ADT

Abstract Data Type

- Defined as a "class of objects whose logical behavior is defined by **a set of values and a set of operations“**
- The definition of ADT only mentions **what operations are to be performed but not how these operations will be implemented.**
- It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.

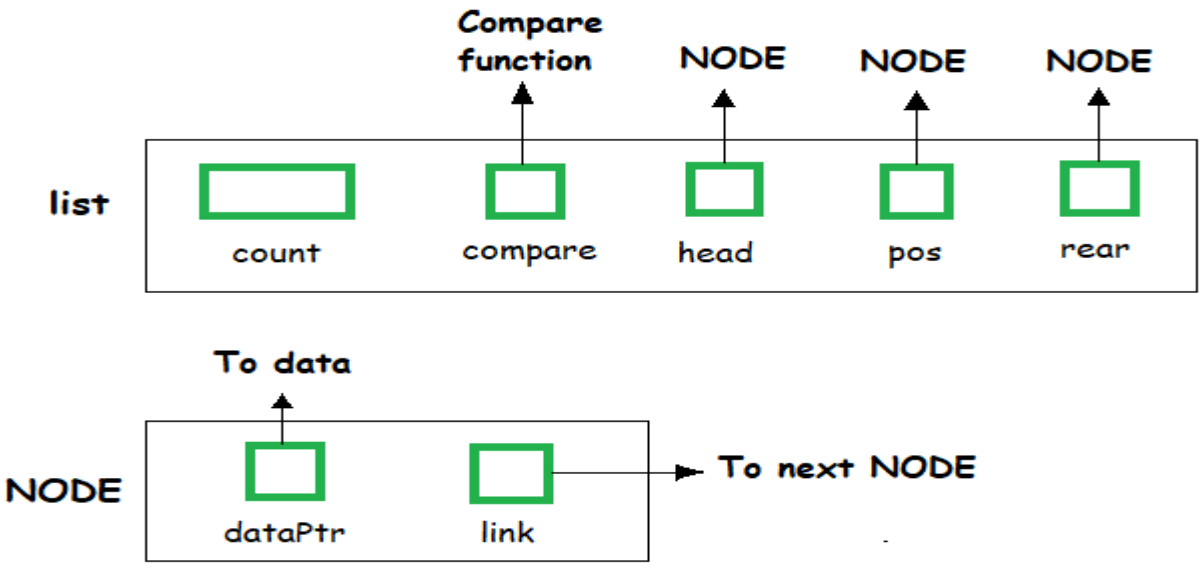
Courtesy: https://www.geeksforgeeks.org/abstract-data-types/

Abstract Data Type

- Called "abstract" because it gives an implementation-independent view.
  - The process of providing only the essentials and hiding the details is known as abstraction.
- Think of ADT as a black box which hides the inner structure and design of the data type.

