Advanced Object-Oriented Programming, Spring 2017

Homework Assignment #5

Due midnight Wednesday, May 31, 2017

Instructions

1. If any question is unclear, please ask for a clarification.
2. Note that all the line numbers in the source listings are for references only.
3. You are required to do all the homework assignments on Linux and use g++ version 4.9.2 or later.
4. You are required to give your TA a demo of your program. Make sure that your program can compile and run on the server machine, which will be used for the demo.
5. For the program that you write, you are required to include a Makefile. Otherwise, the grade for your program will be zero.
6. Unless stated otherwise, you are required to work on the homework assignment individually.
7. No late homework will be accepted.

Programming Project

Part I (50%)

This assignment requires that you complete the implementation of the user-defined manipulators as given in Section 21.4.6.3 of the book entitled *The C++ Programming Language, Special Edition* by Bjarne Stroustrup.

The test program and the output of the test program are as given in Listing 1 and Listing 2. In order to make it easier for you to integrate your program with the test program, as given in Listing 1, you are required to name your program `Form.h`.

Listing 1: The test program.

```
1 #include <iostream>
2 #include "Form.h"
3
4 using std::cout;
5 using std::endl;
6
```

```


Form gen4(4);   // general format, precision is 4

void f(double d)
{
    Form sci8 = gen4;
    sci8.scientific().precision(8);     // scientific format, precision 8
    Form fix8 = gen4;
    fix8.fixed().precision(8);          // fixed format, precision 8
    cout << "default = " << d << endl
         << "gen4    = " << gen4(d) << endl
         << "sci8    = " << sci8(d) << endl
         << "fix8    = " << fix8(d) << endl
         << "default = " << d << endl;
}



int main()
{
    f(1234.56789);
    f(12.3456789);

    return 0;
}
```

Listing 2: The output of the test program.

```
default = 1234.57
gen4    = 1235
sci8    = 1.23456789e+03
fix8    = 1234.56789000
default = 1234.57
default = 12.3457
gen4    = 12.35
sci8    = 1.23456789e+01
fix8    = 12.34567890
default = 12.3457
```

Part II (50%)

This assignment requires that you complete the implementation of the class template hierarchy as given in Section 13.5 of the book entitled *The C++ Programming Language, Special Edition* by Bjarne Stroustrup.

The test program and the output of the test program are as given in Listing 3 and Listing 4. In order to make it easier for you to integrate your program with the test program, as given in Listing 3, you are required to name your program `Vector.h`.

Listing 3: The test program.

```
#include "Vector.h"
#include "Trace.h"

#include <iostream>

using std::cout;
using std::endl;
```

```



vector<float>   fvec(100);
vector<int>     ivec(200);
vector<char>    cvec(300);

vector<float*>  pfvec(100);
vector<int*>    pivec(200);
vector<char*>   pcvec(300);

int main()
{
    TRACE(dummy, "main()");

    fvec[3] = 10.10;
    cout << "  fvec[3] = " << fvec[3] << endl;

    ivec[3] = 10;
    cout << "  ivec[3] = " << ivec[3] << endl;

    cvec[3] = 'A';
    cout << "  cvec[3] = " << cvec[3] << endl;

    float f = 10.10;
    pfvec.elem(3) = &f;
    pfvec[3] = &f;
    cout << "  pfvec.elem(3) = " << *pfvec.elem(3) << endl;
    cout << "  pfvec[3]      = " << *pfvec[3] << endl;

    int a = 10;
    pivec.elem(3) = &a;
    pivec[3] = &a;
    cout << "  pivec.elem(3) = " << *pivec.elem(3) << endl;
    cout << "  pivec[3]      = " << *pivec[3] << endl;

    char c = 'A';
    pcvec.elem(3) = &c;
    pcvec[3] = &c;
    cout << "  pcvec.elem(3) = " << *pcvec.elem(3) << endl;
    cout << "  pcvec[3]      = " << *pcvec[3] << endl;

    return 0;
}
```

Listing 4: The output of the test program.

```
Entering vector<T>::vector(int) (0)
  count = 1
Leaving  vector<T>::vector(int) (0)
Entering vector<T>::vector(int) (0)
  count = 1
Leaving  vector<T>::vector(int) (0)
Entering vector<T>::vector(int) (0)
  count = 1
Leaving  vector<T>::vector(int) (0)
Entering vector<void*>::vector(int) (0)
  count = 1
Leaving  vector<void*>::vector(int) (0)
Entering vector<T*>::vector(int) (0)
Leaving  vector<T*>::vector(int) (0)
Entering vector<void*>::vector(int) (0)
  count = 2
Leaving  vector<void*>::vector(int) (0)
```

```
Entering vector<T*>::vector(int) (0)
Leaving  vector<T*>::vector(int) (0)
Entering vector<void*>::vector(int) (0)
  count = 3
Leaving  vector<void*>::vector(int) (0)
Entering vector<T*>::vector(int) (0)
Leaving  vector<T*>::vector(int) (0)
Entering main() (0)
  fvec[3] = 10.1
  ivec[3] = 10
  cvec[3] = A
  pfvec.elem(3) = 10.1
  pfvec[3]      = 10.1
  pivec.elem(3) = 10
  pivec[3]      = 10
  pcvec.elem(3) = A
  pcvec[3]      = A
Leaving  main() (0)
Entering vector<T*>::~vector (0)
Leaving  vector<T*>::~vector (0)
Entering vector<void*>::~vector (0)
  count = 3
Leaving  vector<void*>::~vector (0)
Entering vector<T*>::~vector (0)
Leaving  vector<T*>::~vector (0)
Entering vector<void*>::~vector (0)
  count = 2
Leaving  vector<void*>::~vector (0)
Entering vector<T*>::~vector (0)
Leaving  vector<T*>::~vector (0)
Entering vector<void*>::~vector (0)
  count = 1
Leaving  vector<void*>::~vector (0)
Entering vector<T>::~vector (0)
  count = 1
Leaving  vector<T>::~vector (0)
Entering vector<T>::~vector (0)
  count = 1
Leaving  vector<T>::~vector (0)
Entering vector<T>::~vector (0)
  count = 1
Leaving  vector<T>::~vector (0)
```

Also, to make it easier for you to test your implementation, also given are the trace program, which includes both `Trace.h` (Listing 5) and `Trace.cpp` (Listing 6).

Listing 5: Trace.h.

```
#ifndef __TRACE_H_INCLUDED__
#define __TRACE_H_INCLUDED__

#include <iostream>
#include <string>

using std::cerr;
using std::endl;
using std::string;

class Trace {
public:
    Trace(const string n)
        : name(n)
        {
            depth++;
            print_name("Entering");
        }

    ~Trace()
        {
            print_name("Leaving ");
            depth--;
        }
private:
    void print_name(const string prefix)
        {
            for (int i = 0; i < depth; i++)
                cerr << "  ";
            cerr << prefix << " " << name << " (" << depth << ")" << endl;
        }
private:
    static int depth;
private:
    const string name;
};

#ifdef __TRACE__
#define TRACE(v,n)      Trace v(n)
#else
#define TRACE(v,n)
#endif

#endif
```

Listing 6: Trace.cpp.

```
#include "Trace.h"

int Trace::depth = -1;
```

Grading Policy

For each part, the following policy applies.

- 10 points for compiling without errors and warnings.
- 30 points for the correctness of your program. Each error will cost you 10 point.
- 10 points for the program structure, readability, efficiency,