IT224 LAB ASSIGNMENT 1

Design and Analysis of Algorithms

Submitted By

Name : Deepjyoti Deka

Roll No : 190103014

Semester: 4th Semester , Branch : CSE

Experiment 1

Aim : To print “Hello World” using C++.

Objective : To write a program that prints the string “Hello World” using inbuilt standard output function.

Theory

This program prints hello world, cout is used to display text on screen, '\n' places cursor on the beginning of next line, iostream header file contains declaration of Standard Input / Output Streams Library. The code will work on all operating systems may be t's Linux, Mac or any other and compilers.

Source Code

```
#include <iostream>
using namespace std;

int main()
{
 cout<<"Hello World \n";
}
```

Output

```
deep@deep-Inspiron-15-3567:~/4thSem/DAA$ g++ helloworld.cpp
deep@deep-Inspiron-15-3567:~/4thSem/DAA$ ./a.out
Hello World
deep@deep-Inspiron-15-3567:~/4thSem/DAA$
```

Result

The “Hello world” string is printed.

Experiment 2

Aim : To copy a string without using strcpy()

Objective

- To illustrate how to copy one string to another.
- To show this without the use of strcpy ()

Theory

Strcpy can be used to copy one string to another. As C strings are character arrays. We must pass character array, or pointer to character array to this function where string will be copied.. But here we are giving one string in input and then with the help of a for-loop we transfer the content of first array to the second array. The standard output function is used to print message on the screen. It is displayed using the inbuilt header file for input/output.

Source Code

```
#include <iostream>
using namespace std;

int main()
{
   int index = 0;
   char copy[40];
   char original[40];
   cout << "Enter the string to be copied! \n";
   cin >> original ;
   while(original[index] != '\0'){
       copy[index] = original[index];
       index++;
   }
   cout << "The copied string is \n";
   cout << copy << endl ;
}
```

Output

```
deep@deep-Inspiron-15-3567:~/4thSem/DAA$ ./a.out
Enter the string to be copied!
enter
The copied string is
enter
deep@deep-Inspiron-15-3567:~/4thSem/DAA$
```

Result

The word “enter” is copied by just using while loop.

Experiment 3

Aim : To concatenate two strings.

Objective : To show the process of combining two strings stored in two different variables.

Theory

The concatenation of two strings is done to form one string by appending the second string to the right-hand side of the first string. The strings are user input and the final combined string is displayed using the standard output function.

Source Code

```
#include <iostream>
using namespace std;

int main()
{

   char first[50];
   char second[50];
   char concat[50];

   cout << "enter first" <<endl;

   cin >> first;

   cout << "enter second" <<endl;

   cin >> second;

   int i , j;

   for( i=0;first[i]!='\0';i++){
       concat[i] = first[i];
   }
   for(int j=0;second[j]!='\0';j++)
   {
```

```
        concat[i] = second[j];
        i++;
    }

    cout << "The joined string is \n";
    cout << concat << endl ;

}
```

Output

```
deep@deep-Inspiron-15-3567:~/4thSem/DAA$ g++ string_concat_using_for.cpp
deep@deep-Inspiron-15-3567:~/4thSem/DAA$ ./a.out
enter first
Gau
enter second
hati
The joined string is
Gauhati
deep@deep-Inspiron-15-3567:~/4thSem/DAA$
```

Result

The strings Gau and hati are concatenated into Gauhati.

Experiment 4

Aim : To delete a character from an input string.

Objective

- To take two character as an input from a user.
- Delete the given character.
- Output the char variable after deletion using a for-loop.

Theory

The logic behind the algorithm is to iterate through the string and find the specified character to delete. The string array and character are taken as inputs. The deletion algorithm iterates through the array until it finds desired character, and then shifts the rest of the array one index towards left.

