Sabancı University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Fall 2023

Homework 3 – Stacks & Queues and SU services

Due: 10/11/2023, 23:55 (11:55 PM)

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 can discuss and talk about the homework. But you CANNOT describe your solution in detail to your friends. You have to write down the code on your own. Plagiarism will not be tolerated!

1. Introduction

This homework's aim is to make you familiar with the logic behind **queues** and **stacks** (and a bit of **simply linked lists of linked lists**, again) as well as **recursions** by practically using them in a real-case scenario (you will probably encounter similar cases as a future programmer).

Namely, SU offers a certain number of services to its academic stuff and students (such as data processing). The service requests are implemented as functions. You will keep the requests in a linked-list base **queue.** The requests of the students and the academic stuff (instructors) are kept separately and served in a First In First Out (FIFO) manner, thus we need **two separate queues** for both of them. Each service (function) request contains a set of instructions (commands) described in the next section. A certain service (function) can be a part of (called from) another function (e.g. print the output data after processed from the supercomputer). Besides that, SU bills users for their total service usage. Price of each service differs according to the commands it contains.

As it is common in modern day systems, for this reason we will need **a single, commonly shared memory stack** to put on and serve in First In Last Out (FILO) manner those subsequent service (function) calls. All of this is illustrated in Fig.1. Also, in order to process those consecutive function calls, you will be asked to implement **a recursive function** that calls an instance of itself at any time a new function call is issued. This will be more clarified in the 'main menu & program guide' section.

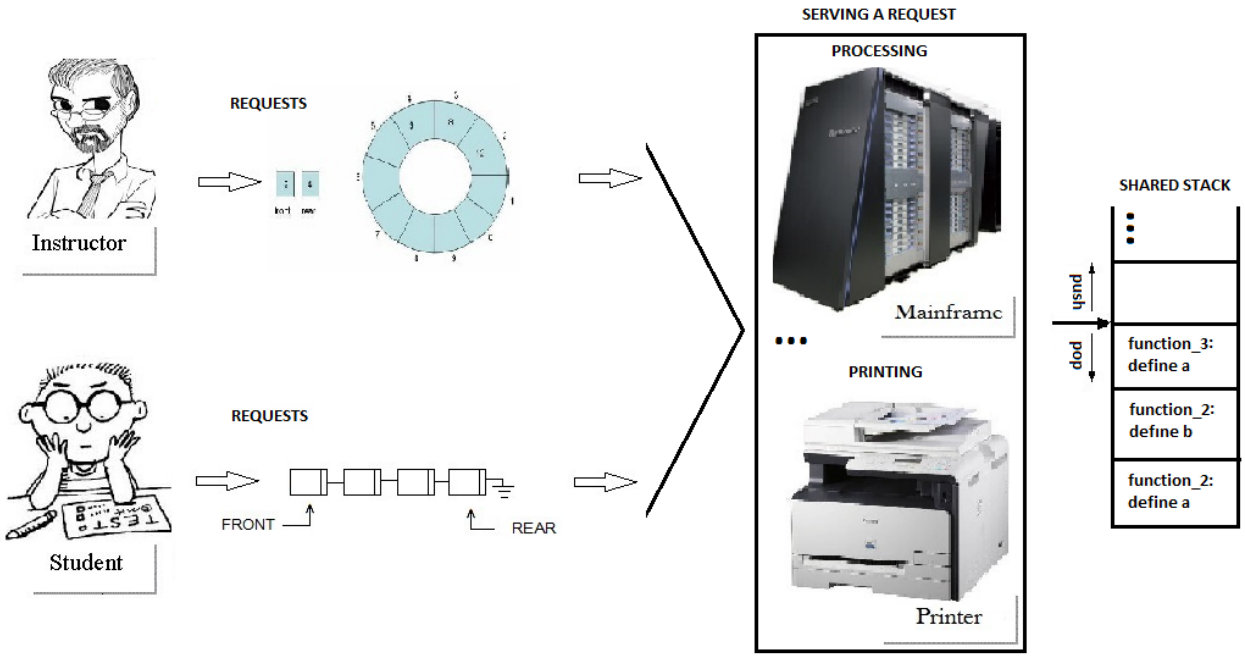

Fig.1. System overview

2. Service offers (input files)

For the time being, SU has organized its services that it offers in separate files (Fig. 2). Those will be your input files (we will use different functions while grading your submissions). Each service is implemented as a function that has a certain number of commands (instructions). Their aim and purpose will be described in the next section. In the first line function's name is given proceeding with a colon (':'). Afterwards in each new line you have a separate command that ends with semicolon(';'). The input format is always the same. You don't have to care about multiple empty spaces or new lines, and even letter cases. The data will be clean and won't contain any errors.

```
function_1:
print stack;
define x;
define y;
print stack;
call function_2;
…
…
…
define b;
print stack;
```

