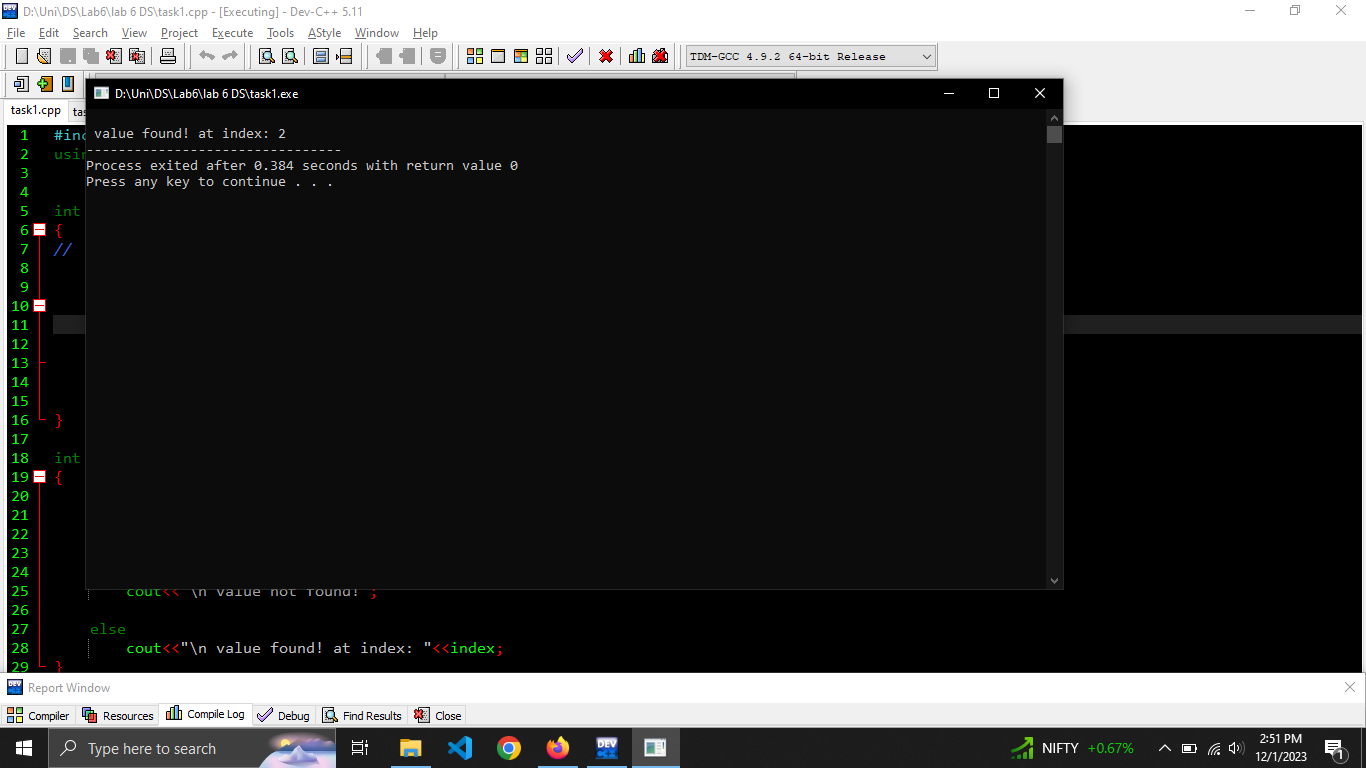
K226007

DS - Lab – 6

Manual 1

Task 1:



#include <bits/stdc++.h>

using namespace std;

void bubbleSort(int \*arr, int size)

{

int i, j, temp;

int swapped;

for(i=0; i<size-1; i++)

{

swapped = false;

for(j=0; j < size-i-1; j++)

{

if(arr[j]> arr[j+1])

{

swap(arr[j], arr[j + 1]);

swapped = true;

}

}

if (swapped == false)

break;

}

}

int BinarySearch(int arr[], int low, int high, int x)

{

int mid;

while(low <= high)

{

mid = (low + (high - low))/2;

if(arr[mid] == x)

{

return mid;

}

if(arr[mid] < x)

low = mid + 1;

else

high = mid - 1;

}

return -1;

}

int main()