Source Code

```
#include <iostream>
using namespace std;

int main()
{
   char word[100],x;

   cout<<"enter the word \n";
   cin>> word;


   cout<<"Enter the character to delete \n";
   cin >> x;

   for (int i = 0 ; word[i]!='\0';i++)
   {
       if(word[i] == x){
```

```
            while(word[i+1] != '\0'){

                word[i] = word[i+1];
                i++;
            }

            word[i] = '\0';
            break;
        }
    }

    cout<< word << "\n";
}
```

Output

```
deep@deep-Inspiron-15-3567:~/4thSem/DAA$ g++ removeCharacterFromString.cpp
deep@deep-Inspiron-15-3567:~/4thSem/DAA$ ./a.out
enter the word
HELLO
Enter the character to delete
E
HLLO
deep@deep-Inspiron-15-3567:~/4thSem/DAA$
```

Result

The character E is deleted from the Hello resulting in HLLO.

Experiment 5

Aim : To delete a substring from a given string.

Objective

- Take string and substring (to delete) inputs from user.
- Formulate an algorithm to remove the substring from the string.
- Shifting the elements left to overwrite the substring.

Theory

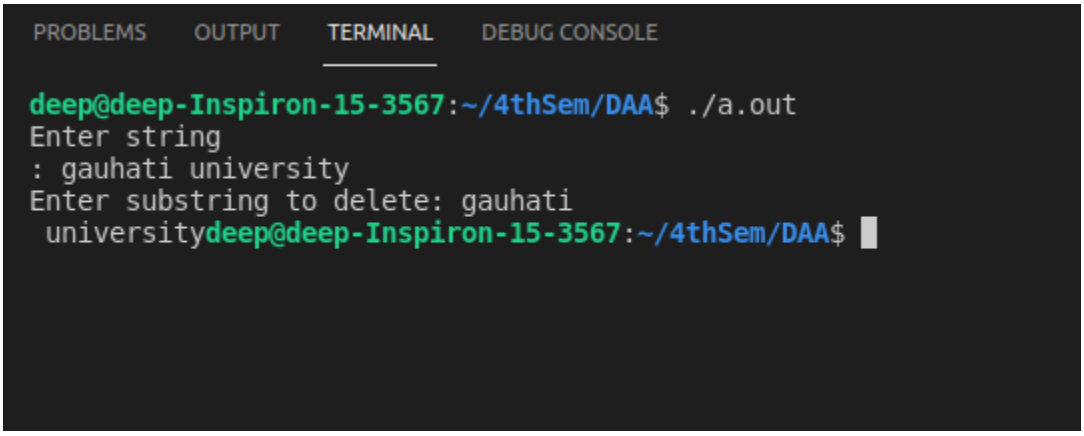
The goal of this experiment is to develop an algorithm to remove a given substring from a string. An approach towards this problem is shifting of elements to overwrite the substring.
The process involves iterating through the string in a loop until it finds the first matching letter of both the strings. Then another loop is executed to continue iterating until it is confirmed that all the corresponding letters of the substring are also present after the first match. If it is present, then that part of the string is overwritten by shifting the elements towards left, from the index after the end of the substring in the string.

Source Code

```
#include <iostream>
using namespace std;

int get_size(char del[])
{
   int x = 0;
   for (int i = 0; del[i] != '\0'; i++)
   {
       x++;
   }
   return x;
}

int main()
{
   char s[20], del[10];
```

```
    cout << "Enter string \n: ";
    cin.getline(s, 20);
    cout << "Enter substring to delete: ";
    cin >> del;
    int elem_size = get_size(del);
    int j = 0, count = 0;

    for (int i = 0; s[i] != '\0'; i++)
    {
        if (s[i] == del[j])
        {
            count++;
            j++;
            if (count == elem_size)
            {
                i = i - elem_size + 1;
                while (s[i + elem_size] != '\0')
                {
                    s[i] = s[i + elem_size];
                    i++;
                }
                s[i] = '\0';
                break;
            }
        }
        else
        {
            j = 0;
            count = 0;
        }
    }
    cout << s;
}
```

Output



Result

The string gauhati is deleted from the gauhati university resulting in university.

