COMP3234 Computer and Communication Networks

Programming Project

Total 16 points

Version 1.1

Final Due Date: 5pm April 5, 2017

Hand-in the assignment via the Moodle System.

P2PChat Application

Overview

In this project, you are going to design and implement an instant chat program that supports message exchanges between peers directly. To find out which chatroom groups are available and discover the contact information of all group members in the chatroom, the system uses a Room server to keep track of the information. In the project, you will implement ONE program only, *the P2PChat program*, which is both a client and a server. You don't have to implement the Room server program as we shall provide the program to you. However, your P2PChat program must contact the Room server program frequently in order to maintain and retrieve grouping information.

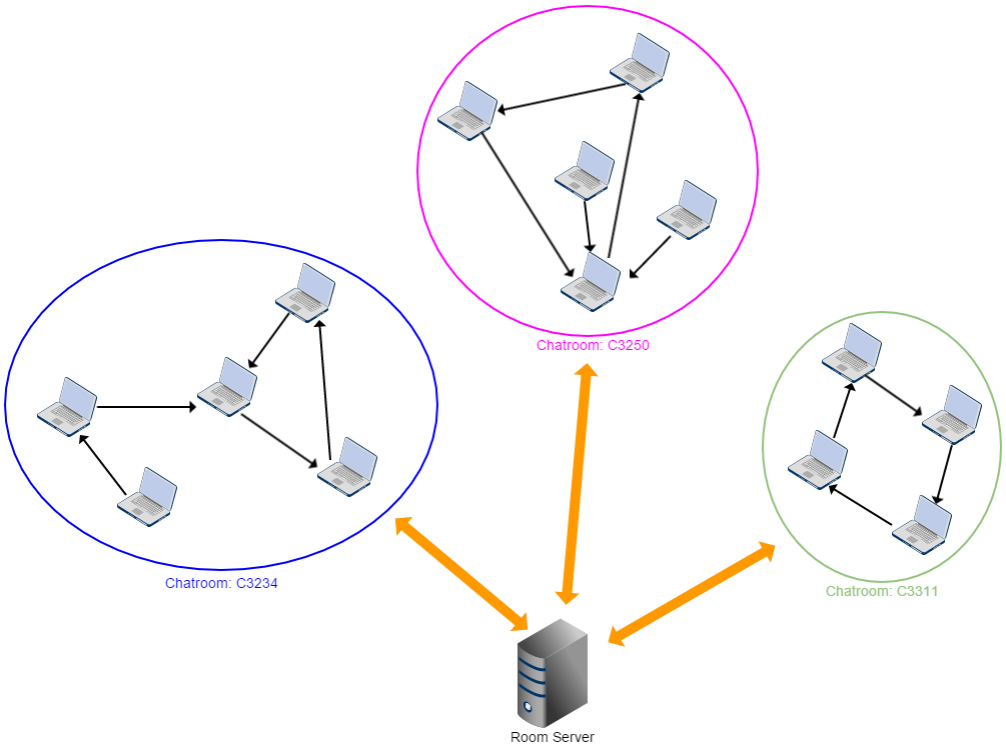

Figure 1. A system view of the P2PChat application

Objectives

1. An assessment task related to ILO4 [Implementation] – "be able to demonstrate knowledge in using Socket Interface to design and implement a network application".
2. A learning activity to support ILO1, ILO2a, ILO2d, & ILO4.
3. The goals of this programming project are:
   - to get a solid experience in using Socket functions to implement a real-life protocol;
   - to get a good understanding of how a text-based peer-to-peer networking protocol works as well as how to implement one.

Design

User Interface

The P2PChat program makes use of the Tkinter module to implement the UI for handling all user input and displaying chat messages. Tkinter is the standard GUI library for Python. You are not required to write the UI, as the UI framework (**P2PChat-UI.py**) will be provided to you. It consists of the necessary initialization Tk code to draw the UI.

Figure 2. The User Interface of the P2PChat program

Message Window – this is the place where all chat messages are displayed.

- To print a message to the Message Window, the system uses the "insert()" method to add the message to the top of the Message Window, e.g., MsgWin.insert(1.0, "\nChat message to be display")

Command Window – this pane is for display system messages that show what actions/events have happened.

