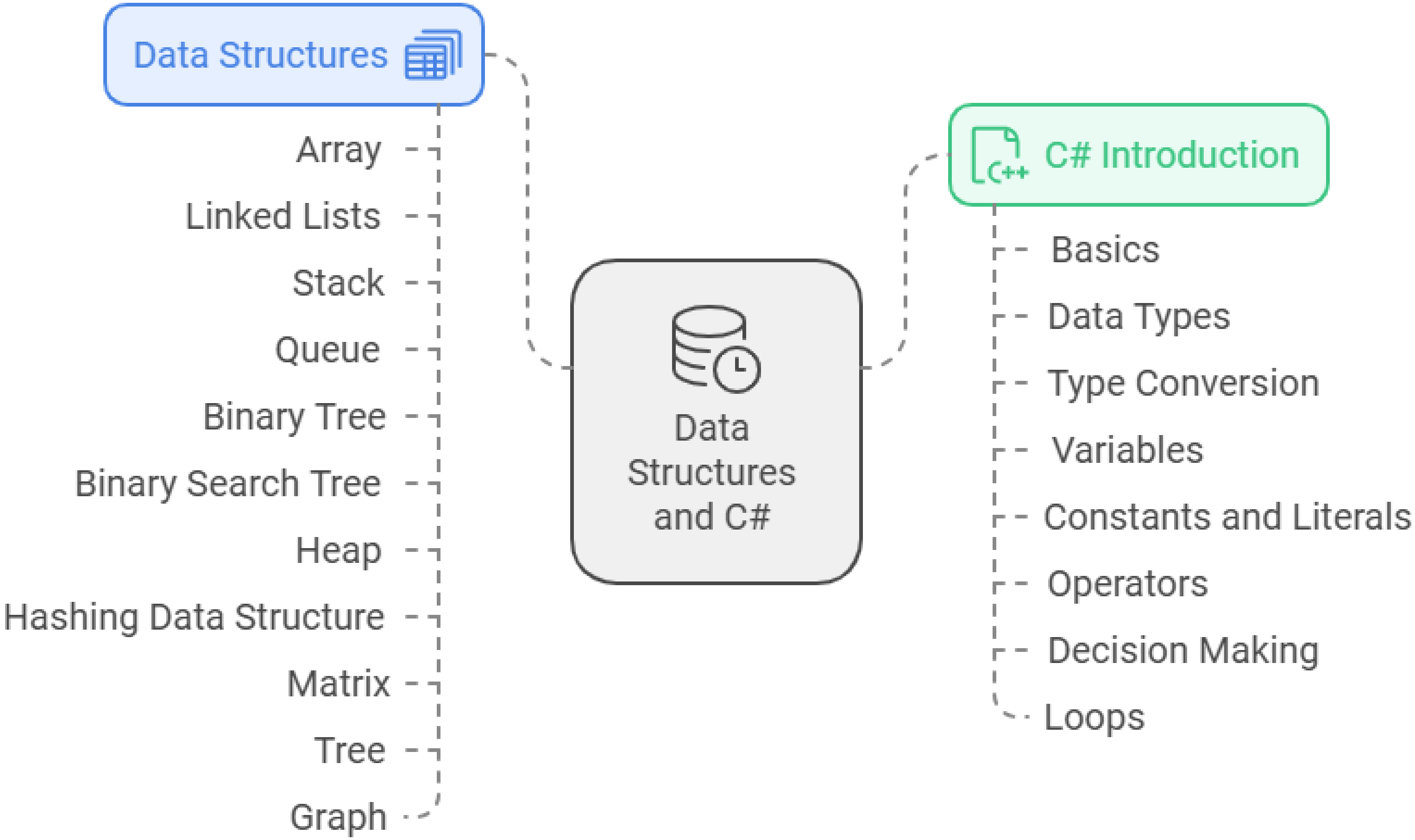
Module 1

Data Structures and C# Programming

Content


Data Structures

Array

Linked Lists

Stack

Queue

Binary Tree

Binary Search Tree

Heap

Hashing Data Structure

Matrix

Tree

Graph

Data Structures and C#

C# Introduction

Basics

Data Types

Type Conversion

Variables

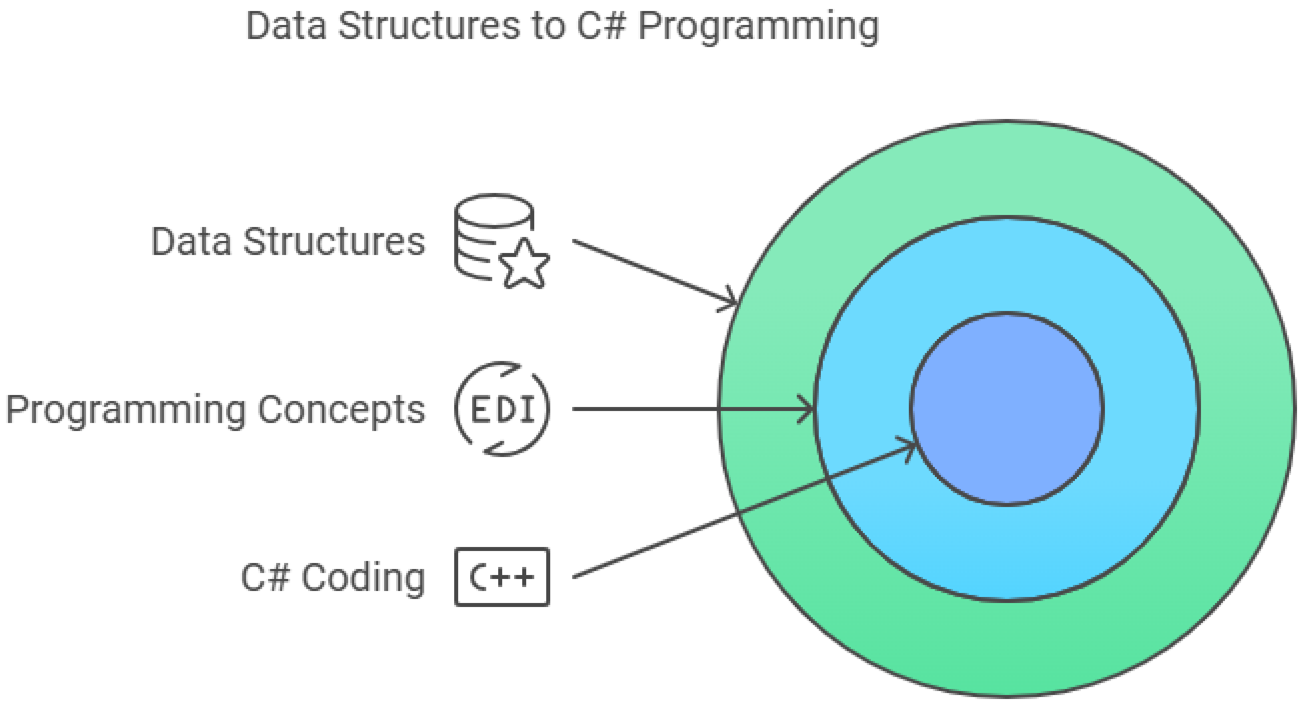
Constants and Literals

Operators

Decision Making

Loops

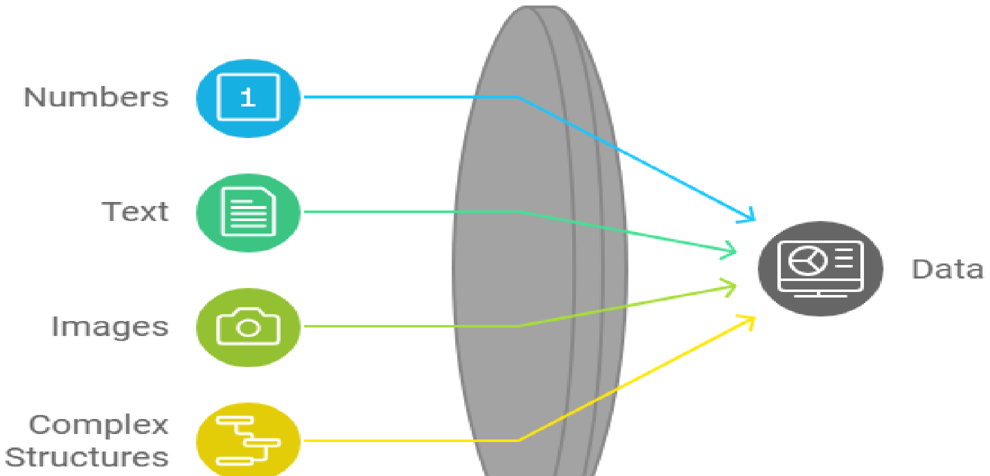
Objective


Data Structures to C# Programming
Data Structures
Programming Concepts
EDI
C# Coding
C++

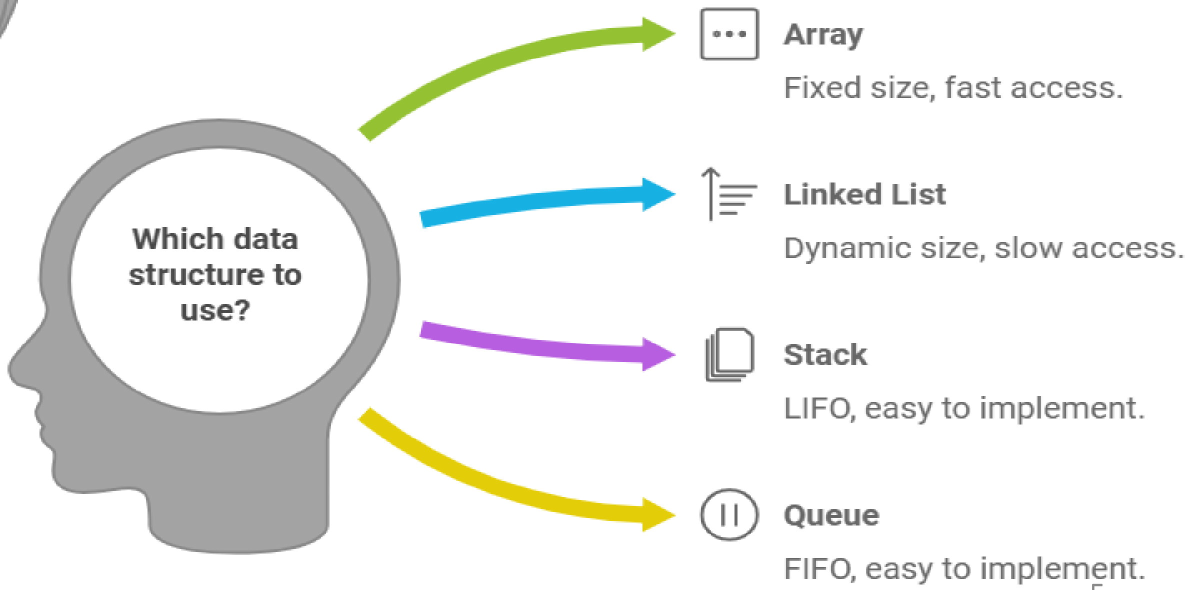
Part 1: Overview Data and Data Structure

I. Data and Data Structures

What is data?

