* The stack is initialized to (-1). Then check is performed to determine if the stack is empty by comparing top to (-1). As element are added to the stack, the position of top is updated. As soon As elements are popped or deleted, the topmost elem is removed and the position of top is updated.

Tutorials (3) 2023/05/18

(1) What is a Linear search, Explain.

A Linear search is checking each element on a list in a sequential manner. It starts with one end to another until it's desired element is found. Until then the whole list has been searched.

(2) What is Binary search, Explain.

It is defined as a searching algorithm used in a sorted array by repeatedly dividing the search intervals in half. There are two forms of Binary search; They are Iterative and recursive methods.

(3) Compare and contrast Linear vs Binary Search.

	Linear search	vs	Binary search
precondition	- Any random order		Sorted
size	- Preffered for small lists.		Preferred for large lists
speed	- Slow		fast
Dimensions	- Multidimensional arrays can also be used.		Only single dimensional arrays can be used.
Time Complexity	- Linear graph O(n)		Logarithemic graph O(log n)
	• Access data sequentially		• Access data randomly
	• perform equal comparisons		• perform ordering comparisons.

(4) Write the pseudocode for Linear search.

```
LinearSearch (item_element)
{
    INITIALIZE index = 0
    WHILE (index < number of items in the list)
{
        If (list[index] == target element)
        {
            Return index
        }
        increment index by 1
    }
    return -1
}
```

(5) Write the pseudocode for binary search.

Algorithm to search a[First] through a[Last]

```
/**
    Precondition:
    a[First] <= a[First+1] <= a[First+2] <= ... <= a[last]
*/
```

To locate the value key:

```
if (first > last)  // A Stopping case
  return -1;
else
  { mid = approximate mid-point between first and last;
  if (key == a[mid]  // Stopping case
    return mid;
  else if key < a[mid]  // A case with recursion
    return the result of searching a[first] through a[mid-1];
  else if key > a[mid]  // A case with recursion.
  return result of seaching a[mid+1] through a[last];
}
```