NHN C/C++ 코딩 규칙

Rule List

목차

1. buffer-overflow에 위험한 함수를 사용하지 않는다.

: windows에서 buffer overflow에 위험한 함수 리스트

strcat	wcscat	lstrcat	strcat	StrCatBuff
_tcscat	_ftcscat	strncat	StrNCat	strcpy
wcscpy	lstrcpy	strcpy	_tcscpy	_ftcscpy
Strncpy	gets	_getws	_getts	Sprint
swprintf	wsprintf	wnsprintf	_stprintf	_snprintf
_snwprintf	_sntprintf	vsprintf	vswprintf	Wvsprintf
wvnsprintf	_vstprintf	_vsnprintf	_vsnwprintf	_vsntprintf
Strlen				

2. 파일이름은 언더바(_)로 시작하지 않는다.

_a.c <- Error
_bsds.h <- Error

아래와 같이 사용한다.

a.c
BdSc.h

3. 같은 파일명은 한번 이상 사용하지 않는다.

/testdir/test1.c
/testdir1/test1.c <- 위반. 파일명 'test1'은 두번 사용된다.

아래와 같이 사용한다.

testdir/test.c
testdir1/test1.c

4. 파일명에 특별한 문자를 사용하지 않는다.

파일이름은 알파벳, 숫자, 언더바(_)만 사용한다.

```
/testdir/test-1.c    <- 위반. – 이 사용됨
/testdir1/test!1.c   <- 위반. !가 사용됨.
```

아래와 같이 사용한다.

```
testdir/test.c
testdir1/test_1.c
```

5. cpp 파일 이름은 대표 class 이름으로 사용한다.

파일이름은 대표적인 class/struce 이름을 포함한다. 파일이 class/struct 모두 있다면, 클래스 중의 하나로 이름을 짓는다.
만약 클래스이름이 'C'로 시작하면, 'C'는 파일이름에서 생략할 수 있다.

```
a.h   <- 위반. 클래스 이름 'TestClass'을 포함해야 한다.
class TestClass()
{ }
```

```
a.cpp <- 위반. 클래스 이름 'Test'를 포함해야 한다.
void Test::Method1()
{ }
```

아래와 같이 사용한다.

```
TestClass.h <- OK
Class TestClass
{ }
```

```
Class1.h <- OK
Class CClass1
{ }
```

```
TestClass.cpp <- OK
Void TestClass::Method1()
{ }
```

6. cpp 파일 이름은 언더바(_)를 사용하지 않는다.

cpp 파일이름은 알파벳과 숫자만을 사용한다.

```
/testdir/test_1.cpp    <- 위반. _이 사용됨
/testdir1/_test.cpp    <- 위반. _이 사용됨.
```

아래와 같이 사용한다.

```
Testdir/test.cpp
testdir1/test_1.c      <- c 파일은 상관 없다
```

7. c 파일이름은 대문자를 사용하지 않는다.

c 파일은 대문자를 사용하지 않는다. 이 규칙은 'c' 파일만 해당된다.

```
/testdir/test_A1.c       <- 위반. 대문자 A가 사용됨
/testdir1/_TestBeta.c    <- 위반. 대문자 T와 B가 사용됨
```

아래와 같이 사용한다.

```
testdir/Test.cpp      <- cpp 파일은 상관 없다
testdir1/test1.c      <- OK
```

8. bool값을 리턴하는 경우 함수 이름은 is혹은 has로 시작한다.

```
bool checkSth() {         <- 위반. 함수 이름은 isSth나 hasSth이어야 한다.
        return false;
}
```

아래와 같이 사용한다.

```
bool isSth() {        <- OK.
    return true;
}

is isSth() {          <- bool을 리턴하지 않기 때문에 상관 없다.
}
```

9. windows에서는 함수 이름을 대문자로 시작한다.

