Data Structures and Algorithms

Trong-Hop Do
University of Information Technology, HCM

Agenda

- Introduction of data structure
- Example of data structure and algorithm
- Example of the relationship between data structure and algorithm
- Overview of memory and array
- Criteria and classification of data structure
- Criteria and representation of an algorithm

What is data structure?

A data structure (DS) is a way of organizing data so that it can be used effectively.

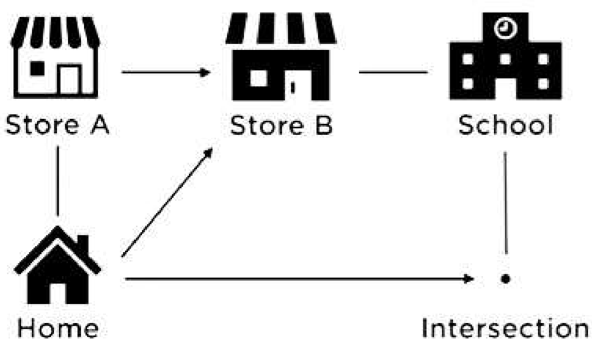
Why are there different types of data structures?

- They all have different strengths and weakness
- Some are fast and storing and recording data, while others are not
- Some are fast at searching and retrieving data, while others are not

Why Data Structure?

- They are essential ingredients in creating fast and powerful algorithms.
- They help to manage and organize data.
- They make code cleaner and easier to understand.

Example of data structure



Home	(49.2, -123.4)
Store A	(49.3, -123.4)
Store B	(49.3, -123.3)
School	(49.3, -123.2)
Intersection	(49.2, -123.2)

(array/list type)

(Home, Store A)
(Store A, Home)
(Home, Store B)
(Home, Intersection)
(Store A, Store B)
(Store B, School)
(School, Store B)
(Intersection, School)

(Hash map/hash table t

Home	(Store A, Store B, Intersection)
Store A	(Store B)
Store B	(School)
School	(Store B, School)
Intersection	(School)

What are algorithms?

Algorithms = operations on different data structures +

set of instructions for executing them

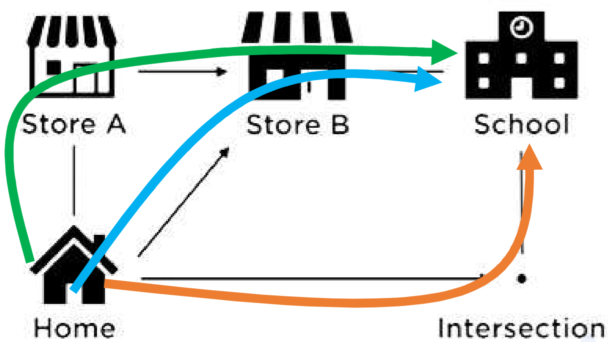
lucid, systematic,
and penetrating
treatment of basic
and dynamic data
structures, sorting,
recursive algorithms,
language structures,
and compiling

NIKLAUS WIRTH

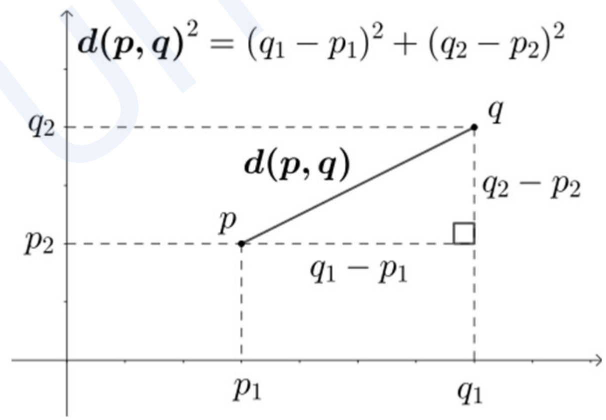
Algorithms + Data Structures = Programs

PRENTICE-HALL
SERIES IN
AUTOMATIC
COMPUTATION

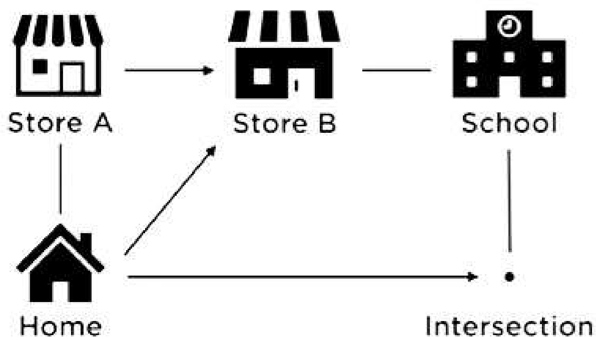
Example: finding shortest path



Home	(49.2, -123.4)
Store A	(49.3, -123.4)
Store B	(49.3, -123.3)
School	(49.3, -123.2)
Intersection	(49.2, -123.2)

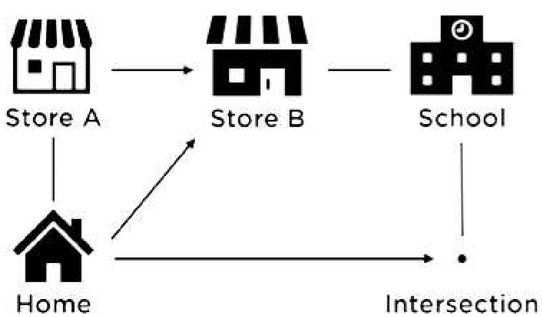


Example: finding shortest path



- Find places you can go from home
- From each of those places, find all paths
- Keep track of the distance you've traveled as you go
- Repeat this process until you get to school
- List up all the paths you've traveled
- Compare the distances of those paths
- Find the shortest path

Example: finding shortest path



Home	(49.2, -123.4)
Store A	(49.3, -123.4)
Store B	(49.3, -123.3)
School	(49.3, -123.2)
Intersection	(49.2, -123.2)

(Home, Store A)
(Store A, Home)
(Home, Store B)
(Home, Intersection)
(Store A, Store B)
(Store B, School)
(School, Store B)
(Intersection, School)

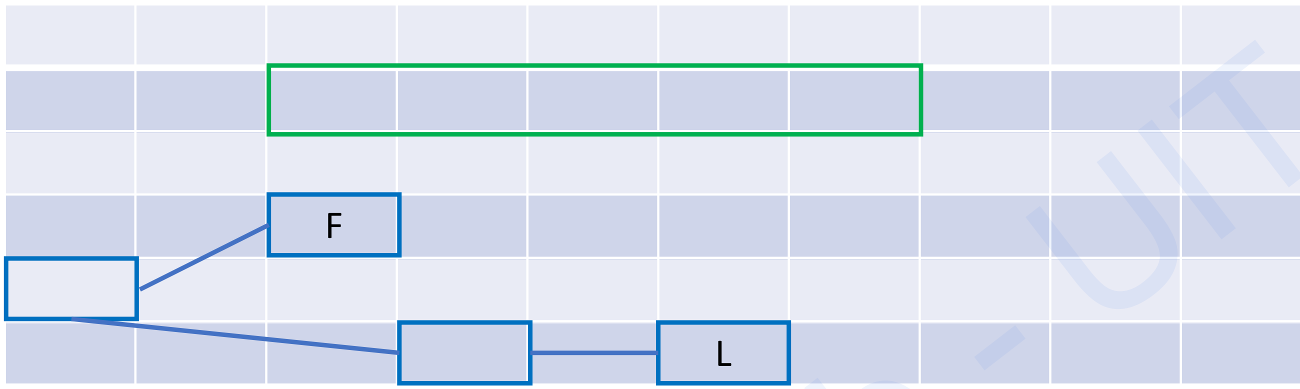
Home	(Store A, Store B, Intersection)
Store A	(Store B)
Store B	(School)
School	(Store B, School)
Intersection	(School)

Step 1: Find places you can go from home

Go all rows and look for Home.

