Sabancı University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Spring 2021

Homework 5 – Operator overloading for virtual wallet application

Due: 21/04/2021, Wednesday, 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 virtual *wallet* and overload some operators for it. The details of these operators will be given below. Our focus in this homework is on the class design and implementation. The usage of this class in the main function will be given to you.

Virtual Wallet Class Design

You will use a `struct` called *money* to model a single monetary item. In this struct, there will be a data member to store *name of the currency* (of type `string`, in case sensitive manner) and another data member to store the *amount* (of type `double`). If you want, you can write constructor(s) for this struct, but not needed for the main program. However, you will not write any member functions for this struct (well, we have not learned how to write member functions for structs in anyway).

The virtual wallet will be implemented as a class. In the *wallet* class, there should be two private data members: (i) the currencies and their amounts in the wallet as a dynamic array of *money* struct (i.e. *money* pointer); (ii) the size of this dynamic array, i.e. the number of distinct currencies stored in the wallet object.

The class should be designed and implemented in such a way that (i) a currency in the array must always have a positive amount, and (ii) a particular currency should exist in at most one item in the array. These design issues will be made more clear in the operator explanations below.

Your *wallet* class implementation must include the following basic class functions:

- Default constructor: creates a *wallet* object with zero size. This constructor does not allocate memory for the elements of the wallet since there are no elements.
- Deep copy constructor: takes a *wallet* object as const-reference parameter and creates a new *wallet* object as the deep copy of the parameter.
- Destructor: By definition, destructor deletes all dynamically allocated memory and returns them back to the heap.

In this homework, you will overload several operators for the *wallet* class. These operators and the details of overloading are given below. You can overload operators as member or free functions. However, in order to test your competence in both mechanisms, we request you to have at least three member and three free functions for these operators (the remaining three is up to you).

<< This operator will be overloaded such that it will put a *wallet* on an output stream (`ostream`). See the sample runs for the required output format. The order of display of the currencies is **not** important. Since the left hand side is an output stream, this function must be a free function.

Note that + operator will be overloaded in two different ways as (i) addition of *money* to the *wallet* and (ii) addition of *wallet* to another *wallet*.

+ This operator will be overloaded such that it will add a *money* item to a *wallet* object. The *money* item is the **right hand side (rhs)** operand of the + operator, and the *wallet* is on the **left hand side (lhs)**. Since this is an operator that needs to return a value (not a reference), you have to work on a brand new wallet object in the implementation of this function. The content of the wallet to be returned must contain the currencies of the lhs wallet plus the money of the rhs. First, you should check if the currency of money on the rhs is already in the lhs wallet or not. If same type of currency already exists in the lhs wallet object, then your program should add the amounts of both sides for that currency in the wallet object to be returned. Otherwise, returned wallet object should include the rhs money's amount for that currency (and other currencies of the lhs of course).

+ This operator will be overloaded such that it calculates and returns the sum of two *wallet* operands. This operator also returns a brand new object without changing the contents of the operands. If a particular currency exists in both operands, of course, you will include it only once in the returned object, but its amount should be equal to the sum of both. If a particular currency exists in one operand only, you will include it directly in the object to be returned.

Here remark that we do not have any assumption about the size of a wallet. While implementing the + operators, the resulting wallet that you are going to return might have a size greater than the size of the operand(s). In such cases, (i) do not allocate memory more than needed; i.e. the amount of currencies in the memory allocated for the returning wallet must be the size of it, and (ii) please be careful and do not make memory leak.

- This operator simulates spending money from a wallet object. Right hand side is a money item, while left hand side is a wallet object. Actually, you will not change the content of the left hand size wallet object per se, but will return another wallet object which contains the currencies of the left hand side reduced by the right hand side. This operator will be overloaded such that it does not allow spending more money (of the particular currency of the right hand side) than what exists in the left hand side wallet. If there is enough money, then spending is done and the returned wallet object is updated accordingly. Otherwise, returned wallet remains as is without any update (i.e. will contain the currencies of the left hand size wallet). In other words, no negative amounts are allowed in the wallet array. If the spending amount is the same as the amount in the wallet, then you should remove that currency from the array of wallet to be returned (i.e. do not store an item with zero amount in the array); this might require to change the array size and allocations as discussed for + operators above (depending on your implementation).

