

**Higher Nationals in Computing**

**UNIT 19**

**DATA STRUCTURES AND ALGORITHMS**

ASSIGNMENT

No.1

Learner’s name: Nguyễn Xuân Trường

Assessor name: Nam Lam

Class: GCS0901

Learner’s ID: GCS17415

Subject’s ID:1649

Assignment due:Assignment submitted:

**ASSIGNMENT 1 FRONT SHEET**

|  |  |  |  |
| --- | --- | --- | --- |
| **Qualification** | **BTEC Level 5 HND Diploma in Computing** | | |
| **Unit number and title** | Unit 19: Data Structures and Algorithms | | |
| **Submission date** |  | **Date Received 1st submission** |  |
| **Re-submission Date** |  | **Date Received 2nd submission** |  |
| **Student Name** | Nguyễn Xuân Trường | **Student ID** | GCS17415 |
| **Class** | GCS0901 | **Assessor name** | Nam Lam |
| **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** | Truong |

**Grading grid**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **P1** | **P2** | **P3** | **M1** | **M2** | **M3** | **D1** | **D2** |
|  |  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **❒ Summative Feedback: ❒ Resubmission Feedback:** | | |
| **Grade:** | **Assessor Signature:** | **Date:** |
| **Internal Verifier’s Comments:** | | |
| **IV Signature:** | | |

**Assignment Brief 1 (RQF)**

**Higher National Certificate/Diploma in Business**

|  |  |
| --- | --- |
| **Student Name/ID Number:** |  |
| **Unit Number and Title:** | Unit 19: Data Structures and Algorithms |
| **Academic Year:** | **2021** |
| **Unit Assessor:** |  |
| **Assignment Title:** | Examine and specify ADT and DSA |
| **Issue Date:** |  |
| **Submission Date:** |  |
| **Internal Verifier Name:** |  |
| **Date:** |  |

|  |
| --- |
| **Submission Format:** |
| *Format:*   * The submission is in the form of an individual written report and a presentation. This should be written in a concise, formal business style using single spacing and font size 12. You are required to make use of headings, paragraphs and subsections as appropriate, and all work must be supported with research and referenced using the Harvard referencing system. Please also provide a bibliography using the Harvard referencing system.   *Submission*   * Students are compulsory to submit the assignment in due date and in a way requested by the Tutor. * The form of submission will be a soft copy posted on <http://cms.greenwich.edu.vn/>. * Remember to convert the word file into PDF file before the submission on CMS.   *Note:*   * The individual Assignment *must* be your own work, and not copied by or from another student. * If you use ideas, quotes or data (such as diagrams) from books, journals or other sources, you must reference your sources, using the Harvard style. * Make sure that you understand and follow the guidelines to avoid plagiarism. Failure to comply this requirement will result in a failed assignment. |
| **Unit Learning Outcomes:** |
| **LO1** Examine abstract data types, concrete data structures and algorithms  **LO2** Specify abstract data types and algorithms in a formal notation |
| **Assignment Brief and Guidance:** |
| **Assignment scenario**  You work as in-house software developer for Softnet Development Ltd, a software body-shop providing network provisioning solutions. Your company is part of a collaborative service provisioning development project and your company has won the contract to design and develop a middleware solution that will interface at the front-end to multiple computer provisioning interfaces including SOAP, HTTP, JML and CLI, and the back-end telecom provisioning network via CLI .  Your account manager has assigned you a special role that is to inform your team about designing and implementing abstract data types. You have been asked to create a presentation for all collaborating partners on how ADTs can be utilised to improve software design, development and testing. Further, you have 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.  **Tasks**  **Part 1**  You will need to prepare a presentation on how to create a design specification for data structures, explaining the valid operations that can be carried out on the structures using the example of:   1. A stack ADT, a concrete data structure for a First In First out (FIFO) queue. 2. Two sorting algorithms. 3. Two algorithms (of your choice) on Tree data structure.   **Part 2**  You will need to provide a formal written report that includes the following:   1. Explanation on how to specify an abstract data type using the example of software stack. 2. Explanation of the advantages of encapsulation and information hiding when using an ADT. 3. Discussion of imperative ADTs with regard to object orientation. |