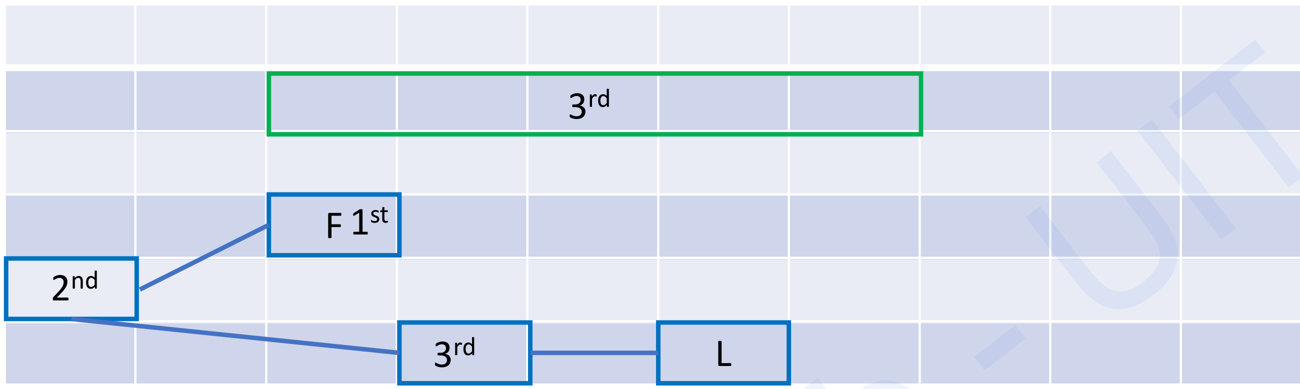
Go row by row **until** find Home.

Example: student list



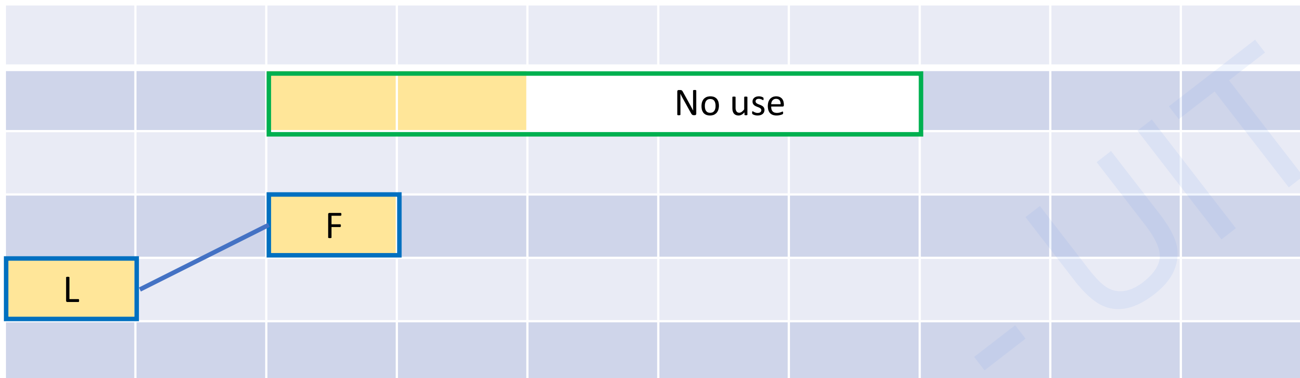
- Problem: store the information of students who register the DS&A course in the cabinet.
- First option: allocate fixed number of **consecutive** boxes.
- Second option: first student comes, find one empty box → second student, find another box, use string to connect first and second boxes → third student, find another box, use string to connect second and third boxes. Always know the location of the first and final boxes.

Example: student list



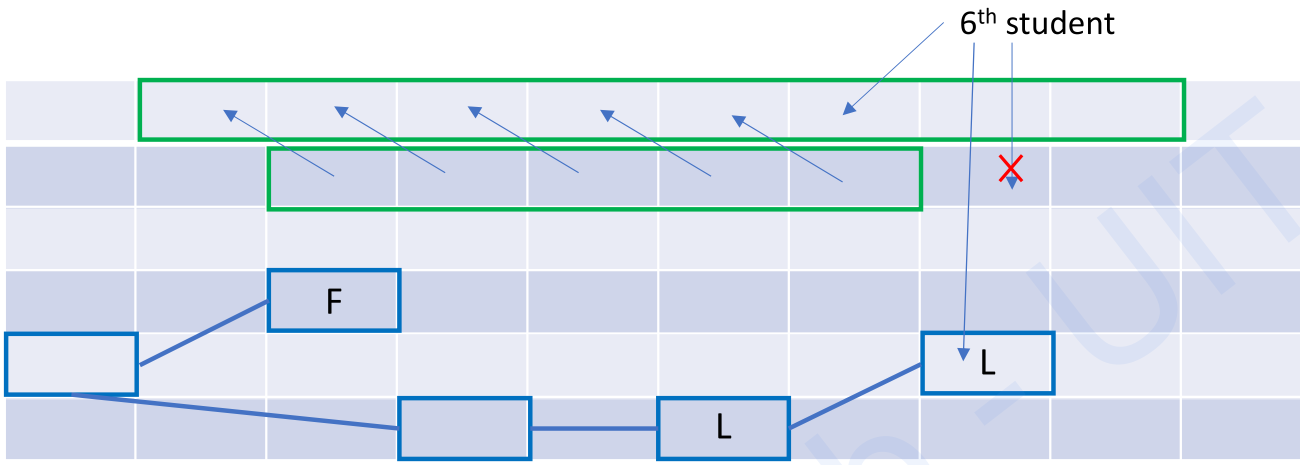
Array	Linked list
Random access i.e. efficient indexing	No random access (no index)
Sequential access is faster (continuous memory location)	Sequential access is slow (not continuous momory location)

Example: student list



Array	Linked list
Random access i.e. efficient indexing	No random access (no index)
Sequential access is faster (continuous memory location)	Sequential access is slow (not continuous momory location)
Memory waste (may happen)	No memory waste

Example: student list



Array	Linked list
Random access i.e. efficient indexing	No random access (no index)
Sequential access is faster (continuous memory location)	Sequential access is slow (not continuous momory location)
Memory waste (may happen)	No memory waste
Fixed size: resizing is expensive	Dynamic size
Insertion and deletion are inefficiesnt (need shifting all elements)	Insertion and deletion are efficient (no shifting)

What is an array?

