

Data Structure:

The term means a value or set of values. It specifies either the value of a variable or a constant. While a data item that does not have subordinate data items is categorized as an elementary item, the one that is composed of one or more subordinate data items is called a group item.

Data is defined as collection of raw facts which do not have any defined context. We can say that Data is collection of numbers and text values which do not have any direct benefit to the organization that collects the data since it is unorganized and unprocessed. For example data collection done by government regarding health, income, dietary habits, family etc. does not have any benefit if it is stored as it is. The reason is that it is very voluminous and does not give any information. The Data thus collected needs to be processed before it can be used for decision making or reporting tasks.

A field is defined as a unit of meaningful information about an entity like date of flight, name of passenger, address etc. A record is a collection of data items. Record is a collection of units of information about a particular entity. Passenger of an airplane, an employee of an organization or an article sold from a store. A file is a collection of related records. A collection of records involving a set of entities with certain aspects in common and organized for some particular purpose is called a file. For example collection of records of all passengers.

A data structure is defined as mathematical or logical model used to organize and manipulate data. The management of data is done through various operations like traversal, insertion, deletion, searching, sorting etc. The logical organization of data as a data structure is done to make the data organization easier, allow access to individual elements of a data structure, defining the association among data elements and provide various operations to process the data to derive information.

To create a computer program and solve a specific problem, the programmer needs some structure to hold the data values. So, data structures are an important aspect of problem solving through computer programming. It must be easy to understand and implement and must exhibit the relationship among data elements required to provide solution. For example to implement a transportation or network problem a graph like data structure is needed. To implement sequential execution of submitted tasks a queue like data structure is the best option.

Linear Data Structure:

Data elements are organized in a linear fashion in a linear data structure. Traversal can only be done sequentially. All the previous elements must be followed first to reach a particular element in linear data structures. It also means that only one data element can be directly reached from the previous element. Every data element in a linear data structure has a direct relation with its prior and following element. Examples of linear data structure are Arrays, Linked Lists, Stacks and Queues. Linear data structures can be represented in memory in two

different ways. One way is to have to a linear relationship between elements by means of sequential memory locations. The other way is to have a linear relationship between elements by means of links.

Non-Linear Data Structure:

In a nonlinear data structure data elements are arranged non-linearly and traversal cannot be done sequentially. Every data element may be linked to more than one data elements. The linkages of the data elements reflect a particular relationship. The relationship between the elements can be hierarchical or random. Examples of linear data structure are Trees and Graphs. If the elements of a data structure are not stored in a sequential order, then it is a non-linear data structure. The relationship of adjacency is not maintained between elements of a non-linear data structure.

Primitive Data Structure:

Primitive data structures are the fundamental data types which are supported by a programming language. Some basic data types are integer, real, character, and boolean. The terms 'data type', 'basic data type', and 'primitive data type' are often used interchangeably.

Non-Primitive Data Structure:

Non-primitive data structures are those data structures which are created using primitive data structures. Examples of such data structures include linked lists, stacks, trees, and graphs. Non-primitive data structures can further be classified into two categories: linear and non-linear data structures.

61. Linear Data Structure Elements:

Array:

An array is an indexed collection of fixed number of homogeneous data elements. The main advantage of arrays is we can represent huge number of values by using single variable. So that readability of the code will be improved. But the main disadvantage of array is fixed in size that is one's we create an array there is no chance of increasing or decreasing in the size based on our requirement.

Hence to use arrays concept compulsory we should know the size in advance, which may not possible always. An array is a homogeneous and linear data structure that is stored in contiguous memory locations. An element in an array can be accessed, inserted or removed by specifying its position or index or subscript along with the name of the array.

Linked List:

A linked list is a fundamental data structure in computer science, used to organize and manage collections of elements. Unlike arrays, which store elements in contiguous memory locations,

a linked list consists of nodes, each containing both data and a reference (or pointer) to the next node in the sequence. This flexible structure enables dynamic memory allocation, making it efficient for inserting and deleting elements anywhere in the list. However, traversal through a linked list can be slower compared to arrays, as it requires following the links sequentially.

Linked lists come in various forms, including singly linked lists with one-way connections, doubly linked lists with both next and previous references, and circular linked lists where the last node points back to the first. Each variation offers specific advantages depending on the use case, making linked lists a versatile and essential component of data structuring and manipulation in programming. These are the following types of Linked List.

Singly Linked List:

Singly Linked List is a type of linked list when each node contains data and a reference to the next node. It's a simple structure that allows traversal in only one direction, from the head (start) to the tail (end) of the list. Singly linked lists are efficient for insertions and deletions at the beginning and middle of the list, but accessing elements further down the list requires linear traversal.

A linear or singly linked list is a linear data structure consisting of a sequence of nodes. Each node consists of two parts – Data and Link to the next node. The address of first node is stored in a pointer Start.

Each node stores the address of its next node in its link part. The linked list is non-contiguous collection of nodes that allows insertion and deletion at any specific location.

Doubly Linked List:

A doubly linked list is a linear data structure consisting of a sequence of nodes. Each node consists of three parts – Data, Forward Pointer to the next node and Backward Pointer to the previous node. The address of first node is stored in a pointer called the Header and address of last node is stored in pointer called the Trailer. The doubly linked list allows traversal in forward and backward direction.

Doubly Linked List is a type of linked list where each node has references to both the next and the previous nodes. This bidirectional linkage allows for easier traversal in both directions, making insertions and deletions at any position more efficient. However, doubly linked lists require more memory due to the additional references in each node.

