










## ASSIGNMENT 1 FRONT SHEET

<b>Qualification</b>	<b>BTEC Level 5 HND Diploma in Computing</b>		
<b>Unit number and title</b>	Unit 19: Data Structures and Algorithms		
<b>Submission date</b>	07/25/2021	<b>Date Received 1st submission</b>	
<b>Re-submission Date</b>		<b>Date Received 2nd submission</b>	
<b>Student Name</b>	Cao Việt Bách	<b>Student ID</b>	GCH200418
<b>Class</b>	GCH0906	<b>Assessor name</b>	Đỗ Hồng Quân
<b>Student declaration</b> 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.			
		<b>Student's signature</b>	

### Grading grid

P1	P2	P3	M1	M2	M3	D1	D2
							

☐ Summative Feedback:

2.1

☐ Resubmission Feedback:

Grade:

Assessor Signature:

Date:

Internal Verifier's Comments:

IV Signature:

## Table of contents

### **1) Introduction**

### **2) ADT (Abstract data type)**

#### **a) List ADT**

#### **b) Stack ADT**

#### **c) Queue ADT**

### **3) Memory Stack**

### **4) Explanation on how to specify an abstract data type using the example of software stack**

#### **a) Introduction of Algebraic specification.**

#### **b) What is pre-condition, post-condition, error-condition?**

#### **c) Specify Stack's operations using Algebraic specification.**

### **5) Conclusion**

### **Reference**

## Table of Figures

**Figure 1. Abstract data type**

**Figure 2. Example of a List**

**Figure 3. Stack ADT functions**

**Figure 4. Example of a Stack**

**Figure 5. Queue ADT functions**

**Figure 6. A Queue example**

**Figure 7. A Queue example**

**Figure 8. Example of the use of memory stack in function calls**

## 1) Introduction

The writer's manager has assigned him a special role which is to inform his team about designing and implementing abstract data types. He has been asked to create a presentation for all collaborating partners on how ADTs can be utilised to improve software design, development and testing. Further, he has been asked to write an introductory report for distribution to all partners on how to specify abstract data types and algorithms in a formal notation.

## 2) ADT (Abstract data type)

ADT stands for Abstract Data Type, which is a type (or class) for objects whose functionality is specified by a collection of values and operations. The definition of ADT only specifies what operations are to be performed, not how they will be carried out. It makes no mention of how data will be stored in memory or which algorithms will be employed to carry out the actions. It's named "abstract" because it provides an implementation-independent view. The process of providing only the essentials and hiding the details is known as abstraction. (Abstract Data Types, 2019)

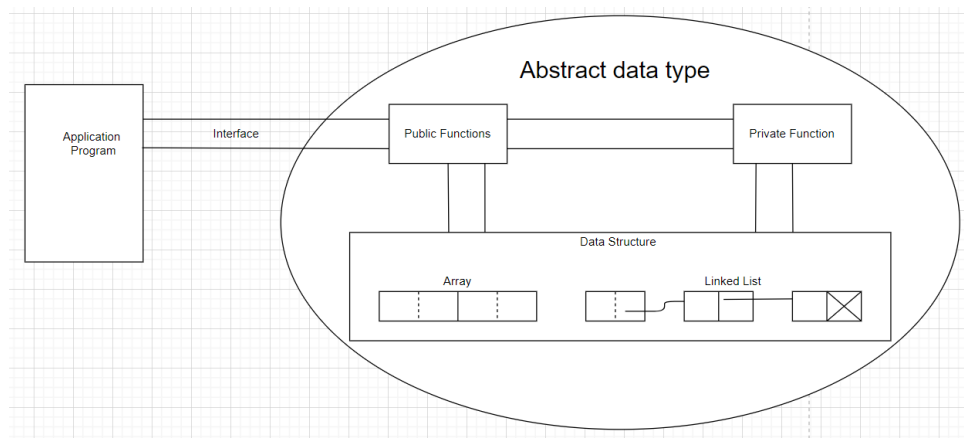


Figure 1. Abstract data type

From Figure 1, An abstract data type (ADT) consists of the following:

- + ) A collection of data
- + ) A set of operations on the data or subsets of the data
- + ) A set of axioms, or rules of behavior governing the interaction of operations

The user of these data types does not need to know how that data types are implemented, for example, people have been using Primitive values like int, float, char data types only with the knowledge that these data types can operate and be performed on without any idea of how they are implemented. So a user only needs to know what these data types can do, but not how they will be implemented. (Abstract Data Types, 2019)

There are many types of abstract data type but List ADT, Stack ADT and Queue ADT are the most common ADTs used in OOP (Object-oriented programming).