input1.txt

```
function_2:
print stack;
define a;
define b;
print stack;
call function_3;
…
…
…
define c;
print stack;
```

input2.txt

```
function_3:
print stack;
define a;
define x;
define z;
print stack;
define c;
print stack;
```

input3.txt

Fig.2. Input file samples

For all of the input files, your job will be to construct a simply linked list of the commands. Eventually, after reading all of your input files (in our case 3 input files), you should finish with a structure shown in Fig.3. Each service (function) appears as a node in the list that has its name in it, a pointer to the next service node and another point to the commands it is consisted of. As system engineers, we decided on this approach (dynamic lists) since the number of services that SU offers (and may offer in the future) is

not known in advance and that each service (function) has different numbers of commands. This way of dealing with things, not only will utilize our memory, but should also help us while processing the services when using a recursive function. We will come back to this after a while.

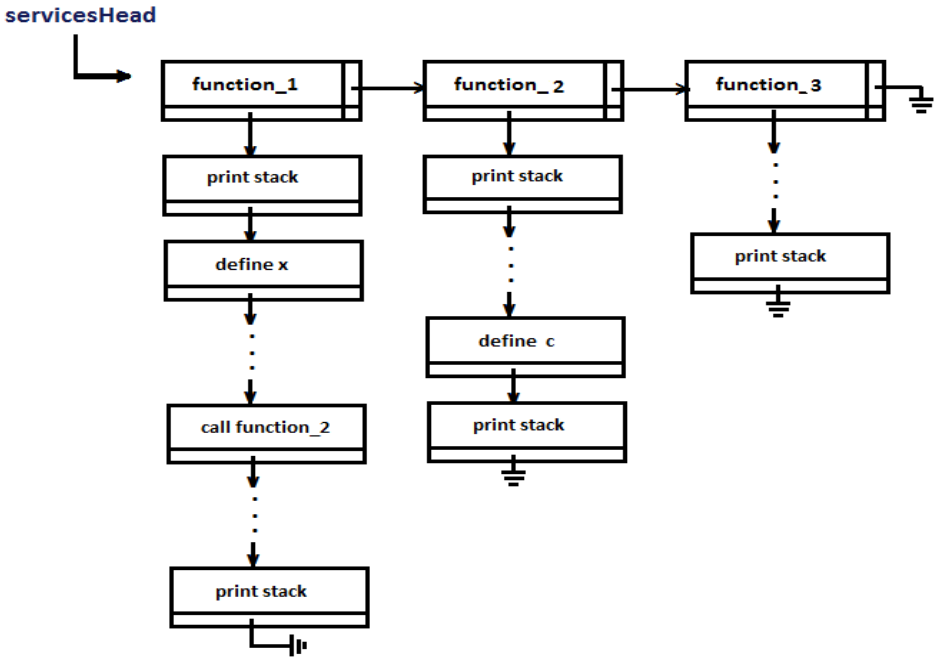

Fig.3. Linked list of Offered services (functions).

3. Logic flow of the system

Students and instructors put their corresponding requests of services in two separate queues. Since services are done in FIFO logic, you should keep in mind the order of the service requests (which request came first, e.g. the front and the rear of both of the queues). Knowing that the number of instructors is known in advance (assume it is an arbitrarily large number as 500), we encourage you to implement instructors queue with an array (similar to the circular array with wraparounds as shown in Fig.1). For simplicity, you can assume that every instructor can send one request at a time. Since the number of students is not known beforehand, we encourage you to implement the student request queue as a dynamic linked list (Fig.1). Of course, the final decision regarding the implementations is left to you.

System attains priority to instructors over students up to 10 instructor requests. After processing 10 instructor requests, system will process 1 student request and then continue processing remaining instructor requests by using the same strategy. For example, assume that there are N instruction requests and M student requests on their individual queues. In that case after k'th round of this strategy, there will be N-(10*k) requests in instructor queue and M-(1*k) requests in student queue.Of course, inside a queue (be it the student's or instructor's one), serving is done in FIFO manner (Fig. 1).

