```
/*
                        Experiment No: 01

            Author:Harsheet Sujit Chordiya
            Roll No.: 37
            Sem & Sec: CSE 3B
            Source file: expt01.c
            Date Compiled: 25-Nov2021

================================================================================
     Aim: To study an Array ADT and to implement various operations
on an Array ADT. (include binary linearsearch)       Problem
Statement: 1.Create an array and implement the operations --
initialize(), create(), traverse(), isFull(),isEmpty(),
                  length(),insertElement(),deleteElement(),sort(),
merge(),display(),linearsearch(),concatenate() and copy().
                  2.Write a C program to demonstrate an array ADT
using defined operations using a menu-driven approach.

================================================================================

                                        THEORY/ ALGORITHM
================================================================================

> Characterization of an ADT.

    ADT stands for Abstract Data Types.

    Abstraction : It is property of an object to concentrate on
    essential aspects neglecting the rest.

    An ADT is a mathematical model of a data structure that specifies
    the type of data stored, the operations supported on them, and
the
    types of parameters of the operations.
    An ADT specifies what each operation does, but not how it does
it.
    Typically, an ADT can be implemented using one of many different
    data structures.
    A useful first step in deciding what data structure to use in a
    program is to specify an ADT for the program.

> Algorithms for Array ADT operations :

    1. create()~

        Procedure CREATE(A, LEN)
            This procedure initialises array with element values.
            LEN represents array length and is passed by reference.
            1. Parameter initialization
                    LEN := A.Length

            2. Iterate to receive input and insert into array
                    Repeat step 2 through step 3 until
                    (LEN = MX_SIZE OR VALUE<> MIN_VALUE)

            3. Check if Array is Full, Otherwise Accept and insert
element
               values.
                   If LEN = MX_SIZE
```

```
                        Print (' Array Bounds Reached.. ')
                        Break
                    Else
                        VALUE := MIN_VAL ( Discard MIN_VALUE )
                            Continue

                        Call INSERT_ELEMENT(A, (LEN), VALUE)

             4. Return Array and Length
                    Return.

    2. isFull()~

            Function IS_FULL (A)
            1. If the array is Full, return TRUE
                    If A.Length = MX_SIZE
                    Return 1

            2. Otherwise, return FALSE
                    Return 0

    3. isEmpty()~

            Function IS_EMPTY (A)
            1. If the array is Empty, return TRUE
                    If A.Length = 0
                    Return 1

            2. Otherwise, return FALSE
                    Return 0
    4. length()~

            Function LENGTH(A)
            1. Local parameter
                    LEN := 0

            2. Iterate Until MX_SIZE
                    While LEN < MX_SIZE
                    [If Element Value is MIN_VAL, Exit Loop]
                        If A(LEN) = MIN_VAL
                            Break
                        LEN := LEN+1

                    End-While

            3. Return Length
                    Return LEN.

    5. insert_Element()~

            Function INSERT_ELEMENT (A, INDEX, VALUE)
            1. Set Array Length
                    LEN := A.Length

            2. Is Array Full ?
                    If LEN = MX_SIZE
                        Print (' Array Full, Insert Failed ')
                        Return LEN

            3. Iterate through array to insert the element at INDEX
```

```
    position.
                    For I := LEN down to INDEX
                        A(I+1) := A(I)

            4. Place the value at desired position.
                    A(INDEX) := VALUE

            5. Return Updated Length
                    Return LEN+1

        6. delete_Element()~

            Function DELETE_ELEMENT (A, INDEX, VALUE)
            1. Set Array Length
                    LEN := A.Length

            2. Is Array Empty ?
                    If LEN = 0
                        Print (' Array Empty, Delete Failed ')
                        Return LEN

            3. Iterate through array to remove the element at INDEX
    position.
                    For I := INDEX to LEN-1
                        A(I) := A(I-1)

            4. Was the array Full , before removal of element ? Set
    Delimiter.
