Republic of Rwanda
City of Kigali

GASABO DISTRICT

DISTRICT COMPREHENSIVE ASSESSMENT, RTQF LEVEL… 2023-2024

SECTOR: ICT AND MULTIMEDIA

TRADE: SOFTWARE DEVELOPMENT

MODULE CODE: SWDDA401

MODULE NAME: DATA STRUCTURE AND ALGORITHM FUNDAMENTALS BY USING JS

DATE OF EXAM: 12/03/2024

DURATION: 3 HOURS

SCHOOL YEAR: 2023-2024

TERM: 2

MAKING GUIDE

1. Define the following terms: 6marks

a. Algorithm

b. Flowchart

c. Leaf

d. Edges

e. Array

Answer:

a. Algorithm:

An algorithm is a set of **step-by-step instructions** that define a clear and finite procedure for solving a problem or accomplishing a specific task. It provides an organized and efficient way to process data, perform calculations, and arrive at a desired outcome.

b. Flowchart:

A flowchart is a visual representation of an algorithm that uses **geometric shapes** and **connecting arrows** to illustrate the flow of logic and decision-making involved in the process. It provides a clear and concise way to communicate the sequence of steps and choices within an algorithm.

c. Leaf:

In the context of trees (data structures), a leaf is a **node** with **no children**. It represents the end of a branch and has no further connections to other nodes in the tree.

d. Edges:

In the context of graphs (data structures), an edge is a **connection** between two **vertices** (nodes) in the graph. It represents a relationship or link between these nodes.

e. Array:

An array is a fundamental **data structure** that stores a collection of **elements** of the same **data type** under a single variable name. Each element in an array has a unique **index** (position) that allows for efficient access, insertion, and deletion of elements within the collection.

2. Convert 1101.101_2 into decimal system.

Answer**:** The binary number 1101.1012 can be converted to its decimal equivalent by separately converting the integer and fractional parts:

Integer part (1101):

1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 8 + 4 + 1 = 13

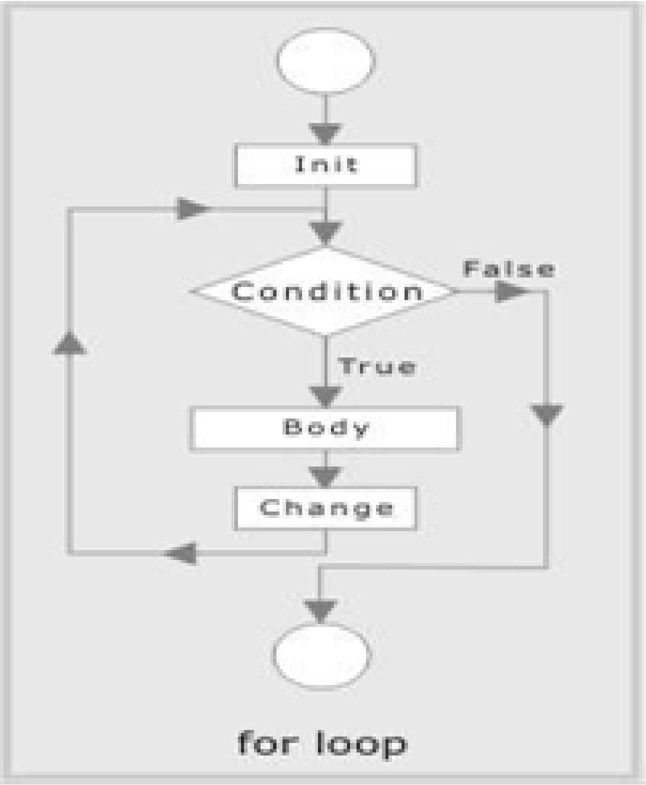
Fractional part (101):

1 * 2^-1 + 0 * 2^-2 + 1 * 2^-3 = 0.5 + 0 + 0.125 = 0.625

Combining both parts:

Decimal equivalent = Integer part + Fractional part = 13 + 0.625 = **13.75**

3. **Draw the flowchart to show the use of for loop.**



4. Differentiate while loop from do while loop **4Marks**

The primary difference between a while loop and a do-while loop lies in when the loop condition is checked.

