Data Structures & Algorithms

Lab Exercise 1 – Templates and STL Sequence Containers

Due: to be demonstrated by week 3 (wk beginning 24-Sep)

Learning outcomes
At the end of this lab you should be able to:

- Write basic template functions and classes.
- Use STL vectors (insert, remove, find and iterate over elements).
- Demonstrate how iterators work with STL containers.
- Demonstrate how the remove/erase idiom works.
- Implement a lambda function

Q1. Write a template function called `scalarProduct()` to compute the scalar (or dot) product of two mathematical vectors. Each vector should be represented by an array, where x is stored in index 0, y in index 1 and z in index 2. The scalar product is:

$a_o b_o + a_1 b_1 + a_2 b_2$

The function should return the scalar product as a single value. Test your function with different numeric types.

Q2. Consider the implementation of an unordered array in UnorderedArray.h (on blackboard) Write a new member function:

```
int search(T val)
```

to return the index of the first occurrence of val in the array. If val does not exist, return -1.

Q3. Further to Q2, write another member function:

```
void remove(int index)
```

to remove the element at the specified index (hint: this means copying each array element one place to the left from position `index` to the end of the array). If the array index is negative or >= max. size of the array, the function should do nothing. Member variable `m_numElements` must be updated as appropriate.

Q4. The STL `std::pair` class is introduced in this question. First take a look at this URL which has a short example illustrating how `std::pair` is used:

http://en.cppreference.com/w/cpp/utility/pair/pair

Now create a function `minMax()` that can find the minimum and maximum elements of a vector. The function should return a `std::pair` of two elements - with the minimum element in the first

position and the maximum element in the second position. The function should be able to find the min and max of a vector of any type, where the elements in the vector are comparable, for example:

X > Y suggests the relational operator > can be applied to both X and Y.

Test your function separately with a vector of a numeric type and a vector of strings.

Q5. Populate a vector with 100 numbers between 1 and 9 as follows:

Index	Number
0	0
1	1
..	..
9	9
10	0
11	1
12	2
..	..
99	9

(use a loop to do this). Prompt the user for a target number between 1 and 9.

(i) Use the STL algorithm `find` as part of your solution to remove all occurrences of the target number from the vector.

(ii) Using a different approach to (i), implement the *erase-remove* idiom to remove all occurrences of the target number from the vector (use a functor). Compare the two solutions. Which is better? Why?

(iii) Comment out your solution for (ii) and rewrite it using a lambda function instead of a functor.

Further reading and documentation for the STL algorithms:

http://www.cplusplus.com/reference/algorithm/

Q6. Consider the following class:

```
class Gamer
{
public:
	Gamer() {}

	// Destructor function
	~Gamer()
	{
		std::cout << "Destructor called" << std::endl;
	}

	// Define a copy constructor
	Gamer(Gamer const & copy)
```

```
	{
		std::cout << "Copy constructor called" << std::endl;
	}
};
```

(i) Create a vector and fill it with objects of type Gamer. Print the size and capacity of your vector. Call the `clear()` function on your vector (look up the `clear()` function to see what it does).

(ii) Print the size and capacity after calling `clear()`. Can you explain what has happened?

(iii) Redefine your vector so it can store pointers to type Gamer, i.e.

```
std::vector<Gamer *>
```

Create a few instances of Gamer on the heap, and store the address of those objects in the vector. Now what happens when you call the `clear()` function?