- To print a message to the Command Window, the system uses CmdWin.insert() to add it to the top of the Command Window, e.g., CmdWin.insert(1.0, "\nSystem message to be displayed")

User Input and buttons – this is the middle pane with 5 buttons and a text field for accepting user input.

- Each button has its associated function defined in the UI framework. For example, when the [List] button is pressed, the system runs the "do_List()" function.
- To read in the text input from the user input field, the system uses the "get()" method, e.g., userentry.get(). To clear the user input field after getting the text, the system uses "userentry.delete(0, END)" method to clear all contents in the input field.

Specification of the P2PChat program

The P2PChat program accepts three command line arguments:

P2PChat <roomserver_address> <roomserver_port> <myport>

with <roomserver_address> specifies the hostname or IP address of the Room server, <roomserver_port> specifies the port number listened by the Room server, and <myport> specifies the listening port number used by the P2PChat program (for accepting TCP connections).

There are in total 5 commands for the end-user to control the P2PChat program; their actions are specified as follows.

- [User] button – before joining any chatroom group, the end-user must register with the P2PChat program his/her nickname (username), which will be appeared in all messages sent by the end-user to the chatroom group as well as be appeared in the member list of that chatroom group stored in the Room server. The end-user can rename his/her username by using this button only before joining any chatroom group. After the end-user has registered his/her username, the program prints a message in the Command Window. You can assume that the username consists of a single word with a length of at most 32 printable ASCII characters (exclude the ':' character which is being used as the sentinel symbol in our protocol).
- [List] button – to get the list of chatroom groups registered in the Room server by sending a LIST request to the Room server. After receiving the list of chatroom groups from the Room server, the program outputs the chatroom names to the Command Window. You can assume that the chatroom name consists of a single word with the length of at most 32 printable characters (exclude the ':' character).
- [Join] button – to join a target chatroom group. The end-user must provide the target chatroom name via the user input field; otherwise, the program displays an error message in the Command Window. If the P2PChat program has already joined a chatroom group, the system should reject this request and display an error message in the Command Window. After getting the chatroom name, the P2PChat program sends a JOIN request to the Room server, and if is succeeded, the Room server should send back the list of members in that chatroom together with their contact info. The member list should include this newly joined P2PChat program. Once, the P2PChat program gets the member list, it tries to join the chatroom network by initiating a TCP connection to one of the chatroom's members. Once successfully linked to a member, this P2PChat program is considered as successfully CONNECTED to the chatroom network and it reports the status to the end-user via the Command Window. More details on the interactions between the P2PChat program, the Room server, and another peer will be covered in the communication protocol section.
- [Send] button – to send a message to the chatroom network. The P2PChat program must be CONNECTED to the chatroom network before sending a message. The message is sent to the chatroom network via all peers that are currently connected (forward link and backward links) to this P2PChat program. The program also displays the message to the Message Window together with the username to identify who gave out this message. You may assume the length of the text content is less than 500 bytes, but it is possible that the ':' character may appear in the text content.
- [Quit] button – to exit from the program. Before termination, the P2PChat program closes all TCP connections and releases all resources.

The state diagram in Figure 3 shows the basic idea of how these commands are used.