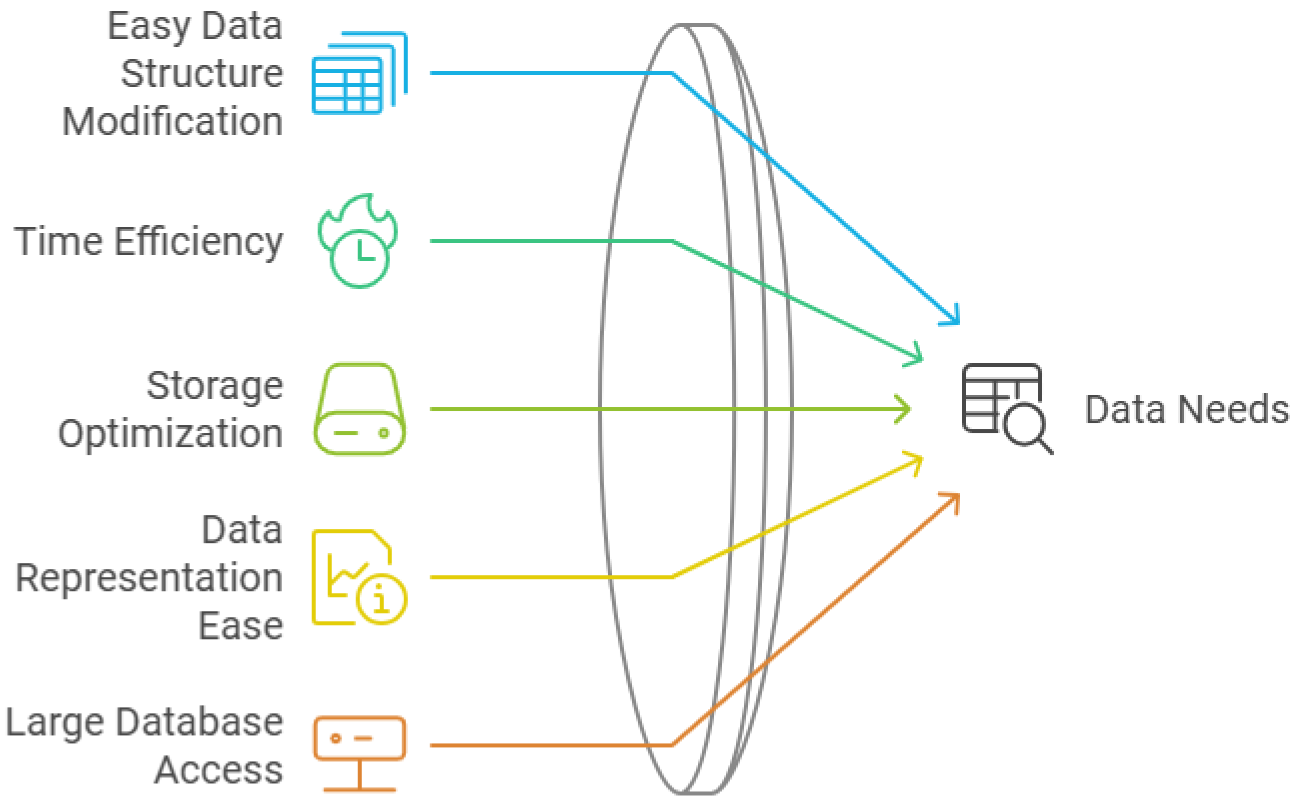


What is a data structure?



II. Need of Data Structure

Data Needs Benefits



Easy Data Structure Modification

Time Efficiency

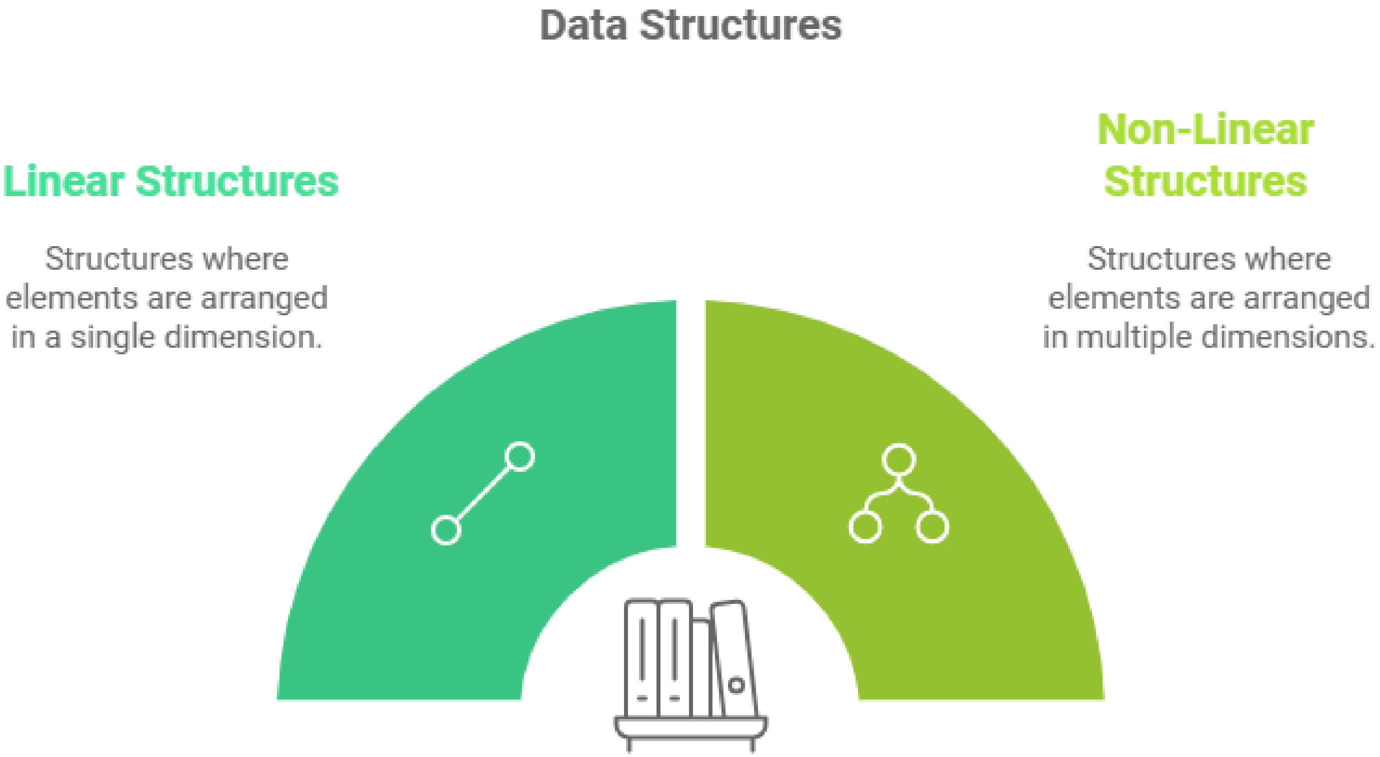
Storage Optimization

Data Representation Ease

Large Database Access

Data Needs

III. Classification of DS



Data Structures

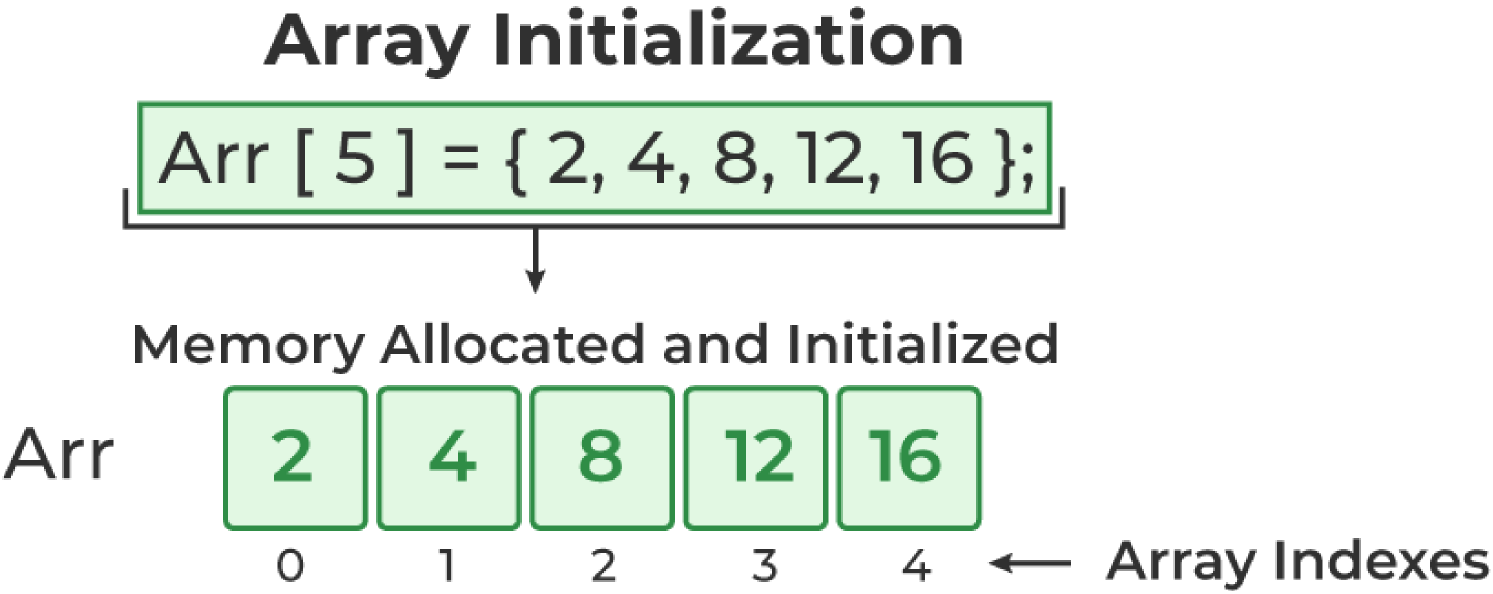
Linear Structures

Structures where elements are arranged in a single dimension.

Non-Linear Structures

Structures where elements are arranged in multiple dimensions.