Array is a collection of items of a single type.

[5, -2, 9, 300]
[“data”, “structure”, “and”, “algorithms”]
~~[1, “aaa”, 3, 4]~~

Woring with array

int sampleArray[5] = { 2, 5, 11, 3, 8}

sampleArray =

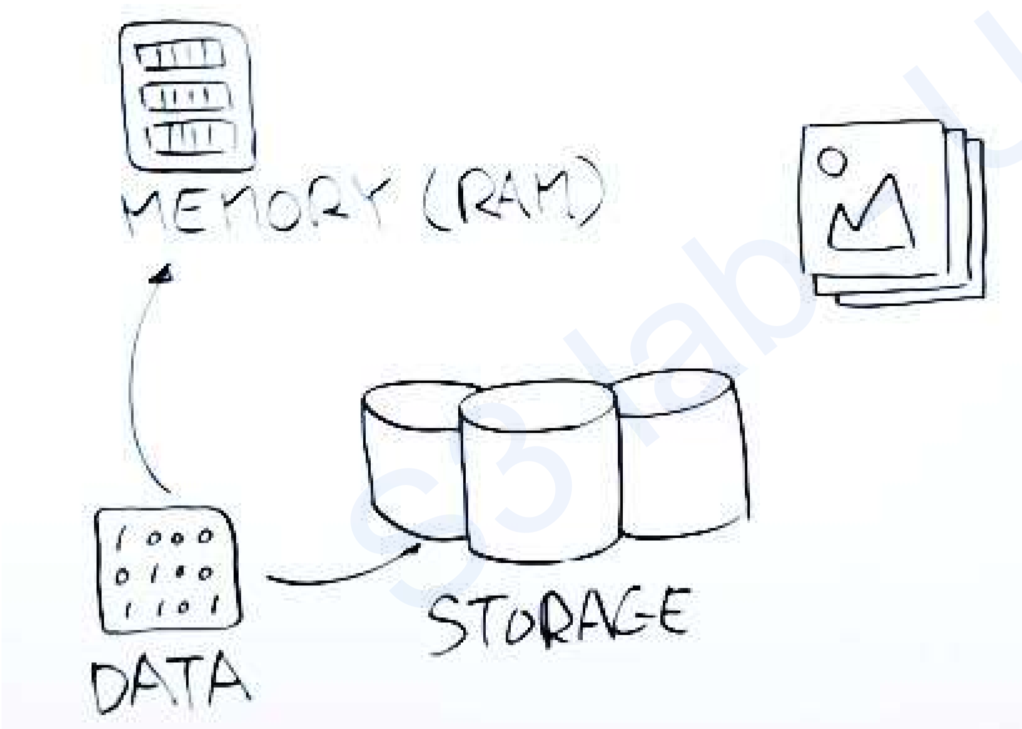
2	5	11	3	8

6	5	-2	3	8

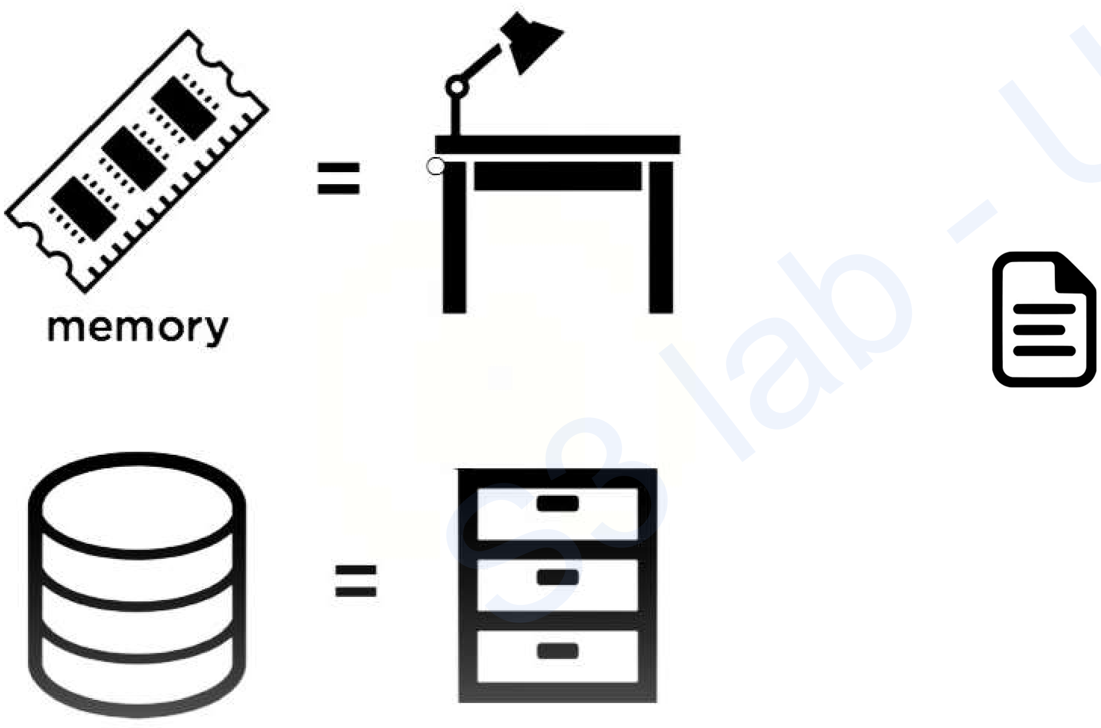
sampleArray[0] = 6

sampleArray[3] = -2

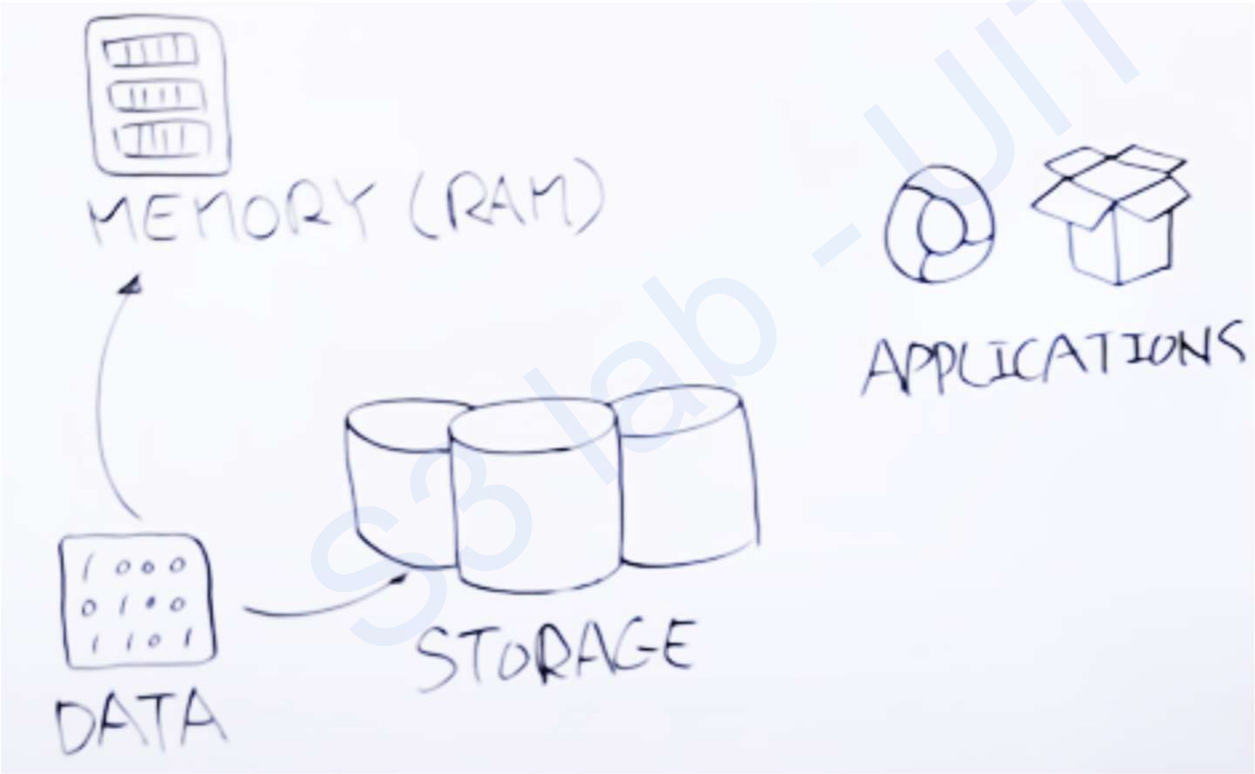
Memory vs storage



Memory vs storage



Memory vs storage

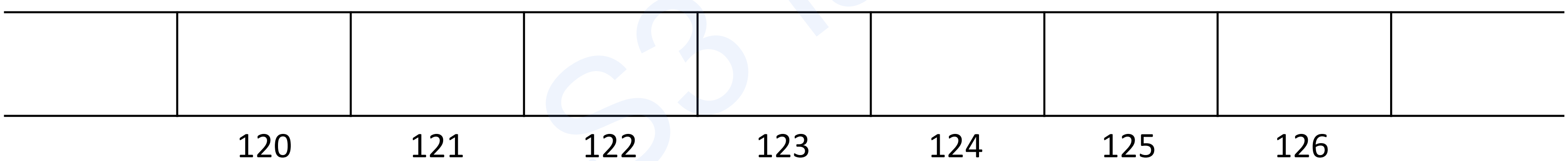


Data in memory

int a = 1;   1 -> 0000…001