|  |  |  |
| --- | --- | --- |
| **Learning Outcomes and Assessment Criteria (Assignment 1)** | | |
| **Pass** | **Merit** | **Distinction** |
| **LO1** Examine abstract data types, concrete data structures and algorithms | | **D1** Analyse the operation, using illustrations, of two network shortest path algorithms, providing an example of each. |
| **P1** Create a design specification for data structures explaining the valid operations that can be carried out on the structures.  **P2** Determine the operations of a memory stack and how it is used to implement function calls in a computer. | **M1** Illustrate, with an example, a concrete data structure for a First In First out (FIFO) queue.  **M2** Compare the performance of two sorting algorithms. |
| **LO2** Specify abstract data types and algorithms in a formal notation | | **D2** Discuss the view that imperative ADTs are a basis for object orientation and, with justification, state whether you agree. |
| **P3** Using an imperative definition, specify the abstract data type for a call stack. | **M3** Examine the advantages of encapsulation and information hiding when using an ADT. |

Table of Contents

[Table of Figures 1](#_Toc90544522)

[**1.** **DATA STRUCTURES** 2](#_Toc90544523)

[**1.1.** **ABSTRACT DATA TYPE (P1)** 2](#_Toc90544524)

[**1.1.1.** **Definition** 2](#_Toc90544525)

[**1.1.2.** **Examples** 2](#_Toc90544526)

[**1.2.** **ADT USAGES** 7](#_Toc90544527)

[**1.2.1.** **Application of Stack in Memory (P2)** 7](#_Toc90544528)

[**1.2.2.** **Application of an ADT (P3)** 9](#_Toc90544529)

[REFERENCES 12](#_Toc90544530)

Table of Figures

[Figure 1 Abstract Data Type Definition 2](#_Toc90544516)

[Figure 2 List ADT definition 3](#_Toc90544517)

[Figure 3 List ADT Example 4](#_Toc90544518)

[Figure 4 Stack ADT Example 5](#_Toc90544519)

[Figure 5 Queue ADT definition 6](#_Toc90544520)

[Figure 6 Queue ADT example 7](#_Toc90544521)

**ASSIGNMENT 1’S ANSWERS**

1. **DATA STRUCTURES**
   1. **ABSTRACT DATA TYPE (P1)**
      1. **Definition**

Abstract Data type (ADT) is a type (or class) for objects whose behaviour is defined by a set of value and a set of operations.

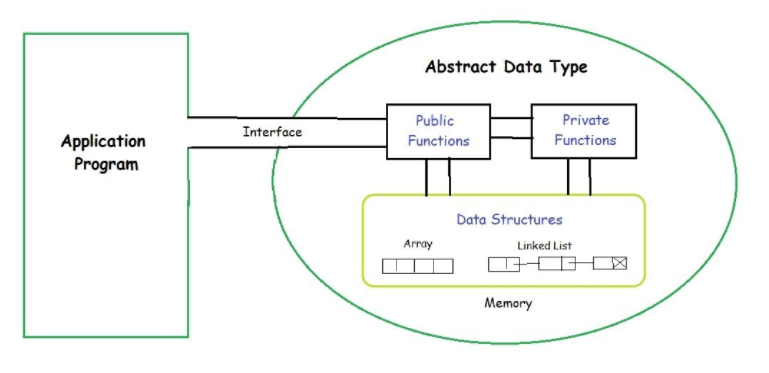


Figure Abstract Data Type Definition

The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called “abstract” because it gives an implementation-independent view. The process of providing only the essentials and hiding the details is known as abstraction.

The user of data type does not need to know how that data type is implemented, for example, we have been using Primitive values like int, float, char data types only with the knowledge that these data type can operate and be performed on without any idea of how they are implemented. So, a user only needs to know what a data type can do, but not how it will be implemented. ADT is which hides the inner structure and design of the data type. Now we’ll define three ADTs namely List ADT, Stack ADT, Queue ADT.

* + 1. **Examples**

**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.

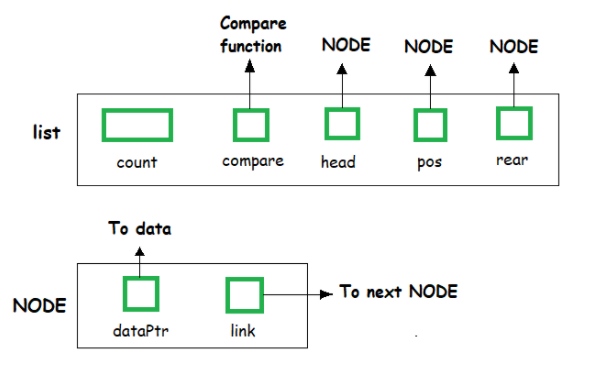


Figure List ADT definition

The data node contains the pointer to a data structure and a self-referential pointer which points to the next node in the list.

