# COMP3512 - Assignment 2

Due: Sunday November 12th, 2017 @ 11:59 PM

This is an exercise that you need to do on a computer. You'll need to commit and push your code to your GitLab repo, and submit for automated marking via Slack.

For this assignment, you will need to write C++ utility methods of std::vector and std::map with Function Templates.

## 1. Project Setup

1. Open Assignment2.sln in Visual Studio 2017
2. Add Assignment2.h file to your project. (refer to Lab 1 if you don't know how)
3. Add the following content in the header file.

#pragma once

#include <iostream>

#include <vector>

#include <map>

#include <algorithm>

namespace assignment2

{

template <typename K, class V>

std::map<K, V> ConvertVectorsToMap(const std::vector<K>& keys, const std::vector<V>& data)

{

// implement ConvertVectorsToMap here

}

template <typename K, class V>

std::vector<K> GetKeys(const std::map<K, V>& m)

{

// implement GetKeys here

}

template <typename K, class V>

std::vector<V> GetValues(const std::map<K, V>& map)

{

// implement GetValues here

}

template <typename T>

std::vector<T> Reverse(const std::vector<T>& v)

{

// implement Reverse here

}

}

template <typename T>

std::vector<T> operator+(const std::vector<T>& v1, const std::vector<T>& v2)

{

// implement operator '+' here

}

template <typename K, class V>

std::map<K, V> operator+(const std::map<K, V>& m1, const std::map<K, V>& m2)

{

// implement operator '+' here

}

template<typename T>

std::ostream& operator<<(std::ostream& os, const std::vector<T>& v)

{

// implement operator '<<' here

}

template <typename K, class V>

std::ostream& operator<<(std::ostream& os, const std::map<K, V>& m)

{

// implement operator '<<' here

}

1. Note that your code may not compile because you have empty implementation.

## 2. Implement ConvertVectorsToMap method

* ConvertVectorsToMap method should take arguments const std::vector<K>& keys and const std::vector<V>& values and combine them into std::map<K, V>, where K and V can be type. For this exercise, assume that template parameter K will be primitive types.

EX>

Keys = [ ‘a’, ‘b’, ‘c’ ]

Values = [ 1, 2, 3 ]

Output = { ‘a’, 1 }, { ‘b’, 2 }, { ‘c’, 3 }

* If there are duplicate keys, it should ignore the second one.

EX>

Keys = [ ‘a’, ‘b’, ‘c’, ‘a’ ]

Values = [ 1, 2, 3, 4 ]

Output = { ‘a’, 1 }, { ‘b’, 2 }, { ‘c’, 3 }

* If the keys vector and values vector are in different length, the size of the map should be the Min(length of keys, length of values).

EX>

Keys = [ ‘a’, ‘b’, ‘c’, ‘d’ ]

Values = [ 1, 2, 3 ]

Output = { ‘a’, 1 }, { ‘b’, 2 }, { ‘c’, 3 }

EX2>

Keys = [ ‘a’, ‘b’, ‘c’ ]

Values = [ 1, 2, 3, 4 ]

Output = { ‘a’, 1 }, { ‘b’, 2 }, { ‘c’, 3 }

## 3. Implement GetKeys method

* GetKeys method should return the list of all keys of input std::map<K, V>.

EX>

Map = { ‘a’, 1 }, { ‘b’, 2 }, { ‘c’, 3 }

Output = [ ‘a’, ‘b’, ‘c’ ]

## 4. Implement GetValues method

* GetValues method should return the list of all values of input std::map<K, V>.

EX>

Map = { ‘a’, 1 }, { ‘b’, 2 }, { ‘c’, 3 }

Output = [ 1, 2, 3 ]

## 5. Implement Reverse method

* Reverse should return a reversed vector. The output must be a new vector (Do not reverse in place)

EX>

Vector = [ 1, 2, 3, 4 ]

Output = [ 4, 3, 2, 1 ] // Vector should still be [ 1, 2, 3, 4 ]

## 6. Implement Operator+ for two std::vector

* ‘+’ operator should combine two vectors together.

EX>

Vector1 = [ 1, 2, 3 ]

Vector2 = [ 4, 5, 6 ]

Output = [ 1, 2, 3, 4, 5, 6 ]

* Each value of a combined vector should be unique

EX>

Vector1 = [ 1, 2, 3 ]

Vector2 = [ 3, 2, 4 ]

Output = [ 1, 2, 3, 4 ]

* The operation should **NOT** be done in place (The operands should not change after the combination).

## 7. Implement Operator+ for two std::map

* ‘+’ operator should combine two maps together.

EX>

Map1 = { ‘a’, 1 }, { ‘b’, 2 }, { ‘c’, 3 }

Map2 = { ‘d’, 4 }, { ‘e’, 5 }, { ‘f’, 6 }

Output = { ‘a’, 1 }, { ‘b’, 2 }, { ‘c’, 3 }, { ‘d’, 4 }, { ‘e’, 5 }, { ‘f’, 6 }

* Duplicated keys should be ignored. The key-value pair in the first map (the first argument) takes precedence.

EX>

Map1 = { ‘a’, 1 }, { ‘b’, 2 }, { ‘c’, 3 }

Map2 = { ‘a’, 4 }, { ‘b’, 5 }, { ‘f’, 6 }

Output = { ‘a’, 1 }, { ‘b’, 2 }, { ‘c’, 3 }, { ‘f’, 6 }

* The operation should **NOT** be done in place (The operands should not change after the combination).

## 8. Implement Operator<< for std::vector

* << operator should print std::vector in the following format

EX>

Vector = [ 1, 2, 3, 4 ]

std::cout << Vector

Should exactly print this(no newline in the end):

1, 2, 3, 4

## 9. Implement Operator<< for std::map

* << operator should print std::map in the following format

EX>

Map = { ‘a’, 1 }, { ‘b’, 2 }, { ‘c’, 3 }, { ‘d’, 4 }

std::cout << Map

Should exactly print:

{ a, 1 }⤶

{ b, 2 }⤶

{ c, 3 }⤶

{ d, 4 }⤶

## 10. Test it

1. Add main.cpp into your project and test however you want. Below is a sample test you can use.

#include "Assignment2.h"

int main() {

std::vector<int> v1;

v1.push\_back(1);

v1.push\_back(2);

v1.push\_back(3);

std::vector<char> v2;

v2.push\_back('a');

v2.push\_back('b');

v2.push\_back('c');

std::vector<int> v3;

v3.push\_back(4);

v3.push\_back(5);

v3.push\_back(6);

std::map<char, int> m1;

m1['a'] = 1;

m1['b'] = 2;

m1['c'] = 3;

std::map<char, int> m2;

m2['d'] = 4;

m2['e'] = 5;

m2['f'] = 6;

std::map<int, char> m = assignment2::ConvertVectorsToMap(v1, v2);

std::vector<char> keys = assignment2::GetKeys(m1);

std::vector<int> values = assignment2::GetValues(m1);

std::vector<int> reversedV1 = assignment2::Reverse(v1);

std::vector<int> combinedVector = v1 + v3;

std::map<char, int> combinedMap = m1 + m2;

std::cout << v1 << std::endl;

std::cout << m1 << std::endl;

return 0;

}

1. Compile and run it to see the expected output.

## 11. Commit, Push and Ask for a Build

You know the drill :)

BUT, please note that

* build cooldown time for assignments is 30 mins instead of 5 mins.