While Loop:

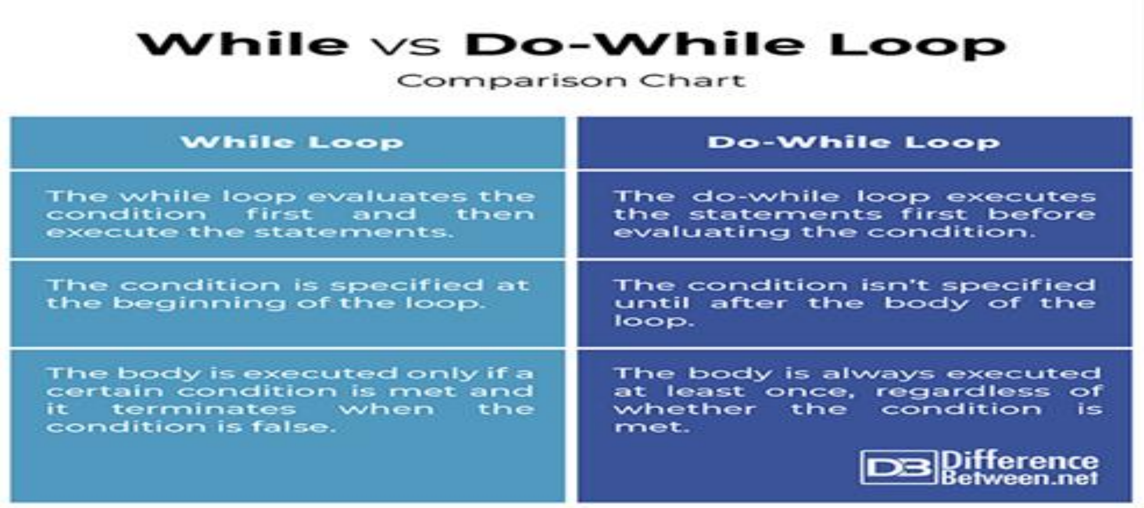
In a while loop, the condition is checked before the execution of the loop body.

If the condition is initially false, the loop body is never executed.

Do-While Loop:

In a do-while loop, the condition is checked after the execution of the loop body.

This guarantees that the loop body will be executed at least once, regardless of the initial condition.



While vs Do-While Loop

Comparison Chart

While Loop	Do-While Loop
The while loop evaluates the condition first and then execute the statements.	The do-while loop executes the statements first before evaluating the condition.
The condition is specified at the beginning of the loop.	The condition isn't specified until after the body of the loop.
The body is executed only if a certain condition is met and it terminates when the condition is false.	The body is always executed at least once, regardless of whether the condition is met.

DB Difference Between.net

5. When an algorithm is written in the form of a programming language, it becomes a ……………

a) Flowchart 2Marks

b) Program

c) Pseudo code

d) Syntax

Answer b) Program

6. From the following sorting algorithms which algorithm needs the minimum number of swaps?

a. Bubble sort

b. Quick sort **2Marks**

c. Merge sort

d. Selection sort

The answer is:

d) Selection sort

7. From the following sorting algorithms which has the lowest worst case complexity?

a. Bubble sort

b. Quick sort **2Marks**

c. Merge sort

d. Selection sort

The answer is:

c) Merge sort

8. A) What do you understand 'Decision Making Statements'? **2marks**

Decision-making statements, also known as **conditional statements**, are fundamental building blocks in programming languages. They allow you to control the flow of execution in your program based on specific conditions.

Here are some key aspects of decision-making statements:

- **Evaluate conditions:** They use Boolean expressions (expressions that evaluate to true or false) to determine the program's flow.
- **Control execution:** Based on whether the condition is true or false, they execute specific blocks of code.
- **Common examples:** Different programming languages provide various decision-making statements, but some common ones include:

  if statement: Executes a code block if the condition is true.

  if-else statement: Executes one block if true, another if false.

  switch statement: Executes different code blocks based on the value of an expression compared to multiple cases.