{

int index = 0;

int arr[] = {2,2,43,5,1,6};

bubbleSort(arr, sizeof(arr)/sizeof(int));

index = BinarySearch(arr,0, sizeof(arr)/sizeof(int), 5);

if(index == -1)

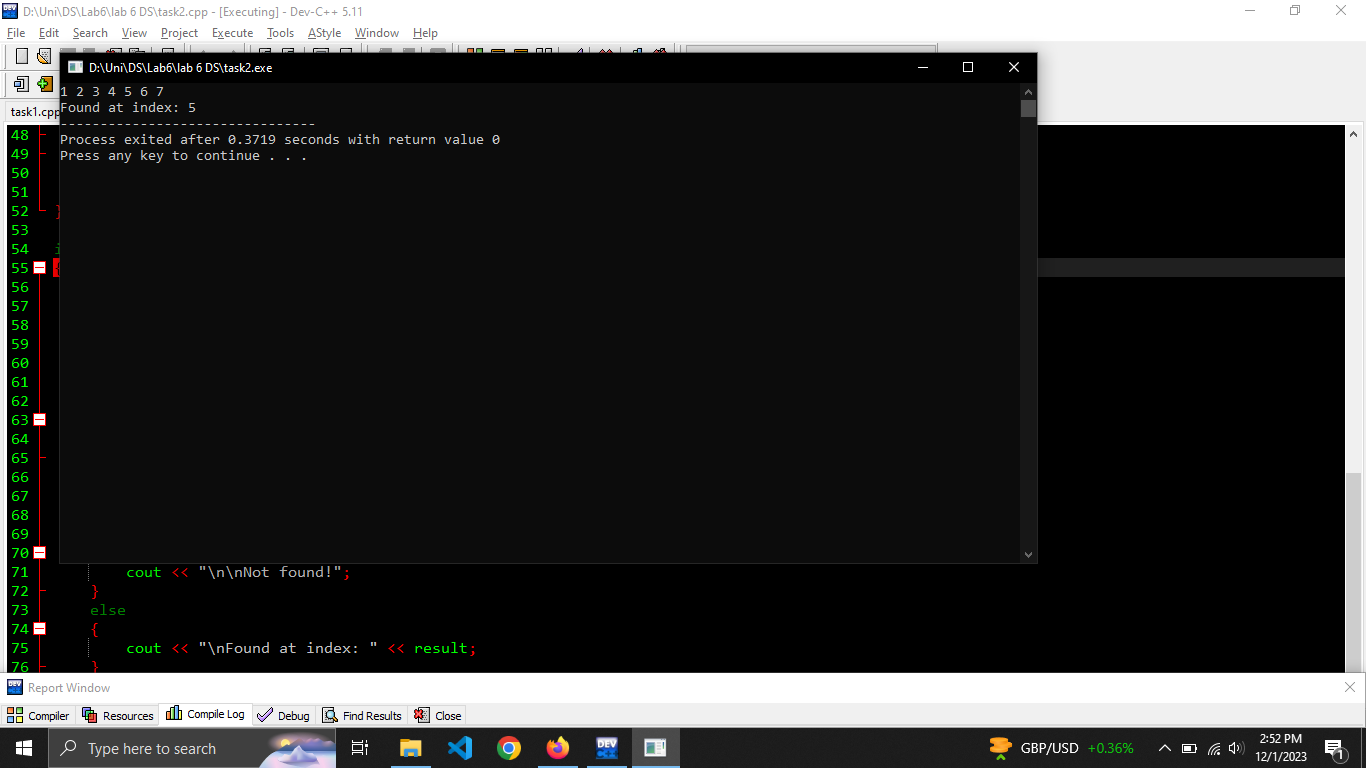
cout<<"\n value not found!";

else

cout<<"\n value found! at index: "<<index;

}

Task 2:



#include <bits/stdc++.h>

using namespace std;

void bubbleSort(int \*arr, int size)

{

int i, j;

for (i = 0; i < size; i++)

{

for (j = 0; j < size - i - 1; j++)

{

if (arr[j] > arr[j+1])

{

swap(arr[j], arr[j+1]);

}

}

}

}

int interpolationSearch(int arr[], int n, int key)

{

int low = 0;

int high = n - 1;

while (low <= high && key >= arr[low] && key <= arr[high])

{

if (low == high)

{

if (arr[low] == key)

{

return low;

}

return -1;

