Implement a menu driven C program that represents a dynamic array. The array should contain the following functions:

Operations	Parameters of the Function	Output of the Function
1. InsertFirst	The element to be inserted	Return success in case of successful insertion otherwise failure
2. InsertLast	The element to be inserted	Return success in case of successful insertion otherwise failure
3. InsertAt	The element to be inserted And the position	Return success in case of successful insertion otherwise failure
4. DeleteFirst	No parameters	Return success in case of successful deletion otherwise failure
5. DeleteLast	No parameters	Return success in case of successful deletion otherwise failure
6. DeleteAt	The position of the element to be deleted	Return success in case of successful deletion otherwise failure
7. Search	An element	The position of the element in the array
8. PrintAll	No parameter	void
9. Quit	No parameter	Return 0/exit
10. LastIndexOf (Bonus)	An element	The position of the last index of the given element in the array
11. DeleteElement (Bonus)	An element	Return success in case of successful deletion otherwise failure

1. shift all elements of the array one position right starting from the last element. This will leave the first position blank. insert **element** at the beginning of the list and increment length of the list. Check whether the list is full or not, if full expand the list. In case of successful insertion, return the index where the element has been inserted, Otherwise return -1.
2. insert **element** at the end of the list and increment length of the list. Check whether the list is full or not, if full expand the list. In case of successful insertion, return the index where the element has been inserted, Otherwise return -1
3. insert **element** at given **position**. Here *0 <= position < length*. Check for validity of range. Check whether the list is full or not, if full expand the list. In case of successful insertion return the index where the element has been inserted. Return -1 if the value of position is outside of the valid range.
4. delete the first element from the list. Shift the remaining elements one position left starting from the second element. Return -1 if the list is empty. Otherwise return any non-negative value (denoting success). Shrink the list if the current length of the list is less than one fourth of the maxLength.
5. delete the last element from the list. Return -1 if the list is empty otherwise return any non-negative value (denoting success). Shrink the list if the current length of the list is less than one fourth of the maxLength.
6. Delete the element at a given **position** from the list.
   Here *0 <= position < length.* Check for validity of range. Return -1 if the list is empty. Return -2 if the **position** value is outside of the valid range. Shrink the list if the current length of the list is less than one fourth of the maxLength.
7. Find **element** in the list and return the index of the first occurrence. Return -1 if element is not present in the list. Use *binary search* for bonus practice.
8. Print all elements of the array.
9. Terminate the program.
10. Find **element** in the list and return the index of the last occurrence. Return -1 if element is not present in the list. *Hint: search in the reverse order/ modify the binary search.*
11. delete the first occurrence of the **element** from the list. Use the search function to avoid redundant use of code. Return -1 if the list is empty. Shrink the list if the current length of the list is less than one fourth of the maxLength.

Important Note:

- Maintain a variable for the ***max_size*** of the array and ***length*** of the array. Use pointer and dynamic memory allocation for the ***array.***
- **Init():** Initialize the list with ***max_size.*** Set ***length*** to 0.

```
int maxsize = 8;
int *array;
int length;

void init(){
    array = (int*) malloc (sizeof(int) * max_size);
    Length = 0;
}
```

- **Expand():** Double the capacity of the list.
  Hint: Double the value of ***max_size,*** Take another integer pointer and allocate memory of size ***max_size*** to it and copy the content from the previous ***array*** to it. You may also use *realloc* and return the address of the newly allocated array. Release the memory occupied by the previous list and assign a new list to the previous one.

- **Shrink():** Half the capacity of the list.
  Hint: Half the value of ***max_size,*** Take another integer pointer and allocate memory of size ***max_size*** to it and copy the content from the previous ***array*** to it. You may also use *realloc* and return the address of the newly allocated array. Release the memory occupied by the previous list and assign a new list to the previous one.

For any confusion and query

Mail me at *muhaiminbinmunir@gmail.com*
Call me at +880 16 7805 4074.
Find me on the 8th floor, Tower 1 (Sunday and Thursday anytime).