Static Type Checking in C++

Type checking is being performed at *compile time* and **no type checking will be perfored at runtime**. ⇒ No Type information is being used at runtime.

Strength

No overhead at runtime, so if a structure is being defined with 4 double floating point numbers (2x8 bytes)
then, that will perfectly fit into a 32 bit register.

Weakness

No support against polymorphic objects, and only narrowing conversions can be made against types that are statically being checked.

Dynamic Type Checking in C++

Compilers that implement **RTTI** (*RunTime Type Information*) can deal with `dynamic_cast` and `typeid`,
where:

- **dynamic_cast** is responsible for casting an object (eg.: a pointer) from type of **T** to type of **U**. If the conversion is not possible, then the result will be null. A typical use would be to check whether one type is a base type of another type:

  ⇒ `Circle* c = dynamic_cast<Circle*> shape; // shape could be of any type`
  `// if shape is a base type of circle, then the cast can be made`

- **typeid** retrieves the class name of an object.
  It uses RTTI if the type of an object is not known at compile time (eg.: a polymorphic object's pointer):

  ⇒ `Person* p = &employee;`
  ⇒ `std::cout << typeid(p).Name() << std::endl; // looked up at runtime`

Strength

RTTI provides great support for polymorphic types and templates.

Weakness

RTTI implies overhead at runtime, so additional bytes of data needs to be stored, as well as time required to lookup type information in a given case. This feature should be used sparingly.