5 Programming in C and C++ (djg11)

(a) A FIFO is implemented in C using a singly-linked list that maintains global head and tail pointers. These are initialised to represent the empty FIFO as follows:

```c
struct fifo_entry *head_ptr = 0;
struct fifo_entry **tail_ptr = &head_ptr;
```

The entries in the FIFO use a union to store either an integer or a double-precision floating point number. A further field records which is stored. Give syntactically-accurate C code that defines `fifo_entry` and either a function `enqueue_int` or a function `enqueue_double` enqueuing a new value into the FIFO.  

(b) To avoid repetitive reallocation of memory, suppose now that FIFO entries that are no longer in use are to be saved in an auxiliary linked list. Give syntactically-accurate code that implements this approach and then discuss two other approaches for store management.

(c) The C code in part (a) suffers from a lack of encapsulation – variables like `head_ptr` are visible to the the rest of the program. Write C++ defining a class `FIFO` which maintains a single FIFO implemented using elements `fifo_entry` as defined in your answer above, but which only exports member functions `enqueue_int`, `enqueue_double`, `isempty` and

```cpp
void dequeue(void do_I(int), void do_D(double));
```

It should not be possible to create an instance of class `FIFO` and storage allocation/deallocation should use C++ mechanisms rather than those of C. Your C++ is not required to be syntactically accurate, but should capture the main concepts. It is not necessary to give full code for the above four member functions – focusing on allocation and deallocation of `fifo_entry` elements suffices.  [Note: do_I and do_D are user-provided processing functions to be applied to the dequeued value.]

(d) The following lines approximate (e.g. omitting access qualifiers) analogous generic/templated class definitions in Java and C++. Explain which types X are valid for use in `Gen<X>` and `Tem<X>`.

```java
class Gen<T> { T v; Gen() { v = 1; } }
```

```cpp
template<typename T> class Tem { T v; Tem() { v = 1; } }
```

[Java]: class Gen<T> { T v; Gen() { v = 1; } };

[C++]: template<typename T> class Tem { T v; Tem() { v = 1; } };

[4 marks]