ASSIGNMENT 2 FRONT SHEET

Qualification	**BTEC Level 5 HND Diploma in Computing**		
Unit number and title	Unit 14: Business Intelligence		
Submission date	July 1st 2020	**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	
Student Name	Nguyen Trung Tinh	**Student ID**	GCD18753
Class	GCD0704	**Assessor name**	Phan Thanh Tra
Student declaration I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.			
		Student's signature	

Grading grid

P4	P5	P6	P7	M4	M5	D3	D4
✓	✓	✓	✓	✓	✓		

❒ **Summative Feedback:** 		❒ **Resubmission Feedback:**
Grade:	**Assessor Signature:**	**Date:**
Internal Verifier's Comments:		
IV Signature:		

Contents

Figures

Tables

LO1 IMPLEMENT COMPLEX DATA STRUCTURES AND ALGORITHMS

P4 Implement a complex ADT and algorithm in an executable programming language to solve a well-defined problem.

In ordinary, queue and stack operations are used for applying store, recover data through the push and pop function. By real, the IT experts are applying their operations in the real-life application. For example:

- Email transferring.
- A ticket line;

- An escalator;
- A car washes.

Now, I'm going to implement one of the above problems in order to provide a specific document about queue application on the real-life, and the object-problem is Email-Transfer.

1. **Problem:** Create a program of mail transferring by queue structure but limited-character for each mail is 250 characters.

2. **Solution:**

We 're known about the structure of queue in the first assignment, a queue is referred to as a First-In-First-Out (FIFO) list. And its main operation includes in:

- Enqueue ().
- Dequeue ().

Applied it into the program of email transferring, we have the corresponding functions with each one:

- Write (): Solve the user mails by delete each one of mails from cache and send to another one.
- Send (): Put the user mail into memory of cache (queue).

For specific situation, I will create 2 arrays are correspond with 2 users (called by user1 and user2) from different positions and it contain the string of mail so we need a transfer () function to transfer data from user1 mail to user2. Moreover, in order to limit the character that each user input, we need to have a Count () function to count each character and eliminate if it exceeds the limited number of characters.

In conclusion, we list the constant that we analyzed:

- Array list: user1 and user2.
- Transfer () function.
- Count () function.
- Queue class and its construct with parameters (front, rear, sent_message…).

3. **Pseudo code:**

First step, I will create Queue that is such as a memory to store letters and it's limited to 20 areas. And I select the array to separate them by index memory but the main intentions are front index, rear index.

```
namespace Transfer_queue
{
    7 references
    public class Queue
    {
        //Let image that receivers
        int front;
        int rear;
        string[] user;
        string sent_message = null;
        2 references
        public Queue(string user_name)
        {
            this.user = new string[20];
            this.front = 0;
            user[front] = user_name;
            this.rear = -1;
        }
```

Figure 1 The Queue class and its constructor.

Sencond step, calling write () function with object user_name (example: sender. write ()) to create a memory area that contain the letter you want to send.

```
3 references
public void write()
{
    try
    {
        Console.WriteLine("Please input your letter (user 1) but it should not be exceeded " +
            "250 characters: ");
        string statement = Console.ReadLine();
        if (Count(statement) > 250)
        {
            Console.WriteLine("exceeded the number of chars");
            write();
        }
        else if (String.IsNullOrEmpty(statement) == true)
        {
            Console.WriteLine("Please input your content");
            write();
        }
        else
        {
            user[++rear] = statement;
            Console.WriteLine("Rear of my queue:  " + rear + ", rear element: " + user[rear]);
            Console.WriteLine("Front of my queue:  " + front + ", front element: " + user[front]);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("your storage is full, please delete by send your message to " +
            "anyone you want " + e.Message);
        Console.WriteLine();
    }
}
```

