

Name : Yashwant Chandrakant Bhosale

MIS : 612303039

DIV : SY COMP DIV 1

array.h : Header file containing structure declaration and function prototypes for array data structure.

code:

```
typedef struct array {
    int *arr;
    int size;
    int len;
} array;

void init(array *a, int size);
void append(array *a, int element);
void insert_at_index(array *a, int index , int element);
void remove_at_index(array *a, int index);
void display(array a);
void max_min(array a);
void swap(int arr[], int i, int j);
void reverse(array *a);
array *merge(array a1, array a2);
void populate(array *a, int size);
```

array.c : File containing function definitions for functions associated with array data structure.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include "array.h"

// function to initialize an array
void init(array *a, int size) {
    a -> size = size;
    a -> len = 0;
```

```

    a -> arr = (int *) malloc(sizeof(int) * size);
    if(!(a -> arr))
        printf("Error: Memory allocation failed!\n");
    return;
}

```

// Function to append an element in the array

```

void append(array *a, int element) {
    if(!a) {
        return;
    }
    if(a->len >= a->size) {
        return;
    }
    (a->arr)[a->len] = element;
    (a->len)++;
    return;
}

```

// Function to insert an element at specific index in the array

```

void insert_at_index(array *a, int index , int element) {
    if(index >= a->size-1)
        return;
    int i;
    for(i = a->size-2; i > index; i--)
        a->arr[i+1] = a->arr[i];
    a -> arr[i] = element;
    return;
}

```

// Function to remove an element from specified index of the array

```

void remove_at_index(array *a, int index) {
    if(index >= a->size-1 || index < 0)
        return;
    for(int i = index; i < a->size-1; i++)

```

```

        a->arr[index] = a->arr[index+1];
    return;
}

// Function to display array
void display(array a) {
    printf("[\t");
    for(int i = 0; i < a.size; i++){
        if(a.arr[i])
            printf("%d,\t", a.arr[i]);
        else
            printf("X,\t");
    }
    printf("]\n");
    return;
}

// Function to display maximum and minimum element in the
array
void max_min(array a) {
    int max = a.arr[0], min = a.arr[0];
    for(int i = 0; i < a.size; i++) {
        if(a.arr[i] > max)
            max = a.arr[i];
        if(a.arr[i] < min)
            min = a.arr[i];
    }
    printf("\nminimum: %d, maximum: %d\n", min, max);
    return;
}

// helper function: swaps two elements in the array
void swap(int arr[], int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    return;
}

```

```

// Function to reverse an array
void reverse(array *a) {
    for(int i = 0; i < a->size / 2; i++){
        swap(a->arr, i, a->size-i-1);
    }
    return;
}

// Function to merge two arrays
array *merge(array a1, array a2) {
    array* a3;
    a3 = (array *) malloc(sizeof(array));
    if(!a3){
        return NULL;
    }
    a3->size = a1.size + a2.size;
    a3->arr = (int *)malloc(sizeof(int) * a3->size);
    int i;
    for(i = 0; i < a1.size; i++) {
        a3->arr[i] = a1.arr[i];
    }
    for(i = a1.size; i < a3->size; i++){
        a3->arr[i] = a2.arr[i-a1.size];
    }
    a3->len = 0;
    return a3;
}

// Function to populate the array
void populate(array *a, int size){
    for (int i = 0; i < size; i++)
        a->arr[i] = rand() % 100;
    a->size = size;
    return;
}

```

main.c: contains main execution of the program

```

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

```

```

list l;
int list_ptr = -1;
void view_menu(){
    printf("\n-----\n");
    printf("1. init\n2. append\n3. insert_at_index\n4.
remove_at_index\n5. display\n6. max /min\n7. reverse\n8.
merge\n9. Populate\n10. Exit\n");
    printf("\n -----\\
n");
}

void read_option(int option) {
    switch(option) {
        case 1: {
            int size;
            array *a = (array *) malloc(sizeof(array));
            printf("Enter size: ");
            scanf("%d", &size);
            init(a, size);
            append_node(&l, a);
            list_ptr++;
            return;
            break;
        }case 2: {
            if(list_ptr == -1){
                printf("Please initialize an array first!");
                return;
            }
            int index, data;
            node *p;
            printf("Enter array no.: ");
            scanf("%d", &index);
            while((index-1) > list_ptr ){
                printf("Enter array no.: ");
                scanf("%d", &index);
            }
            p = l;
            while(p->next && index-1) {
                p = p->next;
                index--;
            }
        }
    }
}

```

```

    }
    printf("Enter data: ");
    scanf("%d", &data);
    append(p->A, data);
    return;
    break;
}case 3: {
    if(list_ptr == -1){
        printf("Please initialize an array first!");
        return;
    }
    node* p;
    int data, index, array;
    printf("Array no.: ");
    scanf("%d", &array);
    printf("Enter data: ");
    scanf("%d", &data);
    printf("Enter index: ");
    scanf("%d", &index);
    p = 1;
    int i = 0;
    while(p->next && i < array) {
        p = p->next;
        i++;
    }
    insert_at_index(p->A, index, data);
    return;
}case 4: {
    if(list_ptr == -1){
        printf("Please initialize an array first!");
        return;
    }
    node* p;
    int index, array;
    printf("Array no.: ");
    scanf("%d", &array);
    printf("Enter index: ");
    scanf("%d", &index);
    p = 1;
    int i = 0;
    while(p->next && i < array) {

```