We have only three types of commands for the services (functions):

1. **define**,
2. **print stack**, and
3. **call.**

The "**define**" command simply defines a variable, whose name comes immediately after the define" command (e.g. define a, define x, define z). When this command is reached, we should push the variable to the program stack. This command costs 1 TRY.

The “**print stack**” command (illustrated below) prints the content of the program stack. When printing the stack content (also known as stack trace) we should also show the corresponding function that defined a certain variable. E.g. if variable *x* was defined by a “define x” command from function_1, on top of the stack we should put (and subsequently print): “function_1: define a”. During the ”print stack” command nothing should be put on top of (push-ed) or taken (pop-ped) from the shared stack (e.g. stack should not be changed). This command costs 2 TRY.

The “**call**” command should enable us to call an existing service (function) and temporarily stop the execution of the current function while proceeding with the execution of the newly called one. For this reason we will need a structure to be available to those services (functions) that will work in a First In Last Out (FILO) order, e.g. if a function calls a function that calls a function, firstly we will need to execute the lastly called function, then the one that called it and in the end the first function. When the called function is finished, we should be able to continue with the original function from the exact place we left, thus with the instruction immediately after the call command. As we said before, we will do this by using a commonly-shared stack for the system. Furthermore, when a function finishes, all of the stack data resulting (pushed) from it, should be deleted (removed, pop-ed) from the stack. Take caution not to delete any data from the predecessor function (the one that called the current, about to finish function). This command costs 5 TRY.

Since every service consist of a variety of commands, every service has varying prices. To calculate the price of a service, we must add up the prices of every command. One needs to keep track of the total bill of every user with a data structure like a user list. Also note that, (user name, user ID) must be unique.

4. Main menu & program guides

Fig. 4 gives to you the main menu piece of code. You should implement the corresponding functions. Fig. 5 shows the main menu view (output) that will occur often during the course of the program.

```
while (true){
   cout << endl;
    cout<<"*********************************************************************"<<endl
    <<"**************** 0 - EXIT PROGRAM                        *************"<<endl
    <<"**************** 1 - ADD AN INSTRUCTOR SERVICE REQUEST   *************"<<endl
    <<"**************** 2 - ADD A STUDENT SERVICE REQUEST       *************"<<endl
    <<"**************** 3 - SERVE (PROCESS) A REQUEST           *************"<<endl
    <<"**************** 4 - DISPLAY USER PAYMENTS               *************"<<endl
    <<"*********************************************************************"<<endl;
    cout << endl;
    int option;
    cout << "Pick an option from above (int number from 0 to 3): ";
    cin>>option;
    switch (option)
    {
        case 0:
            cout<<"PROGRAM EXITING ... "<<endl;
            exit(0);
        case 1:
            addInstructorWorkload();
            break;
        case 2:
            addStudentWorkload();
            break;
        case 3:
            processWorkload();
            break;
        case 4:
            displayUsers();
            break;
        default:
            cout<<"INVALID OPTION!!! Try again"<<endl;
    }//switch
```

Fig.4. Main menu code.

```
**********************************************************************
*************** 0 - EXIT PROGRAM                          *************
*************** 1 - ADD AN INSTRUCTOR SERVICE REQUEST     *************
*************** 2 - ADD A STUDENT SERVICE REQUEST         *************
*************** 3 - SERVE (PROCESS) A REQUEST             *************
*************** 4 - DISPLAY USER PAYMENTS                 *************
**********************************************************************

Pick an option from above (int number from 0 to 3):
```

Fig 5. Main menu output.

