Programming techniques for physical simulations
Exercise 10

November 18, 2009

1. Implement a new version of your Simpson integration routine using virtual functions. Define an abstract base class with a `pure virtual` operator() (function object), and a fully implemented simpson integrator, both for double precision. Second, inherit a concrete class from this abstract base class and implement the operator().
2. Repeat the benchmark of Exercise 7 for the functions i) $f(x) = 0$ ii) $f(x) = x^2$ and iii) $f(x) = sin(x)$.
3. You can use virtual functions to implement a factory design pattern. Look at our example on sorting algorithms and complete it.
4. As an optional exercise you can apply the factory design pattern to your several Simpson integration routines.
5. Please hand in a report on the Penna model exercises by next week.

```
#include <algorithm>
#include <iostream>
#include <iterator>
#include <vector>
#include <list>

using namespace std;

template<class Iterator>
class sorter
{
public:
	virtual void sort(Iterator begin, Iterator end) const = 0;
};

template<class Iterator>
class BubbleSort : public sorter<Iterator>
{
public:
	void sort(Iterator begin, Iterator end) const
	{
		// implement this yourself!
	}
};

template<class Iterator>
class StdSort : public sorter<Iterator>
{
public:
	void sort(Iterator begin, Iterator end) const
	{
		std::sort(begin, end);
	}
};

template<class Iterator>
class InsertionSort : public sorter<Iterator>
{
	typedef typename Iterator::value_type value_t;
	typedef list<value_t> storage_t;
public:
	void sort(Iterator begin, Iterator end) const
	{
		// implement me!
		// one hint: there is a function std::lower_bound() in the Algorithms library
		// that is very useful here
	}
};

template<class Iterator>
sorter<Iterator> * sortingFactory(string algorithm)
{
	// implement me!
}

typedef double T;
typedef vector<T> storage_t;
typedef storage_t::iterator iterator_t;

int main(int argc, char ** argv)
{
	if (argc != 2) {
		cerr << argv[0] << " <sorting algorithm>" << endl;
		exit(1);
	}

	sorter<iterator_t> *mysort = sortingFactory<iterator_t>(string(argv[1]));

	storage_t foo;
	copy(istream_iterator<T>(cin),
		istream_iterator<T>(),
		back_insert_iterator<storage_t>(foo));

	mysort->sort(foo.begin(), foo.end());

	copy(foo.begin(), foo.end(), ostream_iterator<T>(cout, "\n"));
	cout << endl;

	delete mysort;
}
```