Language Independent Issues :

1. A project must not contain unused non-volatile variables.
2. A value returned by non-void function must always get used. But if the return value of a function call is to be discarded we can use a static_cast<void>cast.
3. Type long double should not be used.
4. Range, domain and pole errors shall be checked when using math functions.
5. Do not use deprecated STL library features.
6. Trigraphs should not be used(The Trigraphs are: ??=, ??/, ??', ??(, ??), ??!, ??<, ??>, ??-.)
7. Digraphs shall not be used(The Digraphs are:<%, %>, <:, :>, %:, %:%:)
8. The character \ should not occur as a last character of a C++ comment.
9. The name of a header file should reflect the logical entity for which it provides declarations.
10. Identifiers: An identifier declared in an inner scope shall not hide an identifier declared in an outer scope.
11. The volatile keyword shall not be used unless directly interfacing with hardware.
12. Hexadecimal constants should be upper case.
13. Do not concatenate strings with different encoding prefixes.
14. Type wchar_t shall not be used.
15. Comments should be written using //.. we should not use /* .. */
16. All code should compile without any compilation warnings

Basic Concepts:

1. In header file, only inline and template functions and variables with the keyword extern & constant static are allowed to have definition
2. Header files should have extensions of ".h", ".hpp" ,".hxx".
3. Implementation file should have an extension as ".cpp".
4. When an array is declared with an external linkage,its size shall be explicitly stated.
5. Program shall contain only one definition of every non-inline function or object.
6. Static and thread-local objects shall be constant-initialized.
7. An object shall not be accessed outside of its lifetime.
8. Fixed width integer types shall be used in place of basic numerical types.
9. Functions shall not be declared at block scope.
10. Fixed width integer types shall be used in place of basic numerical types.

Standard Conversions:

1. Expressions with type bool shall not be used as operands other than operator like =, &&, || , ! , ==and! =, unary and the conditional operator.
2. Enum or enum classes shall not be used as operands.
3. NULL should not be used an as integer value.
   Exception - Only nullptr literal shall be used as the null-pointer-constant.

Expressions:

1. Value of an expression shall always be same under any sequence of evaluation.
2. Implicit floating-integral conversions shall be avoided.
3. In object declaration,use of more than two levels of pointer indirection shall be avoided.
4. Return type of a non-void return type lambda expression should be explicitly specified.
5. dynamic_cast should not be used.
6. Traditional C-style casts shall not be used.
7. The increment (++) and decrement (--) operators shall not be mixed with other operators in an expression.
8. The comma operator, && operator and the ||operator shall not be overloaded. Unary & operator should not be overloaded.
9. The result is undefined if the right hand operand of the integer division or the remainder operator is zero.

Special Member Functions :

1. When any of special access members is non default then all special member functions should be defined or declared using “=default”, “=delete” .

2. Non-static data member initializer and constructor should not be used at same time.

3. If all user-defined constructors of a class initialize data members with constant values that are the same across all constructors, then data members shall be initialized using NSDMI instead.

4. Common class initialization of class members shall be done by delegating constructors.

5. If derived class does not have any member to initialize and require all the constructors from the base class shall use inheriting constructors..

6. Class members shall be initiazed using constructor initiazer list rather than assigning values.

7. It is expected behavior that the move/copy constructors are only used to move/copy the object of the class type without any side effects.
8. User-defined copy and move assignment operators should use user-defined simple swap of pointers.
9. Moved-from object shall not be read-accessed.
10. User-defined copy assignment operator and move assignment operator need to prevent self-assignment.
11. Assignment operators should be declared with the ref-qualifier &.

Overloading:

1. user-defined literals operators shall only be used to convert passed parameters to the type of declared return value.
2. assignment operator shall return a reference to “this”.
3. A binary arithmetic operator and a bitwise operator shall return a “prvalue”.
4. A function that contains “forwarding reference” as its argument shall not be overloaded.
5. If constant version of operator[] is to be overloaded then it’s non constant version shall also be implemented.
6. All conversion operators shall be defined explicit.
7. Comparison operators shall be non-member functions with identical parameter types and noexcept.

Templates:

1. A template constructor shall not participate in overload resolution for a single argument of the enclosing class type.
2. A template should check if a specific template argument is suitable for this template.
3. A template constructor shall not participate in overload resolution for a single argument of the enclosing class type.
4. non dependent members on template class parameters should be defined in a separate base class.
5. A non-member generic operator shall not be declared in a namespace that contains class (struct) type, enum type or union type declarations.
6. Explicit specializations of function templates shall not be used.
7. Template specialization shall be declared in the same file (1) as the primary template (2) as a user-defined type, for which the specialization is declared.
8. If any type is used as argument to template then it shall provide all members that are used by the template.

