Coding Rule(Style)
코딩룰(스타일) ver.1.0

2012.11.01

생산자동화팀

권수한 대리

UbiSam

이력

버전	바꾼 날짜	바꾼 내용	바꾼 사람
1.0	2012.11.01	주석규칙 추가	권수한 D

Ubisam's C++ 코딩 스타일

¶ II. 규칙으로 작성한 프로그램 코드의 예

```
void CDataSocket::Init(CWnd *pWnd)
{
    m_pWnd = pWnd;
    m_pFile = new CSocketFile(this);
    m_pArchiveIn = new CArchive(m_pFile, CArchive::load);
    m_pArchiveOut = new CArchive(m_pFile, CArchive::store);
}

void CDataSocket::Receive(CData * pData)
{
    if(m_pArchiveIn != NULL)
    {
        pData->Serialize(*m_pArchiveIn);
    }
}

void CDataSocket::operator >>(CData & data)
{
    Receive (&data);
}


void CDataSocket::Send(CData * pData)
{
    if(m_pArchiveOut != NULL)
    {
        pData->Serialize(*m_pArchiveOut);
    }
}
```

```
void CDataSocket::OnReceive(int nErrorCode)
{
    m_pWnd->SendMessage(UM_DATARECEIVE);
    CSocket::OnReceive(nErrorCode);
}

void CDataSocket::OnClose(int nErrorCode)
{
    if(m_pFile != NULL) {
        delete m_pFile;
        m_pFile = NULL;
    }
    if(m_pArchiveIn != NULL) {
        m_pArchiveIn->Abort();
        delete m_pArchiveIn;
        m_pArchiveIn = NULL;
    }
    if(m_pArchiveOut != NULL) {
        m_pArchiveOut->Abort();
        delete m_pArchiveOut;
        m_pArchiveOut = NULL;
    }

    CSocket::OnClose(nErrorCode);
}
```

Ubisam's C++ 코딩 스타일

¶ 1. 이름

1.1 변수의 표기는 헝가리안 표기법을 사용한다.

헝가리안 표기법이 타이핑하기 불편하거나 귀찮을 수도 있으나 근본적으로 변수의 명확성을 강화한다.

예) `int nNumber, char chName, CString strInfo, szInfo`
`CSurface clsSurface, int *pnData, CWnd clsWndMain`
`CSurface *pclsSurface`

1.2 각 변수 및 함수의 이름은 대문자로 시작한다.

소문자만을 이용하거나 '_'을 이용하거나 또는 소문자 + 첫 단어 대문자를 이용하는 많은 사용 예가 있기는 하지만 결국 여러 스타일을 동일 프로젝트에서 혼용하여 사용하는 것은 통일성과 가독성 등에 저해를 가져온다. 때문에 다양한 방법 중 한가지를 선택할 필요성을 느꼈고, 첫 문자를 대문자로 쓰는 것으로 통일하였다.

예) `int nTypeMax, char szFileInfomation, (O)`
`int ntype_max, char szfile_infomation (X)`
`int GetType(); BOOL SetMaxRange(int CX, int CY), void SetFont(HFONT hFont)`

1.3 헝가리안 표기법 중 포인터는 중첩하여 사용 가능하다.

포인터 뿐만이 아니라 배열의 'a' 나 다른 의미의 명확성을 위해서 몇몇 헝가리안 표기법도 중첩사용이 가능하다.

예) `int *pnCnt, CSurface *pclsSurface, BYTE *pBuf`

Ubisam's C++ 코딩 스타일

1.4 헝가리안 표기법의 예외는 다음과 같다.

•X, Y 와 같은 의미가 명확한 변수 (확장 : (type)DestX, (type)DestY)
•For 문 등에서 사용하는 i, j 등의 이미 알려진 변수
•구조체 멤버 변수
•BYTE *pBuffer 와 같이 BYTE 라는 값이 큰 의미가 없는 경우
•구조체 변수의 경우
(헝가리안 표기법이 만능도 아니고 절대적인 것도 아니다. 단순히 습관적이고 일반화된 선례 등은 예외가 될 수 있다.)

1.5 전역 변수는 'g_' 을 사용한다.

지역변수 및 인자와 구분을 위해서 'g_'을 사용한다. 참고로 전역 변수는 가능한 사용을 자제한다.

