Contents

1 Introduction

This coursework (which counts for 30% of the module mark) is so that you familiarise yourselves with classes, class inheritance, containers, generic functions, and some more advanced ways to use the Standard Template Library (STL).

The application is a simplified phone book.

Its main purpose is to prepare you for the exam (which counts for 70% of the module mark) – it will be difficult to pass the exam if you have not worked on the coursework, so please take it seriously.

The coursework is meant to be taken in **groups of two** (plenty of work for all). If you cannot find others to form a group you will be auto-allocated to groups randomly, so please act quickly. When submitting you will also submit your personal evaluation of the amount of work produced by each group member.

Notes:

1. While in some groups some may work more than others, all should understand what is being produced. Explaining to others your code will help them improve; it will also help you improve, as having to explain it will make it clearer in your mind (as I've discovered when I started teaching…)

2. Deadline is on a Sunday afternoon – better submit something by Friday in case there is a Moodle meltdown over the weekend. "Submit early, submit often."

2 Coursework Description

2.1 Inheritance

Q-1. (1 mark) Implement a class `person`, with members for representing a person's name and surname.

It requires a constructor that initializes the members to some reasonable values and member functions to get the values of the members: `name()`, `surname()`.

Add two Boolean member functions that return `true` if a `person` object has a telephone number or an email respectively: `bool has_telephone_p()`, `bool has_email_p()`.

Q-2. (1 mark) Implement a class `person_with_telephone` that inherits from class `person`, with a member for holding the person's telephone.

Provide an appropriate constructor and member functions to both get (`telephone`) and modify (`set_telephone`) the person's telephone.

Redefine any member functions of the super-class that you may need to.

Q-3. (1 mark) Implement a class `person_with_email` that inherits from class `person`, with a member for holding the person's email.

Provide an appropriate constructor and member functions to both get (`email`) and modify (`set_email`) the person's email.

Redefine any member functions of the super-class that you may need to.

Q-4. (2 marks) Implement a class `person_with_telephone_and_email` that inherits from classes `person_with_telephone` and `person_with_email`.

Think: How many sub-objects of class `person` should a `person_with_telephone_and_email` object have? *(This is treated in Session 7 – Multiple Inheritance)*

Provide an appropriate constructor.

Redefine any member functions of the super-class that you may need to.

Q-5. (3 marks) Appropriately overload `operator<<` so that you can print any type of `person`.

The format to be used is:
`<person S` *`surname`* `N` *`name`* `[ T` *`telephone`*`]? [ E` *`email`*`]? >`

The notation `[a]?` means that `a` may be present or absent.

Examples:
```
<person S Smith N Tom >
<person S Smith N Dick T +49.921.1434 >
<person S Smith N Harry E hsmith@gmail.com >
<person S Smith N Mary T +39.921.1434 E msmith@gmail.com >
<person S John N John T +33.921.1434 E jjohn@gmail.com >
```

Q-6. (5 marks) Implement an external function `read_person` that allows you to read any type of person from an input stream and return it. Its signature should be:
```
istream & read_person(istream &in, person * & p)
```

Note: You *cannot* simply overload the `operator>>` as you have no idea what kind of person object the input represents until you have actually read the input till the closing `'>'` character.

It's supposed to be used as in the following:
```
person *pp = 0;
while (read_person(cin, pp) && pp)
  cout << *pp << endl;
```

The above use scenario explains why the second argument is a *reference* to a `person` pointer. Otherwise, you cannot modify the original pointer `pp`.

Total value up to here: 13 points.

2.2 Containers

Q-7. (10 marks) Implement a class `contacts` that holds all your contacts.

You can assume that two different persons will always have either a different name or a different surname. So redefine the `operator==` for all kinds of persons, so that it checks only whether the name and surname match. Also overload the `operator<` so that it checks first the surnames and then the names of two persons.

You can add a person in your contacts using the member function `add`. This should work even if that person is already in the contacts (useful when you want to update their contact details). So the following should update John's details – note how the person is being passed to `add`:

```
contacts c;
person john1("John", "John");
c.add(& john1);
person_with_telephone john2("John", "John", "+33.02.0101.0202");
c.add(& john2);
```

This example shows why you cannot store person objects into the container – you cannot replace a `person` object with a `person_with_XXX` one (try it to see what you get).

Note: So far we've used the `vector`, `list`, and `map` containers. We've also mentioned the `set` and `deque` ones – see their interfaces here: `www.cplusplus.com/reference/stl/`, `http://www.cplusplus.com/reference/set/set/`, `http://www.cplusplus.com/reference/deque/deque/`

Total value up to here: 23 points.