Experiment 6

Aim : To multiply two matrices.

Objective

• Using the logic of matrix multiplication, the product of two user inputted matrices is found.
• 3 for-loops are implemented for traversing/iterating through the rows and columns of the matrices.

Theory

In matrix multiplication, row elements of the first matrix is multiplied by the column elements of the second matrix. To multiply two matrices, the number of columns of the first matrix should be equal to the number of rows to the second matrix. The matrices are taken as inputs from the user.
For the multiplication, 3 for loops are used. The innermost for loop iterates throughout the current row and column of the first and second matrices respectively and adds the products of the corresponding elements. The penultimate loop is used for iterating throughout the columns of the second matrix, while outermost result is used for iterating throughout the rows of the first matrix.

Source Code

```
#include <iostream>
using namespace std;

int main()
{
   int a[10][10] , b[10][10] , c[10][10];
   int row , column;

   cout << "No of rows ? \n";
   cin>> row;
   cout << "No of column ? \n";
   cin>> column;
   cout << "Enter the first element \n";
```

```
    for(int i = 0 ; i < row ; i++){
        for(int j=0 ; j < column ; j++){
            cin>> a[i][j];
        }
    }
    cout << "Enter the second element \n";

    for(int i = 0 ; i < row ; i++){
        for(int j=0 ; j < column ; j++){
            cin>> b[i][j];
        }
    };
    cout << "Multiply the matrix \n";

    for(int i=0 ; i< row ; i++){
        for(int j=0 ; j < column; j++){

            c[i][j] = 0 ;

            for(int k=0 ; k<column ; k++){

                c[i][j] += a[i][k] * b[k][j];
            }
        }
    };
    cout << "printing the result\n";

    for(int i=0 ; i<row ; i++){
        for(int j=0; j<column ; j++){
            cout << c[i][j] << " " ;
        }
        cout << "\n";
    }
}
```

Output

```
deep@deep-Inspiron-15-3567:~/4thSem/DAA$ ./a.out
No of rows ?
2
No of column ?
2
Enter the first element
1
2
3
4
Enter the second element
1
2
3
4
Multiply the matrix
printing the result
7 10
15 22
deep@deep-Inspiron-15-3567:~/4thSem/DAA$
```

Result

The matrix a and b are multiplied using the matrix multiplication rules.

Experiment 7

Aim : To find the determinant of a universal matrix.

Objective

• Illustrating the use of recursion to find determinant of the sub matrices (minors).
• Using the inbuilt mathematical header file for computation of powers for getting cofactors of the reference row/column.

Theory

The value of determinant of a matrix can be calculated by following procedure –for each element of first row or first column get cofactor of those elements and then multiply the element with the determinant of the corresponding cofactor, and finally add them with alternate signs. As a base case the value of determinant of a 1*1 matrix is the single value itself.

Cofactor of an element, is a matrix which we can get by removing row and column of that element from that matrix.

Source Code

```
#include <iostream>
using namespace std;

int det(int n, int mat[10][10])
{
   int ans = 0;
   int sign = -1;
   int minor[10][10];
   if (n == 2){
       return ((mat[0][0] * mat[1][1]) - (mat[1][0] * mat[0][1]));
   }
   else
   {
       for (int i = 0; i < n; i++)
       {
           int row = 0;
```

```
            for (int j = 1; j < n; j++)
            {
                int col = 0;
                for (int k = 0; k < n; k++)
                {
                    if (k == i)
                        continue;
                    minor[row][col] = mat[j][k];
                    col++;
                }
                row++;
            }
            sign = sign * -1 ;
            ans = ans + ((sign) * mat[0][i] * det(n - 1, minor));
        }
         return ans;
    }
}


int main()
{
    int order;
    cout << "Enter order of the matrix \n";
    cin >> order;
    int a[10][10];
    cout << "Enter the elements:\n";
    for (int i = 0; i < order; i++)
    {
        for (int j = 0; j < order; j++)
        {
            cin >> a[j][i];
        }
    }
    cout << "The determinant is \n ";
    cout << det(order, a) << "\n";
    return 0;
}
```

