**THE INSTITUTE OF FINANCE MANAGEMENT**



**FACULTY OF COMPUTING MATHEMATICS - FCM**

**DEPARTMENT OF COMPUTER SCIENCE**

**BACHELOR DEGREE IN COMPUTER SCIENCE**

**CSU 07317:DATA STRUCTURE AND ALGORITHMS - ASSIGNMENT**

|  |  |  |
| --- | --- | --- |
| **S/N** | **NAME** | **REGISTRATION NUMBER** |
| **1.** | **CHRISTINA S KISAMBU** | **IMC/BCS/2216198** |
| **2.** | **RODGERS J SWAI** | **IMC/BCS/2226482** |
| **3.** | **HABIBA A DAUDI** | **IMC/BCS/2211330** |
| **4.** | **MWESIGA G MTEMBEI** | **IMC/BCS/2227037** |
| **5.** | **SARAH J MKISI** | **IMC/BCS/2214387** |
| **6.** | **SAUDA A KASSIM** | **IMC/BCS/2210043** |
| **7.** | **SHAMSA A BELEKO** | **IMC/BCS/2216303** |
| **8.** | **KELVIN S MGUHI** | **IMC/BCS/2222452** |
| **9.** | **ASHIRI J MWANGIA** | **IMC/BCS/2226943** |
| **10.** | **NELSON L HENDRY** | **IMC/BCS/2222246** |

**YEAR ONE 2022/2023**

Questions

1. Describe contiguous data representation
2. Describe linked data representation
3. By using the programming languages, you are familiar with create data types using the knowledge above.

1.Describe contiguous data representation.

**Contiguous data** representation refers to the organization of data in a sequential and continuous manner in computer memory. The elements of data are stored one after another in a linear way, without any fragmentation, though this type of representation has implications for data structures and memory management.

Example of contiguous data structure are: -

* **Array**. In an array, the elements are stored in sequential memory locations. When you allocate an array, memory is reserved for all the elements in a continuous block, allowing for easy and efficient access to the elements.
* **Matrices**: Matrices can be represented as contiguous data structures. A 2-dimensional array can be used to store matrices in a row-major or column-major order, where each row or column is stored contiguously.
* **Strings**: In many programming languages, strings are implemented as contiguous arrays of characters. This enables efficient indexing, concatenation, and substring operations.

There some aspects of contiguous data representation such as

* **Sequential Storage:** Contiguous data representation involves storing elements of data in a sequential order. Each element is stored immediately after the preceding one in memory.
* **Direct Access to Elements.** Because elements are stored sequentially, direct access to any element in the data structure can be achieved through indexing. The memory address of an element can be calculated based on its position and the size of each element.
* **Memory Efficiency.** Contiguous data representation is often memory-efficient. It allows for efficient use of memory because there is no need for extra space between elements or additional data structures to manage their relationships.
* **Cache Friendliness:** Contiguous data representation can be cache-friendly, particularly in the context of arrays. Accessing consecutive elements in memory takes advantage of spatial locality, improving cache performance.
* **Array Size Determination:** The size of a contiguous data structure, such as an array, is typically fixed at the time of creation. The total size is determined by multiplying the size of each element by the number of elements.

Qn2. Describe Linked data representation.

Linked data representation refers to the organization and storing of data in a way that each element or node contains a reference or link to the next element in the sequence. This allows dynamic memory allocation and flexibility. Linked data structure use this data representation to overcome some of the limitations such as fixed size and expensive insertion and deletion.

The following are some of aspects of linked data representation;

* **Nodes.** The basic building blocks of linked data structure where by each node contains data and a reference to the next node in the sequence.
* **Pointers or References.** The connections between the nodes are established using pointers. The pointers indicate the memory location of the next node in the sequence.
* **Dynamic memory Allocation** Each element can be removed or added without the need to reallocate a fixed amount of memory.
* **Memory Requirement** Linked list do not require elements to be stored in contiguous memory locations, nodes can be scattered throughout the memory.
* **Ease of Insertion and deletion** Insertion or deletion in linked list is more straightforward than in contiguous structures. To insert or delete an element, you only need to adjust the references in the neighboring nodes.

The following are some examples of linked data representation.

* **Singly Linked list:** In a singly linked list, each node contains data and a reference (link) to the next node in the sequence. The last node typically points to null.

Node 1 -> Node 2 -> Node 3 -> ... -> Node N -> null

* **Doubly Linked list:** In a doubly linked list, each node has references to both the next and the previous nodes, allowing for traversal in both directions.

null <- Node 1 <-> Node 2 <-> Node 3 <-> ... <-> Node N -> null

* **Circular Linked List:** IN circular linked list the last node is connected to the first, forming a loop. Node 1 -> Node 2 -> Node 3 -> ... -> Node N -> Node 1
* **Linked List as a Stack:** A linked list can be used to implement a stack, where elements are added or removed from one end (the top).

3.By using the programming language you are familiar with create data types by using the knowledge above.

* Example of contiguous data type by using array (C programming language)

#include <stdio.h>

int main(){

int LA[3] = {}, i;

printf("Array Before Insertion:\n");

for(i = 0; i < 3; i++)

printf("LA[%d] = %d \n", i, LA[i]);

printf("Inserting Elements \n");

printf("The array elements after insertion :\n"); // prints array values

for(i = 0; i < 3; i++) {

LA[i] = i + 2;

printf("LA[%d] = %d \n", i, LA[i]);

}

return 0;

}

* Example of linked data type using linked list (C programming language)

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

int main() {

struct Node\* head = NULL;

struct Node\* second = NULL;

struct Node\* third = NULL;

struct Node\* fourth = NULL;

head = (struct Node\*)malloc(sizeof(struct Node));

second = (struct Node\*)malloc(sizeof(struct Node));

third = (struct Node\*)malloc(sizeof(struct Node));

fourth = (struct Node\*)malloc(sizeof(struct Node));

fourth->data = 7;

fourth->next =NULL ;

head->data = 3;

head->next = second;

second->data = 5;

second->next = third;

third->data = 1;

third->next = fourth;

struct Node\* temp = head;

printf("Linked list elements: ");

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->next;

}

return 0;

}