2 -> $\underbrace{\text{0000…010}}_{\text{32 bits}}$

Memory is a long tape of bytes.



A byte = a small unit of data = 8 bits.

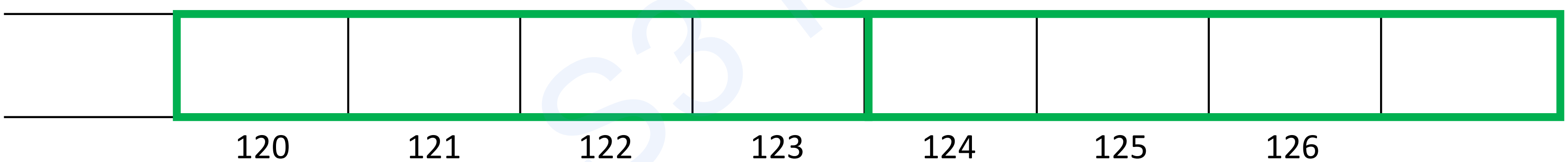
Data in memory

int a = 1;　　1 -> 0000…001

2 -> 0000…010

32 bits

Memory is a long tape of bytes.



A byte = a small unit of data = 8 bits.

Array in memory

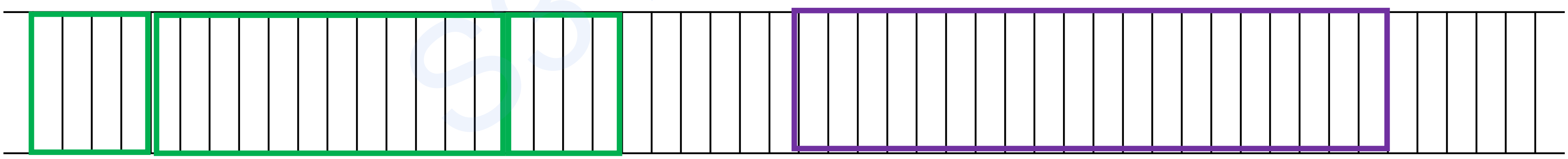
8

int a = 1;

int sampleArray[3] = { 3 , 2 , 5 };

int b = 0;

int newArray[5];



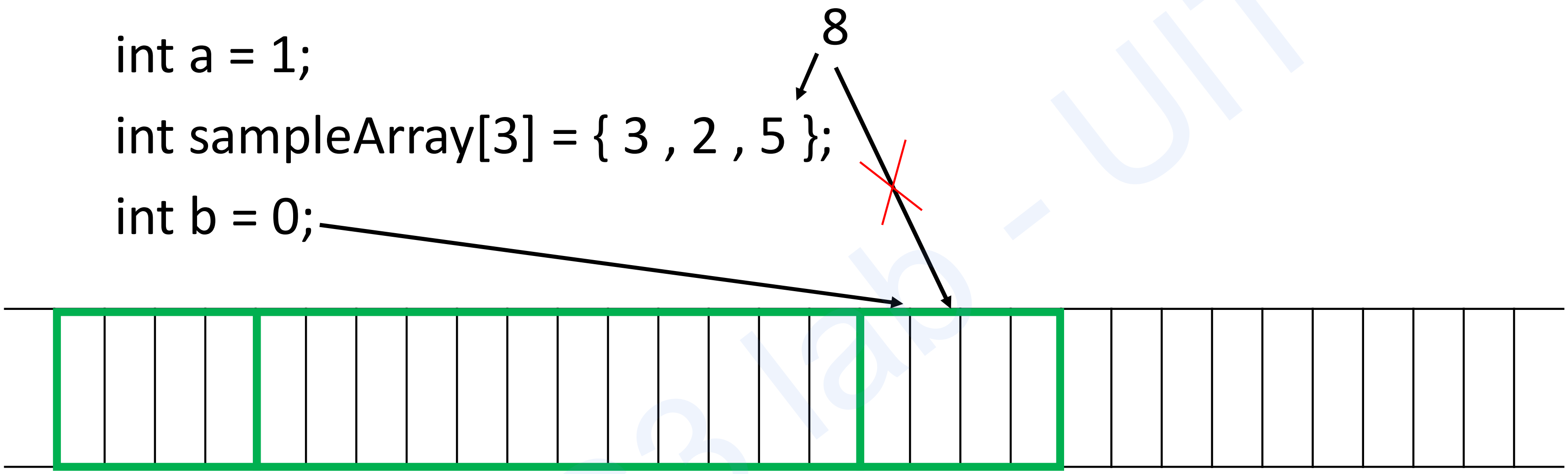
Creating new array and copy old data is the essence of resizable array in other languate.

