

CSCI222

Exercise 1

Learning NetBeans, svn, and cppUnit

10 marks

(must be demonstrated in a laboratory class in week 2, 3, or 4)

Aims:

This exercise introduces the NetBeans C++ Integrated Development Environment, simple use of a “subversion” version management system, and the basics of cppUnit testing.

The exercise primarily involves building a C++ class - “MyRecord” - that will be used in a later exercise that creates an “address book” that can be persisted to disk. The exercise makes use of STL, and extensions such as the “Boost C++ library” (e.g. Boost's regex functions), and classes from the qt4 graphics libraries. There are other elements – e.g. an introduction to the source level debugger, profiler tools, etc – which should be attempted.

Objectives:

On completion of this exercise, students should be able to:

- Use the NetBeans IDE to create and run C++ applications;
- Use the built-in support for cppUnit testing and create tests for functions, and test suites for a class;
- Use Linux performance and code coverage profilers
- Use the source-level debugger to observe details of program execution (and find bugs);
- Use the built-in subversion (svn) client to obtain copies of version managed projects;
- Utilise C++ extension libraries, correctly setting include paths for headers and library linkage requirements;
- Employ a stepwise strategy to building an application; a strategy that exploits unit testing.

Task 1: NetBeans C++ projects

This task illustrates creation of simple NetBeans C++ projects and the use of performance profiling tools.

NetBeans will be installed on the Ubuntu systems in the laboratory (you will typically access it via the 'Dash Home' entry); the lab should have version 8.02. (Some of the illustrations in this document are still screenshots from an older version.)

NetBeans creates a number of subdirectories within your home directory where it stores its private configuration data; it also creates a directory “NetBeansProjects” which will be the default directory for holding subsequently created projects.

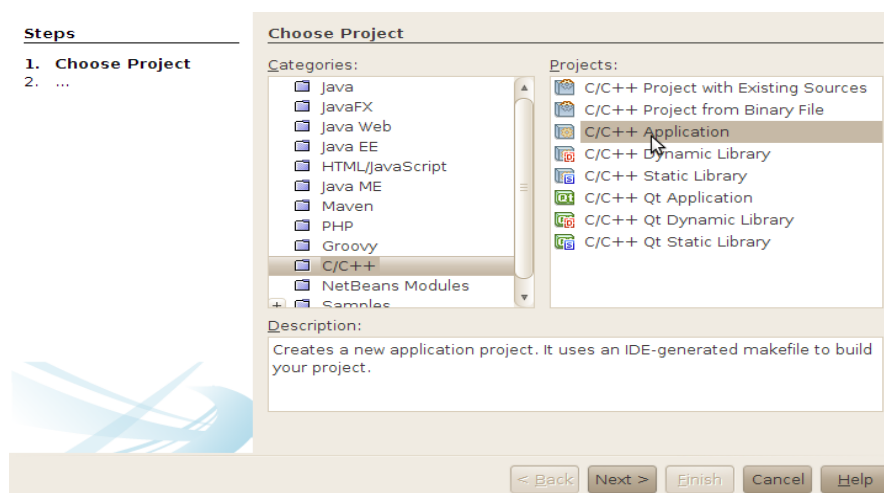
If you have used NetBeans previously, e.g. in CSCI110, you may already have some of these directories set up – e.g. .netbeans, .netbeans-registration, .netbeans-derby. Old data in such directories can confuse the configuration process. It is best to delete all old data before starting to use NetBeans for CSCI222. (At shell level, use commands such as `rm -rf .netbeans`.)

The Ubuntu system has provision for a password management system that controls application specific passwords. This can sometimes interfere – you may get challenged for your “keyring” password when starting NetBeans; if you cannot remember your keyring password, NetBeans will not start. The simplest solution to this problem is to remove any “keyring” data from your account. Change into the “.gnome2/keyrings” subdirectory. Delete all files there. Try again with NetBeans.

Start NetBeans:

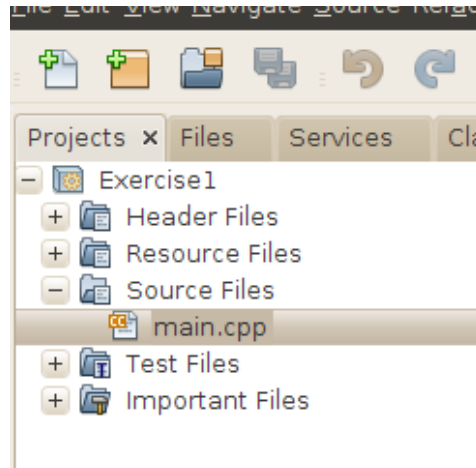


Create a new C++ project (File/New Project menu):

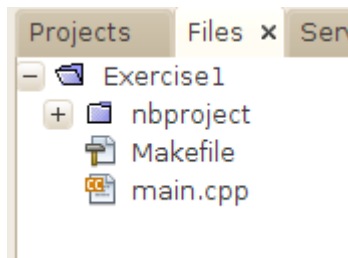


It should be a “C/C++ application”. The second dialog in the creation process allows you to specify the project name, e.g. Exercise1, a location (this will default to having Exercise1 inside the NetBeansProjects directory; it is best not to accept the default, create an extra subdirectory within NetBeansProject – or some other directory of your choice). NetBeans will generate the project along with an essentially empty main():

The project structure is shown:



(The “folders” shown - “Header Files”, “Source Files” etc - are fictions. The actual file structure as generated is:



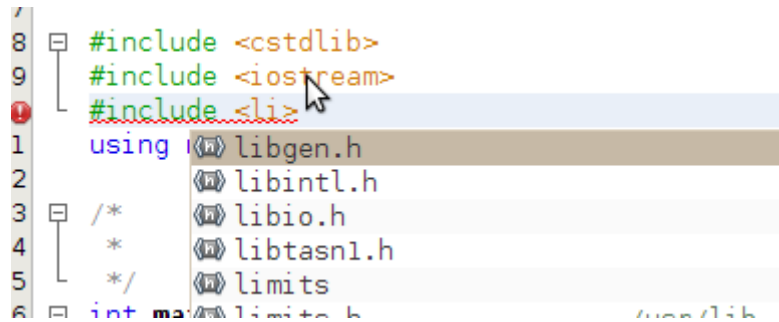
there are no subdirectories for header and source files. Still, these fictitious folders will be useful for organising your code in larger projects.)

The generated main() is standard:

```
main.cpp x
Source History
1  /*
2   * File:   main.cpp
3   * Author: neil
4   *
5   * Created on 9 April 2015, 3:14 PM
6   */
7
8  #include <cstdlib>
9
10 using namespace std;
11
12 /*
13  *
14  */
15 int main(int argc, char** argv) {
16     return 0;
17 }
18
```

This first exercise uses a standard sort function. The application will populate an array of doubles and then sort them using a “selection sort” function.

Start by adding #include statements to the main.cpp file – obviously will need iostream, and others. One of the others will be the file that contains details of numerical limits (e.g. largest double); NetBeans' auto-completion feature cuts in here – it helps find the correct headers:



Next implement the driver code – the sort function is left unimplemented at this stage.

```
#include <cstdlib>
#include <iostream>
#include <limits>
using namespace std;

static const int numitems = 25000;
static const int ranseed = 12345;

static void fillArray(double data[], int n) {
    // deliberately seeding the random number generator with a constant
    // seed so as to get the same value sequence each time.
    // This simplifies performance comparisons.
    srand(ranseed);
    for(int i=0; i<n; i++) {
        double d = (double)rand()/((double)RAND_MAX);
        data[i] = (d - 0.5)*numeric_limits<double>::max();
    }
}

static void showsample(double data[], int n) {
    cout << "First 5 entries in array" << endl;
    for(int i=0; i<5; i++) cout << "data[" << i << "] = " << data[i] << endl;
    cout << "Last 5 entries in array" << endl;
    for(int i=n-1; i>n-6; i--) cout << "data[" << i << "] = " << data[i] << endl;
}

void selectionsort(double data[], int datasize) {
}
```

```

int main(int argc, char** argv) {
    double data[numitems];
    fillArray(data,numitems);
    cout << "Random data generated" << endl;
    showsample(data,numitems);
    cout << "Invoking sort" << endl;
    selectionsort(data,numitems);
    cout << "After sort" << endl;
    showsample(data,numitems);
    return 0;
}

```

Select the project, and right-click to select “Run”. It should run, but of course the data do not get sorted.

Random data generated

First 5 entries in array

data[0] = 6.11553e+307

data[1] = -1.89867e+307

data[2] = 5.08926e+307

data[3] = 5.36504e+307

data[4] = 7.40016e+307

Last 5 entries in array

data[24999] = 4.55828e+307

data[24998] = -4.76844e+307

data[24997] = 8.60262e+307

data[24996] = -5.40848e+307

data[24995] = 8.55222e+307

Invoking sort

After sort

First 5 entries in array

data[0] = 6.11553e+307

data[1] = -1.89867e+307

data[2] = 5.08926e+307

data[3] = 5.36504e+307

data[4] = 7.40016e+307

Last 5 entries in array

data[24999] = 4.55828e+307

data[24998] = -4.76844e+307

data[24997] = 8.60262e+307

data[24996] = -5.40848e+307

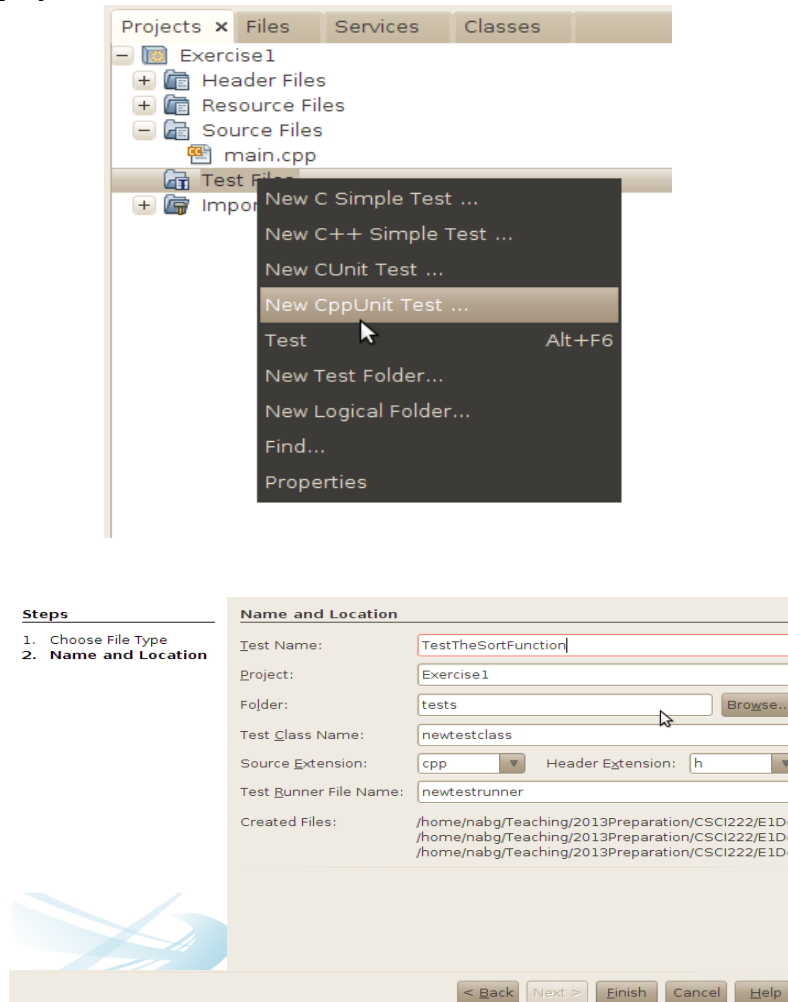
data[24995] = 8.55222e+307

Why run code with no implementation of the principal function?

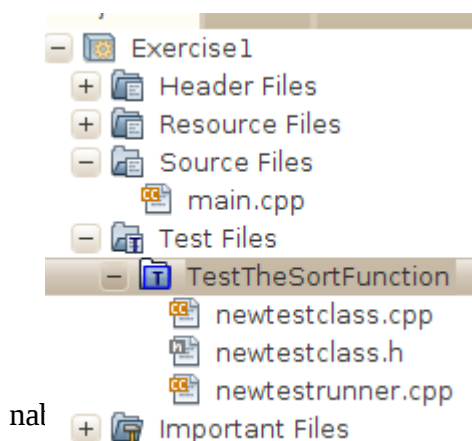
It's part of the ethos of “Test Driven Development”. You start by setting up an overall framework that will allow you to test code before you write the actual code.