//List ADT Type Definitions

typedef struct node

{

void \*DataPtr;

struct node \*link;

} Node;

typedef struct

{

int count;

Node \*pos;

Node \*head;

Node \*rear;

int (\*compare) (void \*argument1, void \*argument2)

} LIST;

The List ADT Functions is given below:

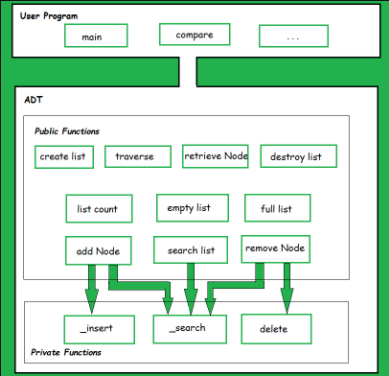


Figure List ADT Example

A list contains elements of the same type arranged in sequential order and following operations can be performed on the list.

* 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.

**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 address is passed to the stack ADT.

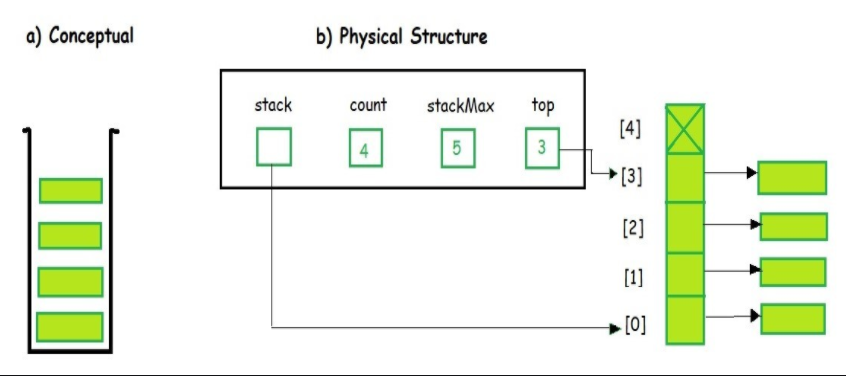


Figure Stack ADT Example

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 top and count of number of entries currently in stack.

//Stack ADT Type Definitions

typedef struct node

{

void \*DataPtr;

struct node \*link;

} StackNode;

typedef struct

{

int count;

StackNode \*top;

} STACK;

A Stack contains elements of the same type arranged in sequential order. All operations take place at a single end that is top of the stack and 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.

**Queue ADT**

The queue abstract data type (ADT) follows the basic design of the stack abstract data type.

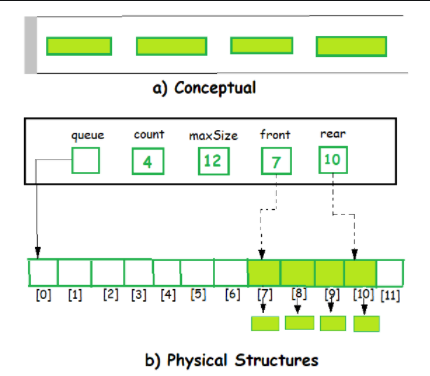


Figure Queue ADT definition

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.

//Queue ADT Type Definitions

typedef struct node

{

void \*DataPtr;

struct node \*next;

} QueueNode;

typedef struct

{

QueueNode \*front;

QueueNode \*rear;

int count;

} QUEUE;

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. Following operations can be performed:

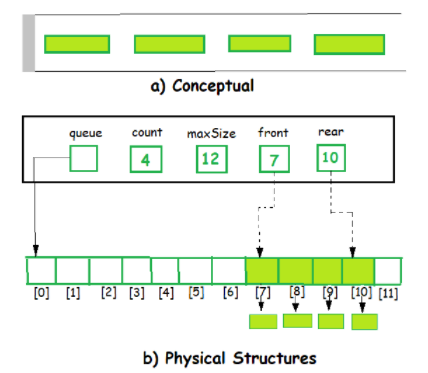


Figure Queue ADT example

* 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.
  1. **ADT USAGES**
     1. **Application of Stack in Memory (P2)**