}

int pos = low + ((double)(key - arr[low]) / (arr[high] - arr[low])) \* (high - low);

if (arr[pos] == key)

{

return pos;

}

if (arr[pos] < key)

{

low = pos + 1;

}

else

{

high = pos - 1;

}

}

return -1;

}

int main()

{

int arr[] = {1, 3, 2, 4, 5, 6, 7};

bubbleSort(arr, sizeof(arr) / sizeof(int));

int size = sizeof(arr) / sizeof(arr[0]);

// cout<<size;

int key = 6;

for (int i = 0; i < size; i++)

{

cout << arr[i] << " ";

}

int result = interpolationSearch(arr, size, key);

if (result == -1)

{

cout << "\n\nNot found!";

}

else

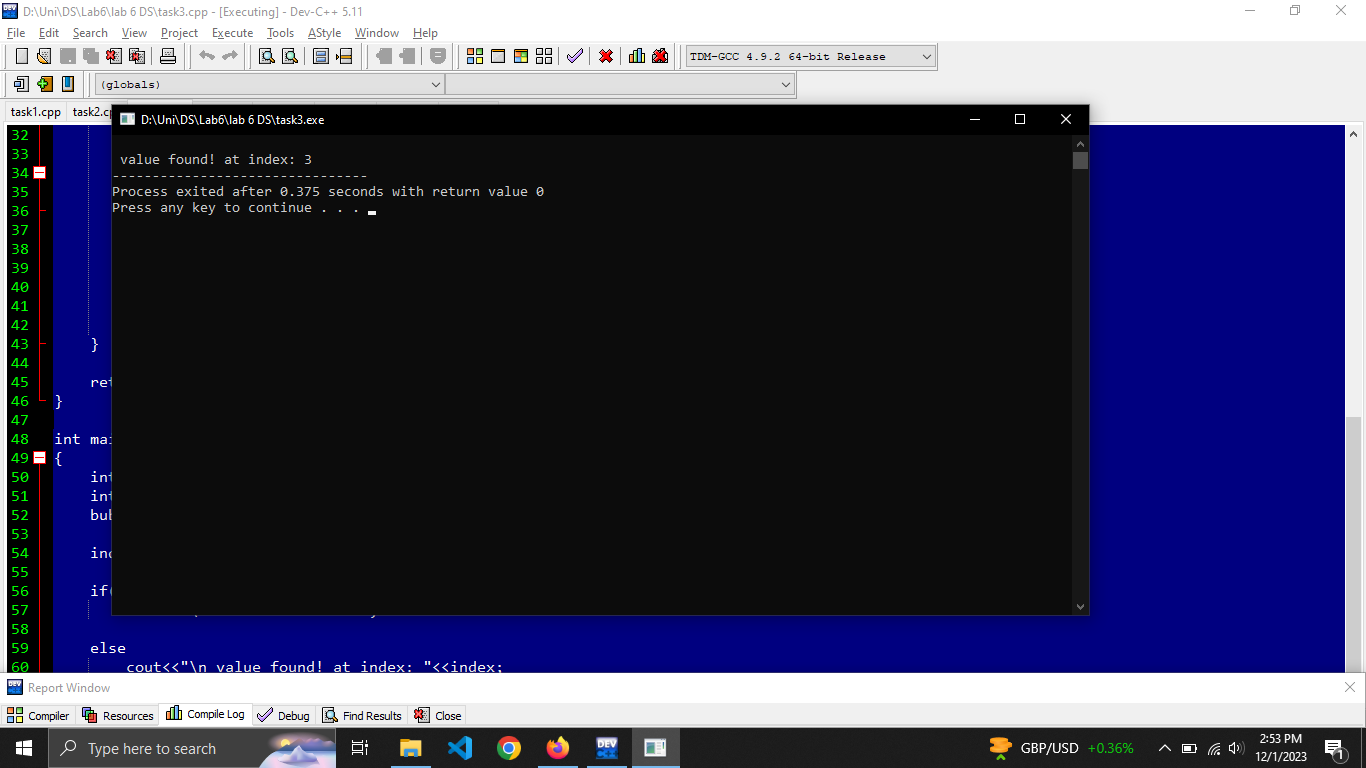
{

cout << "\nFound at index: " << result;

}

}

Task 3:



#include <bits/stdc++.h>

using namespace std;

void bubbleSort(int \*arr, int size)

