**CPP\_Assignment\_9**

**I/O Generic Programming, Function and Class Templates Assignments**

**Mandatory Assignments**

1. Write a templated function to add() a maximum of 3 input strings OR 3 integers and return the final string or number. add() function may be called with atleast 2 arguments.

Ans:

#include<iostream>

using namespace std;

template<class t1>

t1 add(t1 a,t1 b,t1 c)

{

return a+b+c;

}

template<class t2>

t2 concat(t2 a,t2 b,t2 c)

{

return a+b+c;

}

int main()

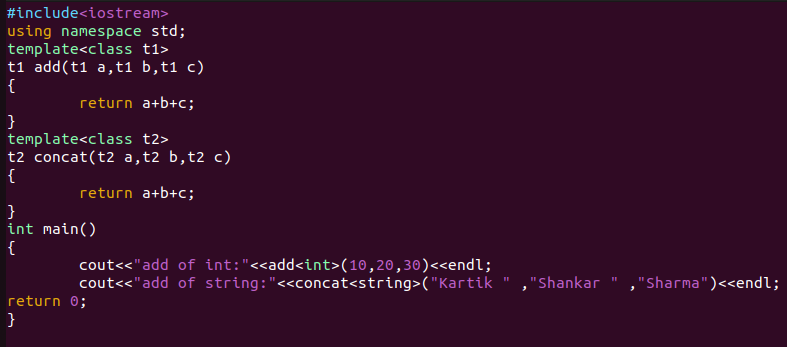
{

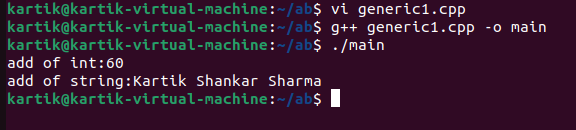
cout<<"add of int:"<<add<int>(10,20,30)<<endl;

cout<<"add of string:"<<concat<string>("Kartik " ,"Shankar " ,"Sharma")<<endl;

return 0;

}





1. Write a templated merge sort function which takes an array of data using merge sort with signature as below. Test with string and float data.

template <class T, int num>

int merge\_sort(T arr[], int num> //num is no. of elements in array

{

//Add implementation here and return SUCCESS i.e 0

}

Ans:

#include<iostream>

#include<vector>

using namespace std;

// template function to merge 2 componenets of the array, arr

// the function merges the sorted components [start, mid] and [mid+1, end]

// such that the resultant array is also sorted

// the function uses a temporary variable, temp to store the result

// after that the result is written back to the original array

template<typename T>

void Merge(T arr[], int start, int end)

{

// x: start index of first half [start, mid]

// y: start index of second half [mid+1, end]

// z: start index of temp

// temp: temporary vector to store the merged array

int z, x, y, mid;

vector<T> temp(end -start + 1);

mid = (start + end) / 2;

z = 0;

x = start;

y = mid + 1;

while (x <= mid && y <= end)

{

if (arr[x] < arr[y])

{

temp[z] = arr[x];

++x, ++z;

}

else

{

temp[z] = arr[y];

++y, ++z;

}

}

while (x <= mid)

{

temp[z] = arr[x];

++x, ++z;

}

while (y <= end)

{

temp[z] = arr[y];

++y, ++z;

}

// write the merged sequence back to the original array

for (int i = start; i <= end; ++i)

{

arr[i] = temp[i - start];

}

}

// template function to perform merge sort on array, arr

template<typename T>

void MergeSort(T arr[], int start, int end)

{

if (start < end)

{

int mid = (start + end) / 2;

MergeSort(arr, start, mid); // merge sort the elements in range [start, mid]

MergeSort(arr, mid + 1, end); // merge sort the elements in range [mid+1, end]

Merge(arr, start, end); // merge the above 2 componenets

}

}

// Template function to print array

// n: size of arr[]

template<typename T>

void PrintArray(T arr[], int n)

{

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

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

cout << "\n\n";

}

int main()

{

int arr[] = { 1, 10, 10, 90, -32, 100, -1, 11, 9, 14, 3, 2, 20, 19 };

int n = sizeof(arr) / sizeof(int);

cout << "Array Before Sorting: " << endl;

PrintArray(arr, n);

MergeSort(arr, 0, n - 1);

cout << "Array After Sorting: " << endl;

PrintArray(arr, n);