The first main menu option is rather trivial and already implemented. We will discuss the others in the subsequent sections. Fig. 6 gives a brief overview of the function processWorkload() function of the main menu. Of course, your job here is to recursively implement processWorkload(x, y, z…..) (so it can have as much parameters as you need. For example: processWorkload(functionName, userName, userID)) function for the SU system.

```
void processWorkload()
{
    if (!instructorsQueue.isEmpty())
    {
        cout<<"Processing instructors queue..."<<endl;

        cout<<"Processing "<<"JOB NAME"<<"'s request (with ID "<<"JOB ID"<<") of service (function):\n"<<"FUNCTION NAME"<endl;
        //You need to implement the processWorkload --> you can modify inputs
        //processWorkload(...);

        cout<<"GOING BACK TO MAIN MENU"<<endl;
    }
    else if (!studentsQueue.isEmpty())
    {
        Create Jira Issue
        //TODO: This should print when you implemented 10 requests for instructors and 1 request for students logic
        //cout<<"10 instructors are served. Taking 1 student from the queue..."<<endl;

        cout<<"Instructors queue is empty. Proceeding with students queue..."<<endl;
        cout<<"Processing "<<"JOB NAME"<<"'s request (with ID "<<"JOB ID"<<") of service (function):\n"<<"FUNCTION NAME"<endl;
        cout<<"-------------------------------------------------------"<<endl;

        //You need to implement the processWorkload --> you can modify inputs
        //processWorkload(...);

        cout<<"GOING BACK TO MAIN MENU"<<endl;
    }
    else
    {
        cout<<"Both instructor's and student's queue is empty.\nNo request is processed."<<endl<<"GOING BACK TO MAIN MENU"<<endl;
    }
}
```

Fig 5. Recursive processWorkload() function

5. Reading input files

If any of the input files fails to open (file doesn't exist, hardware problem, etc.…) the program should terminate after displaying a proper message. Otherwise the input files sequence should be given in the following manner (Fig. 6)

```
If you want to open a service (function) defining file,
then press (Y/y) for 'yes', otherwise press any single key
y
Enter the input file name: input1.txt
Do you want to open another service defining file?
Press (Y/y) for 'yes', otherwise press anykey
y
Enter the input file name: input2.txt
Do you want to open another service defining file?
Press (Y/y) for 'yes', otherwise press anykey
N
```

Fig.7. Reading input files

When we are finished with the input files (and the construction of the linked lists of Fig. 3), we should display all the services with their corresponding commands in the following manner (fig. 8).
Note!!!: You can assume there are no loops in the functions (e.g. a function that calls itself or a function_x that calls a function_y, which instead calls function_x again). We might also have more (or less) than three input files.

```
PRINTIG AVAILABLE SERVICES (FUNCTIONS) TO BE CHOSEN FROM THE USERS
----------------------------------------------------------------


function_1::
print stack;, define x;, define y;, print stack;, call function_2;, print stack;, define z;, print stack;, call function_3;, pr
int stack;, define c;, define b;, print stack;, .

function_2::
print stack;, define a;, define b;, print stack;, call function_3;, print stack;, define x;, define y;, define z;, print stack;
, call function_3;, print stack;, define c;, print stack.
```

Fig.8. Displaying the read services (functions).

After this, we should be sent to the main menu.

6. Add an instructor service request

When choosing the 2nd option ('add an instructor service request'), firstly we are prompted for the name of the service (function) we are asking for. If the asked service doesn't exist, a suitable message should be printed and we are sent back to the main menu (Fig. 9).

```
**********************************************************************
********** 0 - EXIT PROGRAM                          **********
********** 1 - ADD AN INSTRUCTOR SERVICE REQUEST     **********
********** 2 - ADD A STUDENT SERVICE REQUEST         **********
********** 3 - SERVE (PROCESS) A REQUEST             **********
********** 4 - DISPLAY USER PAYMENTS                 **********
**********************************************************************

Pick an option from above (int number from 0 to 3): 1
Add a service (function) that the intructor wants to use:
Funcata
The requested service (function) does not exist.
GOING BACK TO MAIN MENU

**********************************************************************
********** 0 - EXIT PROGRAM                          **********
********** 1 - ADD AN INSTRUCTOR SERVICE REQUEST     **********
********** 2 - ADD A STUDENT SERVICE REQUEST         **********
********** 3 - SERVE (PROCESS) A REQUEST             **********
********** 4 - DISPLAY USER PAYMENTS                 **********
**********************************************************************

Pick an option from above (int number from 0 to 3):
```

Fig. 9. Requesting unavailable service.