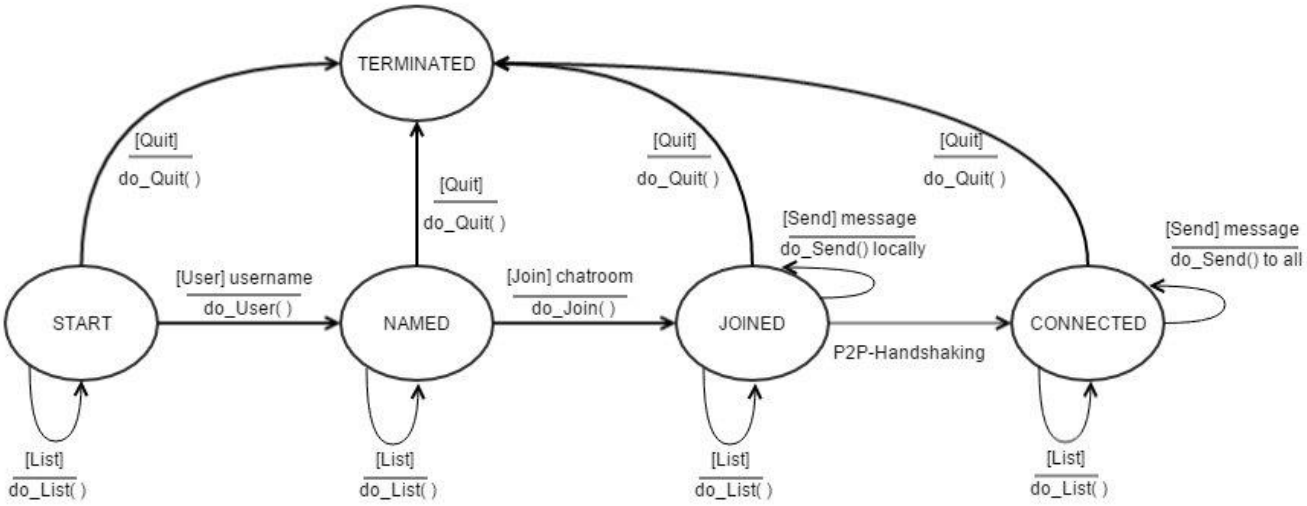

Figure 3. The state diagram of the P2PChat program

Communication Protocol

All communications happened between P2PChat peers and between P2PChat and Room server are in text-based format and all messages must end with the four characters "::\r\n". In order to make sure all parties can communicate successfully, ***you are not allowed to modify, add or change the protocol***.

Here is the set of message exchanges between P2PChat peers and between P2PChat and Room server. All messages start with the first character as the Message Type and follow by zero to more fields that are separated by the ':' character.

Request Message **(from P2PChat to Room server)**	**Response Message** **(from Room server to P2PChat)**
To get the list of active chatroom groups. ***LIST request*** **L::\r\n**	Server responds with: If no chatroom group **G::\r\n** If has one or more chatroom groups **G:Name1:Name2:Name3::\r\n** If encounters error **F:error message::\r\n** Name1, Name2, Name3, … represent the names of all available chatroom groups.
To join a specific chatroom group. ***JOIN request*** **J:roomname:username:userIP:userPort::\r\n** roomname gives the name of the chatroom group; username, userIP, & userPort represent the nickname of the end-user, IP address in dotted-decimal notation (as string), and the listening port number (as string) used by the requesting P2PChat peer.	Server responds with: The list of group members in that group **M:MSID:userA:A_IP:A_port:userB:B_IP:B_port::\r\n** If encounters error **F:error message::\r\n** MSID is the hash value of all membership info in this message; userA, A_IP, & A_port represent the nickname, IP addr, & listening port number (all as strings) used by the member A.

Request Message **(from P2PChat to P2PChat)**	**Response Message** **(from P2PChat to P2PChat)**
To establish a connection to a chatroom member. ***PEER-to-peer handshaking procedure*** **P:roomname:username:IP:Port:msgID::\r\n** roomname gives the name of the chatroom group; username, IP, & port are the nickname, IP address, & listening port of the requesting peer; msgID is the ID number of the last message used by the initiating peer (all peers have this number initially set to zero).	The target P2PChat program responds with: If successfully connected **S:msgID::\r\n** If encounters error *No response; just close the connection* msgID is the ID number of the last message used by this peer.
To forward a message to a connected peer. ***TEXT message forwarding*** **T:roomname:originHID:origin_username:msgID:msgLength:Message content::\r\n** roomname gives the name of the chatroom group; originHID & origin_username are the Hash ID and nickname of the origin sender; msgID & msgLength are the ID number and length of this message; message content is the actual text message.	

The following is a detailed specification of the communication protocol: how the P2PChat program interacts with the Room server, and how the P2PChat program interacts with another P2PChat peer. All information and actions are structured according to the triggering command.

- When the end-user presses the [List] button, the system sends a LIST request to the Room server via a TCP connection. If the TCP connection hasn't been established, the system initiates a connection to the Room server before sending the LIST request. The Room server should respond with the list of chatroom names (if any) or an error response if the server has experienced a problem in this interaction.
- When the end-user presses the [Join] button, the system first checks whether the end-user has already inputted the username (nickname); if not, it should prompt the end-user to input his/her username first. If the system checks that it has already joined a chatroom group, it should inform the end-user and reject this request. Otherwise, the system sends a JOIN request to the Room server via a TCP connection. If the TCP connection hasn't been established, the system initiates a connection to the Room server before sending the JOIN request.
    - Upon receiving this JOIN request, the Room server checks whether the target chatroom has been created. If that chatroom hasn't been created, the Room server creates a new chatroom and add this requesting P2PChat peer to this new chatroom. If the target chatroom already exists, the Room server just adds this requesting P2PChat peer to the chatroom. Afterward, the Room server sends back the membership list to the requesting P2PChat program. The membership list is uniquely identified by a hash value; this value will be changed whenever there are updates on the membership list. If the Room server encounters any problem in handling this request, it sends back an error response to the requesting P2PChat program.
    - Once the P2PChat program has successfully received the chatroom's membership list, it is considered as successfully JOINED the chatroom group. From now on, it should keep the TCP connection to the Room server be opened; otherwise, the Room server would remove this