IV. Array



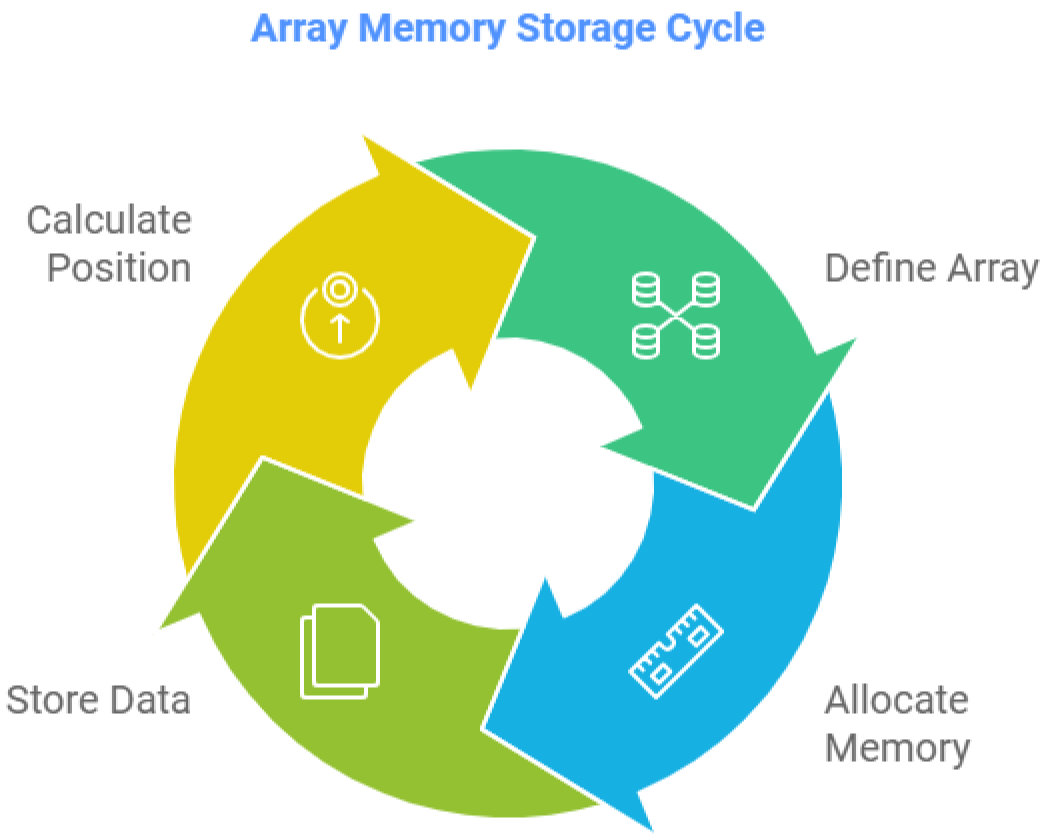
Array Initialization

```
Arr [ 5 ] = { 2, 4, 8, 12, 16 };
```

