Doxygen Guildline

1. Comment style

- There are several ways to mark a comment block as detailed description:

```
1. C-style
/**
 * Comment here
 */

2. Qt-style
/*!
 * Comment here
 */

3. Cpp-style
///
/// Comment here
///
or
//!
//! Comment here
//!

4. Single line comment
/// Comment here
or
//! Comment here
```

Recommend to use C-style or Qt-style for multiple line and single line comment

2. Documenting the code

```
When documenting the code (struct, class, function,...), you can put it in
.h/.hpp or .c/.cpp file but NOT BOTH. If you do that, your documentation
shall become  inconsistent and contradictory over time or you must take
time to maintain all of them.
However, it is recommended is put it in **.h/.hpp**.
```

2.1. Documenting variable

Below are some example for documenting a variable:

```
//! @brief Desciption for var1 (recommended)
int var1;
or
int var2;   //!< Description for var2 (recommended)
```

```
or
/// Description for var3
int var3;
or
int var4;   ///< Description for var4
```

2.2 Documenting struct, union, enum

Below is example for documenting a struct, union, enum:

```
//! @brief Description for Rectangle struct
struct Rectangle{
    int width;  //!< The width of the rectangle
    int height; //!< The height of the rectangle
};


//! @brief Description for UnionName
union UnionName{
    char mem1;  //!< Description for mem1
    int  mem2;  //!< Description for mem2
    long mem3;  //!< Description for mem3
};

//! @brief Description for EnumName
enum EnumName{
    int EVal1;  //!< Description for Eval1
    int Eval2;  //!< Description for Eval2
    int Eval2;  //!< Description for Eval3
};
```

2.3 Documenting the function

```
When documenting a function attribute @brief is mandatory, this attribute
show the user known what exactly function do.

If function need to be passed parameter , attribute @param is mandatory
(with @param[in] for input parameter, @param[out] for output parameter or
@param[in/out] for parameter with is using as input and output). With
function has no param, this attribute can be ignored.

If function return a value, attribute @return is mandatory. It will provide
for user known what value can be return when using this funcction. With
function return void, this attribute can be ignored.

Adding the @details attribute is possible if necessary. In this attribute,
the user can see how the function works. It contains the detail description
of the function, step by step.
```

```
When documenting parameters and return values, do not simply list the
parameter or return value's data type. That information can easily be found
by looking at the first line of the function or method or in its prototype.
Your documentation should instead explain the purpose and meaning of the
parameter or return value. It's okay to mention the data type, but it is
not sufficient.

The other tags (@note, @warning, and @bug) will only be needed in rare
cases.
```

Below is example documenting a function:

```
/**
 * @brief Get path of a file in a disk (summarize what exactly method do?)
 *
 * @details Detail description of method (How function do?)
 *          - Open disk and search the file name in current folder.
 *          - If can not exist in parent foler
 *              + Continous find in all childen folders
 *          ........ Continue descrise by detail
 *
 * @param[in] filename: the file name need to be got the path
 * @param[out] path    : the output buffer which store the path to filename
 *
 * @return true if get successfully
 */
bool getPath(char* filename, char* path);


or

/*!
 * \brief Get path of a file in a disk (summarize what exactly method do?)
 *
 * \details Detail description of method (How function do?)
 *          - Open disk and search the file name in current folder.
 *          - If can not exist in parent foler
 *              + Continous find in all childen folders
 *          ........ Continue descrise by detail
 *
 * \param[in] filename: the file name need to be got the path
 * \param[out] path    : the output buffer which store the path to filename
 *
 * \return true if get successfully
 */
bool getPath(char* filename, char* path);
```

2.4 Documenting the class

Documenting the class is combine of documenting the variable, struct, union, enum and function.

```
//! @brief A Student Class (Brief description for Student class)
class Student
{
private:

    //! @brief struct hold marks of student
    struct marks_t{
        char math;        //!< math mark of student
        char physical;    //!< physical mark of student
    }marks;

    int id;          //!< Student id
    string name;     //!< Student name

public:

    //! @brief Default Constructor
    Student();

    //! @brief Default Destructor
    ~Student();

    /**
     * @brief Get name of this student.
     *
     * @return Name of student.
    */
    string getName(void);

    /**
     * @brief Set name for the student.
     *
     * @param[in] name: The name shall be set for this student.
     */
    void setName(string name);
    ......
}
```

3. Special Command

Refer Doxygen Home Page for more details.