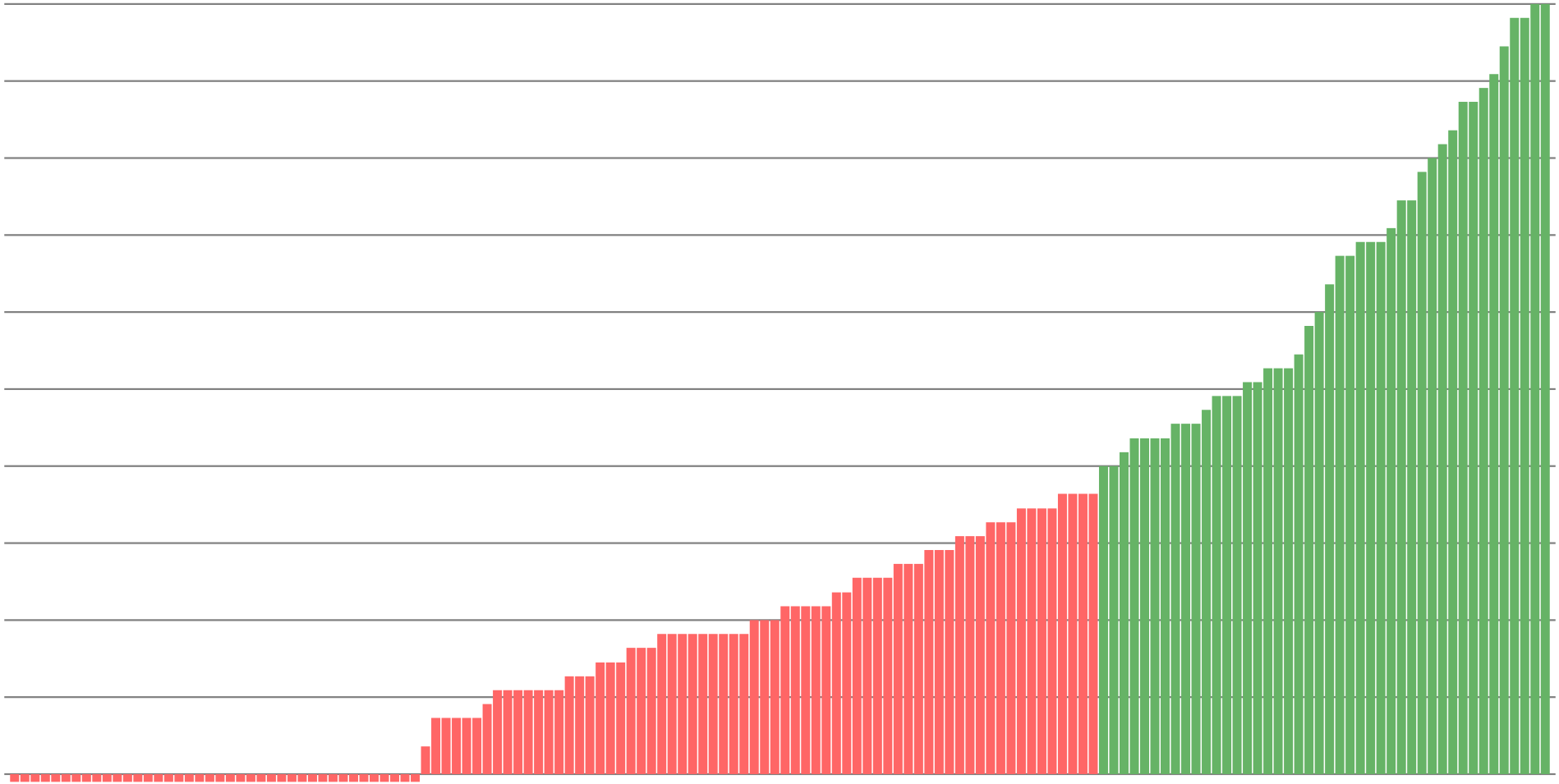
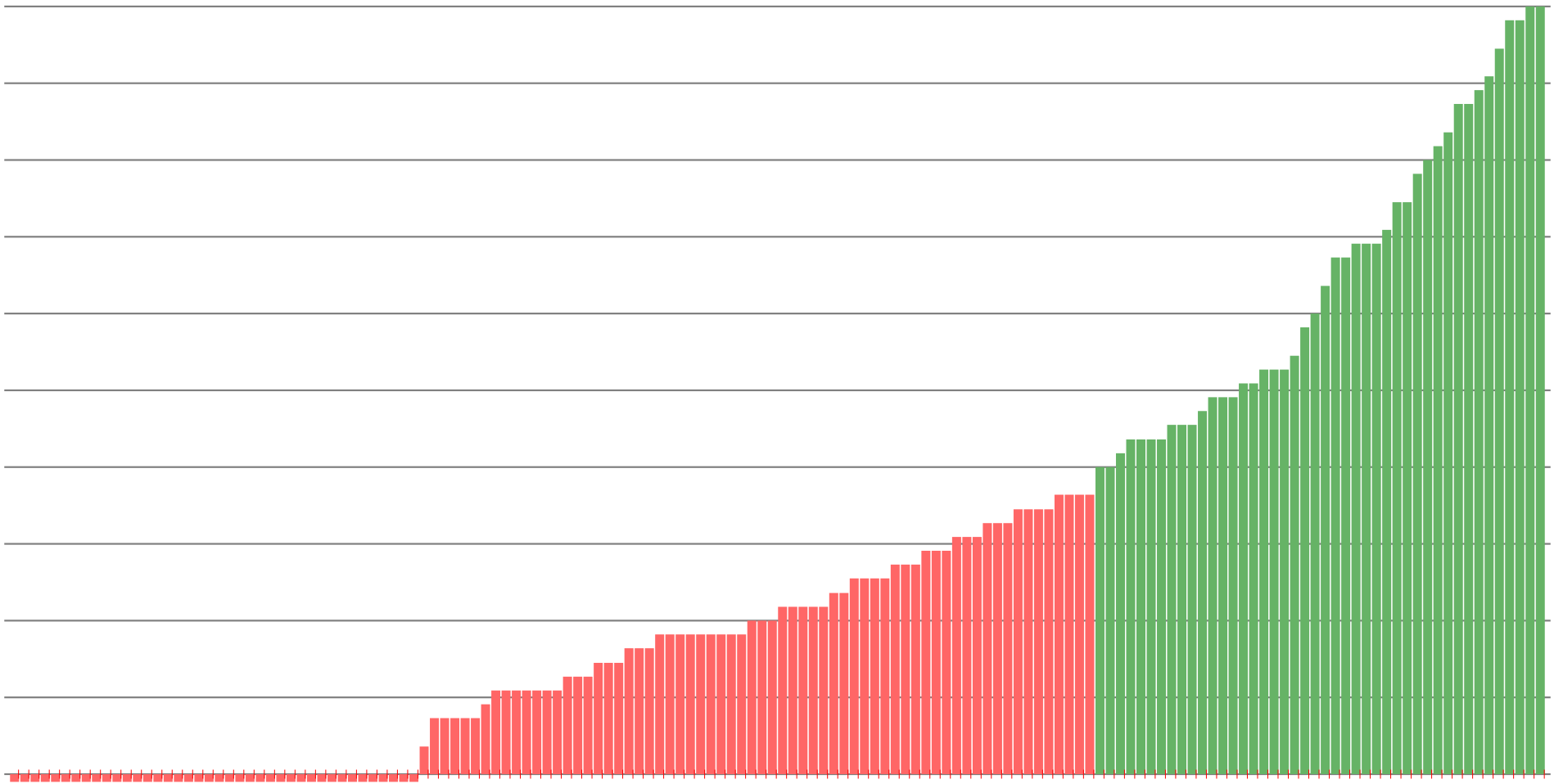
Computer Programming: Skills & Concepts (CP)
The Mock Exam

