Exercise 1: Array ADT and its applications

Create an ADT for the array data structure with the following functions. *arrADT* will have the integer array and size.

a. create(arrADT,size, array) – Create the array with the required number of elements

b. deleteAt(arrADT, pos) – Delete the specified element

c. insertAtEvery(arrADT,data) – Insert data before every element

d. search(arrADT, key) – return the position of the second occurrence of the element. If found return the position, otherwise return 1

e. printArray(arrADT) – prints the elements of the array

f. findPeek(arrADT, int *) – return a set of peek elements

Given an array **arr[]** of integers. Find a peak element i.e. an element that is **not smaller** than its neighbors.

Note: For corner elements, we need to consider only one neighbor.

Example:

Input: array[] = {10, 20, 15, 2, 23, 90, 67}

Output: 20, 90

Explanation: The element 20 has neighbors 10 and 15, both of them are less than 20, similarly 90 has neighbors 23 and 67.

CODE:

main.c:

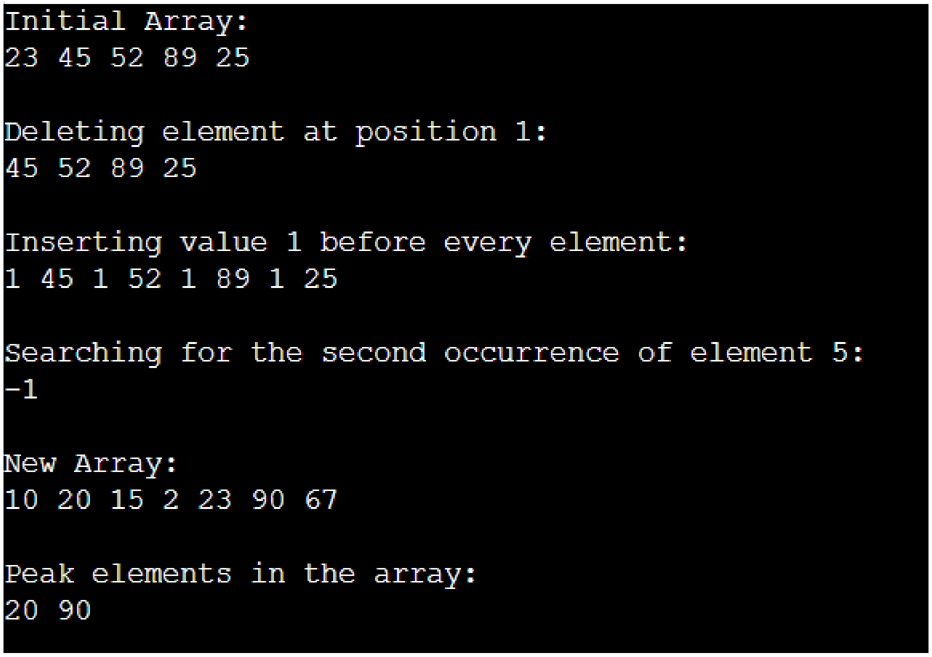
```
#include "arrADT.h"
#include<stdio.h>
#include<stdlib.h>
struct arr *a;
struct arr b;


int main() {
    struct arr a;
    int data1[] = {23, 45, 52, 89, 25};
    int n1 = 5;

    printf("Initial Array:\n");
    create(&a, n1, data1);
    print(&a);

    printf("\nDeleting element at position 1:\n");
    del(&a, 1, n1, data1);

    printf("\nInserting value 1 before every element:\n");
    insertAtEvery(&a, 1, a.size, a.arr);
    print(&a);

    printf("\nSearching for the second occurrence of element 5:\n");
    search(&a, 5);

    int data2[] = {10, 20, 15, 2, 23, 90, 67};
    int n2 = 7;

    printf("\nNew Array:\n");
    create(&a, n2, data2);
    print(&a);

    printf("\nPeak elements in the array:\n");
    findPeek(&a, a.size);

    return 0;
}
```

arrADT.h:

```
#ifndef ARR_ADT_H
#define ARR_ADT_H

#include <stdio.h>
#include <stdlib.h>

struct arr {
	int arr[100];
	int size;
};

/* Create array */
void create(struct arr *p, int s, int a[]) {
	p->size = s;
	for (int i = 0; i < p->size; i++) {
		p->arr[i] = a[i];
	}
}

/* Print array */
void print(struct arr *q) {
	for (int i = 0; i < q->size; i++) {
		printf("%d ", q->arr[i]);
	}
	printf("\n");
}

/* Delete element at given position */
void del(struct arr *b, int pos, int s, int a[]) {
	b->size = s;

	if (pos < 1 || pos > b->size) {
		printf("Invalid position\n");
		return;
	}

	for (int i = pos - 1; i < b->size - 1; i++) {
		b->arr[i] = a[i + 1];
	}

	b->size--;
	print(b);
```

```
}

/* Insert a given element before every element */
void insertAtEvery(struct arr *l, int data, int s, int a[]) {
    l->size = s;
    int new_size = 2 * l->size;
    int temp[200];

    for (int i = 0; i < l->size; i++) {
        temp[2 * i] = data;
        temp[2 * i + 1] = a[i];
    }

    l->size = new_size;
    for (int i = 0; i < new_size; i++) {
        l->arr[i] = temp[i];
    }
}

/* Search for second occurrence of an element */
void search(struct arr *p, int search_element) {
    int count = 0;

    for (int i = 0; i < p->size; i++) {
        if (p->arr[i] == search_element) {
            count++;
            if (count == 2) {
                printf("%d\n", i);
                return;
            }
        }
    }

    printf("-1\n");
}

/* Find peak elements */
void findPeek(struct arr *p, int s) {
    p->size = s;

    if (p->size == 0) {
        return;
    }
    if (p->size == 1) {
        printf("%d\n", p->arr[0]);
        return;
    }

    for (int i = 0; i < p->size; i++) {
        if (i == 0 && p->arr[i] > p->arr[i + 1]) {
            printf("%d ", p->arr[i]);
        }
        else if (i == p->size - 1 && p->arr[i] > p->arr[i - 1]) {
            printf("%d ", p->arr[i]);
        }
        else if (i > 0 && i < p->size - 1 &&
              p->arr[i] >= p->arr[i - 1] &&
              p->arr[i] >= p->arr[i + 1]) {
            printf("%d ", p->arr[i]);
        }
    }
    printf("\n");
}

#endif
```

OUTPUT:



```
Initial Array:
23 45 52 89 25

Deleting element at position 1:
45 52 89 25

Inserting value 1 before every element:
1 45 1 52 1 89 1 25

Searching for the second occurrence of element 5:
-1

New Array:
10 20 15 2 23 90 67

Peak elements in the array:
20 90
```

LEARNING OUTCOME:

This exercise allows for practical application and understanding of Abstract Data Types(ADTs)Through the implementation of essential array functions.It cultivates skills in algorithm design,error handling,and problem solving capabilities.