**The stack ADT and its applications**

A stack is an ordered list of elements in which elements are always inserted and deleted at one end, say the beginning. In the terminology of stacks, this end is called the top of the stack, whereas the other end is called the bottom of the stack. Also, the insertion operation is called push and the deletion operation is called pop. The element at the top of a stack is frequently referred, so we highlight this special form ofgetElement. A stack ADT can be specified by the following basic operations. Once again we assume that we are maintaining a stack of characters. In practice, the data type for each element of a stack can be of any data type. Characters are chosen as place-holders for simplicity.

S = init();

*Initialize S to an empty stack.*

isEmpty(S);

*Returns "true" if and only if the stack S is empty, i.e., contains no elements.*

isFull(S);

*//Returns "true" if and only if the stack S has a bounded size and holds the maximum number of elements it can. top(S);*

*Return the element at the top of the stack S, or error if the stack is empty.*

S = push(S,ch);

*Push the character ch at the top of the stack S.*

S = pop(S);

*Pop an element from the top of the stack S. print(S);*

Print the elements of the stack S from top to bottom.

An element popped out of the stack is always the last element to have been pushed in. Therefore, a stack is often called a Last-In-First-Out or a LIFO list.

**Applications of stacks**

Stacks are used in a variety of applications. While some of these applications are "natural", most other are essentially "pedantic". Here is a list anyway.

* For processing nes ted structures, like checking for balanced parentheses, evaluation of postfix expressions.
* For handling function calls and, in particular, recursion.
* For searching in special data structures (depth -first search in graphs and trees), for example, for implementing backtracking.

**Implementations of the stack ADT**

A stack is specified by the ordered collection representing the content of the stack together with the choice of the end of the collection to be treated as the top. The top should be so chosen that pushing and popping can be made as far efficient as possible.

* Using static arrays

Static arrays can realize stacks of a maximum possible size. If we assume that the stack elements are stored in the array starting from the index 0, it is convenient to take the top as the maximum index of an element in the array. Of course, the other choice, i.e., the other boundary 0, can in principle be treated as the top, but insertions and deletions at the location 0 call for too many relocations of array elements. So our original choice is definitely better.

* Using dynamic linked lists

As we have seen earlier, it is no big deal to create and maintain a dynamic list of elements. The only consideration now is to decide whether the beginning or the end of the list is to be treated as the top of the stack. Deletion becomes costly, if we choose the end of the list as the top. Choosing the beginning as the top makes the implementations of both push and pop easy. So we stick to this convention. As usual, we maintain a dummy node at the top (beginning) for simplifying certain operations.

* + 1. **Application of an ADT (P3)**

There is an example of an application of an ADT to make a music player with the support of ADT. A queue is an abstract data type. If you think about adding music to a playlist, you're adding music to the end of a queue. Songs at the front of the queue are played first then removed. Songs at the back of the queue are played last -- First In First Out (FIFO) or First Come, First Served.

Think about how you might use the list abstract data type to model a list of songs in an album. Or a tree abstract data type to model a hierarchy of song files on disk, in folders by artist and subfolders by albums. Challenge - think about examples of when you might use a stack -- First In, Last Out.

**function** playpauseTrack() {

  // Switch between playing and pausing

  // depending on the current state

**if** (!isPlaying) playTrack();

**else** pauseTrack();

}

**function** playTrack() {

  // Play the loaded track

  curr\_track.play();

  isPlaying = **true**;

  // Replace icon with the pause icon

  playpause\_btn.innerHTML = '<i class="fa fa-pause-circle fa-5x"></i>';

}

**function** pauseTrack() {

  // Pause the loaded track

  curr\_track.pause();

  isPlaying = **false**;

  // Replace icon with the play icon

  playpause\_btn.innerHTML = '<i class="fa fa-play-circle fa-5x"></i>';

}

**function** nextTrack() {

  // Go back to the first track if the

  // current one is the last in the track list

**if** (track\_index < track\_list.length - 1)

    track\_index += 1;

**else** track\_index = 0;

  // Load and play the new track

  loadTrack(track\_index);

  playTrack();

}

**function** prevTrack() {

  // Go back to the last track if the

  // current one is the first in the track list

**if** (track\_index > 0)

    track\_index -= 1;

**else** track\_index = track\_list.length - 1;

  // Load and play the new track

  loadTrack(track\_index);

  playTrack();

}

List ADT is used mostly in this application for adding new object to list or remove, then the Queue will be used for playing or stopping the music. Stack is not useless, it is used for adding the new song to list stack by stack.