### **a) List ADT**

- The data is generally stored in key sequence in a list which has a head structure consisting of count, pointers and address of compare function needed to compare the data in the list. (Abstract Data Types, 2019)
- The data node contains the pointer to a data structure and a self-referential pointer which points to the next node in the list. (Abstract Data Types, 2019)
- + ) get() – Return an element from the list at any given position.
- + ) insert() – Insert an element at any position of the list.
- + ) remove() – Remove the first occurrence of any element from a non-empty list.
- + ) removeAt() – Remove the element at a specified location from a non-empty list.
- + ) replace() – Replace an element at any position by another element.
- + ) size() – Return the number of elements in the list.
- + ) isEmpty() – Return true if the list is empty, otherwise return false.
- + ) isFull() – Return true if the list is full, otherwise return false.
- A List is like a conga line. Everyone holds the hips of the person in front of them and their hips are held in turn by the person to their rear, excepting only those in the front and the back. The only way to add people to the line is to find the right spot and decouple that connection, then insert the new person or people.



Figure 2. Example of a List

### b) Stack ADT

- In Stack ADT Implementation instead of data being stored in each node, the pointer to data is stored. The program allocates memory for the data and the address is passed to the stack ADT. The head node and the data nodes are encapsulated in the ADT. The calling function can only see the pointer to the stack. The stack head structure also contains a pointer to the top and count of the number of entries currently in the stack. (Abstract Data Types, 2019)

- A Stack contains elements of the same type arranged in sequential order. (Abstract Data Types, 2019). All operations take place at a single end that is top of the stack and the following operations can be performed:

- + push() – Insert an element at one end of the stack called top.
- + pop() – Remove and return the element at the top of the stack, if it is not empty.
- + peek() – Return the element at the top of the stack without removing it, if the stack is not empty.
- + size() – Return the number of elements in the stack.
- + isEmpty() – Return true if the stack is empty, otherwise return false.
- + isFull() – Return true if the stack is full, otherwise return false.