CppUnit

Might as well start using the cppUnit testing framework from this point; so add the unit test elements to the project:



NetBeans adds a few code files for testing the project:



The “newtestrunner.cpp” file contains a version of the standard cppUnit driver program; this creates and runs the tests that will be defined in newtestclass.

```
#include <cppunit/BriefTestProgressListener.h>
#include <cppunit/CompilerOutputter.h>
#include <cppunit/extensions/TestFixtureRegistry.h>
#include <cppunit/TestResult.h>
#include <cppunit/TestResultCollector.h>
#include <cppunit/TestRunner.h>

int main() {
    // Create the event manager and test controller
    CPPUNIT_NS::TestResult controller;

    // Add a listener that collects test result
    CPPUNIT_NS::TestResultCollector result;
    controller.addListener(&result);

    // Add a listener that print dots as test run.
    CPPUNIT_NS::BriefTestProgressListener progress;
    controller.addListener(&progress);

    // Add the top suite to the test runner
    CPPUNIT_NS::TestRunner runner;
    runner.addTest(CPPUNIT_NS::TestFixtureRegistry::getRegistry().makeTest())
    runner.run(controller);

    // Print test in a compiler compatible format.
    CPPUNIT_NS::CompilerOutputter outputter(&result, CPPUNIT_NS::stdCout());
    outputter.write();

    return result.wasSuccessful() ? 0 : 1;
}
```

The generated newtestclass (files newtestclass.h and newtestclass.cpp) contain the basic structure for a cppUnit test class; it has been given two dummy functions – testMethod() and testFailedMethod() - that illustrate the format for simple tests. It also contains the standard setUp() and tearDown() functions (whose use will be explained in the lectures covering cppUnit testing).

The setUp() and tearDown() functions should be left with null implementations in this example.

```

#ifndef NEWTESTCLASS_H
#define NEWTESTCLASS_H

#include <cppunit/extensions/HelperMacros.h>

class newtestclass : public CPPUNIT_NS::TestFixture {
    CPPUNIT_TEST_SUITE(newtestclass);

    CPPUNIT_TEST(testMethod);
    CPPUNIT_TEST(testFailedMethod);

    CPPUNIT_TEST_SUITE_END();

public:
    newtestclass();
    virtual ~newtestclass();
    void setUp();
    void tearDown();

private:
    void testMethod();
    void testFailedMethod();
};

CPPUNIT_TEST_SUITE_REGISTRATION(newtestclass);

} newtestclass::newtestclass() {
}

} newtestclass::~~newtestclass() {
}

} void newtestclass::setUp() {
}

} void newtestclass::tearDown() {
}

} void newtestclass::testMethod() {
    CPPUNIT_ASSERT(true);
}

} void newtestclass::testFailedMethod() {
    CPPUNIT_ASSERT(false);
}

```

We need to test the implementation of a sort function – so we had better first decide on how we could test whether it is working correctly for large sets of data.

One very simple (but insufficient) test would check that the sum of the values in the sorted and

original sets of data were the same.

A second rather more costly test would verify that each data element in (a copy of) the original array was matched by an element in the sorted array.

A third test could verify that the “sorted” data were ordered.

We can define test functions that apply such tests. First, delete the auto-generated testMethod() and testFailedMethod() from both the header and implementation files, and then define the functions testSum(), testElements(), testOrdering():

```
#ifndef NEWTESTCLASS_H
#define NEWTESTCLASS_H

#include <cppunit/extensions/HelperMacros.h>

class newtestclass : public CPPUNIT_NS::TestFixture {
    CPPUNIT_TEST_SUITE(newtestclass);

    CPPUNIT_TEST(testSum);
    CPPUNIT_TEST(testElements);
    CPPUNIT_TEST(testOrdering);

    CPPUNIT_TEST_SUITE_END();

public:
    newtestclass();
    virtual ~newtestclass();
    void setUp();
    void tearDown();

private:
    void testSum();
    void testElements();
    void testOrdering();
};

#endif /* NEWTESTCLASS_H */
```

Test function named in the CPPUNIT macros that generate code that invokes these tests.

Test functions declared here

The functions are defined in newtestclass.cpp. These test functions have to call the selectionsort defined in main.cpp.

```
void selectionsort(double data[], int datasize) {
}

int main(int argc, char** argv) {
    double data[numitems];
```

An “extern” reference to this function will have to appear in the newtestclass.cpp file:

```
#include "newtestclass.h"
extern void selectionsort(double[], int);

CPPUNIT_TEST_SUITE_REGISTRATION(newtestclass);
```

The test functions should now be defined:

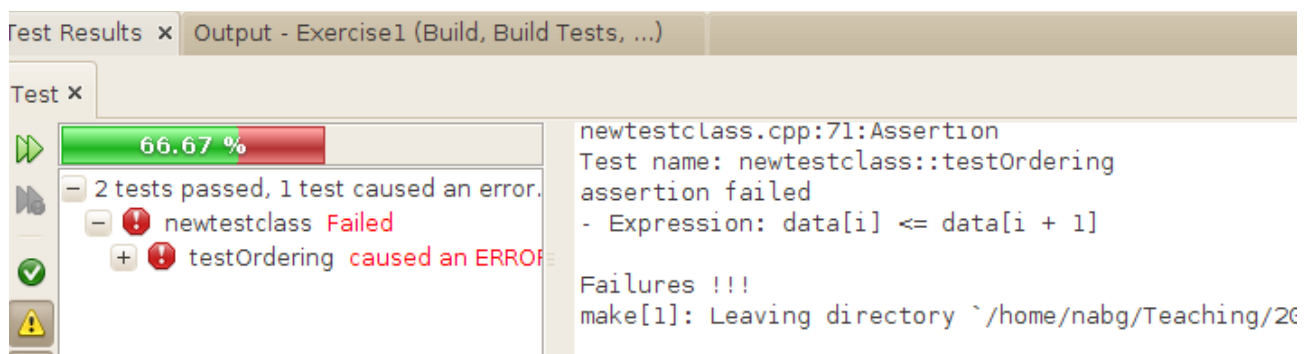
```
void newtestclass::testSum() {
    double data[] = {0.1, -0.5, 1.2, -2.7, 7.8, 9.1, -13.3, 102.4, 68.7, 31.2};
    int num = sizeof (data) / sizeof (double);
    double initsum = 0.0;
    for (int i = 0; i < num; i++) initsum += data[i];
    selectionsort(data, num);
    double finalsum = 0.0;
    for (int i = 0; i < num; i++) finalsum += data[i];
    // The sums should be equal, but with floating point one could get
    // round-off so the check defines a tolerance; here, 0.1
    CPPUNIT_ASSERT_DOUBLES_EQUAL(initsum, finalsum, 0.1);
}

void newtestclass::testElements() {...}

void newtestclass::testOrdering() {
    double data[] = {0.1, -0.5, 1.2, -2.7, 7.8, 9.1, -13.3, 102.4, 68.7, 31.2};
    int num = sizeof (data) / sizeof (double);
    // Sort the data
    selectionsort(data, num);
    // Hopefully now in ascending order!
    for (int i = 0; i < num - 1; i++)
        CPPUNIT_ASSERT(data[i] <= data[i + 1]);
}
```

(The definition of testElements() is left to you; I'm not going to do ALL the work of this exercise! Your code should check two copies of an array; they start with the same contents; one is “sorted” using the selectionsort function; they should still have the same contents!)

Select the project, and right-click select “Test”. The testrunner program is run and the tests are executed. By default, the testrunner program generated by NetBeans uses a visual feedback progress bar. (Actually, visual feedback works only if the C++ tool set includes Qt's qmake, usually in /usr/bin/qmake. If qmake is not available, you will get a text report.)



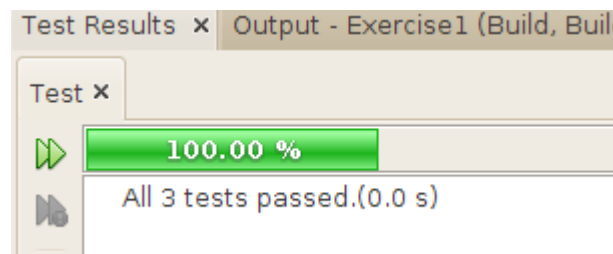
It doesn't work yet.

Maybe we should implement an actual sort function that will put the data elements into ascending order.

How about this as an implementation?

```
void selectionsort(double data[], int datasize) {
    for (int i = 0; i < datasize - 1; i++) {
        int min = i;
        for (int j = i + 1; j < datasize; j++)
            if (data[j] < data[min]) min = j;
        double temp = data[min];
        data[min] = data[i];
        data[i] = temp;
    }
}
```

Run the test:



Happiness is a “green bar” indicating that all tests have been passed.

Now we have some confidence that the sort function has been correctly implemented, we can run the driver program with its larger data set.

Random data generated

First 5 entries in array

data[0] = -5.78147e+307

data[1] = -1.80349e+307

data[2] = -5.99353e+307

data[3] = -5.17517e+307

data[4] = -7.87505e+307

Last 5 entries in array

data[24999] = 3.68557e+307

data[24998] = -4.11509e+307

data[24997] = 8.07361e+307

data[24996] = -3.31652e+307

data[24995] = 7.11684e+307

Invoking sort

After sort

First 5 entries in array

data[0] = -8.98785e+307

data[1] = -8.9875e+307

data[2] = -8.98535e+307

nabg 2015

```
data[3] = -8.98319e+307
data[4] = -8.98307e+307
Last 5 entries in array
data[24999] = 8.98701e+307
data[24998] = 8.98616e+307
data[24997] = 8.98392e+307
data[24996] = 8.98354e+307
data[24995] = 8.98297e+307
```

RUN SUCCESSFUL (total time: 1s)

Things seem to be working.

Profiling

The next step is to use profiling tools. Some are standard – being part of the g++ compiler suite; we will only use these standard tools. In a professional environment, you would have additional commercial tools such as memory leak detectors. (These add profiling code to the memory manager parts of your program to track the allocation and freeing of memory – as with new and delete in C++. Valgrind – not currently installed in the lab – is a free leak detector for Linux. Worth installing on your own machine.)

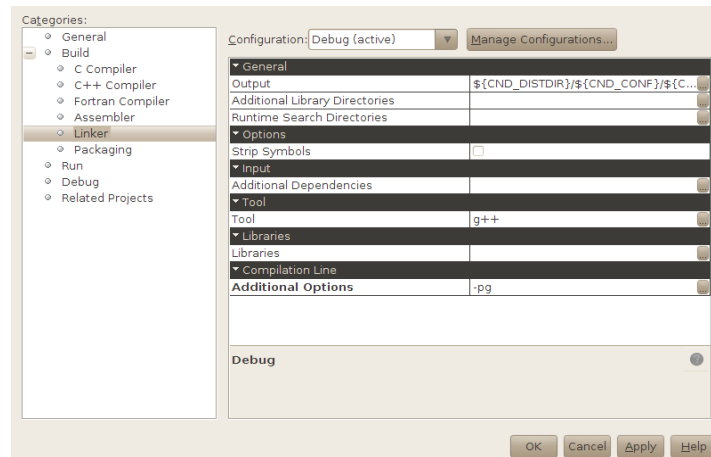
The two standard profiling tools are gprof and gcov. The gprof profiling tool helps you identify where your program spends most of its time, while the gcov tool is for “code coverage” testing. These tools are standard in the Linux g++ distribution. There is no special support built into NetBeans. You use the profilers by first setting compiler and linker options before you “Clean and Build” a NetBeans C++ project, and then you run the application from within NetBeans. The profiling code that will have been added to your application will generate data files with the raw information used by the profiler programs. You run the profiler programs at the command line in a Linux Xterm terminal session.

The results for this little sort application are pretty much a foregone conclusion – almost all the time will be spent in that sort routine, and those statements in that routine are going to be executed numerous times.

But it's still worthwhile running the profilers – simply to get experience on how to set things up and view results.

gprof performance profiling:

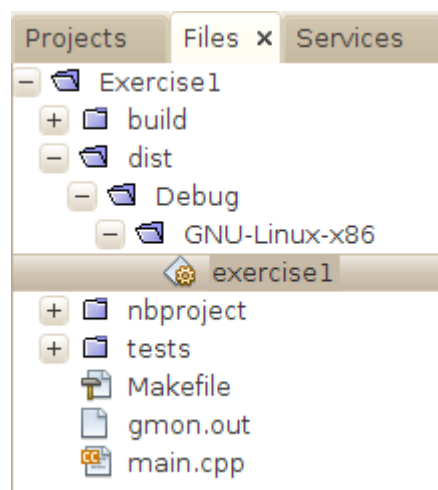
The C++ compiler, and linker options as defined in the project's “Properties” (select the project, right-click and select the Properties option in the pop-up menu) need to be changed; they must now both specify -pg in the Compilation Line/Additional Options element:



(You must add the **-pg** in both C++ compiler and Linker sections – the screenshot shows just linker)

Whenever you change the project properties, you must use the “Clean and Build” option to rebuild the program completely.