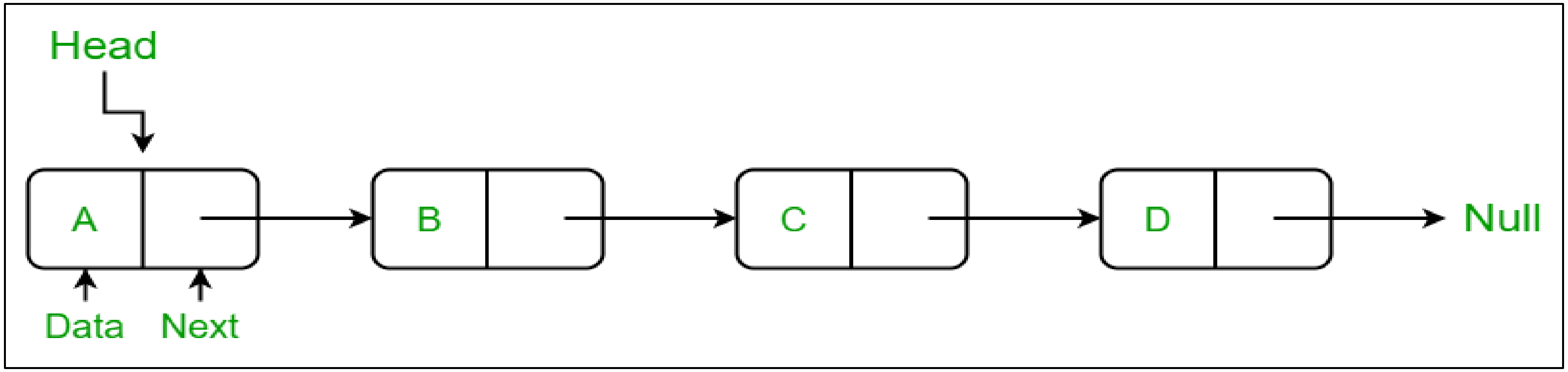
Memory Allocated and Initialized

Arr	2	4	8	12	16	
	0	1	2	3	4	← Array Indexes

IV. Array


Array Memory Storage Cycle

V. Linked Lists



Understanding Linked Lists

Pointers

Used to link elements together

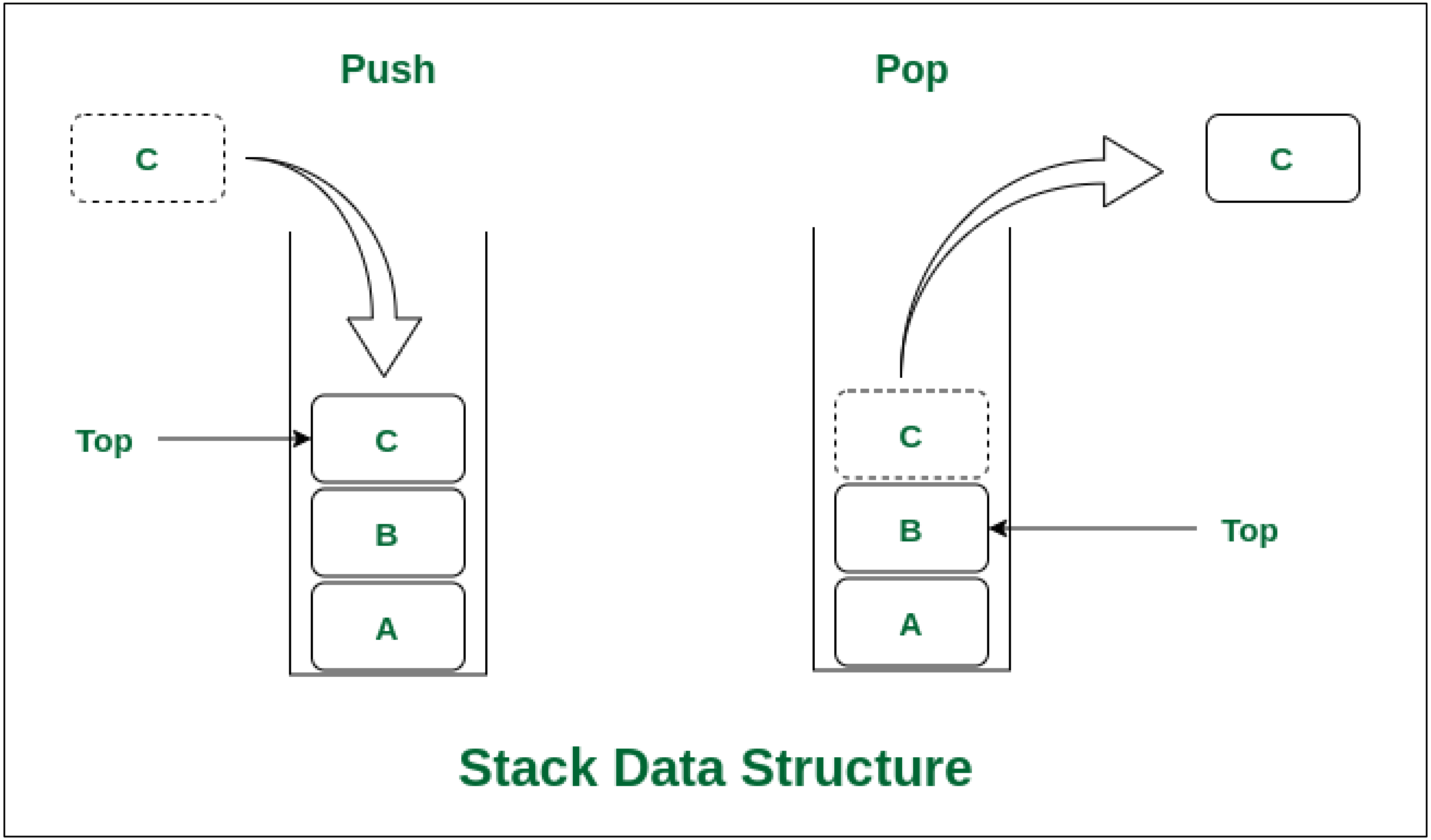
Linear Data Structure

Represents a sequence of elements

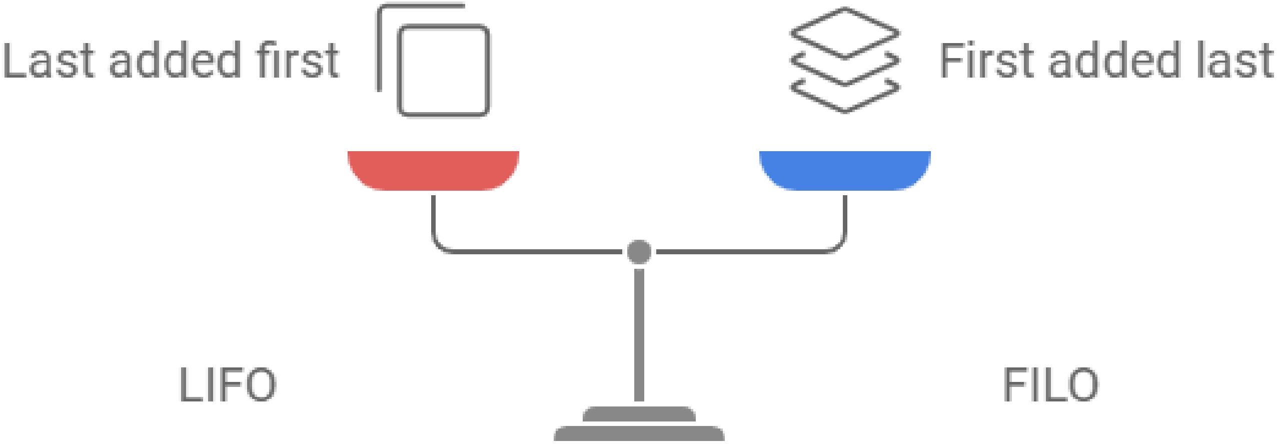
Non-contiguous Storage

Elements are not stored in adjacent memory locations

VI. Stack

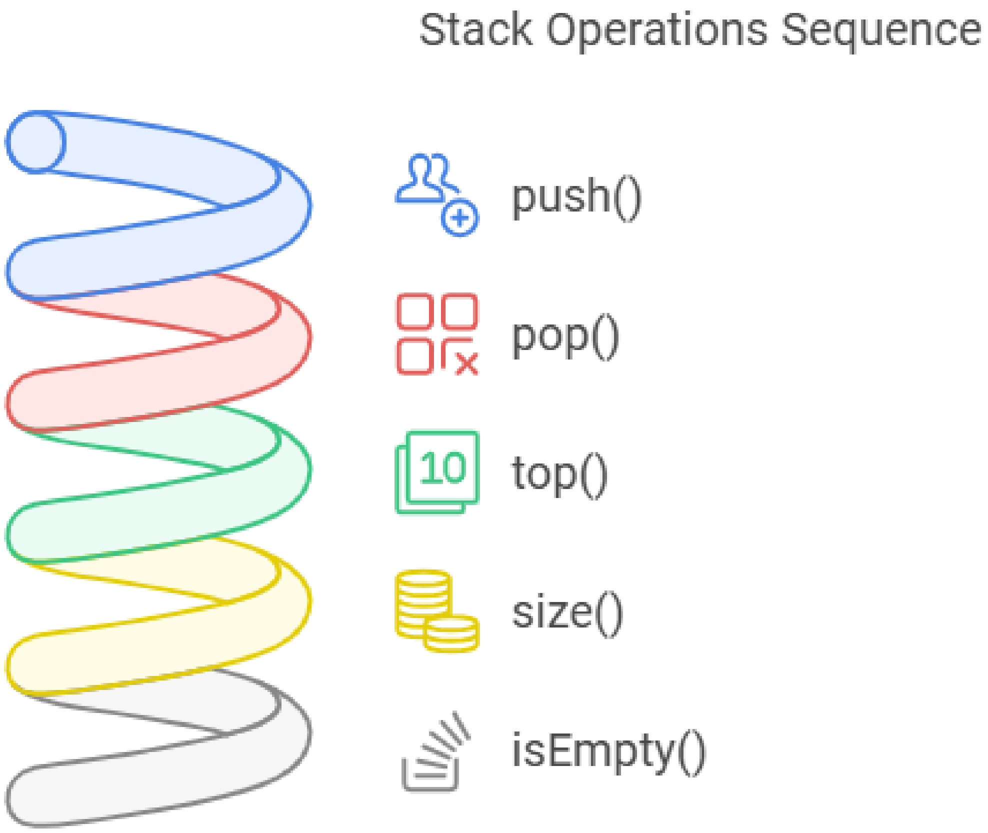


VI. Stack



Understanding stack operation orders.

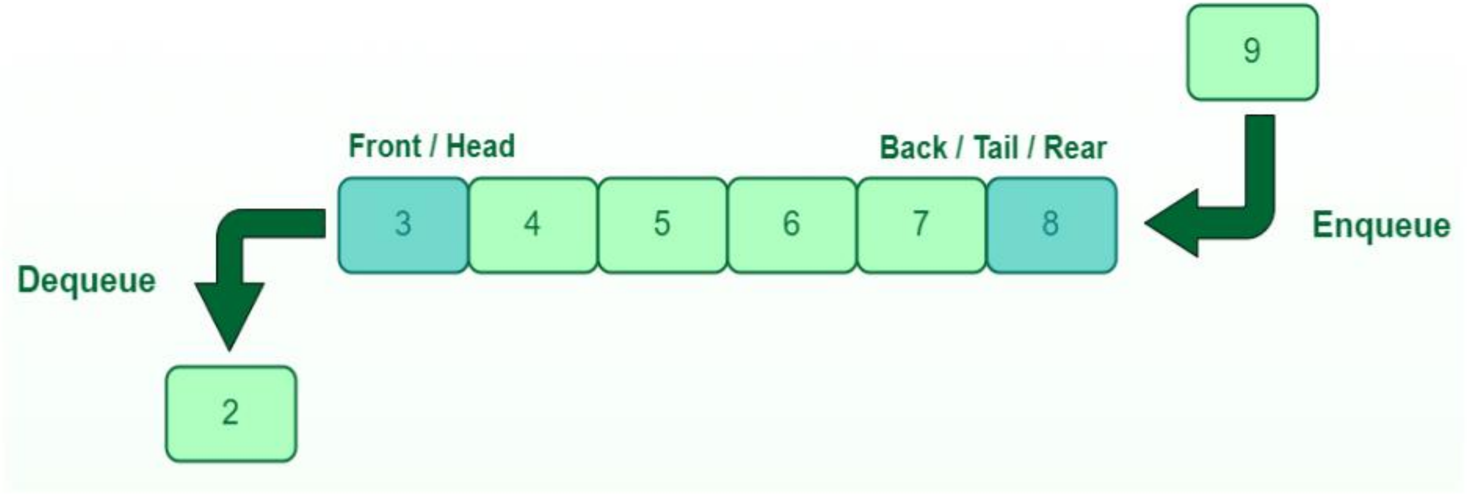
VI. Stack



Stack Operations Sequence

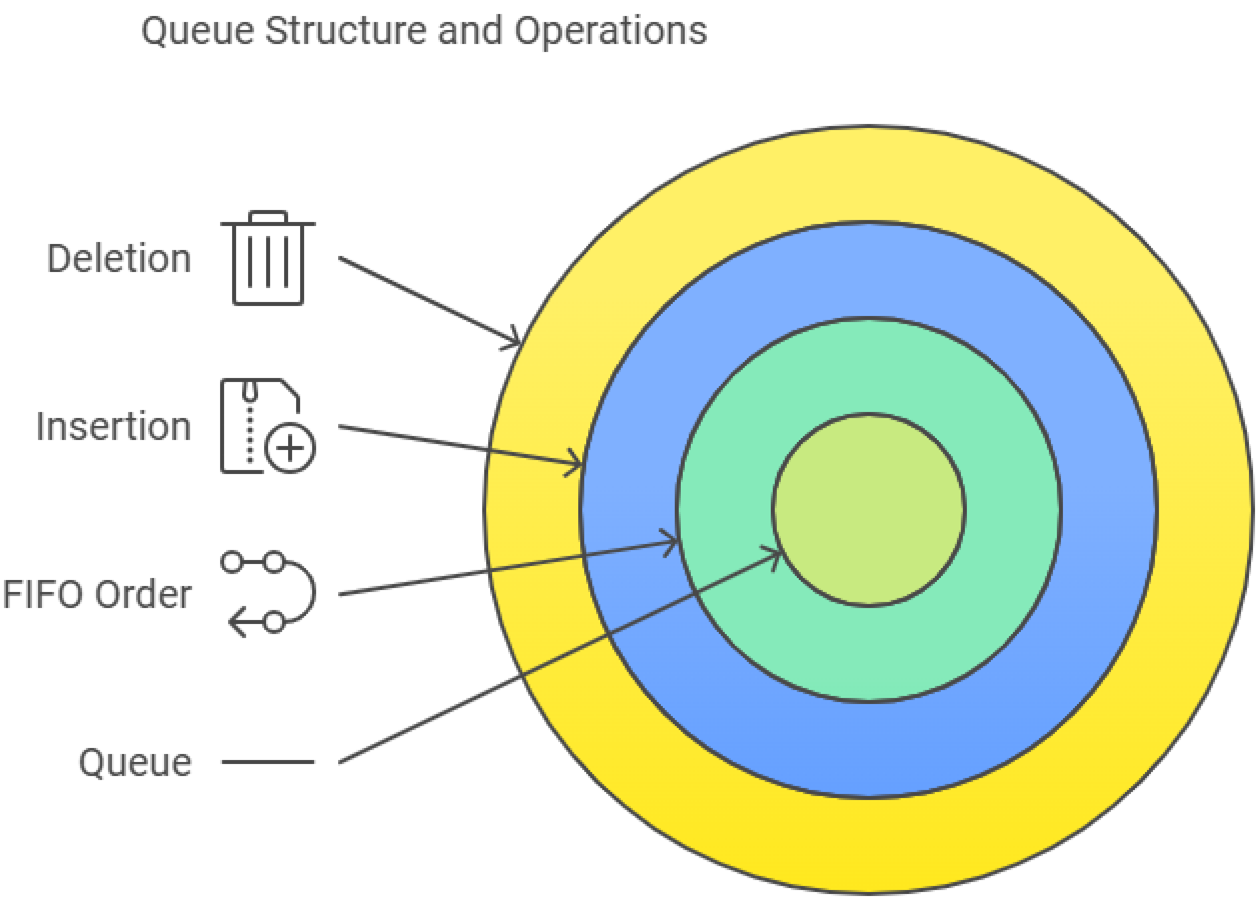
- push()
- pop()
- top()
- size()
- isEmpty()