Monday 20 November 2017

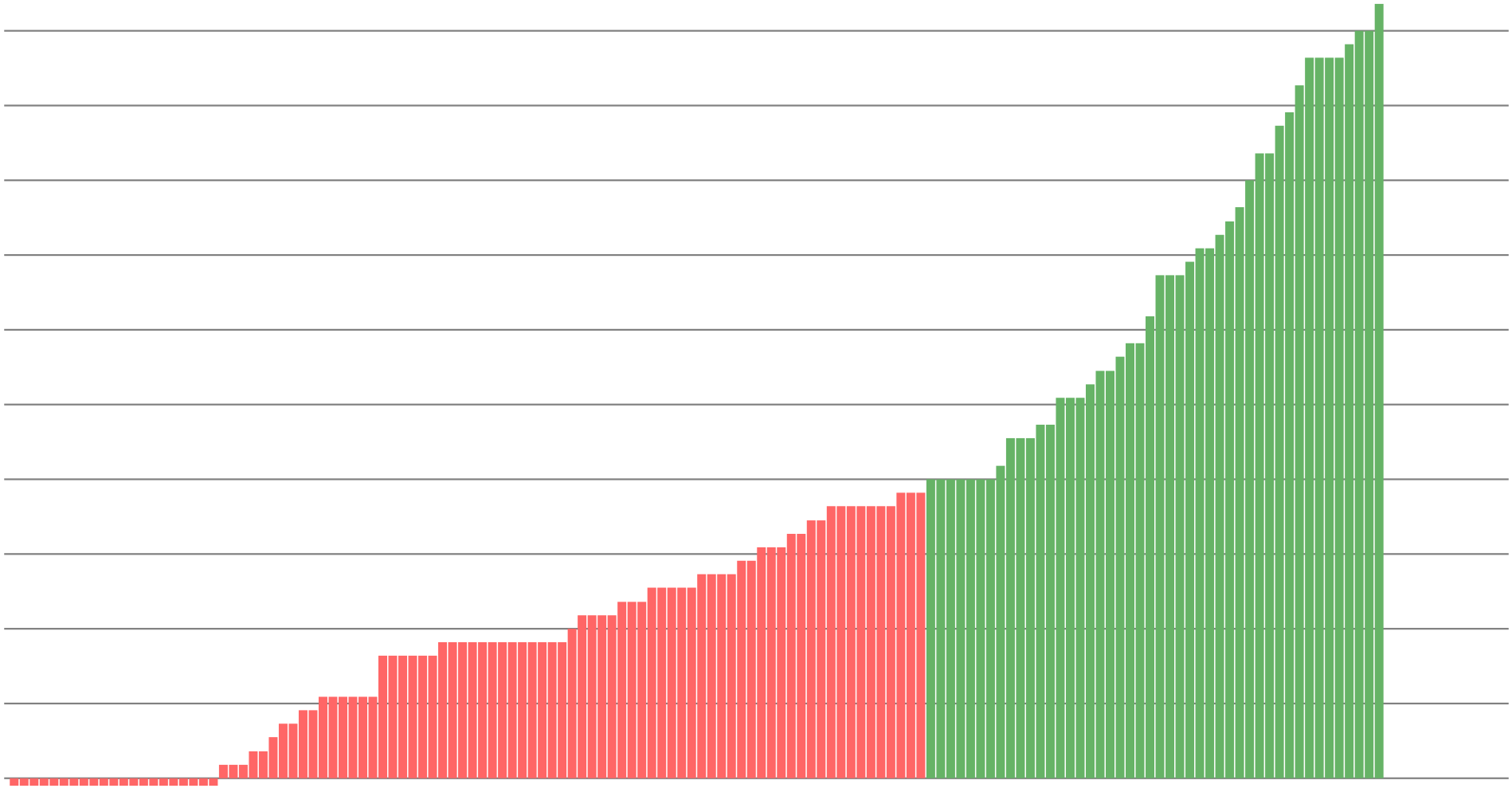
So how did you do?