Circular Linked List:

A circular linked list forms a closed loop, where the last node points back to the first node, creating a circular structure. This type can be implemented as either singly or doubly linked. Circular linked lists are useful for applications like implementing circular buffers or managing processes in a round-robin scheduling algorithm.

Stack:

Stack is a linear data structure that follows Last in First out (LIFO) principle. Insertion (push) and Deletion(pop) are done at one end and other end is closed. Stacks are also used to implement traversal in DFS for a graph. Stack makes an essential data structure for OS functions and execution of arithmetic expressions. The maximum size of the stack must be defined a priori and cannot be changed. Trying to push a new element into a full stack causes an implementation-specific exception.

Queue:

A queue is a linear data structure that follows the First-In-First-Out (FIFO) principle. Elements are added to the back (enqueue) and removed from the front (dequeue). It resembles a real-world queue, like people waiting in line. Queues are useful for managing tasks in order, such as in breadth-first searches and print job scheduling.

Queue is a linear data structure that follows the first-in first-out scheme. Insertions are done at the rear of the queue and removals at the front of the queue. Two variables needed for these operations are the front and rear.

Deque(Double-ended Queue):

A variation of Queue is Dequeue in which deletions and insertions can be done at both ends. A deque is a versatile linear structure that supports insertion and deletion at both ends. It can function as a queue or a stack, providing flexibility for various applications. Deques are efficient for tasks like implementing algorithms that require simultaneous access to both ends, such as palindrome checking and implementing sliding window problems.

Priority Queue:

A priority queue is a data structure where elements have assigned priorities, and the highest-priority element is accessed first. It doesn't follow a strict order like queues. It's often implemented as a binary heap, enabling efficient insertion and retrieval of the highest-priority element. Priority queues are crucial in scenarios where tasks must be processed based on urgency or importance, such as in Dijkstra's algorithm for shortest path finding and Huffman coding in data compression.

priority queue is a linear data structure that behaves like a queue where each element has a priority associated with it on the basis of which deletions are done. The element with maximum (minimum) priority will be deleted from the queue. It can also be defined as a data structure that supports operations of min or max insert, delete or search.

62. Non-Linear Data Structure:

Heap:

A max(min) heap is a complete binary tree with the property that the value of each node is at least as large(small) as the values at its children. If the elements are distinct then the root node will be the largest (smallest).

The insertion will always be done as a leaf node first and then the heap will be “happified” to bring the newly inserted node at its correct position. Heapify means that the node will be compared with its parent and if it is larger(smaller) than its parent then there will be an interchange between parent node and the newly inserted child node. This process will end up in a heap after insertion.

Deletion will be done at the root node. The last node of the complete binary tree will replace the deleted root node and then the process of Heapify will be done starting from the new root node.

Tree:

A tree is a Non-Linear Data Structure that is an abstract model of a hierarchical structure consisting of nodes with a parent-child relation. Its applications are Organization charts, File systems, Programming environments. There are four things associated with any tree - Distinction between nodes, Value of nodes, orientation structure and the number of levels.

Root is starting point of the tree is a node called root characterized by a node without parent. External/Leaf Node is a node without any children. Internal node is all the nodes other than the root leaf nodes. It can be said to be a node with at least one child. Ancestors of a node parent, grandparent and any other nodes which lie on the path from root to that node.

Depth of a node is the number of ancestors for any give node. Height of a tree is maximum depth among all the leaf nodes. Descendant of a node is child grandchild and all the other nodes lying on path from a node to a leaf node. Subtree is a tree consisting of a node and its descendants.

Graph:

A graph $G(V,E)$ is a defined with two sets. V , a set of vertices, and E the set of edges between two pair of vertices from the set V . When the edges of the graph have to be defined with the direction, it is called a directed graph or digraph, and the edges are called directed edges or arcs. In a digraph edge (a,b) is not the same as edge (b, a) . In a directed edge (a,b) a is called the source/starting point and b is called the sink/destination/ end point. In an undirected graph the direction of edge is not specified.

When the edges of the graph have to be defined with the direction, it is called a directed graph or digraph, and the edges are called directed edges or arcs. In a digraph edge (a,b) is not the same as edge (b, a) . In a directed edge (a,b) a is called the source/starting point and b is

called the sink/destination/ end point. In an undirected graph the direction of edge is not specified. Two vertices that are connected to each other with an edge are called neighbors. They are also said to be adjacent to each other.

63. Data Structure Operations:

Traversal:

Traversal of a data structure can be defined as the process of visiting every element of a data structure at least once. This operation is most commonly used for printing, searching, displaying or reading the elements stored in the data structure. It is usually done by using a variable or pointer that indicates the current element of the data structure being processed.

This variable or pointer is updated after visiting an element so that the location or address of next element can be found. When the last element of the data structure is reached the traversal process ends. Traversal can be useful when all the elements of the data structure have to be manipulated in similar way.

In an array traversal begins from the first element. A variable stores the index of first element of the array and it is incremented after visiting each node. When the value of this variable is equal to the index of last node, the traversal of the array ends.

Insertion:

Once a data structure is created, it can be extended by adding new elements. The process of adding new elements in an existing data structure is called insertion. Where the element can be added depends upon the data structure that a user is dealing with. Insertion operation always ends up in increasing the size of the data structure.

A linked list and an array allow a user to insert a new element at any location. A stack and a queue allow a user to insert a new element only at a specific end. New nodes can be added to a graph or tree in a random fashion.

For all the data structure the insertion can be done till the data structure has enough space to store new elements either due to its defined size or memory availability. A condition when a user tries to insert a new element in a data structure that does not have the needed space for new element is called Overflow

Deletion:

Once a data structure is created, a user can remove any of the existing elements and free up the space occupied by it. The process of deleting an existing element from a data structure is called deletion. Which element can be deleted depends upon the data structure that a user is dealing with. Deletion operation always ends up in reducing the size of the data structure.