P2PChat peer from the chatroom's member list. In addition, the P2PChat program starts running the ***KEEPALIVE procedure***.

- After receiving the membership list, the P2PChat program (let's call it ***H***) uses the following logic to select a P2PChat peer for initiating a TCP connection to that peer so as to connect ("*forward link*") to the chatroom network.

```
1. for each member in the list, calculates the Hash ID by applying the sdbm_hash()
   function on a string composed by username+IPAddr+port of that member.
2. sort the membership list according to the Hash IDs.
3. use H's Hash ID (myHashID) to find its index X in the sorted membership list (gList).
4. start = X+1
5. while gList[start].HashID not equal to myHashID do
6.     if there is an existing TCP connection between the member at gList[start] and H
       (i.e., a "backward link")
7.         start = (start + 1) % gList.size (i.e., wrap around if reaching the end)
8.         goto step 5
9.     else
10.        establish a TCP connection to the member at gList[start]
11.        if successfully established
12.            run the Peer-to-peer Handshaking procedure
13.            if successful
14.                declare successfully FORWARD LINKED
15.                add this connection to H's socket list
16.                update gList to indicate this link
17.                jump out of the while loop
18.            else
19.                start = (start + 1) % gList.size
20.                goto step 5
21.        else
22.            start = (start + 1) % gList.size
23.            goto step 5
24. if failure to establish a forward link
25.     report error and schedule to retry the above logic later
```

- From now on, the P2PChat program keeps listening to any incoming TCP connection. Once a new TCP connection is accepted, the initiating peer (i.e. the other end) starts running the Peer-to-peer Handshaking procedure. If successfully completed the handshaking procedure, the program should declare that it has a new ***BACKWARD LINK*** to the chatroom network, add this TCP connection to its socket list and update the member list to indicate this link. In here, the P2PChat program may face a situation: the newly connected TCP connection is initiated from an unknown peer, which is not in the member list held by this P2PChat program. To verify whether this unknown peer is in the most updated member list, this P2PChat program resends a JOIN request to the Room server, which triggers the Room server to send back the latest member list. If the unknown peer is in the updated member list, the P2PChat program should continue with the handshaking procedure; otherwise, it should close the TCP connection that indicates the handshaking procedure is unsuccessful.

- Once a peer has at least one link (forward or backward) connected to the chatroom network, it is considered as in the **CONNECTED** state. A CONNECTED peer could have any number of backward links but at most one forward link at any time.
- KEEPALIVE procedure
  - To maintain its membership in the chatroom group, the P2PChat program must resend the JOIN request to the Room server in ***every 20 seconds***; this indicates that this P2PChat peer is still an active member in that chatroom group. If the Room server does not hear from a particular P2PChat peer for ***more than 60 seconds***, it will remove that peer from the member list. After receiving the JOIN request, the Room server sends back the latest member list to this P2PChat peer. This P2PChat program has to handle the updated list appropriately. Each response message to the JOIN request carries a hash value, which is generated by applying the sdbm_hash() function to all membership info in the current member list. If the member list hasn't been updated, the hash value would not be changed. If the member list is updated, a new hash value would be generated.
- After CONNECTED to the chatroom network, the P2PChat program may receive TEXT message forwarded from any peer that is (forward or backward) linked to this P2PChat program. Upon receiving such TEXT message, the program first checks the roomname to see if this is the chatroom group the program is in. If not, it drops the TEXT message and outputs an error message. If this is the correct chatroom group, the program should use the originHID and msgID to make sure that this is a new message from the origin peer that the program hasn't seen before. If the program has seen this TEXT message before, it drops the TEXT message and outputs an error message. If this is a new message from that originHID peer, the program displays the origin username and the message content to the Message Window and forwards this TEXT message to all peers that are linked (forward or backward) to this program (except the origin peer and the peer which this TEXT message was coming from). In addition, the program memorizes this msgID as the last message ID used by this originHID peer. In here, a P2PChat peer may experience this situation: the message was originated from an unknown peer. Since this peer does not have the complete info of the unknown peer, it has to contact the Room server to get the latest member list; therefore, this P2PChat peer has to resend a JOIN request to the Room server.

