1. Programming Practice – Programming Style (C++)

1. Names of functions, classes, structures, variables and constants
   - Use names which are informative, concise, memorable, and pronounceable
   - The broader the scope of a variable, the more information should be conveyed by its name, use descriptive names for globals, short names for locals
   - Functions, classes and structures should have names that suggest their role in a program
   - Use consistent naming conventions,
     (i) Active names for functions, e.g. get_time
     (ii) Names of pointers that begin or end with p or ptr, e.g. head_ptr
     (iii) Global variables start with capital letters, e.g. Table
     (iv) Constants are written with capitals, e.g. INITIAL_SIZE
     (v) Private variables start with an underscore, e.g. _size
   - Define constant numbers as constants, use `const` keyword

2. Expressions and statements
   - Indent to show structure
   - Use the natural form for expressions
   - Parenthesize to resolve ambiguity
   - Break up complex expressions
   - Use conventional idioms
   - Use cascaded if-else statements for multi-way decisions
   - Use the language to calculate the size of an object

3. Comments
   - Write short comments that help to read the program
   - Comments should add information that is not evident from the code or that is spread through the source
   - Comments should not report self-evident information
   - Comment functions and global data

2. C++: STL vector class, STL merge algorithm, STL sorting algorithm, timing, recursive version of Merge-Sort algorithm in pseudo-code

Standard Template Library of C++, merge algorithm, sort algorithm, timing
Basic components of STL: containers, algorithms and iterators

Online documentation & links
http://www.sgi.com/tech/stl/
Merge algorithm:

```cpp
OutputIterator merge(InputIterator1 first1, InputIterator1 last1,
                     InputIterator2 first2, InputIterator2 last2,
                     OutputIterator result);
```

Merge combines two sorted ranges `[first1, last1)` and `[first2, last2)` into a single sorted range. That is, it copies elements from `[first1, last1)` and `[first2, last2)` into `[result, result + (last1 - first1) + (last2 - first2))` such that the resulting range is in ascending order. The return value is `result + (last1 - first1) + (last2 - first2)`.

Example:
The code is located in the directory stl_merging

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    typedef vector<int> IntVector;
    typedef IntVector::iterator IntVectorIt;

    size_t n,m;
    cout << "Enter the size of the first array for merging" << endl;
    cin >> n;
    cout << endl;
    cout << "Enter the size of the second array for merging" << endl;
    cin >> m;
    cout << endl;

    //Allocate vectors of sizes n, m and n+m
    //Define iterators and their initial and terminal values
    IntVector Vector1(n);
    IntVectorIt start1,end1,it1;
    start1 = Vector1.begin();
    end1 = Vector1.end();

    IntVector Vector2(m);
    IntVectorIt start2,end2,it2;
    start2 = Vector2.begin();
    end2 = Vector2.end();
```
IntVector Vector3(n+m);
IntVectorIt start3,end3,it3;
start3 = Vector3.begin();
end3 = Vector3.end();

//Enter n consecutive even numbers into vector 1
int i=0;
for(it1=start1;it1!=end1;it1++)
{
    *it1=2*i;
    i++;
}

//Output numbers stored in vector 1
cout << "The content of vector 1" << endl;
for(it1=start1;it1!=end1;it1++)
    cout << *it1 << " ";
cout << endl;

//Enter m consecutive odd numbers into vector 2
i=0;
for(it2=start2;it2!=end2;it2++)
{
    *it2=2*i+1;
    i++;
}

//Output numbers stored in vector 2
cout << "The content of vector 2" << endl;
for(it2=start2;it2!=end2;it2++)
    cout << *it2 << " ";
cout << endl;

//Perform merging of vectors 1 and 2 into vector 3
merge(start1,end1,start2,end2,start3);

//Print out the result of merging
cout << "The content of vector 3 after merging" << endl;
for(it3=start3;it3!=end3;it3++)
    cout << *it3 << " ";
cout << endl;

return 0;
Sort Algorithm:

```cpp
void sort(RandomAccessIterator first, RandomAccessIterator last);
```

**Sort** sorts the elements in [first, last) into ascending order, meaning that if i and j are any two valid iterators in [first, last) such that i precedes j, then *j is not less than *i.

Example:
The code is located inside directory stl_sorting

```cpp
#include <ctime>
#include <cstdlib>
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    typedef vector<int> IntVector;
    typedef IntVector::iterator IntVectorIt;

    size_t n;

    //Seed the random-number generator with current time so that
    //the numbers will be different every time we run.
    srand((unsigned)time(NULL));

    cout << "Enter the size of the random array to be sorted" << endl;
    cin >> n;
    cout << endl;
    IntVector NumbersVector(n);
    IntVectorIt startv,endv,itv;
    startv = NumbersVector.begin();
    endv = NumbersVector.end();
    for(itv=startv;itv!=endv;itv++)
        *itv=rand()%n;

    //Start timing sorting procedure
    clock_t start=clock();
sort(startv,endv);
```
//Stop timing sorting procedure
clock_t stop=clock();

//Print out elapsed time in miliseconds
cout << "The elapsed time: ";
cout << 1000*(stop-start)/CLOCKS_PER_SEC << endl;

return 0;
}

Merge Sort algorithm - a recursive version with arrays, without implementation of merge function

void mergesort(int a[], int s, int r)
{
    if(r<=s) return;
    int m=(r+s)/2;
    mergesort(a, s, m);
    mergesort(a, m+1, r);
    merge(a, s, m, r);
}

Merge operation merge(a, s, m, r) performs merging of the segments a[s..m], a[m+1,r] and stores the result in a[s..r].

Example:
s=0, r=11, a=[8,7,6,5,2,3,4,11,9,10,12,1]
The tree of recursive calls:

```
  s=0, r=11, m=5
     /   \
  s=0, r=5, m=2  s=6, r=11, m=8
     /   \
  s=0, r=2, m=1  s=6, r=8, m=7
     /   \
  s=0, r=1, m=0  s=6, r=7, m=6
    /   \
  s=0, r=0  s=7, r=7
    /   \
  s=1, r=1  s=9, r=9
    /   \
  s=3, r=3  s=10, r=10
    /   \
  s=3, r=4  s=11, r=11
```
<table>
<thead>
<tr>
<th>Level 0</th>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 7 6 5 2 3 4 11 9 10 12 1</td>
<td>3 4 11 9 10 12 1</td>
<td>5 4 9 11 1 10 12</td>
<td>8 1 4 9 10 11 12</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8 9 10 11 12</td>
<td>2 3 4 5 6 7 8 9 10 11 12</td>
<td>6 7 8 9 10 11 12</td>
<td></td>
</tr>
</tbody>
</table>