# Design aims of C++ with comments

- **Close to the Hardware**: fundamental types (eg.: `bool`, `int`, `char` etc.) are in the language to support manipulating hardware resources directly without extra overhead.
  ⇒ if a structure is based on a `char` (1 byte) and an `int` (4 bytes), then that will consume 5 bytes of memory.
  On top of that, inline functions are supported without causing extra call overhead, as compile time constants supported too.

- **Allows Ideas to be Expressed in Code**: C++'s abstraction mechanisms are for supporting the programmer to express ideas, relationships with abstraction mechanisms such as:

  – **Classes -** Custom (aka. User Defined) types (eg.: Polygon) that support direct representation built from fundamental primitive types.

  – **Abstract Classes -** General Concepts that cannot be represented directly (eg.: Shape) that provides inheritance support.

  – **Inheritance -** To provide specialized classes from more generalized base classes (eg. Polygon inherits from Shape).

  – **Relationships -** To identify relationships between ideas by the usage of Class Hierarchies (eg. inheritence chain).

- **Efficient, Platform Independent and Backward and C Compatible**: very little overhead can be found at certain features (eg.: VTBL [virtual table lookups]), and many compiler implementations exist on many different platforms. The new versions are backward compatible, and C-compatibilty (as C is subset of C++) is maintained for not breaking any existing (millions of billions) lines of C++ code.

- **Supports Generalized Programming**: by supporting Templates, C++ provides help in building foundation libraries. The STL is a great example for that.

- **Type Safety**: Primitive and User Defined types are being checked at compile time (C++ is a strongly typed language).