The application should again be run, the timing data get written to a file “gmon.out” that is created in the working directory.



The gprof profiler must be run at the command line. Start a Linux terminal session and navigate to the directory with your Exercise1 project:

```
neil@nabgX:~$ cd Teaching/222-15/Exercises/Projects/
neil@nabgX:~/Teaching/222-15/Exercises/Projects$ ls
Exercise1
neil@nabgX:~/Teaching/222-15/Exercises/Projects$ cd Exercise1/
neil@nabgX:~/Teaching/222-15/Exercises/Projects/Exercise1$ ls
build dist gmon.out main.cpp Makefile nbproject tests
neil@nabgX:~/Teaching/222-15/Exercises/Projects/Exercise1$
```

The gprof application runs in the directory with the gmon.out file, it takes a command line argument that is the pathname to the executable program:

```

s/Exercisel$ gprof ./dist/Debug/GNU-Linux-x86/exercisel

```

The profile report shows the amount of time in different functions:

```

granularity: each sample hit covers 2 byte(s) for 1.03% of 0.98 seconds

index % time    self  children   called    name
-----
[1]   100.0    0.00    0.98        1/1    <spontaneous>
      0.98    0.00        1/1    main [1]
      0.00    0.00        2/2    selectionsort(double*, int) [2]
      0.00    0.00        1/1    showsample(double*, int) [10]
      0.00    0.00        1/1    fillArray(double*, int) [13]
-----
[2]   100.0    0.98    0.00        1/1    main [1]
      0.98    0.00        1/1    selectionsort(double*, int) [2]
-----
[9]    0.0    0.00    0.00  25000/25000    fillArray(double*, int) [13]
      0.00    0.00        25000    std::numeric_limits<double>::max() [9]

```

```

Each sample counts as 0.01 seconds.
%   cumulative   self           self             total
time  seconds    seconds   calls   ms/call  ms/call  name
100.53    0.98    0.98         1    975.19    975.19  selectionsort(double*, int)
  0.00    0.98    0.00      25000     0.00     0.00  std::numeric_limits<double>::max()
  0.00    0.98    0.00         2     0.00     0.00  showsample(double*, int)
  0.00    0.98    0.00         1     0.00     0.00  _GLOBAL__sub_I_Z13selectionsortPdi
  0.00    0.98    0.00         1     0.00     0.00  __static_initialization_and_destruction_0(int, int)
  0.00    0.98    0.00         1     0.00     0.00  fillArray(double*, int)

```

As expected, all the time is spent in that sort function.

In larger applications, the time-consuming parts of the program aren't always obvious. It's a heuristic that 80% of the CPU time will be spent in only 20% of the code; so if your program is a little too slow, it's worth trying to identify that 20% before you spend time on trying to optimise your code.

gcov code coverage:

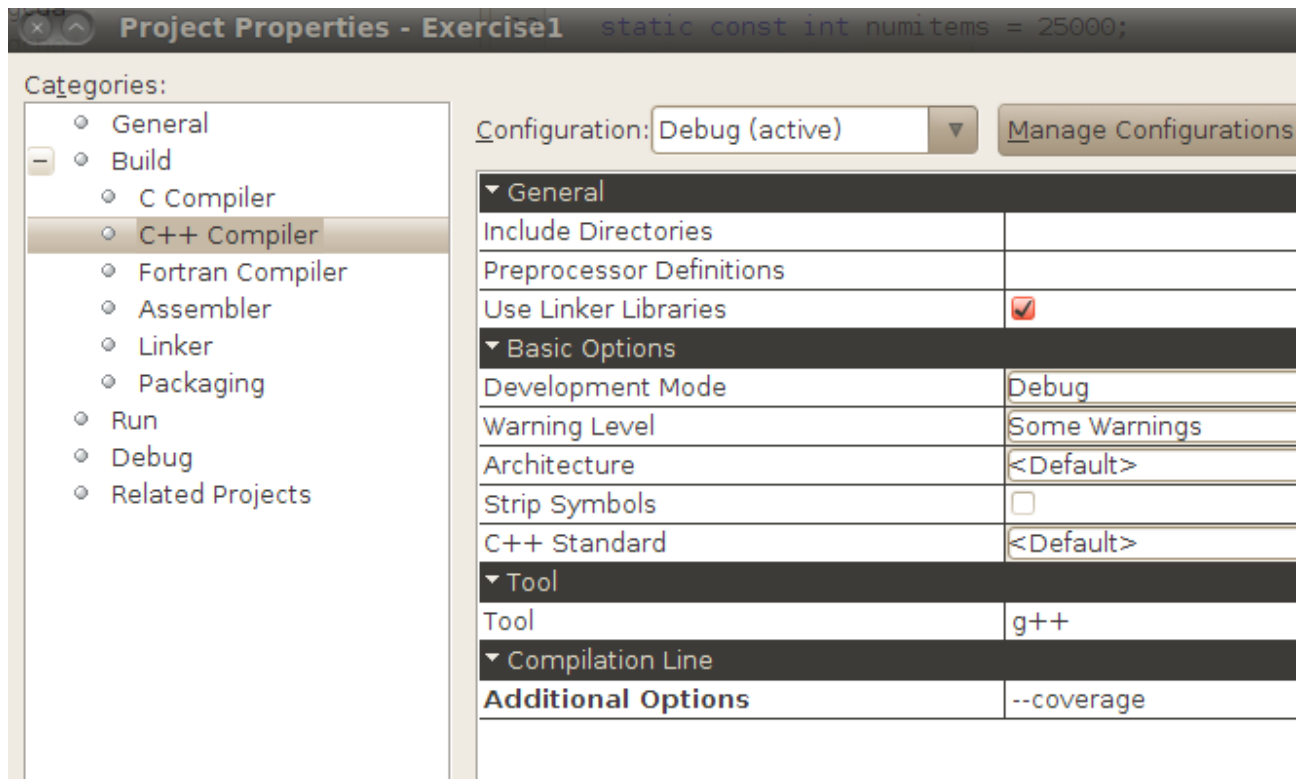
Code coverage analysis provides those annotated listings that show how often each statement in your program has been executed. These listings are useful as they provide some guide as to whether you have executed all parts of your code – including paths that deal with atypical data.

Code coverage works by adding extra code that maintains counters for each program block; when a run of the program terminates, the values in these counters are written to disk. The code coverage system can accumulate data over several separate runs of a program that use different data inputs; the outputs from the different runs are combined automatically.

When you think that you have run sufficient tests, you use the gcov tool to analyse the recorded

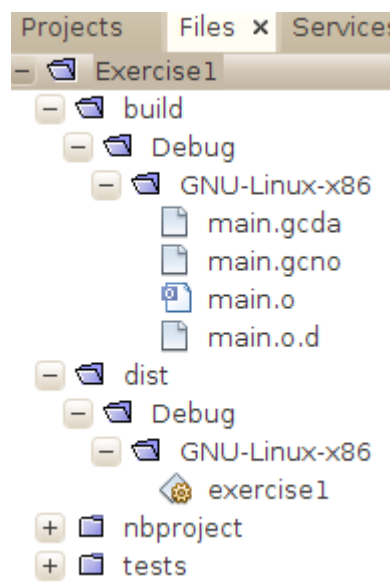
data.

For code coverage analysis, both the compiler and linker stages will need `--coverage` specified as a compilation line additional option (only compiler shown in screenshot):



The program should be “Clean and Build” rebuilt, and then run.

The profiling data are recorded in a couple of files (e.g. data for code in main.cpp is in the files main.gcda and main.gcno) – one holds counts, the other maps the code blocks to lines of code in the source file. In the NetBeans set up, these files are created in the “build” subdirectory of the project:



In most cases, with programs that accept input, you will want to run the program several times using different input data so as to accumulate more comprehensive profiling data. The example sorting program need only be run once.

When sufficient profiling data have been acquired, you should run the gcov tool in the project directory. The gcov program can produce a variety of reports, but most are difficult for non-experts to interpret (in part the problem is due to the C++ compiler creating its own linkage names for all the functions defined in the program making it hard to interpret a report on functions). The most useful report for average programmers is the “branch-count” report. These have to be generated for each source file.

For this example, we would need a branch-count profile of main.cpp, with the data being in the build/Debug/GNU-Linux-x86 subdirectory; the command will be `gcov -o path-to-data-files -c main.cpp`:

```
nabg@info-bxsrr1s:~/Teaching/2013Preparation/CSCI222/E1Demos/Exercise1$ pwd
/home/nabg/Teaching/2013Preparation/CSCI222/E1Demos/Exercise1
nabg@info-bxsrr1s:~/Teaching/2013Preparation/CSCI222/E1Demos/Exercise1$ ls
basic_ios.h.gcov  char_traits.h.gcov  gmon.out  istream.gcov  locale  facets.h.gcov  main.cpp.gcov  nbproject  tests
build             dist               ios_base.h.gcov  limits.gcov  main.cpp  Makefile       ostream.gcov
nabg@info-bxsrr1s:~/Teaching/2013Preparation/CSCI222/E1Demos/Exercise1$ gcov -o ./build/Debug/GNU-Linux-x86/ -c main.cpp
```

That command creates a .gcov text file with the report. You can view the contents of the file using the 'cat' command. The report shows the number of times each statement has been executed:


```

-: 0:Programs:1
-: 1:/*
-: 2: * File:   main.cpp
-: 3: * Author: neil
-: 4: *
-: 5: * Created on 9 April 2015, 3:14 PM
-: 6: */
-: 7:
-: 8:#include <cstdlib>
-: 9:#include <iostream>
-: 10:#include <limits>
-: 11:using namespace std;
-: 12:
-: 13:static const int numitems = 25000;
-: 14:static const int ranseed = 12345;
-: 15:
1: 16:static void fillArray(double data[], int n) {
-: 17:     // deliberately seeding the random number generator with a constant
-: 18:     // seed so as to get the same value sequence each time.
-: 19:     // This simplifies performance comparisons.
1: 20:     srand(ranseed);
25001: 21:     for (int i = 0; i < n; i++) {
25000: 22:         double d = (double) rand() / (double) RAND_MAX;
25000: 23:         data[i] = (d - 0.5) * numeric_limits<double>::max();
-: 24:     }
1: 25:}
-: 26:
2: 27:static void showsample(double data[], int n) {
2: 28:     cout << "First 5 entries in array" << endl;
2: 29:     for (int i = 0; i < 5; i++) cout << "data[" << i << "] = " << data[i] << endl;
2: 30:     cout << "Last 5 entries in array" << endl;
2: 31:     for (int i = n - 1; i > n - 6; i--) cout << "data[" << i << "] = " << data[i] << endl;
2: 32:}
-: 33:
1: 34:void selectionsort(double data[], int datasize) {
25000: 35:     for (int i = 0; i < datasize - 1; i++) {
24999: 36:         int min = i;
312512499: 37:         for (int j = i + 1; j < datasize; j++)
312487500: 38:             if (data[j] < data[min]) min = j;
24999: 39:         double temp = data[min];
24999: 40:         data[min] = data[i];

```

Task 1 – completion (3 marks)

When demonstrating your work on this exercise you should show your laboratory supervisor your cppUnit test code and demonstrate execution of the testrunner program, and show the results of either the performance profiler or code coverage profiler.

Task 2: Source-level debugger

There must have been occasions in the past where you wished that you could see what was happening inside your program. The g++ compiler suite includes the gdb debugger that can provide some insights – but use of gdb at the command line is a little arcane, and most students find it too complex. NetBeans incorporates a visual interface to gdb; this makes the use of this debugger a lot simpler.

For this task, you will create a second project – Exercise1B. The program is again simple – just a driver program and a sort routine. This time it's a version of Quicksort (taken from [my old C++ notes circa 1996](#)). I think that this implementation of Quicksort is bug-free; its value for this example is that it is a recursive function so we can view something a bit more complex as it builds up a nested call stack.

The driver code for task 2 is the same as that for task 1 – so you can simply cut and paste that code. (NetBeans allows you to copy files from one project to another by file copy-and-paste operations in its “Projects” display pane. You should avoid doing this with C++ projects! NetBeans has a XML configuration file for C++ projects that contains records of the header and source files etc; this configuration file isn't updated properly if you copy-and-paste files.)