B) Why we use default and break in Switch case statement? **2Marks**

Answer:

Default:

- Provides a **fallback** option for when the expression value doesn't match any case label.
- Its code block executes only if no other case matches.
- Often used for handling unexpected or invalid values.

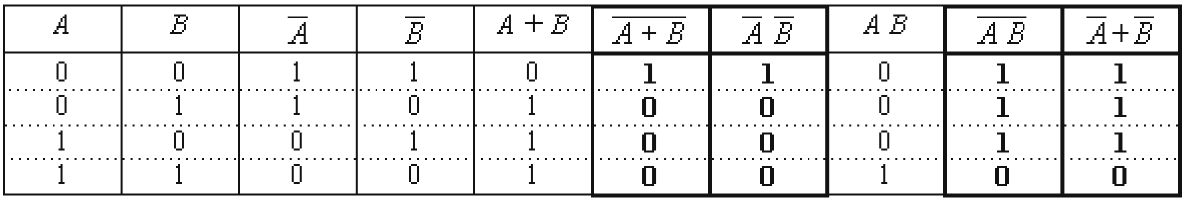
Break:

- **Terminates the execution** of the switch statement after a matching case is found.

- Prevents the code from falling through and executing subsequent cases unintentionally.
- Ensures each case executes its intended code block independently.

9. Demonstrate (show) by the truth table the following De Morgan theorems.
Demonstration of Morgan Low

$\overline{A + B} = \overline{A}\,\overline{B}$ and $\overline{AB} = \overline{A} + \overline{B}$ **5marks**



A	B	$\overline{A}$	$\overline{B}$	$A+B$	$\overline{A+B}$	$\overline{A}\,\overline{B}$	$A\,B$	$\overline{A\,B}$	$\overline{A}+\overline{B}$
0	0	1	1	0	**1**	**1**	0	**1**	**1**
0	1	1	0	1	**0**	**0**	0	**1**	**1**
1	0	0	1	1	**0**	**0**	0	**1**	**1**
1	1	0	0	1	**0**	**0**	1	**0**	**0**

10.

11. Linear (Sequential) Search

- **How it works:** Linear search is the simplest searching technique. It involves checking each element in a data structure (like an array or linked list) sequentially, one by one, until the desired element (search target) is found. If the element is not found after traversing the entire structure, the search is unsuccessful.
- **When it's used:** Linear search is best suited for:
  - **Unsorted data:** If the data is not organized in any particular order, linear search is the primary option.
  - **Small datasets:** For a small number of elements, linear search can be sufficiently fast.

2. Binary Search

- **How it works:** Binary search is a much more efficient algorithm that relies on the concept of dividing and conquering. It works on **sorted data** (either an array or a suitable tree structure). Here's how it works:
  1. Compare the search target with the middle element in the sorted array.
  2. If there's a match, the element is found.
  3. If the target is smaller than the middle element, recursively search in the left half of the array.
  4. If the target is larger than the middle element, recursively search in the right half of the array.

12. .
13. .