A linked list and an array allow a user to delete a existing element at any location i.e. start, mid or end. A stack and a queue allow a user to delete an element only at a specific end. Nodes can be deleted from a graph or tree in a random fashion.

For all the data structure the deletion can be done till the data structure has elements. A condition when a user tries to delete an element from a data structure that does not have any element is called Underflow

Search:

The process of locating an element in data structure and returning its index or address is called Search. This is the most commonly used operation in a data structure. Since a data structure stores data in an organized fashion for convenient processing, it is important that an element could be easily located in a data structure.

When performing search in a data structure a key value is needed, this is matched with the values stored in the data structure. When the value of the element matches the key value successful search is done and the search operation returns the location of that element. If the key value does not match any element in the data structure and reaches end, it is unsuccessful search and a null location is returned as a result of search operation.

Sorting:

The process of arranging the data elements in a data structure in a specific order (ascending or descending) by specific key values is called sorting. The students records stored in an array can be sorted by their registration numbers, names or the scores. In each sorting the criteria will be different to fulfill the processing needs of a user.

Sorting may result in physical relocation of elements or it can be just a rearrangement of key values as index without changing the physical address of complete data element so that a subsequent traversal operation displays the data in sorted manner.

Merge:

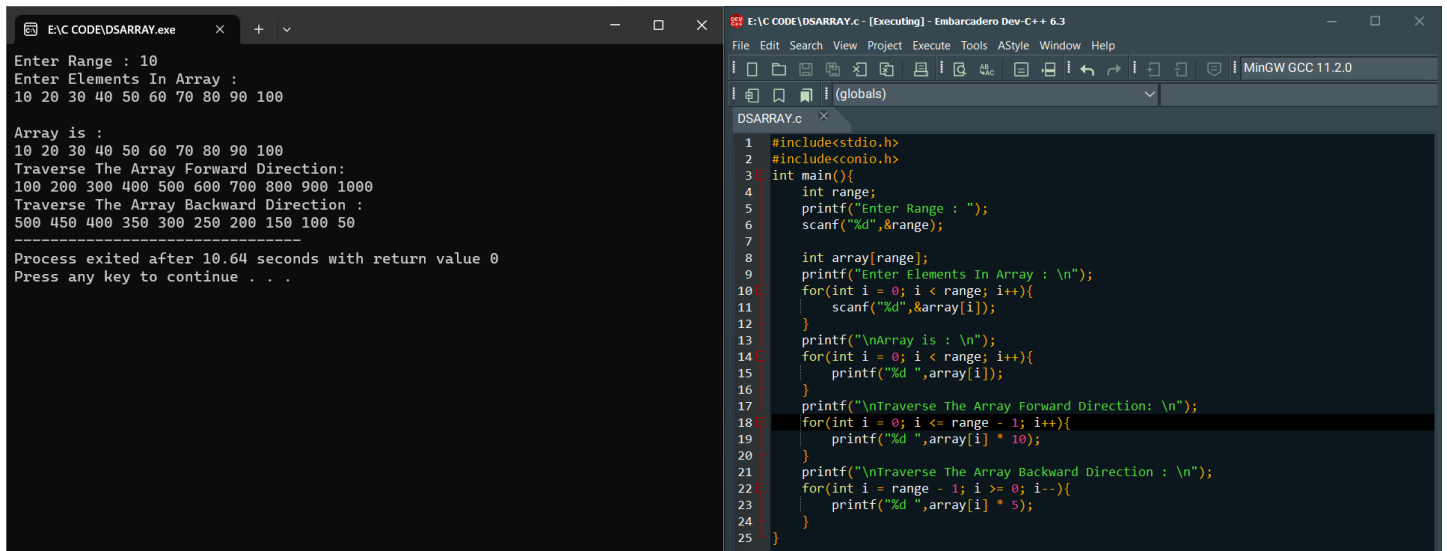
Merging can be defined as the process of combining elements of two data structures. In a simpler form merging can be treated as appending set of elements of one data structure after elements of another data structure of same element structure. For example two arrays containing student data of two different classes with same fields to form one list. The final data structure may or may not be sorted.

Another form of merging can be to merge two data structure of different constructions to create totally new data structure. For example one data structure with student registration number and name can be merged with another data structure containing course and result data of same set of students to form one list(or file)

Copy:

The process of copying is the operation that makes a new data structure from an existing data structure. In the process of copying the original data structure retains its structure and data elements. The new data structure has same data elements. Both these data structure can be used to perform all the operations discussed previously. Changes in one data structure will not affect the elements or count of elements of the other data structure.

Array Traversing:

