```c
/**
 * @author  Aviruk Basak, CSE214047, Sem 3, Year 2
 * @topic   1 dimensional array data structure implemented in C
 * @date    3-8-2022
 * @cc      gcc -Wall -D ARR_TYPE=int -D TYPE_FORMAT='"%d"' -o dsa-int-array dsa-array.c
 */

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

# if !defined(ARR_TYPE) || !defined(TYPE_FORMAT)
#    undef  ARR_TYPE
#    undef  TYPE_FORMAT
#    define ARR_TYPE     int
#    define TYPE_FORMAT  "%d"
# endif

# define ERR_NULLPTR      (200)
# define ERR_OUTOFBOUNDS  (201)
# define ERR_ARRAYFULL    (203)
# define ERR_MENUDRIVELIM (204)

# define MENUDRIVE_LIMIT  ((size_t) 10000)

typedef ARR_TYPE  ArrayType;
typedef ARR_TYPE* Array;

void arr_nullPtrCheck(char *fname, Array arr);
Array new_array(size_t size);
Array arr_resize(Array arr, size_t size, size_t new_sz);
size_t arr_length(Array arr, size_t size);
void arr_print(Array arr, size_t size);
void arr_traverse(Array arr, size_t size, void (*callback)(size_t index, ArrayType val));
size_t arr_search(Array arr, size_t size, ArrayType val);
size_t *arr_searchAll(Array arr, size_t size, ArrayType val, size_t *matches);
Array arr_insert(Array arr, size_t size, size_t index, ArrayType val);
Array arr_delIndex(Array arr, size_t size, size_t index);
Array arr_delValue(Array arr, size_t size, ArrayType val);
Array arr_concat(Array arr1, size_t sz1, Array arr2, size_t sz2, size_t *new_sz);
Array arr_merge(Array arr1, size_t sz1, Array arr2, size_t sz2, size_t *new_sz);
Array arr_intersect(Array arr1, size_t sz1, Array arr2, size_t sz2, size_t *new_sz);
void arr_free(Array* arr_ptr);

void arr_nullPtrCheck(char *fname, Array arr)
{
    if (!arr) {
        printf("array: %s: null pointer\n", fname);
        exit(ERR_NULLPTR);
    }
}

Array new_array(size_t size)
{
    // calloc initialises all indices with 0, last index 0 indicates atleast 1 empty index
    Array arr = calloc(sizeof(ArrayType), size);
    arr_nullPtrCheck("new", arr);
    return arr;
}

Array arr_resize(Array arr, size_t size, size_t new_sz)
{
    size_t i;
    arr_nullPtrCheck("resize", arr);
    // calloc initialises all indices with 0, last index 0 indicates atleast 1 empty index
    Array arr2 = calloc(sizeof(ArrayType), new_sz);
    arr_nullPtrCheck("resize", arr2);
    for (i = 0; i < new_sz; i++) {
        if (i >= size)
            break;
        arr2[i] = arr[i];
    }
    arr_free(&arr);
    return arr2;
}

size_t arr_length(Array arr, size_t size)
{
    size_t i;
    for (i = size -1; i >= 0; i--) {
        if (arr[i] != 0) {
            return i +1;
        }
    }
    return 0;
}

void arr_print(Array arr, size_t size)
{
    size_t i, len;
    arr_nullPtrCheck("print", arr);
    len = arr_length(arr, size);
    printf("{ ");
    for (i = 0; i < len; i++) {
        printf(TYPE_FORMAT "%s", arr[i], i == len -1 ? " " : ", ");
    }
    printf("}\n");
}

void arr_traverse(Array arr, size_t size, void (*callback)(size_t index, ArrayType val))
{
    size_t i;
    arr_nullPtrCheck("traverse", arr);
    for (i = 0; i < size; i++) {
        callback(i, arr[i]);
    }
}

size_t arr_search(Array arr, size_t size, ArrayType val)
{
    size_t i, len;
    len = arr_length(arr, size);
    for (i = 0; i < len; i++) {
        if (arr[i] == val)
            return i;
    }
    return size;
}

size_t *arr_searchAll(Array arr, size_t size, ArrayType val, size_t *matches)
{
    size_t *indices, i, len;
    arr_nullPtrCheck("search all", arr);
    *matches = 0;
    indices = NULL;
    len = arr_length(arr, size);
    for (i = 0; i < len; i++) {
        if (arr[i] == val) {
            indices = realloc(indices, ++(*matches) * sizeof(size_t));
            if (!indices) {
                printf("array: search all: null pointer\n");
                exit(ERR_NULLPTR);
            }
            indices[*matches -1] = i;
        }
    }
    return indices;
}

Array arr_insert(Array arr, size_t size, size_t index, ArrayType val)
{
    size_t i;
    arr_nullPtrCheck("insert", arr);
    if (index >= size) {
        printf("array: insert: index out of bounds\n");
        exit(ERR_OUTOFBOUNDS);
    } else if (arr[size -1] != 0) {
        printf("array: insert: array is full\n");
        exit(ERR_ARRAYFULL);
    }
    for (i = size -1; i > index; i--) {
        arr[i] = arr[i -1];
    }
    arr[index] = val;
    return arr;
}

Array arr_delIndex(Array arr, size_t size, size_t index)
{
    size_t i;
    arr_nullPtrCheck("delete index", arr);
    if (index >= size) {
        printf("array: delete index: index out of bounds\n");