Window C/C++ 코드에만 이 규칙이 적용되며, 유닉스에서는 소문자로 함수 이름을 시작한다.

```
bool checkSth()     <- 위반. 함수 이름이 소문자 'c'로 시작한다.
{
    return false;
}

bool _checkSth()    <- 위반. 함수 이름이 소문자 'c'로 시작한다.
{
    return false;
}
```

아래와 같이 사용한다.

```
bool IsSth()        <- OK.
{
    return true;
}

bool _IsSth()       <- OK.
{
}
```

10. private 함수 이름은 언더바(_)로 시작한다.

이 규칙은 cpp파일에만 적용된다.

```
class A
{
private:
    bool GetSth();  <- 위반.private 함수는 _로 시작한다.
};
```

아래와 같이 사용한다.

```
class A
{
public :
    bool GetSth();    <- public 함수는 상관 없다.
private:
    bool _GetSth();   <- OK.
};
```

11. 들여쓰기는 tab을 사용한다.

```
void Hello()
{
[SPACE][SPACE]Hello(); <- 위반. 들여쓰기에 스페이스가 사용되었다.
```

```
}
```

아래와 같이 사용한다.

```
void Hello()
{
[TAB]               <- 빈 라인이기 때문에 상관 없다.
[TAB]Hello();       <- Good
}
```

12. 들여쓰기는 space를 사용한다.

```
void Hello()
{
[TAB]           <- 빈 라인이기 때문에 상관 없다.
[TAB]Hello();   <- 위반. 들여쓰기에 tab이 사용되었다.
}
```

아래와 같이 사용한다.

```
void Hello()
{
[TAB]                   <- 상관 없다.
[SPACE][SPACE]Hello(); <- Good.
}
```

13. enum 블록내의 아이템은 들여쓰기 한다.

```
enum A {
    A_A,      <== Good
    A_B
}
```

14. eunm 아이템은 분리된 라인에 작성한다.

```
enum A {
    A_A, A_B <== Violation
}
```

아래와 같이 사용한다.

```
enum A {
    A_A,      <- Good
    A_B
```

```
}
```

15. 함수의 긴 파라미터 리스트는 정렬한다.

```
void functionA(int a, int b
               int c);                          <- 위반

void functionB(int a, int c,
                    int d)                      <- 위반
```

아래와 같이 사용한다.

```
void functionA(int a, int b
                int c);                         <- OK
```

16. 조건문은 정렬한다.

```
if (a == b &&
  a == c)      <- 위반
```

아래와 같이 사용한다.

```
if (a == b &&
    a == c)   <- OK
```

17. operator 주위에 공백을 둔다.

이항 연산자 앞과 뒤에 공백을 제공한다.

단항 연산자 앞과 뒤에 공백을 제공하나, (A++), [--BB], {--KK}와 같이 사용할 때는 공백이 없어도 좋다.

일부 연산자(" , ", " ; ")는 연산자 뒤에 공백을 제공해야 한다.

```
for (a;b;c)    <- 위반. (a; b; c)로 변경
Hello(a,b,c)   <- 위반. (a, b, c) 로 변경
int k = 2+3;   <- 위반. 2 + 3 로 변경
c+++c;         <- 위반. c++ + c 로 변경
```

아래와 같이 사용한다.

```
int k = (2 + 3);     <- OK. '(' 앞과 뒤에 공백이 없다.
int k = -2;          <- OK. minus는 minus값을 의미하기 때문에 이 규칙과 상관 없다.
for (a; b; c) {}     <- OK
```

```
Hello(a, b, c);       <- OK
tt[c++]               <- OK.
```

18. word 주위에 공백을 둔다.