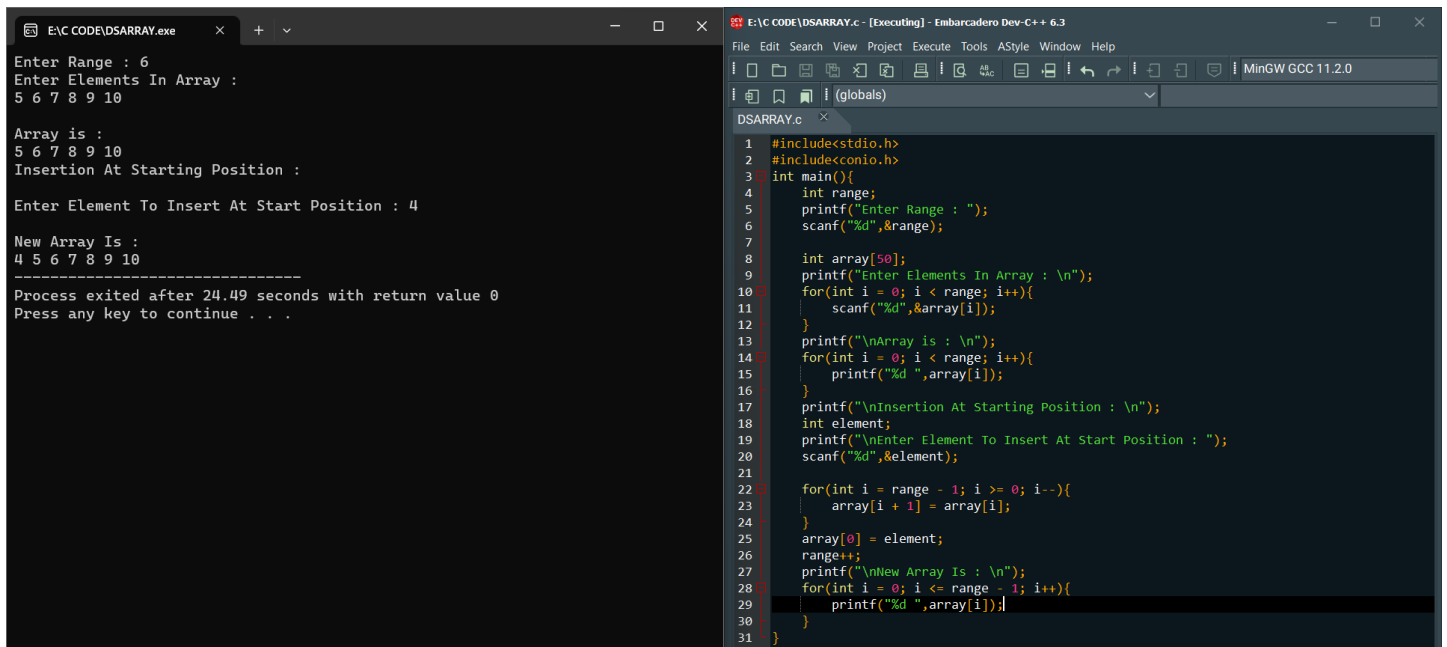


```
E:\C CODE\DSARRAY.exe
Enter Range : 10
Enter Elements In Array :
10 20 30 40 50 60 70 80 90 100

Array is :
10 20 30 40 50 60 70 80 90 100
Traverse The Array Forward Direction:
100 200 300 400 500 600 700 800 900 1000
Traverse The Array Backward Direction :
500 450 400 350 300 250 200 150 100 50
-----
Process exited after 10.64 seconds with return value 0
Press any key to continue . . .

E:\C CODE\DSARRAY.c - [Executing] - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
MinGW GCC 11.2.0
DSARRAY.c
1 #include<stdio.h>
2 #include<conio.h>
3 int main(){
4     int range;
5     printf("Enter Range : ");
6     scanf("%d",&range);
7
8     int array[range];
9     printf("Enter Elements In Array : \n");
10    for(int i = 0; i < range; i++){
11        scanf("%d",&array[i]);
12    }
13    printf("\nArray is : \n");
14    for(int i = 0; i < range; i++){
15        printf("%d ",array[i]);
16    }
17    printf("\nTraverse The Array Forward Direction: \n");
18    for(int i = 0; i <= range - 1; i++){
19        printf("%d ",array[i] * 10);
20    }
21    printf("\nTraverse The Array Backward Direction : \n");
22    for(int i = range - 1; i >= 0; i--){
23        printf("%d ",array[i] * 5);
24    }
25 }
```