}

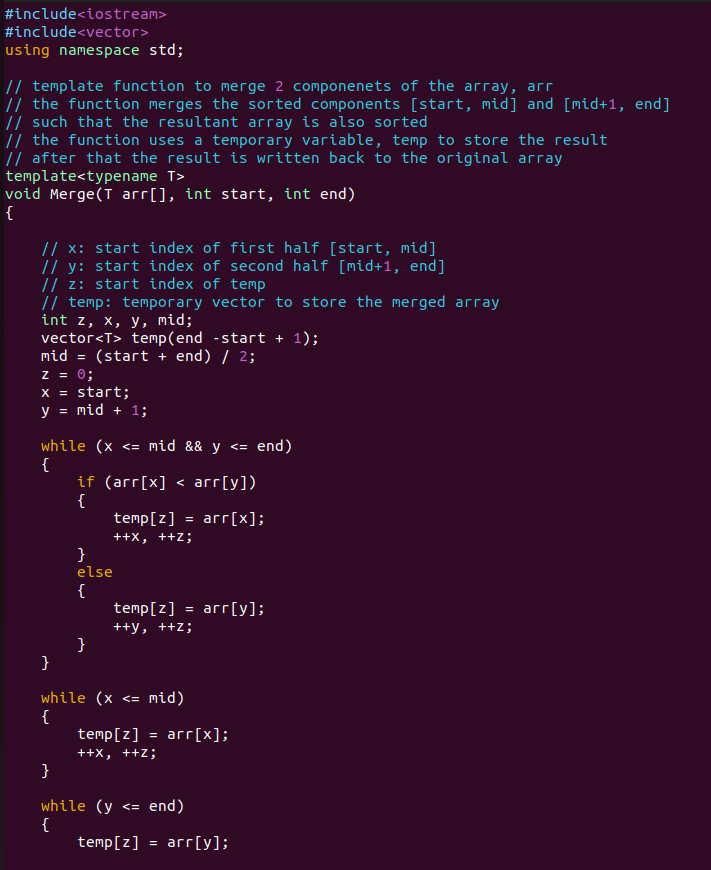
Output:

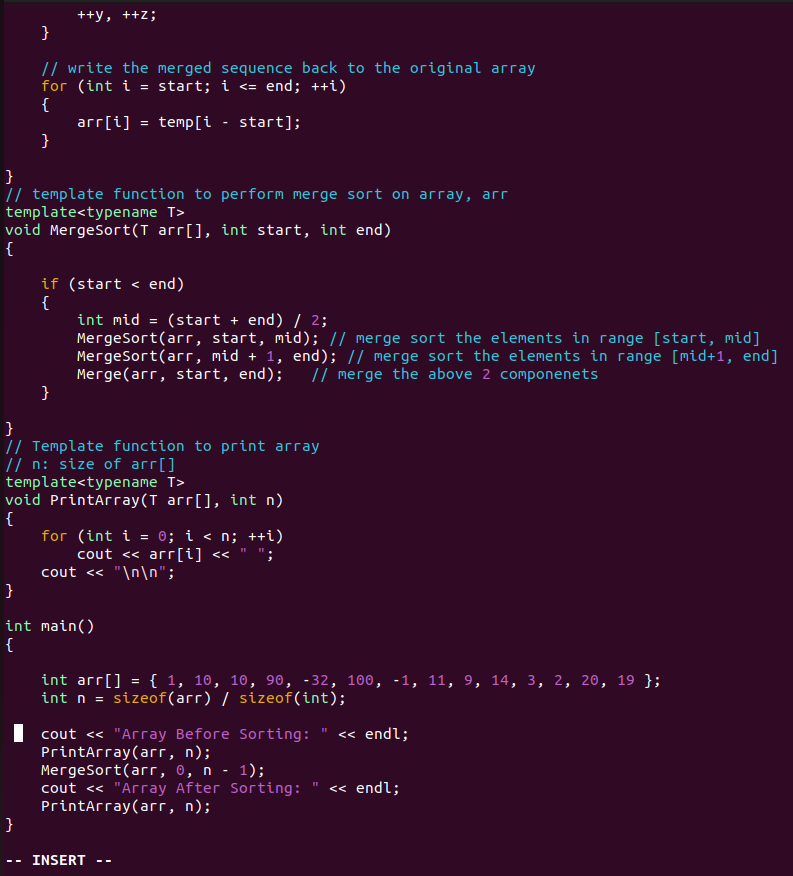
Array before sorting:

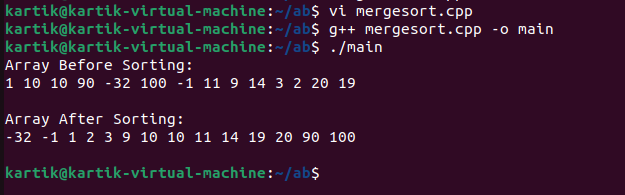
1 10 20 60 -6 -5 -65 54 87 32

Array after sorting:

-65 -6 -5 1 10 20 32 54 60 87







1. Refer the file “function\_template\_classobj.cpp”. Complete the sections marked with TBD 1..4x to get the expected out as mentioned in the file.

* TBD1: Constructor
* TBD2: Create object allocating memory from heap, handle exception
* TBD3: Implement template function myadd()
* TBD4a, b: Deallocate the assigned memory

Verify your output with expected output

1. Create a templated class named “MyArrayClass” with templated 1D array of MAX\_ELEMENTS.
   1. Populate the array using function below.

// populate MAX\_ELEMENTS in the 1D array with input arr content, count = no . of elements in input array

PopulateData(T \*arr, int count);

Ans:

#include<iostream>

#include<string>

using namespace std;

template<class t>

t MyArrayClass(t a[],int n)

{

t s=0;

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

{

s=s+a[i];

}

return s;

}

int main()

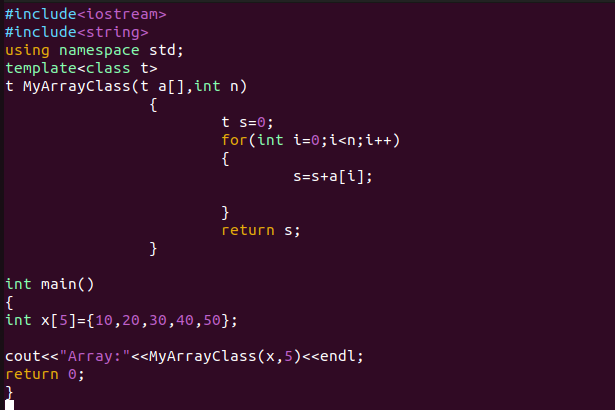
{

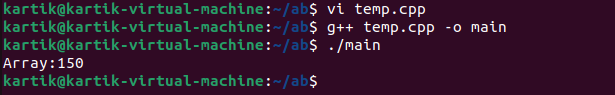
int x[5]={10,20,30,40,50};

cout<<"Array:"<<MyArrayClass(x,5)<<endl;

return 0;

}





* 1. For character data provide customized handling i.e convert given character to uppercase and store.

Ans:

#include<iostream>

#include<cstring>

using namespace std;

int main(){

char s[30];

cout<<"enter lowercase:";

cin>>s; /\* running the loop from 0 to the length of the string

\* to convert each individual char of string to uppercase

\* by subtracting 32 from the ASCII value of each char

\*/

for(int i=0;i<=strlen(s);i++)

/\* Here we are performing a check so that only lowercase

\* characters gets converted into uppercase.

\* ASCII value of a to z(lowercase chars) ranges from 97 to 122

\*/

{

if(s[i]>=97&&s[i]<=122)

{

s[i]=s[i]-32;

