Hi, congratulation guys! I believe you all doing good for midterm. Now let’s get a little idea about your assignment 4 before you go on a trip ( Is it due this weekend? Or next weekend.)

Here is just an example for the function “merge\_sort” which professor ask for. Generally it’s not easy to explain by words but you may try to read the code. I will add more comments later to help you understand. Once you read the code, you can try to write your own code for it and feel free to email me if you have any question during the break. I may not be able to meet but I can reply your questions by mail everyday.

You all do a really good job this semester and learn more than me when I took 121 class. I am so proud of you. It’s time to enjoy the break !

Explanation:

There are two functions in this assignment.

The first one is “**print\_vector**” which is a simple function. Just take a vector as parameter and print all the element of the vector by a for loop. You have many different way to loop the vector and I just give you three example ( you can try to use the second or third loop style which I comment them)

The second function is “**merge\_sort**” which need to return a sorted vector( which you create in the function ). Remember since the function need to return a value to the caller(main function), you have to create a same type vector to receive the return value ( this is what you need to do in **main** function )

Let’s check the implementation of “**merge\_sort**” function:

Since it is a recursive function, the first thing is defining the base case ( the stop condition ).

In this function we want the recursive call stop when your parameter only contain one value ( you can use the pre-defined function “size()” to see how many elements are there in the vector parameter )

vector<int> merge\_sort(const vector<int> &v)

{

**if(v.size() == 1) return v;**

**}**

The base case: if the vector parameter only contain one element (size = 1), the function directly return the vector back to the caller.

I didn’t check if the parameter is an empty vector ( size = 0 ), but if you want you can simply add the condition for example:

**if(v.size() == 0)**

**{**

**cout << "You can't sort an empty vector" << endl;**

**exit(1);**

**}**

In other words, if the vector parameter contain more than one element( size != 1 ), we need to do the following works by professor’s requirements:

1. If the vector contains only 1 element, return the vector unchanged.
2. Otherwise, split the vector into two halves, named left and right.
3. Recursively sort each half (i.e. call merge\_sort(left); and merge\_sort(right);).
4. Merge left and right into a new vector named sorted, in the following manner:
   1. As long as left and right both have elements not in sorted, compare the smallest such elements of each list, take the smaller of the two, and append it to sorted.
   2. Once all of the elements of either left or right are in sorted, take the leftover elements and append them tosorted.
5. Return sorted.

If you want to break the vector in half, you must find the middle position of the vector, I just find the instruction online to do this job, you can find many different method to do it:

**unsigned long mid = v.size()/2; //find the middle position by size divided by 2**

**//create a vector called left which contain the first half of the original vector v**

**vector<int> left(v.begin(), v.begin()+mid);**

**//create another vector called right which contain the second half of vector v**

**vector<int> right(v.begin() + mid, v.end());**

you need to know what inside the parentheses is the range of position in the original vector v like this :

**(v.begin(), v.begin()+mid) //left side of comma is the start position of the range, right side**

**of the comma is the end position of the range**

means you want from the first element of the vector v to the middle .begin() is the pre-defined function of vector which gives you the iterator(consider similar as a pointer) points to the first element of the vector, and you can move the pointer by add or sub the iterator just you use a pointer with a regular array

and you use the range of elements to initial the new vectors **left** and **right**

The third task is to recursively call the merge\_sort function to keep break the vector in half, remember the merge\_sort function is designed to return a vector<int>. Since it returns a vector, you need a receiver to receive the vector.

**left = merge\_sort(left);**

**right = merge\_sort(right);**

On the previous step, I break the original vector parameter into left and right, but I need to keep break each of them in half until they become size = 1. So you recursively break **left** and also break **right** and assign the result to it self. This part you need to really trace by hand to see what is the code doing, you can print out the vector left and right to see the order of work which the recursive call do. Change the code like this:

**left = merge\_sort(left);**

**print\_vector(left);**

**right = merge\_sort(right);**

**print\_vector(right);**

Then you will see the order how you break the vector(left first) until hitting the base case and start to break the vector(right).

After you break the vector, you need to merge them into a new vector called **sorted.** It checks for all the elements you have in vector **left** and **right** and put all the element in the vector(sorted).

**Notice at the same time you put the element in the new vector, you also sort value of the elements (this is what we do in these three while loop)**

**Then finally return the sorted vector to the caller ~**

**By SI rules I can’t provide the code for assignments and professor also doesn’t like students have exactly the same code to turn in which you miss the opportunity to learn.**

**But what you are learning for now is fairly not easy to start (I learn merge sort in 335 class). So hope this example can help you start and get the concept. Then by trying hard to do this assignment you will learn a lot. Feel free to ask me any questions and enjoy the break!**

#include <iostream>

#include <vector>

using namespace std;

void print\_vector(const vector<int> &v)

{

unsigned long n = v.size(); // or you want int n = v.size();

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

{

cout << v[i] << " ";

}

cout << endl;

//the second way to loop the vector

// for(auto el : v)

// {

// cout << el << " ";

// }

// cout << endl;

//

//The third way to loop the vector(by iterator)

// for (auto it = v.begin(); it != v.end(); ++it)

// {

// cout << \*it << " ";

// }

// cout << endl;

}

vector<int> merge\_sort(const vector<int> &v)

{

if(v.size() == 1) return v;

**//if size != 1 (greater than 1) then run the following code**

vector<int> sorted;

unsigned long mid = v.size()/2;

vector<int> left(v.begin(), v.begin()+mid);

vector<int> right(v.begin() + mid, v.end());

left = merge\_sort(left);

right = merge\_sort(right);

vector<int>::iterator v\_left = left.begin();

vector<int>::iterator v\_right = right.begin();

while(v\_left != left.end() && v\_right != right.end())

{

if(\*v\_left < \*v\_right)

{

sorted.push\_back(\*v\_left);

v\_left++;

}

else

{

sorted.push\_back(\*v\_right);

v\_right++;

}

}

while(v\_left != left.end())

{

sorted.push\_back(\*v\_left);

v\_left++;

}

while(v\_right != right.end())

{

sorted.push\_back(\*v\_right);

v\_right++;

}

return sorted;

}

int main(int argc, const char \* argv[]) {

vector<int> v = {8,2,3,4,5,6,7,1};

vector<int> receiver = merge\_sort(v); **//receiver is the same type as your function return**

print\_vector(receiver);

return 0;

}