Array Insertion:



```
E:\C CODE\DSARRAY.exe
Enter Range : 6
Enter Elements In Array :
5 6 7 8 9 10

Array is :
5 6 7 8 9 10
Insertion At Starting Position :

Enter Element To Insert At Start Position : 4

New Array Is :
4 5 6 7 8 9 10
-----
Process exited after 24.49 seconds with return value 0
Press any key to continue . . .

E:\C CODE\DSARRAY.c - [Executing] - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
MinGW GCC 11.2.0
DSARRAY.c
1 #include<stdio.h>
2 #include<conio.h>
3 int main(){
4     int range;
5     printf("Enter Range : ");
6     scanf("%d",&range);
7
8     int array[50];
9     printf("Enter Elements In Array : \n");
10    for(int i = 0; i < range; i++){
11        scanf("%d",&array[i]);
12    }
13    printf("\nArray is : \n");
14    for(int i = 0; i < range; i++){
15        printf("%d ",array[i]);
16    }
17    printf("\nInsertion At Starting Position : \n");
18    int element;
19    printf("\nEnter Element To Insert At Start Position : ");
20    scanf("%d",&element);
21
22    for(int i = range - 1; i >= 0; i--){
23        array[i + 1] = array[i];
24    }
25    array[0] = element;
26    range++;
27    printf("\nNew Array Is : \n");
28    for(int i = 0; i <= range - 1; i++){
29        printf("%d ",array[i]);
30    }
31 }
```



```
E:\C CODE\DSARRAY.exe x + v
Enter Range : 7
Enter Elements In Array :
10 20 30 40 50 60 70

Array is :
10 20 30 40 50 60 70
Insertion At Specified Position :
Enter Element To Insert At Specified Position : 33

Enter The Position Value : 4

New Array Is :
10 20 30 33 40 50 60 70
-----
Process exited after 30.39 seconds with return value 0
Press any key to continue . . .

E:\C CODE\DSARRAY.c - [Executing] - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
MinGW GCC 11.2.0
DSARRAY.c x
1 #include<stdio.h>
2 #include<conio.h>
3 int main(){
4     int range;
5     printf("Enter Range : ");
6     scanf("%d",&range);
7
8     int array[50];
9     printf("Enter Elements In Array : \n");
10    for(int i = 0; i < range; i++){
11        scanf("%d",&array[i]);
12    }
13    printf("\nArray is : \n");
14    for(int i = 0; i < range; i++){
15        printf("%d ",array[i]);
16    }
17    printf("\nInsertion At Specified Position : ");
18    int element;
19    printf("\nEnter Element To Insert At Specified Position : ");
20    scanf("%d",&element);
21    int pos;
22    printf("\nEnter The Position Value : ");
23    scanf("%d",&pos);
24
25    for(int i = range - 1; i >= pos - 1; i--){
26        array[i + 1] = array[i];
27    }
28    range++;
29    array[pos - 1] = element;
30    printf("\nNew Array Is : \n");
31    for(int i = 0; i <= range - 1; i++){
32        printf("%d ",array[i]);
33    }
34 }
```

```
E:\C CODE\DSARRAY.exe x + v
Enter Range : 7
Enter Elements In Array :
10 20 30 40 50 60 70

Array is :
10 20 30 40 50 60 70
Insertion At End Position :
Enter Element To Insert At Specified Position : 80

New Array Is :
10 20 30 40 50 60 70 80
-----
Process exited after 17.19 seconds with return value 0
Press any key to continue . . .

E:\C CODE\DSARRAY.c - [Executing] - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
MinGW GCC 11.2.0
DSARRAY.c x
1 #include<stdio.h>
2 #include<conio.h>
3 int main(){
4     int range;
5     printf("Enter Range : ");
6     scanf("%d",&range);
7
8     int array[50];
9     printf("Enter Elements In Array : \n");
10    for(int i = 0; i < range; i++){
11        scanf("%d",&array[i]);
12    }
13    printf("\nArray is : \n");
14    for(int i = 0; i < range; i++){
15        printf("%d ",array[i]);
16    }
17    printf("\nInsertion At End Position : ");
18    int element;
19    printf("\nEnter Element To Insert At Specified Position : ");
20    scanf("%d",&element);
21
22    array[range] = element;
23    range++;
24    printf("\nNew Array Is : \n");
25    for(int i = 0; i <= range - 1; i++){
26        printf("%d ",array[i]);
27    }
28 }
```

Array Deletion:

```
E:\C CODE\DSARRAY.exe x + v
Enter Range : 6
Enter Elements In Array :
10 20 30 40 50 60

Array is :
10 20 30 40 50 60
Deletion At Start Position :
New Array Is :
20 30 40 50 60
-----
Process exited after 12.37 seconds with return value 0
Press any key to continue . . .

E:\C CODE\DSARRAY.c - [Executing] - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
MinGW GCC 11.2.0
DSARRAY.c x
1 #include<stdio.h>
2 #include<conio.h>
3 int main(){
4     int range;
5     printf("Enter Range : ");
6     scanf("%d",&range);
7
8     int array[50];
9     printf("Enter Elements In Array : \n");
10    for(int i = 0; i < range; i++){
11        scanf("%d",&array[i]);
12    }
13    printf("\nArray is : \n");
14    for(int i = 0; i < range; i++){
15        printf("%d ",array[i]);
16    }
17    printf("\nDeletion At Start Position : ");
18    for(int i = 0; i <= range - 1; i++){
19        array[i] = array[i + 1];
20    }
21    range--;
22    printf("\nNew Array Is : \n");
23    for(int i = 0; i <= range - 1; i++){
24        printf("%d ",array[i]);
25    }
26 }
```

E:\C CODE\DSARRAY.exe

+ -

Enter Range : 8
Enter Elements In Array :
10 20 30 40 50 60 70 80

Array is :
10 20 30 40 50 60 70 80
Deletion At Specified Position :
Enter The Position Value : 4

New Array Is :
10 20 30 50 60 70 80

Process exited after 15.31 seconds with return value 0
Press any key to continue . . .

E:\C CODE\DSARRAY.c - [Executing] - Embarcadero Dev-C++ 6.3

File Edit Search View Project Execute Tools AStyle Window Help

MinGW GCC 11.2.0

(globals)

DSARRAY.c

```
1 #include<stdio.h>  
2 #include<conio.h>  
3 int main(){  
4     int range;  
5     printf("Enter Range : ");  
6     scanf("%d",&range);  
7  
8     int array[50];  
9     printf("Enter Elements In Array : \n");  
10    for(int i = 0; i < range; i++){  
11        scanf("%d",&array[i]);  
12    }  
13    printf("\nArray is : \n");  
14    for(int i = 0; i < range; i++){  
15        printf("%d ",array[i]);  
16    }  
17    printf("\ndeletion At Specified Position : ");  
18    int position;  
19    printf("\nEnter The Position Value : ");  
20    scanf("%d",&position);  
21    for(int i = position - 1; i <= range - 1; i++){  
22        array[i] = array[i + 1];  
23    }  
24    range--;  
25    printf("\nNew Array Is : \n");  
26    for(int i = 0; i <= range - 1; i++){  
27        printf("%d ",array[i]);  
28    }  
29 }
```

E:\C CODE\DSARRAY.exe

+

▼

Enter Range : 5
Enter Elements In Array :
10 20 30 40 50

Array is :
10 20 30 40 50
Deletion At End Position :
New Array Is :
10 20 30 40

Process exited after 13.25 seconds with return value 0
Press any key to continue . . .