함수 범위 내에서 if, else, for 단어의 앞과 뒤에 공백을 둔다.

switch와 while은 바로 다음에 "("를 사용한다.

```
void function()
{
    for(k;j;c) {      <== Violation. 'for (k;j;c)' 로 변경
    }
    if(k) {           <== Violation. 'if (k)' 로 변경
    }else {           <== Violation. '} else' 로 변경
    }
}
```

아래와 같이 사용한다.

```
#define KK for(a;b;c)    <== 함수 범위 내가 아니기 때문에 상관 없다.

void function() {
    for (k;j;c) {        <== OK
    }
    if (k) {             <== OK
    } else {             <== OK
    }
}
```

19. 한 라인에 120자를 넘지 않는다.

```
int
HEEEEEEEEEEEEEEEEEELLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO = 1;
<== 위반. 너무 길다.
```

아래와 같이 사용한다.

```
int K;       <== OK. 짧다.
```

20. namespace의 brace({)는 분리된 라인에 작성한다.

```
namespace AA {            <== ERROR
}
```

아래와 같이 사용한다.

```
namespace
{
}
```

21. 함수 정의의 brace({)는 분리된 라인에 작성한다.

```
void A() {                <== 위반

}

void A()
{
    }                          <== 위반. 다른 컬럼에 있다.
```

아래와 같이 사용한다.

```
void A()
{                         <== OK
}

void K()
{
    while(True) {         <== 상관 없다.
    }
}
```

22. 타입 정의의 brace({)는 분리된 라인에 작성한다.

```
class K() {             <== ERROR
}

struct K {              <== ERROR
}
```

아래와 같이 사용한다.

```
struct A()
{                       <== OK
}

class K()
{ <== CORRECT
public :
    void Hello() {      <== 함수 정의은 상관 없다.
    }
}
```

23. 함수 내부의 brace({)는 라인 끝에 위치시킨다.

```
void A() {              <== 상관 없다.
    for (;;)
    {                   <== ERROR
    }
}
class K()
{                       <== 상관 없다.
    if (true)
    {                   <== Error
    }
}
```

아래와 같이 사용한다.

```
void A() {              <== 상관 없다.
    for (;;) {              <== OK
    }
}
class K()
{                       <== 상관 없다.
    if (true) {             <== OK
    }
}
```

24. 함수 내부에서의 들여쓰기 블록

```
void A() {
    for (;;)                <== 위반
        {
        }
}

void K()
{                           <== 함수 내부가 아니기 때문에 상관 없다.
    if (true)
    {
        if (KK) {
        AA;                 <== 위반
        }
    }

    switch(TT) {
    case WEWE:              <== 위반
        WOW;
    }
}
```

아래와 같이 사용한다.

```
void K()
{
    if (true)
    {
        if (KK) {          <== OK
            AA;            <== OK
        }
    }

    switch(TT) {          <== OK
        case WEWE:        <== OK
            WOW;
    }
}
```

25. 함수 내부에서 닫는 brace(})는 같은 컬럼에 위치시킨다.

```
void A() {            //<== 상관 없다.
    for (;;)
    { <== ERROR
      }
}
class K()
{                     //<== 상관 없다.
    if (true)
    {                //<== Error
  }
}
```

아래와 같이 사용한다.

```
void A() {          //<== 상관 없다.
    for (;;)
    {              //<== OK
    }
}
```

26. 한 문장이라도 brace를 사용한다.

```
void Function() {
    for (;;)
        print("WOW");      //<== 위반

    while(i > 7)
        i++;              //<== 위반
}
```

아래와 같이 사용한다.

```
void Function()
{
    for (;;)
    {
        print("WOW");    //<== OK
    }
    while(i > 7) {      //<== OK
        i++;
    }
}
```

27. class 정의에 doxygen 주석을 제공한다.

각 클래스 정의 앞에 doxygen 스타일의 주석을 사용한다.

```
class A {    //<== 위반. No doxygen comment.
};

/*          //<== 위반. doxygen 주석이 아니다.
 *
 */
class B {
};
```

아래와 같이 사용한다.

```
/**
 * blar blar
 */
class A {           //<== OK
};

class B;            //<== 전방 선언은 상관 없다.
```

28. namespace 정의에 doxygen 주석을 제공한다.

각 namespace 키워드 앞에 doxygen 스타일 주석을 사용한다.

```
namespace AA        //<== 위반. namespace AA에 대한 주석이 없다.
{
}

/*                 //<== 위반. 주석이 있지만, doxygen 주석이 아니다.
* blar blar
*/
namespace BB
{
}
```

아래와 같이 사용한다.

```
/**                 <== OK!
 * blar blar
 */
namespace AA
{
}
```

29. struct 정의에 doxygen 주석을 제공한다.