                    If LEN = MX_SIZE
                        A(LEN-1) := MIN_VAL

            5. Return Updated Length
                    Return LEN-1

        7. sort()~
            (Using Bubble Sort)

            1. Begin BubbleSort of Array.

            2. Iterate through all the elements of Array to sort it
    in asending
               order.
                    If A[I] > A[I+1]
                        swap(A[I], A[I+1])

            3. Return Updated Array.
                    Return A


            Function
        8. copy()~

            Function COPY(A, B)
            1. Iterate through Array A
                    For I := 0 to A.Length-1
                        B(I) := A(I)

            2. Set Array Boundary
                    If A.Length < MX_SIZE
```

```
                            B(I) := MIN_VAL

170             3. Return Length of Copied Array
                        Return A.Length

        9. traverse()~

175             Function TRAVERSE(A)
                1. Set Length
                        LEN := A.Length

                2. Is Array Empty ? Generate Error.
180                     If LEN = 0
                            Print(' Array Empty, Failed.. ')

                3. Otherwise, Execute Operation
                        Else
185                         For I := 0 to LEN-1
                                A(I) := A(I) + 100

                        End-If

190 ================================================================================

                            VIVA QUESTION

    ================================================================================
195 1.Define an algorithm.
        Ans: An algorithm is a series of instructions telling a computer
    how to
        transform a set of facts about the world into useful information.
    It is a set of
        instructions for solving a problem or accomplishing a task.
    2.What do you understand by a Data structure?
200     Ans: A data structure is a specialized format for organizing,
    processing,
        retrieving and storing data. In computer science and computer
    programming, a
        data structure may be selected or designed to store data for the
    purpose of
        using it with various algorithms.It is a collection of data
    values, the
        relationships among them, and the functions or operations that
    can be applied to
205     the data,[4] i.e., it is an algebraic structure about data.
    3. How does an algorithm differ from a program.
        Ans: An algorithm is more like an idea, a way to solve a problem,
    while a
        program is more linked to the execution of one or more tasks by a
    computer.A
        program can implement one or more algorithms, or it may be so
    simple that we
210     don´t have to use an algorithm.The task of a developer usually
    starts by
        designing algorithms to solve the problems and then implement
    them and include
        them in a program.
    4. Give an algorithm to remove duplicate element values from an array.
        Ans: Step 1: Input the number of elements of the array.
215      Step 2: For i = 1 to n-1
```

```
        Step 3: Set min = arr[i]
        Step 4: Set position = i
        step 5: For j = i+1 to n-1 repeat:
        if (min > arr[j])
220     Set min = arr[j]
        Set position = j
        [end of if]
        end of loop(j)]
        Step 6: swap a[i] with arr[position]
225     [end of loop(i)]
        Step 7: Repeat from i = 1 to n
        if (arr[i] != arr[i+1])
        temp[j++] = arr[i]
        temp[j++] = arr[n-1]
230     Repeat from i = 1 to j
        arr[i] = temp[i]
        Step 8: Print array.

    ================================================================================
235
                                        Code
    ================================================================================
    */
    //  header file inclusion
240 #include <stdio.h>
    #include <stdlib.h>
    #include <time.h>


245 // pre-processor directives
    #define SIZE 10
    #define MIN_VAL -999

    // Function declaration
250 void intializeArrayADT(int*);
    int lengthArrayADT(int*);
    void displayArrayADT(int*,int);
    int insertArrayADT(int *,int,int ,int);
    void isFullArrayADT(int* );
255 void isEmptyArrayADT(int*);
    void create(int*, int*);
    int linearsearch(int*,int,int*);
    void copy(int*,int*,int);
    void merge(int*,int*,int*,int);
260 void traverse(int*,int);
    void sort(int*,int);
    void delete(int*,int,int);
    int binarysearch(int*,int,int,int);
    void concatenate(int*,int*,int,int);
265


    //Main Function

270 int main(){
        int list[SIZE],list1[SIZE],list2[SIZE],list3[SIZE];
        int len=0,len1=0,len2=0,i,choice,temp=1,index,value,temp1;

        //intialize list
275     intializeArrayADT(list);
```

```
        intializeArrayADT(list1);
        intializeArrayADT(list2);
        intializeArrayADT(list3);

    do{
            printf("\nAn ArrayADT \n");
            printf("\t1.Create\t 2.Length \t 3.isFull\n\t 4.