Array in memory

int a = 1;

int sampleArray[3] = { 3 , 2 , 5 };

int b = 0;



Criteria of data structure

- Every data structure must satisfy these criteria
  - Convey all necessary information
  - Can be accessed and worked with in appropriate ways.
  - Suitable for specific algorithms (easy implementation, run fast, save memory)
  - Implementable (for programming languages)
  - Economic system resource

Need of Data Structures

As applications are getting complexed and amount of data is increasing day by day, there may arrise the following problems:

- **Processor speed**: To handle very large amout of data, high speed processing is required, but as the data is growing day by day to the billions of files per entity, processor may fail to deal with that much amount of data.
- **Data Search**: Consider an inventory size of 106 items in a store, If our application needs to search for a particular item, it needs to traverse 106 items every time, results in slowing down the search process.
- **Multiple requests**: If thousands of users are searching the data simultaneously on a web server, then there are the chances that a very large server can be failed during that process

In order to solve the above problems, data structures are used. Data is organized to form a data structure in such a way that all items are not required to be searched and required data can be searched instantly.

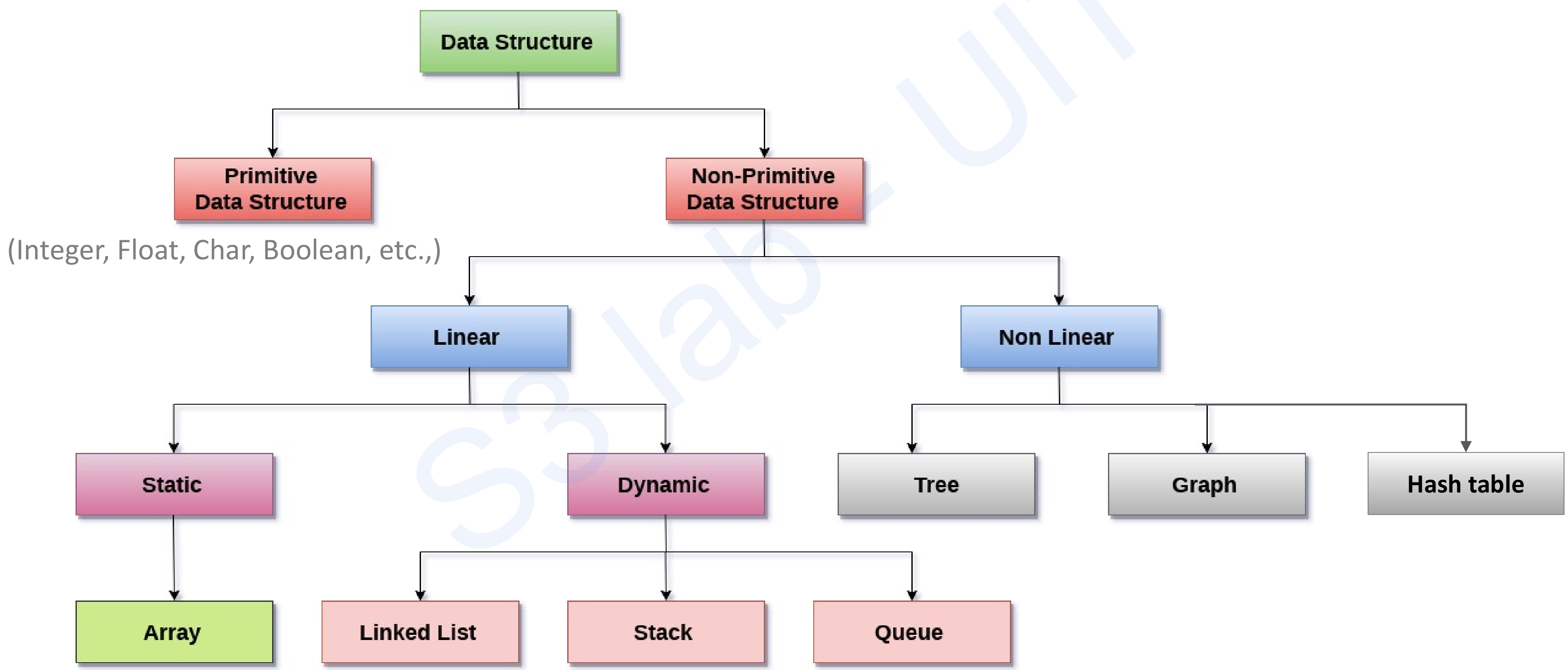
Advantages of Data Structures

- **Efficiency**: Efficiency of a program depends upon the choice of data structures. For example: suppose, we have some data and we need to perform the search for a perticular record. In that case, if we organize our data in an array, we will have to search sequentially element by element. hence, using array may not be very efficient here. There are better data structures which can make the search process efficient like ordered array, binary search tree or hash tables.

- **Reusability**: Data structures are reusable, i.e. once we have implemented a particular data structure, we can use it at any other place. Implementation of data structures can be compiled into libraries which can be used by different clients.

- **Abstraction**: Data structure is specified by the ADT which provides a level of abstraction. The client program uses the data structure through interface only, without getting into the implementation details.

Classification of data structures

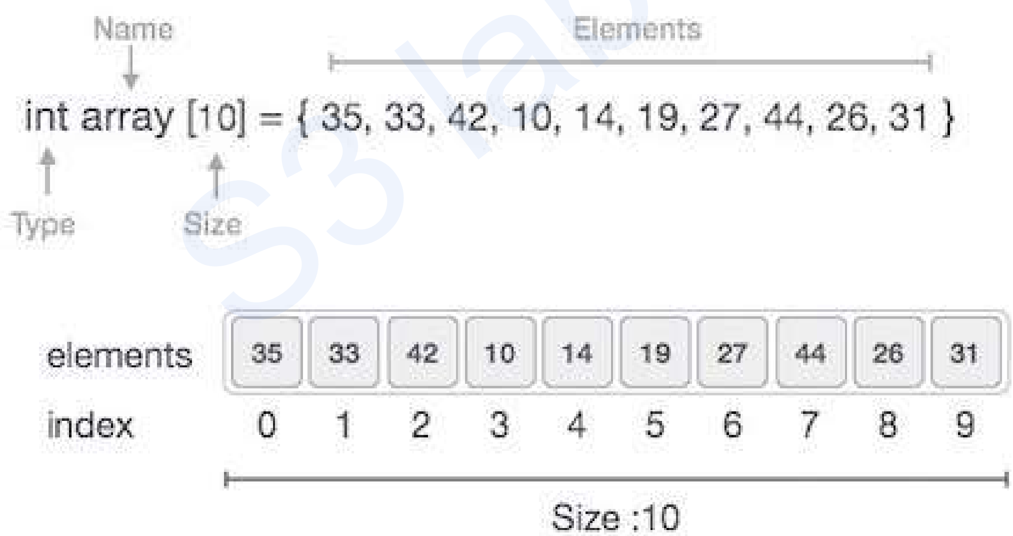


Linear data structure

Linear Data Structures: A data structure is called linear if all of its elements are arranged in the linear order. In linear data structures, the elements are stored in non-hierarchical way where each element has the successors and predecessors except the first and last element.