= This operator will be overloaded such that it will assign the *wallet* object on the right hand-side of the operator to the *wallet* object on the left hand side. 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).

+= This operator will be overloaded such that it will add two *wallet* objects and assign the resulting *wallet* object to the *wallet* object on the left hand side of the += operator. Addition logic of two wallets had been explained above. This function must be implemented in a way to allow cascaded assignments.

== This operator will be overloaded such that it will check two *wallet* objects for equality. This operator returns true when they are equal; returns false otherwise. Two wallets are said to be equal if they contain the same currencies with the same amounts (caution: order of the currencies in the arrays of both wallets could be different).

>= This operator takes a wallet object on the left hand side and a money item on the right hand side. It should return false if the currency of the right hand side does not exist in the left hand side wallet. If exists, then you should compare the amount of the corresponding currency of the left hand side wallet with the amount of the right hand side money. If the left hand side's amount is greater than or equal, then the function returns true; otherwise, returns false.

<= This operator takes a money item on the left hand side and a wallet object on the right hand side. It should return false if the currency of the left hand side does not exist in the right hand side wallet. If exists, then you should compare the amount of left hand side money with the amount of the corresponding currency of the right hand side wallet. If the left hand side's amount is less than or equal, then the function returns true; otherwise, returns false. Since left hand side is not a wallet object, this function must be a free function. Hint: think of other operators to utilize in the implementation of this one.

In order the 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, accessors (getters) and mutators (setters), if needed. However, you are **not allowed to use friend functions and friend classes**.

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 or simply Google it!

You have to have separate header and implementation files for your class, in addition to the main.cpp. Make sure that the header file name that you submit and `#include` are the same; otherwise your program does not compile.

main.cpp

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 (of course, you can edit to add `#include`s, etc. to the beginning of the file). All class related functions, definitions and declarations (including free operator functions) must be done in class header and implementation files. Inputs 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. 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 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, and following the rules mentioned in this homework specification 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.

You are allowed to use sample codes shared with the class by the instructor and TA, but you are not allowed to use any codes that you might find somewhere else (any online, machine or human source). However, you cannot start with an existing .cpp or .h file directly and update it; you have 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 */
```

Sample Runs

Some sample runs are given below, but these are not comprehensive, therefore you have to consider **all possible cases** to get full mark. Especially try different wallet contents (other than the ones given in the sample runs) and extreme cases. In order to finish the input entry, we used `^Z`. This character can by typed as Ctrl-Z using keyboard. For Mac users, it is Ctrl-D.

Sample Run 1:

```
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
BTC 2
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
BTC 1
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
ETH 10.4
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
HBAR 200.4
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
ETH 3
```

```
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
BTC 0.2
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
HBAR 49.6
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
XRP 239.4
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
^Z

Current myWallet after input and self assignment:
BTC 3.2 - ETH 13.4 - HBAR 250 - XRP 239.4

Current myWallet2 after copy constructor:
BTC 3.2 - ETH 13.4 - HBAR 250 - XRP 239.4

What do you want to spend from myWallet?: XRP 239.4

Current myWallet after spending:
BTC 3.2 - ETH 13.4 - HBAR 250

Current myWallet2 after spending:
BTC 3.2 - ETH 13.4 - HBAR 250 - XRP 239.4

myWallet and myWallet2 are not the same

Empty myWallet3:

Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
HBAR 251
Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
DOGE 52.1
Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
^Z