isEmpty\t5.Display \t6.Insert \n\t 7.linearsearch\t 8.delete \t
9.Copy\n");
            printf("\t 10.Traverse \t 11.sort \t 12.Merge \n\t
13.Concatenate \t 14.binary search \t 0. Exit\n");
            printf("Enter Operation code:\n");
            scanf("%d",&choice);
            switch (choice)
            {

                case 0: printf("You opted to exit");
                    break;
                case 1: create(list,&len);
                    break;
                case 2:len=lengthArrayADT(list);
                    printf("The length of array is: %d \n",len);
                    break;
                case 3:isFullArrayADT(list);
                    break;
                case 4:isEmptyArrayADT(list);
                    break;
                case 5:
                    len=lengthArrayADT(list);
                    displayArrayADT(list,len);
                    break;
                case 6:
                    if (len==SIZE){
                        printf("The array is Full \n");
                        break;
                    }
                    else{
                        while(temp!=0){
                            printf("Enter the Element followed by its
index(strating from 0 and enter index as -999 to exit insert menu)-
\n");
                            scanf("%d%d",&value,&index);
                            if(index==-999)
                                temp=0;
                            else
                                insertArrayADT(list,len,index,value);

                        }
                        printf("All the elemnts were stored\n");

                    }
                    break;
                case 7:
                    printf("Enter the value to linearsearch:\n");
                    scanf("%d",&value);
                    len=lengthArrayADT(list);
                    temp1=linearsearch(list,value,&len);
                    if(temp1==1)
                        printf("The entered value is availble in the
array\n");
```

```
330                     else
                            printf("The entered value is not availble in the
    array\n");
                        break;
                case 8:
                    printf("Enter the index to delete\n");
335                 scanf("%d",&index);
                    len=lengthArrayADT(list);
                    delete(list,index,len);
                    printf("Deletion is successful\n");
                    break;
340             case 9: len=lengthArrayADT(list);
                    copy(list,list1,len);
                    break;

                case 10:len=lengthArrayADT(list);
345                 traverse(list,len);
                    printf("The list is traversed and incremented by
    100\n");
                    break;
                case 11:len=lengthArrayADT(list);
                    // Insertion Sort
350                 sort(list,len);
                    printf("The list is successfully sorted\n");
                    break;
                case 12:
                        len=lengthArrayADT(list);
355                     merge(list,list2,list3,len);
                        break;
                case 13: create(list2,&len1);
                    len=lengthArrayADT(list);
                    len1=lengthArrayADT(list2);
360                 concatenate(list,list2,len,len1);
                    break;
                case 14: printf("Enter the element you eant to search
    with binary:\n");
                    scanf("%d",&value);
                    len=lengthArrayADT(list);
365                 temp1=binarysearch(list,0,value,len);
                    if(temp1==1)
                        printf("The value is available\n");
                    else
                        printf("Value is not available\n");
370                 break;

                default:
                    break;
            }
375

        }
        while(choice!=0);
    }
380
    //Function defination
    void create(int *arr,int *len){

        if(*len==SIZE){
385         printf("Array bounds reached\n:");