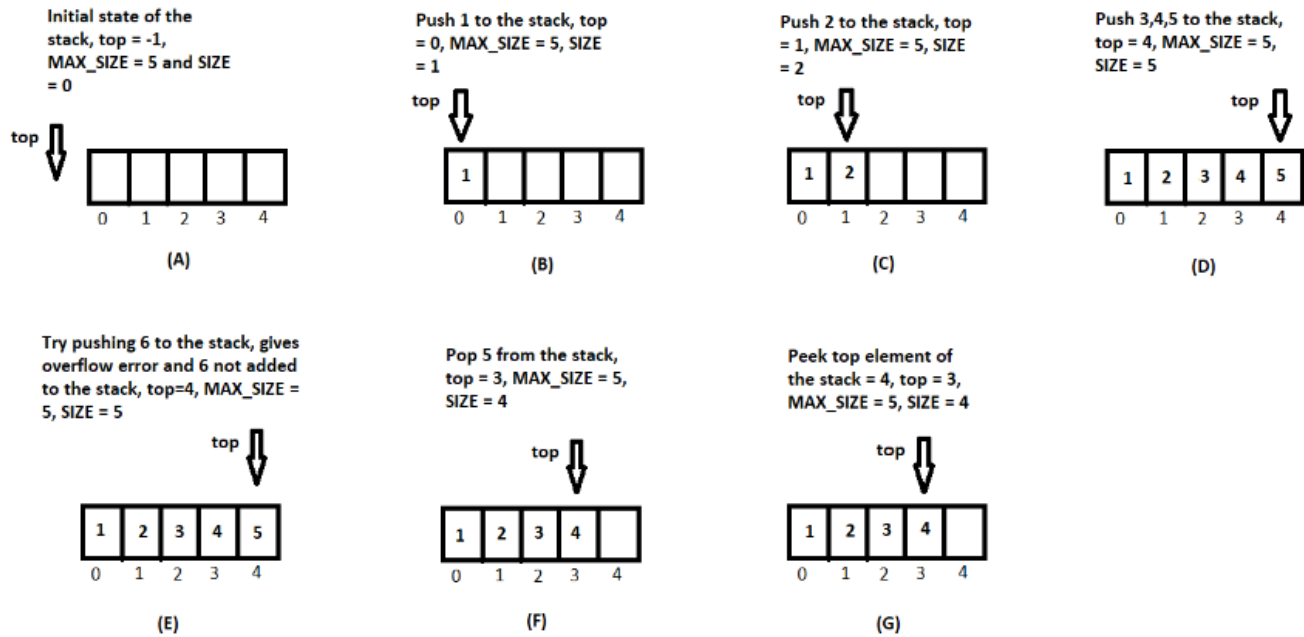


Figure 3. Stack ADT functions

- In Figure 4, the writer provides some real-life examples of a Stack: a list of envelopes and a ball containers



Figure 4. Example of a Stack

- The first image figure 3 shows a container loaded with balls. We will always strive to take the last inserted ball, so it is a last in first out situation. The second image depicts a list of mail envelopes, with most people selecting the topmost envelope, followed by the second top, and so on.

### c) Queue ADT

- The queue abstract data type (ADT) follows the basic design of the stack abstract data type. Each node contains a void pointer to the data and the link pointer to the next element in the queue. The program's responsibility is to allocate memory for storing the data. (Abstract Data Types, 2019)



- According to “Abstract Data Types”, a Queue contains elements of the same type arranged in sequential order. Operations take place at both ends, insertion is done at the end and deletion is done at the front. (Abstract Data Types, 2019). Following operations can be performed:

- +) enqueue() – Insert an element at the end of the queue.
- +) dequeue() – Remove and return the first element of the queue, if the queue is not empty.
- +) peek() – Return the element of the queue without removing it, if the queue is not empty.
- +) size() – Return the number of elements in the queue.
- +) isEmpty() – Return true if the queue is empty, otherwise return false.
- +) isFull() – Return true if the queue is full, otherwise return false.

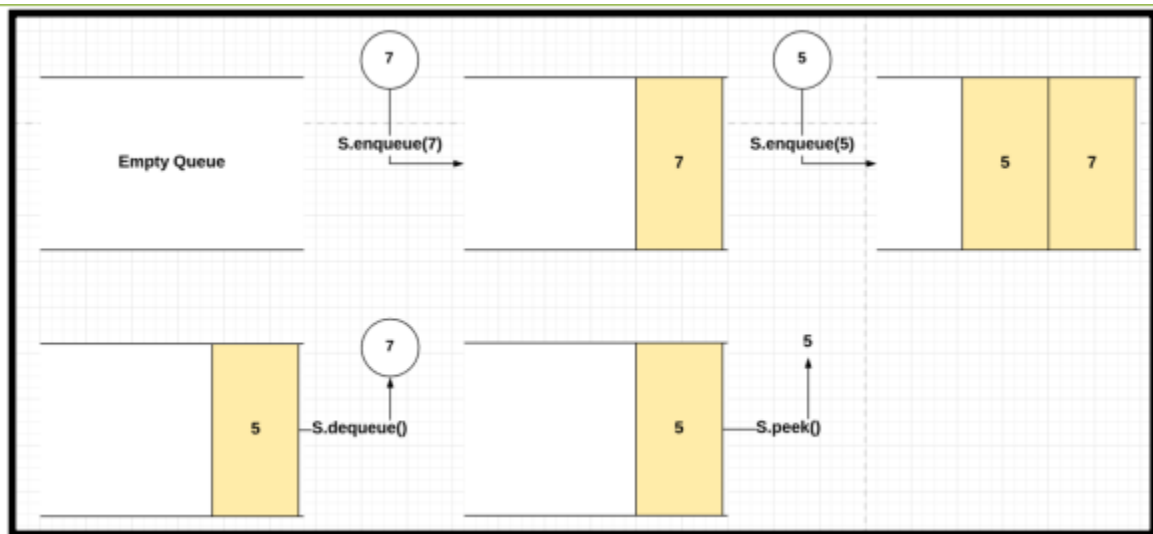


Figure 5. Queue ADT functions

- In Figure 6 and Figure 7, the writer provides some real-life examples of a Queue: a queue of car that are waiting for washing:



