The Hitchhikers Guide to the G51PRG resit

Steven R. Bagley

ABSTRACT

This document outlines the G51PRG resit, giving details of the structure of the resit and some pointers about what to revise to get the best marks.

1. Structure

The G51PRG resit is a single, two-hour written exam. There are five questions on the exam paper of which three are to be answered. You are **required** to answer question one and then you may chose any two of the other four questions. You should aim to allocate your time in the exam to the weighting of the questions.

You will be asked to both read and write C code during the exam, but since you will not have access to your notes or books then we will be lenient when marking if the exact syntax is not quite right, e.g. if you used `print()` instead of `printf()` when writing some code, we would ignore this mistake. However, if the mistake means that it is no longer possible to see what you meant or produced something that was also valid C code (e.g. you wrote `if(x = 1)` instead of `if(x == 1)`) then you would be penalized.

2. Question One

Question one is worth 50 marks, and you should aim to spend around half of the exam answering this question. In this question, you will be presented with a small, but substantial, C program (such as the example at `http://g51prg.cs.nott.ac.uk/sampleCode.c`) and asked to write an explanation as to how this works. The program will be of similar length to your answer for the final coursework.

For this question, you should aim to explain the general reason why the program works, not just what each line does. You can assume that we understand C code, and so answers that just translate the C code into English will get low marks. Rather you should aim to explain the purpose of the program, what it does and how the different sections work together to form the whole program and why it is done that way rather than in any other way. The report will be graded on how clearly it explains the working of the program.

To illustrate the sort of explanation needed by this question consider this fragment of C code:

```
double a[] = {1.2, 3.5, 4.6, 7.5, 983, 2.4};
double sum = 0.0;

for(int i =0; i < 6; i++)
{
    sum = sum + a[i];
}
printf("Average is %d\n", sum / 6.0);
```

An answer explaining this code that would score low marks would read something like this:

> *This code creates an array of doubles. It then creates a variable called* `sum` *which is set to the value* `0.0`*. It then has a* `for` *loop that runs from* `0` *to* `5` *in the variable* `i`*. Inside the* `for` *loop it stores in the variable* `sum` *the result of adding element* `i` *from the array to the value of* `sum`*. After the for loop it prints out the value of* `sum` *divided by* `6.0`*.*

As you can see, the above answer just converts the C into English but offers no explanation of why these operations are done. A far better answer would read as follows:

> *This block of code is used to calculate the average of the values stored in the array* `a`*. The sum of all these values is calculated by using a* `for` *loop to iterate over each element in the array adding all the values together and storing the result in the variable* `sum`*. The average is calculated by dividing this total by the number of values in the array.*

This answer much more clearly explains what the purpose of the code is (calculate the average) and how this is achieved (by iterating over each item in the array). This example talks more about the strategic approach that code takes rather than the actual implementation details, however if there are some details of the implementation that you feel are particularly noteworthy or odd then you should comment on them.

3. The other questions

Each of the other questions on the exam are worth 25 marks each, and you should aim to spend around a quarter of the exam time answering them. These questions each look at a different area of the material covered by the course. The questions are all made of separate sub-parts which require a mixture of explanations of C operators and concepts, reading and explaining code and writing new code.

4. Revision

As can be seen from the above discussion, the exam is likely to cover material from across the length and breadth of the course. And so you should aim to revisit much of the course in your revision. One method that may help you to revise, alongside just reading the lecture notes, is to revisit the coursework exercises (particularly the final one), and to answer them afresh. This will give you practice at writing C code again. You may also like to see if you could extend them in some way. One east way to get started would be to attempt 'the other two' exercises from coursework 4 that you didn't previously do. We'll make the solutions to all three problems available later in the yar, as well as putting up some more example exercises.

In general terms, you should aim to have a working knowledge of the C operators, variables and their scope, procedures and functions, iteration and recursion, and pointers and memory management. It may also help to re-read the general feedback on the first three exercises, since they contain some hints'n'tips on how to write good software.

Finally, if you passed G51PRG but would like to brush up your general programming skills before the G51ISO (or any other programming-related) resit then you might like to try implementing some of your PRG exercises in Java — |his will let you practice your Java skills with a problem that you already understand the logic behind.

Don't Panic!