{

int i, j, temp;

int swapped;

for(i=0; i<size-1; i++)

{

swapped = false;

for(j=0; j < size-i-1; j++)

{

if(arr[j]> arr[j+1])

{

swap(arr[j], arr[j + 1]);

swapped = true;

}

}

if (swapped == false)

break;

}

}

int BinarySearch(int arr[], int low, int high, int x)

{

int mid;

while(low <= high)

{

mid = (low + (high - low))/2;

if(arr[mid] == x)

{

return mid;

}

if(arr[mid] < x)

low = mid + 1;

else

high = mid - 1;

}

return -1;

}

int main()

{

int index = 0;

int arr[] = {2,2,43,5,1,6};

bubbleSort(arr, sizeof(arr)/sizeof(int));

index = BinarySearch(arr,0, sizeof(arr)/sizeof(int), 5);

if(index == -1)

cout<<"\n value not found!";

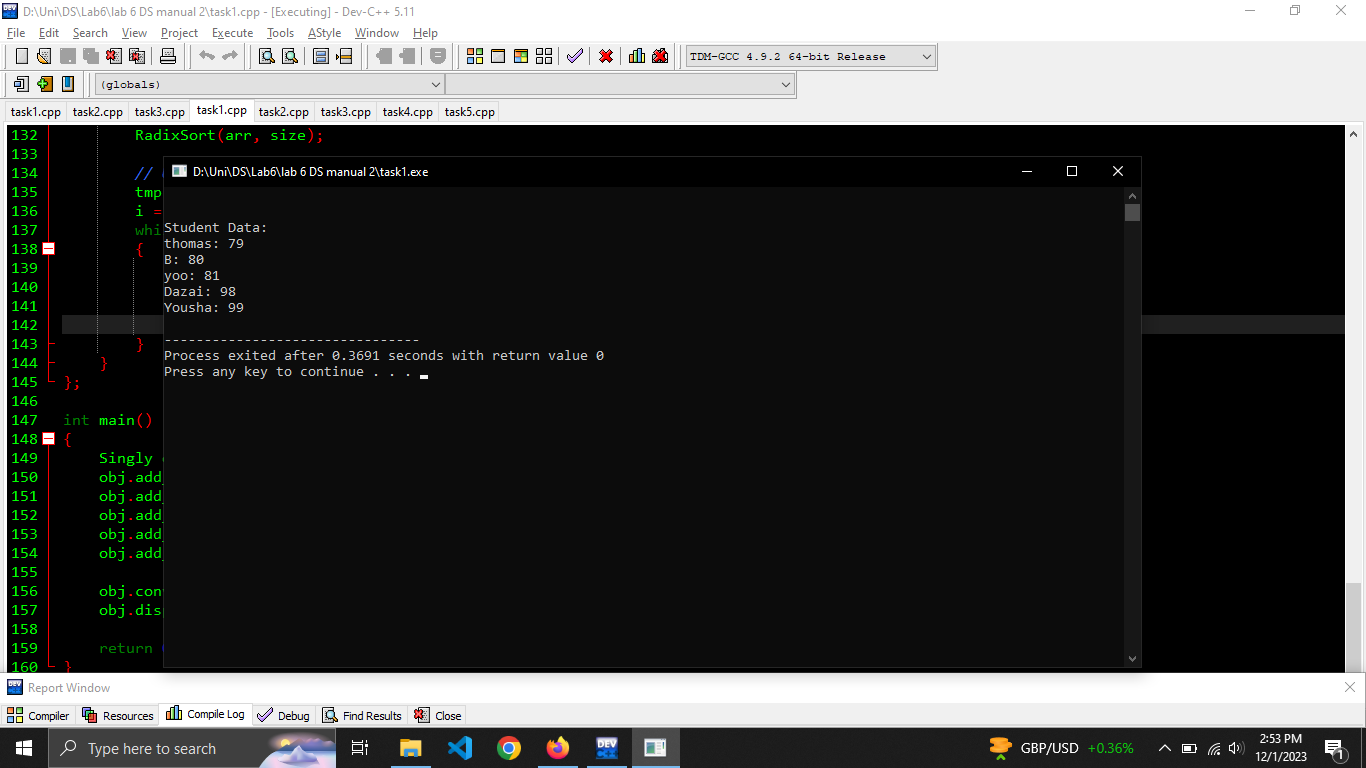
else

cout<<"\n value found! at index: "<<index;