Figure 6. A Queue example



Figure 7. A Queue example

- A new car will join the line from the end and the car standing at the front will be the first to be washed and leave the line. Similarly in a queue data structure, data added first will leave the queue first.

### 3) Memory Stack

According to “Stack vs Heap Memory Allocation”, memory in a C/C++/Java program can either be allocated on a stack or a heap. A stack is a special area of a computer's memory that stores temporary variables created by a function. In stack, variables are declared, stored, and initialized during runtime. It is a temporary storage memory. When the computing task is complete, the memory of the variable will be automatically erased. (Stack vs Heap: Know the Difference, 2021)

The example below shows how memory stack is used to implement function calls in a computer:

```
int fcn1 (int int1) {
    int x;
    x = fcn2(int1 * 2); // function call
    return (x - 3);
}
int fcn2 (int int2) {
    int y = int2 - 85;
    return y;
}
int main() {
    int z;
    z = fcn1 (10); // function call
    printf("%d\n",z);
    return 0;
}
```

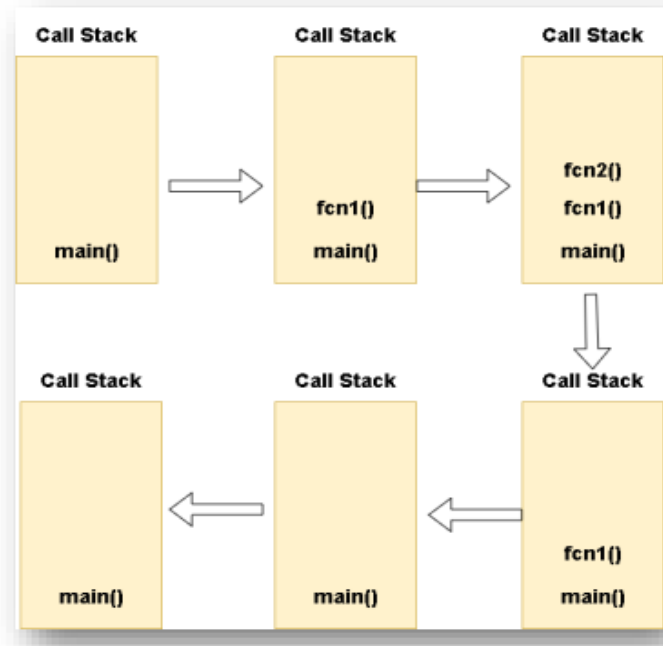


Figure 8. Example of the use of memory stack in function calls

- Each time a function calls another function (function `main()` calls function `fcn1()`, function `fcn1()` calls function `fcn2()`), a stack frame (also known as an activation record) is pushed onto the stack. These functions maintain the return address that the called function needs to return to the calling function and also contain automatic variables—parameters and any local variables the function declares.
- When the called function returns, the computer gives the writer the result and deletes the variables in its stack (the computer memory).
- Control transfers to the return address in the popped stack frame

#### 4) Explanation on how to specify an abstract data type using the example of software stack

##### a) Introduction of Algebraic specification.

Algebraic specification is a formal specification language that first appeared in the mid-1970s as a method for dealing with data structures in an implementation-independent manner. The approach was based on specifying data types in a manner similar to that used in modern algebra to study various mathematical structures (e.g., groups, rings, fields, etc.). It describes functions in the form of an algebra. An algebra consists of a signature and the axioms. The signature consists of a family of object sets (carrier sets) and a number of operations in this carrier set with their functionality. The axioms describe the characteristics of this operations by means of algebraic equations. (Ehrig et al,1992)