You must remember the Quicksort algorithm – the one that recursively partitions the array into subsets that are sorted separately. The basic code is as follows:

```

static int Partition(double d[], int left, int right) {
    double val = d[left];
    int lm = left - 1;
    int rm = right + 1;
    for (;;) {
        do
            rm--; while (d[rm] > val);
        do
            lm++; while (d[lm] < val);
        if (lm < rm) {
            double tempr = d[rm];
            d[rm] = d[lm];
            d[lm] = tempr;
        } else
            return rm;
    }
}

static void Quicksort(double d[], int left, int right) {
    if (left < right) {
        int split_pt = Partition(d, left, right);
        Quicksort(d, left, split_pt);
        Quicksort(d, split_pt + 1, right);
    }
}

int main(int argc, char** argv) {
    double data[numitems];
    fillArray(data, numitems);
    cout << "Random data generated" << endl;
    showsample(data, numitems);
    cout << "Invoking sort" << endl;
    Quicksort(data, 0, numitems - 1);
    cout << "After sort" << endl;
    showsample(data, numitems);
    return 0;
}

```

It runs (it's not buggy) – and it is quicker than selectionsort (10ms instead of ~1s):

Exerciselb (Build, Run) × Exerciselb (Run) ×

```
data[3] = -5.17517e+307
data[4] = -7.87505e+307
Last 5 entries in array
data[24999] = 3.68557e+307
data[24998] = -4.11509e+307
data[24997] = 8.07361e+307
data[24996] = -3.31652e+307
data[24995] = 7.11684e+307
Invoking sort
After sort
First 5 entries in array
data[0] = -8.98785e+307
data[1] = -8.9875e+307
data[2] = -8.98535e+307
data[3] = -8.98319e+307
data[4] = -8.98307e+307
Last 5 entries in array
data[24999] = 8.98701e+307
data[24998] = 8.98616e+307
data[24997] = 8.98392e+307
data[24996] = 8.98354e+307
data[24995] = 8.98297e+307
```

RUN SUCCESSFUL (total time: 210ms)

Supposedly, it can be improved by combining the partitioning recursive Quicksort mechanism with a different sort that is applied when the partitions are small (in this example, that will be taken as a partition size of 15). The revised code is:

```

void SelectionSort(double data[], int left, int right) {
    for (int i = left; i < right; i++) {
        int min = i;
        for (int j = i + 1; j <= right; j++)
            if (data[j] < data[min]) min = j;
        double temp = data[min];
        data[min] = data[i];
        data[i] = temp;
    }
}

int Partition(double d[], int left, int right) {
    double val = d[left];
    int lm = left - 1;
    int rm = right + 1;

    for (;;) {
        do
            rm--; while (d[rm] > val);
        do
            lm++; while (d[lm] < val);
        if (lm < rm) {
            double tempr = d[rm];
            d[rm] = d[lm];
            d[lm] = tempr;
        } else
            return rm;
    }
}

void Quicksort(double d[], int left, int right) {
    if (left < (right - kSMALL_ENOUGH)) {
        int split_pt = Partition(d, left, right);
        Quicksort(d, left, split_pt);
        Quicksort(d, split_pt + 1, right);
    } else SelectionSort(d, left, right);
}

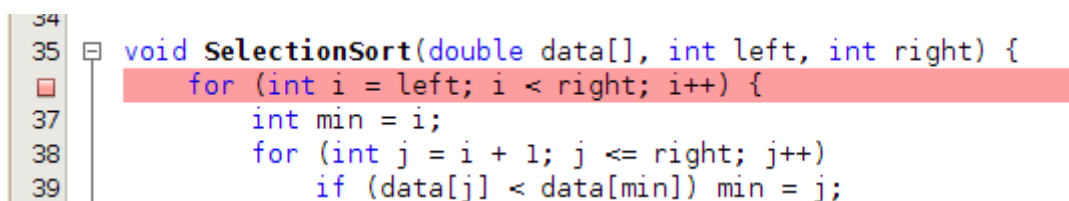
```

(It's still correct and still runs and produces sorted data. But it wasn't actually faster.)

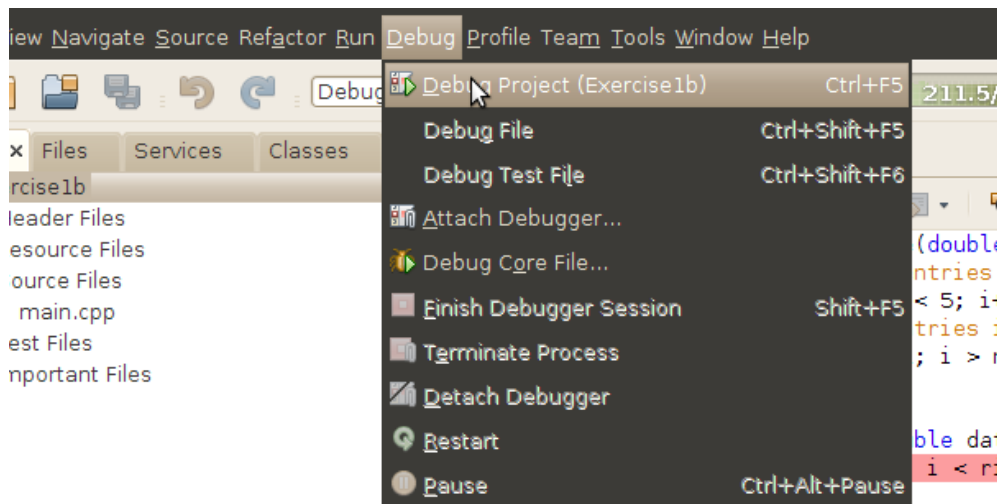
What about using the visual debugger to see how it works?

Just as a simple example, how about stopping execution at the start of the “Selectionsort” function. This will allow a user to view the call stack, examine partially sorted arrays etc.

You start by placing a breakpoint in the source code (click in the line-numbering margin):



Then start the program in debug mode:



The program will immediately start and run until it reaches that breakpoint (or, in the case of a buggy program, until it reaches something that causes a run-time exception like arithmetic overflow, or dereferencing an uninitialised pointer). In this case it should reach the breakpoint:

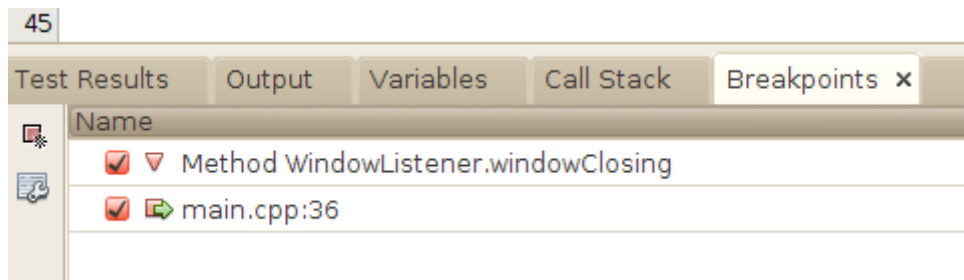
```
34
35 void SelectionSort(double data[], int left, int right) {
36     for (int i = left; i < right; i++) {
37         int min = i;
38         for (int j = i + 1; j <= right; j++)
39             if (data[j] < data[min]) min = j;
40         double temp = data[min];
41         data[min] = data[i];
42         data[i] = temp;
43     }
44 }
45
```

Test Results Output × Variables Call Stack Breakpoints

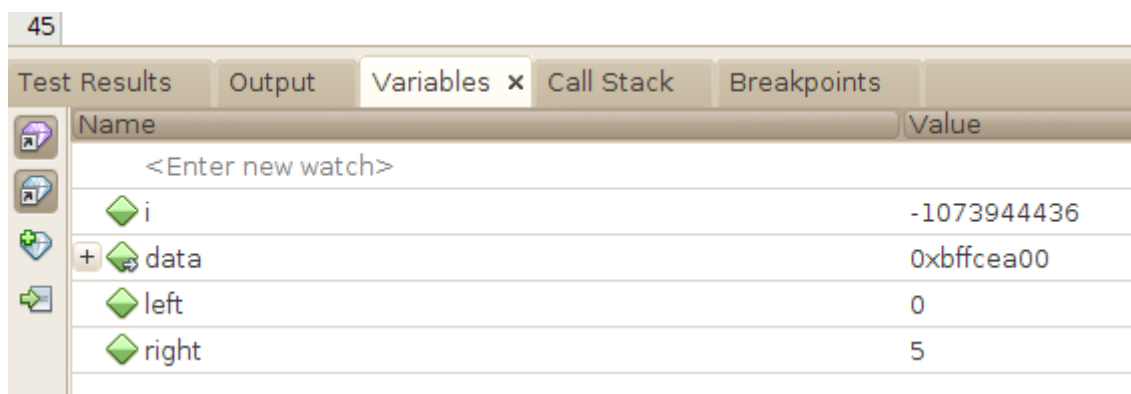
Exercise1b (Build, Debug) × Exercise1b (Debug) ×

Random data generated
First 5 entries in array
data[0] = -5.78147e+307
data[1] = -1.80349e+307
data[2] = -5.99353e+307
data[3] = -5.17517e+307
data[4] = -7.87505e+307
Last 5 entries in array
data[24999] = 3.68557e+307
data[24998] = -4.11509e+307
data[24997] = 8.07361e+307
data[24996] = -3.31652e+307
data[24995] = 7.11684e+307
Invoking sort

The breakpoints pane in the debug display allows you to view the breakpoints that you have set (if there isn't a "Breakpoints" tab, go to the Window/Debugging menu and enable this feature):

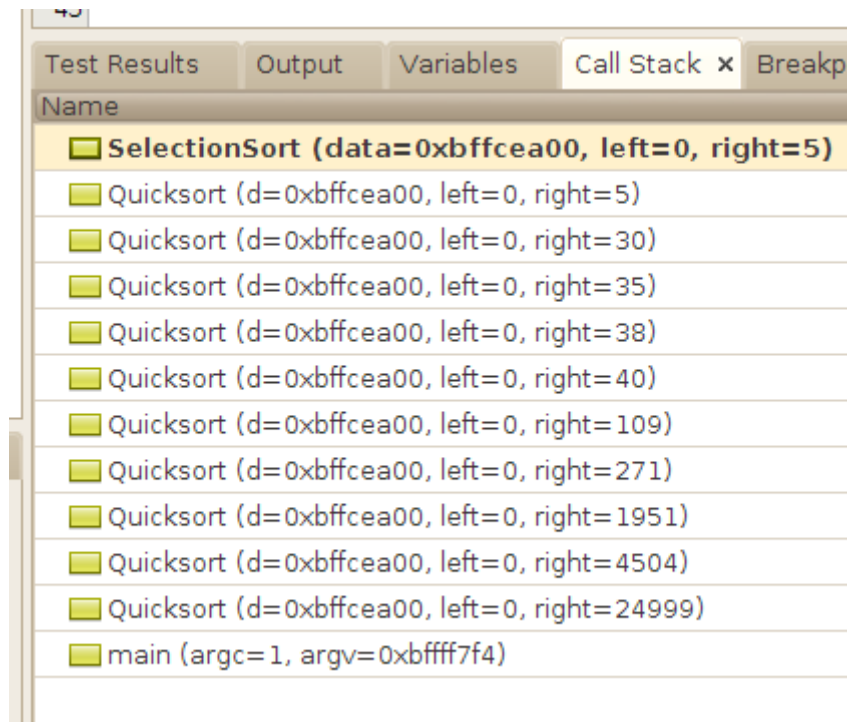


The variables pane shows the value of local variables in the current stack frame:



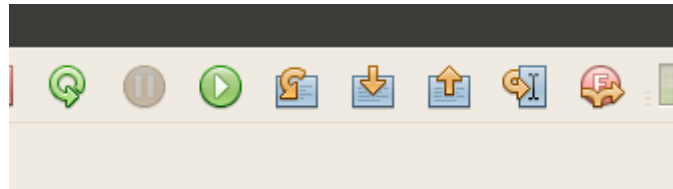
(Note the value for "i" - junk! The for-loop hasn't been started yet so the value is just from random bits left in memory by a previously running program.)

The "Call Stack" pane shows the call stack (which is what you would expect):



You can select an entry in the call stack, and then switch to the “Variables” pane to view the stack frame for that entry.

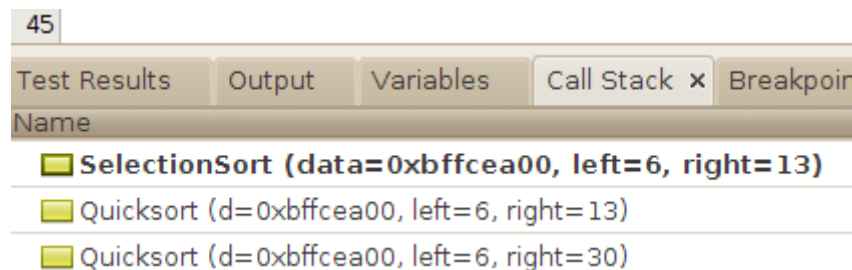
Execution can be resumed and controlled using the action buttons in the options bar:



You can “step over” the code of the SelectionSort



The program will then resume and pause again when it next re-enters the selection sort function and hits the breakpoint:



You could set other breakpoints, or step line by line – using the variables pane to inspect the values of variables as they change.