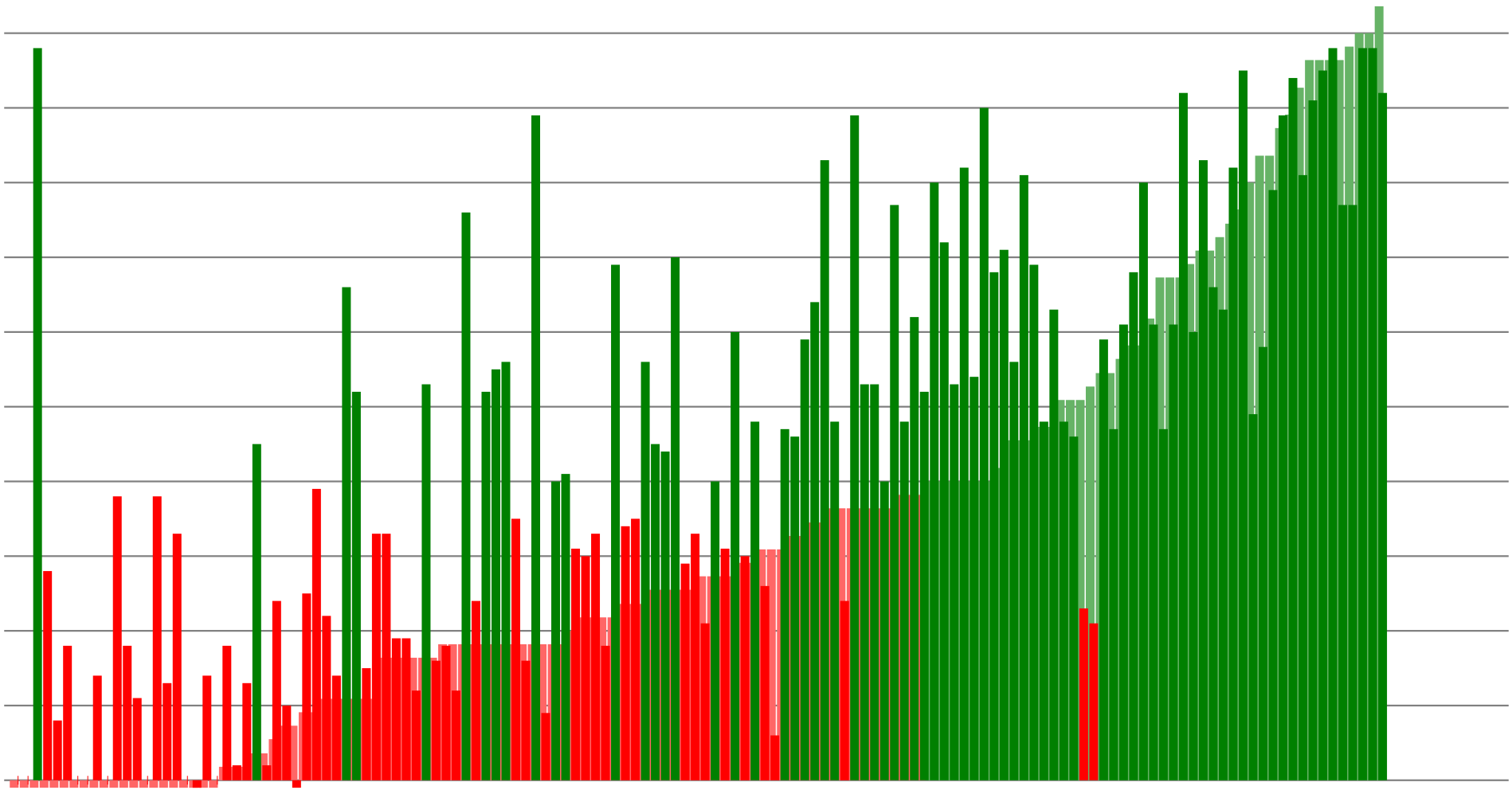
So how did you do?



What happened last year



What happened last year



Light colour: mark on mock. Solid colour: mark on exam.

What went wrong (for those who did badly on the mock)?

Partly, the mock is intentionally on the hard side – you have slightly less time, and the questions (particularly A2 and B1) are at the higher end of the difficulty range.

My best guess: you've been doing without understanding.

Previous chart shows there is hope! (But it needs work!)

Practise! Do exercises (lab tasks, past qs) without guidance. Revise lectures if needed. Discuss with others. Repeat until solved.

Don't be satisfied until you can do it **and** understand why.
Secret confession: even at my stage, one sometimes fixes a bug in a big program without quite understanding why the fix works . . .

Advice from one of last year's students: the book is good. Go through it, doing all the exercises. (But not everybody likes it.)

Serious problems

Those in the H–G range generally seemed to have little idea of really basic concepts, or the syntax. For example, in B1(a), I saw:

```
  if (c == a || c == e || c == i || c == o || c == u || c ==
 E || c == I || c == O || c == U || c == Y){
```

or even

```
  if (c == "a" || "e" || "i" || "o" || "u") {
```

Not understanding the function concept

A few people don't get the idea of a function: they `scanf` the argument of the function, instead of using the argument! e.g.

```
void printPig(char *w) {
  /* BEGIN ANSWER (c) -- do not delete this line */

  int i,n;

   printf("Please enter your word.\n");
   scanf("%s", w);

   ...
```

But I saw less of this than in previous years.

Function confusion

Another problem with functions was confusion between declaring functions, defining functions, and using functions. E.g. many people put

```
int toupper(int c);
```

in their code for A1, which broke otherwise correct code.

If a function is given to you in a library, you do **not** *declare it – you just use it, e.g. with*

```
c = toupper(c);
```

General issues

Follow instructions! When we say, don't change code outside the marked areas, we mean it.