Figure 2 The write () function to add to queue

Next step, calling send() function with object user_name (example: sender. send ()) to eliminate letter from memory area (front_index) and send to receiver's memory area.

```
public string send(Queue receiver)
{
    if (front == rear + 1)
    {
        string error = "Please input your messager";
        return error;
    }
    else
    {
        try
        {
            Console.WriteLine(user[front] + " has send already.");
            front++;
            Console.WriteLine(user[front]);
            ++receiver.rear;
            transfer(receiver);
            Console.WriteLine("Loading.....................................");
            Console.WriteLine("The message transferring is successfull");
            Console.WriteLine("The letter of sender:" + "user name:{0}", user[front]);
            Console.WriteLine("The memory of receiver:" + "user name:{0} and content:{1}", receiver.user[receiver.front], receiver.user[receiver.rear]);
            Console.ReadLine();
            return user[front];
        }
        catch(Exception e)
        {
            if (receiver.rear==receiver.user.Length -1) {
                Console.WriteLine(e.Message);
                return send(receiver);
            }
            else
            {
                Console.WriteLine("check your message!");
                return send(receiver);
            }
        }
    }
```

Figure 3 The send () function to send the mail to receiver with specific message

Beside the main above functions, we have other ones supporting for them to implement the correct maintenances. Such as transfer() function, with the capability of swapping a memory element of queue with another one and execute it as sending the mail from one to others, count() function to count the characters to avoid the exceed of character limits.

```
1 reference
public bool transfer( Queue receiver)
{
    try {
        receiver.user[receiver.rear++] = user[front++];
        user[front++] = sent_message;
        return true;
    }
    catch(Exception e){
        Console.WriteLine("Cannot transfer");
        return false;
    }
}
```

Figure 4Transfer () function.

```
1 reference
static int Count(string value)
{
    int result = 0;
    foreach (char c in value)
    {
        if (!char.IsWhiteSpace(c))
        {
            result++;
        }
    }
    return result;
}
```

Figure 5 Transfer () and Count () functions.

```
2 references
public void print()
{
    if (front == rear + 1)
    {
        Console.WriteLine("Queue is Empty");
        return;
    }
    else
    {
        for (int i = front; i <= rear; i++)
        {
            Console.WriteLine(user[i]);
        }
    }
}
```

Figure 6 print () function to show each queue.

P5 Implement error handling and report test results.

1. **The error handling of queue:**

When executing code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, the program languages will normally stop and generate an error message except the way is the use of "if" statement. The technical term for this is: the program languages will throw an **exception** (throw an error).

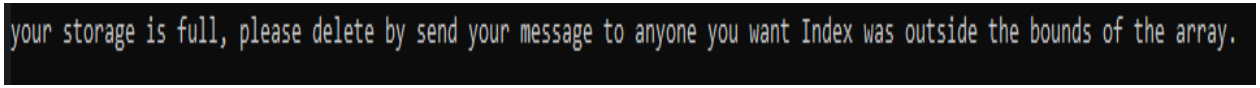
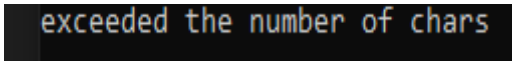
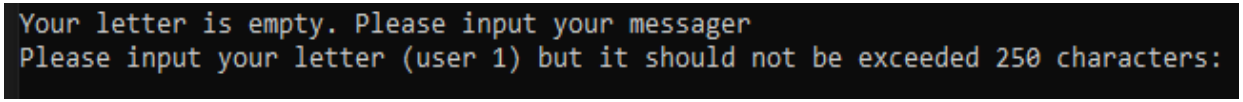
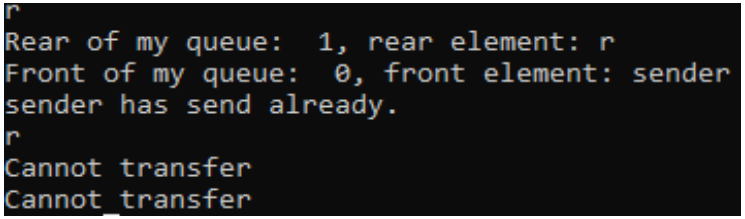
In the transfer() function, memory of receiver can be out of index array so a notification send to sender is recommended and in the send() function is correspond with transfer() function because this function is supported for each other.

```
2 references
public bool transfer( Queue receiver)
{
    try {
        receiver.user[++receiver.rear] = user[front];
        user[front] = sent_message;
        return true;
    }
    catch(Exception e){
        Console.WriteLine("Cannot transfer");
        return false;
    }
}
```

```
public void send(Queue receiver)
{
    if (rear==front)
    {
        Console.WriteLine("Your letter is empty. Please input your messager");
        write();
    }
    else
    {
        try
        {
            Console.WriteLine(user[front] + " has send already.");
            front++;
            Console.WriteLine(user[front]);
            ++receiver.rear;
            transfer(receiver);
            Console.WriteLine("Loading.....");
            Console.WriteLine("The message transferring is successfull");
            Console.WriteLine("The letter of sender:" + "user name:{0}", user[front]);
            Console.WriteLine("The memory of receiver:" + "user name:{0} and content:{1}", receiver.user[receiver.front], receiver.user[receiver.rear]);
            Console.ReadLine();
        }
        catch(Exception e)
        {
            if (receiver.rear==receiver.user.Length -1) {
                Console.WriteLine(e.Message);
            }
            else
            {
                Console.WriteLine("check your message!");
            }
        }
    }
}
```

