAllThings`**: A member function that will pick all the things in the current cell.
7. **`Resurrect:`** A member function that will revive (make alive) a robot if it becomes stalled (die).
8. **`IsAlive:`** A member function that should check whether the robot is alive or stalled (dead).

Needless to say, you have to utilize these member functions in your program, as needed. We have provided you an additional member function named **`Teleport`**, which should be used in your reset implementation.
SUCourseUserName_YourLastname_YourName_HWnumber.cpp, this file will include the game loop, robot creation and any other user defined functions that will be needed to implement the game.

In your program, please do not employ variables in order to keep track of a characteristic (private data member) of a robot object. This is against the philosophy behind OO (Object Oriented) programming. Whenever you need to access or change these private data members, use member functions. For example, do not employ a variable in the program to keep track of the position of a robot. Use member functions for this purpose.

In the member function implementations that change the visual characteristic of a robot, you need to use the command: **`theRobotWindow->Redraw(this);`** at the end of the member function implementation (do not use this anywhere else in the program).

If you need some other member functions, you are free to design and implement. However, do not forget the OO programming philosophy that proposes to have member functions for general class use. <u>If a member function is too specific for a particular application, it should not be written as a member function</u>, it might be a free function in the program. Again according to OO programming principles, <u>member functions should do a single job</u>, rather than multiple jobs. Please remember this also while writing your member functions. All of these OO programming rules will be considered in the grading process.

Moreover, your program must be modular and you should avoid code duplication. Thus you have to show your ability to use functions in an appropriate way. This will affect your grade. In general, if your main function or any user-defined function is too long and if you do everything in main or in another user-defined function, you grade may be lowered.

AND PLEASE DO NOT WRITE EVERYTHING IN MAIN AND THEN TRY TO SPLIT THE TASK INTO SOME FUNCTIONS JUST TO HAVE SOME FUNCTIONS OTHER THAN MAIN. THIS IS TOTALLY AGAINST THE IDEA OF FUNCTIONAL DESIGN AND NOTHING BUT A TRICK TO GET SOME POINTS. INSTEAD PLEASE DESIGN YOUR PROGRAM BY CONSIDERING NECESSARY FUNCTIONS <u>AT THE BEGINNING</u>.

Try to use parametric and non-void functions <u>wherever appropriate</u>. Do NOT use any global variables (variables defined outside the functions) to avoid parameter use.

In this homework (and in the coming ones) you are <u>not allowed to</u> use instructions such as “exit” and “goto”. These cause difficulty to control the flow of your programs. Thus we do not approve using them. You are also <u>not encouraged</u> to use “break” and “continue”. The use of “break” and “continue” prevent you from forming good readable loop conditions and hence prevent you from learning how to form good loops. Think cleverly in order not to use any of these instructions. If you don't know these commands, <u>do not even try to learn them</u> (we explained “break” in class).

Other Important Details (READ THIS. VERY IMPORTANT)

In the zip package, we provided you the necessary class files. Robots_Modified.h and Robots_Modified.cpp are original files for the Robot class. You will add the member functions mentioned before, and you will do the necessary modifications in the Robots_Modified.h, Robots_Modified.cpp files. In Robots_Modified.h file, you have to add the prototypes of the new member functions. Please add these prototypes to the end of the "public" section of the class definition. In the Robots_Modified.cpp file, you are going to write the implementations of the new member functions. You have to write these implementations between the following comment lines that are already provided in the file:

```
/*************** START: CS201 students' hw5 member functions *******************/


/*************** END: CS201 students' hw5 member functions *******************/
```

MiniFW_Modified.h and MiniFW_Modified.cpp files do not need to be updated.

In order to check if an arrow key is pressed or not, you can use the function called **`IsPressed`**. This function takes a parameter, which is the name of the arrow key, and returns true if currently that key is pressed. The names of arrow keys are **`keyUpArrow`**, **`keyDownArrow`**, **`keyLeftArrow`**, and **`keyRightArrow`**. For the reset key, you may use the name **`keyHome`** as well. This function is defined in minifw_modified.h, so you have to include this file at the beginning of your program. An example use of **`IsPressed`** will be explained in recitations.

New Member Function Provided

We provide a new member function for the Robot class. This function is implemented in the modified class files. There will be an example of using this new member functions in recitations.

```
void Robot::Teleport(int x, int y, Direction d);
```

This function beams up the object robot to the cell with coordinates (*x*, *y*), if this cell is not occupied by another robot. The robot is directed towards direction *d* at its new location. If the new cell is occupied by another robot, then both robots are killed automatically.

Demo Application

Due to interactive feature of this homework, we are not able to provide sample outputs, but we provide an executable file that can be run to understand how your program should work.

We have already written the program for this homework and the corresponding executable file (demohw5.exe) is given to you within the same zip package as this homework document. Before pressing "run" button on the menu, either you have to open a sample robot world (files with .rw extension), or you have to create an environment specified in this homework.