Linear data structure - Array

- **Arrays:** An array is a collection of similar type of data items and each data item is called an element of the array. The data type of the element may be any valid data type like char, int, float or double.
- The elements of array share the same variable name but each one carries a different index number known as subscript. The array can be one dimensional, two dimensional or multidimensional.

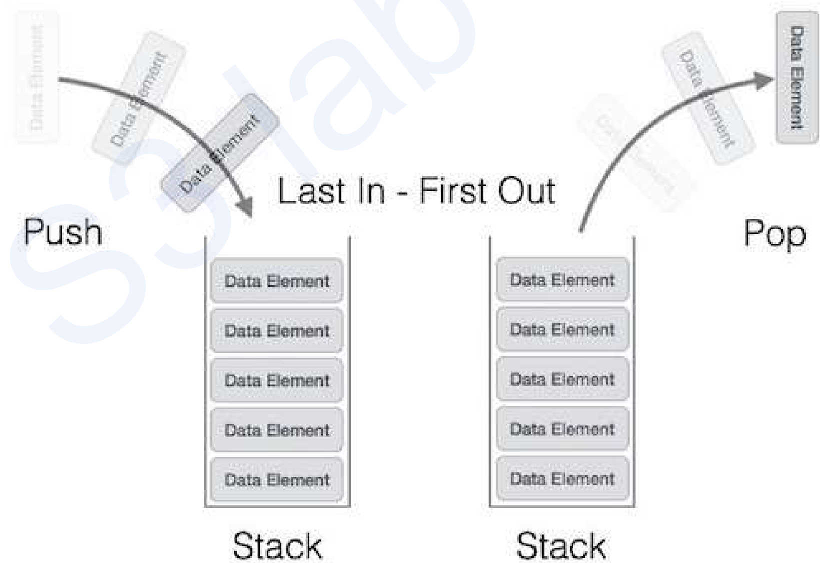


Linear data structure – Linked List

- **Linked List:** Linked list is a linear data structure which is used to maintain a list in the memory. It can be seen as the collection of nodes stored at non-contiguous memory locations. Each node of the list contains a pointer to its adjacent node.
- There are three types of linked list
  - Simple Linked List − Item navigation is forward only.
  - Doubly Linked List − Items can be navigated forward and backward.
  - Circular Linked List − Last item contains link of the first element as next and the first element has a link to the last element as previous.

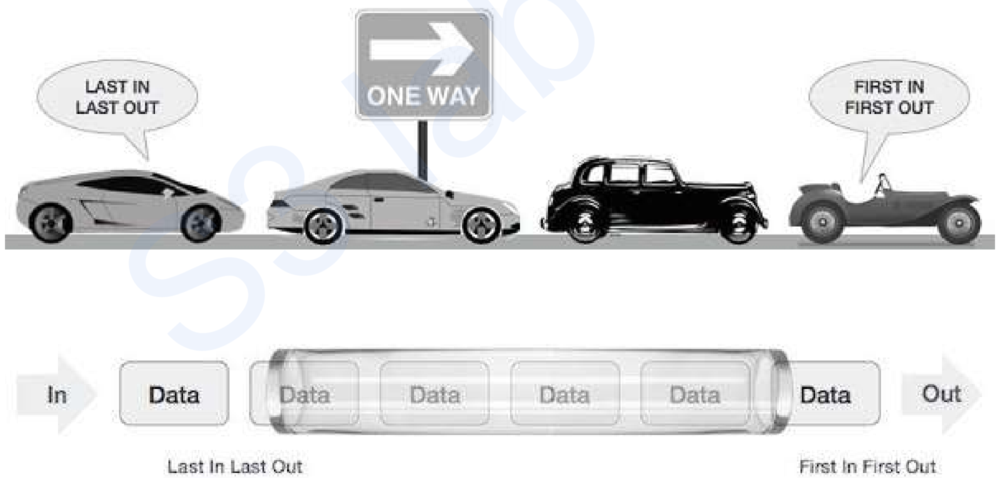
Linear data structure - Stack

- **Stack:** Stack is a linear list in which insertion and deletions are allowed only at one end, called **top**.
- A stack is an abstract data type (ADT), can be implemented in most of the programming languages. It is named as stack because it behaves like a real-world stack, for example: – piles of plates or deck of cards etc.



Linear data structure - Queue

- **Queue:** Queue is a linear list in which elements can be inserted only at one end called **rear** and deleted only at the other end called **front**.
- It is an abstract data structure, similar to stack. Queue is opened at both end therefore it follows First-In-First-Out (FIFO) methodology for storing the data items.



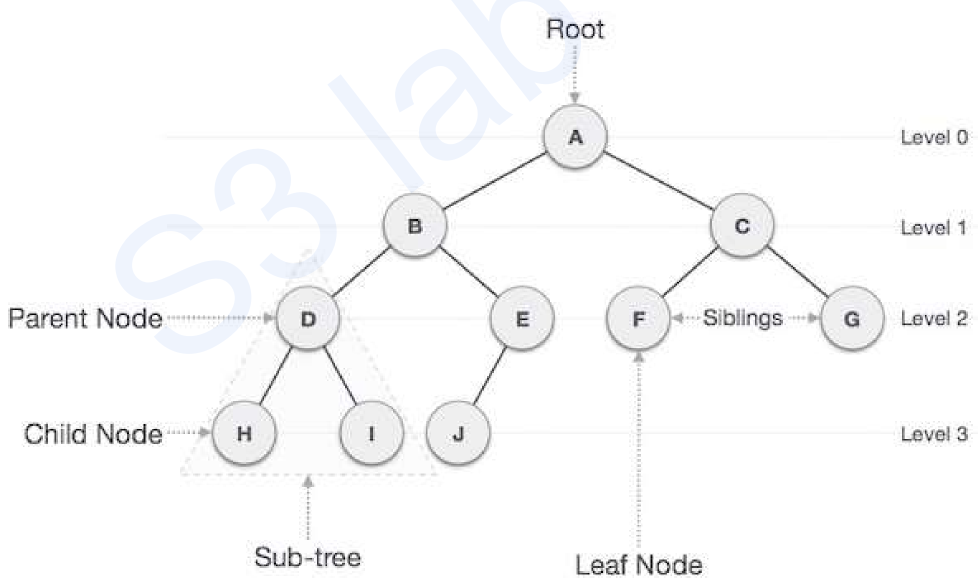
Queue

Non Linear Data Structures

Non Linear Data Structures: This data structure does not form a sequence i.e. each item or element is connected with two or more other items in a non-linear arrangement. The data elements are not arranged in sequential structure.