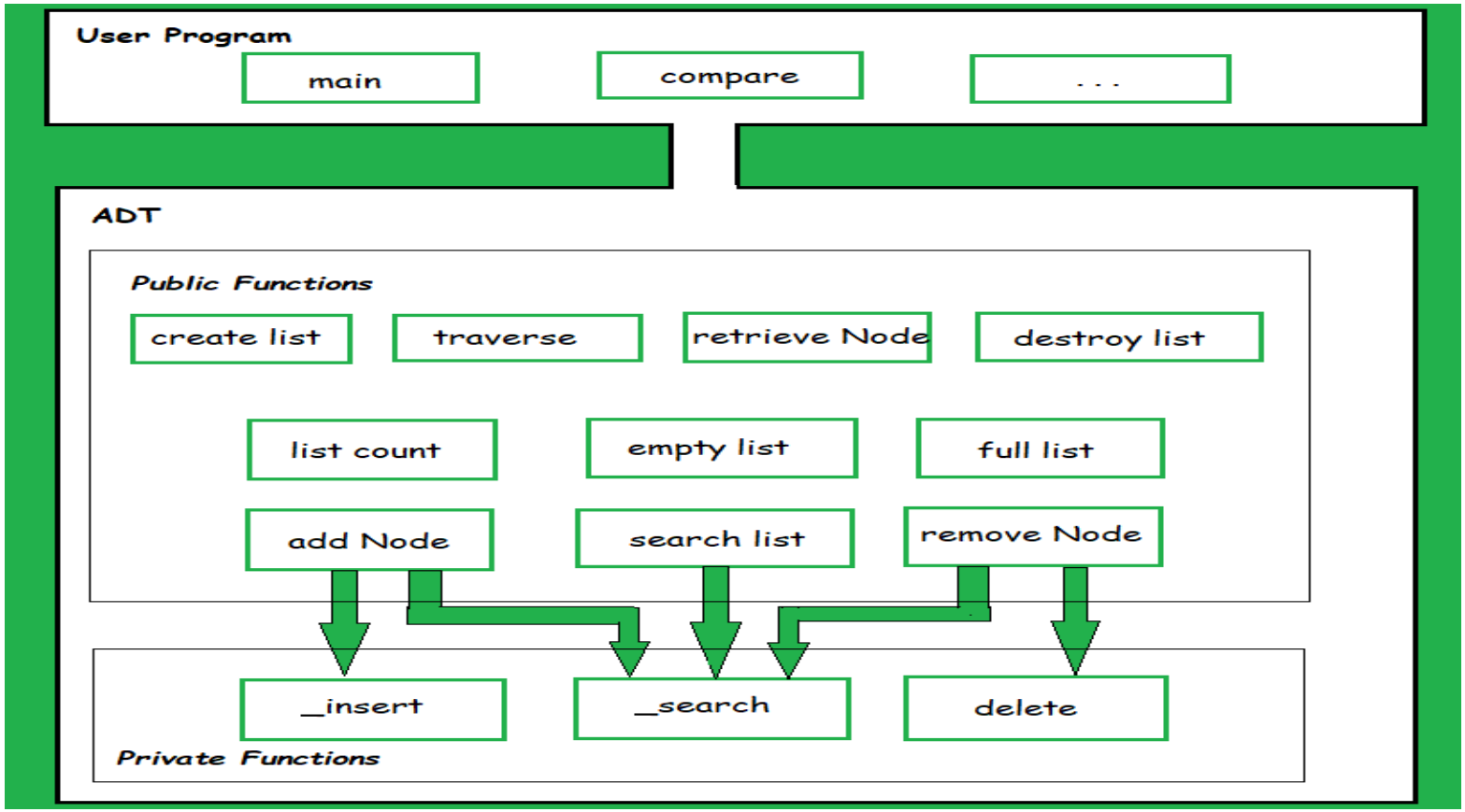
List ADT

- The data is generally stored in key sequence in a list which has a head structure consisting of *count*, *pointers* and *address of compare function* needed to compare the data in the list.



List ADT

- List ADT Functions

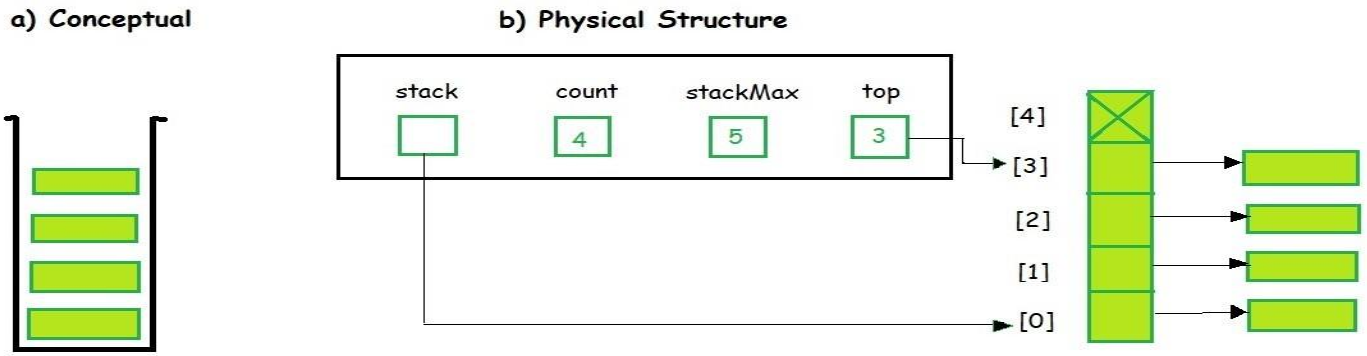


List ADT

- List ADT Functions
- **get() – Return an element from the list at any given position.**
- **insert() – Insert an element at any position of the list.**
- **remove() – Remove the first occurrence of any element from a non-empty list.**
- **removeAt() – Remove the element at a specified location from a non-empty list.**
- **replace() – Replace an element at any position by another element.**
- **size() – Return the number of elements in the list.**
- **isEmpty() – Return true if the list is empty, otherwise return false.**
- **isFull() – Return true if the list is full, otherwise return false.**

Stack ADT

- Instead of data being stored in each node, the pointer to data is stored.
- The program allocates memory for the *data* and *address* is passed to the stack ADT.



