LAB-C17 Merging Sorted vectors

V. Matos © 2022

In this assignment, you will complete a partially implemented C++ application. Your main task is to merge two *sorted vectors* containing pointers to Person objects.

Refer to the figure below for an example. The initial portion illustrates the contents of vectors v1 and v2, both sorted in ascending order based on name values. The final portion displays the resulting vector, v3, which is formed by merging v1 and v2 while maintaining the ascending order of names.

```
Vector v1
0  00000241FE9EA130 Person [Salutation:Mrs. Name:Daisy]
1  00000241FE9E8120 Person [Salutation:Mrs. Name:Robinson]
2  00000241FE9E9E60 Person [Salutation:Mr.  Name:Rodgers]
3  00000241FE9E9B90 Person [Salutation:Mr.  Name:T]
Vector v2
0  00000241FE9F5240 Person [Salutation:Ms.  Name:America]
1  00000241FE9F8070 Person [Salutation:Mr.  Name:Bond]
2  00000241FE9F8100 Person [Salutation:Mr.  Name:Clean]
3  00000241FE9F8190 Person [Salutation:Mrs. Name:Doubtfire]
4  00000241FE9F8220 Person [Salutation:Mr.  Name:Magoo]
5  00000241FE9F82B0 Person [Salutation:Mr.  Name:Spock]
Vector v3 = v1 + v2
0  00000241FE9F5240 Person [Salutation:Ms.  Name:America]
1  00000241FE9F8070 Person [Salutation:Mr.  Name:Bond]
2  00000241FE9F8100 Person [Salutation:Mr.  Name:Clean]
3  00000241FE9EA130 Person [Salutation:Mrs. Name:Daisy]
4  00000241FE9F8190 Person [Salutation:Mrs. Name:Doubtfire]
5  00000241FE9F8220 Person [Salutation:Mr.  Name:Magoo]
6  00000241FE9E8120 Person [Salutation:Mrs. Name:Robinson]
7  00000241FE9E9E60 Person [Salutation:Mr.  Name:Rodgers]
8  00000241FE9F82B0 Person [Salutation:Mr.  Name:Spock]
9  00000241FE9E9B90 Person [Salutation:Mr.  Name:T]
```

Figure 1. A run-sample of the Merging app

Follow the steps below to complete the assignment:

Step 1: Implement the missing code for the Person class provided in Appendix A. Specifically, you need to overload the << operator for output and the < operator for comparison.

Step2. You must implement the generic method

```
template <class T>
vector<T*> mergeVectors(vector<T*> v1, vector<T*> v2)

{

... //Your code goes here!

}
```

Vectors v1 and v2 are generic containers of type vector<T*>, holding pointers to objects of an arbitrary class T. To ensure proper comparison of T-type instances, the class T must provide an overloaded operator<. The function should return a vector<T*> containing pointers to the merged and sorted combination of objects from v1 and v2.

Step 3: Implement a generic function to print the contents of a vector that holds pointers to objects of type T. Specifically, you will define the following method:

```
template<class T>
void showVector(string msg, vector<T*> v) {

...//Your code goes here

}
```

Here, msg is a descriptive caption, such as "Vector v1", that provides context for the output. The parameter v is a vector containing pointers to objects of type T.

As a precondition, assume that class T includes overloaded versions of both the `operator<` for comparisons and the `operator<<` for formatted output. These overloaded operators ensure that the vector's elements can be correctly sorted and displayed.

Caution: Do not use any sorting algorithms to accomplish this task. The merging process should maintain the order of the input vectors without additional sorting.

Below is a sample main() function that you should use to test your application.

```
int main()
{
	vector<Person*>  v1
	{
		new Person("Mrs.", "Daisy"),
		new Person("Mrs.", "Robinson"),
		new Person("Mr.",  "Rodgers"),
		new Person("Mr.",  "T"),

	};
	showVector("Vector v1", v1);

	vector<Person*>  v2
	{
		new Person("Ms.",  "America"),
		new Person("Mr.",  "Bond"),
		new Person("Mr.",  "Clean"),
		new Person("Mrs.", "Doubtfire"),
		new Person("Mr.",  "Magoo"),
		new Person("Mr.",  "Spock")

	};
	showVector("Vector v2", v2);
```

```
	vector<Person*> v3 = mergeVectors(v1, v2);

	showVector("Vector v3 = v1 + v2", v3);

	//TODO - cleanup (reléase memory)

}
```

Appendix A. Incomplete version of the Person class

```
class Person
{
private:
	string salutation;
	string name;

public:
	// Getter-Setters (Only what you absolutely need!)

	// Lazy constructor(s)
	Person(string salutation = "n.a", string name = "n.a")
	{
		//TODO
	}

	//user-defined methods
	void print() {
	   // TODO
	 }

	YOUR CODE FOR:  OPERATOR<  GOES HERE


	YOUR CODE FOR:  OPERATOR<<  GOES HERE


	//Destructor
	~Person()
	{
		cout << this << " Person  Destructor called " << endl;
	}
};
```