Note: Please run the sample world files with the following inputs for good puzzles ☺

Filename	X, Y for Boulder1	X, Y for Boulder2	X, Y for Boulder3
puzzle1.rw	(1, 1)	(1, 3)	(2, 5)
puzzle2.rw	(2, 1)	(2, 2)	(2, 3)

No abrupt program termination please!

You may want to stop the execution of the program at a specific place in the program. Although there are ways of doing this in C++, it is not a good programming practice to abruptly stop the execution in the middle of the program. Therefore, your program flow should continue until the end of the main function and finish there.

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) of CS201. Office hours of TAs are at the class website. Recitations will partially be dedicated to clarify the issues related to homework, so it is to your benefit to attend recitations.

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 will 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://myweb.sabanciuniv.edu/gulsend/su_current_courses/cs-201-spring-2008/plagiarism/

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 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.

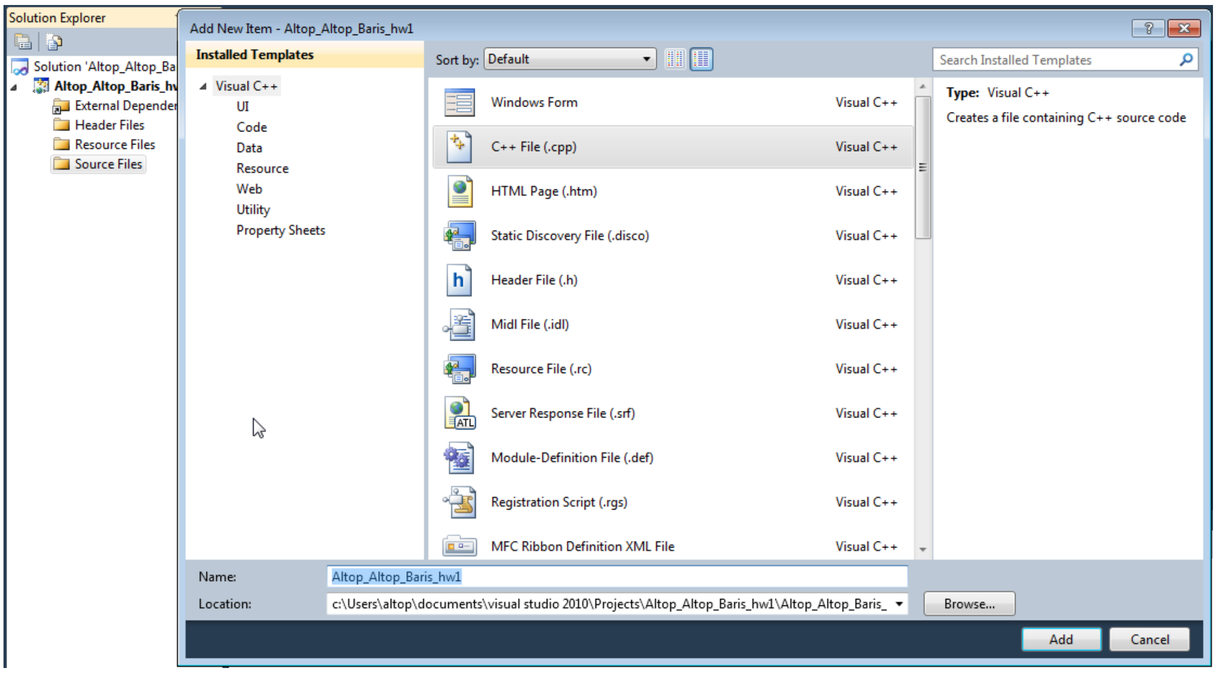
What and where to submit (IMPORTANT)

Submissions guidelines are below. Most parts of the grading process are automatic. 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**"


Solution Explorer
Solution 'Altop_Altop_Ba
Altop_Altop_Baris_hv
External Depender
Header Files
Resource Files
Source Files
Add New Item - Altop_Altop_Baris_hw1
Installed Templates
Visual C++
UI
Code
Data
Resource
Web
Utility
Property Sheets
Sort by: Default
Search Installed Templates
Windows Form Visual C++
C++ File (.cpp) Visual C++
HTML Page (.htm) Visual C++
Static Discovery File (.disco) Visual C++
Header File (.h) Visual C++
Midl File (.idl) Visual C++
Resource File (.rc) Visual C++
Server Response File (.srf) Visual C++
Module-Definition File (.def) Visual C++
Registration Script (.rgs) Visual C++
MFC Ribbon Definition XML File Visual C++
Browse...
Add
Cancel

Type: Visual C++
Creates a file containing C++ source code

Name: Altop_Altop_Baris_hw1
Location: c:\Users\altop\documents\visual studio 2010\Projects\Altop_Altop_Baris_hw1\Altop_Altop_Baris_