Figure 7 try catch in mail transfer.

2. **Test case:**

Case	Actual Result	Status	Expected result
Memory of queue is more exceeded than requirement (20).		**Done**	your storage is full, please delete by send your message to anyone you want
The character is larger than 250 characters.		**Done**	exceeded the number of chars
No letter is additional for sending.		**Done**	Your letter is empty. Please input your massage. Please input your letter (sender) but it should not be exceeded 250 characters:
Cannot transfer as user 2 is full memory.		**Done**	Cannot transfer

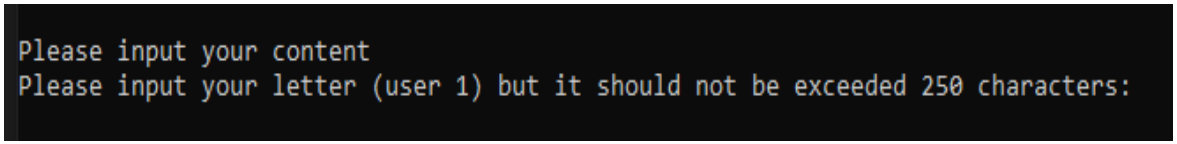
Input letter with the empty-content.		**Done**	Please input your content
Check node of queue with the out index.		**Done**	your message not exist.

Table 1 The analysis of error test case table

M4 Demonstrate how the implementation of an ADT/algorithm solves a well-defined problem.

1. **The system model of mail transferring:**

The mail transferring operated as queue and its simulation is such as the below models.

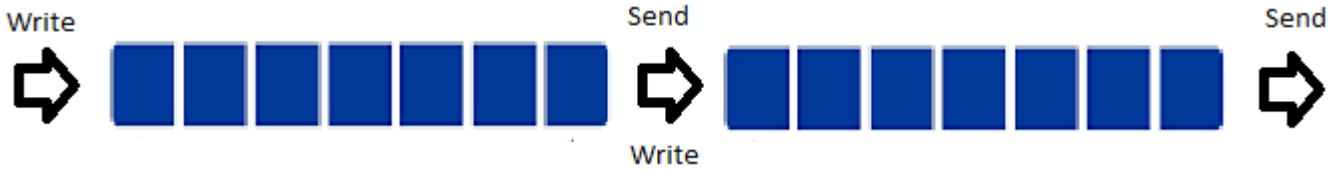


Figure 8 The mail transferring model.

Figure 9 The sender models.