VII. Queue



Queue Data Structure

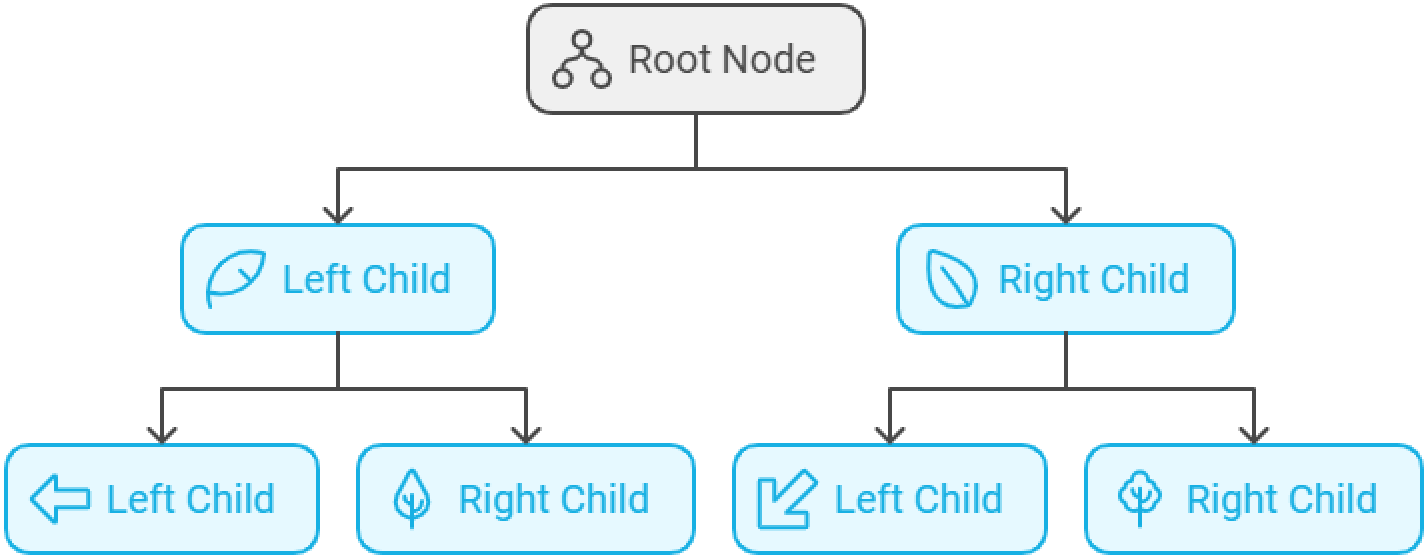
VII. Queue


Queue Structure and Operations
Deletion
Insertion
FIFO Order
Queue

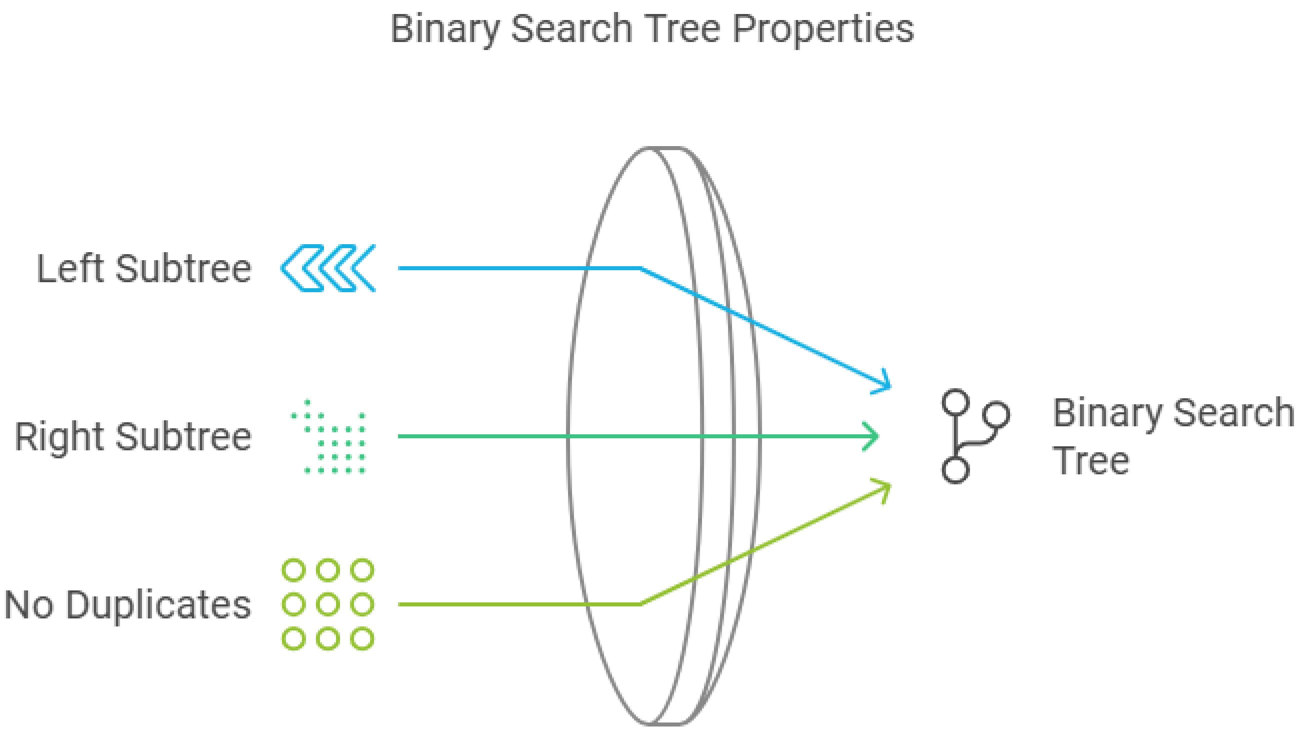
VII. Queue

- Enqueue
- Dequeue
- Peek
- Rear
- isFull
- isNull

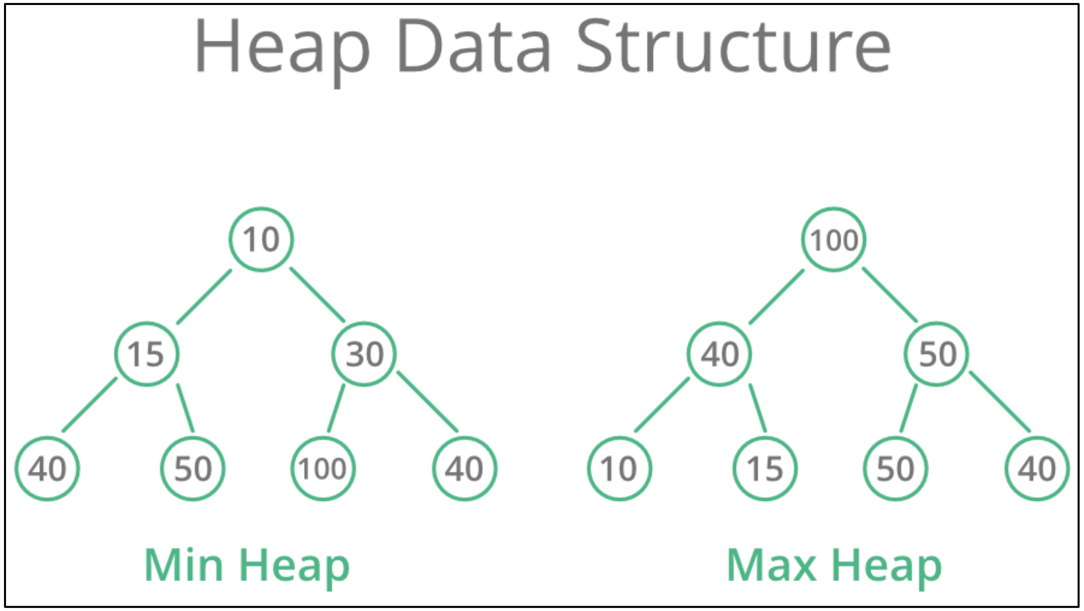
VIII. Binary Tree

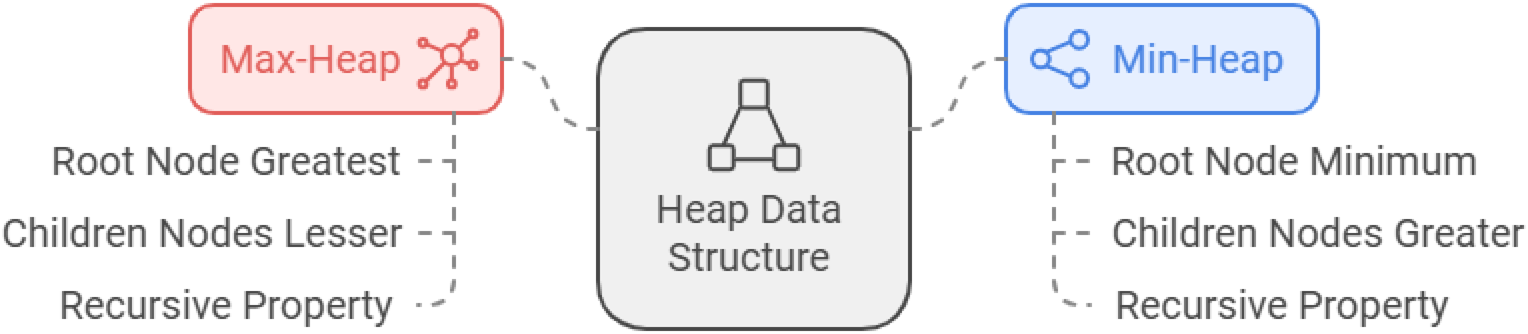


IX. Binary Search Tree



X. Heap





Max-Heap

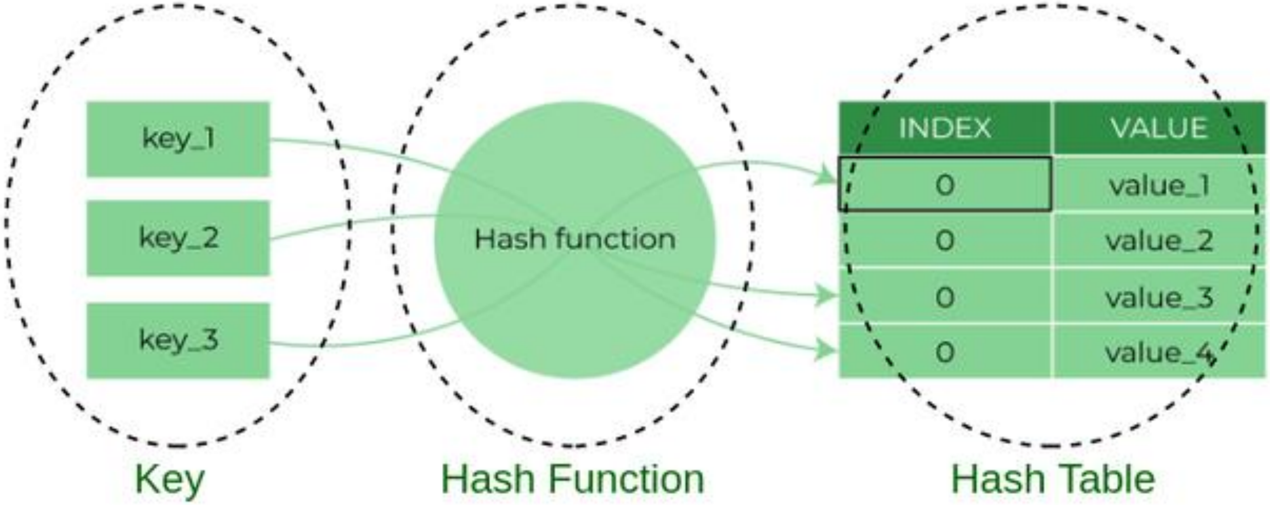
- Root Node Greatest
- Children Nodes Lesser
- Recursive Property

