CHARUTAR VIDYA MANDAL'S
SEMCOM
Vallabh Vidyanagar

Faculty Name: Ami D. Trivedi
Class: FYBCA
Subject: US02CBCA01
(Advanced C Programming and Introduction to Data Structures)

*UNIT – 3 (Introduction to Data Structures)

*MEANING & USAGE OF DATA STRUCTURES

Representing information is fundamental to computer science. The primary purpose of most computer programs is **NOT** to perform calculations, but to store and retrieve information--- usually as fast as possible. For this reason, the study of data structures and the algorithms that manipulate information is the heart of computer science.

A data structure is used to organize the information in computer such a way that it satisfies our needs. A data structure is a specialized format for organizing and storing data.

A data structure is any data representation and its associated operations. Even an integer or floating point number stored on the computer can be viewed as a simple data structure. More typically, a data structure means an organization or structuring for a collection of data items. A sorted list of integers stored in an array is an example of such a structuring.

Any data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways. In computer programming, a data structure may be selected or designed to store data for the purpose of working on it with various algorithms.

A data structure is used to provide efficient processing for structuring information in to memory.

Data Structure

Definition: "It is a way of organizing data in the memory that considers not only the data items stored but also their relationship to each other".

Storage structure

Definition: The representation of a particular data in memory of a computer is called a storage structure.

File structure:

Definition: A storage structure representation in secondary memory is called a file structure.

APPLICATIONS OF DATA STRUCTURE

• Compiler Design • Operating System • Database Management System • Statistical Analysis Packages • Numerical Analysis	• Graphics • Artificial Intelligence • Simulation • Network Analysis

ADVANTAGES OF DATA STRUCTURE:

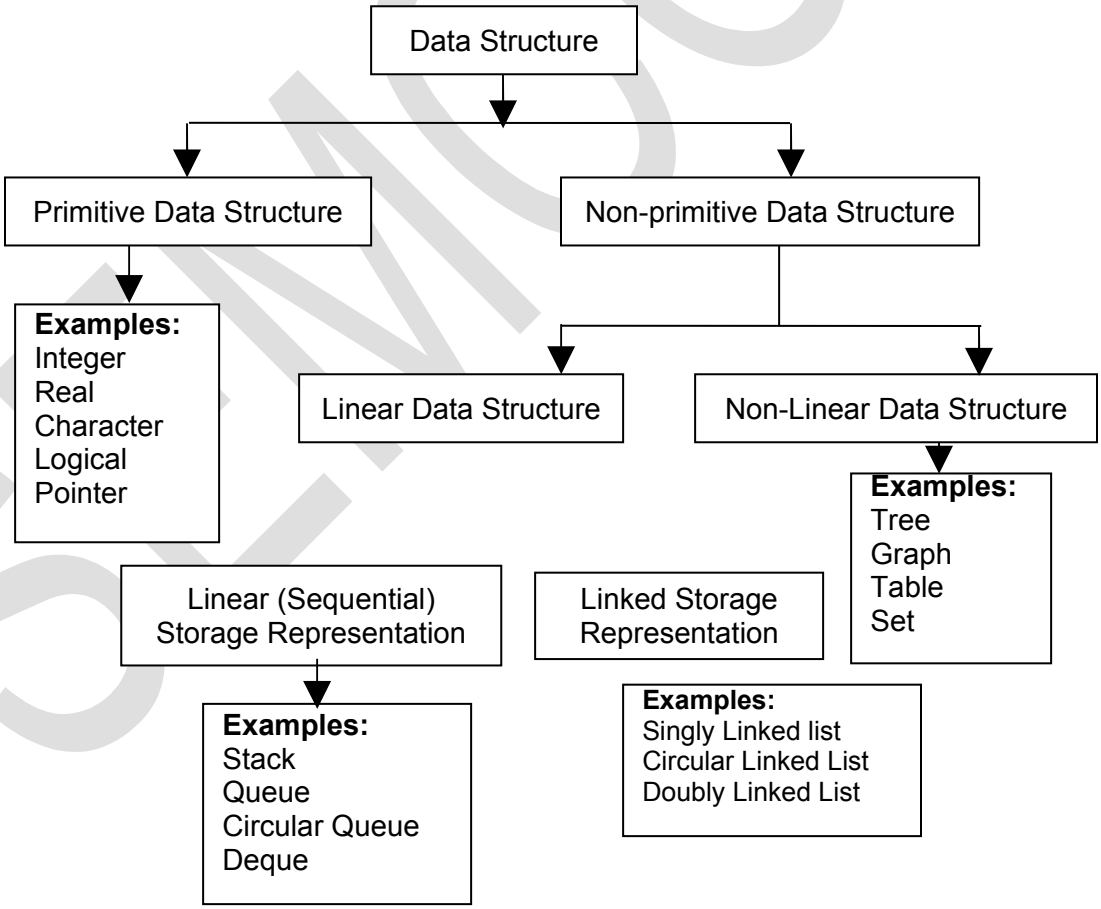
The major advantages of data structure are:

1. It gives different level of organizing data.
2. It tells how data can be stored and accessed in its elementary level.

CLASSIFICATION OF DATA STRUCTURE

The data structure is classified into mainly two categories:

1. Primitive data structure
2. Non-primitive data structure



Definition: Primitive Data Structure

The data structure that typically are directly operated upon by machine level instructions. In other words, the structuring of data at the most primitive level within computer is known as primitive data structure.

e.g. integer , real, character, logical and pointer information are primitive data structures.

