Driving Test 2011

Wednesday, 23 March 2pm – 4pm

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 a directory, e.g. to* `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` *with a CC to yourself. You are then free to leave.*

```
Username  :  ******
Password  :  ******
```

For some of the tasks you need to `#include<cmath>` for the functions `sqrt()`, `exp()` and `log()`.

1. *Prime number sums* [**30 marks**]
   Let p_k denote the k^{th} prime number for $k \geq 1$. Define the finite sums
   $$S_n = \sum_{k=1}^{n} p_k = 2 + 3 + \dots\,, \qquad S_n^{\pm} = \sum_{k=1}^{n} (-1)^{k+1}\, p_k = 2 - 3 + \dots\,,$$
   $$T_n = \sum_{k=1}^{n} \frac{1}{p_k} = \frac{1}{2} + \frac{1}{3} + \dots\,, \qquad T_n^{\pm} = \sum_{k=1}^{n} \frac{(-1)^{k+1}}{p_k} = \frac{1}{2} - \frac{1}{3} + \dots\,.$$
   Write a program that can compute S_n, $S_n^{\pm}$, T_n and $T_n^{\pm}$ for arbitrary $n \geq 1$.

2. *Classical Gram–Schmidt* [**30 marks**]
   Let $\langle \cdot, \cdot \rangle$ denote the standard inner product on $\mathbb{R}^m$; i.e. $\langle \underline{a}, \underline{b} \rangle = \underline{a}^T\, \underline{b} = \sum_{i=1}^{m} a_i\, b_i$ for all $\underline{a}$, $\underline{b} \in \mathbb{R}^m$. Let $\|\underline{a}\| = (\langle \underline{a}, \underline{a} \rangle)^{\frac{1}{2}}$. Given n linearly independent vectors $\underline{a}_1, \underline{a}_2, \dots, \underline{a}_n \in \mathbb{R}^m$; $m \geq n$; the classical Gram–Schmidt algorithm is defined as follows:

   ```
   $\underline{v}_1 = \underline{a}_1$
   $\underline{q}_1 = \underline{v}_1 / \|\underline{v}_1\|$
   for $i = 2$ to $n$
       $\underline{v}_i = \underline{a}_i - \sum_{j=1}^{i-1} \langle \underline{a}_i, \underline{q}_j \rangle\, \underline{q}_j$
       $\underline{q}_i = \underline{v}_i / \|\underline{v}_i\|$
   end
   ```

   Hence the algorithm computes the vectors $\underline{q}_1, \underline{q}_2, \dots, \underline{q}_n \in \mathbb{R}^m$. It is easy to show that for linearly independent $\{\underline{a}_i\}_{i=1}^n$ the algorithm is well-defined and that the vectors $\{\underline{q}_i\}_{i=1}^n$ are orthonormal and span the same subspace as $\{\underline{a}_i\}_{i=1}^n$. Defining the matrices $A = [\underline{a}_1, \underline{a}_2, \dots, \underline{a}_n] \in \mathbb{R}^{m \times n}$ and $Q = [\underline{q}_1, \underline{q}_2, \dots, \underline{q}_n] \in \mathbb{R}^{m \times n}$ in terms of their columns, the classical Gram–Schmidt algorithm can also be interpreted as transforming A into Q.

   Write a program that performs the classical Gram–Schmidt algorithm on the given set of vectors $\{\underline{a}_i\}_{i=1}^n$ and then returns $\{\underline{q}_i\}_{i=1}^n$.

   [*Note: There is no restriction on how you implement the algorithm, as long as it works for arbitrary dimensions $m \geq n$. For instance, you might prefer to have your program work on the rows of the matrix $A^T \in \mathbb{R}^{n \times m}$ and then return Q^T. Moreover, you may assume that the given vectors $\{\underline{a}_i\}_{i=1}^n$ are indeed linearly independent. You do not need to perform any well-posedness or consistency checks in your program.*]

3. *Step functions* [**40 marks**]
For $a, b \in \mathbb{R}$ with $a < b$ the characteristic function $\chi_{[a,b)} : \mathbb{R} \to \mathbb{K}$ is defined by

$$\chi_{[a,b)}(x) = \begin{cases} 1 & a \leq x < b, \\ 0 & \text{else.} \end{cases}$$

Here $\mathbb{K} = \mathbb{R}$ or $\mathbb{C}$. For $\alpha \in \mathbb{K}$ we call $\alpha\, \chi_{[a,b)}$ a generalized characteristic function. A finite sum of generalized characteristic functions is called a step function, i.e.

$$s = \sum_{i=1}^{n} \alpha_i\, \chi_{[a_i,b_i)}\,, \tag{1}$$

where $\alpha_i \in \mathbb{K}$ and $a_i, b_i \in \mathbb{R}$ with $a_i < b_i$, $i = 1 \to n$.

Clearly products of generalized characteristic functions are again generalized characteristic functions and so sums and products of step functions are again step functions. For a step function s of the form (1) we can define the integral

$$\int_{\mathbb{R}} s(x)\, \mathrm{d}x = \sum_{i=1}^{n} \alpha_i\,(b_i - a_i) \in \mathbb{K}\,.$$

Clearly, the representation (1) for a step function s is not unique. However, in terms of this assignment this will be of no consequence.