}

Manual 2

Task 1:



#include <iostream>

using namespace std;

class Node

{

public:

string name;

int score;

Node\* next;

Node() {}

Node(string name, int score)

{

this->name = name;

this->score = score;

next = NULL;

}

};

class Array{

public:

string name;

int score;

};

class Singly

{

public:

Node\* head;

Node\* tail;

Singly()

{

head = NULL;

tail = NULL;

}

void add\_node(string name, int score)

{

Node\* tmp = new Node(name, score);

if (head == NULL)

{

head = tmp;

tail = tmp;

}

else

{

tail->next = tmp;

tail = tmp;

}

}

void display()

{

Node\* tmp = head;

cout << "\n\nStudent Data:\n";

while (tmp != NULL)

{

cout << tmp->name << ": " << tmp->score << endl;

tmp = tmp->next;

}

}

// Radix Sort

int get\_max(Array \*arr, int size)

{

int max = arr[0].score;

for (int i = 1; i < size; i++)

{

if (arr[i].score > max)

{

max = arr[i].score;

}

}

return max;

}

void count\_sort(Array \*arr, const int size, int div)

{

Array \*output = new Array[size];

int count[10] = {0};

for (int i = 0; i < size; i++)

{

count[(arr[i].score / div) % 10]++;

}

for (int i = 1; i < 10; i++)

{

count[i] += count[i - 1];

}

for (int i = size - 1; i >= 0; i--)

{

output[count[(arr[i].score / div) % 10] - 1].score = arr[i].score;

output[count[(arr[i].score / div) % 10] - 1].name = arr[i].name;

count[(arr[i].score / div) % 10]--;

}

for (int i = 0; i < size; i++)

{

arr[i].score = output[i].score;

arr[i].name = output[i].name;

}

}

void RadixSort(Array \*arr, int size)

{

int m = get\_max(arr, size);

for (int div = 1; m / div > 0; div \*= 10)

count\_sort(arr, size, div);

}

void convert\_array(const int size)

{

Node\* tmp = head;

Array \*arr = new Array[size];

int i = 0;

while (tmp != NULL)

{

arr[i].name = tmp->name;

arr[i].score = tmp->score;

tmp = tmp->next;

i++;

}

RadixSort(arr, size);

// Update scores in linked list

tmp = head;

i = 0;

while (tmp != NULL)

{

tmp->name = arr[i].name;

tmp->score = arr[i].score;

tmp = tmp->next;

i++;

}

}

};

int main()

{

Singly obj;

obj.add\_node("Yousha", 99);

obj.add\_node("B", 80);

obj.add\_node("thomas", 79);

obj.add\_node("yoo", 81);

obj.add\_node("Dazai", 98);