14. a) Convert 620_{10} into hexadecimal

b) Convert 1110100110_2 into hexadecimal

c) Convert 111 110 000 from base 2 to base 8

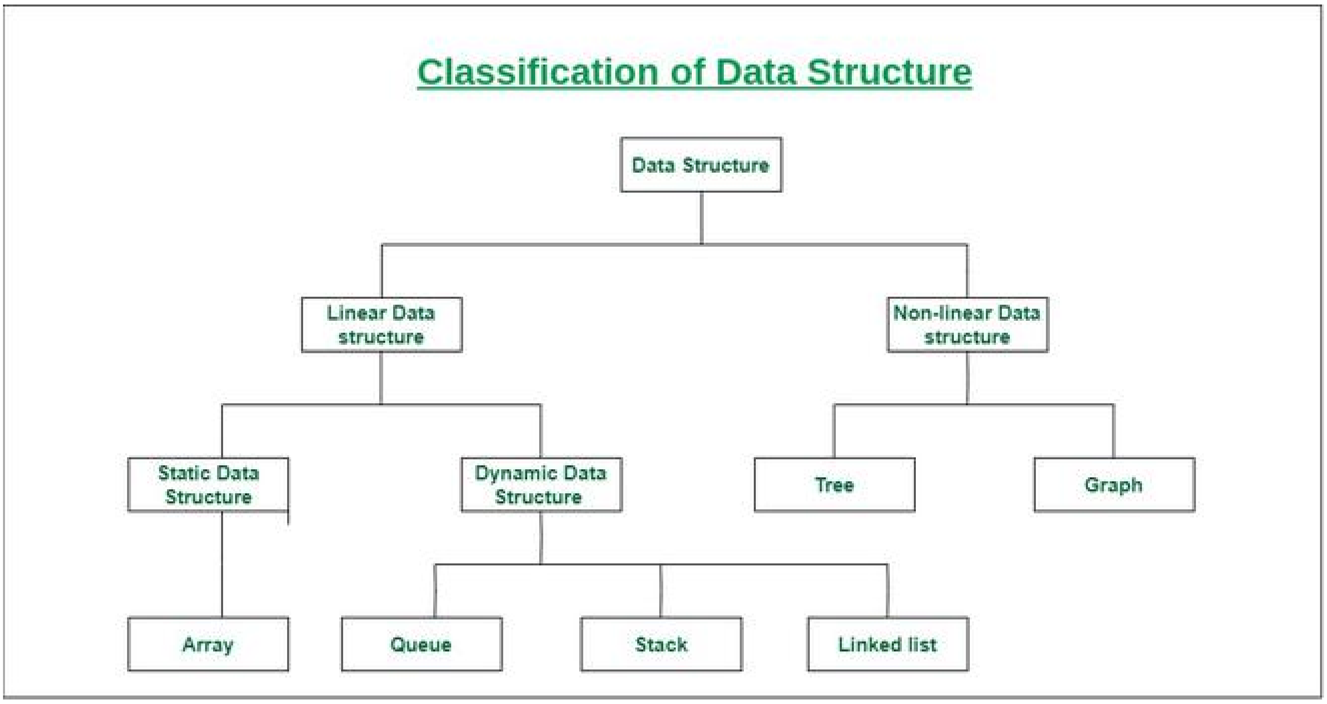
d) Convert 193 from base 10 to binary

- **a. 26C**
- **b. 3A6**
- **c. 760**
- **d. 11000001**

15.

```
Var Note as integer

Start

Write(“enter the note”)

Read(Note)

If (Note>=16) then

Write(“Grade A”)

Else if(Note>=14)then

Write(“Grade B”)

Else if (Note>=12)then

Write(“Grade C”)

Else

Write(“Grade D”)

End if

end
```

