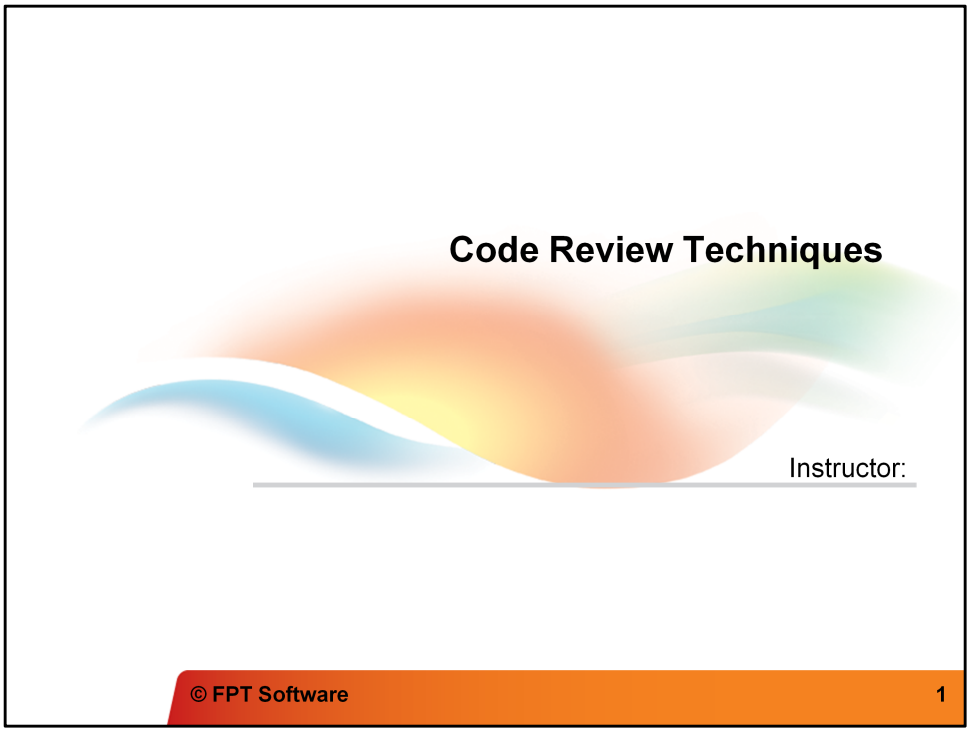



Latest updated by: HanhTT1

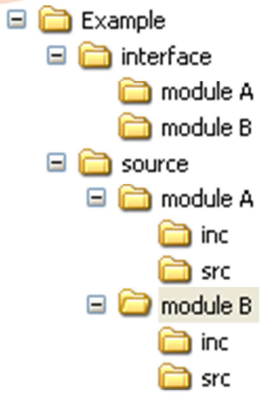
## Agenda

- This document presents a compilation of the C/C++ coding standards
- An attempt has been made to outline a style that is easily readable, amenable to documentation and acceptable to most programmers.
- Most of the details given here apply equally to C and C++ programs

## Management file

- Interface/module A: contains header file of which are used by other modules
- Source/module A/inc : contains header file of which are only used in module A

Refer: Standard_C Coding Convention .pdf



## Source File Header

```
/**************************************************************************
* ++ * Author: American Management Systems
* Module Name : xxxxxxxx.cpp
*
 * Description :
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
*
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 *
 * Mod. History : DD.MMM.YY - Full Name
*                                File first created
*                                CR# PROJECT
*                                Description
* * --
**************************************************************************/
```

## File Contents

| | C Program Language |
|---|---|
| Struct declaration | header file (.h) |
| Struct implementation | source file (.c) |
| Function prototypes | header file (.h) |
| Function definitions | source file (.c) |

## Header File Layout

```
#ifndef  MeaningfulNameH        // first line of the header file
#define MeaningfulNameH         // second line of the header file
.
.                               // body of the header file
.
#endif // MeaningfulNameH       // last line of the header file
```

## Comments

```
/* single line comments look like this */

/*
 * Important single line comments look like multi-line comments.
*/

 /*
* Multiline comments look like this. Put the opening and closing
* comment sequences on lines by themselves. Use complete sentences
* with proper English grammar, capitalization, and punctuation.
*/

/* but you don't need to punctuate or capitalize one-liners */
```

Sample: Standard_C Coding Convention .pdf

## Declarations and Types

- When declaring a global function or variable in a header file, use an explicit extern.

```
extern int errno;
extern void free(void *);
```

- Creating the typedef eliminates the clutter of extra struct and union keywords

```
typedef struct Foo Foo;
typedef struct Bar Bar;
struct Foo {
        Bar *bar;
};
struct Bar {
        Foo *foo;
};
```