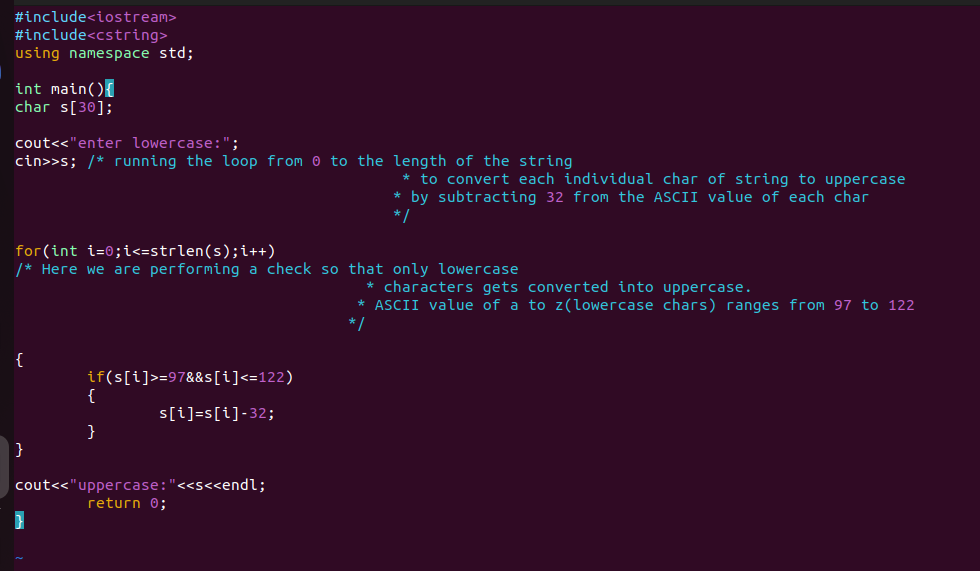
}

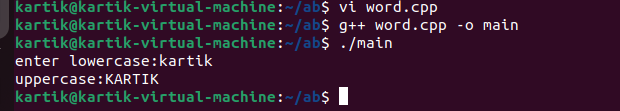
}

cout<<"uppercase:"<<s<<endl;

return 0;

}





* 1. Test with int, float and char data.

Ans:

#include<iostream>

using namespace std;

template<class t1>

t1 add(t1 a,t1 b,t1 c)

{

return a+b+c;

}

template<class t2>

t2 concat(t2 a,t2 b,t2 c)

{

return a+b+c;

}

template<class t3>

t3 addi(t3 a,t3 b,t3 c)

{

return a+b+c;

}

int main()

{

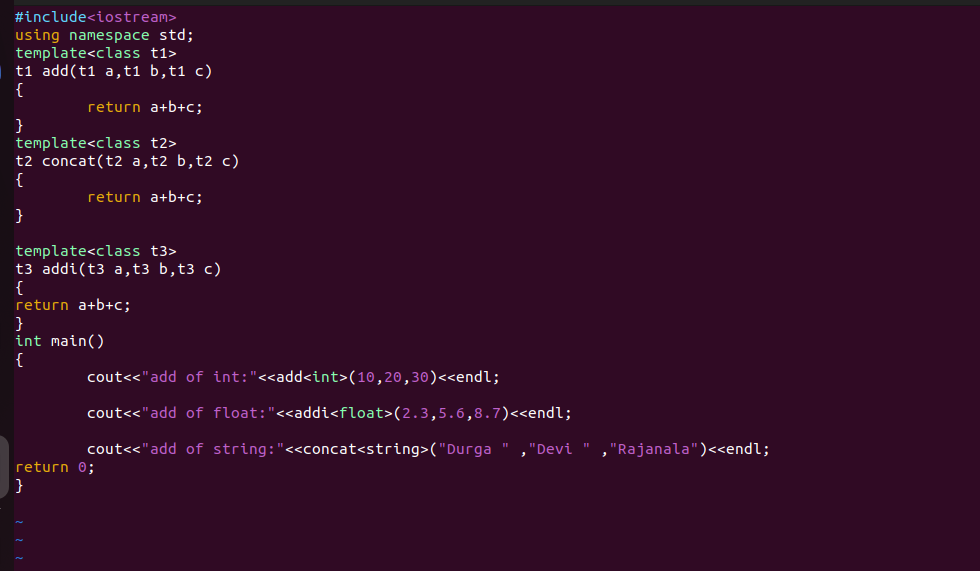
cout<<"add of int:"<<add<int>(10,20,30)<<endl;

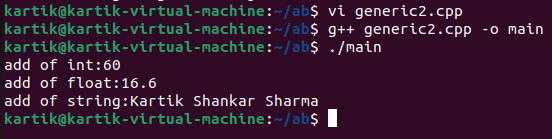
cout<<"add of float:"<<addi<float>(2.3,5.6,8.7)<<endl;

cout<<"add of string:"<<concat<string>("Kartik " ,"Shankar " ,"Sharma")<<endl;

return 0;

}





**Optional Assignments:**

Opt1. Implement a templated stack class with push(), pop(), isempty(), isfull() functions to work with any type of data . Test stack with string and integer data. Extract and display the object type of the stack objects created

Opt2. Implement a class Sort to sort any type of data. Test with char, integer, string datasets.