Definition: Non - Primitive Data Structure

A complex or non-primitive data structure consists of structured set of primitive Data structure. The non primitive data structure are not directly operated by machine level instructions.

e.g. Array, Stack, Queue, Lists, File etc. are non primitive data structure.

Difference between Primitive and Non primitive data structure:

	Primitive data type		**Non-primitive data type**
1.	A data type that is directly operated by machine level instruction is known as primitive data type.	**1.**	A data type that is structured set of primitive data types is known as Non- primitive data type.
2.	E.g. Integer ,real, character ,logical and pointer	**2.**	E.g. Array, stack, Queue, Linked list and file
3.	It is a fundamental data type	**3.**	It is a set of primitive data type
4.	There is no other name	**4.**	It is also known as composite or complex data type

OPERATIONS ON DATA STRCUTURE

There are four basic operations that are applied to the data structures. These are creation, destroy, selection and updation.

CREATION

An operation that is used to create a data structure is known as creation operation. A variable can be created in C, PASCAL, FORTRAN, ALGO, PL\I plus and many other languages using declaration statement.

For example in C language, the following declaration statement
int n;
creates a variable n. It also occupies the required memory to the variable n. In this case 2 bytes are occupied by the variable n.

DESTROY

An operation that provides complementary (opposite) effect of CREATION operation is known as destroy operation.

Certain languages like FORTRAN do not allow a programmer to destroy a data structure, once they are created. Some languages such as C, C++. PASCAL, PL\I allow programmer to destroy a data structure.

In C language the following statement

```
free (ptr); // It is previously declared as int *ptr; in a program.
```

will release the memory pointed by variable 'ptr'.

SELECTION

An operation which is used to access data within the data structure is known as selection. The form of selection depends upon the type of data structure being accessed.

In C language, the following statement

```
int a[5];
```

declares an array of five elements. Now, we want to display an element from 4^{th} position on the screen using following statement.

```
printf ("%d", a[3] );
```

It is a selection operation. This is because we access the elements within array data structure.

UPDATE

An operation that is used to change data in the data structure is known as UPDATE operation.

e.g. An assignment statement (=) is a good example of update operation.

In C, the following statement

```
sum = a + b ;
```

will total the value of variables a and b and stores the result into sum variable.

CATEGORIES OF NONPRIMITIVE DATA STRUCTURE

A non primitive data structure is classified into two categories:

1. Linear data structures
2. Non-Linear data structures

1. LINEAR DATA STRUCTURES

A data structure is said to be linear if all the elements are traversed sequentially.

All the elements of a linear data structure will form a sequence. A linear data structure maintains a linear structure. Therefore, we can traverse either forward or backward from a node.

There are two ways of representing linear data structures in memory.

One way is to have the linear relationship between the elements by means of sequential memory locations. Such linear structures are stack, queue and array.

The other way is to have the linear relationship between the elements represented by means of links. Such linear data structure is linked list.

Examples of Linear Data Structure are:

- Array
- Stack
- Queue
- Linked List

2. NON LINEAR DATA STRCUTURES

A data structure is said to be non linear if all the elements are NOT traversed sequentially.

All the elements of a non-linear data structure will not form a sequence. A non-linear data structure does not maintain a linear structure. It branches to more than one node and cannot be traversed in a single run.

Examples of Non- Linear Data Structure are:

- Trees
- Graphs
- Tables
- Sets

Priority Queue (Definition)

A priority queue is a collection of elements such that each element has been assigned a priority and the order in which elements are deleted and processed with following rules:

- An element of higher priority is processed before any element of lower priority.
- Two elements with same priority are processed according to order in which they are added to the queue.

*STACKS

One of the most important linear data structures of variable size is the stack. In linear list, we are allowed to delete an element from any position in the list and insert an element to any position in the list.

An important type of lists allows the insertion or deletion of an element to occur only at one end. This is called a stack.

Definition: Stack is a linear non primitive data structure in which insertion and deletion of elements occur only at one end.

The insertion operation is referred to as "**push**" and the deletion operation as "**pop**". The most accessible element of a stack is known as **top** element and least accessible element of a stack is known as **bottom** element.

Since insertion and deletion operations are performed at one end of a stack, the element can be removed in the opposite order from that in which they were added to the stack. This phenomenon (event) will be observed in conjunction with recursive functions. Such a linear list is frequently referred to as a **LIFO** (last-in-first-out) list.