E:\C CODE\DSARRAY.c - [Executing] - Embarcadero Dev-C++ + 6.3

File Edit Search View Project Execute Tools AStyle Window Help

MinGW GCC 11.2.0

(globals)

DSARRAY.c

```
1 #include<stdio.h>
2 #include<conio.h>
3 int main(){
4     int range;
5     printf("Enter Range : ");
6     scanf("%d",&range);
7
8     int array[50];
9     printf("Enter Elements In Array : \n");
10    for(int i = 0; i < range; i++){
11        scanf("%d",&array[i]);
12    }
13    printf("\nArray is : \n");
14    for(int i = 0; i < range; i++){
15        printf("%d ",array[i]);
16    }
17    printf("\nDeletion At End Position : ");
18    range--;
19    printf("\nNew Array Is : \n");
20    for(int i = 0; i <= range - 1; i++){
21        printf("%d ",array[i]);
22    }
23 }
```

Linear Search:

The screenshot displays the Embarcadero Dev-C++ IDE with the DSARRAY.c program open. The left pane shows the program's execution output, and the right pane shows the source code.

Output (Left Pane):

```

Enter Range : 10
Enter Elements In Array :
50 80 40 10 20 60 70 90 30 100
Array is :
arr[0] : 50
arr[1] : 80
arr[2] : 40
arr[3] : 10
arr[4] : 20
arr[5] : 60
arr[6] : 70
arr[7] : 90
arr[8] : 30
arr[9] : 100
Enter Element To Perform Linear Search : 20
20 is at 4 index 5 position!
-----
Process exited after 24.76 seconds with return value 0
Press any key to continue . . .

```

Source Code (Right Pane):

```

1  #include<stdio.h>
2  #include<conio.h>
3  int main()
4  {
5      int range;
6      printf("Enter Range : ");
7      scanf("%d",&range);
8
9      int array[50];
10     printf("Enter Elements In Array : \n");
11     for(int i = 0; i < range; i++){
12         scanf("%d",&array[i]);
13     }
14     printf("Array is : \n");
15     for(int i = 0; i < range; i++){
16         printf("arr[%d] : %d \n",i,array[i]);
17     }
18     int element;
19     printf("Enter Element To Perform Linear Search : ");
20     scanf("%d",&element);
21     for(int i = 0; i <= range - 1; i++){
22         if(array[i] == element){
23             printf("%d is at %d index %d position!",element,i,(i+1));
24             break;
25         }
26     }

```

Binary Search:

The image displays a Windows development environment with two panes. The left pane shows the output of a C++ program, and the right pane shows the source code in a file named DSARRAY.c.

Left Pane Output:

```
Enter Range : 10
Enter Elements In Array :
100 90 80 70 60 50 40 30 20 10
Array is :
arr[0] : 100
arr[1] : 90
arr[2] : 80
arr[3] : 70
arr[4] : 60
arr[5] : 50
arr[6] : 40
arr[7] : 30
arr[8] : 20
arr[9] : 10
Enter Element To Perform Binary Search Descending Order : 60
60 found at 4 index 5 position!
-----
Process exited after 15.88 seconds with return value 0
Press any key to continue . . .
```

Right Pane Source Code (DSARRAY.c):

```
1 #include<stdio.h>
2 #include<conio.h>
3 int main(){
4     int range;
5     printf("Enter Range : ");
6     scanf("%d",&range);
7
8     int array[50];
9     printf("Enter Elements In Array : \n");
10    for(int i = 0; i < range; i++){
11        scanf("%d",&array[i]);
12    }
13    printf("Array is : \n");
14    for(int i = 0; i < range; i++){
15        printf("arr[%d] : %d \n",i,array[i]);
16    }
17    int element;
18    printf("Enter Element To Perform Binary Search Descending Order : ");
19    scanf("%d",&element);
20    int last = 0, first = range - 1;
21
22    while(last <= first){
23        int mid = last + (first - last) / 2;
24        if(array[mid] < element)
25            first = mid - 1;
26        else if(array[mid] == element){
27            printf("%d found at %d index %d position!",element, mid, (mid+1));
28            break;
29        }
30        else
31            last = mid + 1;
32    }
33 }
```

The image displays a C++ development environment with two panes. The left pane shows the execution output of a program named DSARRAY.exe, and the right pane shows the source code of DSARRAY.c.

Left Pane (Output):

```
Enter Range : 10
Enter Elements In Array :
10 20 30 40 50 60 70 80 90 100
Array is :
arr[0] : 10
arr[1] : 20
arr[2] : 30
arr[3] : 40
arr[4] : 50
arr[5] : 60
arr[6] : 70
arr[7] : 80
arr[8] : 90
arr[9] : 100
Enter Element To Perform Binary Search Ascending Order : 20
20 found at 1 index 2 position!
-----
Process exited after 19.51 seconds with return value 0
Press any key to continue . . .
```

Right Pane (Source Code: DSARRAY.c):

```
1 #include<stdio.h>
2 #include<conio.h>
3 int main()
4 {
5     int range;
6     printf("Enter Range : ");
7     scanf("%d",&range);
8
9     int array[50];
10    printf("Enter Elements In Array : \n");
11    for(int i = 0; i < range; i++){
12        scanf("%d",&array[i]);
13    }
14    printf("Array is : \n");
15    for(int i = 0; i < range; i++){
16        printf("arr[%d] : %d \n",i,array[i]);
17    }
18    int element;
19    printf("Enter Element To Perform Binary Search Ascending Order : ");
20    scanf("%d",&element);
21    int last = 0, first = range - 1;
22
23    while(last <= first){
24        int mid = last + (first - last) / 2;
25        if(array[mid] > element)
26            first = mid - 1;
27        else if(array[mid] == element){
28            printf("%d found at %d index %d position!",element, mid, (mid+1));
29            break;
30        }
31        else
32            last = mid + 1;
33    }
```