Current myWallet3 after reading:
HBAR 251 - DOGE 52.1

myWallet and myWallet3 are not the same

myWallet does not have more than 250 HBAR
myWallet3 has more than 250 HBAR

Current myWallet3 after myWallet3 = myWallet3 + myWallet:
HBAR 501 - DOGE 52.1 - BTC 3.2 - ETH 13.4

Current myWallet4 after myWallet4 = myWallet + myWallet2 + myWallet3:
BTC 9.6 - ETH 40.2 - HBAR 1001 - XRP 239.4 - DOGE 52.1

After myWallet = myWallet2 = myWallet3 += myWallet2
Current myWallet:
HBAR 751 - DOGE 52.1 - BTC 6.4 - ETH 26.8 - XRP 239.4

Current myWallet2:
HBAR 751 - DOGE 52.1 - BTC 6.4 - ETH 26.8 - XRP 239.4

Current myWallet3:
HBAR 751 - DOGE 52.1 - BTC 6.4 - ETH 26.8 - XRP 239.4
```

Sample Run 2:

```
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
DOGE 223.1
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
^Z

Current myWallet after input and self assignment:
DOGE 223.1

Current myWallet2 after copy constructor:
DOGE 223.1

What do you want to spend from myWallet?: DOGE 223.1

Current myWallet after spending:


Current myWallet2 after spending:
DOGE 223.1

myWallet and myWallet2 are not the same

Empty myWallet3:

Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
BTC 22
Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
ETH 32.1
Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
^Z

Current myWallet3 after reading:
BTC 22 - ETH 32.1

myWallet and myWallet3 are not the same

myWallet does not have more than 250 HBAR
myWallet3 does not have more than 250 HBAR

Current myWallet3 after myWallet3 = myWallet3 + myWallet:
BTC 22 - ETH 32.1

Current myWallet4 after myWallet4 = myWallet + myWallet2 + myWallet3:
DOGE 223.1 - BTC 22 - ETH 32.1

After myWallet = myWallet2 = myWallet3 += myWallet2
Current myWallet:
BTC 22 - ETH 32.1 - DOGE 223.1

Current myWallet2:
BTC 22 - ETH 32.1 - DOGE 223.1

Current myWallet3:
BTC 22 - ETH 32.1 - DOGE 223.1
```

Sample Run 3:

```
Please enter the name of the currency and amount to put it to myWallet (to stop
enter ctrl+z on an empty line):
HBAR 270
Please enter the name of the currency and amount to put it to myWallet (to stop
enter ctrl+z on an empty line):
BTC 19.2
Please enter the name of the currency and amount to put it to myWallet (to stop
enter ctrl+z on an empty line):
XRP 2312
Please enter the name of the currency and amount to put it to myWallet (to stop
enter ctrl+z on an empty line):
^Z

Current myWallet after input and self assignment:
HBAR 270 - BTC 19.2 - XRP 2312

Current myWallet2 after copy constructor:
HBAR 270 - BTC 19.2 - XRP 2312

What do you want to spend from myWallet?: BTC 20

Current myWallet after spending:
HBAR 270 - BTC 19.2 - XRP 2312

Current myWallet2 after spending:
HBAR 270 - BTC 19.2 - XRP 2312

myWallet and myWallet2 are exactly the same

Empty myWallet3:

Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
BTC 10.8
Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
XRP 94
Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
^Z

Current myWallet3 after reading:
BTC 10.8 - XRP 94

myWallet and myWallet3 are not the same

myWallet has more than 250 HBAR
myWallet3 does not have more than 250 HBAR

Current myWallet3 after myWallet3 = myWallet3 + myWallet:
BTC 30 - XRP 2406 - HBAR 270

Current myWallet4 after myWallet4 = myWallet + myWallet2 + myWallet3:
HBAR 810 - BTC 68.4 - XRP 7030

After myWallet = myWallet2 = myWallet3 += myWallet2
Current myWallet:
BTC 49.2 - XRP 4718 - HBAR 540