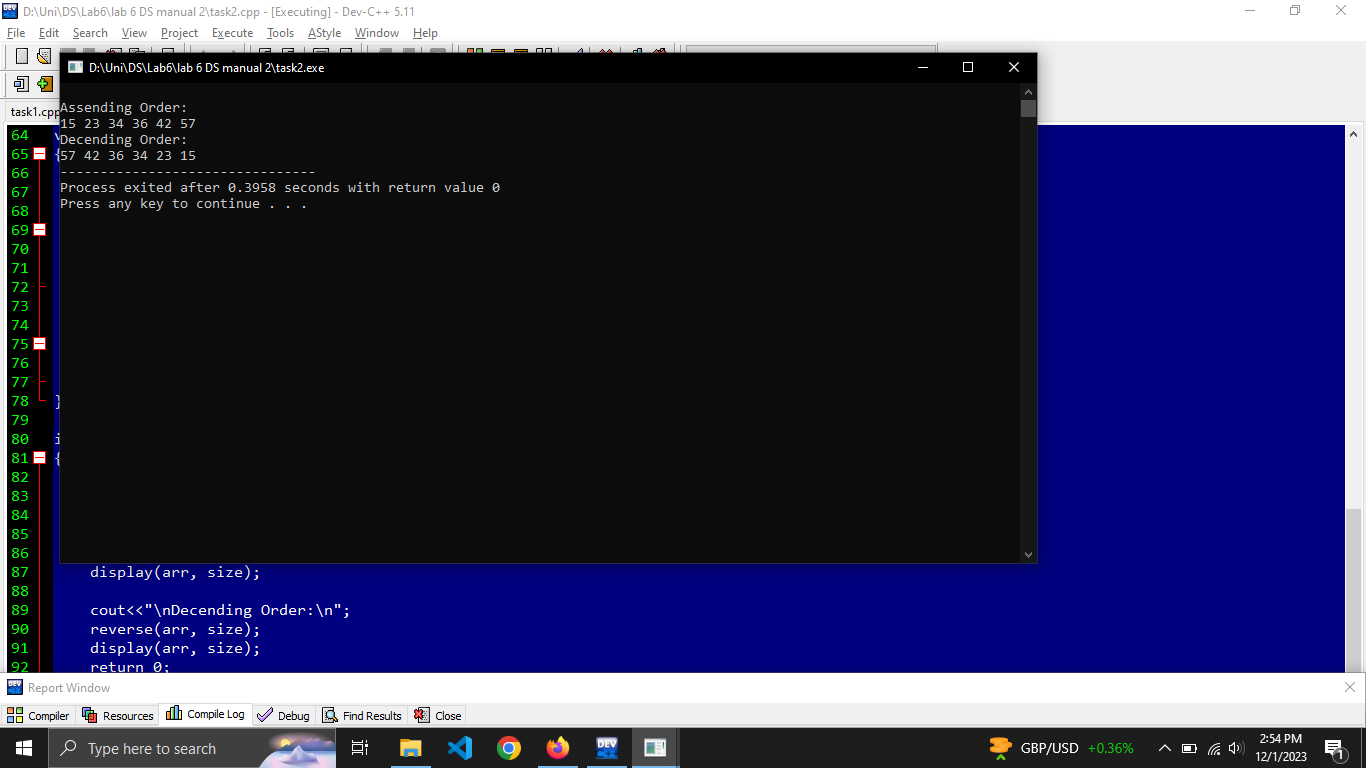
obj.convert\_array(5);

obj.display();

return 0;

}

Task 2:



#include <iostream>

using namespace std;

int get\_max(int arr[],const int size)

{

int max = arr[0];

for(int i=0; i<size; i++)

{

if(arr[i] > max)

{

max = arr[i];

}

}

return max;

}

void count\_sort(int arr[], const int size, int div)

{

int output[size];

int count[10] = {0};

for (int i = 0; i < size; i++)

{

count[(arr[i] / div) % 10]++;

}

for(int i = 1; i < 10; i++)

{

count[i] += count[i - 1];

}

for(int i = size - 1; i>=0; i--)

{

output[count[(arr[i]/div) % 10] - 1] = arr[i];

count[(arr[i]/div )%10]--;

}

for(int i=0; i<size; i++)

{

arr[i] = output[i];

}

}

void RadixSort(int arr[], const int size)

{

int m = get\_max(arr, size);

for(int div = 1; m/div > 0; div \*=10)

count\_sort(arr, size, div);

}

void display(int arr[], const int size)

{

for(int i=0; i<size; i++)

{

cout<<arr[i]<<" ";

}

}

void reverse(int arr[],const int size)

{

int tmp[size];

int j = 0;

for(int i=size-1; i>=0; i--)

{

tmp[j] = arr[i];

j++;

}

for(int i=0; i<size; i++)

{

arr[i] = tmp[i];

}

}

int main()

{

int arr[] = {23,34,42,15,57,36};

int size = sizeof(arr)/ sizeof(arr[0]);

RadixSort(arr, size);

cout<<"\nAssending Order:\n";

display(arr, size);

cout<<"\nDecending Order:\n";