        }
```

```
        else{
            for(int i=0; i<=SIZE; i++){
                printf("Enter the element %d:\n",i);
390             scanf("%d",&arr[i]);
                if(arr[i]==MIN_VAL)
                    break;


395         }
        }
    }
    void intializeArrayADT(int *arr)
    {
400     arr[0]=MIN_VAL;
        return;
    }
    void displayArrayADT(int* arr,int len){
        int kount=0;
405     if(arr[0]==MIN_VAL)
            printf("Array has no elements\n");
        else{
            printf("Array contents are\n");
            for(kount=0;kount<len;kount++)
410         {
                printf("%d\t \n",arr[kount]);

            }
        }
415 }
    int lengthArrayADT(int *arr){
        int len=0,kount=0;
        if(arr[0]==MIN_VAL)
            return len;
420     while(arr[kount]!=MIN_VAL){
            len+=1;
            kount++;
        }
        return(len);
425 }
    int insertArrayADT(int *arr, int len,int index ,int value){
        int kount=0;
        if(len==SIZE){
            printf("\nArray limit Excedded:\n");
430         return len;
        }
        for(kount=len;kount>=index;kount--)
            arr[kount+1]=arr[kount];
        // Insert value
435     arr[index]=value;
        return (len+1);
    }
    void isFullArrayADT(int *arr){
        int num= lengthArrayADT(arr);
440     if(num==SIZE){
            printf("The array is full\n");
        }
        else{
            printf("The array is not full\n");
445     }
    }
```

```c
void isEmptyArrayADT(int *arr){
    if(arr[0]==MIN_VAL){
        printf("The array is Empty\n");
    }
    else{
        printf("The array is not Empty\n");
    }
}
int linearsearch(int *arr, int value, int *len){
    for(int i=0;i<*len;i++){
        if(arr[i]==value){
            return 1;
        }

    }

}
void delete(int *arr,int index,int len){
    for(int i=index;i<len-1;i++){
        arr[i]=arr[i+1];
    }
    arr[len-1]=MIN_VAL;

}
void traverse(int *arr, int len){
    for(int i=0;i<len;i++){
        arr[i]=100+arr[i];
    }


}
void sort(int *arr,int len){
    int i,j,count,temp=0;
    for(i=0;i<len;i++){
        count=arr[i];
        j=i-1;
        while((count<arr[j])&&(j>=0)){
            arr[j+1]=arr[j];
            j=j-1;
        }
        arr[j+1]=count;
    }
}
void copy(int*arr,int*arr1,int len){
    int i;
    for(i=0;i<=len;i++)
    {
        arr1[i]=arr[i];
    }

    printf("The copied array is\n");
    displayArrayADT(arr1,len);
}
void concatenate(int*arr,int*arr1,int len,int len1){
    int i=0,temp=0,t=len+len1;
    for(i=len;i<t;i++){
        arr[i]=arr1[temp];
        temp++;
    }
    printf("After concatenation list is:\n");
```

```
        displayArrayADT(arr,t);

    }
510 void merge(int *arr,int *arr1,int *arr2,int len)
    {
        int i=0,j=0,k=0,x=0,y,z;
        intializeArrayADT(arr1);
        intializeArrayADT(arr2);
515     printf("\nArray elements that are to be merged :\n");
        create(arr1,&x);
        x=lengthArrayADT(arr1);
        sort(arr,len);
        sort(arr1,x);
520     y=len+x;
        for(i=0,j=0;arr[i]!=MIN_VAL && arr1[j]!=MIN_VAL;)
        {
            if(i>=len||j>=x)
               break;
525         if(arr[i]<=arr1[j])
            {
                arr2[k]=arr[i];
                i++;k++;
            }
530         else
            {
                arr2[k]=arr1[j];
                j++;k++;
            }
535
        }
        if(i>=len)
        {
            for(;j<x;j++,k++)
540              arr2[k]=arr1[j];
        }
        if(j>=x)
        {
            for(;i<len;i++,k++)
545              arr2[k]=arr[i];
        }
        if(k<(2*SIZE))
        {
           arr2[k]=MIN_VAL;
550     }
        printf("Merged Array :");
        displayArrayADT(arr2,k);
    }
    int binarysearch(int *arr,int l, int value, int len){
555     int mid;
        if (value>=l){
            mid=l+(value-l)/2;
            if (arr[mid]==len)
                return 1;
560         if(arr[mid]> len)
                return binarysearch(arr,l,mid-1,len);