        exit(ERR_OUTOFBOUNDS);
    }
    for (i = index; i < size -1; i++) {
        arr[i] = arr[i +1];
    }
    arr[i] = 0;
    return arr;
}

Array arr_delValue(Array arr, size_t size, ArrayType val)
{
    size_t *indices = NULL, matches = 0, i;
    arr_nullPtrCheck("delete value", arr);
    indices = arr_searchAll(arr, size, val, &matches);
    if (!indices) {
        printf("array: delete value: null pointer\n");
        exit(ERR_NULLPTR);
    }
    for (i = 0; i < matches; i++) {
        arr_delIndex(arr, size, indices[i]);
    }
    free(indices);
    return arr;
}

Array arr_concat(Array arr1, size_t sz1, Array arr2, size_t sz2, size_t *new_sz)
{
    size_t i, j, k, len1, len2;
    arr_nullPtrCheck("concat", arr1);
    arr_nullPtrCheck("concat", arr2);
    len1 = arr_length(arr1, sz1);
    len2 = arr_length(arr2, sz2);
    *new_sz = len1 + len2;
    Array arr3 = new_array(*new_sz);
    for (i = k = 0; i < len1 && k < *new_sz; i++, k++) {
        arr3[k] = arr1[i];
    }
    for (j = 0; j < len2 && k < *new_sz; j++, k++) {
        arr3[k] = arr2[j];
    }
    return arr3;
}

Array arr_merge(Array arr1, size_t sz1, Array arr2, size_t sz2, size_t *new_sz)
{
    size_t i, j, k, len1, len2;
    arr_nullPtrCheck("merge", arr1);
    arr_nullPtrCheck("merge", arr2);
    len1 = arr_length(arr1, sz1);
    len2 = arr_length(arr2, sz2);
    *new_sz = len1 + len2;
    Array arr3 = new_array(*new_sz);
    i = j = k = 0;
    while (i < len1 && j < len2) {
        if (arr1[i] < arr2[j]) {
            arr3[k] = arr1[i];
            i++;
            k++;
        } else if (arr2[j] < arr1[i]) {
            arr3[k] = arr2[j];
            j++;
            k++;
        } else {
            arr3[k] = arr1[i] = arr2[j];
            i++;
            j++;
            k++;
        }
    }
    while (i < len1) {
        arr3[k] = arr1[i];
        i++;
        k++;
    }
    while (j < len2) {
        arr3[k] = arr2[j];
        j++;
        k++;
    }
    return arr3;
}

Array arr_intersect(Array arr1, size_t sz1, Array arr2, size_t sz2, size_t *new_sz)
{
    size_t i, j, k, len1, len2;
    arr_nullPtrCheck("intersect", arr1);
    arr_nullPtrCheck("intersect", arr2);
    len1 = arr_length(arr1, sz1);
    len2 = arr_length(arr2, sz2);
    *new_sz = len1 + len2;
    Array arr3 = new_array(*new_sz);
    i = j = k = 0;
    while (i < len1 && j < len2) {
        if (arr1[i] < arr2[j]) {
            i++;
        } else if (arr2[j] < arr1[i]) {
            j++;
        } else {
            arr3[k] = arr1[i] = arr2[j];
            i++;
            j++;
            k++;
        }
    }
    return arr3;
}

void arr_free(Array* arr_ptr)
{
    if (arr_ptr && *arr_ptr) {
        free(*arr_ptr);
        *arr_ptr = NULL;
    }
}

int main()
{
    int choice;
    size_t i, size, len, menudrive_iterations = 0;
    printf("enter array max size: ");
    scanf("%zu", &size);
    printf("enter no of elements to store: ");
    scanf("%zu", &len);
    Array arr = new_array(size);
    printf("enter %zu elements = ", len);
    for (i = 0; i < len; i++) {
        scanf(TYPE_FORMAT, &arr[i]);
    }
    do {
        printf(
            "\nchoices:\n"
            "    0: exit\n"
            "    1: print array\n"
            "    2: search for matching value\n"
            "    3: search for every matching value\n"
            "    4: insert a value\n"
            "    5: delete value at index\n"
            "    6: delete every match of a value\n"
            "    7: concatenate two arrays\n"
            "    8: merge two sorted arrays\n"
            "    9: intersection of two sorted arrays\n"
            "enter your choice: "
        );
        scanf("%d", &choice);
        printf("\n");
        switch (choice) {
            // exit
            case 0: {
                break;
            }
            // print
            case 1: {
                printf("arr  = ");
                arr_print(arr, size);
                printf("len  = %zu\n", arr_length(arr, size));
                printf("size = %zu\n", size);
                break;
            }
            // search
            case 2: {
                ArrayType val;
                size_t index;
                printf("enter value to be searched: ");
                scanf(TYPE_FORMAT, &val);
                index = arr_search(arr, size, val);
                if (index == size) {
                    printf("value not found\n");
                } else {
                    printf("value '" TYPE_FORMAT "' found at index = %zu\n", val, index);
                }