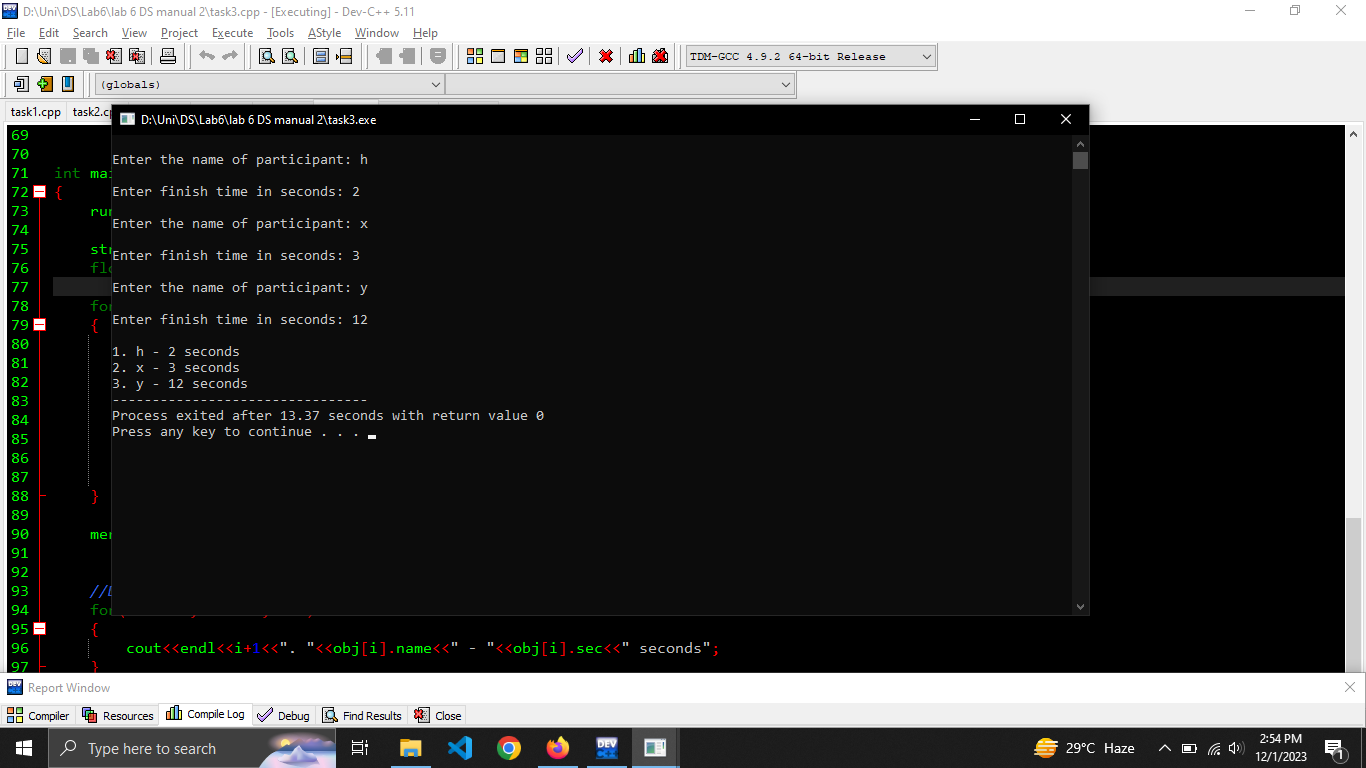
reverse(arr, size);

display(arr, size);

return 0;

}

Task 3:



#include <bits/stdc++.h>

using namespace std;

const int size = 3;

class runner

{

public:

string name;

float sec;

};

void merge(runner \*obj, int l, int m, int r)

{

int i = l;

int j = m+1;

int k = l;

int temp[size];

while(i<=m && j<=r)

{

if(obj[i].sec < obj[j].sec)

{

temp[k] = obj[i].sec;

i++;

k++;

}

else

{

temp[k] = obj[j].sec;

j++;

k++;

}

}

while(i<=m)

{

temp[k]= obj[i].sec;

i++;

k++;

}

while(j<=r)

{

temp[k] = obj[j].sec;

j++;

k++;

}

for(int s=l; s<=r; s++)

{

obj[s].sec = temp[s];

}

}

void mergeSort(runner \*obj, int l, int r)

{

if(l<r)

{

int mid = l+r/2;

mergeSort(obj, l , mid);

mergeSort(obj, mid+1, r);

merge(obj, l, mid, r);

}

}

int main()

{

runner \*obj = new runner[size];

string name;

float sec;

for(int i=0; i<size; i++)

{

cout<<"\nEnter the name of participant: ";

cin>>name;

cout<<"\nEnter finish time in seconds: ";

cin>>sec;

obj[i].name = name;

obj[i].sec = sec;

}

mergeSort(obj, 0, size-1);

//Display

for(int i=0; i<size; i++)

{

cout<<endl<<i+1<<". "<<obj[i].name<<" - "<<obj[i].sec<<" seconds";

}

}