            return binarysearch(arr,mid+1,value,len);
        }
        else
565         return 0;
```

```
}


/*
================================================================================
                                        Execution Trail
================================================================================

harsheet196@harsheet196:~/Desktop/csp252$  gcc expt01.c -o expt.out
harsheet196@harsheet196:~/Desktop/csp252$ ./expt.out
An ArrayADT
    1.Create     2.Length    3.isFull
     4. isEmpty 5.Display   6.Insert
     7.linearsearch  8.delete    9.Copy
     10.Traverse     11.sort     12.Merge
     13.Concatenate      14.binary search    0. Exit
Enter Operation code:
1
Enter the element 0:
1
Enter the element 1:
2
Enter the element 2:
3
Enter the element 3:
-999

An ArrayADT
    1.Create     2.Length    3.isFull
     4. isEmpty 5.Display   6.Insert
     7.linearsearch  8.delete    9.Copy
     10.Traverse     11.sort     12.Merge
     13.Concatenate      14.binary search    0. Exit
Enter Operation code:
2
The length of array is: 3

An ArrayADT
    1.Create     2.Length    3.isFull
     4. isEmpty 5.Display   6.Insert
     7.linearsearch  8.delete    9.Copy
     10.Traverse     11.sort     12.Merge
     13.Concatenate      14.binary search    0. Exit
Enter Operation code:
3
The array is not full

An ArrayADT
    1.Create     2.Length    3.isFull
     4. isEmpty 5.Display   6.Insert
     7.linearsearch  8.delete    9.Copy
     10.Traverse     11.sort     12.Merge
     13.Concatenate      14.binary search    0. Exit
Enter Operation code:
4
```

```
    The array is not Empty

    An ArrayADT
630      1.Create     2.Length    3.isFull
          4. isEmpty 5.Display   6.Insert
          7.linearsearch  8.delete    9.Copy
          10.Traverse     11.sort     12.Merge
          13.Concatenate      14.binary search    0. Exit
635 Enter Operation code:
    5
    Array contents are
    1
    2
640 3

    An ArrayADT
        1.Create     2.Length    3.isFull
          4. isEmpty 5.Display   6.Insert
645       7.linearsearch  8.delete    9.Copy
          10.Traverse     11.sort     12.Merge
          13.Concatenate      14.binary search    0. Exit
    Enter Operation code:
    6
650 Enter the Element followed by its index(strating from 0 and enter
    index as -999 to exit insert menu)
    4
    3
    Enter the Element followed by its index(strating from 0 and enter
    index as -999 to exit insert menu)
    5
655 4
    Enter the Element followed by its index(strating from 0 and enter
    index as -999 to exit insert menu)
    324
    -999
    All the elemnts were stored
660
    An ArrayADT
        1.Create     2.Length    3.isFull
          4. isEmpty 5.Display   6.Insert
          7.linearsearch  8.delete    9.Copy
665       10.Traverse     11.sort     12.Merge
          13.Concatenate      14.binary search    0. Exit
    Enter Operation code:
    5
    Array contents are
670 1
    2
    3
    4
    5
675

    An ArrayADT
        1.Create     2.Length    3.isFull
          4. isEmpty 5.Display   6.Insert
680       7.linearsearch  8.delete    9.Copy
          10.Traverse     11.sort     12.Merge
          13.Concatenate      14.binary search    0. Exit
    Enter Operation code:
```

```
7
Enter the value to linearsearch:
3
The entered value is availble in the array

An ArrayADT
    1.Create     2.Length    3.isFull
     4. isEmpty 5.Display   6.Insert
     7.linearsearch  8.delete    9.Copy
     10.Traverse     11.sort     12.Merge
     13.Concatenate      14.binary search    0. Exit
Enter Operation code:
7
Enter the value to linearsearch:
10
The entered value is not availble in the array

An ArrayADT
    1.Create     2.Length    3.isFull
     4. isEmpty 5.Display   6.Insert
     7.linearsearch  8.delete    9.Copy
     10.Traverse     11.sort     12.Merge
     13.Concatenate      14.binary search    0. Exit
Enter Operation code:
8
Enter the index to delete
2
Deletion is successful

An ArrayADT
    1.Create     2.Length    3.isFull
     4. isEmpty 5.Display   6.Insert
