# SAPO 2012 environment manual

SACO Scientific Committee

September 2012

## Contents

# 1 Introduction

This document describes the environment you will be using for the final round of the South African Programming Olympiad. Section 2 describes the machines you will be using, where to save files etc, without dealing with programming. Section 3 describes the compilers and development environments. You will also be using a web-based handin system, described in section 4. Specific issues related to the interaction between your programming environment and the web server may be found in section 5.

# 2 Local environment

You will be using an Ubuntu Linux workstation attached to the UCT network. Before each competition round the contest organisers will log you into a workstation.

If technical difficulties occur, you may have to move to another workstation. Because of this, you should not store any important files anywhere except your home directory. This directory is called /home/username and is linked to the UCT fileserver. Here username would be the username that your were logged on with. You may get the path of you current directory by running 'pwd' in a terminal. **You should save all your work often to the home directory.**

You should also not reboot the computer unless it is absolutely necessary. If you do need to restart your machine during the contest, ask an organiser to log you into another workstation.

There will be shortcuts to the various tools in Applications Menu and on the Desktop. There will be a shortcut set up on the Desktop for the contest webpage.

# 3 Compilers and IDEs

## 3.1 C and C++

For C and C++, you will be using GCC 4.4. This version of GCC adheres particularly strictly to the C++ specification. Some things that may work in other compilers that will not work under GCC include:

- `main` *must* return `int`, and *must* return 0. This might not be enforced by gcc, but failure to do so can lead the marking system to think that your program crashed (in which case you score 0).
- `stricmp` does not exist; use `strcasecmp` instead
- `strrev` does not exist; write one yourself
- Newer versions of the C++ standard introduce newer versions of the standard C++ headers like `iostream`. The new versions have no `.h` extension, and place all their defines in the `std` namespace. The old versions are deprecated and gcc will give you an error if you try to use them. The end result is that the hello world program now appears as follows:

```
#include <iostream>
using namespace std;
int main()
```

```
{
    cout << "Hello world\n";
    return 0;
}
```

Note that using the `endl` operator in an I/O stream causes the stream to be flushed, which has a performance overhead. It is faster to use `"\n"` instead.

The supported IDE for C/C++ is Code::Blocks. This is a Linux-based IDE that supports syntax highlighting and auto-completion. You may also use an available text editor such as gedit or kate to write your code in and compile it yourself from a terminal using the GCC compiler.

### 3.2 Java

We will be using version 1.6 of the Sun Java Development Kit. You should be aware that while the `Scanner` class introduced in 1.5 makes I/O easier, it is roughly 10 times slower than using a `BufferedReader` together with a `StringTokenizer`. For tasks with large amounts of input, it may not be possible to obtain a full score using `Scanner`.

The supported IDE is Eclipse. When Eclipse starts, you will be presented with a welcome screen, which you can close. Select File → New → Project, then Java project. For the project name, use the short name of the problem. You should also change the Java compliance level from 1.4 to 6.0. Once you have created a project, you can add a file by selected File → New → Class.

### 3.3 Python

Python version 2.6.5 will be used both in the competition environment and the Linux-based handin system. The supported IDE is IDLE.

## 4 Web-based handin system

The South African Programming Olympiad uses a web-based handin system, similar to the one used at the International Olympiad in Informatics. The web server can be accessed via the link found on the desktop. This page has links to the handin system itself, as well as various documentation. You should use Firefox with the system, as other web browsers may have several bugs that prevent it from working properly. The first time you connect, you will be warned that the certificate is self-signed; select to accept the certificate permanently and continue.

You will be allocated a username (initial and then surname – if your name if Raymond Luxury-Yacht, it will be rluxuryyacht) and a password for accessing this system. Once you have logged in, there are a number of menu options:

**Problems** This takes you to a page with some information about the problems available; in particular, the input data for output-only problems and the resource limits for other problems.

**Submit** This page allows you to hand in solutions, as well as to view files that you have handed in. The process is slightly different, depending on whether you are submitting source code or an output file (the problem will determine which must be handed in). This page will also allow you to view a log of all submissions for each problem. In order to view these submissions, you need to click the 'view all' link. From there, you can view each of your submissions.

When you hand in source code as a solution, your source code will be compiled and run on a sample test case. If it successfully compiles and runs and scores 100% on the sample case, it will be saved and will overwrite any existing solution. This does not guarantee that you will score any marks on the real test data.