Task 2 – completion (1 mark)

The laboratory demonstrator may ask you to demonstrate use of the debugger on this SelectionSort example or demonstrate its use in code from later tasks in this exercise.

Task 3: Subversion and use of libraries

This task provides some initial experience with the use of the subversion version management system and the use of C++ libraries.

The “virindi” server (virindi.cs.uow.edu.au) runs an instance of the subversion repository with an Apache/WebDav client interface.

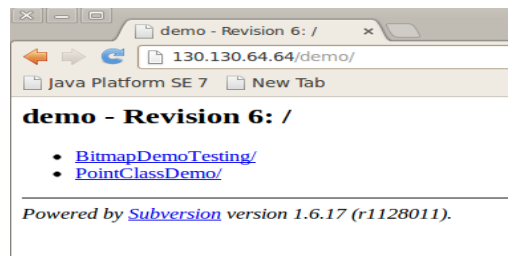
There are a few example projects hosted in this repository, and there are repositories set up for use in the group development assignment. The examples are in repositories that permit anonymous login and download; the group directories require passwords.

The example projects:

- /demo:

These examples date from 2007, the versions in the demo repository were created in 2009.

The projects are “BitmapDemoTesting” and “PointClassDemo” - these may get referenced in lectures and are used in an older version of this exercise.



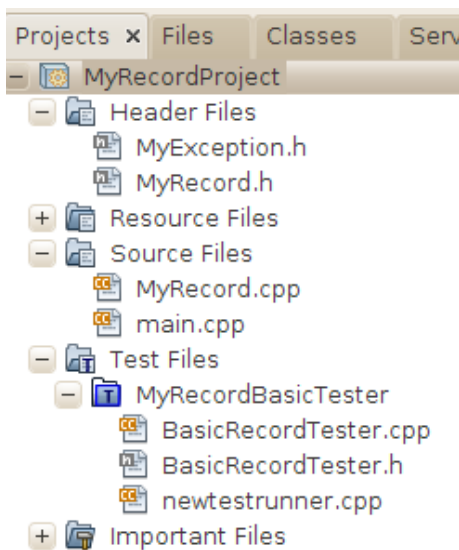
- /nabg

This repository contains the initial files for the “MyRecord” example.

MyRecord

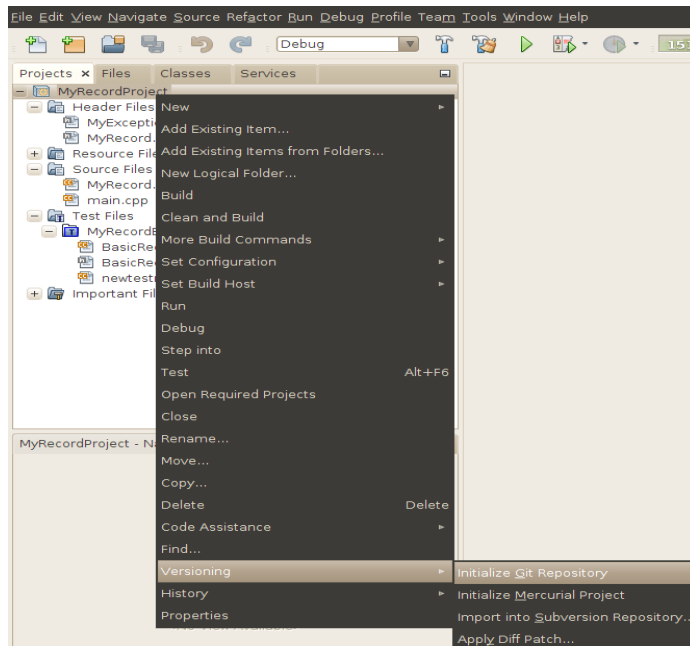
The MyRecord example is part of a larger example that builds a kind of address book. Entries in the address book have data such as a nickname, actual name, addresses (possibly several, e.g. “home”, “term-time”), and phone numbers (several “home”, “mobile”, “work”). An address book will consist of a collection of MyRecord entries, using some form of persistent storage.

The project started as a standard NetBeans C++ project, utilising cppUnit testing.



I put these initial files into a subversion system (I was using a temporary virtual machine at 130.130.64.64, not the actual virindi server). The steps are illustrated here; you will simply download this project. In a later assignment, you will start a project and then place it under version control in much the same way.

I “imported” my NetBeans project into subversion:



Creating a new project, or updating a project, will require a login with user-name and password (these will be supplied to groups for the group assignment). I was using my repository; this is configured to allow public downloads.

Import

Steps

1. **Subversion Repository**
2. Repository Folder
3. Files to Import

Subversion Repository

Specify the location of Subversion repository.

Repository URL: http://hostname/repository_path[@R...

User: (leave blank for anonymous)

Password:

☐ Save Username and Password

[Proxy Configuration...](#)

Steps

1. Subversion Repository
2. **Repository Folder**
3. Files to Import

Repository Folder

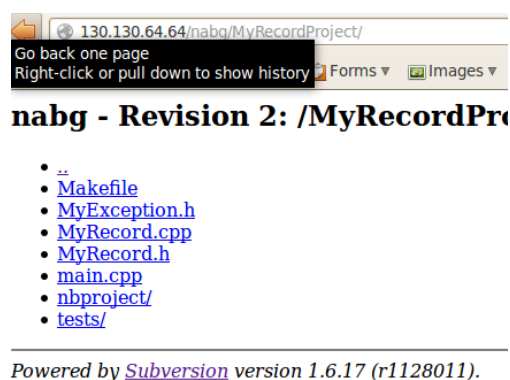
Specify the Repository Folder you want to Import into

Repository Folder:

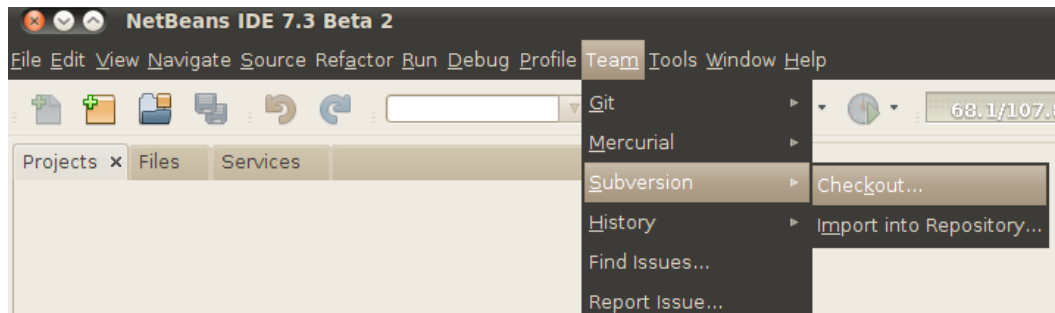
Specify the Message:

Create the MyRecordProject starting point.

The files are in the repository for you to download:



Start a new NetBeans session, and use Team/Subversion/Checkout to get your copy of those initial files:



Steps

1. Subversion Repository
2. Folders to Checkout

Subversion Repository

Specify the location of Subversion repository.

Repository URL:

http://hostname/repository_path

User: (leave blank for anonymous login)

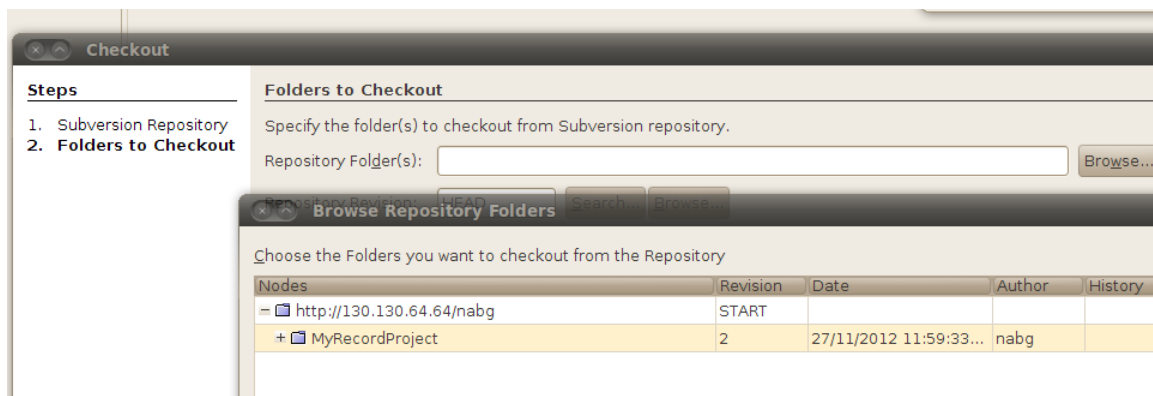
Password:

☐ Save Username and Password

(The server 130.130.64.64 was a temporary machine. Your laboratory supervisor will tell you the URL of the repository for the practical classes – probably <http://virindi.cs.uow.edu.au>. The Repository URL will therefore be <http://virindi.cs.uow.edu.au/nabg>.)

The /nabg repository should permit anonymous login for checkouts; so leave user and password blank.

Browse the repository to find the folder to checkout:



NetBeans will checkout the files, recognise that they constitute a NetBeans project, place this project in a directory that you select (option in the dialog). Create a new directory to hold the project and an associated hidden directory that is used for subversions own data. (The files in the repository were built with an older version of NetBeans; at some point, NetBeans may ask if you want the project data updated – *accept*.)

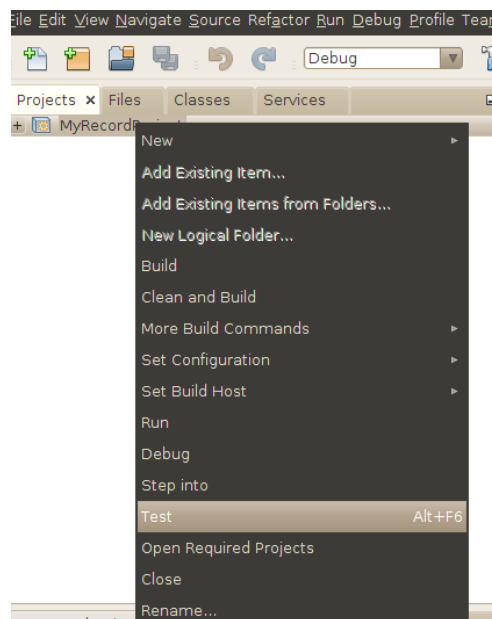
```

nabg@info-bxsrr1s:~/Teaching/2013Preparation/CSCI222/SVNDemo2$ ls -la
total 16
drwxr-xr-x  4 nabg nabg 4096 2012-11-27 13:28 .
drwxr-xr-x 18 nabg nabg 4096 2012-11-27 13:28 ..
drwxr-xr-x  4 nabg nabg 4096 2012-11-27 13:28 MyRecordProject
drwxr-xr-x  4 nabg nabg 4096 2012-11-27 13:28 .svn
nabg@info-bxsrr1s:~/Teaching/2013Preparation/CSCI222/SVNDemo2$ ls -la .svn
total 72
drwxr-xr-x  4 nabg nabg  4096 2012-11-27 13:28 .
drwxr-xr-x  4 nabg nabg  4096 2012-11-27 13:28 ..
-rw-r--r--  1 nabg nabg     2 2012-11-27 13:28 entries
-rw-r--r--  1 nabg nabg     2 2012-11-27 13:28 format
drwxr-xr-x 21 nabg nabg  4096 2012-11-27 13:28 pristine
drwxr-xr-x  2 nabg nabg  4096 2012-11-27 13:28 tmp
-rw-r--r--  1 nabg nabg 47104 2012-11-27 13:28 wc.db
nabg@info-bxsrr1s:~/Teaching/2013Preparation/CSCI222/SVNDemo2$

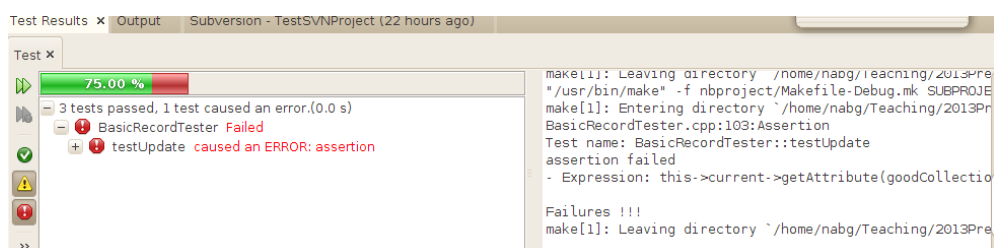
```

(Avoid having multiple version managed projects in the same directory; the subversion specific data tend to get confused.)