```

        p = p->next;
        i++;
    }
    remove_at_index(p->A, index);
    return;
}case 5: {
    if(list_ptr == -1){
        printf("Please initialize an array first!");
        return;
    }
    node *p;
    int array, i =0;
    printf("Array no: ");
    scanf("%d", &array);
    p = 1;
    while(p->next && i < array) {
        p = p->next;
        i++;
    }
    display(*(p->A));
    return;
}case 6: {
    if(list_ptr == -1){
        printf("Please initialize an array first!");
        return;
    }
    node *p;
    int array, i =0;
    printf("Array no: ");
    scanf("%d", &array);
    p = 1;
    while(p->next && i < array) {
        p = p->next;
        i++;
    }
    max_min(*(p->A));
    return;
}
case 7: {
    int index;
    node *p;

```

```

printf("Enter array no.: ");
scanf("%d", &index);
while((index-1) > list_ptr ){
    printf("Enter array no.: ");
    scanf("%d", &index);
}
p = 1;
while(p->next && index-1) {
    p = p->next;
    index--;
}
reverse(p->A);
return;
}case 8:{
    if(list_ptr < 1) {
        printf("We need at least two arrays to
perform merge..\n");
        return;
    }
    int a1, a2;
    node *p, *q;
    printf("1st array no.: ");
    scanf("%d", &a1);
    printf("2nd array no.: ");
    scanf("%d", &a2);

    p = 1;
    q = 1;
    while(p-> next && a1-1){
        p = p->next;
        a1--;
    }
    while(q->next && a2-1) {
        q = q->next;
        a2--;
    }
    array *a3 = merge(*(p->A), *(q->A));
    append_node(&l, a3);
    list_ptr++;
    return;
}

```



```

    }
    case 9: {
        if(list_ptr == -1){
            printf("Please initialize an array first!");
            return;
        }
        int index;
        node *p;
        printf("Enter array no.: ");
        scanf("%d", &index);
        while((index-1) > list_ptr ){
            printf("Enter array no.: ");
            scanf("%d", &index);
        }
        p = 1;
        while(p->next && index-1) {
            p = p->next;
            index--;
        }
        populate(p->A, p->A->size);
        return;
    }
    case 10: {
        return;
    }
    default: {
        printf("invalid option!\n");
        break;
    }
}
}

```

```

int main(){
    int option;
    init_list(&l);
    while(1){
        view_menu();
        printf("enter option: ");
        scanf("%d", &option);
        read_option(option);
        if(option == 10){

```

```

        break;
    }
    printf("\nArray List: \n");
    display_list(l);
}
return 0;
}

```

Output:

init:

```

yashwantbhosale@fedora:~/Programming/DSA/college-assignments/array$ gcc -Wall main.c array.c list.c
yashwantbhosale@fedora:~/Programming/DSA/college-assignments/array$ ./a.out

-----
1. init
2. append
3. insert_at_index
4. remove_at_index
5. display
6. max /min
7. reverse
8. merge
9. Populate
10. Exit
-----
enter option: 1
Enter size: 5

Array List:
{
array: 1      [      ]
}

```

populate:

```

array: 1      [      ]
}

-----
1. init
2. append
3. insert_at_index
4. remove_at_index
5. display
6. max /min
7. reverse
8. merge
9. Populate
10. Exit
-----
enter option: 9
Enter array no.: 1

Array List:
{
array: 1      [      83,      86,      77,      15,      93,      ]
}

```

append:

Here, First we initialize a new array using init option which is 1.

Then we use option 2 to append a new data element to the array which is 12.

{Note: after each command I am printing list of arrays so that user can understand what is going on}

```
yashwantbhosale@fedora:~/Programming/DSA/college-assignments/array$ ./a.out
2. append
3. insert_at_index
4. remove_at_index
5. display
6. max /min
7. reverse
8. merge
9. Populate
10. Exit

-----
enter option: 1
Enter size: 5

Array List:
{
array: 1      [      ]
}

-----
1. init
2. append
3. insert_at_index
4. remove_at_index
5. display
6. max /min
7. reverse
8. merge
9. Populate
10. Exit

-----
enter option: 2
Enter array no.: 1
Enter data: 12

Array List:
{
array: 1      [    12,    ]
}
```

insert_at_index: Here, first we have a populated array. Then after choosing option 3 (for inserting at index) we enter data (11) and index (2), then data is inserted at specified index.