2.3 Generic Algorithms

Study the appendix A to see how one can use generic algorithms to write complex code quickly – try to use `for_each` in the following questions.

Q-8. (2 marks) Implement a member function `find` that matches either a person's name or surname.

Given that `find` searches for persons with either a matching name or surname, it should take care to return a person only once, even if both her/his name and surname both match the search string, *e.g.*, person "John John" should be returned only once when searching for "John".

Q-9. (3 marks) Implement a member function in `contacts`, called `in_order`, that prints your contacts.

It should print these contacts in alphabetic order (A–Z), according to surname and then name.

It should print these contacts to some output stream that it takes as a parameter. So for the five names aforementioned it should print:

```
<person S John N John T +33.921.1434 E jjohn@gmail.com >
<person S Smith N Dick T +49.921.1434 >
<person S Smith N Harry E hsmith@gmail.com >
<person S Smith N Mary T +39.921.1434 E msmith@gmail.com >
<person S Smith N Tom >
```

Q-10. (2 marks) Implement a member function in `contacts`, called `with_telephone`, that prints your contacts that have a telephone.

It should print these contacts in alphabetic order (A–Z), according to surname and then name.

It should print these contacts to some output stream that it takes as a parameter. So it should print:

```
<person S John N John T +33.921.1434 E jjohn@gmail.com >
```

```
<person S Smith N Dick T +49.921.1434 >
<person S Smith N Mary T +39.921.1434 E msmith@gmail.com >
```

Total value: 30 points.

3 Design, Etc.

There are design issues to be considered. Try to think of possible usage scenarios for your classes/functions, how you can make them faster, how you can make them shorter/simpler (code re-factoring), how you can help programmers make fewer mistakes when using your classes/functions, *etc.* You will lose marks for silly/unprofessional things such as:

- Your code prints messages to the "user". There is no user, it's a lonely game – get used to it. A top-level menu (inside `main`) to select different functionality is acceptable (though unnecessary).
- Your program compiles with the flags "`-Wall -pedantic -g -std=c++11`" and g++ 4.9.2 but has warnings – you should have corrected them.
- Your code does not compile with g++ 4.9.2 – I don't have time to install *N* different IDEs and try code on Linux, Windows, MacOS, etc. – it should work with g++ 4.9.2 on Linux.

Include text files with the input you've used to test your code – this shows how well you've tested it (which also shows how well you understand the code). To test it better, try to break your code.

4 Compiling your Program

Sample basic `Makefile` for the coursework.

```
1 # set the compiler and compiler flags
2 CXX= g++
3 CXXFLAGS= -Wall -pedantic -g -std=c++11
4 CC=$(CXX)
5 CFLAGS=$(CXXFLAGS)
```

5 Submitting your Coursework

Submissions will be done through Moodle.

Please make sure as early as possible that you have access to the module on Moodle.

1. Create a folder named after your login (lowercase letters), e.g., bond007.
2. Copy all your files inside this folder. Do **NOT** use Windows shortcuts or Unix symbolic links.
3. Create a file called `group.txt` and write inside it your estimation of the amount of work each group member did, as a percentage.
4. Compile your code. Does it compile? Does it produce any warnings when compiled with the flags in the makefile?
5. Execute your program – does it execute correctly?
6. Remove object files and executables – these are large and are not useful to us, as we need to recompile.
7. Zip your folder, e.g., `zip -r bond007.zip bond007`

   Use zip not rar, tar, etc.
8. Submit your zip file. A ZIP file, not a RAR/TAR/whatever file!

A C++ Algorithms and C++11 Lambda Functions

Consider the generic function `for_each` below:

```
// Defined inside <algorithm> -- DO NOT COPY THIS CODE!
template<typename InputIter, typename UnaryFunction>
UnaryFunction for_each(InputIter first, InputIter last,  UnaryFunction fn) {
  while (first != last)
    fn(* first++); // Increment first but dereference its old value.
  return fn;
}
```

This generic function allows you to loop over the elements of some container, starting from the one that is pointed to by iterator `first` and ending at the one *right before* iterator `last` (*i.e.*, `last` is *NOT* dereferenced), applying to each of them function `fn`.

Note that the original iterator `first` is not modified, as it is being passed by copy, not by reference.

