**Stack**:

A Stack is a Last in, First Out (LIFO) data structure. This means that items are added to the top and removed from the top.

Stacks are used in calculations, and to hold return addresses when subroutines are called.

A Stack may be implemented as either a static or dynamic data structure.

|  |  |
| --- | --- |
| 5 |  |
| 4 |  |
| 3 | D (top of stack) |
| 2 | C |
| 1 | B |
| 0 | A |

**Operations on a stack**:

|  |  |
| --- | --- |
| push(item) | Adds a new item to the top of the stack |
| pop() | Removes and returns the top item from the stack |
| peek() | Returns the top item from the stack but does not remove it |
| isEmpty() | Tests to see whether the stack is empty and returns a Boolean value |
| isFull() | Tests to see whether the stack is full and returns a Boolean value. |

**Advantages of the Stack:**

* Data is accessed by a single memory location, which is the top of the stack.
* The Stack makes best use of processor time. Adding or Deleting data is always a single operation O(1). No matter how big the stack is.
* Stack makes the best use of memory space as there no gaps in the data storage.

**Disadvantages of the Stack:**

* It is an inflexible data structure as you can only access it in one place.
* You can only push & pop.
* You cannot search or sort.
* The computer can’t insert, edit or delete data anywhere but in one place, which is the top.
* The computer cannot traverse this data structure.