각 struct/union 정의 앞에 doxygen 스타일 주석을 사용한다.

```
struct A {   //<== 위반. 주석이 없다.
};

/*           //<== 위반. doxygen 주석이 아니다
 *
 */
union B {
};
```

아래와 같이 사용한다.

```
/**
 * blar blar
 */
struct A {      //<== OK
};

struct A;       //<== 전방 선언은 상관 없다.
```

30. 헤더파일의 함수에 doxygen 주석을 제공한다.

헤더파일의 각 함수 앞에 doxygen 주석을 사용한다.
private 이 아닌 함수만 작성해도 좋다.

```
= a.h =
void FunctionA();    //<== 위반. 주석이 없다.

/*                  //<== 위반. doxygen 주석이 아니다
 *
 */
void FunctionB();
```

아래와 같이 사용한다.

```
= a.h =
/**
```

```
 * blar blar
 */
void FunctionA();         //<== OK

/**
 * blar
 */
void FunctionB() {       //<== OK.
}

class A {
private :
    void FunctionC();    //<== private 함수이기 때문에 상관 없다.
}
= a.c =
void FunctionD();        //<== c 파일에 정의되었기 때문에 상관 없다.
```

31. 구현부(cpp)에서 함수에 doxygen 주석을 제공한다.

static/private 함수가 아닌 함수들에 대해 doxygen 주석을 작성한다.

cpp파일에 함수 정의부가 구현되어 있으면, private 함수인 경우 오른쪽에 '// NS'라고 적는다.

예)

```
= a.cpp =
void KK::C()  // NS
{
}
```

```
a.cpp =
void FunctionA() {    //<== 위반. 주석이 없다.
}

/*                   //<== 위반. doxygen 주석이 아니다.
 *
 */
void FunctionB()
{
}
```

아래와 같이 사용한다.

```
= a.cpp =

/**                        //<== OK
 * blar blar
 */
void FunctionA()
{
}

/**
 * blar blar
 */
```

```
void FunctionB();        //<== OK.

class A {
private :
    void FunctionC() {  //<== private 함수이기 때문에 상관 없다.
    }
}
static void FunctionD() //<== c style private 함수이기 때문에 상관 없다.
{
}
= a.h =
void FunctionB();      //<== 헤더에 선언되었기 때문에 상관 없다.
```

32. 함수 파라미터 이름은 생략하지 않는다.

함수 선언시 파라미터 이름은 생략하지 않는다. 함수 선언부만 체크한다.

```
void functionA(int a, int); //<== 위반. 두번째 파라미터 int의 이름이 없다.

void functionB(int );       //<== 위반. 첫번째 파라미터 이름이 없다.
```

아래와 같이 사용한다.

```
void functionA(int a, int b, int c, int d, int e); //<== Good.

void functionB(int, int, int c, int d)             //<== 함수 정의부는 상관 없다.
{
}
```

33. 함수에 파라미터 5개 이상은 사용하지 않는다.

각 함수에 5개 이상의 파라미터를 사용하지 않는 대신, 구조체를 사용한다.

```
void functionA(int a, int b, int c, int d, int e, int j); //<== 위반, 5개 이상

void functionB(int a, int b, int c, int d, int e, int j, int k)  //<== 위반
{
}
```

아래와 같이 사용한다.

```
void functionA(int a, int b, int c, int d, int e); //<== Good. 5 parameters.

void functionB(int a, int b, int c, int d)         //<== Good. 4 parameters
{
}
```

34. 함수는 200라인을 넘지 않는다.

함수는 공백은 포함하지 않고 200라인 이상 작성하지 않는다.