Run the unit tests that have already been built into the project:



Oops – a failure!



The MyRecord class needs to store collections of (key, value) pairs for things like phones, addresses, and other general properties. I chose to use an STL map; its a map<string, string>.

```
private:
    string id; // Also known as "nickname", or even "primary key"
    string name; // full name
    string email;
    string image;
    string info;
    vector<string> roles;
    map<string,string> phones;
    map<string,string> addresses;
    map<string,string> other;
```

The application must allow new (key, value) pairs to be added, and allow the overwriting of a value when necessary. The test that failed was the part of the unit test code that checked that data could be overwritten.

```
void BasicRecordTester::testUpdate() {
    string goodCollection = "Other";

    string val1 = "Pizza";
    string key2 = "Eax drink";
    string val2 = "Red bull";

    this->current->addKeyValue(goodCollection, key2, val2); // Should add value

    CPPUNIT_ASSERT(this->current->getAttribute(goodCollection, key2) == val2); // Check likes Red Bull

    // Actually prefers 4X
    string valnew = "4X";
    this->current->addKeyValue(goodCollection, key2, valnew);

    CPPUNIT_ASSERT(this->current->getAttribute(goodCollection, key2) == valnew); // Check now likes 4X
}
```

It seems that the value is not being changed.

The implementation code for the addKeyValue() method is:

```

void MyRecord::addKeyValue(string& collectionname, string& key, string& value) throw (MyException) {
    if (key.empty() || value.empty()) {
        string msg = "Empty key or value";
        throw MyException(msg);
    }
    // If same collection/key combination specified more than once, later value
    // overwrites earlier value for that key.

    if (collectionname == "Phones") {
        this->phones.insert(pair<string,string> (key, value));
    } else
        if (collectionname == "Addresses") {
            this->addresses.insert(pair<string,string> (key, value));
        } else
            if (collectionname == "Other") {
                this->other.insert(pair<string,string> (key, value));
            } else {
                string errmsg = "There is no collection called " + collectionname;
                throw MyException(errmsg);
            }
    }
}

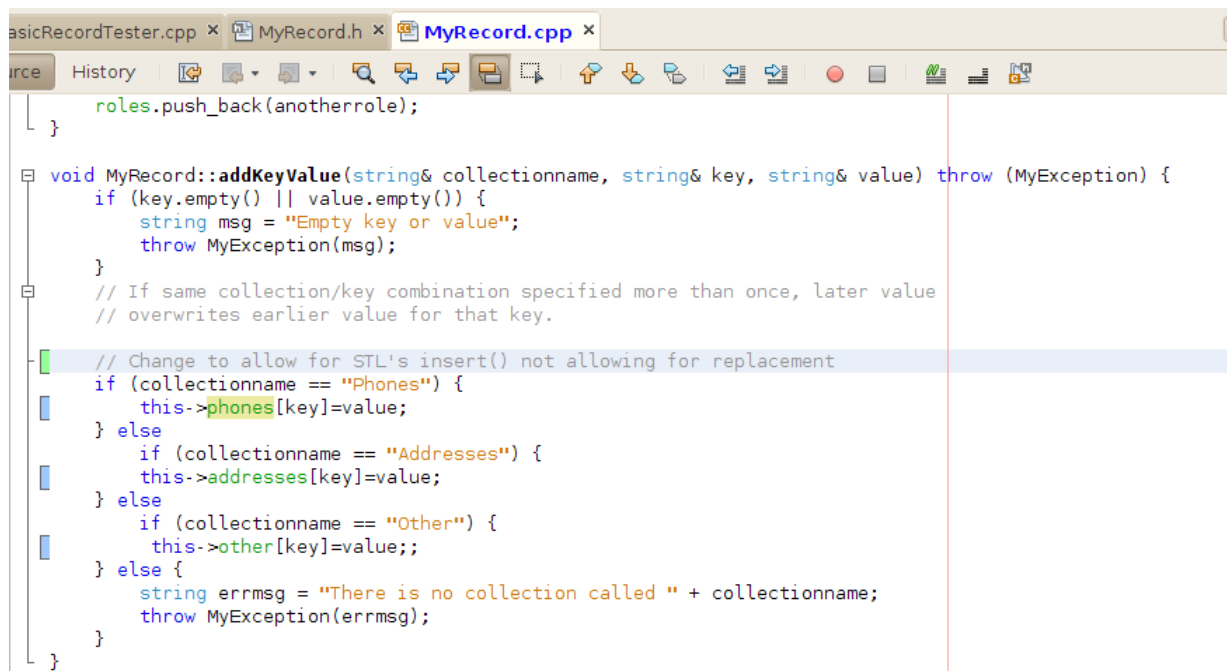
```

Larry Wall (the Perl guy) did say that the three main qualities of a programmer were laziness, impatience, and hubris. This is a typical instance. There are numerous collection classes in the libraries for different languages that represent some key=>value map. Most of these collection classes have a method (“put”, “insert”, ...) that allows you to insert a (key, value) pair or replace the value for an existing key. I simply assumed that STL's map worked in the same way (impatience), I didn't bother to check the documentation (laziness), and assumed that the code I wrote would be correct (hubris). I had just been using QMap from the C++ qt libraries and insert() worked fine there. But STL differs from all the other libraries. STL's insert() only inserts values; it doesn't replace a value of a key that is present. Further, STL doesn't have a replace() operation.

So how do you change values for an STL map?

You have to use the overloaded operator[] function.

The code for addKeyValue() needs to be changed systematically -

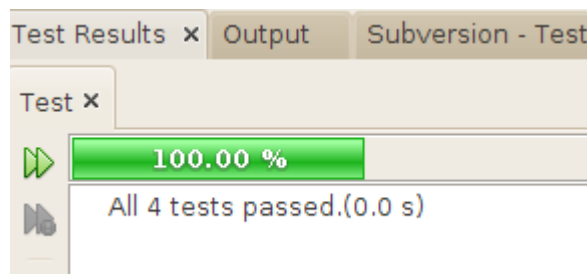


```
roles.push_back(anotherrole);
}

void MyRecord::addKeyValue(string& collectionname, string& key, string& value) throw (MyException) {
    if (key.empty() || value.empty()) {
        string msg = "Empty key or value";
        throw MyException(msg);
    }
    // If same collection/key combination specified more than once, later value
    // overwrites earlier value for that key.
    // Change to allow for STL's insert() not allowing for replacement
    if (collectionname == "Phones") {
        this->phones[key]=value;
    } else
        if (collectionname == "Addresses") {
            this->addresses[key]=value;
        } else
            if (collectionname == "Other") {
                this->other[key]=value;;
            } else {
                string errormsg = "There is no collection called " + collectionname;
                throw MyException(errormsg);
            }
}
```

Note how NetBeans uses special colour highlighting to identify changes to a file that is under version control. The blue highlights identify code that has been changed; green highlights mark inserted lines; while red identifies deletions.

Make the changes, and re-run the tests:



Using libraries

The MyRecord class now “works correctly”. Well, maybe not.

```
void MyRecord::setEmail(string& email) throw (MyException) {
    this->email = email;
}
```

It seems that we are happy to accept any old string as an email address; but that's just not right.

The class should validate the data for operations like setEmail(), checking that the string represents a plausible email address. How to check emails? Use regex tests of course. (Maybe you should do more and try making a SMTP connection to the mail handler referenced in the email; but that's going to extremes.)

There are regex test functions in the old C libraries – but they are very clumsy.

The “boost” C++ libraries provide for better regex handling. So, let us try using these.

Trim the string

We will first need to trim any leading or trailing spaces from an input string, and then check that the remainder looks like an email address. We will need a regex expression to match email addresses – can ask Google to find that.

STL's string doesn't have a trim() method! Ask Google again; Google says use Evan Teran's code. His code consists of inline functions – so define these in an extra C++ header file (stringtrimmer.h) that is added to the headers folder of your MyRecordProject.

```
#ifndef STRINGTRIMMER_H
#define STRINGTRIMMER_H

#include <algorithm>
#include <functional>
#include <cctype>
#include <locale>

// trim from start
static inline std::string &ltrim(std::string &s) {
    s.erase(s.begin(), std::find_if(s.begin(), s.end(),
        std::not1(std::ptr_fun<int, int>(std::isspace))));
    return s;
}

// trim from end
static inline std::string &rtrim(std::string &s) {
    s.erase(std::find_if(s.rbegin(), s.rend(),
        std::not1(std::ptr_fun<int, int>(std::isspace))).base(), s.end());
    return s;
}

// trim from both ends
static inline std::string &trim(std::string &s) {
    return ltrim(rtrim(s));
}
```

regex for email

Google suggested that the following constructor be used to create a boost library regex checker for emails:

```
"^[a-z0-9_\\+\\-]+(\\.[a-z0-9_\\+\\-]+)*@[a-z0-9-]+(\\.[a-z0-9-]+)*\\.([a-z]{2,4})$"
```

The `boost/regex.hpp` standard header file should be `#included` in the `MyRecord.cpp` file, and `using namespace boost` should be specified; then code should be added to `MyRecord::setEmail()`

- Use the `boost::regex` constructor to create a regex object “validationExpression” (an instance of boost's regex class called `validationExpression` with that string as initialization parameter);
- Trim the argument string for the function;
- Use the boost library's `regex_match()` function to test whether the trimmed string matches the validation expression for an email; (`regex_match` takes the string and the regex variable as arguments)
if it doesn't match, throw a `MyException` with suitable error message; (if in doubt, ask Google to find you a boost/regex example);
You can ask google for help with `boost::regex` – lots of examples are out there.
- If the trimmed string passes validation, assign it to the instance member variable for email.

An additional test function, “testEmailValidation”, should be added to the `BasicRecordTester`. This should verify that the email strings are trimmed and that invalid email addresses cause exceptions.

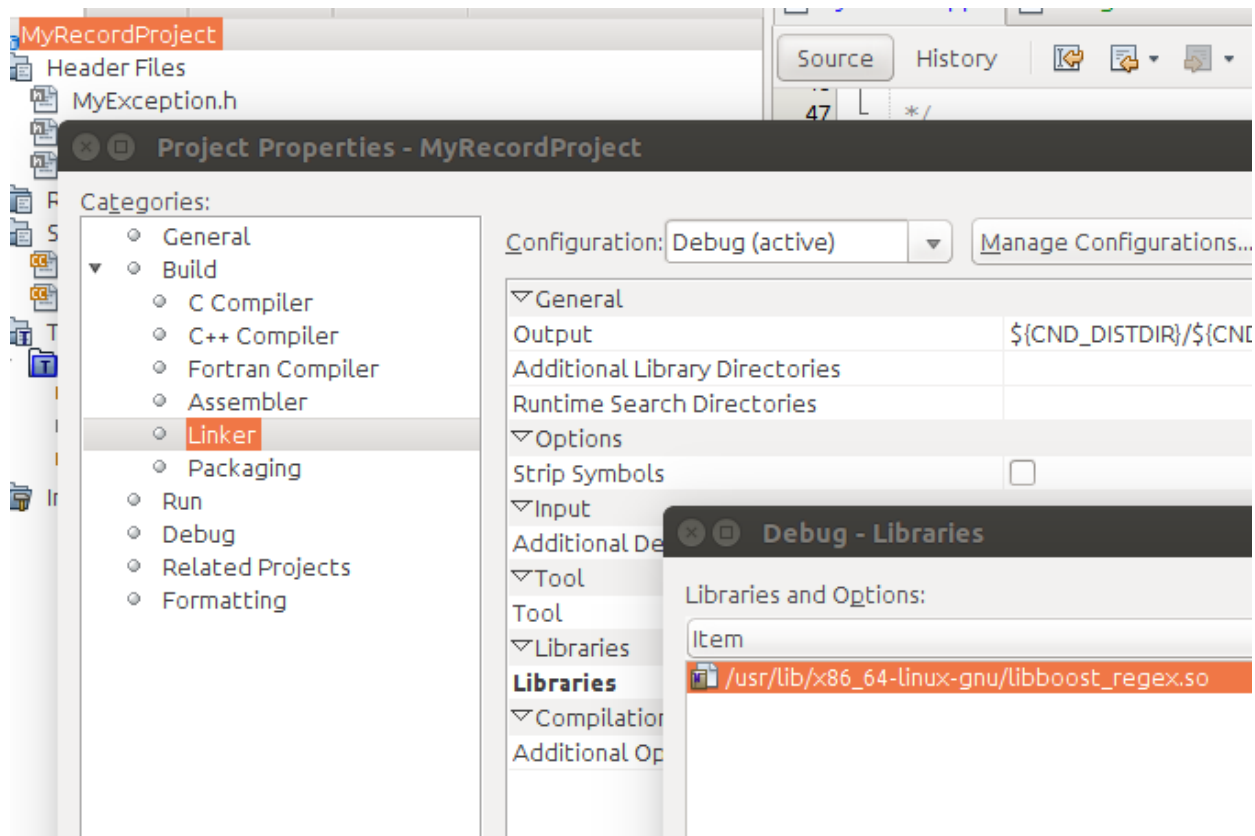
(No, I haven't spelled out the code for you. You should be able to do it yourself! In total, it's less than 10 lines of code for the test function and the modified `setEmail()` function.)