Output

```
deep@deep-Inspiron-15-3567:~/4thSem/DAA$ ./a.out
Enter order of the matrix
3
Enter the elements:
1
2
3
4
5
6
7
8
9
The determiant is
 0
deep@deep-Inspiron-15-3567:~/4thSem/DAA$
```

Result

The determinant of the matrix is calculated using the rules recursively.

Experiment 8

Aim : To implement KMP search algorithm for pattern searching.

Objective :

• To illustrate the use of KMP search algorithm for finding out the number of times a pattern has occurred in a string, both of which are inputted by the user.

Theory

Pattern searching is an important problem in computer science. When we do search for a string in notepad/word file or browser or database, pattern searching algorithms are used to show the search results.

The Naive pattern searching algorithm doesn't work well in cases where we see many matching characters followed by a mismatching character.

The KMP (Knuth Morris Pratt) matching algorithm uses degenerating property (pattern having same sub-patterns appearing more than once in the pattern) of the pattern and improves the worst case complexity to O (n). The basic idea behind KMP's algorithm is: whenever we detect a mismatch (after some matches), we already know some of the characters in the text of the next window. We take advantage of this information to avoid matching the characters that we know will anyway match.

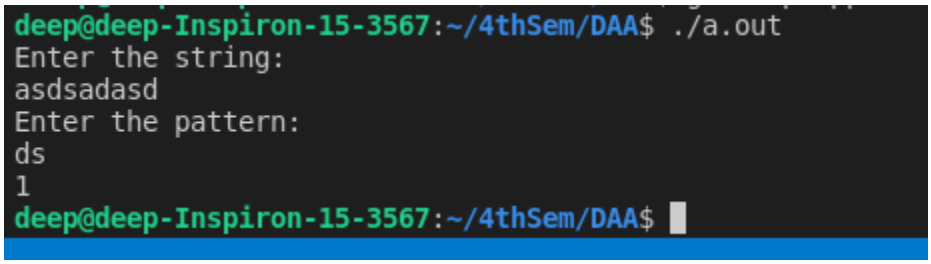
Source Code

```
#include <iostream>
using namespace std;

int kmp(char arr[], char pattern[], int p)
{
   int count = 0, i = 0, j = 0, flag;
   while (arr[i] != '\0')
   {
       j = 0;
       flag = 0;
```

```
        if (pattern[j] == arr[i])
        {
            while (pattern[j] == arr[i])
            {
                flag++;
                if (flag == p)
                {
                    count++;
                }

                i++;
                j++;
            }
        }
        if (j > 0)
        {
            continue;
        }
        else
        {
            i++;
        }
    }
    return (count);
}


int main()
{
    char arr[10], pattern[10];
    int patternSize = 0;
    int answer;
    cout << "Enter the string: "<<endl;
    cin >> arr;
    cout << "Enter the pattern: "<<endl;
    cin >> pattern;

    for (int i = 0; pattern[i] != '\0'; i++)
    {
```

```
        patternSize++;
    }


    answer =  kmp(arr, pattern, patternSize);
    cout << answer <<endl;

}
```

Output



Result

The algorithm successfully detects the pattern from the given string.

Experiment 9

Aim : To implement Array rotation & Array reverse functions.

Objective

- To shift the array towards left to the start from a given index as option 1.
- To reverse the array as option 2.
- Do-while loop implementation for repetition of the choice menu.
- Taking inputs from user.

Theory

Array rotation is basically the process of moving the positions of the elements towards one side. Left rotation means shifting towards left and vice versa. A simple approach towards formulating an algorithm for this problem would be to take a new array start filling the elements from the index inputted by the user in case of left rotation, which is currently what's aimed to achieve here.
Array Reverse is simply a process of reversing the arrangement of elements in an array. The algorithm for reversing the array involves using start and end as the pivot to move towards the middle while swapping the corresponding elements.

