* Remember the following points when programming in C++ -
* Represent ideas directly in code.
* Represent relationships among ideas directly in code (e.g. hierarchical, parametric, and ownership relationships)
* Represent independent ideas independently in code.
* Keep simple things simple (without making complex ideas impossible)
* Prefer statically type-checked solutions (when applicable)
* Keep information local (e.g. avoid global variables, minimise the use of pointers)
* Don’t over-abstract (i.e. don’t generalise, introduce class hierarchies, or parameterise beyond obvious needs and experience.)
* Take the opportunities offered by new C++ facilities to modernise design and programming techniques –
* Use constructors to establish invariants.
* Use constructor/destructor pairs to simplify resource management.
* Avoid naked *new* and *delete*.
* Use containers and algorithms rather than built-in arrays and ad-hoc code.
* Prefer standard library facilities to locally developed code.
* Use exceptions, rather than error code, to report errors that cannot be handled locally.
* Use move semantics to avoid copying large objects.
* Use unique\_ptr to reference objects of polymorphic type.
* Use shared\_ptr to reference shared objects, i.e. objects without a single owner that is responsible for their destruction.
* Use templates to maintain static type safety (eliminate casts ) and avoid unnecessary use of class hierarchies.
* Suggestions for C programmers –
* Don’t think of C++ as C with new features added.
* Don’t write C in C++. That is often seriously sub-optimal for both maintenance and performance.
* Use the C++ standard library as a teacher of new techniques and programming styles.
* Macro substitution is almost never necessary in C++.
* Don’t declare a variable before you need it and initialise it immediately.
* Don’t use *malloc*(). The *new* operator does the same job better.
* Avoid void\*, unions, and casts, except deep within the implementation of some function or class.
* Minimise the use of arrays and C-style strings.
* Avoid pointer arithmetic except in very specialised code (such as a memory manager) and for simple array traversal.
* Do not assume that something laboriously written in C-style is more efficient than a shorter alternative. Often, the opposite is true.