35. system 의존적인 타입을 사용하지 않는다.

```
int k;
short b;
long t;
```

아래와 같이 사용한다.

```
int32_t b;       //<== Good
```

36. enum의 첫번 째 아이템은 초기화 한다.

```
enum A {
    A, B        //<== 위반
}
```

아래와 같이 사용한다.

```
enum A {
    A=4, B     //<== OK
}
```

37. 매크로 상수는 대문자로 작성한다.

```
#define Kk 1         //<== 위반. 소문자 'k'가 사용되었다.
#define tT "sds"    //<== 위반. 소문자 't'가 사용되었다.
```

아래와 같이 사용한다.

```
#define KK 3         //<== OK. KK는 대문자이다.
#define kk(A) (A)*3 //<== 매크로 함수는 상관 없다.
```

38. 상수 선언은 매크로를 사용하지 않는다.

대신, enum이나 const 변수를 사용한다.

그러나, 매크로 함수는 사용해도 좋다. 매크로가 언더바(_)로 시작한다면, 특별한 목적을 위해 정의되었다고 간주한다.

```
#define KK 1         //<== 위반
#define TT "sds"    //<== 위반
```

아래와 같이 사용한다.

```
#define KK(A) (A)*3       //<== macro 함수는 상관 없다.
const int k = 3;          //<== OK
const char *t = "EWEE";   //<== OK
```

39. double 대입은 사용하지 않는다.

```
k = t = 1;        //<== 위반. double assignments are used.
void a() {
    b = c = 2;    //<== 위반. double assignments are used.
}
```

아래와 같이 사용한다.

```
k = 1;           //<== OK
t = 1;
void a() {
    b = 2;
    c = 2;
}
```

40. 물음표 키워드(?)는 사용하지 않는다.

```
void a() {
    c = t ? 1 : 2;      //<== 위반. ? 키워드 사용되었다.
}
```

아래와 같이 사용한다.

```
void a() {
    if (t) {            //<== OK.
        c = 1;
    } else {
        c = 2;
    }
}
```

41. goto 문을 사용하지 않는다.

```
void FunctionA()
{
    while(True)
    {
        goto AAA;      //<== 위반. goto문이 사용되었다.
    }
AAA:
}
```

아래와 같이 사용한다.

```
void FunctionA()
{
    while(True)
    {
        break;        //<== OK.
    }
}
```

42. 파일 정보에대한 주석을 제공한다.

copyright를 포함한 파일 주석을 파일의 상단에 작성한다.

```
= start of file =
#define "AA"      //<== 위반. 파일 처음은 파일 주석으로 시작해야 한다.
///
/// blar blar
/// Copyright reserved.
///

= start of file =
/**         //<== 위반. copyright 문구가 없다.
* blar blar
* blar blar
*/
```

아래와 같이 사용한다.

```
= start of file =
///
/// blar blar
/// Copyright reserved.  <== OK
///

= start of file =
/**
* blar blar
* Copyright reserved.   <== OK
* blar blar
*/
```

43. include 경로를 하드코딩하지 않는다.

```
#include "c:\Hello.h"           //<== 위반. Window style 절대 경로
#include "/usr/include/Hello.h" //<== 위반. Linux style 절대 경로
```

아래와 같이 사용한다.

```
#include "Hello.h"
#include "include/Hello.h"
```

44. use reentrant function

Use reentrant functions. Do not use not reentrant functions.(ctime, strtok, toupper)

```
void A() {
    k = ctime();          //<== Violation. ctime() is not the reenterant function.
    j = strok(blar blar); //<== Violation. strok() is not the reenterant function.
}
```

아래와 같이 사용한다.

```
void A() {
    k = t.ctime();       //<== Correct. It may be the reentrant function.
}

void A() {
    k = ctime;          //<==  Correct. It may be the reentrant function.
}
```

45. 너무 깊은 블록은 피한다.

함수 내에서 block 깊이가 4 이상이 되지 않게 한다.

```
void f() {
    {{{{{        //<== 위반. 너무 깊다. 4 block 이상이다.

    }}}}}
}
```

아래와 같이 사용한다.

```
void f() {
    {{{{       //<== OK!

    }}}}
}
```

<참조>

1. http://dev.naver.com/projects/nsiqcppstyle
2. http://nsiqcppstyle.appspot.com/rule_doc/