## Function Names

- Function do a task should have names as: Do[Something] or Make[Something] (the first is a verb and then a noun).
- Function process an event: On[EventName], OnReceiveData, OnSendMailHeader,…
- Call-back function: [Event]Proc, ThreadProc, WinProc,…

## Variable Naming Conventions

- Variable names are composed of two important parts:
  - a type
  - a qualifier (that belongs to a small set of standard qualifiers)
- example:

  *szPassword* the type is sz and the qualifier is Password

## Base Types

| | |
|---|---|
| sz | null-terminated string |
| ch | character |
| w | unsigned integer (word) |
| l | long |
| f | boolean (flag) |
| x | structure |
| c | C++ class |

## Examples

| | |
|---|---|
| szFilename | Filename null-terminated string; |
| chYesLit | Character representing a Yes; |
| wNumUsers | Number of users; |
| lOffset | Long offset; |
| xRect | Structure defining a rectangle; |
| pxRect | Pointer to a rectangle structure; |
| hWindow | Handle to a window. |
| cPerson | Class person... |

## Naming Conventions

| Identifier | C program language |
|---|---|
| struct | EachWordCapitalized |
| typedef | EachWordCapitalized |
| enum | EachWordCapitalized |
| pointers | namePtr |
| function, method | internalWordsCapitalized |
| object, variable | internalWordsCapitalized |
| #define, macro | ALL_CAPS_AND_UNDERSCORES |
| const, static final | ALL_CAPS_AND_UNDERSCORES |
| source file | .c |

## Style Guidelines

- When an expression will not fit on a single line, break it according to these general principles:
  - Break after a comma.
  - Break before an operator.
  - Prefer higher-level breaks to lower-level breaks (see example).
  - Align the new line with the start of the expression at the same level on previous line.

## Style Guidelines

```
someMethod(firstInteger,  secondInteger,  thirdInteger,
                    FourthInteger); //Break at comma, and align
someMethod1(firstInteger,
                    SomeMethod(secondInteger,  thirdInteger));
                                //Break at higher-level.
firstInteger  =  secondInteger * ( thirdInteger +
   fourthInteger)
                  + fifthInteger ; //Break before operator.
```

## Formatting

```
if (value == 0)                    // correct
{
    doSomething();
}
if ( value==0 )                    // wrong, no spaces around ==
{
    doSomething();
}
if ( value == 0 )                  // wrong, spaces around parentheses
{
    doSomething();
}
```

## Indentation and Braces

```
int main()
{
        doSomething();
        switch ( value )
        {
                case 1:
                        while ( value == 0 )
                        {
                                doSomething();
                        }
                        break;
                case 2:
                case 3:
                        doSomething();
                        break;
                default:
                        break;
                }
}
```

## Control Statements

```
if (value == 0)
{                                   // Correct
        doSomething();
}

if (value == 0) doSomething(); // not recommended – no block, not indented

if (value == 0)
        doSomething();      // not recommended - no block
```

## Include Statements

```
#include <ltstdlib.h>                    // Correct
#include <ltstdio.h>                     //
#include <ltXm/Xm.h>                     //
#include "meaningfulname.h"              //

#include "/proj/util/MeaningfulName.h"          // wrong – absolute path given
#include <ltstdlib.h>                           // wrong – out of order
#include </usr/include/stdio.h>                 // wrong – path given for system file
```

