CodingStyle

indent

- In general, indent is 4 space
- For class access modifier (public, protected, private), use 2 space
- No indent for "friend" modifier

naming

- Type use Pascal Case naming convention
- Variable use lowercase-underscore convention
- Macro use uppercase-underscore convention
- Template parameter use uppercase-underscore convention
- Enum type should use uppercase-underscore convention

symbol

- A pointer declartion should has no space between star and type specifier
- Left curly brace should start with new line

class definition

- Virtual must specify override modifier if the virtual member function override parent's method
- Private member variable name should append an underscore
- All member variable should cluster together
- A ".hpp" file should only has one public class
- Any class in namespace with "detail" postfix is a private class

ban word

- goto, int, long, short

layout

- One line source code should not longer than 70 character

A valid code example

```cpp
#include <utility>
#include <cinttypes>
#define RETURN_ZERO return 0;
class YourType
{
    virtual void your_method() = 0;
};
enum MyEnum
{
    CAT, DOG, SNAKE
};
template<class TPL_ARG>
class MyType : public YourType
{
friend class YourType;
  public:
    int16_t length;
    int16_t weight;
  private:
    int32_t my_private_var_;
    TPL_ARG animal_;
  public:
    MyType() = default;
    auto get_var()
    {
        return my_private_var_;
    }
    auto my_method(
          std::size_t arg_name_so_long_must_break
        , std::size_t this_arg_is_so_long_too
    )
    {
        RETURN_ZERO;
    }
    void your_method() override
    {
        my_private_var_ ++;
    }
};
```

```cpp
// Suggested by Alex
#include <CPT/utility/class1.hpp>  //[OK]
#include "class2.hpp"              //[OK]
#include "xxx/class3.hpp"          //[OK]
#include "CPT/utility/class1.hpp"  //[BAD]
#include "../class1.hpp"           //[BAD]
#include <../class1.hpp>           //[BAD]

namespace aaa {
namespace bbb {
  // ...
} // namespace bbb
} // namespace aaa

/// Enumeration style
enum Animal { CAT, DOG, SNAKE };
enum Pokemon
{
    PIKACHU, BULBASUR, CHARMANDER, SQUIRTLE
};
enum Food
{
    BRAISED_PORK_RICE
  , BEEF_NOODLES
  , BUBBLE_TEA
};

/// Pod type
int32_t
int64_t
size_t or std::size_t ?

/// Struct
struct Base
{
    // ...
};
struct Derive
     : public Base
{
    // ...
};

///
class MyObjectBaseA
{
  public:
    virtual void method_aa(void) = 0;
    virtual void method_ab(void) { }
    virtual ~MyObjectBaseA(void) { }
};
class MyObjectBaseB
{
  public:
    void method_ba(void) = 0;
    void method_bb(void) { }
};
class MyObjectDerive
    : public MyObjectBaseA
    , public MyObjectBaseB
{
  private:
    // Group1
    int32_t var1_;
    int32_t var2_;
    // Group2
    int32_t var3_;

  protected:
    int32_t var4_;

  public:
    int32_t var5;

  public:
    void method_aa(void) override
    {
        std::cout << "this is aaa\n";
    }

  protected:
    void protected_method(void)
    {
        // ...
    }

  private:
    void private_method(void)
    {
        // ...
    }
};

/// template
template <class T>
class { };

template <
    class AAA
  , class BBB
  , class CCC
  , template<class>class DDD
>
class
{
    // ...
};

// lambda function
void foo( ... )
{
    // OK
    std::sort(
          vec.begin()
        , vec.end()
        , [](const auto& a, const auto& b)
        {
            return a < b;
        }
    );

    // OK
    std::sort(
        vec.begin()
      , vec.end()
      , [](const auto& a, const auto& b)
        {
            return a < b;
        }
    );
}

std::unorder_map<std::string, std::map<std::string, std::string>>
function_name(void)
{
    // ...
}

std::unorder_map<
    std::string
  , std::map<
        std::string
      , std::string
    >
>
function_name(void)
{
    adfsdf
}

auto function_name(
    const int32_t aaa
  , const int32_t bbb
  , const int32_t ccc
)
{
    return aaa + bbb + ccc;
}
```