Initial Self-Assessment Lab Test

1. If I want to run a single-file program `program.cc`, do I need to first compile and generate the object `program.o` with `g++ -c program.cc`, and then link to produce the executable with `g++ program.o`?

   Solution:

   One *could* do that, but in this case it can all be done in a single line:

   ```
   g++ program.cc
   ```

2. If I want to use any of the features of the C++ 2011 standard in my program `program.cc`, how should I compile it with `g++`?

   Solution:

   ```
   g++ -std=c++11 program.cc
   ```

   Note that `std` stands for "standard".

3. How can I tell `g++` to warn me against everything that is found suspicious during compilation?

   Solution:

   ```
   g++ -Wall program.cc
   ```

   Note that `Wall` stands for "Warning all". It is wise to *always* use the `-Wall` flag.

4. How can I tell `g++` to optimize the generated code?

   Solution:

   For instance, with

   ```
   g++ -O2 program.cc
   ```

   nearly all supported optimizations not involving a space-speed tradeoff are performed.

   Note that `O` stands for "Optimization".

5. I have an executable `a.out` and want to read data from a file `sample.inp` rather than from the keyboard, and write the output on a new file `sample.out` instead of to the screen. How can I do that?

   Solution:

   ```
   ./a.out < sample.inp > sample.out
   ```

6. How can I find the differences between two files `sample.out` and `sample.cor`?

   Solution:

   For example: `diff sample.out sample.cor`

7. In C++, how can I create a bidimensional matrix *matrix* of **int**s with *n* rows and *m* columns, all of them initialized to 1?

   Solution:

   The standard library of C++ does not have a built-in type “matrix”. A way to create the matrix is by creating a vector of *n* rows, each of which is a vector of *m* integers initialized to 1. By using the standard template class *vector<T>* and the constructor *vector<T>(size, init)* (which creates a vector of *size* copies of *init*):

   ```
   vector<int> row(m, 1);
   vector<vector<int≫ matrix(n, row);
   ```

   Equivalently (and better) in a single line:

   ```
   vector<vector<int≫ matrix(n, vector<int>(m, 1));
   ```

8. I have to sort a vector *v* of **int**s increasingly. Should I write my own sorting procedure?

   Solution:

   No (unless there is another reason for doing so). Use the *sort* procedure of the standard C++ library:

   ```
   #include <algorithm>
   //...
   sort(v.begin(), v.end());
   ```

9. And what if I have to sort *decreasingly*?

   Solution:

   The *sort* procedure admits a third parameter: the sorting criterion. It is a function or a function object that takes as parameters two objects of the container (in this case, two **int**s) and returns **true** when the first argument should come *before* the second one. For instance, in this case:

   ```
   bool before(int a, int b) { return a > b;}
   // ...
   sort(v.begin(), v.end(), before);
   ```

   Function objects of class *greater<***int***>*, available in the standard library, behave essentially the same as the aforementioned function *before*, and give an elegant solution:

   ```
   sort(v.begin(), v.end(), greater<int>());
   ```

   Another example of function *before*, now defined over *structs*:

   ```
   // first small surnames, in case of tie big names, in case of tie the younger one
   bool before(const Info& a, const Info& b) {
     if (a.surname ≠ b.surname) return a.surname < b.surname;
     if (a.name ≠ b.name) return a.name > b.name;
     return a.age < b.age;
   }
   ```

10. Let *s* be a *stack*<*pair*<**int**,**int**≫. Can the following code be written more compactly? (assuming that *aux* is not used any more)

    pair<**int**,**int**> *aux*;
    aux.first = 1;
    aux.second = 2;
    s.push(*aux*);

    Solution:

    One can make the compiler generate the adequate temporary object by calling a constructor of *pair*<**int**,**int**>. For example, any of the following would do:

    s.push(*pair*<**int**,**int**>(1, 2));
    s.push(*make_pair*(1, 2));
    s.push({1, 2}); // This is C++11