Exception:

1. As exceptions are derived from std::exception , then only instance type of that class can be thrown.
2. An exception object shall not be a pointer. As it creates ambiguity.
3. All thrown exceptions should be unique.
4. A checked exception is a type of exception that must be either caught or declared in the method in which it is thrown.
5. Program shall not be abruptly terminated. Means invocation of std::abort(), std::quick_exit(), std::_Exit(), std::terminate() shall not be done.

Preprocessor Directives:

1. The ’, ", / * , //, \ characters shall not occur in a header file name or in #include directive.
2. If include directive is added at start of code, and later no member in that directive is used, then such include directives should be avoided.

Library:

1. No need to define identifiers, macros and functions which is already defined in libraries. Programmer should use that by calling it only.
2. All project’s code including used libraries (including standard and user-defined libraries) and any third-party user code shall conform to the AUTOSAR C++14 Coding Guidelines.
3. Non-standard entities shall not be added to standard namespaces. Means adding declarations or definitions to namespace std, member function of a standard library class which leads to undefined behavior.

Input/Output Library:

1. Inputs from independent components should be verified as it may harm the whole system/software
2. #include<cstring> should not be used.
3. An character buffer should guarantee sufficient space for data and null terminator i.e., ‘/0’.
4. We should use fseek,fsetpos, or rewind function while reading from file or writing into a file.

Algorithms library:

1. The function which is using predicate object should be const.or we should use std::ref.
2. We should not use std::rand() for pseudo random number generations,rather than we should use random number engine.

Exception - std::random_shuffle is deprecated since C++ 14.

Container Library :

1. Elements in a container(vector,set) should be accessed through via valid references,iterator , pointers.

Statements and Declaration:

1. Switch statement at least have two case-clause other than default.
2. A for-loop counter should not be a floating pointer
3. We cannot use do-while loop because it checks the condition at the end.
4. Goto statement shall not be used because it complicates the logic
5. Const keyword should be written in this format:

   IntPtr const ptr2 = &value 1

6. The auto specifier shall not be used apart from following cases: (1) to declare that a variable has the same type as return type of a function call, (2) to declare that a variable has the same type as initializer of non-fundamental type, (3) to declare parameters of a generic lambda expression, (4) to declare a function template using trailing return type syntax.
7. The typedef specifier shall not be used.
8. A class, structure, or enumeration shall not be declared in the definition of its type.
9. Enumeration can be initialized fully or none.
10. The asm declaration shall not be used because the inline code restricts the probability of code.
11. A function should not return any reference (pointer) to a parameter that is pass by reference to const.
12. Functions shall not call themselves, either directly or indirectly.
13. Functions shall not be defined using the ellipsis notation.

Declarators:

1. We should not declare any variable using comma (,).

   For eg: int a,b,c; //non-compliant

   int a;

int b; //compliant

2. If we are overriding any virtual function, then we should only those parameters which are declare in virtual function.
3. If we are overload any function, then we should not use ellipsis (…) in function.
4. If there is multiple output return by a function, then it should be return via struct or tuple.
5. If you pass any reference parameter, then those parameters should not be NULL.
6. Interface shall be precisely and strongly typed.
7. If we are initializing any variable then we should not use equals sign, we only use {} for initialisation

   Exceptions:
   The variables having data type as size_t/auto should not be initialized using {},rather we should use ().
   For eg size_t i(0){};

Language support library :

1. C-style arrays shall not be used.
2. Functions malloc, calloc, realloc and free shall not be used.(Allocate memory using new and release it using delete.)
3. “operator new” and “operator delete” shall be defined together i.e. correctly declare overloads for operator new and delete.
4. The signal handling facilities of <csignal> shall not be used. It is recommended to use lambda expressions instead.
5. Objects that do not outlive a function shall have automatic storage duration.
6. The form of the delete expression shall match the form of the new expression used to allocate the memory.Ensure that the form of delete matches the form of new used to allocate the memory.

General utilities library :

1. An already-owned pointer value shall not be stored in an unrelated smart pointer.
2. A std::unique_ptr shall be used to represent exclusive ownership.
3. A std::shared_ptr shall be used to represent shared ownership.
4. A std::unique_ptr shall be used over std::shared_ptr if ownership sharing is not required

Classes:

1. Unions shall not be used because their usage can be misleading and misinterpreted by developers.
2. Bits field must be used when interfacing to hardware or conforming to communication protocols.

Derived Classes:

1. Classes should not be derived from virtual bases because it introduces number of undefined and confusing behaviors.
2. Non-virtual public or protected member functions shall not be redefined in derived classes to avoid unnecessary complexity and errors.(But redefinition from private inheritance do not violate this rule.)
3. Virtual function declaration must have one of the three specifiers virtuals, override, final.
4. Virtual functions are not allowed to use in final class.

Member Access control:

1. Non-POD Class,data-member must be private.
2. Friend declaration shall not be used, it reduces encapsulation and result code is difficult to maintain.