     7.linearsearch  8.delete    9.Copy
     10.Traverse     11.sort     12.Merge
     13.Concatenate      14.binary search    0. Exit
Enter Operation code:
5
Array contents are
1
2
4
5


An ArrayADT
    1.Create     2.Length    3.isFull
     4. isEmpty 5.Display   6.Insert
     7.linearsearch  8.delete    9.Copy
     10.Traverse     11.sort     12.Merge
     13.Concatenate      14.binary search    0. Exit
Enter Operation code:
9
The copied array is
Array contents are
1
2
4
5
```

```
    An ArrayADT
745     1.Create     2.Length    3.isFull
         4. isEmpty 5.Display   6.Insert
         7.linearsearch  8.delete    9.Copy
         10.Traverse     11.sort     12.Merge
         13.Concatenate      14.binary search    0. Exit
750 Enter Operation code:
    10
    The list is traversed and incremented by 100

    An ArrayADT
755     1.Create     2.Length    3.isFull
         4. isEmpty 5.Display   6.Insert
         7.linearsearch  8.delete    9.Copy
         10.Traverse     11.sort     12.Merge
         13.Concatenate      14.binary search    0. Exit
760 Enter Operation code:
    5
    Array contents are
    101
    102
765 104
    105


    An ArrayADT
770     1.Create     2.Length    3.isFull
         4. isEmpty 5.Display   6.Insert
         7.linearsearch  8.delete    9.Copy
         10.Traverse     11.sort     12.Merge
         13.Concatenate      14.binary search    0. Exit
775 Enter Operation code:
    11
    The list is successfully sorted

    An ArrayADT
780     1.Create     2.Length    3.isFull
         4. isEmpty 5.Display   6.Insert
         7.linearsearch  8.delete    9.Copy
         10.Traverse     11.sort     12.Merge
         13.Concatenate      14.binary search    0. Exit
785 Enter Operation code:
    12

    Array elements that are to be merged :
    Enter the element 0:
790 34
    Enter the element 1:
    0
    Enter the element 2:
    2
795 Enter the element 3:
    -999
    Merged Array :Array contents are
    0
    2
800 34
    101
    102
    104
```

```
    105

    An ArrayADT
        1.Create     2.Length    3.isFull
         4. isEmpty 5.Display   6.Insert
         7.linearsearch  8.delete    9.Copy
         10.Traverse     11.sort     12.Merge
         13.Concatenate      14.binary search    0. Exit
    Enter Operation code:
    13
    Enter the element 0:
    23
    Enter the element 1:
    56
    Enter the element 2:
    -999
    After concatenation list is:
    Array contents are
    101
    102
    104
    105
    23
    56

    An ArrayADT
        1.Create     2.Length    3.isFull
         4. isEmpty 5.Display   6.Insert
         7.linearsearch  8.delete    9.Copy
         10.Traverse     11.sort     12.Merge
         13.Concatenate      14.binary search    0. Exit
    Enter Operation code:
    14
    Enter the element you eant to search with binary:
    3
    Value is not available

    An ArrayADT
        1.Create     2.Length    3.isFull
         4. isEmpty 5.Display   6.Insert
         7.linearsearch  8.delete    9.Copy
         10.Traverse     11.sort     12.Merge
         13.Concatenate      14.binary search    0. Exit
    Enter Operation code:
    14
    Enter the element you eant to search with binary:
    23
    The value is available

    An ArrayADT
        1.Create     2.Length    3.isFull
         4. isEmpty 5.Display   6.Insert
         7.linearsearch  8.delete    9.Copy
         10.Traverse     11.sort     12.Merge
         13.Concatenate      14.binary search    0. Exit
    Enter Operation code:
    0
    *** stack smashing detected ***: terminated
    Aborted (core dumped)
```

```
    ================================================================================
865
                                CONCLUSION

    ================================================================================

870         After performng this experiment we were able to understand
    Array ADT
            better and learnt the way to implement it.


    ================================================================================
875 */



880
```