Quick Sort:

DSARRAY.c

```
1  #include<stdio.h>
2  #include<conio.h>
3  int partition(int a[], int start, int end){
4      int pivot = a[end];
5      int i = (start - 1);
6      for(int j = start; j <= end - 1; j++){
7          if(a[j] < pivot){
8              i++;
9              int t = a[i];
10             a[i] = a[j];
11             a[j] = t;
12         }
13     }
14     int t = a[i + 1];
15     a[i + 1] = a[end];
16     a[end] = t;
17     return (i + 1);
18 }
19 void quick(int a[], int start, int end){
20     if(start < end){
21         int p = partition(a, start, end);
22         quick(a, start, p - 1);
23         quick(a, p + 1, end);
24     }
25 }
```

```
26 int partition1(int a[], int start, int end){
27     int pivot = a[end];
28     int i = (start - 1);
29     for(int j = start; j <= end - 1; j++){
30         if(a[j] > pivot){
31             i++;
32             int t = a[i];
33             a[i] = a[j];
34             a[j] = t;
35         }
36     }
37     int t = a[i + 1];
38     a[i + 1] = a[end];
39     a[end] = t;
40     return (i + 1);
41 }
42 void quick1(int ar[], int start, int end){
43     if(start < end){
44         int p1 = partition1(ar, start, end);
45         quick1(ar, start, p1 - 1);
46         quick1(ar, p1 + 1, end);
47     }
48 }
```

```

49 int main(){
50     int range;
51     printf("Enter Range Value : ");
52     scanf("%d",&range);
53
54     int array[range];
55     printf("\nEnter Array Elements: \n");
56
57     for(int i = 0; i <= range - 1; i++){
58         scanf("%d",&array[i]);
59     }
60     printf("\nUnsorted Array Is : \n");
61     for(int i = 0; i <= range - 1; i++){
62         printf("%d ",array[i]);
63     }
64     quick(array, 0, range - 1);
65     printf("\nAscending Sorted Array : \n");
66     for(int i = 0; i <= range - 1; i++){
67         printf("%d ",array[i]);
68     }
69     quick1(array, 0, range - 1);
70     printf("\nDescending Sorted Array : \n");
71     for(int i = 0; i <= range - 1; i++){
72         printf("%d ",array[i]);
73     }
74 }

```

E:\C CODE\DSARRAY.exe

Enter Range Value : 20

Enter Array Elements:
14 23 12 11 43 37 13 75 29 57 86 92 38 27 81 16 57 72 61 10

Unsorted Array Is :
14 23 12 11 43 37 13 75 29 57 86 92 38 27 81 16 57 72 61 10

Ascending Sorted Array :
10 11 12 13 14 16 23 27 29 37 38 43 57 57 61 72 75 81 86 92

Descending Sorted Array :
92 86 81 75 72 61 57 57 43 38 37 29 27 23 16 14 13 12 11 10

Process exited after 44.67 seconds with return value 0
Press any key to continue . . .

Bubble Sort:

DSARRAY.c

```
1  #include<stdio.h>
2  #include<conio.h>
3  #include <stdbool.h>
4  void ascendingbubble(int ar[], int len){
5      bool flag;
6      for(int i = 0; i < len - 1; i++){
7          flag = false;
8          for(int j = 0; j < (len - 1 - i); j++){
9              if(ar[j] > ar[j + 1]){
10                 int temp = ar[j];
11                 ar[j] = ar[j + 1];
12                 ar[j + 1] = temp;
13                 flag = true;
14             }
15         }
16         if(flag == false)
17             break;
18     }
19 }
```

```
20 void descendingbubble(int ar[], int len){
21     bool flag;
22     for(int i = 0; i < len - 1; i++){
23         flag = false;
24         for(int j = 0; j < (len - 1 - i); j++){
25             if(ar[j] < ar[j + 1]){
26                 int temp = ar[j];
27                 ar[j] = ar[j + 1];
28                 ar[j + 1] = temp;
29                 flag = true;
30             }
31         }
32         if(flag == false)
33             break;
34     }
35 }
```

E:\C CODE\DSARRAY.exe

Enter Range Value : 20

Enter Array Elements:

14 23 12 11 43 37 13 75 29 57 86 92 38 27 81 16 57 72 61 10

Unsorted Array Is :

14 23 12 11 43 37 13 75 29 57 86 92 38 27 81 16 57 72 61 10

Ascending Sorted Array :

10 11 12 13 14 16 23 27 29 37 38 43 57 57 61 72 75 81 86 92

Descending Sorted Array :

92 86 81 75 72 61 57 57 43 38 37 29 27 23 16 14 13 12 11 10

Process exited after 93.76 seconds with return value 0

Press any key to continue . . .