예) `int g_nGameTimer, char g_szAppName`

1.6 클래스의 멤버 변수는 'm_' 을 사용한다.

지역변수 및 인자와 구분을 위해서 'm_'을 사용한다.

예) `m_surBack, m_szName, m_dwCode`

Ubisam's C++ 코딩 스타일

1.7 포인터(*) 및 참조(&) 표기는 변수 앞에 붙여서 선언한다.

변수 앞에 붙여서 표기 하는 가장 중요한 이유는 `char* szName, szType` 과 같이 표기 할 경우 `szType` 이 포인터인지 아닌지 구분이 잘되지 않기 때문이다.

올바른 예)

```
POINT *pPoint, char *szName;
POINT &pPoint, char &szName;
```

잘못된 예)

```
POINT* pPoint, char * szName;
POINT& pPoint, char & szName;
```

1.8 `#define enum` 및 `const` 는 대문자와 단어 사이에 '_' 를 붙인다.

예)

```
#define WM_USER_PING_THREAD_END                WM_USER+0x102   // Tab으로 간격 맞춤
#define PACKET_LOGIN_ANSWER_BAD_USER_OVER      8
enum BUTTON_STATE{ BUTTON_NORMAL, BUTTON_OVER, BUTTON_DOWN, BUTTON_DISABLE };
struct POINT { POINT_X, POINT_Y };
```

Ubisam's C++ 코딩 스타일

1.9 #define 보다 const 형 또는 static const 형을 되도록 사용한다. (#define 과 같은 대문자 선언 시에 사용)

하지만 배열 등에 직접적으로 const 형은 컴파일 오류가 발생하기 때문에 현재 보류한다.)

예) `static const int MAX_CARD = 1000`

1.10 클래스 내부에서 사용하는 상수 값은 클래스 내부에서 선언된 enum을 사용한다.

#define은 전역적으로 사용된다. 따라서 MAX_ITEM과 같은 일반적인 상수이름은 다른 곳에서 겹칠 우려가 있기 때문에
사용하기가 꺼려진다. 하지만 enum은 클래스 내부에서만 사용되기 때문에 MAX_ITEM와 같은 일반화된 상수를 사용할 수가 있다.

1.11 구조체 이름은 대문자로 하고 각 단어 구분은 '_' 로 한다.

구조체의 이름은 관습적으로 대문자만을 사용한다.

예) `struct TEST{ … }`
`struct BANNER_UNIT{ … }, struct GAME_ITEM{ … },`

Ubisam's C++ 코딩 스타일

~~1.12 구조체의 멤버 변수는 소문자를 이용하고 단어 별 구분 시에 '_'을 사용한다.~~

~~헝가리안 표기법이 사용되지 않는 이유는 오랜 기간 관습적으로 소문자 표기법이 이용되었기 때문이다.~~

~~예) struct GAME_UNIT~~
~~{~~
~~BYTE type;~~
~~char id[13];~~
~~WORD version;~~
~~BYTE* buffer;~~
~~CString ip_name,~~
~~};~~

1.13 일반적으로 지역 변수는 선언된 타입과 동일한 이름을 사용한다.

클래스나 구조체등의 이름을 이용하여 지역 변수를 선언하면 알아보기가 쉽고 변수명 작명을 위해 고민하지 않아도 된다.
단, 클래스의 경우 인스턴스 명칭 앞에 cls를 붙인다.

예) `BANNER_UNIT BannerUnit, FAST_TABLE FastTable, CSkyPwd clsSkyPwd`

Ubisam's C++ 코딩 스타일

1.14 템플릿 타입은 대문자 한 자로 표기한다.

관습에 의하여 템플릿은 한 자로 표기한다.

예) `template<class T>, template<class C, class D>`

1.15 클래스를 이름은 첫 자를 C 로 시작한다.

클래스를 구분하는 헝가리안 표기법의 일종이다.

예) `CSurface, CPoint`

1.16 클래스 함수의 경우 클래스 명에서 이미 명시된 부분은 제외한다.

이미 클래스 이름으로도 충분히 알 수 있는 내용을 중복하여 적는 것은 비효율적이며 올바르지 않는 방법이다. 이것은 마치 "사각형의 사각형 가로크기는 얼마인가요?" 와 같은 표현이다.