##### b) What is pre-condition, post-condition, error-condition?

A pre-condition is a condition, also known as a predicate, that must be true before a method can be executed. In other words, the method informs clients that "this is what I expect from you." So, the method we're calling expects something to be in place prior to or at the time the method is called. Unless the precondition is met, the operation is not guaranteed to work properly. (Preconditions and Postconditions, 2018)

A post-condition is a condition, or predicate, that can be guaranteed after a method is completed. In other words, the method tells clients, "This is what I promise to do for you." If the operation is correct and the precondition(s) are met, the postcondition is guaranteed to be true. (Preconditions and Postconditions, 2018)

The error-condition is the implicit action for many conditions. This provides a common condition that can be used to check for a number of different conditions, rather than checking each condition separately. (ERROR condition, 2021)

### **c) Specify Stack's operations using Algebraic specification.**

**Type** Stack

**Operations**

\* NewStack:

Pre: True

Post:

+) IsEmpty(NewStack) ----> True

+) IsFull(NewStack) ----> False

+) ----> Stack

\* Push:

Pre: True

Post:

+) Top(Stack) ----> element

+) IsEmpty(Stack) ----> False

+) Stack, element ----> Stack

\* Pop:

Pre:

+) IsEmpty(Stack) ----> False

Post:

+) Stack ----> Stack

\* Top:

Pre:

+) IsEmpty(Stack) ----> False

Post:

+) Stack ----> element

\* IsEmpty:

Pre: True

Post:

+) Stack ----> Boolean

\* IsFull:

Pre: True

Post:

+) Stack ----> Boolean

**Axioms (Error-condition)**

- + ) Pop(NewStack): ----> Error
- + ) Top(NewStack): ----> Error
- + ) Pop(Push(Stack,element)) ----> Stack
- + ) Top(Push(Stack,element)) ----> element
- + ) IsEmpty(Push(Stack,element)) ----> False
- + ) Stack ----> IsFull(Pop(Stack))

**Note:** The “pre-condition: true” indicates that the operation is valid for any stack state: There are no pre-conditions.

**5) Conclusion**

In this assignment, the writer has created a design specification for data structures explaining the valid operations that can be carried out on the structures. Furthermore, he also determined the operations of a memory stack and how it is used to implement function calls in a computer and using an imperative definition to specify the abstract data type for a software stack.

## References

GeeksforGeeks. 2019. Abstract Data Types. [online] Available at: <  
<https://www.geeksforgeeks.org/abstract-data-types/> > [Accessed 11 June 2022].

Guru99. 2021. Stack vs Heap: Know the Difference. [online] Available at: <  
<https://www.guru99.com/stack-vs-heap.html> > [Accessed 11 June 2022].

Ehrig, H., Mahr, B., Classen, I. and Orejas, F., 1992. Introduction to Algebraic Specification. Part 1: Formal Methods for Software Development. The Computer Journal, [online] 35(5), pp.460-467. Available at: <  
<https://academic.oup.com/comjnl/article-pdf/35/5/460/1125061/35-5-460.pdf> > [Accessed 11 June 2022].

Medium. 2018. Preconditions and Postconditions. [online] Available at: <  
<https://medium.com/@mlbors/preconditions-and-postconditions-5913fc0fcda4> > [Accessed 11 June 2022].

IBM. 2021. ERROR condition. [online] Available at: <  
<https://www.ibm.com/docs/en/epfz/5.3?topic=conditions-error-condition> > [Accessed 11 June 2022].

GeeksforGeeks. 2021. Stack vs Heap Memory Allocation. [online] Available at: <  
<https://www.geeksforgeeks.org/stack-vs-heap-memory-allocation/> > [Accessed 11 June 2022].

# Index of comments

---

- 2.1
- P1: Main operations of Stack and Queue ADT have been introduced. It expects more examples and figures to support your discussion.
  - P2: The work has provided an example as basis to explain the use of memory stack in function calls.
  - P3: The specification for software Stack needs improvements in which the design of its operations must consider the correct number of parameters and the return types.