THE UNIVERSITY OF LAHORE ISLAMABAD CAMPUS

Data Structure and Algorithms

Lab Report

Name:	Muhammad Arslan Ishaq
Registration #:	CSU-F16-112
Lab Report #:	01
Dated:	16-04-2018
Submitted To:	Mr. Usman Ahmed

The University of Lahore, Islamabad Campus
Department of Computer Science & Information Technology

Experiment # 1
Introduction to Arrays and its operation

Objective
The objectives of this lab session are to understand the basic and various operations on arrays in C++.

Software Tool
1. Dev C++

1 Theory

We have already studied array in our computer programming course. We would be using the knowledge we learned there to implement different operation on arrays.

Traversing Linear Arrays:-
Let A be the collection of data elements stored in the memory of the computer. Suppose we want to print the contents of each element of A or suppose we want to count the number of elements of A with a given property. This can be accomplished by traversing A that is by accessing and Processing each element of A exactly once.
The following algorithm traverses a linear array. The simplicity of the algorithm comes from the fact that LA is a linear structure. Other linear structures such as linked list can also be easily traversed. On the other hand the traversal of non-linear structures such as trees and graphs is considerably more complicated.

Algorithm:-
(Traversing a Linear Array) Here LA is a linear Array with lower Bound LB and upper Bound UB. This algorithm traverses LA.
Applying an operation PROCESS to each element of LA.
1. Repeat Step 3 and 4 while K¡=UB.
Visit element Apply PROCESS to LA X.
Increase Counter Set X=X+1
. End of Step 2 Loop
5. Exit.

Inserting and Deleting: -

Let A be a collection of data elements in the memory of computer. Inserting refers to the operation of adding another element to the collection A and deleting refers to the operation of removing one of the elements from A. Here we discuss the inserting and deleting when A is a linear array.

Inserting an element at the end of the linear array can be easily done provided the memory space allocated for the array is large enough to accommodate the additional element. On the other hand suppose we need to insert an element in the middle of the array. Then on average half of the elements must be moved downward to the new location to accommodate the new element and keep the order of other elements.

Similarly deleting the element at the end of an array presents no difficulties but deleting the element somewhere in the middle of the array would require that each subsequent element be moved one location upward in order to fill up the array.

Algorithm of Insertion operation: -

```
1. [Initialize Counter] Set J=N.
2. Repeat Step 3 and 4 while JK.
3. [Move Jth element downward] Set LA [J+1] =LA[J].
4. [Decrease Counter] Set J=J-1. End of Step 2 Loop.
5. [Insert element] Set LA[K]=ITEM.
6. [Reset N] Set N=N+1.
7. Exit.
```

2 Lab Task

Write a C++ program to implement all the above described algorithms and display the following menu and ask the user for the desired operation.

The program should have the option for reusing it after you have completed the desired task.

2.1 Program

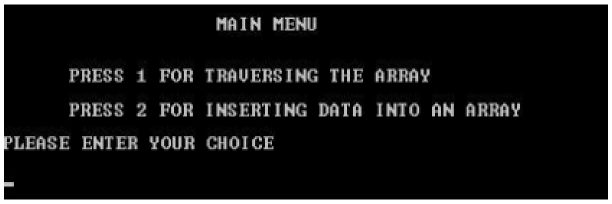
```
#include<iostream.h>
#include<conio.h>
void traversing (int*,int);
void insertion  (int*,int,int,int);
int  deletion   (int*,int,int);
void main()
{ clrscr();
  int array[30],choice,location,item,i,length,position;
  cout<<"Program for Operation of Array by-Arslan \n";
  cout<<"\nEnter length of array : ";
  cin>>length;
  cout<<"Enter element of array \n";
  for(i=0;i<=length-1;i++)
     { cin>>array[i];
     }
  again:
  cout<<"Travers= 1\nInsert = 2\nDelete = 3\nExit   = 4 \n";
  cout<<"Enter choice: ";
  cin>>choice;
  switch(choice)
        { case 1:          traversing(array,length);
          break;
   case 2:          cout<<"Enter Item and Position to add \n";
          cin>>item>>position;
          insertion(array,length,item,position);
          for(i=0;i<=length;i++)
          {cout<<array[i]<<endl;
          }
          break;
   case 3:          cout<<"Enter position to delete \n";
          cin>>position;
          length =deletion(array,length,position);
          cout<<"New array after delete \n";
          for(i=0;i<length;i++)
      {cout<<array[i]<<"\n";}
          break;
   case 4:          break;
   default:          cout<<"wrong value\n";
    goto again;
         }
getch();
}

void traversing(int array[],int length)
        {   for(int i=0;i<=length-1;i++)
       {cout<<array[i]<<"  ";}
        }


void insertion(int array[],int length,int item,int position)
       { int j=length-1;
         while(j>=position)
       {   array[j+1]=array[j];
         j=j-1;
       }
       array[position]=item;
       length=length+1;
       }

int deletion(int array[],int length,int position)
      {
         while(position<length)
      { array[position]=array[position+1];
        position=position+1;
      }
         length=length-1;
         return length;

       }
```



```
                    MAIN MENU

     PRESS 1 FOR TRAVERSING THE ARRAY
     PRESS 2 FOR INSERTING DATA INTO AN ARRAY
PLEASE ENTER YOUR CHOICE
_
```

Figure : 1 Output

3 Conclusion

Array provides efficient insertion and deletion of arbitrary elements. Deletion here is deletion by iterator, not by value. Traversal is quite fast. A dequeue provides efficient insertion and deletion only at the ends, but those are faster than for a Array, and traversal is faster as well.A set only makes sense if you want to find elements by their value, e.g. to remove them. Otherwise the overhead of checking for duplicate as well as that of keeping things sorted will be wasted. An array is useful for passing large sections of data at time, for instance, in a buffer of audio data or a networking packet.