Includes and linkage directives

In this case, you don't need to add any extra include paths for the project's compilation options. The boost libraries are standard and are normally placed in `/usr/include` which is on the standard include path for C++ projects. A “`#include <boost/regex.hpp>`” directive will enable the compiler to find the headers.

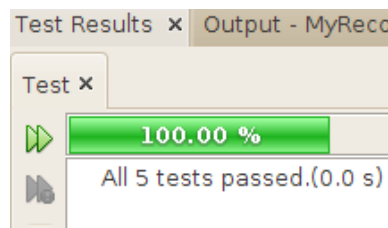
However, you do need to add the actual boost link libraries. You should have learnt how to write makefiles in CSCI204 and so in principal you should be able to add the necessary elements to the makefile used. But with an IDE like NetBeans, there are “wizards” that make all such configuration steps a lot easier. Select the `MyRecordProject`, right-click select “Properties” and edit the linker properties.

In the Linker/Libraries option, you need to add the `/usr/lib/x86_64-linux-gnu/lib_boost.regex.so` link library file. Use “Add Library File”. A file dialog will be displayed for you to find this file – there are an awful lot of libraries in `/usr/lib` so you will have to scroll a bit to find the `x86-64-linux-gnu` subdirectory; there are an awful lot of libraries in the 64-bit linux collection so you will again have to scroll down to find the `lib_boost` section.



Test

When you have correctly set up the link libraries, you should be able to run the cppUnit test on the enhanced MyRecord implementation:



Task 3 – completion (2 marks)

You will need to show the code that you added to a downloaded copy of the MyRecord project and demonstrate that it now handles validation of emails.

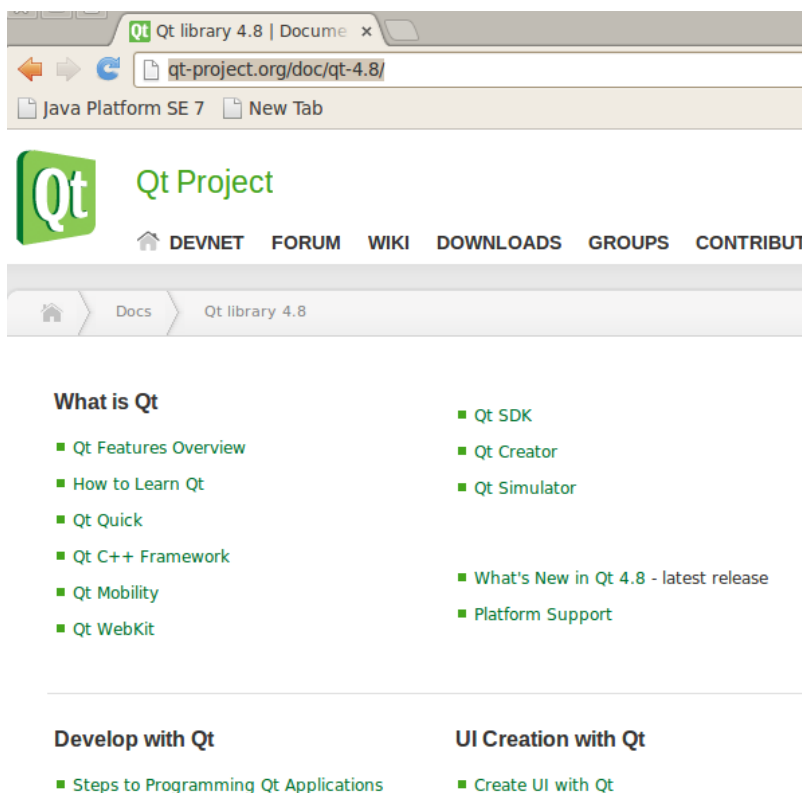
Task 4: Further use of libraries

This trivial task provides more experience with the use of C++ libraries; in this task, you will be using code from the Qt C++ graphics library. In a later exercise, you will actually build graphical interfaces for C++ programs using Qt. Here, it is just a matter of using some classes for image processing.

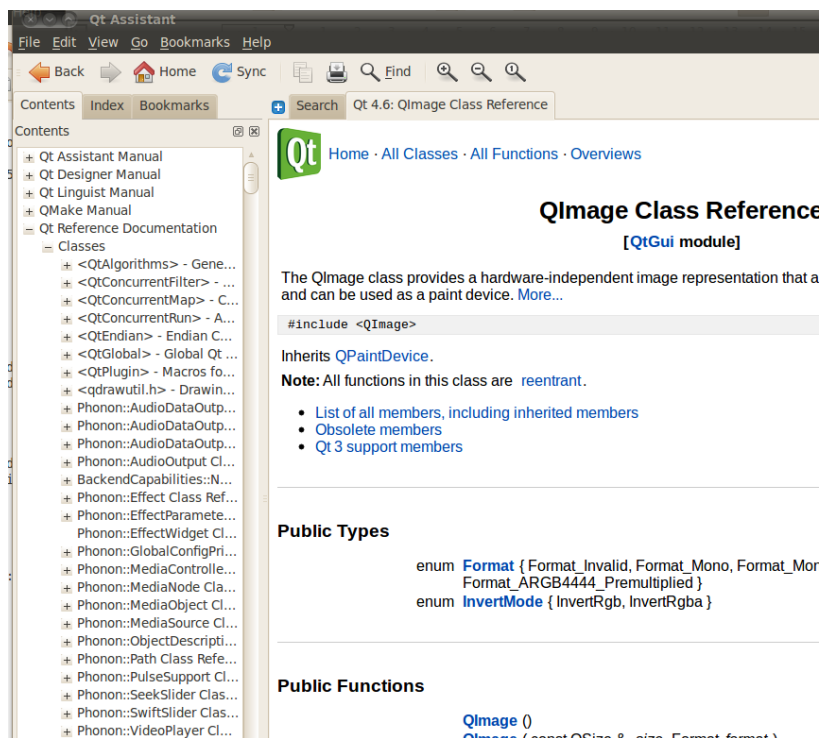
This task involves running a little application that reads in an image file (png, jpg, gif, ...), scales the image down to thumbnail size, converts the thumbnail image to jpg format, and finally encodes the jpg data as a hex text string. (Later, this code will be incorporated into the address book application where it will be used to get the text format for an image that is to be stored in a MyRecord instance).

This exercise is simple – largely cut-and-paste. Its purpose is to illustrate slightly more complex cases involving setting include paths and including library link files. The code is supplied, and it largely relies on classes from the well-debugged Qt libraries, so there is no “unit test” element.

The Qt libraries are extensively documented. The documentation is available on the Internet at <http://qt-project.org/doc/qt-4.8/>. (Qt is now on version 5. Version 5 adds lots of features to support touch screen interfaces along with some other extensions. Version 4.8 is still the “standard” for Ubuntu.)



The laboratory computers have QtAssistant installed; this provides a local copy of the documentation along with extensive tutorials and other reference information.



Start a new NetBeans C++ project - "QtImages".

Edit the main.cpp file:

```

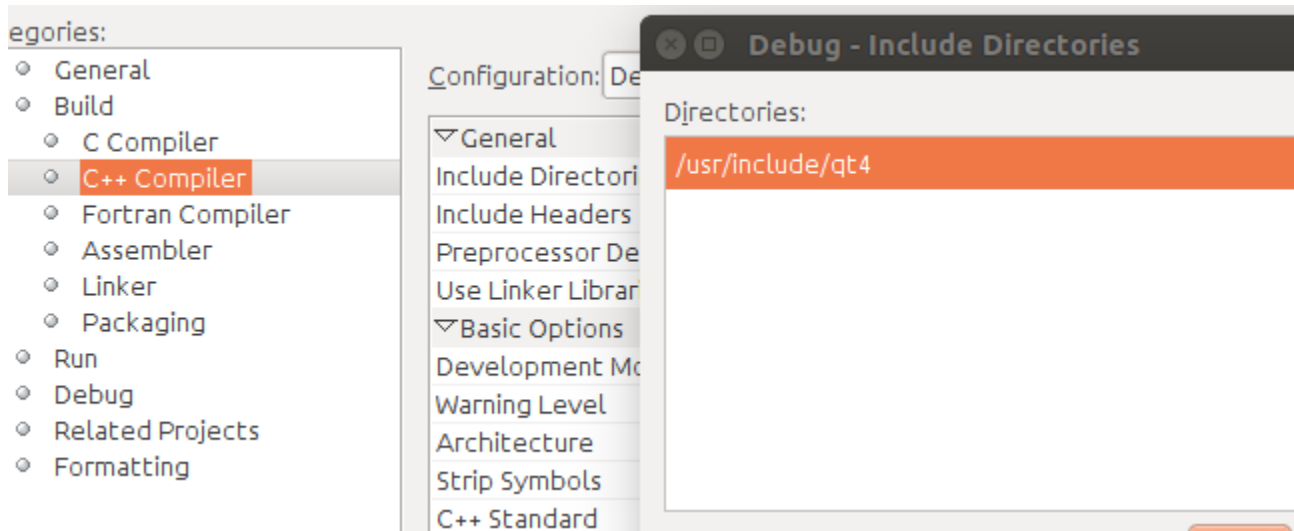
1  /
2
3  8  #include <cstdlib>
4  9  #include <iostream>
5  10 #include <qt4/Qt/qimage.h>
6  11 #include <qt4/Qt/qstring.h>
7  12 #include <qt4/Qt/qbuffer.h>
8  13 #include <qt4/Qt/qbytearray.h>
9  14 using namespace std;
10 15
11 16 /*
12 17  *
13 18  */
14 19 int main(int argc, char** argv) {
15 20     cout << "Enter name of an image file : ";
16 21     string filename;
17 22     cin >> filename;
18 23     // Qt library has its own string class, convert STL string
19 24     QString qtfilename(filename.c_str());
20 25     QImage animage;
21 26     bool readimage = animage.load(qtfilename);
22 27     if(!readimage) {
23 28         cout << "Image load failed. Bye" << endl;
24 29         exit(1);
25 30     }
26 31     cout << "Image loaded" << endl;
27 32     return 0;
28 33 }

```

The #include entries have added the headers needed for QString and QImage (along with headers for a buffer class, and byte array class that will also be needed), but there are problems.

The problems result from header files such as `qimage.h` including other qt header files. These other header files will be down somewhere inside `/usr/include/qt`, but not in places that will automatically get checked.

The project's "compiler/include" properties must be edited to add the necessary qt directories to the places where it looks for other referenced header files. Edit the project properties and add the directory `/usr/include/qt4` to the compiler's include files:



While editing the properties, add `/usr/lib/x86_64-linux-gnu/libQtCore.so` and `/usr/lib/x86_64-linux-gnu/libQTGui.so` to the linker's library files.

Copy an image file (any format, Qt reads them all) into your project's directory, storing it as `default.jpg` (or `default.png` or whatever).

The program fragment should happily load any image file that you give it. An instance of Qt's class `Image` will typically hold the image in an array of 4 byte integers (for the red, green, and blue colours, and the alpha transparency) though there are other options for things like grey-scale images. If you were writing an image analysis program (e.g. a face recognition program), your code could access these pixel data.

Complete the program required here:

- Create another `QImage` by scaling down the image read in; the original image should be scaled to a width of 50 pixels (with the height being scaled proportionately);
- Get the `QSize` dimensions of the two images and print these values to standard output;
- Saves the resized image to the file "scaledimage.jpg" (doesn't matter what file format was read, the scaled image should be saved in JPEG format);
- Create a `QByteArray`, and `QBuffer` that uses the byte array; save the image again, this time saving the data in the buffer (explicitly defining the output format as JPEG);
- Create another `QByteArray` with the data converted to base64 encoding.
- Output the base64 encoded image as text.