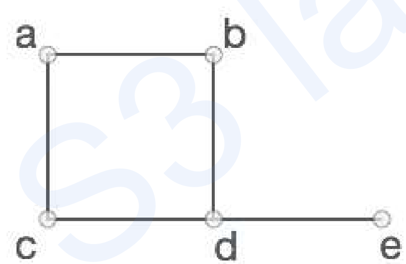
Non Linear Data Structures - Tree

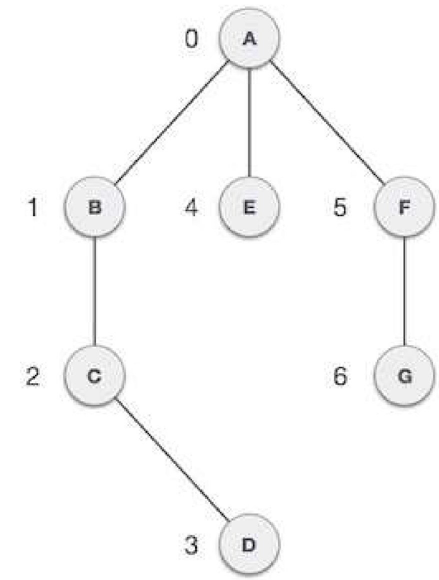
- **Trees:** Trees are multilevel data structures with a hierarchical relationship among its elements known as nodes. The bottommost nodes in the herierchy are called **leaf node** while the topmost node is called **root node**. Each node contains pointers to point adjacent nodes.
- Tree data structure is based on the parent-child relationship among the nodes. Each node in the tree can have more than one children except the leaf nodes whereas each node can have atmost one parent except the root node. Trees can be classfied into many categories which will be discussed later in this course.



Non Linear Data Structures - Graph

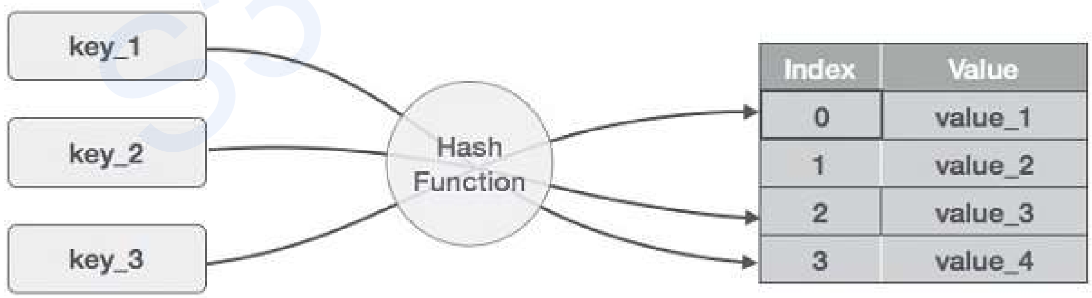
- Graphs can be defined as the pictorial representation of the set of elements (represented by **vertices**) connected by the links known as **edges**. A graph is different from tree in the sense that a graph can have cycle while the tree can not have the one.





Non Linear Data Structures – Hash table

- Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data.
- Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of the size of the data. Hash Table uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from.



Operation on data structures

The following operations are commonly performed on any data-structure

- **Insertion** − adding a data item
- **Deletion** − removing a data item
- **Traversal** − accessing and/or printing all data items
- **Searching** − finding a particular data item
- **Sorting** − arranging data items in a pre-defined sequence

Algorithms on data structure

From the data structure point of view, following are some important categories of algorithms

- **Search** − Algorithm to search an item in a data structure.
- **Sort** − Algorithm to sort items in a certain order.
- **Insert** − Algorithm to insert item in a data structure.
- **Update** − Algorithm to update an existing item in a data structure.
- **Delete** − Algorithm to delete an existing item from a data structure.

Criteria of algorithms

- Every Algorithm must satisfy the following properties:

1. **Input**- 0 or more inputs.
2. **Output**- 1 or more outputs.
3. **Definiteness**- Every step of the algorithm should be clear and well defined.
4. **Finiteness**- The algorithm should have finite number of steps.
5. **Correctness**- Every step of the algorithm must generate a correct output.
6. **Feasibleness** - Feasible with specified computational device
7. **Independent** – An algorithm should have step-by-step directions, which should be independent of any programming code.

Espressing and algorithm

Algorithms can be expressed in many kinds of notation, including natural languages, pseudocode, flowcharts (and other software generated chart like drakon-charts, lucidchart), programming languages or control tables (processed by interpreters)

Example: Pig latin algorithm

Pig latin algorithm: Translate the provided string to pig latin.

- Pig Latin algorithm takes the first consonant (or consonant cluster) of an English word, moves it to the end of the word and suffixes an "ay".

- If a word begins with a vowel you just add "way" to the end.

Expressing using Natural language

- Advantage
  - Natural to human
  - Can convey steps of an algorithms to a wide audience (including programmer and non-programmer)
- Drawbacks
  - Has a tendency to be ambiguous and too vaguely defined (since it has no imposed structure).
  - Difficult for others to follow the algorithm and feel confident in its correctness.
  - Less structured formats compared to flow charts and pseudocode (which can more precisely express an algorithm).

Expressing using Pseudocode

- Programmers often like to express an algorithm in pseudocode: code that uses all the constructs of a programming language, but doesn't actually run anywhere.
- Every programmer writes pseudocode differently, since there is no official standard, so you may run into pseudo-code that looks very different.
- Expressing an algorithm in pseudocode helps a programmer think in familiar terms without worrying about syntax and specifics. It also gives computer scientists a language-independent way to express an algorithm, so that programmers from any language can come along, read the pseudo-code, and translate it into their language of choice.

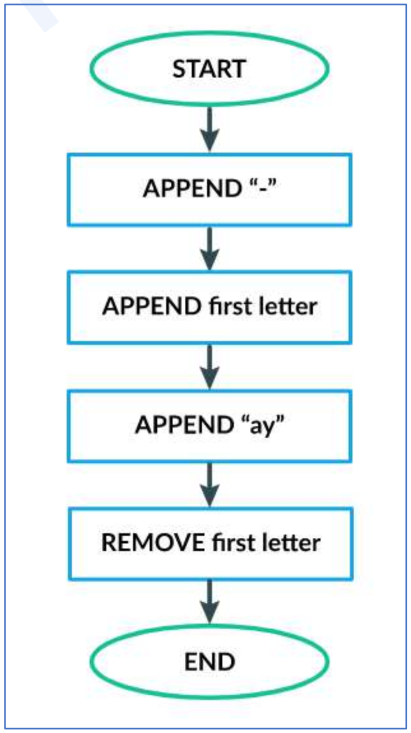
Pig Latin algorithm

```
FOR EACH word IN words
{
  APPEND(word, "-")
  letter ← FIRST_LETTER(word)
  IF (IS_VOWEL(letter)) {
    APPEND(word, "yay")
  } ELSE {
    APPEND(word, letter)
    APPEND(word, "ay")
    REMOVE_FIRST(word)
  }
}
```

Expressing using Flow chart

- A more formal way to express an algorithm is with a flow chart, a diagram with boxes connected by arrows.
- Each rectangle represents a step in the sequence, and the arrows flow from one step to the next.