Current myWallet2:
BTC 49.2 - XRP 4718 - HBAR 540

Current myWallet3:
BTC 49.2 - XRP 4718 - HBAR 540
```

Sample Run 4:

```
Please enter the name of the currency and amount to put it to myWallet (to stop
enter ctrl+z on an empty line):
HBAR 3273.23
Please enter the name of the currency and amount to put it to myWallet (to stop
enter ctrl+z on an empty line):
ETH 234
Please enter the name of the currency and amount to put it to myWallet (to stop
enter ctrl+z on an empty line):
BTC 22.22
Please enter the name of the currency and amount to put it to myWallet (to stop
enter ctrl+z on an empty line):
^Z

Current myWallet after input and self assignment:
HBAR 3273.23 - ETH 234 - BTC 22.22

Current myWallet2 after copy constructor:
HBAR 3273.23 - ETH 234 - BTC 22.22

What do you want to spend from myWallet?: HBAR 3000

Current myWallet after spending:
HBAR 273.23 - ETH 234 - BTC 22.22

Current myWallet2 after spending:
HBAR 3273.23 - ETH 234 - BTC 22.22

myWallet and myWallet2 are not the same

Empty myWallet3:

Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
HBAR 275
Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
^Z

Current myWallet3 after reading:
HBAR 275

myWallet and myWallet3 are not the same

myWallet has more than 250 HBAR
myWallet3 has more than 250 HBAR

Current myWallet3 after myWallet3 = myWallet3 + myWallet:
HBAR 548.23 - ETH 234 - BTC 22.22

Current myWallet4 after myWallet4 = myWallet + myWallet2 + myWallet3:
HBAR 4094.69 - ETH 702 - BTC 66.66

After myWallet = myWallet2 = myWallet3 += myWallet2
Current myWallet:
HBAR 3821.46 - ETH 468 - BTC 44.44

Current myWallet2:
HBAR 3821.46 - ETH 468 - BTC 44.44

Current myWallet3:
HBAR 3821.46 - ETH 468 - BTC 44.44
```

Sample Run 5:

```
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
USD 100
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
EUR 200
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
CHF 300
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
^Z

Current myWallet after input and self assignment:
USD 100 - EUR 200 - CHF 300

Current myWallet2 after copy constructor:
USD 100 - EUR 200 - CHF 300

What do you want to spend from myWallet?: USD 1001

Current myWallet after spending:
USD 100 - EUR 200 - CHF 300

Current myWallet2 after spending:
USD 100 - EUR 200 - CHF 300

myWallet and myWallet2 are exactly the same

Empty myWallet3:

Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
CHF 100
Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
USD 200
Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
EUR 300
Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
^Z

Current myWallet3 after reading:
CHF 100 - USD 200 - EUR 300

myWallet and myWallet3 are not the same

myWallet does not have more than 250 HBAR
myWallet3 does not have more than 250 HBAR

Current myWallet3 after myWallet3 = myWallet3 + myWallet:
CHF 400 - USD 300 - EUR 500

Current myWallet4 after myWallet4 = myWallet + myWallet2 + myWallet3:
USD 500 - EUR 900 - CHF 1000

After myWallet = myWallet2 = myWallet3 += myWallet2
Current myWallet:
CHF 700 - USD 400 - EUR 700

Current myWallet2:
```

```
CHF 700 - USD 400 - EUR 700

Current myWallet3:
CHF 700 - USD 400 - EUR 700
```

Sample Run 6:

```
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
USD 100
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
EUR 200
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
CHF 300
Please enter the name of the currency and amount to put it to myWallet (to stop enter
ctrl+z on an empty line):
^Z

Current myWallet after input and self assignment:
USD 100 - EUR 200 - CHF 300

Current myWallet2 after copy constructor:
USD 100 - EUR 200 - CHF 300

What do you want to spend from myWallet?: BTC 2

Current myWallet after spending:
USD 100 - EUR 200 - CHF 300

Current myWallet2 after spending:
USD 100 - EUR 200 - CHF 300

myWallet and myWallet2 are exactly the same

Empty myWallet3:

Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
CHF 300
Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
USD 100
Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
EUR 200
Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
^Z

Current myWallet3 after reading:
CHF 300 - USD 100 - EUR 200

myWallet and myWallet3 are exactly the same

myWallet does not have more than 250 HBAR
myWallet3 does not have more than 250 HBAR

Current myWallet3 after myWallet3 = myWallet3 + myWallet:
CHF 600 - USD 200 - EUR 400

Current myWallet4 after myWallet4 = myWallet + myWallet2 + myWallet3:
USD 400 - EUR 800 - CHF 1200
```

```
After myWallet = myWallet2 = myWallet3 += myWallet2
Current myWallet:
CHF 900 - USD 300 - EUR 600

Current myWallet2:
CHF 900 - USD 300 - EUR 600

Current myWallet3:
CHF 900 - USD 300 - EUR 600
```

Sample Run 7:

```
Please enter the name of the currency and amount to put it to myWallet (to stop
enter ctrl+z on an empty line):
^Z

Current myWallet after input and self assignment:


Current myWallet2 after copy constructor:


What do you want to spend from myWallet?: CS204 204

Current myWallet after spending:


Current myWallet2 after spending:


myWallet and myWallet2 are exactly the same

Empty myWallet3:

Please enter the name of the currency and amount to put it to myWallet3 (to stop
enter ctrl+z on an empty line):
^Z

Current myWallet3 after reading:


myWallet and myWallet3 are exactly the same

myWallet does not have more than 250 HBAR
myWallet3 does not have more than 250 HBAR

Current myWallet3 after myWallet3 = myWallet3 + myWallet:


Current myWallet4 after myWallet4 = myWallet + myWallet2 + myWallet3:


After myWallet = myWallet2 = myWallet3 += myWallet2
Current myWallet:


Current myWallet2:


Current myWallet3:
```

What and where to submit (PLEASE READ, IMPORTANT)

You should prepare (or at least test) your program using MS Visual Studio 2012 C++. We will use the standard C++ compiler and libraries of the abovementioned platform while testing your homework. It'd be a good idea to write your name and last name in the program (as a comment line of course).

Submissions guidelines are below. Some parts of the grading process might be 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.

Name your cpp file that contains your main program using the following convention:

"SUCourseUserName_YourLastname_YourName_HWnumber.cpp"

Your SUCourse user name is 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

In some homework assignments, you may need to have more than one .cpp or .h files to submit. In this case, use the same filename format but add informative phrases after the hw number (e.g. Cago_Ozbugsizkodyazaroglu_Caglayan_hw3_myclass.cpp or Cago_Ozbugsizkodyazaroglu_Caglayan_hw3_myclass.h). However, do not add any other character or phrase to the file names. Sometimes, you may want to use some user defined libraries (such as strutils of Tapestry); in such cases, you have to provide the necessary .cpp and .h files of them as well. If you use standard C++ libraries, you do not need to provide extra files for them.

Make sure that you #include correct header file names. If you rename your header file names just before submission and does not update `#include` statements accordingly, the file name you use in your program would not match with the real file name and your program does not compile. This causes getting zero in your homework.

These source files are the ones that you are going to submit as your homework. However, even if you have a single file to submit, you have to compress it using ZIP format. To do so, first create a folder that follows the abovementioned naming convention ("SUCourseUserName_YourLastname_YourName_HWnumber"). Then, copy your source file(s) there. And finally compress this folder using WINZIP or WINRAR programs (or another mechanism). 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 the files that belong to the latest version of your homework.

You will receive zero if your compressed zip file does not expand or it does not contain the correct files. The naming convention of the zip file is the same. 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

Hw5_hoz_HasanOz.zip, HasanOzHoz.zip

are **NOT** valid names.

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

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!

Albert Levi, Vedat Peran