Here you can see an example of using `for_each`.

```
#include <iostream>
#include <algorithm>
using namespace std;

  template<class T> struct print_it : public unary_function<T, void> {
    ostream &os; // A reference!
    int      count;
     print_it(ostream& out) : os(out), count(0) {}
     void operator() (T x) { os << x << ' '; ++count; }
  };

int main() {
  int       A[] = {1, 4, 2, 8, 5, 7};

  print_it<int> P = for_each(begin(A), end(A),
                             print_it<int>(cout));
  cout << endl << P.count << " objects printed." << endl;
  return 0;
}
```

The class `print_it` modeling the `UnaryFunction` overloads its `operator() (T x)`. Why?
Because, when `for_each` is called with it, it effectively uses it as such:

"`print_it<int>(cout) ( * first++ );`"

What this does is: 1) creates a temporaty object `print_it<int>(cout)`; 2) uses the constructor that takes an output stream on that temporary object (that's why the `cout` is there); and 3) on that temporary object, calls the `operator() (int i)` – `T` is an `int` here, passing it an integer.
Read this and think about it until you understand it.
Notes:

1. A `struct` (as `print_it` here) is effectively a `class` with all fields and methods declared as `public`.
2. The syntax `begin(x)` and `end(x)` works for both containers and arrays! In fact it works ***better*** than `x.begin()` for containers. This is because, these functions return the appropriate type of iterator (either normal one or constant one), while `x.begin()` returns always a normal iterator and `x.cbegin()` returns a constant iterator. Let your compiler choose the iterator it needs (and avoid errors) by using `begin(x)`.
3. `*first++` means: "use ++ to increment `first` by one and return its old value, then apply to it (*i.e.*, the ***old*** value) the value-at-address operator (*i.e.*, "`*`").

Question – once the `for_each` has finished, where will `A` be pointing to?

In the new C++ standard (C++11), one can use ***lambda*** functions to ***significantly*** simplify the use of algorithms such as `for_each` (and `sort`, `find`, *etc.*) as in the following:

```
#include <iostream>
#include <algorithm>
#include <set>
using namespace std;

int main() {
  int       A[] = {1, 4, 2, 8, 5, 7};
  int       count = 0;

  for_each(begin(A), end(A), [ &count ] (int i) { cout << i << ' '; ++count; });
  cout << endl << count << " objects printed." << endl; //*** THAT'S ALL NOW!!!
                  // BELOW ARE MORE EXAMPLES.
  sort(begin(A), end(A), // sort A
       [] (const int & a, const int & b) { return a < b; });
  for_each(begin(A), end(A), [] (const int & i) { cout << i << ' '; });
  cout << endl;

  sort(begin(A), end(A), // sort A in reverse
       [] (const int & a, const int & b) { return a > b; });
  for_each(begin(A), end(A), [] (int i) { cout << i << ' '; }); cout << endl;

  set<int> s1(begin(A), end(A)); s1.insert(1); s1.insert(13); s1.insert(10);
  for_each(begin(s1), end(s1), [] (int i) { cout << i << ' '; }); cout << endl;

  auto comparison = [] (int i1, int i2) { // odd numbers come first
    return ((i1%2) == (i2%2)) ? (i1 < i2) : (1 == i1%2);
  };
  set<int, decltype(comparison)> // non-standard comparison operator!!!
    s2(begin(A), end(A), comparison);
  s2.insert(1); s2.insert(13); s2.insert(10);
  for_each(begin(s2), end(s2), [] (int i) { cout << i << ' '; }); cout << endl;
  auto an_iterator = s2.find(86); // type is "auto" !!!
  if (an_iterator != end(s2))
   cout << "86 is in the set\n";
  else
   cout << "86 is not in the set\n";

  return 0;
}
```

The notation `[ &v1, &v2, v3 ] (T a) { ... }` defines an anonymous lambda function that captures variables `v1` and `v2` from the current environment by reference (so that it can update them internally), while it captures variable `v3` by copy. It takes one parameter of type `T` called `a`.
Note that `for_each` can be potentially faster than writing your own loop with the lambda function's code as the body of the loop (because `for_each` can do "loop unrolling").
If we want to capture all currently visible variables by reference, then we write `[&] (T a) { ... }`.
If we want to capture all currently visible variables by copy, then we write `[=] (T a) { ... }`.
If we want to capture all currently visible variables by copy but v1 by reference, then we write `[=, &v1] (T a) { ... }`. For all by reference but v3 by copy: `[&, v3] (T a) { ... }`.
If we do not want to capture any currently visible variables, then we simply write `[] (T a) { ... }`.
You need to add either the compiler flag `-std=c++0x` or `-std=c++11` (in newer versions of g++), as in: `g++ -std=c++0x -pedantic -Wall -g foo.cc`

Did you spot the "`auto`" in "`auto comparison = ....`"?

Or the "`decltype(comparison)`" in "`set<int, decltype(comparison)>`"?