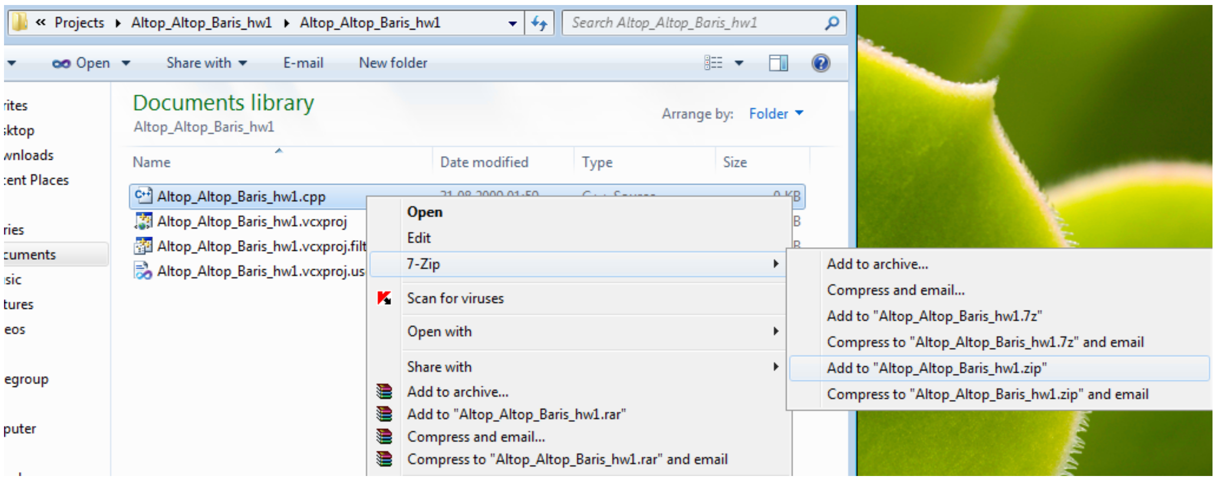
- ❑ Your SUCourse user name is actually your SUNet user name which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the file name must be:

Cago_Ozbugsizkodyazaroglu_Caglayan_hw5.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 file 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.


« Projects ▸ Altop_Altop_Baris_hw1 ▸ Altop_Altop_Baris_hw1
Search Altop_Altop_Baris_hw1
Open
Share with
E-mail
New folder
Documents library
Altop_Altop_Baris_hw1
Arrange by: Folder
Name
Date modified
Type
Size
Altop_Altop_Baris_hw1.cpp
Altop_Altop_Baris_hw1.vcxproj
Altop_Altop_Baris_hw1.vcxproj.filt
Altop_Altop_Baris_hw1.vcxproj.us
Open
Edit
7-Zip
Scan for viruses
Open with
Share with
Add to archive...
Add to "Altop_Altop_Baris_hw1.rar"
Compress and email...
Compress to "Altop_Altop_Baris_hw1.rar" and email
Add to archive...
Compress and email...
Add to "Altop_Altop_Baris_hw1.7z"
Compress to "Altop_Altop_Baris_hw1.7z" and email
Add to "Altop_Altop_Baris_hw1.zip"
Compress to "Altop_Altop_Baris_hw1.zip" and email

❑ Check that your compressed file opens up correctly and it contains your **cpp** file. You will receive no credits if your compressed zip file does not expand or it does not contain the correct file.

❑ 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_hw5.zip is a valid name, but hw4_hoz_HasanOz.zip, HasanOzHoz.zip are NOT valid names.

<u>Submission:</u>

❑ Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).

1) Click on "Assignments" at CS201 SUCourse (not the CS201 web site).
2) Click Homework 5 in the assignments list.
3) Click on "Add Attachments" button.
4) Click on "Browse" button and select the zip file that you generated.
5) Now, you have to see your zip file in the "Items to attach" list.
6) Click on "Continue" button.
7) Click on "Submit" button. We cannot see your homework if you do not perform this step even if you upload your file.

<u>Resubmission:</u>

❑ After submission, you will be able to take your homework back and resubmit. In order to resubmit, follow the following steps.

1) Click on "Assignments" at CS201 SUCourse.

2) Click Homework 5 in the assignments list.
3) Click on "Re-submit" button.
4) Click on "Add/remove Attachments" button
5) Remove the existing zip file by clicking on "remove" link. This step is very important. If you do not delete the old zip file, we receive both files and the old one may be graded.
6) Click on "Browse" button and select the new zip file that you want to resubmit.
7) Now, you have to see your new zip file in the "Items to attach" list.
8) Click on "Continue" button.
9) Click on "Submit" button. We cannot see your homework if you do not perform this step even if you upload your file.

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!

Emre İnal, Ece Egemen and Gülşen Demiröz