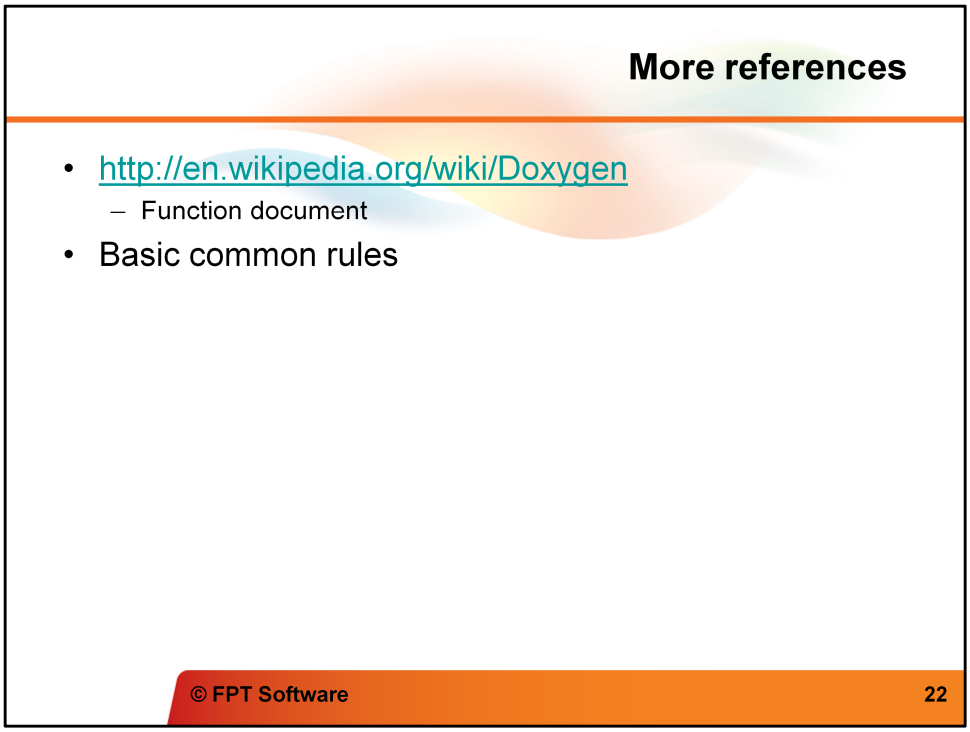
## Standard containers

- Definition of some of standard containers:

| Containers | Description |
|---|---|
| vector | one dimensional array of T |
| list | doubly-linked list of T |
| deque | double-ended queue of T |
| queue | queue of T |
| stack | stack of T |
| map | associative array of T |
| set | set of T |
| bitset | array of booleans |

## Standard containers

- The user can choose between containers based on efficiency concerns and the need for specialization operations.
- For example:
  - if lookup based on key is common, a **map** can be used
  - if general list operations dominate, a **list** can be used
  - if many additions and removals of element occurs at the end of the container, a **deque**, a **stack**, or a **queue** should be considered
  - by default, a **vector** should be used




Chỉ tham khảo Function document.
Các format cho File document xem tài liệu của FSoft

## Function document

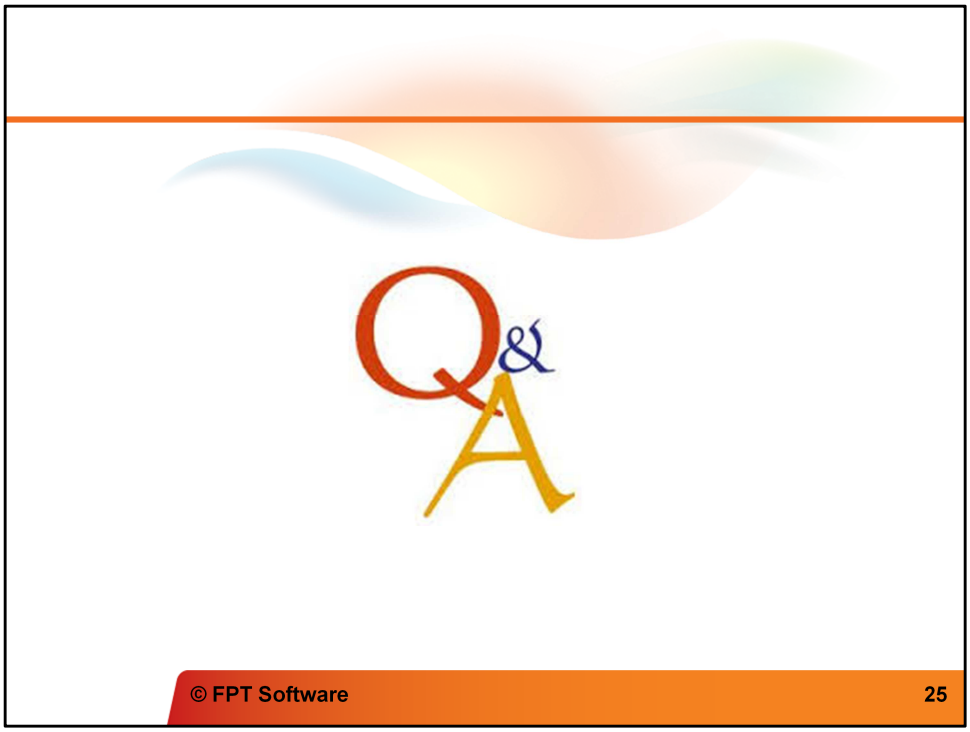
```
/**
<A short one line description>

<Longer description>
<May span multiple lines or paragraphs as needed>

@param  <Param name> Description of method's or function's input
    parameter
@param  ...
@return Description of the return value
*/
```

## Basic common rules

- C-CPlus_Coding guideline.xlsx


Q&A