                break;
            }
            // searchAll
            case 3: {
                ArrayType val;
                size_t i, *indices, matches;
                printf("enter value to be searched: ");
                scanf(TYPE_FORMAT, &val);
                indices = arr_searchAll(arr, size, val, &matches);
                if (matches == 0) {
                    printf("value not found\n");
                } else {
                    printf("value '" TYPE_FORMAT "' found at indices: ", val);
                    for (i = 0; i < matches; i++) {
                        printf("%zu%s", indices[i], i == matches -1 ? "" : ", ");
                    }
                    printf("\n");
                }
                free(indices);
                break;
            }
            // insert
            case 4: {
                ArrayType val;
                size_t index;
                printf("enter index of insertion: ");
                scanf("%zu", &index);
                printf("enter value to be inserted: ");
                scanf(TYPE_FORMAT, &val);
                arr = arr_insert(arr, size, index, val);
                printf("arr = ");
                arr_print(arr, size);
                break;
            }
            // delIndex
            case 5: {
                size_t index;
                printf("enter index of deletion: ");
                scanf("%zu", &index);
                arr = arr_delIndex(arr, size, index);
                printf("arr = ");
                arr_print(arr, size);
                break;
            }
            // delValue
            case 6: {
                ArrayType val;
                printf("enter value to delete: ");
                scanf(TYPE_FORMAT, &val);
                arr = arr_delValue(arr, size, val);
                printf("arr = ");
                arr_print(arr, size);
                break;
            }
            // concat
            case 7: {
                size_t size2, len2, new_sz;
                printf("enter second array max size: ");
                scanf("%zu", &size2);
                printf("enter no of elements to store: ");
                scanf("%zu", &len2);
                Array arr2 = new_array(size2);
                printf("enter %zu elements = ", len2);
                for (i = 0; i < len2; i++) {
                    scanf(TYPE_FORMAT, &arr2[i]);
                }
                Array arr3 = arr_concat(arr, size, arr2, size2, &new_sz);
                printf("new arr = ");
                arr_print(arr3, new_sz);
                arr_free(&arr3);
                break;
            }
            // merge
            case 8: {
                size_t size2, len2, new_sz;
                printf("enter second array max size: ");
                scanf("%zu", &size2);
                printf("enter no of elements to store: ");
                scanf("%zu", &len2);
                Array arr2 = new_array(size2);
                printf("enter sorted %zu elements = ", len2);
                for (i = 0; i < len2; i++) {
                    scanf(TYPE_FORMAT, &arr2[i]);
                }
                Array arr3 = arr_merge(arr, size, arr2, size2, &new_sz);
                printf("new arr = ");
                arr_print(arr3, new_sz);
                arr_free(&arr3);
                break;
            }
            // intersect
            case 9: {
                size_t size2, len2, new_sz;
                printf("enter second array max size: ");
                scanf("%zu", &size2);
                printf("enter no of elements to store: ");
                scanf("%zu", &len2);
                Array arr2 = new_array(size2);
                printf("enter sorted %zu elements = ", len2);
                for (i = 0; i < len2; i++) {
                    scanf(TYPE_FORMAT, &arr2[i]);
                }
                Array arr3 = arr_intersect(arr, size, arr2, size2, &new_sz);
                printf("new arr = ");
                arr_print(arr3, new_sz);
                arr_free(&arr3);
                break;
            }
            default: {
                printf("choice invalid\n");
            }
        }
        menudrive_iterations++;
    } while (choice && menudrive_iterations < MENUDRIVE_LIMIT);
    // MENUDRIVE_LIMIT to avoid an accidental infinite loop due to scanf I/O error
    if (menudrive_iterations >= MENUDRIVE_LIMIT) {
        printf("exceeded menu drive limit of '%zu' iterations\n", MENUDRIVE_LIMIT);
        exit(ERR_MENUDRIVELIM);
    }
    arr_free(&arr);
    return 0;
}
```