

Assignment 3, Deadline Thurs. 3rd Dec. 2009, 12.30pm (2213). Demonstrations must be completed by Tues. 1st Dec. 5.00pm. Worth 25% of CA.

Late assignments without documented extenuating circumstances **will lose 10% of marks per day or part of day late. Those seven days late will receive zero marks.**

Computing Department Rules. Assignments must typed or neatly handwritten and neatly presented and spelling-checked. Sloppily presented work will be severely penalised or returned for correction and resubmission. Please use the Computing Department cover sheet and signed declaration. **Please number pages.**

Poor layout of programs will be penalised. *Please include comments, but not useless comments.* You may be questioned about the content of assignments.

Software will be available in my public folder (2009a3).

1. Compile, link and execute `AddressT1.cpp`.

[10 marks]

Demonstration 100% of marks.

2. Compile, link and execute `PersonT1.cpp`.

[10 marks]

Demonstration 100% of marks.

3. Compile, link and execute `StudentT1.cpp`.

[10 marks]

Demonstration 100% of marks.

4. Virtual functions.

- (a) Remove the `virtual` qualifier from `toString()` in `Person.h` and `Student.h`. Recompile, link and execute `StudentT1.cpp`.

[10 marks]

Demonstration 100% of marks.

- (b) There should be a discrepancy between the `Person` pointer (contents) list in (a) and the one in 3. above. Include a printout of the two lists and explain the difference.

[10 marks]

5. Your own test program for Student.

- (a) Write your own test program StudentT2.cpp for Student. You should avoid my automated test data generation and use instructions like those below

```
string s =  
    string("James,Mulligan;19, First Road,Kilmacrenan,Donegal,Ireland;L200799");  
Student st(s);  
sList.push_back(Student st(s));
```

In addition to using the constructor

```
Student(std::string str =  
    "blankf,blankl;blanknum,blanks,blankc,blankr,blankc;blankid");
```

you should also show examples of use of the constructor

```
Student(std::string& firstName, std::string& lastName,  
    Address& address, std::string& id);
```

There should be at least eight (8) students in your list. Demonstration.

[20 marks]

- (b) Commented printout of StudentT2.cpp in hand-in report.

[20 marks]

- (c) Commented printout of lists before and after sorting in hand-in report.

[20 marks]

6. (a) Copy LecturerT1.cpp to LecturerT2.cpp and use vector rather than list. Compile and execute. Demonstration.

[20 marks]

- (b) Add code to LecturerT2.cpp to sort the vector and print out the sorted list. Demonstration.

[20 marks]

- (c) Add code to LecturerT2.cpp to sort the vector according to *office number*; print out the sorted list. See StudentT1 for a closely related example. Demonstration.

[20 marks]

- (d) Commented printout of relevant parts of LecturerT2.cpp in the hand-in report.

[20 marks]

- (e) Commented printout of the unsorted and sorted tables in hand-in report.

[20 marks]

7. Now you are going to write an *address book* program.

- (a) First we need a class `Contact`; `Contact` inherits from `Person`, just like `Student` and `Lecturer`. `Contact` should have instance variables for email address and for telephone number. It should have `==` and `<` operators; the `<` operator should take account of email address if the other fields are equal. Include properly commented `Contact.h` and `Contact.cpp` in your hand-in report.

[50 marks]

- (b) Write a test program `AddressBook.cpp` that tests `Contact` just like `StudentT1.cpp` or `StudentT2.cpp`. The test program must demonstrate:

- (i) Basic test program;
- (ii) Sorting according to the `==` and `<` operators;
- (iii) Sorting according to a *compare* function that compares `Contact` entries based on their telephone number; see `cmp0` in `StudentT1.cpp` and its use there.
- (iv) Writing to file;
- (v) Reading from file;
- (vi) Reading from a file when the program starts; use command line arguments.
I.e. from CMD, `> AddressBook addlist.txt` reads `addlist.txt` into a vector of `Contacts`.

Demonstration

[6 × 10 = 60 marks]

- (c) Include a properly commented `AddressBook.cpp` in your hand-in report.

[50 marks]

- (d) Include a UML diagram showing the relationship between `AddressBook`, `Contact`, `Person`, and `Address` your hand-in report.

[20 marks]