Do what the question asks. Don't do anything that you are not asked to do. The statement

print the translated word

means exactly that.

General issues

Follow instructions! When we say, don't change code outside the marked areas, we mean it.

Do what the question asks. Don't do anything that you are not asked to do. The statement

print the translated word

means exactly that. It does **not** mean

print the translated word followed by a newline character

(if it did, it would say so)

General issues

Follow instructions! When we say, don't change code outside the marked areas, we mean it.

Do what the question asks. Don't do anything that you are not asked to do. The statement

> *print the translated word*

means exactly that. It does **not** mean

> *print the translated word followed by a newline character*

(if it did, it would say so), and definitely not

> *print* `"Your Pig Latin translation is "` *followed by the translated word followed by a newline*

This is not just important for exams. In real life programming, you are usually interacting with other software, using a precisely defined interchange format. (We are more careful about specifications in the real exams.)

Specifications

Some questions tell you precisely what to do.

Some questions show you example output. You should match the example output as closely as you can.

What if the question *really* doesn't tell you what to do? (E.g., how to align tables when the number of digits in entries varies.)
You can make any reasonable assumption, and if you state your assumption in a comment, we'll know it.
(But again, we try to avoid such problems.)

E.g. in `band.c`, how should you print a table of size > 10? You can *assume* that you should print them nicely aligned:

```
....
 0  1  0  1  0  1  0  1  8  9  8  9
 0  0  2  2  0  0  2  2  8  8 10 10
 0  1  2  3  0  1  2  3  8  9 10 11
```

E.g. in `band.c`, how should you print a table of size > 10? You can *assume* that you should print them nicely aligned:

```
....
 0  1  0  1  0  1  0  1  8  9  8  9
 0  0  2  2  0  0  2  2  8  8 10 10
 0  1  2  3  0  1  2  3  8  9 10 11
```

but you can also *assume* that a space between numbers is enough:

```
....
0 1 0 1 0 1 0 1 8 9 8 9
0 0 2 2 0 0 2 2 8 8 10 10
0 1 2 3 0 1 2 3 8 9 10 11
```

E.g. in `band.c`, how should you print a table of size > 10? You can *assume* that you should print them nicely aligned:

```
....
 0  1  0  1  0  1  0  1  8  9  8  9
 0  0  2  2  0  0  2  2  8  8 10 10
 0  1  2  3  0  1  2  3  8  9 10 11
```

but you can also *assume* that a space between numbers is enough:

```
....
0 1 0 1 0 1 0 1 8 9 8 9
0 0 2 2 0 0 2 2 8 8 10 10
0 1 2 3 0 1 2 3 8 9 10 11
```

but you *cannot reasonably assume*

```
....
 0 1 0 1 0 1 0 1 8 9 8 9
 0 0 2 2 0 0 2 2 8 81010
 0 1 2 3 0 1 2 3 8 91011
```

If you think the question is *genuinely ambiguous* about something you need to know, then call an invigilator.

We hope this won't happen, but some things can slip past even the three-stage scrutiny process. One of us will be in the exam all the way through.

Question A1

Major issues:

- character/string confusion
- variable/data confusion
- function confusion

Minor issues:

- not remembering the existence of `toupper()`
- not understanding what 'ASCII value' is, even though the question tells you

Question A2

Major issues:

- not reading the question (relative positions of other elements to remain unchanged)
- confusion about how to shift things

Minor issues:

- off-by-one errors
- tests wrong way round when searching for max

Question A3

Major issues:

- not understanding for-loops
- not understanding arrays

Minor issues:

- off-by-one errors (can segfault, not so minor!)
- unreasonable formatting of output

Question B1(a)

Major issues:

- confusion between strings and characters
- not using boolean expressions correctly

Minor issues:

- not using `toupper()`, but, e.g., adding 32
- not noticing that `'y'` was defined as a vowel

Style issues:

- writing 97 instead of `'a'`
- doing tests against all of `AEIOUYaeiouy` instead of uppercasing and then testing against `AEIOUIY`.

Question B1(b)

Major issues:

- not understanding for-loops
- not understanding arrays and/or the rotate algorithm, e.g.
- commands in loop that should be after loop

Minor issues:

- off-by-one errors with indices

Question B1(c)

Major issues mostly already seen, but:

- only rotating once instead of until vowel-initial

Minor issues:

- printing extra stuff