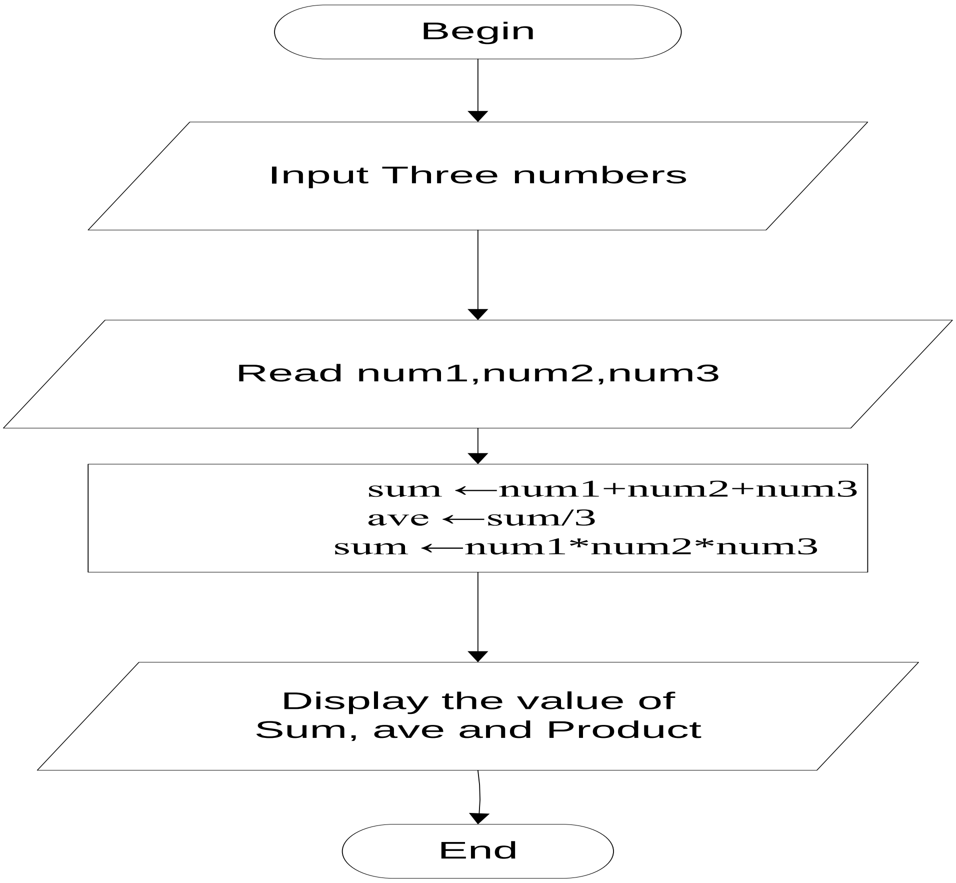
16.



17. **Algorithm:**

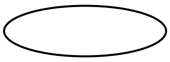
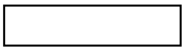
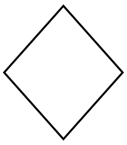
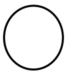
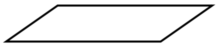
```
Variables num1, num2, num3, sum, ave, product: integers
BEGIN
WRITE ("Enter three numbers:”)
READ (num1, num2, num3)
sum ←num1+num2+num3
ave ←sum/3
product←num1*num2*num3
WRITE ("the sum of ”, num1,num2 ,” and ”,num3,” is ”, sum, ” their average is ”, ave, ” and their product is ”, product)
END
```

Flowchart:



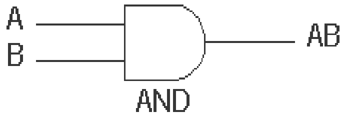
SECTION TWO: ***Choose Three Questions 30Marks***

18.

Name of the symbol	**Diagram**	**Description**
Oval		**It used to start or/and to end a flowchart**
Rectangle		It is used for data processing
Rhombus		**It is used for writing a condition**
Circle		**It is used to join more flow lines**
Parallelogram		**It is used to input data and to display result**

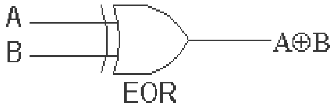
19. Draw the logical symbols of gates:

a. AND: **1Mark**



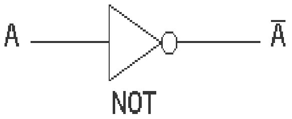
1Mark

b. XOR **1Mark**



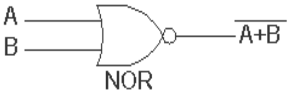
1Mark

c. NOT **1Mark**



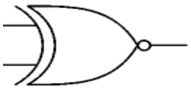
1Mark

d. NOR **1Mark**



1Mark

e. XNOR **1Mark**



1Mark

20.