If the requested service exists, then we are prompted for the name and ID of the instructor. Subsequently, his/hers request is put in the instructors queue and a message is displayed and we are sent back to the main menu (fig.10) Note that name is only one word.

```
**********************************************************************
**************** 0 - EXIT PROGRAM                          *************
**************** 1 - ADD AN INSTRUCTOR SERVICE REQUEST     *************
**************** 2 - ADD A STUDENT SERVICE REQUEST         *************
**************** 3 - SERVE (PROCESS) A REQUEST             *************
**************** 4 - DISPLAY USER PAYMENTS                 *************
**********************************************************************

Pick an option from above (int number from 0 to 3): 1
Add a service (function) that the intructor wants to use:
function_1
Give instructor's name: Kamer
Give instructor's ID (an int): 12345
Prof. Kamer's service request of function_1:
has been put in the instructor's queue.
Waiting to be served...

**********************************************************************
**************** 0 - EXIT PROGRAM                          *************
**************** 1 - ADD AN INSTRUCTOR SERVICE REQUEST     *************
**************** 2 - ADD A STUDENT SERVICE REQUEST         *************
**************** 3 - SERVE (PROCESS) A REQUEST             *************
**************** 4 - DISPLAY USER PAYMENTS                 *************
**********************************************************************

Pick an option from above (int number from 0 to 3):
```

Fig. 10. Instructor's service request added to the instructors queue.

7. Add a student service request

When choosing the 3rd option ('add a student service request'), similarly as we did with the instructor, firstly we are prompted for the name of the service (function) we are asking for. If the asked service doesn't exist, a suitable message should be printed and we are sent back to the main menu (fig.11).

```
**********************************************************************
**************** 0 - EXIT PROGRAM                          *************
**************** 1 - ADD AN INSTRUCTOR SERVICE REQUEST     *************
**************** 2 - ADD A STUDENT SERVICE REQUEST         *************
**************** 3 - SERVE (PROCESS) A REQUEST             *************
**************** 4 - DISPLAY USER PAYMENTS                 *************
**********************************************************************

Pick an option from above (int number from 0 to 3): 2
Add a service (function) that the student wants to use:
helo
The requested service (function) does not exist.
GOING BACK TO MAIN MENU

**********************************************************************
**************** 0 - EXIT PROGRAM                          *************
**************** 1 - ADD AN INSTRUCTOR SERVICE REQUEST     *************
**************** 2 - ADD A STUDENT SERVICE REQUEST         *************
**************** 3 - SERVE (PROCESS) A REQUEST             *************
**************** 4 - DISPLAY USER PAYMENTS                 *************
**********************************************************************

Pick an option from above (int number from 0 to 3):
```

Fig. 11. Requesting unavailable service.

If the requested service exists, then we are prompted for the name and ID of the student. Subsequently, his/hers request is put in the instructors queue and a message is displayed and we are sent back to the main menu. In fig.12 we illustrate the addition of two student requests.

```
Pick an option from above (int number from 0 to 3): 2
Add a service (function) that the student wants to use:
function_2
Give student's name: Orhun
Give student's ID (an int): 34611
Orhun's service request of function_2: has been put in the student's queue.
Waiting to be served...

**********************************************************************
********** 0 - EXIT PROGRAM                          **********
********** 1 - ADD AN INSTRUCTOR SERVICE REQUEST     **********
********** 2 - ADD A STUDENT SERVICE REQUEST         **********
********** 3 - SERVE (PROCESS) A REQUEST             **********
********** 4 - DISPLAY USER PAYMENTS                 **********
**********************************************************************

Pick an option from above (int number from 0 to 3): 2
Add a service (function) that the student wants to use:
function_1
Give student's name: Amro
Give student's ID (an int): 20343
Amro's service request of function_1: has been put in the student's queue.
Waiting to be served...
```

Fig. 12. Two student service requests added to the students queue.

8. Serve (process) a request

When the 4th option is chosen, we serve a single request, starting with instructor's ones (if available). Whenever we encounter the 'define' command of a function, we put it in the shared stack, when we encounter the 'print stack' command we print the stack trace and when we encounter the 'call' function command, the corresponding function is called and processed recursively. You can assume that if there is a 'call' command inside a function, the called function exists in the linked list of offered services you build in the beginning. When a function is about to finish, you should pause the program for a while (use system("pause")) and clear the stack from its data (as we already described). Initially the stack is empty.

Fig. 13. Shows the output when processing an available instructor request.

