H62SED; Software Eng. Design P Sewell Oct 2012

2012 exam paper solutions

General Comment

Please note that this is a completely open book exam and moreover the students have complete electronic access to all the example programs used to present the course. Therefore, please take into consideration that successful completion of this exam does not involve large quantities of typing, rather judicious "cut, paste and modification". Of course the test is knowing what to cut and paste and making the design decisions!

The tasks are graded from that required for a basic pass (task 1) through to task 5 which are there to provide scope for 1st class students to do their thing! It is not intended that equal marks are awarded for equal "volumes" of code.

In the marking scheme x+y% for functions means x% for a correct interface and y% for a correct implementation. Just x% requires both correct interface and implementation.

Clearly, there are different equally valid ways of implementing many of the functions and I will use my judgement to deal with any variations within the spirit of the scheme below.

Task1 (40%)

Sensible choice of variable names!!!	5%
Correct use of private	5%
Constructors	2+3%
Copy constructor	2+3%
Destructor	2+3%
Operator=	5+5%
gets/sets	5%

Task 2 (15%) Issues: correct attempt as run time error checks, no sets, inlining

Task 3 (15%) Issues: pointer to supplier, friends, streams,

Task 4 (15%) Issues: correct receiver fn in power_company, run time error checking, not leaving a "whole" in the lists

Task 5 (15%) Issues: public inheritance, virtual~, correctly putting task 4 in customer

```
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <fstream>
//---------------------------------------------------------------------------

class power_company;

class power_station;

class customer;

class customer_factory;

using namespace std;

//---------------------------------------------------------------------------
class power_station {

public:

        power_station(float _cost_efficiency,float _max_gen_capability,bool _status=false)

                :cost_efficiency(_cost_efficiency),
                 max_gen_capability(_max_gen_capability),
                 status(_status){}

        power_station(const power_station& p)

                :cost_efficiency(p.cost_efficiency),
                 max_gen_capability(p.max_gen_capability),
                 status(p.status){}

        ~power_station(){}

        const power_station& operator=(const power_station& p) {

                if(this==&p) return(*this);

                cost_efficiency=p.cost_efficiency;

                max_gen_capability=p.max_gen_capability;

                status=p.status;

                return(*this);
        }

        float get_cost_efficiency(){return(cost_efficiency);}

        void set_cost_efficiency(const float _cost_efficiency){cost_efficiency=_cost_efficiency;}

        float get_max_gen_capability(){return(max_gen_capability);}

        void set_max_gen_capability(const float
_max_gen_capability){max_gen_capability=_max_gen_capability;}

        float get_status(){return(status);}

        void set_status(float _status){status=_status;}
```

```
private:

	float cost_efficiency;		// Pence/kWh

	float max_gen_capability; // kW

	bool status;			// true on, false off
};
//---------------------------------------------------------------------------
class power_company {

public:

	power_company(int _no_power_stations,
			int _no_customers,
			power_station** _power_stations,
			customer** _customers)

			:no_power_stations(_no_power_stations),
			 no_customers(_no_customers){

		if(no_power_stations>10||no_power_stations<0) {

				cout<<"\nThrow an error in power_company constructor - illegal
no_power_stations";exit(0);
		}

		if(no_customers>100||no_customers<0) {

				cout<<"\nThrow an error in power_company constructor - illegal
no_customers";exit(0);
		}

		for(int i=0;i<no_power_stations;++i) power_stations[i]=_power_stations[i];

		for(int i=0;i<no_customers;++i) customers[i]=_customers[i];
	}

	power_company(const power_company& p){

		*this=p;
	}

	~power_company(){}

	const power_company& operator=(const power_company& p) {

		if(this==&p) return(*this);

		no_power_stations=p.no_power_stations;

		no_customers=p.no_customers;

		for(int i=0;i<no_power_stations;++i) power_stations[i]=p.power_stations[i];

		for(int i=0;i<no_customers;++i) customers[i]=p.customers[i];

		return(*this);
	}
```

```
        int get_no_power_stations(){return(no_power_stations);}

        int get_no_customers(){return(no_customers);}

        // No sets!

        void add_power_station(power_station* const p){

                if(no_power_stations==10) {

                        cout<<"\nThrow an error in power_company constructor - full";exit(0);
                }

                power_stations[no_power_stations++]=p;
        }

        void add_customer(customer* const p){

                if(no_customers==100) {

                        cout<<"\nThrow an error in power_company constructor - full";exit(0);
                }

                customers[no_customers++]=p;
        }

        void remove_customer(customer* const p){

                for(int i=0;i<no_customers;++i){

                        if(p==customers[i]){

                                for(int j=i+1;j<no_customers;++j){

                                        customers[j-1]=customers[j];
                                }

                                --no_customers;
                        }
                }
        }

private:

        int no_power_stations;

        int no_customers;

        power_station* power_stations[10];

        customer* customers[100];
};
//---------------------------------------------------------------------------
class customer {

public:

        customer(power_company* _power_supplier):power_supplier(_power_supplier){
```

```
		power_supplier->add_customer(this);
	}

	customer(const customer& c):power_supplier(c.power_supplier){

		power_supplier->add_customer(this);
	}

	virtual ~customer(){

		power_supplier->remove_customer(this);
	}

	virtual const customer& operator=(const customer& c) {

		power_supplier=c.power_supplier;
	}

private:

	power_company* power_supplier;

};
//---------------------------------------------------------------------------
class customer_factory:public customer {

	friend ostream& operator<<(ostream&,const customer_factory&);

	friend istream& operator>>(istream&,customer_factory&);
public:

	customer_factory(power_company* _power_supplier,float _maximum_power_requirement)


	:customer(_power_supplier),maximum_power_requirement(_maximum_power_requirement)
{}

	customer_factory(const customer_factory& c)

		:customer(c),maximum_power_requirement(c.maximum_power_requirement){}

	virtual ~customer_factory(){}

	virtual const customer& operator=(const customer_factory& c) {

		maximum_power_requirement=c.maximum_power_requirement;

		customer::operator=(c);
	}

private:

// interesting argument whether this is held by reference or pointer - not value

	power_company* power_supplier;

	float maximum_power_requirement;
};
//---------------------------------------------------------------------------
ostream& operator<<(ostream& out,const customer_factory& cf){
```

```
	out<<cf.maximum_power_requirement; // awkward to do the power supplier!

	return(out);
}
istream& operator>>(istream& in,customer_factory& cf){

	in>>cf.maximum_power_requirement;

	return(in);
}
//-------------------------------------------------------------------------------
```