- When the end-user presses the [Send] button, the program first checks whether the end-user has inputted any content via the input text field. If not, it just ignores this event; otherwise, it checks whether the program has joined or connected to a chatroom network. If it is connected to a chatroom network, the program creates a new TEXT message with a new msgID and forwards this message to all peers that are linked (forward or backward) to this program. The program then displays the username and the message content to the Message Window. To make the program more useful and flexible, the program should accept any ASCII text contents from the end-user; thus, it is possible that the special character ':' may appear many times in the inputted content. The P2PChat program should send exactly what inputted by the end-user without adding extra padding characters; it is the receiving peers that do the work to extract and display the text content correctly.

Specification of the Room Server program

The behavior of your P2PChat program MUST match with the behavior of the Room server program. Without this compatibility, the Room server program cannot talk to the P2PChat program successfully. Below is the pseudocode of the provided Room server program, you can use it as a starting point for your P2PChat program development.

Room server main process

```
set up a listening socket
create monitor thread
loop forever
        client-sock = accept()
        create a client thread and pass client-sock to the thread
        add client thread to clientList
```

Client thread

```
loop forever
        msg = recv(client-sock)
        if msg == 0
                encounter connection error
                remove this thread from clientList
                close client-sock
                thread exit
        else msg > 0
                if msg 1st byte == 'L'
                        read chatroom list
                        send back a response
                else if msg 1st byte == 'J'
                        extract other contents from msg
                        calculate Hash ID
                        if client Hash ID is already listed in the chatroom
                                renew the timestamp
                        else
                                if chatroom already exists
                                        add the client to the chatroom
                                        record the timestamp
                                else
                                        create a new chatroom
                                        add the client as the first member
                                        record the timestamp
                        retrieve membership list
                        send back the response
                else
                        handle error situation
                        send back a negative response
```

Monitor thread

```
loop forever
        sleep 20 seconds
        for each chatroom
                for each member in this chatroom
                        if current time - last timestamp > 60 seconds
                                remove this member from the chatroom
                                remove this client from the clientList
                        else
                                do nothing
                if no member in this chatroom
                        destroy this chatroom
```

shutdown handler
terminate the monitor thread
for each chatroom
    for each member
        terminate the client thread
        close client-sock
        remove this client from the clientList
    destroy this chatroom
for each remaining client in clientList
    terminate the client thread
    close client-sock
    remove the client from clientList

Once the Room server started, it runs forever. To terminate the server, end-user can use the SIGINT (kill -2 / Ctrl-c) signal to terminate the Room server and all its threads. The Room server binary program – *room_server* - is available to you for your testing. This program uses the number 32340 as its default listening port number, and it accepts one optional input argument, which is for assigning a different listening port number to the program. In addition, the program is compiled with a lot of debug printouts, which is useful for revealing the problems between the P2PChat program and the Room server.

Computer Platform to Use

For this project, you can develop and test your P2PChat program on any platform installed with **Python 3.x**. However, we only provide the executable binaries of the Room server program (room_server) under Ubuntu 14.04 (32-bit and 64-bit) and Mac OS X 10.12.

Submissions

In order to guide you through in developing the P2PChat program, we divide the project into two stages. Each stage has its own focus; thus, has unique test cases and marking scheme. Under this arrangement, it is easier for you to manage the project, and hopefully, easier for you to get higher marks.

You **can work individually** or **form a team of two to work on the project** (if you do not have much programming practice in the past, I recommend you forming a team to work on the project). Please register with us on your intention and here is the URL to the registration page on the course's Moodle site: http://moodle.hku.hk/mod/feedback/view.php?id=919396

Stage one (6 points)

In this stage, the focus of your implementation is purely on the interactions between the P2PChat program and the Room server. You should complete the implementation of the following functions:

- do_User()
- do_List()
- part of the do_Join() function, which involves the interactions between the P2PChat program and the Room server in handling the JOIN request and the subsequent JOIN requests in the KEEPALIVE procedure. You are not required to keep/store/update the membership list at this stage; however, you should simply display the members' info to the Command Window.