Source Code

```
#include<iostream>

using namespace std;

void leftRotateOne(int arr[], int n){
   int temp = arr[0], i;
   for (i = 0; i < n -1;i++){
       arr[i] = arr[i+1];
   }
   arr[n-1] = temp;
}

void leftRotate(int arr[], int d, int n){
   for (int i = 0; i < d; i++){
       leftRotateOne(arr,n);
   }
```

```
}

void leftRotation(int arr[], int d, int n){
   leftRotate(arr, d, n);
       for (int i=0;i<n;i++){
           cout << arr[i] << " ";
       }
}


int main(){

   int n=1, d , arr[n];
   cout << "Enter array size : " << endl;
   cin >> n;
   cout << "Enter the elements: " << endl;
   for (int i=0; i < n ;i ++ ){
       cin >> arr[i];
   }

   cout << "Before rotation: " << endl;
   for (int i=0;i<n;i++){
       cout << arr[i] << " ";
   }

   cout << "\n";

   cout << "Enter the position to rotate: " << endl;
   cin >> d;
   if (d == n){
       cout << "Enter position less than size of the array" << endl;
   }
   else{
       leftRotation(arr, d, n);
   }

}
```

Output

```
deep@deep-Inspiron-15-3567:~/4thSem/DAA$ ./a.out
Enter array size :
4
Enter the elements:
1
2
3
4
Before rotation:
1 2 3 4
Enter the position to rotate:
2
3 4 1 2 deep@deep-Inspiron-15-3567:~/4thSem/DAA$
```

Result

The rotation of the array 1 2 3 4 is applied at index 2 and result is 3 4 2 1 as intendent.

Experiment 10

Aim : To find size versus time graph mapping for Quicksort algorithm.

Objective

• To write a program to print the time required by a quicksort algorithm to sort the elements.
• Repeat the experiment for different values of n - the number of elements in the list to be sorted.
• Plot a graph of the Size versus Time taken.
• Use of a time recording library to capture start and end time of a process.

Theory

Quicksort is a sorting algorithm based on the divide and conquer approach where an array is divided into sub-arrays by selecting a random pivot element (element selected from the array). While dividing the array, the pivot element should be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side of the pivot. The left and right sub-arrays are also divided using the same approach. This process continues until each sub-array contains a single element. At this point the array will be sorted. The mapping of the size versus time is a visualisation of the increasing time with increasing size of the input size. The values of the times taken would differ from pc to pc.