Heap Data Structure

Min-Heap

- Root Node Minimum
- Children Nodes Greater
- Recursive Property

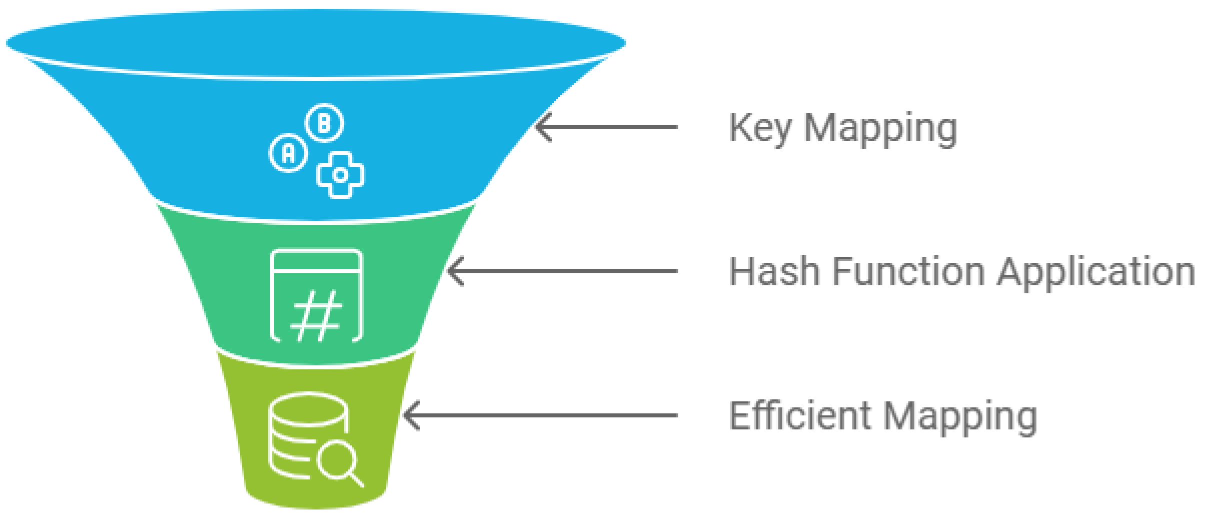
XI. Hashing Data Structure



INDEX	VALUE
0	value_1
0	value_2
0	value_3
0	value_4

Components of Hashing

XI. Hashing Data Structure

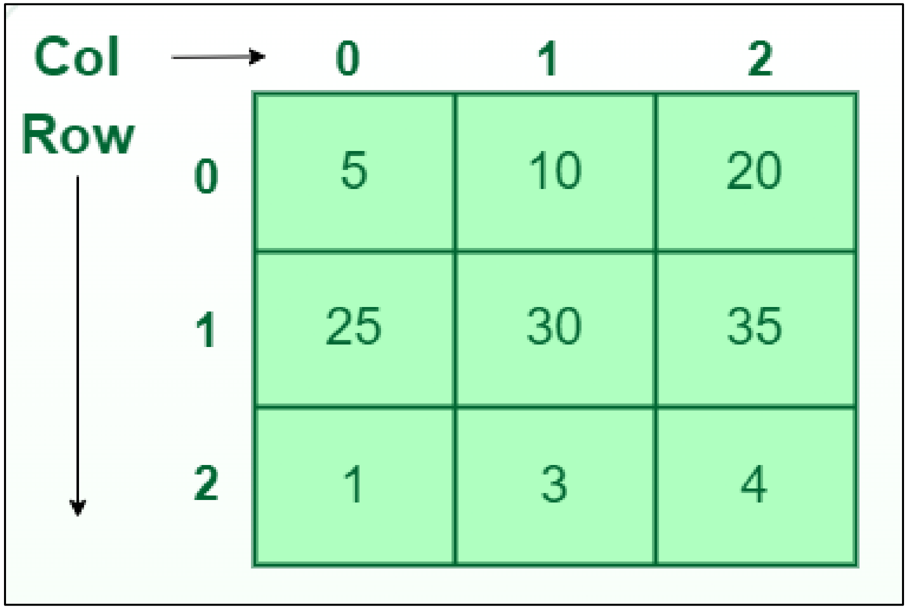


Efficient Data Access through Hashing

Data Input

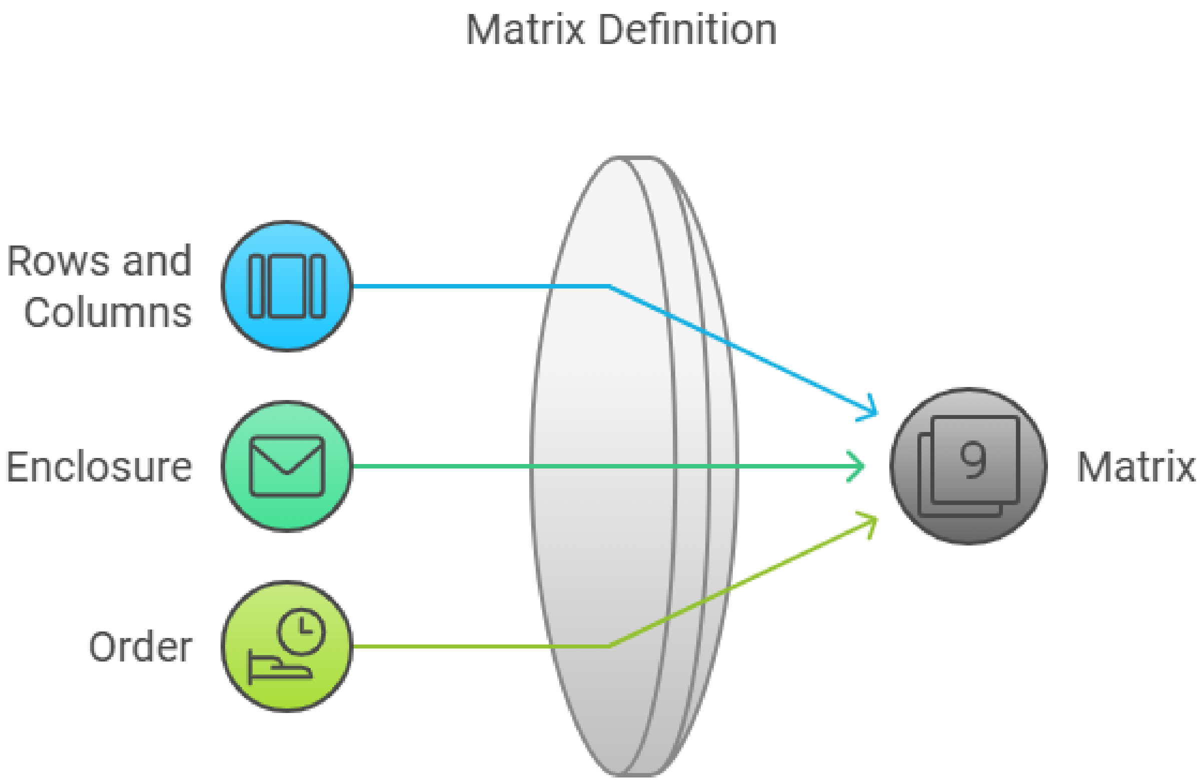
Fast Data Retrieval

XII. Matrix

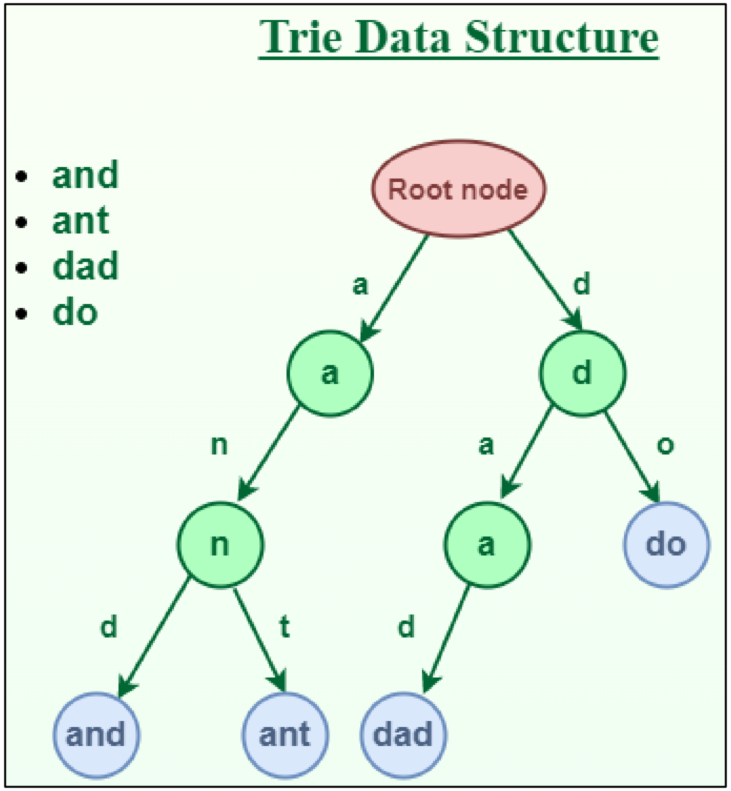


	0	1	2
0	5	10	20
1	25	30	35
2	1	3	4

XII. Matrix


Matrix Definition
Rows and Columns
Enclosure
Order
9
Matrix

XIII. Tree


Trie Data Structure
• and
• ant
• dad
• do
Root node
a
d
a
d
n
a
o
n
a
do
d
t
d
and
ant
dad

XIII. Tree

Which data structure to use for efficient information retrieval?

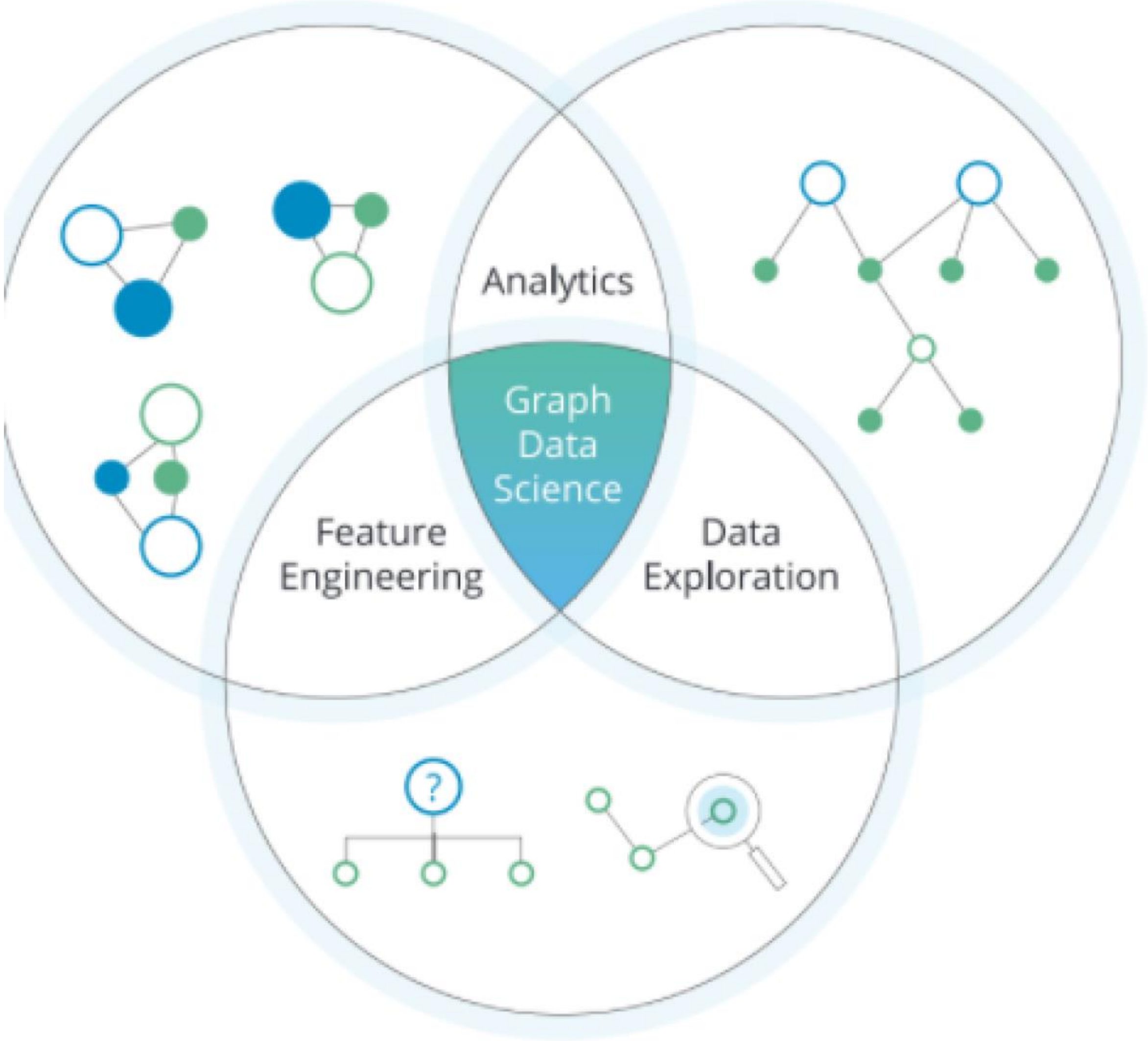
Use Tree

Optimal search complexities (key length)