11. When I compile my program `program.cc` I get the output below. Where is the error?

```
user@machine:$ g++ program.cc
program.cc: In function ‘int main()’:
program.cc:10:8: error: no match for ‘operator<<’ (operand types are ‘std::ostream
{aka std::basic_ostream<char>}’ and ‘std::vector<int>’)
   cout << v << endl;
        ^
In file included from /usr/include/c++/5/iostream:39:0,
                 from program.cc:1:
/usr/include/c++/5/ostream:108:7: note: candidate: std::basic_ostream<_CharT,
_Traits>::__ostream_type& std::basic_ostream<_CharT, _Traits>::operator<<(std::basic_ostream<
_Traits>::__ostream_type& (*)(std::basic_ostream<_CharT, _Traits>::__ostream_type&))
[with _CharT = char; _Traits = std::char_traits<char>; std::basic_ostream<_CharT,
_Traits>::__ostream_type = std::basic_ostream<char>]
       operator<<(__ostream_type& (*__pf)(__ostream_type&))
       ^
/usr/include/c++/5/ostream:108:7: note:   no known conversion for argument
1 from ‘std::vector<int>’ to ‘std::basic_ostream<char>::__ostream_type&
(*)(std::basic_ostream<char>::__ostream_type&) {aka std::basic_ostream<char>&
(*)(std::basic_ostream<char>&)}’
/usr/include/c++/5/ostream:117:7: note: candidate: std::basic_ostream<_CharT,
_Traits>::__ostream_type& std::basic_ostream<_CharT, _Traits>::operator<<(std::basic_ostream<
_Traits>::__ios_type& (*)(std::basic_ostream<_CharT, _Traits>::__ios_type&))
[with _CharT = char; _Traits = std::char_traits<char>; std::basic_ostream<_CharT,
_Traits>::__ostream_type = std::basic_ostream<char>; std::basic_ostream<_CharT,
_Traits>::__ios_type = std::basic_ios<char>]
       operator<<(__ios_type& (*__pf)(__ios_type&))
       ^
/usr/include/c++/5/ostream:117:7: note:   no known conversion for argument
1 from ‘std::vector<int>’ to ‘std::basic_ostream<char>::__ios_type& (*)(std::basic_ostream<cha
{aka std::basic_ios<char>& (*)(std::basic_ios<char>&)}’
/usr/include/c++/5/ostream:127:7: note: candidate: std::basic_ostream<_CharT,
_Traits>::__ostream_type& std::basic_ostream<_CharT, _Traits>::operator<<(std::ios_base&
(*)(std::ios_base&)) [with _CharT = char; _Traits = std::char_traits<char>;
std::basic_ostream<_CharT, _Traits>::__ostream_type = std::basic_ostream<char>]
       operator<<(ios_base& (*__pf) (ios_base&))
       ^
```

```
/usr/include/c++/5/ostream:127:7: note:   no known conversion for argument
1 from ‘std::vector<int>’ to ‘std::ios_base& (*)(std::ios_base&)’
/usr/include/c++/5/ostream:166:7: note: candidate: std::basic_ostream<_CharT,
_Traits>::__ostream_type& std::basic_ostream<_CharT, _Traits>::operator<<(long
int) [with _CharT = char; _Traits = std::char_traits<char>; std::basic_ostream<_CharT,
_Traits>::__ostream_type = std::basic_ostream<char>]
       operator<<(long __n)
       ^
/usr/include/c++/5/ostream:166:7: note:   no known conversion for argument
1 from ‘std::vector<int>’ to ‘long int’
...
```

Solution:

Do not get overwhelmed by lengthy error reports. Focus on the (very) first lines. Here

```
program.cc:10:8: error: no match for ‘operator<<’ (operand types are ‘std::ostream
{aka std::basic_ostream<char>}’ and ‘std::vector<int>’)
   cout << v << endl;
        ^
```

is telling us that at line 10, column 8, the operator $\ll$ is misused.