Stack ADT

- The head node and the data nodes are encapsulated in the ADT. The calling function can only see the pointer to the stack.
- The stack head structure also contains a pointer to *top* and *count* of number of entries currently in stack.

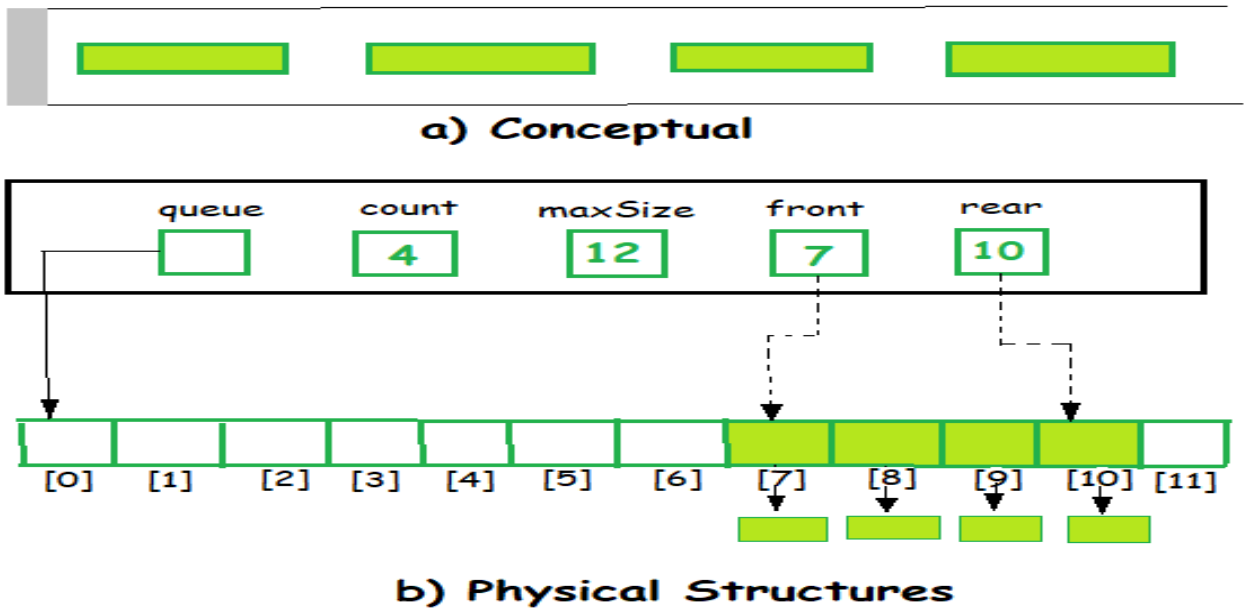
Stack ADT

- A Stack contains elements of the same type arranged in sequential order.
- All operations take place at a single end that is top of the stack and following operations can be performed:

- **push() – Insert an element at one end of the stack called top.**
- **pop() – Remove and return the element at the top of the stack, if it is not empty.**
- **peek() – Return the element at the top of the stack without removing it, if the stack is not empty.**
- **size() – Return the number of elements in the stack.**
- **isEmpty() – Return true if the stack is empty, otherwise return false.**
- **isFull() – Return true if the stack is full, otherwise return false.**

Queue ADT

- The queue abstract data type (ADT) follows the basic design of the stack abstract data type.



- Each node contains a void pointer to the *data* and the *link pointer* to the next element in the queue. The program's responsibility is to allocate memory for storing the data.

Queue ADT

- A Queue contains elements of the same type arranged in sequential order.
- Operations take place at both ends, insertion is done at the end and deletion is done at the front.

Queue ADT

- Following operations can be performed:

- **enqueue() – Insert an element at the end of the queue.**
- **dequeue() – Remove and return the first element of the queue, if the queue is not empty.**
- **peek() – Return the element of the queue without removing it, if the queue is not empty.**
- **size() – Return the number of elements in the queue.**
- **isEmpty() – Return true if the queue is empty, otherwise return false.**
- **isFull() – Return true if the queue is full, otherwise return false.**