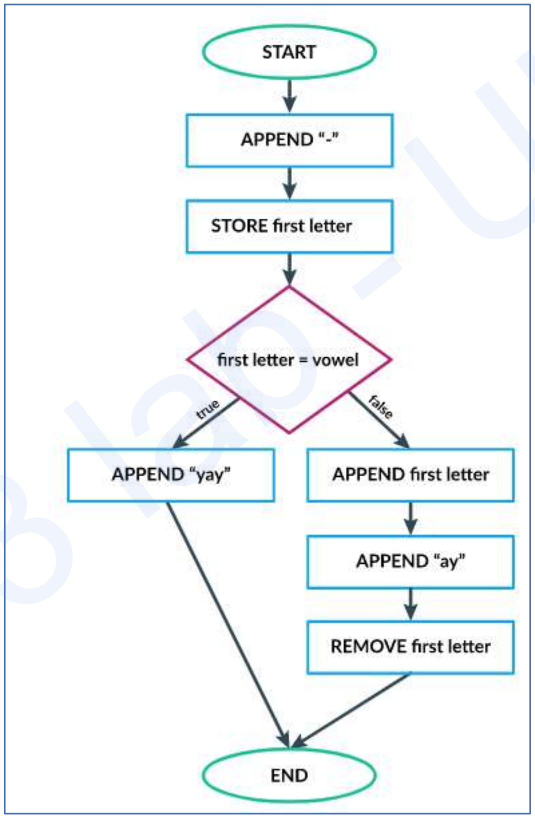
Simple version



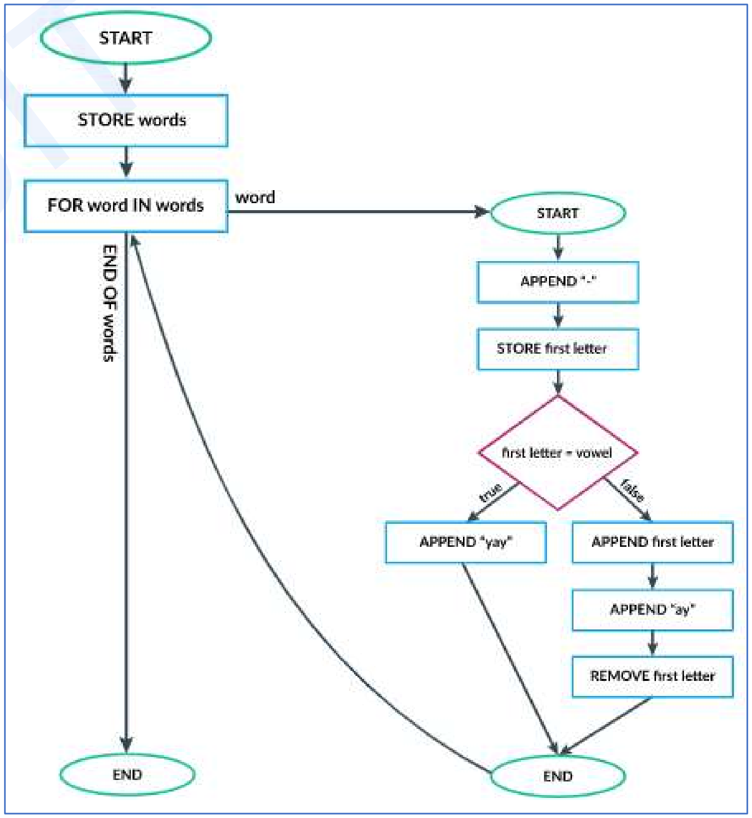
Expressing using Flow chart

- Expressing an algorithm in a flow chart allows us to visualize the algorithm at a high level, plus it forces us to think very carefully about sequencing and selection. Which arrow goes to what node? Are there missing arrows? Those are the kinds of valuable questions that can come up while translating an algorithm into a flow chart.

Improved version

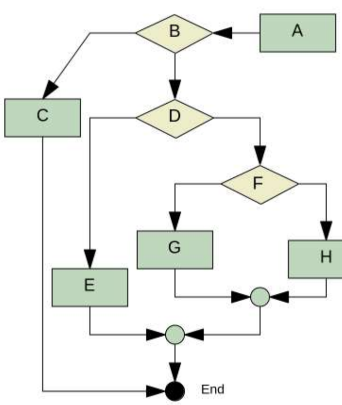


Complete version

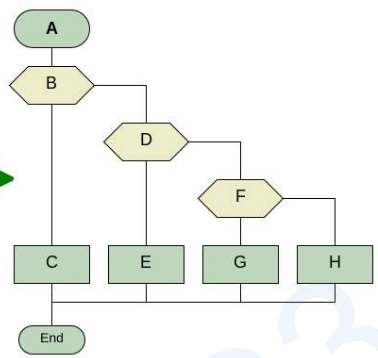


Expressing using software generated chart

Messy flow chart



Drakon chart



	Icon	Name of Icon
1		Title
2		End
3		Action
4		Question
5		Choice
6		Case
7		Headline
8		Address
9		Insertion
10		Shelf
11		Formal parameters
12		Begin of FOR loop
13		End of FOR loop

	Icon	Name of Icon
14		Output
15		Input
16		Pause
17		Period
18		Start timer
19		Synchronizer
20		Realtime parallel process
21		Comment
22	XXX XXX XXX	Right comment
23	XXX XXX XXX	Left comment
24		Loop arrow
25		Silhouette arrow
26		Connector
27		Concurrent process

	Macroicon	Name of Macroicon
1		Title with parameters
2		Fork
3	N = 2 N > 2	Switch (number of cases N >= 2)
4		SIMPLE loop
5		SWITCH loop
6		FOR loop
7		WAIT loop
8		Action by timer
9		Shelf by timer
10		Fork by timer

	Macroicon	Name of Macroicon
11		Switch by timer
12		SIMPLE loop by timer
13		SWITCH loop by timer
14		FOR loop by timer
15		WAIT loop by timer
16		Insertion by timer
17		Output by timer
18		Input by timer
19		Start timer by timer
20		Parallel process by timer
21		TREE loop

Expressing using programming language

```
// Returns true if the given character is an English vowel
function isVowel(char) {
    var vowels = ['a', 'e', 'i', 'o', 'u'];
    return vowels.indexOf(char.toLowerCase()) !== -1;
}

// Pig Latin algorithm
var words = ["peanut", "butter", "and", "jelly"];
for (var i = 0; i < words.length; i++) {
    var word = words[i];
    word += "-";
    var firstLetter = word.charAt(0);
    if (isVowel(firstLetter)) {
        word += "yay";
    } else {
        word += firstLetter;
        word += "ay";
        word = word.slice(1);
    }
    words[i] = word;
}

// Display magnificent result
println(words);
```

Thank you!