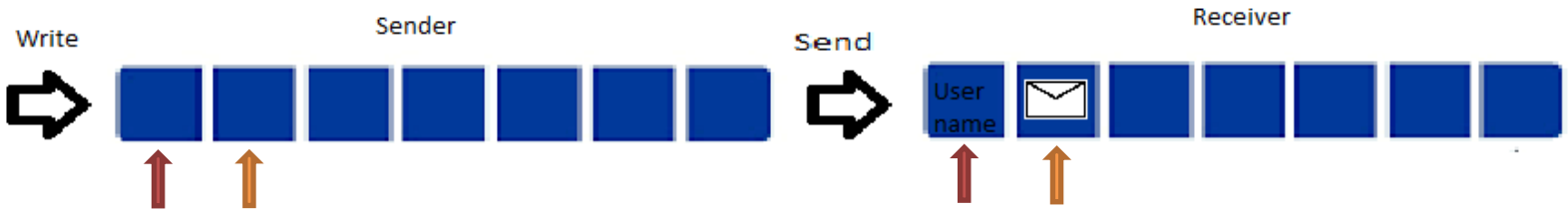


Figure 10 The mail transferring models

2. The implementation of pseudo code:

First, create a memory for user1(sender) and user2(receiver) up to 20 memory areas.

```
static void Main(string[] args)
{
    Console.WriteLine("the sender is user 1");
    Queue user1 = new Queue("user 1");
    Console.WriteLine("the receiver is user 2");
    Queue user2 = new Queue("user 2");
```

Figure 11 The implementation of user creation.

Secondly, calling the write () function to input the content of message with a specific variable as a stored-package but warning to the limitation of characters (250).

```
    string letter1 = user1.write();
    string letter2 = user1.write();
    string letter3 = user1.write();
    string letter4 = user1.write();
```

Figure 12 The implementation of writing the new text messages.

Thirdly, calling the send () function to send the content of message to the receiver object (user2).

```
    user1.send(user2);
    user1.send(user2);
```

Figure 13 The implementation of sending the text messages to specific receiver (user2).

Finally, in order to check, show or read the content of text message, user 2 must implement the following command.

```
    user2.print();
```

Figure 14 The execute command for user 2 to show the sent content.

LO4 ASSESS THE EFFECTIVENESS OF DATA STRUCTURES AND ALGORITHMS

P6 Discuss how asymptotic analysis can be used to assess the effectiveness of an algorithm

1. **The effectiveness of an algorithm analysis (performance analysis):**

   There are so many important things that should be concerned, like teamwork communication, modularity, security, maintainability, etc. Why to worry about performance?

   We can have all the above things only if we have performance. Therefore, performance is like currency through which we can buy all the above things. Another reason for studying performance is – speed.

   To summarize, performance is equal with scale. Imagine a text editor that can load 1000 pages, but can spell check 1 page per minute or an image editor that takes 1 hour to rotate your image 90 degrees left or … you get it. If a software feature cannot cope with the scale of tasks users need to perform – it is as good as dead.

2. **Asymptotic analysis:**

   Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance. Using asymptotic analysis, we can conclude the best case, average case, and worst-case solutions of an algorithm. (geeksforgeeks, 2020)

   We can have three cases to analyze an algorithm:

   1) Worst Case:

   In the worst case analysis, we calculate upper bound on running time of an algorithm. We must know the case that causes maximum number of operations to be executed. For Linear Search, the worst case happens when the element to be searched (x in the above code) is not present in the array. When x is not present, the search() functions compares it with all the elements of arr[] one by one. Therefore, the worst case time complexity of linear search would be $\Theta(n)$.

   2) Average Case:

   In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by total number of inputs. We must know (or predict) distribution of cases. For the linear search problem, let us assume that all cases are uniformly distributed (including the case of x not being present in array). So we sum all the cases and divide the sum by (n+1).

Average Case Time = $\frac{\sum_{i=1}^{n+1} \theta(i)}{(n+1)}$

$= \frac{\theta((n+1)*(n+2)/2)}{(n+1)}$

$= \Theta(n)$

3) Best Case:

In the best case analysis, we calculate lower bound on running time of an algorithm. We must know the case that causes minimum number of operations to be executed. In the linear search problem, the best case occurs when x is present at the first location. The number of operations in the best case is constant (not dependent on n). So time complexity in the best case would be Θ(1)

Most of the times, we do worst case analysis to analyze algorithms. In the worst analysis, we guarantee an upper bound on the running time of an algorithm which is good information.

The average case analysis is not easy to do in most of the practical cases and it is rarely done. In the average case analysis, we must know (or predict) the mathematical distribution of all possible inputs.

The Best Case analysis is bogus. Guaranteeing a lower bound on an algorithm doesn't provide any information as in the worst case, an algorithm may take years to run.

P7 Determine two ways in which the efficiency of an algorithm can be measured, illustrating your answer with an example.