(You can easily find all the methods required by looking at the documentation of Qt classes `QImage` and `QByteArray` in `QtAssistant`.) Your program should run:

```

36     QSize szr = resized.size();
37     cout << "Resized image height " << szr.height
38     cout << "Trying to save image " << endl;
39     QString qtoutfilename("./scaledimage.jpg");
40     bool writeok = resized.save(qtoutfilename);
41     if (!writeok) cout << "File save failed!" << endl;
42     QByteArray ba;
43     QBuffer buf(&ba);
44     animage.save(&buf, "JPG");
45     cout << "Written image in JPG format to a byte
46
47     QByteArray hexed = ba.toBase64();
48     cout << "convert to base64 data" << endl;
49

```

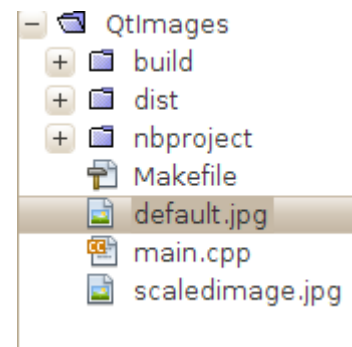
Output x

QtImages (Build, Run) x QtImages (Run) x

```

> Enter name of an image file : default.jpg
> Image loaded
> Image height 425, width 300
> Trying to resize image
> Resized image height 71, width 50
> Trying to save image
> Written image in JPG format to a byte array
> convert to base64 data
> convert base64 data to string
/9j/4AAQSkZJRgABAQEASwBLAAD/2wBDAAgGBgcGBQgHBwcJCQgK
> convert base64 data to string
/9j/4AAQSkZJRgABAQEASwBLAAD/2wBDAAgGBgcGBQgHBwcJCQgK

```



Dimensions: 50 x 71

Dimensions: 300 x 425

(The Qt image code depends on other libraries for some of its image file decoding and encoding; if you find that your program only works for a subset of image file types then you probably don't have all the image libraries in your linux installation.)

Task 4 – completion (1 mark)

Demonstrate that the Qt based application compiles, links, and runs correctly.

Task 5: Simple address book application

You can make this task a continuation of your MyRecord project or start a new project and cut-and-paste your working code. (In principal, it would be better to continue in the same project as then the existing unit test code would also carry over. In practice, better to have a new project because you don't have individual subversion repositories to store the working version of your MyRecordProject which you would need if you were asked to demonstrate it later.)

NetBeans has “copy” and “paste” options in its project's pane. So you can copy a project, and paste it with a new name. This feature works fine for Java, PHP and other projects. **DO NOT USE WITH C++ (or C) PROJECTS!** NetBeans has some very confusing mechanisms for “sharing” header files between projects; you are likely to find that you are building with a mix of updated .cpp files and the old header files. (This can also happen if you copy a project folder in the shell.)

The problems are due to the NetBeans own project data having absolute path names for files. You want to use absolute path names when referencing libraries, but want relative paths - “the header file in this directory” - for project files. If you ever end up with bizarre compilation errors such as the compiler insisting that you haven't declared a member function in a class when you can see the declaration, then you are probably having a problem with the project referencing an older header file in another project.

If such problems do occur, it is possible (though tedious) to fix things by editing the nbproject/configurations.xml file so that relative names are used for project files and absolute paths for libraries.

So **new** C++ application project – SimpleAddressBook.

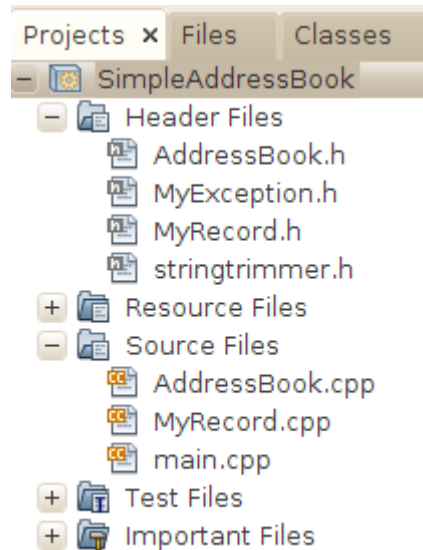
The SimpleAddressBook application:

- A “menu-select” program using text inputs and outputs (you probably implemented programs like this in CSCI114, and CSCI124, and again in CSCI204);
- It allows a user to create an in-memory collection of MyRecord objects (using an STL map – `map<string, MyRecord>`); in this version, the data are not persistent – they are lost when the program terminates;
- Its top-level menu has options Quit; Add Record; List Records; Display Record;
- The Add Record option will lead to a sequence of prompts for data that are to be entered into a new instance of class MyRecord; the completed record is then added to the program's collection (the record identifier serves also as the key for the collection);
- The List Record option simply lists all the keys (identifiers) for records in the collection;
- The Display Record option ask for a record identifier, and if the input value is valid displays details of that record.

The program does not use a bare STL map, instead this is packaged in a simple AddressBook class.

(If you need help with STL, or any other aspect of C++, try the www.cplusplus.com web site, e.g. <http://www.cplusplus.com/reference/map/map/find/>)

The project:



(Remember to add the boost regex code to the libraries for the project.)

You will need to define a minimal “AddressBook” class along the following lines:

```
class MyRecord;

typedef MyRecord* RecordPtr;

class AddressBook {
public:
    AddressBook();

    ~AddressBook(); // No provision for subclassing
    void add(RecordPtr prec);
    const map<string, RecordPtr>& viewData() const { return this->entries; }
    bool keyExists(string& id) const;
    RecordPtr getRecord(string& id) const;
private:
    map<string, RecordPtr> entries;
```

Methods like keyExists() should use functions like STL map::find or map::count.

You need a simple “menu-select” mainline (details of the auxiliary functions are left for you to fill in – practice your hard-earned CSCI114 skills):

```

11 | #include "AddressBook.h"
12 | #include "MyRecord.h"
13 | #include "stringtrimmer.h"
14 |
15 | using namespace std;
16 |
17 | static string getTrimmedString() {...}
25 |
26 |
27 | static void doAddRecord(AddressBook& abook) {...}
78 |
79 | static void doListIds(const AddressBook& abook) {...}
93 |
94 | static void doDisplayRecord(const AddressBook& abook) {...}
114 |
115 | static void consumeNewLine() {...}
122 |
123 | int main(int argc, char** argv) {
124 |     AddressBook mybook;
125 |     cout << "Demo address book application";
126 |     bool continuing = true;
127 |     while(continuing) {
128 |         cout << "Menu selection\n1 Quit\n2 Add record\n3 List identifiers\n4 Display record\n";
129 |         cout << "Select : ";
130 |         int choice;
131 |         cin >> choice;
132 |         if(!cin.good()) break;
133 |         consumeNewLine();
134 |         switch(choice) {
135 |             case 1: continuing = false; break;
136 |             case 2: doAddRecord(mybook); break;
137 |             case 3: doListIds(mybook); break;
138 |             case 4: doDisplayRecord(mybook); break;
139 |         }
140 |     }
141 |     return 0;
142 | }
143 |

```

The “add record” function does not need to get data for all the fields in a MyRecord (console input is so tiresome). Values for id, name, email, Phone/Mobile, and some roles should suffice.

A run of your program should produce a console log like the following:

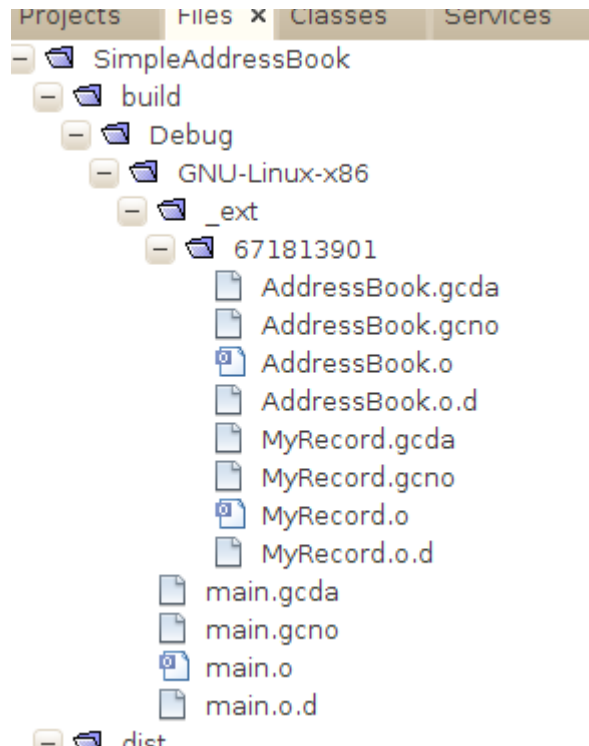
SimpleAddressBook (Build, Run)	SimpleAddressBook (Run)
Demo address book application	Record Dick added
Menu selection	Menu selection
1 Quit	1 Quit
2 Add record	2 Add record
3 List identifiers	3 List identifiers
4 Display record	4 Display record
Select : 2	Select : 3
Enter record identifier : Tom	Dick
Person name :Thomas Hardy	Tom
Email : tom@work.com	Menu selection
Mobile phone number : 0412378967	1 Quit
Enter roles, terminate with blank entry	2 Add record
DB-admin	3 List identifiers
Sys-admin	4 Display record
SQL wizard	Select : 4
Record Tom added	Enter record identifier : Dick
Menu selection	Record Dick
1 Quit	Richard Wagner
2 Add record	Mobile 04568989792
3 List identifiers	Roles
4 Display record	C++ guru
Select : 2	STL expert
Enter record identifier : Dick	Boost library wizard
Person name :Richard Wagner	Menu selection
Email : dickie@work.com	1 Quit
Mobile phone number : 04568989792	2 Add record
Enter roles, terminate with blank entry	3 List identifiers
C++ guru	4 Display record
STL expert	Select : 1
Boost library wizard	
Record Dick added	
Menu selection	
	RUN FINISHED; exit value 0; real time: 1m 57s; user:

Continue with a “code coverage” test. Change the project properties so that code coverage is in place, and run the program two or three times remembering to test things like the handling of invalid email identifies. Produce coverage listings for your main.cpp and for the two class .cpp files. (Some parts of your MyRecord code won't get executed because not setting properties like “addresses”, “info”, “image”.)

Part of my main.cpp.gcov

```
2: 81:
2: 82:static void doListIds(const AddressBook& abook) {
2: 83:    if(abook.viewData().size()<1) {
1: 84:        cout << "There are no records to list!" << endl;
1: 85:        return;
-: 86:    }
1: 87:    map<string,RecordPtr>::const_iterator it;
-: 88:    // Most collection classes for things like map can give you a vector
-: 89:    // with the keys; STL just has to be different and use this iterator
-: 90:    // idiom.
3: 91:    for(it=abook.viewData().begin(); it!=abook.viewData().end(); it++) {
2: 92:        string key = it->first;
2: 93:        cout << key << endl;
-: 94:    }
-: 95:}
-: 96:
4: 97:static void doDisplayRecord(const AddressBook& abook) {
4: 98:    cout << "Enter record identifier : ";
4: 99:    string id = getTrimmedString();
4: 100:    if(!abook.keyExists(id)) {
2: 101:        cout << "You don't have a record with that identifier" << endl;
-: 102:        return;
-: 103:    }
2: 104:    RecordPtr p = abook.getRecord(id);
```

NetBeans arranges that the data files for gcov for classes like MyRecord and AddressBook go in a subdirectory of the build/Debug/GNU-Linux-x86 directory:



So the command to get the analysis run will be something like:

```
$ gcov -o build/Debug/GNU-Linux-x86/_ext/671813901/ -c MyRecord.cpp
```

Part of my MyRecord.cpp.gcov

```

-: 53:
2 3: 54: void MyRecord::addKeyValue(string& collectionname, string& key, string& value) throw
3 3: 55:     if (key.empty() || value.empty()) {
#####: 56:         string msg = "Empty key or value";
#####: 57:         throw MyException(msg);
-: 58:     }
-: 59:     // If same collection/key combination specified more than once, later value
-: 60:     // overwrites earlier value for that key.
-: 61:
-: 62:     // Change to allow for STL's insert() not allowing for replacement
11 3: 63:     if (collectionname == "Phones") {
12 3: 64:         this->phones[key]=value;
13 -: 65:     } else
#####: 66:         if (collectionname == "Addresses") {
#####: 67:             this->addresses[key]=value;

```

The ##### marks identify blocks not executed in the recent tests.

Task 5 – completion (3 marks)

Demonstrate your working SimpleAddressBook project.

Finally!

When you have finished all your demonstrations to lab tutors, clean out the projects. The compiled code and linked executables are consuming lots of your limited disk space.

In each project, use right-click “More Build Commands/Clean”.