**function** loadTrack(track\_index) {

  clearInterval(updateTimer);

  resetValues();

  curr\_track.src = track\_list[track\_index].path;

  curr\_track.load();

  track\_art.style.backgroundImage =

     "url(" + track\_list[track\_index].image + ")";

  track\_name.textContent = track\_list[track\_index].name;

  track\_artist.textContent = track\_list[track\_index].artist;

  now\_playing.textContent =

     "PLAYING " + (track\_index + 1) + " OF " + track\_list.length;

  updateTimer = setInterval(seekUpdate, 1000);

  curr\_track.addEventListener("ended", nextTrack);

}

// Function to reset all values to their default

**function** resetValues() {

  curr\_time.textContent = "00:00";

  total\_duration.textContent = "00:00";

  seek\_slider.value = 0;

}

**Reset all the values of the previous track**

A resetValues() function is created which handles the resetting of the duration value and the slider to their initial values before a new track starts. This prevents the jumping of the seek slider while the new track loads.

**Loading the track**

The audio element is assigned a new source using its src property. It may be given any path from the filesystem or a URL. The load() method is then used on the audio element to get the track ready.

**Updating the track art to be shown**

The track art is fetched from the array and assigned with the help of the

# REFERENCES

Geeksforgeeks (2019) Abstract Data Types [online] Available at: <https://www.geeksforgeeks.org/abstract-data-types/>

Wikipedia (2012) Call Stack [online] Available at: <https://en.wikipedia.org/wiki/Call_stack>

Katie, M () Stacks in Computer Memory: Definition & Uses [online] Available at: <https://study.com/academy/lesson/stacks-in-computer-memory-definition-uses.html>

Brainkart () Stack ADT and its applications, Applications, Implementations [online] Available at: <https://www.brainkart.com/article/Stack-ADT-and-its-applications,-Applications,-Implementations_10074/>

Geeksforgeeks () Create a Music Player using JavaScript [online] Available at: <https://www.geeksforgeeks.org/create-a-music-player-using-javascript/>

Ben, S (2016) What are some ways or examples of how abstract data types are used in software? [online] Available at: <https://www.quora.com/What-are-some-ways-or-examples-of-how-abstract-data-types-are-used-in-software>

C. Thomas Wu (2010) An Introduction to Object-Oriented Programming with Java fifth edition [ebook] Available at: <http://2.droppdf.com/files/ySgol/an-introduction-to-object-oriented-programming-with-java-5th-edition.pdf>

Adam, D (2004) Data Structures and Algorithms in Java, 2nd [ebook] Available at: <https://slidelegend.com/data-structures-and-algorithms-in-java-2nd-ed-college-of-_59bd5a2d1723dd54e8da40db.html>

Luca, D, T and Granville, B (2008) Data Structures and Algorithms: Annotated Reference with Examples, 1st edition [ebook] Available at: <https://www.academia.edu/30843807/DSA_Data_Structures_and_Algorithms_Annotated_Reference_with_Examples>

Arnab, C (2019) Abstract Data Type in Data Structures [online] Available at: <https://www.tutorialspoint.com/abstract-data-type-in-data-structures>

guru99 () Stack vs Heap: Know the Difference [online] Available at: <https://www.guru99.com/stack-vs-heap.html#:~:text=A%20stack%20is%20a%20special,is%20a%20temporary%20storage%20memory.&text=The%20stack%20section%20mostly%20contains,local%20variable%2C%20and%20reference%20variables>.

cseworldonline () Overview of Data Structures [online] Available at: <https://www.cseworldonline.com/data-structure/Introduction-Data-Structures.php#:~:text=The%20possible%20operations%20on%20the,performed%20at%20one%20end%20only>.

geeksforgeeks (2021) Types of Exception in Java with Examples [online] Available at: <https://www.geeksforgeeks.org/types-of-exception-in-java-with-examples/>

Margie, S () DEFINITION software stack [online] Available at: <https://searchapparchitecture.techtarget.com/definition/software-stack>