Tutorial 7

Question 1

The following function accepts an int argument and returns half of its value as a double:

```
double half(int number)
{
     return number / 2.0;
}
```

Write a template that will implement this function to accept an argument of any type

Question 2

As the following Rectangle class is written, the width and length members are doubles. Rewrite the class as a template that will accept any data type for these members.

```
class Rectangle
{
     private:
          double width;
          double length;
     public:
          void setData(double w, double l)
               { width = w; length = l;}
          double getWidth()
               { return width; }
          double getLength()
               { return length; }
          double getArea()
               { return width * length; }
};
```

Question 3

Write templates for the two functions minimum and maximum. The minimum function should accept two arguments and return the value of the argument that is the lesser of the two. The maximum function should accept two arguments and return the value of the argument that is the greater of the two. Design a simple driver program that demonstrates the templates with various data types.

Question 4

Write a program that dynamically allocates an array large enough to hold a user defined number of temperatures. Once all the temperature are entered, the array should be passed to a function that sorts them in ascending order. Another function should be called that calculates the average temperature. The program should display the sorted list of tenperature and averages with appropriate headings. Use pointer notation rather than array notation whenever possible. Input Validation: Do not accept negative numbers for temperature.

Question 5

Modify the code in question 4 to use an STL vector instead of a dynamically allocated array. Refer to tables 1 and 2 in appendix of this documents.

Question 6

The following function uses reference variables as parameters. Rewrite the function so it uses pointers instead of reference variables, and then demonstrate the function in a complete program:

```
int doSomething(int &x, int &y)
{
int temp = x;
x = y * 10;
y = temp * 10;
 return x + y;
}
```

Appendix A

Member Function	Description
`size()`	Returns the number of elements in the `vector`.
`push_back()`	Accepts as an argument a value to be inserted into the `vector`. The argument is inserted after the last element. (Pushed onto the back of the `vector`.)
`pop_back()`	Removes the last element from the `vector`.
`operator[]`	Allows array-like access of existing `vector` elements. (The `vector` must already contain elements for this operator to work. It cannot be used to insert new values into the `vector`.)

Table 1 :Vector class template member functions

Table 2: More Vector class member functions
Please use what you need only

Member Function	Examples and Description
`at(element)`	Returns the value of the element located at *element* in the vector. *Example:* `x = vect.at(5);` This statement assigns the value of element 5 of vect to x.
`back()`	Returns a reference to the last element in the vector. *Example:* `cout << vect.back() << endl;`
`begin()`	Returns an iterator pointing to the vector's first element. *Example:* `iter = vect.begin();`
`capacity()`	Returns the maximum number of elements that may be stored in the vector without additional memory being allocated. (This is not the same value as returned by the `size` member function.) *Example:* `x = vect.capacity();` This statement assigns the capacity of `vect` to `x`.
`clear()`	Clears a vector of all its elements. *Example:* `vect.clear();` This statement removes all the elements from `vect`.

`empty()`	Returns true if the vector is empty. Otherwise, it returns false.
	Example:
	`if (vect.empty())`
	`  cout << "The vector is empty.";`
	This statement displays the message if `vect` is empty.

`empty()`	Returns true if the vector is empty. Otherwise, it returns false.
	Example:
	`if (vect.empty())`
	`  cout << "The vector is empty.";`
	This statement displays the message if `vect` is empty.

`end()`	Returns an iterator pointing to the vector's last element. *Example:* `iter = vect.end();`
`erase()`	Causes the vector element pointed to by the iterator `iter` to be removed. *Example:* `vect.erase(iter);`
`erase(itor1, iter2)`	Causes all the vector elements from the iterator `iter1` to the iterator `iter2` to be removed. *Example:* `vect.erase(firstIter, secondIter);`
`front()`	Returns a reference to the vector's first element. *Example:* `cout << vector.front() << endl;`

`insert (iter, value)`	Inserts a value into a `vector`. *Example:* `vect.insert(iter, 22);` This statement inserts the value 22 into `vect`. The value is inserted into the element before the one pointed to by `iter`.
`resize(n, value)`	Resizes a vector by *`n`* new elements. The elements are initialized with *`value`*. *Example:* `vect.resize(10, 0);` This statement adds ten new elements to `vect` and initializes the new elements with the value 0.
`reverse()`	Reverses the order of the items stored in a vector. *Example:* `vect.reverse();`