Source Code

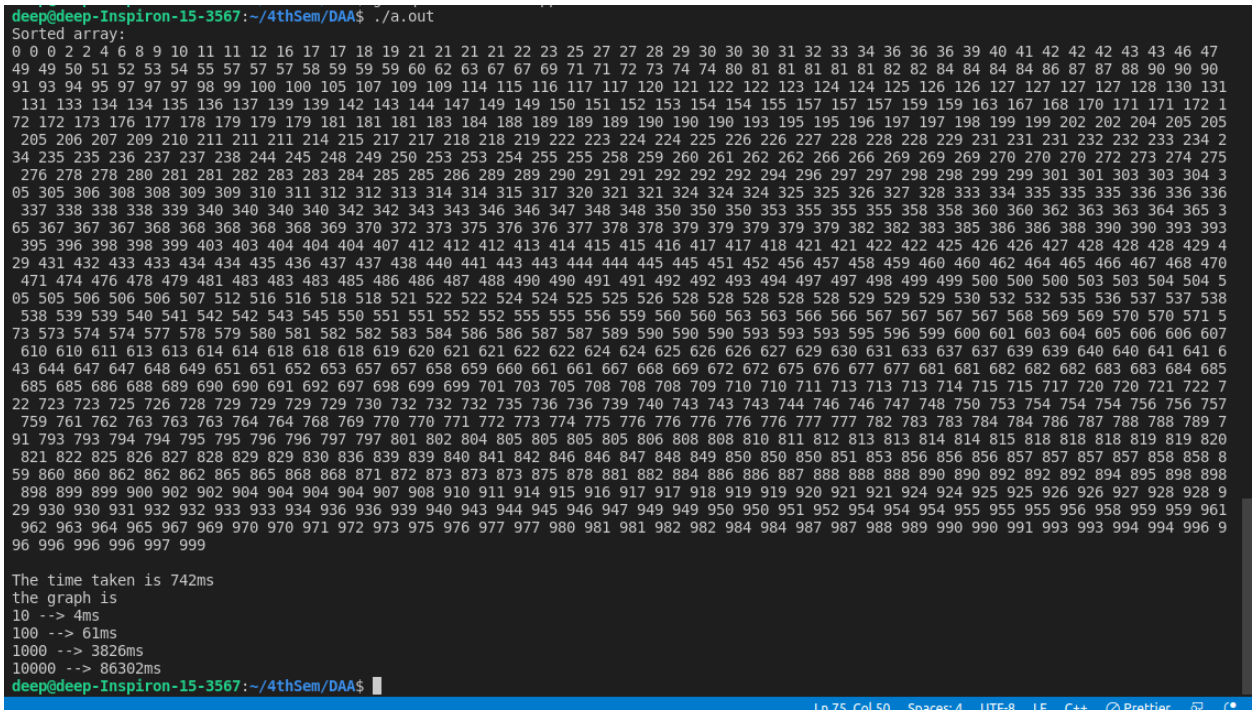
```
#include <iostream>
#include <chrono>
using namespace std;
using namespace std::chrono;

void swap(int *a, int *b)
{
   int t = *a;
   *a = *b;
   *b = t;
}

int partition(int arr[], int left, int right)
{
   int pivot = arr[right];
```

```
    int i = (left - 1);

    for (int j = left; j < right; j++)
    {

        if (arr[j] < pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }

    swap(&arr[i + 1], &arr[right]);
    return (i + 1);
}

void quickSort(int arr[], int left, int right)
{
    if (left < right)
    {

        int pivot = partition(arr, left, right);

        quickSort(arr, left, pivot - 1);
        quickSort(arr, pivot + 1, right);
    }
}
int graphtime(int arr[], int n)
{
    auto start = high_resolution_clock::now();
    quickSort(arr, 0, n - 1);
    auto end = high_resolution_clock::now();
    auto time_taken = duration_cast<microseconds>(end - start);
    return (time_taken.count());
}

void graph()
{
    int size[] = {10, 100, 1000, 10000};
```

```
    int x[4], k = 0;

    for (int n = 0; n < 4; n++)
    {
        int arr[size[n]];
        for (int j = 0; j < n; j++)
        {
            arr[j] = rand() % size[n];
        }
        x[k] = graphtime(arr, size[n]);
        k++;
    }

    for (int i = 0; i < 4; i++)
    {
        cout << size[i] << " --> " << x[i] << "ms" << endl;
    }
}

int main()
{
    int  array[1000];

    for (int i = 0 ; i<999 ; i++){
        array[i] = rand() % 1000;
    }

    int n = sizeof(array) / sizeof(array[0]);

    auto start = high_resolution_clock::now();
    quickSort(array, 0, n - 1);
    auto end = high_resolution_clock::now();
    auto timetaken = duration_cast<microseconds>(end-start);


    cout << "Sorted array: \n";

    for (int i = 0; i < n; i++)
```

```
        cout << array[i] << " ";
        cout << endl;

    cout << "\nThe time taken is " << timetaken.count() <<"ms"<<endl;

    cout<<"the graph is \n";

    graph();
}
```

Output



Result

1000 random elements are sorted and the graph of 10 , 100 , 1000 , 10000 elements is plotted..