For some problems, detailed feedback may be enabled. When detailed feedback is enabled, will receive detailed feedback for any submission you make for the problem. When available, after the submission is accepted, it will be evaluated with some of the official test runs. You will then receive a summary of the evaluation results on the executed test runs. Problems which have detailed feedback enabled, will give you detailed feedback regardless of the number of submissions.

When handing in output files, make sure that you fill in the case number field. The file that you upload will be saved (replacing any existing answer for that case) if it passes the format check. Note that the format check is not exhaustive: it remains your responsibility to check that you are handing in the correct file with the correct case number, and that the formatting is correct. You can also submit a zip file containing multiple output files. The output files must be named `<probname><casenum>.out`; only correctly named files that pass the format check will be saved.

**Testing** You can use this page to see how your program will run on the server, on test data other than the sample test case. Be aware that the test page will *never* store the program you upload – if you want to test the same code multiple times you will need to choose the file multiple times. As a convenience, you can test your handed in version by not specifying a file.

**Marks** This option will only be available after the contest. You will be able to view your marks for all problems, as well as download a zip file containing detailed output files and results.

Once the contest is over another option will also appear on the test page: an option to run the tests on the real test data, and have the output marked.

**Logout** This is self-explanatory.

# 5 Miscellaneous

The handin system is running on Gentoo GNU/Linux. Because of the difference in operating systems, bugs in your program may manifest themselves in different ways on the handin system, perhaps not even appearing at all on the local system. Here are some possible reasons for these differences in behaviour.

**Memory limitations** When run on the server you are limited to a fixed amount of memory (usually 64MiB). If you try to use more than this, then

**C** `malloc` will return `NULL`

**C++** `new` will throw a `bad_alloc` exception

**Java** `new` will throw `java.lang.OutOfMemoryError`

**Python** Operations will throw `MemoryError` or just crash Python.

If you use up all the space with global variables in C/C++, your program will probably be killed before any of your code executes.

Note that there is some overhead for your code, stack and so on, so not all of the memory is available to you for variables. Count on losing at least 2–5MB to overheads. You will be given far more memory than you actually need, so overheads should not be a major issue.

Python is the worst culprit for memory consumption. Lists are very memory-inefficient and will consume several times as much memory as one would except. The `range` operator constructs a temporary list, so if you are looping over a large range, use the `xrange` operator instead.

**Memory accesses (C/C++)** If you have been use to programming C/C++ on Windows then you may find that Windows tends to be a lot more forgiving than Linux when it comes to memory accesses. In particular it will generally let you read from an invalid address. Sources of invalid addresses include NULL pointers, uninitialised pointers, pointers to memory that have been freed, and out of range array indices. If your program is crashing with signal `SIGSEGV` (known as a segmentation fault) on the server then this is a likely cause of the fault.

**Uninitialised memory (C/C++)** If you use a variable without initialising it, your program might behave differently depending on what is in that variable when it starts. You can often get the compiler to warn you of this by specifying the compiler flags `-O2 -Wall` (on GCC), although it is not perfect.

**Compiler flags** Read the rules to determine the compiler flags that are used on the server. You may find that the only reason the bug appears on the server and not your machine is that you were not using the same compiler flags. This is usually a sign that you have done something wrong, such as accessing memory out of bounds (where the flags might change the way memory is layed out) or not initialising a variable (without the optimisation flags the compilers may zero the memory for you, but will skip this when optimising).

**Java class names** Your source file is compiled as *probname*`.java` on the server, so the name of your main class must be *probname* in lowercase. This differs from the Java convention of starting class names with an uppercase letter.

**Exit code** A program that returns with a non-zero exit code is considered to have crashed. C/C++ programmers must declare their `main` function to return `int` and must explicitly return 0 at the end of the `main` function.

**Headers, units and modules** Some C/C++ headers and Python modules are specific to DOS or Windows. You may find that the server will complain about a missing header file — if so then this is probably the problem. A good example of this is the `<conio.h>` C header.

Some Java textbooks use the add-on package `java.lancs` to provide simple input routines. This package is not part of the Java standard and will not be available on the server. You should instead use `BufferedReader`. We can assist you with this at the practice round, but no assistance will be given during the contest.

### 5.1 Documentation

Documentation for all the languages may be found on the handin server using the link provided on the Desktop. This includes documentation on the C++ Standard Template Library.

## 6 Summary

**Handin server account** initial and surname

**Shortcuts** Desktop and Menu

**Home directory** `/home/username`