3 Programming in C and C++ (AM)

(*a*) You have acquired a C program which declares an array `v`, populates it, and later writes it to a file in binary:

```
#define NITEMS 100
struct Elem { signed long val; char flags; } v[NITEMS];
...
fwrite(file, 1, sizeof(v), v);
```

When run on a legacy processor (which no longer exists) this produces a file containing 500 bytes, but when re-compiled and executed on three modern desktops using various compilers it produces files containing respectively 800, 1200 and 1600 bytes. On all implementations `char` is an 8-bit value.

(*i*) Explain what might be happening in the four versions in terms of compiler assumptions of alignment and size, in bits, of type `long`. Also give the values of `sizeof(struct Elem)`. [3 marks]

(*ii*) You now wish to read a file produced by the legacy processor into a version of the program running on your new desktop machine (one of the three above). Outline the changes, if any, you would need to make to the call to `fread`, mirroring the call to `fwrite` above, so that the resulting `v` may be successfully processed by the rest of the program. You may make any sensible assumptions about the legacy machine and your desktop machine provided you state them explicitly. Indicate how your program might be able to read both legacy- and new-format binary files. [10 marks]

(*b*) (*i*) Write a C++ class `T` which contains a `const` integer field `n`. `T` should also have constructor(s) which initialise `n` to an integer argument passed as a parameter or to zero if no argument is given; `T` should also have a destructor. The constructor(s) and destructor should print the value of the `n` field of the object being constructed or destructed. Indicate why, or why not, any of your fields or methods are qualified with `virtual`. [3 marks]

(*ii*) Explain how objects of class `T` are allocated and deallocated, for each of the three areas: heap, stack and static store, noting one case where appropriate use of `virtual` is essential. What, if any, overlap in programmer convenience is there between stack-allocated objects with destructors and `try-finally` in Java? [4 marks]