If there is not any instructor request (or we processed 10 instructor requests) we proceed with a student one (of course, if available). Fig. 14 shows the processing of a student request (that we added in the previous section). These figures also shows the program behavior in case of a called function is not defined (for example if input3.txt is not given to the program, function_3 is not defined.).

```
Pick an option from above (int number from 0 to 3): 3
Processing instructors queue...
Processing prof.Kamer's request (with ID 12345) of service (function):
function_1
-------------------------------------------------------
Executing print stack; command from function_1
PRINTING THE STACK TRACE:
The stack is empty
Executing print stack; command from function_1
PRINTING THE STACK TRACE:
function_1: define x;
function_1: define y;
Calling function_2 from function_1
Executing print stack; command from function_1
PRINTING THE STACK TRACE:
function_1: define x;
function_1: define y;
Executing print stack; command from function_1
PRINTING THE STACK TRACE:
function_1: define x;
function_1: define y;
function_1: define z;
Calling function_3 from function_1
Executing print stack; command from function_1
PRINTING THE STACK TRACE:
function_1: define x;
function_1: define y;
function_1: define z;
Executing print stack command from function_1
PRINTING THE STACK TRACE:
function_1: define x;
function_1: define y;
function_1: define z;
function_1: define c;
function_1: define b;
function_1 is finished. Clearing the stack from it's data...
GOING BACK TO MAIN MENU
```

Fig.13. Processing an instructor request of function_1. (Notice that function_2 and function_3 are not defined.)

```
Pick an option from above (int number from 0 to 3): 3
Instructors queue is empty. Proceeding with students queue...
Processing Orhun's request (with ID 34611) of service (function):
function_2
-------------------------------------------------------
Executing print stack; command from function_2
PRINTING THE STACK TRACE:
The stack is empty
Executing print stack; command from function_2
PRINTING THE STACK TRACE:
function_2: define a;
function_2: define b;
Calling function_3 from function_2
Executing print stack; command from function_2
PRINTING THE STACK TRACE:
function_2: define a;
function_2: define b;
Executing print stack; command from function_2
PRINTING THE STACK TRACE:
function_2: define a;
function_2: define b;
function_2: define x;
function_2: define y;
function_2: define z;
Calling function_3 from function_2
Executing print stack; command from function_2
PRINTING THE STACK TRACE:
function_2: define a;
function_2: define b;
function_2: define x;
function_2: define y;
function_2: define z;
Executing print stack command from function_2
PRINTING THE STACK TRACE:
function_2: define a;
function_2: define b;
function_2: define x;
function_2: define y;
function_2: define z;
function_2: define c;
function_2 is finished. Clearing the stack from it's data...
GOING BACK TO MAIN MENU
```

Fig. 14. Serving a student request. (Notice that the function_3 is not defined.)

If both the instructors and students queues are empty, we have the following output (fig.15).

```
Pick an option from above (int number from 0 to 3): 3
Both instructor's and student's queue is empty.
No request is processed.
GOING BACK TO MAIN MENU
```

Fig. 15. No request served (empty instructor and student queues)

9. Displaying user payments

5th option simply prints the list of users and their total payments (fig.16). Prints should be ordered according to order of request addition (not processing).

```
Pick an option from above (int number from 0 to 3): 4
Name: Kamer ID: 12345 0 TRY
Name: Orhun ID: 34611 0 TRY
Name: Amro ID: 20343 0 TRY
```

Fig. 16. Payment Display

Some Remarks

You can modify and use stack, queue and linked lists classes from your labs or implement those data structures from scratch.

You need to implement functionalities as they are asked, for example you can't implement an iterative function for processWorkload(). Also refer to the **Function Overloading** concept in C++ before implementing the recursive function.

Some Important Rules

Although some of the information is given below, first, please read the homework submission and grading policies in the course webpage and lecture notes of the first week. In order to get a full credit, your programs must be efficient and well commented and indented. Presence of any redundant computation or bad indentation, or missing, irrelevant comments may decrease your grades if we detect them. 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.

When we grade your homework we pay attention to these issues. Moreover, in order to observe the real performance of your codes, we are going to run your programs in *Release* mode and **we may test your programs with very large test cases**.

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 above mentioned platform while testing your homework. You need to place your first and last name in the program (as a comment line of course).

Submissions guidelines are below. Some 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.

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 cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the file name must be :

Cago_Ozbugsizkodyazaroglu_Caglayan_hw2.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. 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:

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_hw1.zip is a valid name, but

hw1_hoz_HasanOz.zip, HasanOzHoz.zip

are **NOT** valid names.

Submit via SUCourse ONLY! You will received no credits if you submit by other means (email, 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!

CS204 Team (Kamer Kaya, Muhammed Orhun Gale)