Finally, given a partition $a = x_0 < x_1 < \cdots < x_n = b$ of an interval $[a, b] \subset \mathbb{R}$, and a function $f : [a, b] \to \mathbb{K}$, we can define the corresponding step function approximation to f by

$$s_n^f = \sum_{i=1}^{n} f(x_{i-1})\, \chi_{[x_{i-1},x_i)} \tag{2}$$

and then use the Riemann sum $\int_{\mathbb{R}} s_n^f(x)\, \mathrm{d}x = \int_a^b s_n^f(x)\, \mathrm{d}x$ as an approximation to $\int_a^b f(x)\, \mathrm{d}x$. Common partitions are uniform partitions $x_i = a + i\,\frac{b-a}{n}$, $i = 0 \to n$; and logarithmic partitions $x_i = \log(e^a + i\,\frac{e^b - e^a}{n})$, $i = 0 \to n$.

(a) Write a template class `gen_charfn` that implements a generalized characteristic function $\alpha\,\chi_{[a,b)}$. Here a minimalist implementation would only hold α, a, b. However, it might be useful for later tasks to also implement member functions for function evaluation, integration and multiplication with another generalized characteristic function.

(b) Write a template class `step_function` that implements a step function s. As the underlying data structure you may want to choose `vector<gen_charfn<T> >`. Overload the function call operator to return $s(x)$ for $x \in \mathbb{R}$ and implement a member function `intR()` which returns the intregral $\int_{\mathbb{R}} s(x)\, \mathrm{d}x$. Overload the addition operator to compute the sum between two step functions.

(c) Write a global template function `interpolate` that, given a function f and a partitioning of an interval $[a, b]$, computes the interpolating step function s_n^f from (2). In addition, provide the functions `uniform_partition` and `logarithmic_partition` that for $n \geq 1$ compute the corresponding partitions of a given interval $[a, b]$.

(d) Overload the multiplication operator `step_function<T>::operator*()` three ways: for the multiplication with a scalar $\beta \in \mathbb{K}$, with a generalized characteristic function and with another step function.

[Hint: The member function `vector<T>::clear()` may or may not prove useful.]

Name		CID	

Answer Sheet **A**

1. To five decimal places, state the values of the following sums.

Sum	Value
S_{50}	
$S^{\pm}_{100}$	
T_{200}	
$T^{\pm}_{250}$	

Note: For $N = 10$ *the numbers are:* $S_N = 129$, $S^{\pm}_N = -13$, $T_N = 1.53344$, $T^{\pm}_N = 0.25298$.

2. To three decimal places, state the matrix Q that your program computes for the given matrices A.

A	Q
$\begin{pmatrix} 2 & 14 \\ 3 & 0 \\ 13 & 23 \end{pmatrix}$	
$\begin{pmatrix} 1 & 0 & -1 & 2 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 3 & -3 \\ 3 & 2 & -1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	

Note: For $A = \begin{pmatrix} 1 & 2 \\ 2 & 2 \\ 0 & 0 \end{pmatrix}$ *the result is* $Q \approx \begin{pmatrix} 0.447 & 0.894 \\ 0.894 & -0.447 \\ 0 & 0 \end{pmatrix}$, *while for* $A = \left(\begin{smallmatrix} 2 & 2 \\ 0 & 2 \end{smallmatrix}\right)$ *the result is* $Q = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$.

3. Write a program that executes the following statements correctly.

```
double f(double x);
double f2(double x) { return f(x)*f(x); }

int main() {
  int n = 100;
  double a = -0.5; double b = 0.5;
  vector<double> partuni, partlog;
  step_function<double> s, t, u, v, w;
  uniform_partition(a, b, n, partuni);
  interpolate(f, partuni, s);
  cout << "Integral with uniform = " << s.intR() << endl;
  logarithmic_partition(a, b, n, partlog);
  interpolate(f, partlog, t);
  cout << "Integral with logarithmic = " << t.intR() << endl;
  u = s + t;
  cout << "u(0.1) = " << u(0.1) << endl;
  cout << "Integral sum = " << u.intR() << endl;
  v = s * t;
  cout << "Integral of s * t = " << v.intR() << endl;
  interpolate(f2, partlog, w);
  cout << "Integral of f^2 with logarithmic = " << w.intR() << endl;
  return 0;
}
```

where your implementation of the function `f()` returns the value of $f(x) = \exp(x^2 + \frac{1}{1+x})$ for $x \in \mathbb{R}$. Below fill in the outputs of your program.

Line	Output
`cout << "..." << s.intR() << endl;`	
`cout << "..." << t.intR() << endl;`	
`cout << "u(0.1) = " << u(0.1) << endl;`	
`cout << "..." << u.intR() << endl;`	
`cout << "..." << v.intR() << endl;`	
`cout << "..." << w.intR() << endl;`	

Finally, on including `#include<complex>`, replace `step_function<double>` with `step_function<complex<double> >` and let $f(x) = \exp(i\,x^2 + \frac{1}{1+x}) \in \mathbb{C}$ for $x \in \mathbb{R}$. The imaginary unit i can be defined as `complex<double> i(0.0, 1.0);`, while the complex exponential function is also called `exp()`. What are the outputs now?

Line	Output
`cout << "..." << s.intR() << endl;`	
`cout << "..." << t.intR() << endl;`	
`cout << "u(0.1) = " << u(0.1) << endl;`	
`cout << "..." << u.intR() << endl;`	
`cout << "..." << v.intR() << endl;`	
`cout << "..." << w.intR() << endl;`	