1. **Ways are measuring for the efficiency of an algorithm:**

There are *two aspects* of algorithmic performance:

- ✓ Time
  - Instructions take time.
  - How fast does the algorithm perform?
  - What affects its runtime?
- ✓ Space
  - Data structures take space
  - What kind of data structures can be used?
  - How does choice of data structure affect the runtime?

2. **Example:**

Given two algorithms for a task, how do we find out which one is better?

One naive way of doing this is – implement both the algorithms and run the two programs on your computer for different inputs and see which one takes less time. There are many problems with this approach for analysis of algorithms.

1) It might be possible that for some inputs, first algorithm performs better than the second. And for some inputs second performs better.

2) It might also be possible that for some inputs, first algorithm performs better on one machine and the second works better on other machine for some other inputs.

Asymptotic Analysis is the big idea that handles above issues in analyzing algorithms. In Asymptotic Analysis, we evaluate the performance of an algorithm in terms of input size (we don't measure the actual running time). We calculate, how the time (or space) taken by an algorithm increases with the input size.

For example, let us consider the search problem (searching a given item) in a sorted array. One way to search is Linear Search (order of growth is linear) and the other way is Binary Search (order of growth is logarithmic). To understand how Asymptotic Analysis solves the above mentioned problems in analyzing algorithms, let us say we run the Linear Search on a fast computer **A** and Binary Search on a slow computer **B** and we pick the constant values for the two computers so that it tells us exactly how long it takes for the given machine to perform the search in seconds. Let's say the constant for **A** is 0.2 and the constant for **B** is 1000 which means that A is 5000 times more powerful than B. For small values of input array size n, the fast computer may take less time. But, after a certain value of input array size, the Binary Search will definitely start taking less time compared to the Linear Search even though the Binary Search is being run on a slow machine. The reason is the order of growth of Binary Search with respect to input size is logarithmic while the order of growth of Linear Search is linear. So the machine dependent constants can always be ignored after a certain value of input size.

Here are some running times for this example:

Linear Search running time in seconds on A: 0.2 * n

Binary Search running time in seconds on B: 1000*log(n)

n	**Running time on A**	**Running time on B**
10	**2 sec**	**~ 1 h**
100	**20 sec**	**~ 1.8 h**
10^6	**~ 55.5 h**	**~ 5.5 h**
10^9	**~ 6.3 h**	**~ 8.3 h**

M5 Interpret what a trade-off is when specifying an ADT using an example to support your answer

1. **Space-time trade-off:**

   - A **space–time** or **time–memory trade-off** in computer science is a case where an algorithm or program trades increased space usage with decreased time. Here, *space* refers to the data storage consumed in performing a given task (RAM, HDD, etc), and *time* refers to the time consumed in performing a given task (computation time or response time).
   - The utility of a given space–time tradeoff is affected by related fixed and variable costs (of, e.g., CPU speed, storage space), and is subject to diminishing returns.

2. **Example on ADT:**

   Let's return to the

   - ✓ Queue with array.
   - ✓ n: size of array.

   Compare by count the operations of ADT.

Operations steps implementation	**Time Complexity**	**Space Complexity**	
		Auxiliary Space: O(n1)	**Input Space:** O(n2)
`Write()`	`O(1/n)`	n	n+1
`Send()`	`O(1/(n+1))`	n	n+1
`Transfer()`	`O(1/(n+1))`	n	n+1
`Print()`	`O(1/(n^2 + 1))`	n	n^2 + 1
`Total`	`O(1/n^2)`	3n	n^2 + 3n + 4

Table 2 The execute time-space comparison of algorithm on email transferring system

Looking at the above table, we can conclude that we can conclude that if n is growing-up, time complexity will decrease.

References

geeksforgeeks. (2020). *Analysis of Algorithms | Set 1 (Asymptotic Analysis)*. Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/analysis-of-algorithms-set-1-asymptotic-analysis/

Index of comments

2.1 P4: did explain the code but should also add comments to the code
P5: acceptable, good table of test cases
P6: pretty good analyses but should include some graphs/charts...
P7: understood the problem but should take the example from P4
M4: acceptable
M5: understandable explanation