Insertion Sort:

```
DSARRAY.c x
1  #include<stdio.h>
2  #include<conio.h>
3  void ascendinginsertion(int ar[], int len){
4      for(int i = 0; i < len; i++){
5          int temp = ar[i];
6          int j = i - 1;
7          while(j >= 0 && ar[j] > temp){
8              ar[j + 1] = ar[j];
9              j--;
10         }
11         ar[j + 1] = temp;
12     }
13 }
14 void descendinginsertion(int ar[], int len){
15     for(int i = 1; i < len; i++){
16         int temp = ar[i];
17         int j = i - 1;
18         while(j >= 0 && ar[j] < temp){
19             ar[j + 1] = ar[j];
20             j--;
21         }
22         ar[j + 1] = temp;
23     }
24 }
```

```
E:\C CODE\DSARRAY.exe x + v
Enter Range Value : 20

Enter Array Elements:
14 23 12 11 43 37 13 75 29 57 86 92 38 27 81 16 57 72 61 10

Unsorted Array Is :
14 23 12 11 43 37 13 75 29 57 86 92 38 27 81 16 57 72 61 10
Ascending Sorted Array :
10 11 12 13 14 16 23 27 29 37 38 43 57 57 61 72 75 81 86 92
Descending Sorted Array :
92 86 81 75 72 61 57 57 43 38 37 29 27 23 16 14 13 12 11 10
-----
Process exited after 47.56 seconds with return value 0
Press any key to continue . . .
```

Selection Sort:

```
DSARRAY.c
1  #include<stdio.h>
2  #include<conio.h>
3  void ascendingselection(int ar[], int len){
4      for(int i = 0; i < len - 1; i++){
5          int min = i;
6          for(int j = i + 1; j < len; j++){
7              if(ar[j] < ar[min]){
8                  min = j;
9              }
10         }
11         if(min != i){
12             int temp = ar[i];
13             ar[i] = ar[min];
14             ar[min] = temp;
15         }
16     }
17 }
18 void descendingselection(int ar[], int len){
19     for(int i = 0; i < len - 1; i++){
20         int max = i;
21         for(int j = i + 1; j < len; j++){
22             if(ar[j] > ar[max]){
23                 max = j;
24             }
25         }
26         if(max != i){
27             int temp = ar[i];
28             ar[i] = ar[max];
29             ar[max] = temp;
30         }
31     }
32 }
```

```
E:\C CODE\DSARRAY.exe
Enter Range Value : 20

Enter Array Elements:
14 23 12 11 43 37 13 75 29 57 86 92 38 27 81 16 57 72 61 10

Unsorted Array Is :
14 23 12 11 43 37 13 75 29 57 86 92 38 27 81 16 57 72 61 10
Ascending Sorted Array :
10 11 12 13 14 16 23 27 29 37 38 43 57 57 61 72 75 81 86 92
Descending Sorted Array :
92 86 81 75 72 61 57 57 43 38 37 29 27 23 16 14 13 12 11 10
-----
Process exited after 48.48 seconds with return value 0
Press any key to continue . . .
```


Merge Sort:

```
1  #include<stdio.h>
2  #include<conio.h>
3  void merge(int a[], int beg, int mid, int end) {
4      int i, j, k;
5      int n1 = mid - beg + 1;
6      int n2 = end - mid;
7      int LeftArray[n1], RightArray[n2];
8      for (int i = 0; i < n1; i++)
9          LeftArray[i] = a[beg + i];
10     for (int j = 0; j < n2; j++)
11         RightArray[j] = a[mid + 1 + j];
12     i = 0;
13     j = 0;
14     k = beg;
15     while (i < n1 && j < n2) {
16         if(LeftArray[i] <= RightArray[j]) {
17             a[k] = LeftArray[i];
18             i++;
19         } else {
20             a[k] = RightArray[j];
21             j++;
22         }
23         k++;
24     }
25     while (i < n1) {
26         a[k] = LeftArray[i];
27         i++;
28         k++;
29     }
30     while (j < n2) {
31         a[k] = RightArray[j];
32         j++;
33         k++;
34     }
35 }
```

```
36 void mergeSort(int a[], int beg, int end) {
37     if (beg < end) {
38         int mid = (beg + end) / 2;
39         mergeSort(a, beg, mid);
40         mergeSort(a, mid + 1, end);
41         merge(a, beg, mid, end);
42     }
43 }
```

```
36 void mergeSort(int a[], int beg, int end) {
37     if (beg < end) {
38         int mid = (beg + end) / 2;
39         mergeSort(a, beg, mid);
40         mergeSort(a, mid + 1, end);
41         merge(a, beg, mid, end);
42     }
43 }
```

DSARRAY.c

```
1  #include<stdio.h>
2  #include<conio.h>
3  void merge(int a[], int beg, int mid, int end) {
4      int i, j, k;
5      int n1 = mid - beg + 1;
6      int n2 = end - mid;
7      int LeftArray[n1], RightArray[n2];
8      for (int i = 0; i < n1; i++)
9          LeftArray[i] = a[beg + i];
10     for (int j = 0; j < n2; j++)
11         RightArray[j] = a[mid + 1 + j];
12     i = 0;
13     j = 0;
14     k = beg;
15     while (i < n1 && j < n2) {
16         if(LeftArray[i] >= RightArray[j]) {
17             a[k] = LeftArray[i];
18             i++;
19         } else {
20             a[k] = RightArray[j];
21             j++;
22         }
23         k++;
24     }
25     while (i < n1) {
26         a[k] = LeftArray[i];
27         i++;
28         k++;
29     }
30     while (j < n2) {
31         a[k] = RightArray[j];
32         j++;
33         k++;
34     }
35 }
```

E:\C CODE\DSARRAY.exe

Enter Range Value : 20

Enter Array Elements:

14 23 12 11 43 37 13 75 29 57 86 92 38 27 81 16 57 72 61 10

Unsorted Array Is :

14 23 12 11 43 37 13 75 29 57 86 92 38 27 81 16 57 72 61 10

Descending Order Sorting :

92 86 81 75 72 61 57 57 43 38 37 29 27 23 16 14 13 12 11 10

Process exited after 207.3 seconds with return value 0

Press any key to continue . . .