```
yashwantbhosale@fedora:~/Programming/DSA/college-assignments/array$ ./a.out
{
array: 1      [    83,    86,    77,    15,    ]
}

-----
1. init
2. append
3. insert_at_index
4. remove_at_index
5. display
6. max /min
7. reverse
8. merge
9. Populate
10. Exit

-----
enter option: 3
Array no.: 1
Enter data: 11
Enter index: 2

Array List:
{
array: 1      [    83,    86,    11,    15,    ]
}
```

remove_at_index: Here, initially we have some array with some data. Now, when we remove element at specified index (2), the data at that index is overwritten with the data at next index and so on.

So here initially the data at index 2 was 11, when we performed remove at index operation data at next location was overwritten on the 2nd index and now **last element of the array virtually contains invalid data and is ready to use as a new location.**

```
yashwantbhosale@fedora:~/Programming/DSA/college-assignments/array -- ./a.out
Array List:
{
array: 1      [      83,      86,      11,      15,      ]
}

-----
1. init
2. append
3. insert_at_index
4. remove_at_index
5. display
6. max /min
7. reverse
8. merge
9. Populate
10. Exit
-----
enter option: 4
Array no.: 1
Enter Index: 2

Array List:
{
array: 1      [      83,      86,      15,      15,      ]
}

-----
1. init
```

Display: display function is being called after every operation.

```
yashwantbhosale@fedora:~/Programming/DSA/college-assignments/array -- ./a.out
Array List:
{
array: 1      [      83,      86,      15,      15,      ]
}

-----
1. init
2. append
3. insert_at_index
4. remove_at_index
5. display
6. max /min
7. reverse
8. merge
9. Populate
10. Exit
-----
enter option: 5
Array no: 1
[      83,      86,      15,      15,      ]

Array List:
{
array: 1      [      83,      86,      15,      15,      ]
}

-----
1. init
```

max/min: This function is supposed to print the maximum and minimum element in the array.

```
yashwantbhosale@fedora:~/Programming/DSA/college-assignments/array$ ./a.out
{
array: 1      [      93,      35,      86,      92,      ]
}
-----
1. init
2. append
3. insert_at_index
4. remove_at_index
5. display
6. max /min
7. reverse
8. merge
9. Populate
10. Exit
-----
enter option: 6
Array no: 1

minimum: 35, maximum: 93

Array List:
{
array: 1      [      93,      35,      86,      92,      ]
}
}
-----
1. init
```

reverse: This function is supposed to reverse the array.

```
yashwantbhosale@fedora:~/Programming/DSA/college-assignments/array$ ./a.out
Array List:
{
array: 1      [      93,      35,      86,      92,      ]
}
}
-----
1. init
2. append
3. insert_at_index
4. remove_at_index
5. display
6. max /min
7. reverse
8. merge
9. Populate
10. Exit
-----
enter option: 7
Enter array no.: 1

Array List:
{
array: 1      [      92,      86,      35,      93,      ]
}
}
-----
1. init
```

merge: This function is supposed to merge two arrays into a single array.

array 1 and **array 2** merged into single **array 3**.

```
yashwantbhosale@fedora:~/Programming/DSA/college-assignments/array$ ./a.out
Array List:
{
array: 1      [      92,      86,      35,      93,      ]
array: 2      [      49,      21,      62,      27,      90,      ]
}
}
-----
1. init
2. append
3. insert_at_index
4. remove_at_index
5. display
6. max /min
7. reverse
8. merge
9. Populate
10. Exit
-----
enter option: 8
1st array no.: 1
2nd array no.: 2

Array List:
{
array: 1      [      92,      86,      35,      93,      ]
array: 2      [      49,      21,      62,      27,      90,      ]
array: 3      [      92,      86,      35,      93,      49,      21,      62,      27,      90,      ]
}
}
-----
```

Use of linked list: Here linked list is being used to keep a track of all initialized arrays. Whenever we initialize a new array a new node is appended in the list and after every operation all initialized arrays are displayed.

List.h: Header file for list contains struct declaration and function prototypes

```
#include "array.h"
typedef struct node {
    array *A;
    struct node *next;
} node;

typedef node * list;
void append_node(list *l, array *array);
void display_list (list l) ;
void init_list(list *l);
```

list.c: Contains function definitions for functions associated with list

```
#include <stdio.h>
#include <stdlib.h>
#include "list.h"
// Function to initialize new list
void init_list(list *l) {
    *l = NULL;
    return;
}

/* Appends new node to the list, this function is called when
new array is initialized */
void append_node(list *l, array *Array) {
    if(!(*l)) {
        printf("This is first node!\n");
        node *new_node = (node *) malloc(sizeof(node));
        *l = new_node;
        (*l) -> Array = Array;
        (*l) -> next = NULL;
        return;
    }else {
        node *p = *l;
        while(p -> next) {
            p = p -> next;
        }
    }
```

```

        }
        node* new_node= (node *) malloc(sizeof(node));
        new_node -> Array = Array;
        new_node -> next = NULL;
        p->next = new_node;
    }
    return;
}

/* function to display all arrays in the list */
void display_list (list l) {
    if(!l) {
        printf("empty list\n");
        return;
    }
    node *p = l;
    int i = 0;
    while(p) {
        printf("Array %d: ", i);
        if(p -> Array){
            display(*(p->Array));
        }
        else
            printf("array is empty\n");
        i++;
        p = p -> next;
    }
    printf("size in display_list: %d\n", l->Array->size);
    printf("\n");
    return;
}

```