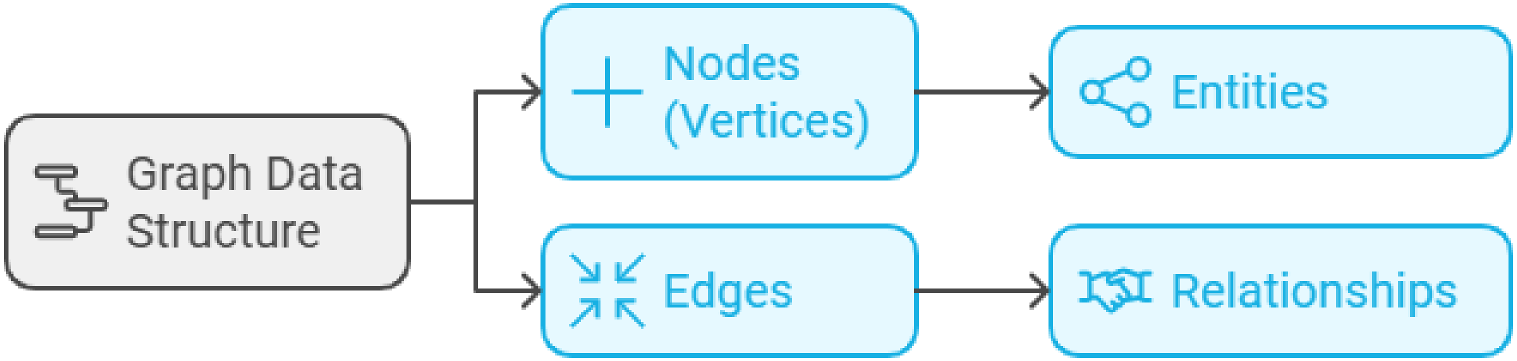
Use other data structures

Higher search complexities

XIV. Graph


Analytics
Graph Data Science
Feature Engineering
Data Exploration
?

XIV. Graph


Graph Data Structure

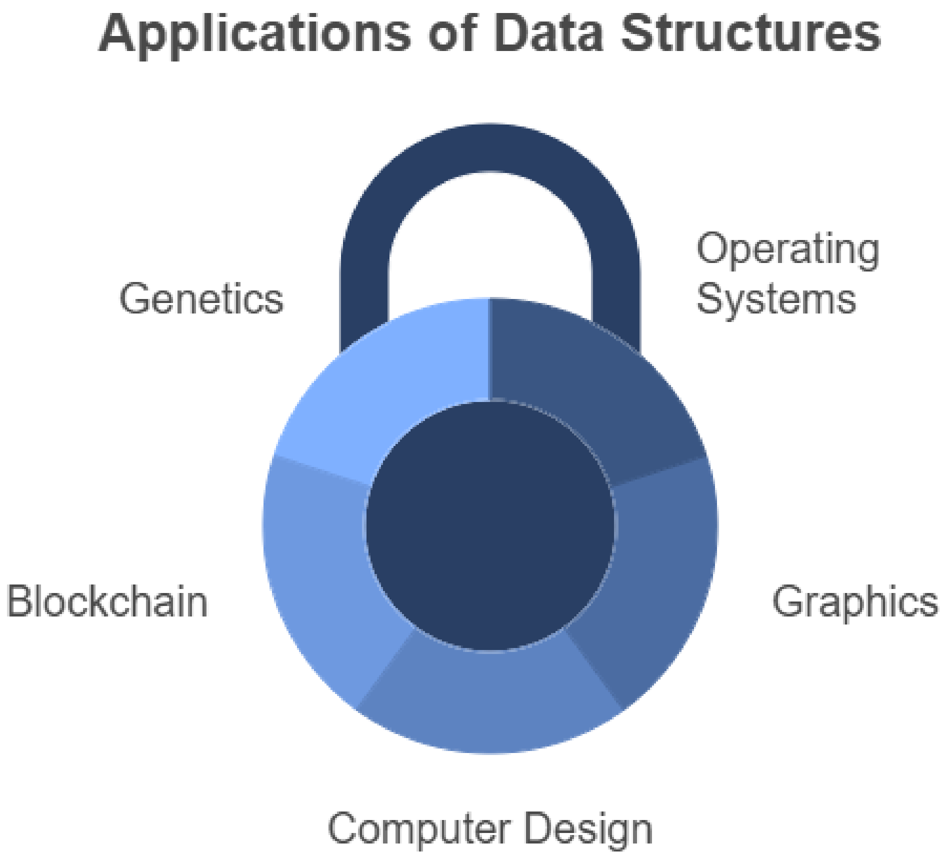
Nodes (Vertices)

Edges

Entities

Relationships

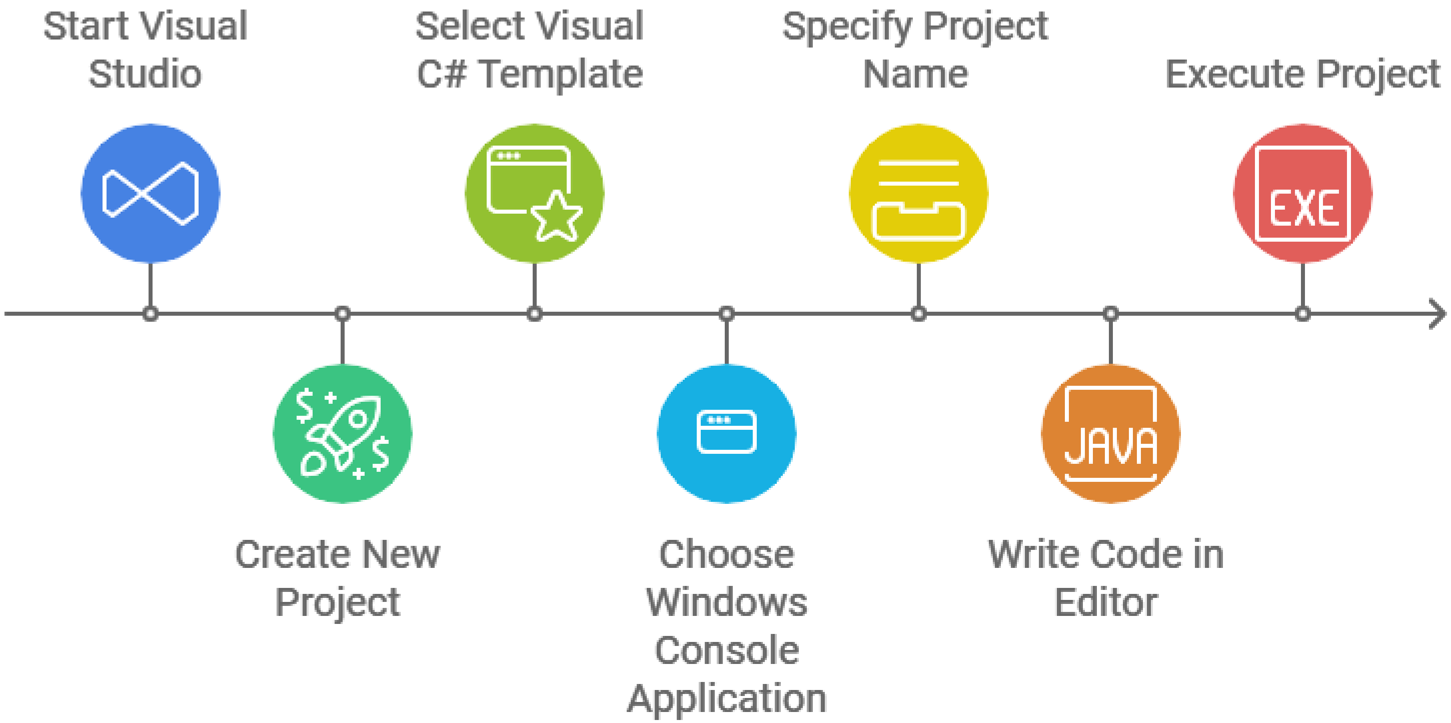
XIV. Graph


Applications of Data Structures
Genetics
Operating Systems
Blockchain
Graphics
Computer Design

