Assignment 4

Deadline:	Monday 5th October 2015
Evaluation:	10 marks – which represents 4% of your final grade
Late Submission:	1 mark per day late
Teams:	The assignment can be done individually or in teams of up to four people (one assignment per team submitted including all student names in the source code).
Purpose:	Practice with designing and using template classes, memory allocation.

Problem to solve:
You must design a template class named `Vector` that models a dynamic array. You may assume that the type `T` supports the assignment operator (`operator=`). The file **a4.h**, containing the declaration of the template class `Vector`, is provided for you on the stream page. You should write all definitions for the members of the class `Vector`, test your solution and submit your completed file **a4.h** for marking.

Requirements:
You should create the template class `Vector` with a default type `int`.

`Vector` class must have the following **private data members**:

`int mCapacity`
`mCapacity` is the maximum number of items that can be stored in the `Vector` before reallocation occurs.

`int mSize`
`mSize` is the actual number of items in the `Vector`.

`T *mData`
`mData` is a pointer to an array of items of type `T`.

`Vector` class must have the following **constructors**:

`Vector()`
Default constructor to create an empty `Vector` - `mSize` and `mCapacity` should be set to zero and no memory should be allocated for the array (make sure that the value of `mData` is not arbitrary).

`Vector(int size)`
Custom constructor that creates a `Vector` with `size` items. Each item is initialised by its default constructor. The constructor should not act as a type conversion.

`Vector(int size, const T &value)`
Custom constructor that creates a `Vector` with `size` items. Each item is initialised to `value`.

`Vector(int size, const T *data)`
Custom constructor that creates `Vector` of `size` items. Each item is initialised by the value of the corresponding item in the array `data`.

`Vector(const Vector<T> &other)`
Copy constructor – create a deep copy of the other `Vector`.

`Vector(Vector<T> &&other)`
Move constructor – steal the data from the other `Vector`.

Vector class must have **destructor**:

`~Vector()`
Delete any memory still held by the `Vector`.

Vector class must have the following **Member functions**:

`const Vector<T>& operator=(const Vector<T> &other)`
Assignment operator, assign one `Vector` to another. This operator should create a copy of each element. You should checks for self assignment and allocate memory if needed.

`Vector<T>& operator=(Vector<T> &&other)`
Move semantics assignment operator, steal the data from one `Vector` and give it to another. You should checks for self assignment and allocate memory if needed.

`T& operator[](int i)`
Returns the value at index `i`, the method should check for a range error (both a const and non-const version).

`void assign(int first, int last, const Vector<T> &other)`
Assigns the range of items `first` to `last` (inclusive) should be copied from the `Vector other` and assigned to this `Vector` at the same indexes. Should check the range of `first` to `last` in both the source and destination vector and assign memory if necessary. Gaps are not allowed in the destination `Vector`.

`void insert(int i, const T& value)`
Inserts `value` into the Vector at position `i` (if an element is inserted at position 0, then it should be first element in the vector). The items from `i` should be shuffled along. If the vector is full, allocate additional space before inserting the item.

`void erase(int i)`
Erase the item at position `i`, and shuffle the other elements accordingly. Be sure to check the range.

`void erase(int first, int last)`
Erase the items between `first` to `last` (erase operation should be inclusive – the elements at index first and last should be erased) and shuffle the rest of the items accordingly. Be sure to check for the range.

`void clear()`
Clear the items in the `Vector` the size should be 0 after this operation. Does not necessarily delete memory.

`int length() const`
Returns the number of actual items in the `Vector`.

`int memorySize() const`
Returns the size of the memory space allocated for the `Vector`.

`bool isEmpty() const`
Returns true if the `Vector` has no items and false otherwise.

Extra Instructions

1. Place the following comments at the top of your program code and **provide the appropriate information** (for **every** member working in the team):

```
// Family Name, Given Name, Student ID, Assignment number, 159.234
/* explain what the program is doing . . . */
```

2. Create the function *`displayInfo`* as shown below and **provide the appropriate information**:

```
void displayInfo() {
   cout << "----------------------------------------" << endl;
   cout << "Assignment 4 Semester 2 2015"            << endl;
   cout << " Submitted by: Mouse, Mickey, 12345678"  << endl;
   cout << "          and  Duck,  Donald, 98765432"  << endl;
   cout << "----------------------------------------" << endl;
   cout << endl;
}
```

The *displayInfo* should be the first thing that you display on the screen. If I supply the *main()* then you are responsible only for the implementation of the function *displayInfo* and I will call it in my *main()*.

Hand-in: Submit **a4.h** electronically on the course stream site.
If you have any questions about this assignment, please ask the lecturer.