Driving Test 2009

Wednesday, 25 March 3pm – 5pm

Using the account details given below, attempt all 3 tasks. Write one C++ file per task. As your working directory you may choose `C:\temp\OOP_your_name`*. Make sure to write in the answers to each question on the* ANSWER SHEET*. Once you have finished, save the 3 files* `TaskN.cpp`, `N=1...3`, *to the directory* `C:\temp\SUBMIT_your_name` *and contact the invigilator.* **Do not log off!** *Hand in your answer sheet and present your submission directory. The invigilator will then reconnect the network cable so that you can email your solutions to* `rn@ic.ac.uk` *and to yourself. You are then free to leave.*

```
Username  :  Exam09
Password  :  *********
Domain    :  MA215-xx   (this computer)
```

1. *Complex numbers* [**40 marks**]
   Implement a class `complex` so that all of the statements below are executed correctly. Use only `private` member data and do not use `friend` functions.

```
double x = 4.1, y = -3.5;
complex z(1,-1), v(0,1), w(-2,3), res;
cout << z - w << endl;
res = x * z - v * y;
res = - res + z * w;
cout << res << endl;
z += v / w;
cout << z << endl;
cout << my_power(w, 3) << endl;
cout << my_power(v, -2) << endl;
```

   Here the signature of the global function `my_power` is

```
complex my_power(const complex &z, int n);
```

   and it computes the n-th power of the complex number z.

   [*A maximum of* 30 marks *can be obtained for this task when using* `public` *member data or* `friend` *functions.*]

2. *Laurent series* [**40 marks**]
   Laurent series are an important tool in complex analysis, especially to investigate the behaviour of functions near singularities. The formal definition of a Laurent series is $f(z) = \sum_{n=-\infty}^{\infty} a_n\,(z-c)^n$. As a first step towards Laurent series, implement a template `Laurent_polynomial`, which defines "polynomials" of the form $p(x) = \sum_{n=-N}^{N} a_n\,x^n$, so that the following code executes correctly.

```
Laurent_polynomial<double> p(2, 1.0);      // N = 2, a_n = 1.0, n = -N,...,N.
Laurent_polynomial<double> q(5, -1.5), s;
double x(2.1), y(-1.25);
cout << "p(y) = " << p.value(y) << endl;   // value of polynomial p at y
s = p + q;
for (int i = -s.getN(); i <= s.getN(); i++)
  cout << "s[" << i << "] = " << s[i] << endl; // print coefficients a_n of s
cout << "s(x) = " << s.value(x) << endl;
```

   [*Hint: You can either define a* `vector<T>` *as member data of the template* `Laurent_polynomial<T>`*, or you can derive the template from the class* `vector<T>`*. In the latter case, you will need to use the construction* `vector<T>::operator[](j)` *when overloading the access operator.*]

3. *Complex Laurent polynomials* [**20 marks**]
   Combine your class `complex` from 1. with your template from 2. in order to execute the following code.

```
complex f(1,1), g(0,-1);
Laurent_polynomial<complex> p(2, f);
Laurent_polynomial<complex> q(5, g), s;
complex x(1,2), y(0,-1);
cout << "p(y) = " << p.value(y) << endl;
s = p + q;
for (int i = -s.getN(); i <= s.getN(); i++)
  cout << "s[" << i << "] = " << s[i] << endl;
cout << "s(x) = " << s.value(x) << endl;
```

   [*Hint: If you have not already done so, you should transform the function* `my_power()` *from* 1. *into a template function, so that it can compute the n-th power for any type* `T` *that allows initialization with* `1` *and has multiplication, as well as devision, defined.*]

Name		CID	

Answer Sheet A

1. On replacing the second line of the code in the test with
   `complex z(2,-2), v(0,2), w(-4,6), res;`
   what are the outputs of your program?

Line	Output
`cout << z - w << endl;`	
`cout << res << endl;`	
`cout << z << endl;`	
`cout << my_power(w, 3) << endl;`	
`cout << my_power(v, -2) << endl;`	

2. On replacing the first line of the code in the test with
   `Laurent_polynomial<double> p(3, 2.0);`
   what is the output of your program?

Line	Output
`cout << "p(y) = " << p.value(x) << endl;`	
`cout << "s(x) = " << s.value(x) << endl;`	

3. On replacing the first line of the code in the test with
   `complex f(2,-2), g(0,5);`
   what is the output of your program?

Line	Output
`cout << "p(y) = " << p.value(x) << endl;`	
`cout << "s(x) = " << s.value(x) << endl;`	