Qualities of good algorithm:

a) **Input:** The algorithm must take zero or more input.

b) **Output:** The algorithm may produce one or more outputs.

c) **Finiteness or Termination:** an algorithm should terminate infinite number of steps and each step must finish in finite amount of time.

d) **Generality**: an algorithm must be generalized in order to handle a range of input data.

e) **Definiteness**: Each step of algorithm must be defined unambiguously.

f) **Effectiveness**: A human should be able to calculate the exact values involved in the procedure of the algorithm using paper & pencil.

21. (a) Some operators: **select only five 1mark per each**

a) Arithmetic operators

b) Relational operators

c) Logical operators

d) Assignment operators

e) Increment &decrement operators

f) Conditional operators

g) Bitwise operators

h) Special operators

(b)The main difference between **primitive** and **non-primitive** data types are:

- Primitive types are predefined (already defined). Non-primitive types are created by the programmer and is not defined. **1mark**
- Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot. **1mark**
- A primitive type has always a value, while non-primitive types can be null. **1mark**
- A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter. **1mark**
- The size of a primitive type depends on the data type, while non-primitive types have all the same size. **1mark**

22. Examine table below and Fill where it is necessary

Parameter	**Stack**	**Queue**
Basics	It is a linear data structure. The objects are removed or inserted at the same end.	**It is also a linear data structure. The objects are removed and inserted from two different ends. (1mark)**
Working Principle	**It follows the Last In, First Out (LIFO) principle.(1mark)**	It follows the First In, First Out (FIFO) principle.
Operations	**Stack uses push and pop as two of its operations.(1mark)**	Queue uses **enqueue** and **dequeue** as two of its operations.
Structure	Insertion and deletion of elements take place from one end only. It is called the top	**It uses two ends- front and rear. Insertion uses the rear end, and deletion uses the front end. (1mark)**

SECTION THREE: ***Choose One question 15Marks***

23.
```
var ,sum,,average as reals
        var i,n as integers
        array marks(n) as real
         START
        WRITE("enter the number of students ")
        READ(n);
        WRITE("enter the marks of students")
        FOR i←0 TO n
        READ(marks(i))
        NEXT i
        sum←0
        FOR i←0 TO n
```

```
sum←sum+marks(i)
 NEXT i
average←sum/n
WRITE(“the number of students is”, n)
WRITE(“ the students’  marks are”,marks(i))
WRITE(“sum of those mark is”,sum)
WRITE(“average is”, average)
END
```

24. a) Outline three types of linked list **5 MARKS**.

Solution

Simple Linked List
Doubly Linked List
Circular Linked List

b) List Any three Basic Operations From Linked List **5MARKS**

Solution

Following are the basic operations supported by a list.

- **Insertion**
- **Deletion**
- **Display**
- **Update**
- **search**

c) // Node class to represent each element in the linked list **5MARKS**

```
class Node {
  constructor(data) {
    this.data = data;
    this.next = null;  // Initially points to null, indicating no next element
  }
}

// Linked List class to manage the list operations
class LinkedList {
  constructor() {
    this.head = null;  // Head initially points to null, indicating an empty list
    this.size = 0;
  }

  // Insert a new node at the beginning of the list (head)
  insertAtBeginning(data) {
    const newNode = new Node(data);
    newNode.next = this.head;  // Link the new node to the current head
    this.head = newNode;  // Update the head to point to the new node
    this.size++;
  }

  // Insert a new node at the end of the list
  insertAtEnd(data) {
    const newNode = new Node(data);
    if (!this.head) {  // If the list is empty, set the new node as the head
      this.head = newNode;
    } else {
      let current = this.head;
      while (current.next) {  // Traverse to the last node
        current = current.next;
      }
      current.next = newNode;  // Link the last node to the new node
    }
    this.size++;
  }

  // Print the contents of the linked list
  printList() {
    let current = this.head;
    while (current) {
      console.log(current.data);
      current = current.next;
    }
  }
}

// Example usage
const linkedList = new LinkedList();
linkedList.insertAtBeginning(10);
linkedList.insertAtEnd(20);
linkedList.insertAtBeginning(5);
linkedList.printList();   // Output: 5, 10, 20
```