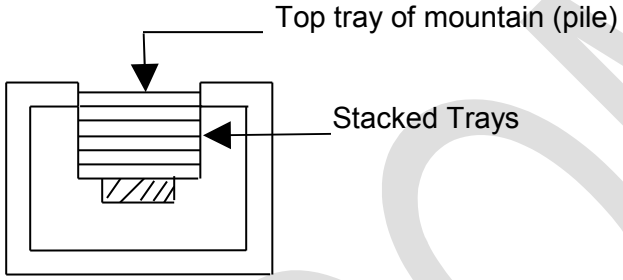


Fig: Representation of a stack

A stack can be represented by a vector as follow:

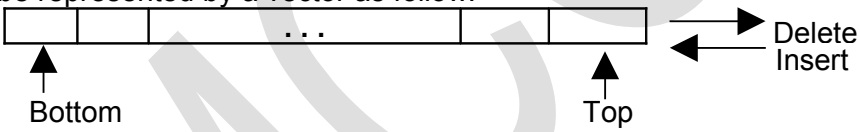


Fig: Representation of a stack by a vector

EXAMPLES OF STACK

A common example of a stack phenomenon (event), which permits the selection of only end element, is a mountain of trays in a cafeteria.

A person desiring a tray finds that only one is available to him or her at the surface (top) of the tray counter. The removal of the tray causes the load on the spring to be lighter and next tray to appear at the surface of the counter.

A tray which is placed on the pile causes the entire pile to be pushed down and that tray to appear above the tray counter. Such an arrangement of trays is shown in above fig.

Another familiar example of a stack is railway system for shunting cars. In this system, the last car to be placed on the stack is the first to leave. Using repeatedly the insertion and deletion operations permits the car to be arranged on the output railway line in various orders.

CHARACTERISTICS

A stack data structure has following characteristics:

- It is non-primitive data structure.
- Its nature is LIFO (Last In First Out).

- The insertion and deletion operation occur only at one end.
- It is linear data structure of variable size.
- Elements are removed in opposite order from the order in which they are added to stack.

OPERATIONS ON STACKS

Operations on a stack are simulated (pretended) by using a vector (array). This vector is consisting of some large number of elements. Elements should be sufficient to handle all possible insertions which are expected to be made to the stack.

A pointer **TOP** keeps track of the top element in the stack.

Initially, when the stack is empty, TOP has a value of zero. When the stack contains a single element, TOP has a value of "one", and so on. Each time a new element is inserted in the stack, the pointer TOP is incremented by "one" before the element is placed on the stack.

The pointer is decremented by "one" each time a deletion is made from the stack.

Following operations can be performed on Stack.

1. Push	An insertion operation is known as Push. Used to insert an element on a stack.
2. Pop	A deletion operation is known as Pop. Used to delete an element from a stack.
3. Peep	To find the value of **ith** element from a stack.
4. Change	To change the value of **ith** element from a stack.

ALGORITHMS

1. PUSH (S, TOP, X)

This function inserts element X at the top of stack.

S	vector containing N elements
X	element to be inserted to the top of a stack
TOP	a pointer denoting the top element in the stack

1.	[Check for stack overflow] If TOP ≥ N then Write ('STACK OVERFLOW') Return
2	[Increment TOP] TOP ← TOP + 1
3.	[Insert element] S [TOP] ← X
4.	[Finished]

	Return

The first step of this algorithm checks for an **overflow** condition. If such a condition exists, then the insertion cannot be performed and an appropriate error message results.

2. POP (S, TOP)

This function removes the top element from a stack and returns this element.

S	vector containing N elements
TOP	a pointer denoting the top element in the stack

1.	[Check for underflow on stack] If TOP = 0 then Write ('STACK UNDERFLOW ON POP') take action in response to underflow Exit
2.	[Decrement pointer] TOP ← TOP + 1
3.	[Return former top element of stack] Return (S [TOP + 1])

An **underflow** condition is checked for in the first step of the algorithm. If there is an underflow, then some appropriate action should take place.

3. PEEP (S, TOP, I)

This function is used to find the value of the **i**th element from the top of a stack. The element is not deleted by this function.

S	vector containing N elements
TOP	a pointer denoting the top element in the stack
I	find value of **i**th element from top of stack

1.	[Check for stack underflow] If TOP - I + 1 ≤ 0 then Write ('STACK UNDERFLOW ON PEEP') take action in response to underflow Exit
2.	[Return former top element of stack] Return (S [TOP – I + 1])

4. CHANGE (S, TOP, X, I)

This function changes the contents of the **i**th element from the top of a stack.

S	vector containing N elements
TOP	a pointer denoting the top element in the stack
X	contain value which will be changed in ith element
I	value of **i**th element from top of stack will be changed

1.	[Check for stack underflow] If TOP – I + 1 = 0 then Write ('STACK UNDERFLOW ON CHANGE') Return
2.	[Change the element from the top of the stack] S [TOP – I + 1] ← X
3.	[Finished] Return

APPLICATIONS OF STACKS (LIST ONLY)

- Recursion
- Polish expression & their compilation
- Stack machine

Disclaimer

The study material is compiled by Ami D. Trivedi. The basic objective of this material is to supplement teaching and discussion in the classroom in the subject. Students are required to go for extra reading in the subject through library work.