H62SED; Software Eng. Design P Sewell Nov2014

2014 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.

Please note well: I am deliberately testing the student's ability to make engineering decisions as well as their knowledge of *standard good practice* in the subject.

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.

All marks will take account of *suitable* use of in file commentary/documentation

Obviously, the style of the suggested solution is consistent with the level of the module and not necessarily *ideal!*

Task1 (40%) - just for module_record with no inheritance

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

Task 2(25%)

Correct (suitable!) embedding of module_records within student records	5%

Must include *definitions* of all basic functionality as task 1	5%.
Implementation of functions with particular care on array management - memory leaks,	10%
Suitable choice of default behaviour eg. get_module_mark if no modules etc	5%

Task 3 (10%)

For both classes - preferably one using that of the other

ostream<<	2+3%
ostream>>	2+3%

Task 4 (5%)	5+5%

Task 5 (20%)

public inheritance	3%
correct splitting of member variables	5%
constructors	5%
virtual ~	2%
virtual is_passed	5%

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

using namespace std;

//---------------------------------------------------------------------------
class module_record {

public:

        module_record(int _module_ID=-1,int _exam_mark=0)

                :module_ID(_module_ID),exam_mark(_exam_mark){}

        module_record(const module_record& mr)

                :module_ID(mr.module_ID),exam_mark(mr.exam_mark){}

        module_record(istream& in){

                in>>*this;
        }

        module_record& operator=(const module_record& mr){

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

                module_ID=mr.module_ID;

                exam_mark=mr.exam_mark;

                return(*this);
        }

        virtual ~module_record(){}

        int get_module_ID() const {return(module_ID);}

        void set_module_ID(const int _module_ID){

                module_ID=_module_ID;
        }

        int get_exam_mark() const {return(exam_mark);}

        void set_exam_mark(const int _exam_mark){

                exam_mark=_exam_mark;
        }

        virtual bool is_passed(){return(exam_mark>=40);}

        friend ostream& operator<<(ostream& out,const module_record&);

        friend istream& operator>>(istream& in,module_record&);

private:
```

```
        int module_ID;
        int exam_mark;
};
//---------------------------------------------------------------------------
class student_record {
public:
        student_record(int _student_ID_number,int _no_modules,module_record** _modules)
                        :student_ID_number(_student_ID_number),no_modules(_no_modules){
                modules=new module_record*[no_modules];
                for(int i=0;i<no_modules;++i) {
                        modules[i]=new module_record(*_modules[i]);
                }
        }
        student_record(const student_record& sr)
                        :student_ID_number(sr.student_ID_number),no_modules(sr.no_modules){
                modules=new module_record*[no_modules];
                for(int i=0;i<no_modules;++i) {
                        modules[i]=new module_record(*(sr.modules[i]));
                }
        }
        student_record(istream& in){
                in>>*this;
        }
        student_record& operator=(const student_record& sr){
                if(this==&sr) return(*this);
                student_ID_number=sr.student_ID_number;
                delete[] modules;
                no_modules=sr.no_modules;
                modules=new module_record*[no_modules];
                for(int i=0;i<no_modules;++i) {
                        modules[i]=new module_record(*(sr.modules[i]));
                }
                return(*this);
        }
        virtual ~student_record(){
```

```
	delete[] modules;
}

int get_student_ID_number() const {return(student_ID_number);}

void set_student_ID_number(const int _student_ID_number){

	student_ID_number=_student_ID_number;
}

int get_no_modules() const {return(no_modules);}

int get_module_mark(int module_ID) const {

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

		if(modules[i]->get_module_ID()==module_ID) {

			return(modules[i]->get_exam_mark());
		}
	}

	return(-1); // to denote didn't take the module
}

void set_module_mark(const module_record& mr) {

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

		if(modules[i]->get_module_ID()==mr.get_module_ID()) {

			return(modules[i]->set_exam_mark(mr.get_exam_mark()));
		}
	}

	module_record** _modules=new module_record*[no_modules+1];

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

		_modules[i]=modules[i];
	}

	_modules[no_modules]=new module_record(mr);

	++no_modules;

	delete[] modules;

	modules=_modules;
}

bool get_number_of_failed_modules() const {

	int no_failed(0);

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

		if(!modules[i]->is_passed()) ++no_failed;
	}
```

```
		return(no_failed);
	}

	bool passed_all_modules() const {

		return(get_number_of_failed_modules()==0);
	}

	friend ostream& operator<<(ostream& out,const student_record&);

	friend istream& operator>>(istream& in,student_record&);

private:

	int student_ID_number;

	int no_modules;

	module_record** modules; // Better to use pointers if going to use inheritance
};
//---------------------------------------------------------------------------
ostream& operator<<(ostream& out,const module_record& mr){

	out<<mr.module_ID<<" "<<mr.exam_mark;

	return(out);
}
//---------------------------------------------------------------------------
istream& operator>>(istream& in,module_record& mr){

	in>>mr.module_ID>>mr.exam_mark;

	return(in);
}
//---------------------------------------------------------------------------
ostream& operator<<(ostream& out,const student_record& sr){

	out<<sr.student_ID_number;

	out<<"\n"<<sr.no_modules;

	for(int i=0;i<sr.no_modules;++i) out<<"\n"<<*(sr.modules[i]);

	return(out);
}

istream& operator>>(istream& in,student_record& sr){

	in>>sr.student_ID_number;

	in>>sr.no_modules;

	delete[] sr.modules;

	sr.modules=new module_record*[sr.no_modules];

	for(int i=0;i<sr.no_modules;++i) sr.modules[i]=new module_record(in);

	return(in);
}
```

```
//------------------------------------------------------------------------
class module_with_project_record:public module_record {

public:

	module_with_project_record(int _module_ID=-1,int _exam_mark=0,int _project_mark=0)

		:module_record(_module_ID,_exam_mark),project_mark(_project_mark){}

	module_with_project_record(const module_with_project_record& mr)

		:module_record(mr.get_module_ID(),mr.get_exam_mark()),project_mark(mr.project_mark){}

	module_with_project_record& operator=(const module_with_project_record& mr){

	// Note not doing virtual assignment!

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

		set_module_ID(mr.get_module_ID());

		set_exam_mark(mr.get_exam_mark());

		project_mark=mr.project_mark;

		return(*this);
	}

	virtual ~module_with_project_record(){}

	int get_project_mark() const {return(project_mark);}

	void set_project_mark(const int _project_mark){

		project_mark=_project_mark;
	}

	virtual bool is_passed(){

		return(module_record::is_passed()&&project_mark>=40);
	}

private:

	int project_mark;

};
//------------------------------------------------------------------------
int main(){}
//------------------------------------------------------------------------
```