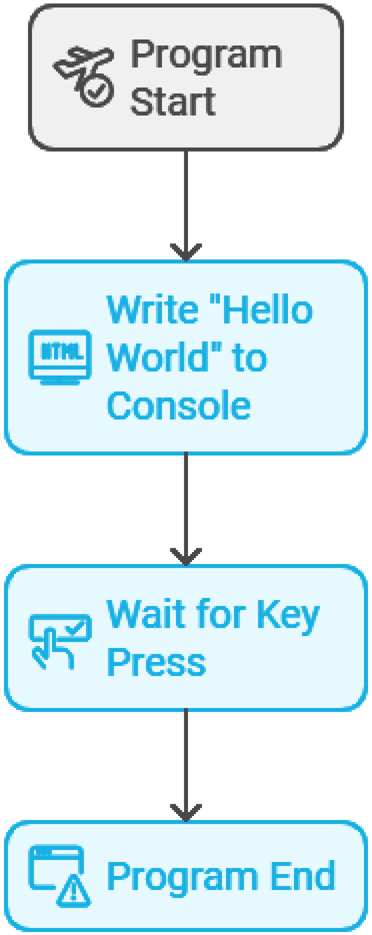
Part 2: Introduction to C#

I. Basics of C# syntax and structure

Creating and Running a Console Application in Visual Studio

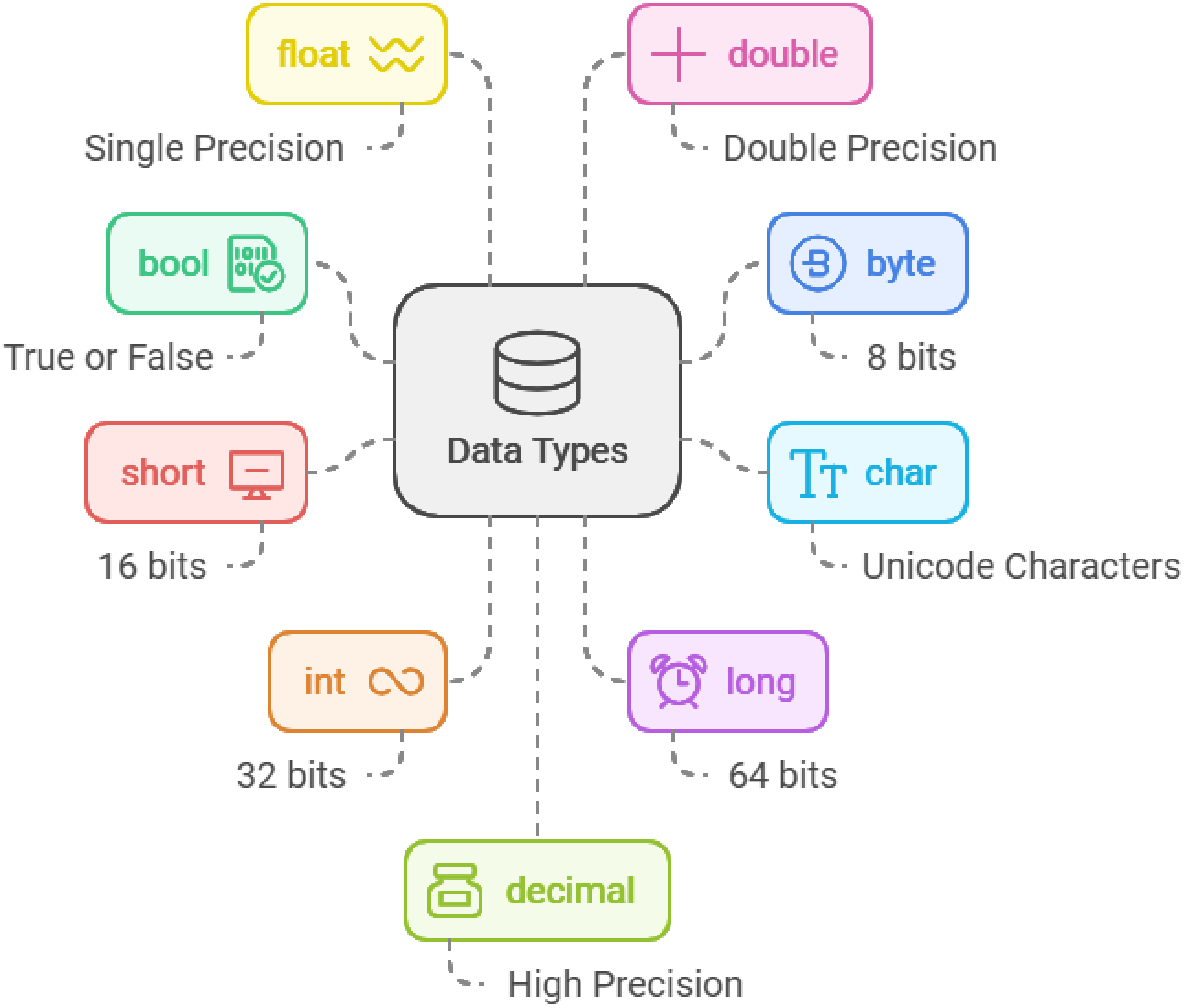


I. Basics of C# syntax and structure



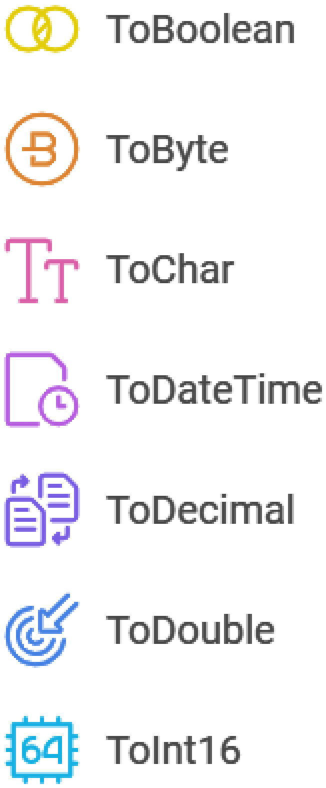
```
using System;
namespace HelloWorldApplication {
  class HelloWorld {
    static void Main(string[] args) {
      /* my first program in C# */
      Console.WriteLine("Hello World");
      Console.ReadKey();   }
  }
}
```

II. C# - Data Types



III. C# - Types Conversion

- **Implicit type conversion** − These conversions are performed by C# in a type-safe manner. Conversions from smaller to larger integral types.
- **Explicit type conversion** − These conversions are done explicitly by users using the pre-defined functions. Explicit conversions require a cast operator.



ToBoolean

ToByte

ToChar

ToDateTime

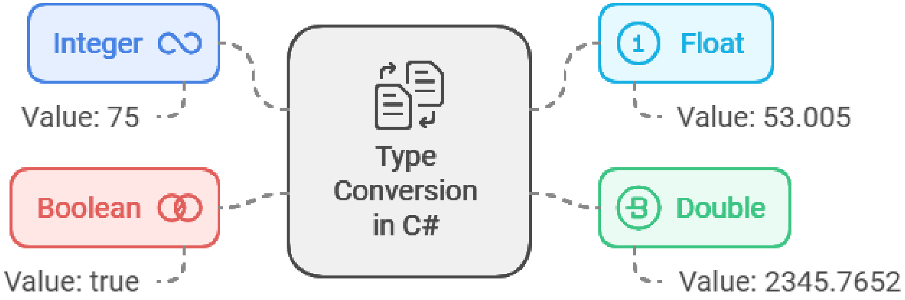
ToDecimal

ToDouble

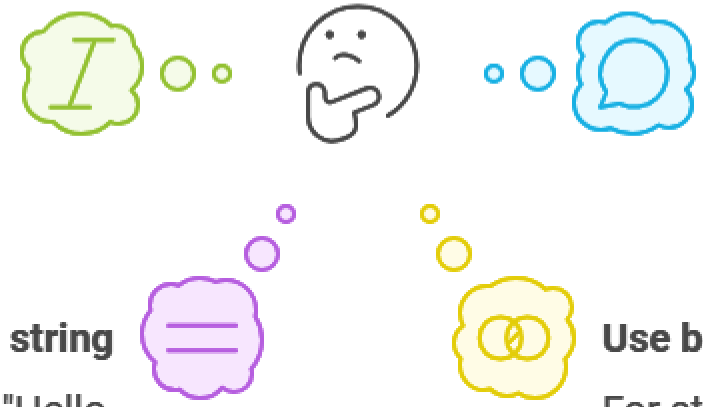
ToInt16

III. C# - Types Conversion

```
using System;
namespace TypeConversionApplication {
  class StringConversion {
    static void Main(string[] args) {
      int i = 75; float f = 53.005f; bool b = true;
      double d = 2345.7652;
     Console.WriteLine(i.ToString());
      Console.WriteLine(f.ToString());
      Console.WriteLine(d.ToString());
      Console.WriteLine(b.ToString());
      Console.ReadKey();   }
  }
}
```



IV. C# - Variable



Which data type to use for a variable in C#?

Use int

For storing whole numbers (e.g., 1, 2, 3).

Use float

For storing decimal numbers (e.g., 1.5, 2.3).

Use string

For storing text (e.g., "Hello, World!").

Use bool

For storing true/false values.

IV. C# - Variable

```
using System;
namespace VariableDefinition {
  class Program {
    static void Main(string[] args) {
      short a;
      int b ;
      double c;
      /* actual initialization */
      a = 10;
      b = 20;
      c = a + b;
      Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c);
      Console.ReadLine();
    }
  }
}
```

V. C# - Constants and Literals

Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal.

a) Character Constants

Escape sequence	Meaning		Escape sequence	Meaning
\\	\ character		\'	' character
\"	" character		\?	? character
\a	Alert or bell		\b	Backspace
\b	Backspace		\f	Form feed
\n	Newline		\r	Carriage return
\t	Horizontal tab		\v	Vertical tab

V. C# - Constants and Literals

b) Integer Literals

Following are other examples of various types of Integer literals −

```
30          /* int */
30l         /* long */
85          /* decimal */
0x4b        /* hexadecimal */
```

c) Floating-point Literals

Here are some examples of floating-point literals −

```
3.14159        /* Legal */
314159E-5F     /* Legal */
.e55           /* Illegal: missing integer or fraction */
```

V. C# - Constants and Literals

d) String Literals

Here are some examples of string literals.

"hello, dear"

"hello, \

dear"

"hello, " "d" "ear"

@"hello dear"

```
using System;
namespace DeclaringConstants {
  class Program {
    static void Main(string[] args) {
      const double pi = 3.14159;
      // constant declaration
      double r;
      Console.WriteLine("Enter Radius: ");
      r = Convert.ToDouble(Console.ReadLine());
      double areaCircle = pi * r * r;
      Console.WriteLine("Radius: {0}, Area: {1}", r, areaCircle);
      Console.ReadLine();    }
  }
}
```

VI. C# - Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

a) Arithmetic Operators

Operator	Description	Example
+	Adds two operands	A + B = 30
-	Subtracts second operand from the first	A - B = -10
*	Multiplies both operands	A * B = 200
/	Divides numerator by de-numerator	B / A = 2
%	Modulus Operator and remainder	B % A = 0

VI. C# - Operators

b) Relational Operators

Operator	Description	Example
==	Equal or not, if yes becomes true.	(A == B) is not true.
!=	Equal or not, if values are not equal is true.	(A != B) is true.
>	Greater than the value. If yes becomes true.	(A > B) is not true.
<	Less than the value. if yes becomes true.	(A < B) is true.
>=	Greater than or equal, if yes becomes true.	(A >= B) is not true.
<=	Less than or equal value, if yes becomes true.	(A <= B) is true.

VI. C# - Operators

c) Logical Operators

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero is true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non zero is true.	(A || B) is true.
!	Called Logical NOT Operator. If true then Logical NOT is false.	!(A && B) is true.

VI. C# - Operators

d) Bitwise Operators

Operator	Description	Example
&	Binary AND Operator copies a bit	(A & B) = 12, 0000 1100
|	Binary OR Operator copies a bit	(A | B) = 61, 0011 1101
^	Binary XOR Operator copies the bit	(A ^ B) = 49, 0011 0001
~	Binary Ones Complement Operator	(~A) = -61, 1100 0011

VI. C# - Operators

e) Assignment Operators

Operator	Description	Example
=	Simple assignment operator, Assigns values	C = A + B assigns value of A + B into C
+=	Add AND assignment operator, It adds and assign the result	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It adds and assign the result	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It adds and assign the result	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It adds and assign the result	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It adds and assign the result	C %= A is equivalent to C = C % A

VII. C# - Decision Making

No.	Statement & Description
1	**if statement**. An **if statement** consists of a Boolean expression followed by one or more statements.
2	**if...else statement**. An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is false.
3	**nested if statements**. You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).
4	**switch statement**. A **switch** statement allows a variable to be tested for equality against a list of values.
5	**nested switch statements**. You can use one **switch** statement inside another **switch** statement(s).

VIII. C# - Loops

No.	Loop Type & Description
1	**while loop**. It repeats a statement or a group of statements while a given condition is true. It tests the condition before executing the loop body.
2	**for loop**. It executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	**do...while loop**. It is similar to a while statement, except that it tests the condition at the end of the loop body
4	**nested loops**. You can use one or more loop inside any another while, for or do..while loop.

Loop Control Statements

No.	Control Statement & Description
1	break statement. Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.
2	continue statement. Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

Quizzes

1. What are Data Structures?
2. Why Create Data Structures?
3. What are some applications of Data structures?
4. Explain the process behind storing a variable in memory.
5. Can you explain the difference between file structure and storage structure?
6. Describe the types of Data Structures?
7. What is a stack data structure? What are the applications of stack?
8. What are different operations available in stack data structure?
9. What is a queue data structure? What are the applications of queue?
10. What are different operations available in queue data structure?
11. Differentiate between stack and queue data structure.
12. What is C# structure C# - Data Types?
13. How to transfer data from one to another (Types Conversion)?
14. What is C# - Variable C# and Constants?
15. What is C# - Operators? What are the assignment operators?
16. What is C# - Decision Making and C# - Loops?

THANK
YOU!