Test cases:

- Full function of the [User] button: reject empty username, registration of username, change the username before JOINED, and reject the request of change of username after JOINED
- Full function of the [List] button: successfully get and display the chatroom list by using the LIST request at any time
- P2PChat-Roomserver interactions: successfully create a new chatroom group and display the member list, successfully join an existing chatroom group and display the member list, and successfully maintain the JOINED state by running the KEEPALIVE procedure.

Submission [Optional]: Please name your program file as: *P2PChat-stage1.py* and add necessary documentation.
Deadline: March 15, 2017 (Wednesday) at 5:00pm.

Stage two (9 points)

In this stage, the focus of your implementation is on the interactions between the P2PChat peers. You should complete the implementation of the following functions:

- the remaining part of the do_Join() function, which involves all the interactions between the P2PChat peers, such as:
  - set up a forward link to one member in the chatroom network
  - accept the backward link from any peer in the chatroom network
  - receive an incoming TEXT message and forward the TEXT message to connected peers if is an unseen message
- do_Send()
- do_Quit()

Test cases:

- Full function of the [Send] button: reject empty user input, display the messages correctly (even with ':' in the input text) in the Message Window, and forward and display the messages to all connected peers successfully.
- Full function of the [Quit] button: do all housekeeping works. Tutors will manually inspect your program to check on this aspect.
- P2PChat peer interactions:
  - connect at least 5 instances of the P2PChat program (one after the other) to one chatroom group and see whether they can successfully connect and exchange messages under normal situation, i.e., no duplicated messages, no missing messages, the text contents are correctly displayed in the Message Window of each peer.
  - connect at least 5 instances of the P2PChat program to one chatroom group, then select a peer with the largest number of backward links and terminate it; wait for a few seconds and check whether the remaining peers can maintain the connectivity of the overlay network and successfully exchange messages.
  - connect at least 5 instances of the P2PChat program to one chatroom group, then terminate two peers with the largest number of backward links; wait for a few seconds and check whether the remaining peers can maintain the connectivity and successfully exchange messages.
  - connect at least 5 instances of the P2PChat program to one chatroom group, then terminate two peers with the largest number of backward links; wait for a few seconds and restart one peer to join the chatroom group again; check whether the chatroom network can work correctly.

Final submission: Please name your program file as: *P2PChat.py* and add necessary documentation.
Deadline: April 5, 2017 (Wednesday) at 5:00pm.

Format for the documentation

1. At the head of the submitted program file, state clearly the
   - Student name and No.:
   - Student name and No.:
   - Development platform:
   - Python version:
   - Version:
2. Inline comments (try to be detailed so that your code could be understood by others easily)

Grading Policy

1. The tutors will first test your final submission after the project deadline. If your program fully complies with the project specification, your team will get all marks for all two stages automatically. Otherwise, the tutors will test your Stage one submission (if any) to give marks.
2. After submission of Stage one program, if you want to know whether your program works correctly, your team is welcome to make an appointment with the tutors and demonstrate your program to the tutors. Therefore, you could receive feedback to improve your program for the second stage.
3. As the tutors will check your source code, please write your program with good readability (i.e., with good code convention and sufficient comments) so that you will not lose marks due to possible confusions.

Documentation (1 point)	• Include required program and student's info at the beginning of the program • Include necessary documentation to clearly indicate the logic of the program
Stage 1 (6 points)	• Correctness of the [User] button (1.5 points) • Correctness of the [List] button (1.5 points) • Correctness of the P2PChat-Roomserver interactions (3 points)
Stage 2 (9 points)	• Correctness of the [Quit] button (1 point) • Correctness of the [Send] button (2 points) • Correct interactions between multiple P2PChat peers under normal situation (3 points) • Correct responses and interactions between multiple P2PChat peers with peers leaving and rejoining the chatroom network (3 points)

Plagiarism

Plagiarism is a very serious academic offence. Students should understand what constitutes plagiarism, the consequences of committing an offence of plagiarism, and how to avoid it. **Please note that we may request you to explain to us how your program is functioning as well as we may also make use software tools to detect software plagiarism.**