올바른 예) `Rect.GetWidth()`
잘못된 예) `Rect.GetRectWidth()`

Ubisam's C++ 코딩 스타일

1.17 역 BOOL형 변수는 사용하지 않는다. (역방향형)

빠른 이해와 잘못된 사용을 방지하기 위해서 BOOL 은 항상 '예' 인 값을 사용한다. 역 BOOL 형 변수는 '사람이 아니다가 아니다' 와 같은 복잡한 상황을 연출하기 때문이다.

올바른 예) `BOOL bError`
잘못된 예) `BOOL bNoError`

1.18 Enum의 경우 열거형으로 사용시 변수명의 앞에 표기한다.
(단, 클래스 내부에서 #define의 대용으로 사용시에는 예외이다.)

enum을 열거형으로 사용시 타입을 항상 표기함으로써 값들이 사용시 어떠한 열거형 인지 구분을 쉽게 해준다.

예)
```
enum COLOR
{
    COLOR_RED,
    COLOR_GREEN,
    COLOR_BLUE,
};
```

Ubisam's C++ 코딩 스타일

1.19 함수명의 경우 동사가 앞에 오고 명사가 뒤에 오도록 한다.

영문법에 맞는 규칙으로 통일되게 사용한다.

예) `GetLine(), PlaySound(), MoveWindow()`

1.20 BMP나 MP3, ID, XML등의 약어의 표현은 첫 대문자로 한다.

예) `CBmpFile, m_Mp3, strUserId, CXml`

1.21 변수는 한 줄에 하나씩 선언하고. 초기화는 반드시 한다.

예)
```
int m_nIndex = 0;
Cstring strString = “”;
```

Ubisam's C++ 코딩 스타일

¶ 2. 일반

2.1 `bool, true, false` 는 사용하지 않고 `BOOL, TRUE, FALSE` 로 사용한다.

`BOOL` 등은 4바이트 자료형이고 `bool` 등은 1비트(실제로는 1바이트) 자료형이다. 32비트 운영체계의 경우 속도와 비교하는데 걸리는 성능 부분에도 유리하고 알아보기도 좋기 때문에 위와 같은 대문자로 된 자료형을 사용한다.

2.2 한 라인이 너무 길지 않도록 주의한다.

정확히 컬럼의 크기가 정해진 것은 아니지만 일반적인 프로젝트 팀에서 사용하는 일반적인 해상도 (1024~1280) 에서 한눈에 보기 좋도록 작성한다.

2.3 성능 향상 및 가독성 증가를 위해서 본 가이드라인은 위배 될 수 있다. (단! 확고한 근거가 존재해야 한다)

가이드라인의 목표는 보기 좋고 통일된 코드작성이 가장 중요한 목표이다. 특수한 경우 가독성 이나 성능 향상을 위해서 더 좋은 방법이 있다. 그러한 경우에는 본 가이드라인을 따르지 않는다.

Ubisam's C++ 코딩 스타일

2.4 변수의 형 변환은 (자료형) 과 같은 변환을 사용한다.

일반적으로 여러 서적과 C++ 권고안에서 static_cast, const_cast등의 cast를 이용한 형 변환을 권고하고 있으나 타이핑의 불편, 가독성의 감소, 다양한 cast의 적절한 사용 등으로 인한 이유로 사용하는데 어려움이 있다.

예)

```
int nSize = (int)byLength; (O)
int nSize = static_cast<int> byLength; (X)
```

2.5 Call by Reference 함수 호출시에 인자는 NULL 입력이 가능하다면 포인터(*)를 이용하고 그렇지 않다면 참조(&)을 사용한다.

포인터 방식의 최대 단점은 NULL을 원하지 않은 경우에도 NULL이 입력될 수 있다는 점이다. 따라서 이러한 것을 근본적으로 막기 위해서는 NULL이 입력되지 않기를 원하는 함수의 인자는 참조를 사용한다.

예) `int Draw(CSurface &surface);` (NULL이 입력 불가한 경우)

```
   void GetValue(int *pId, int *pAge, int *pDate = NULL);
```
(NULL이 입력 가능한 경우)

Ubisam's C++ 코딩 스타일

2.6 전역 함수, API 함수, 표준 C++ 라이블러리의 경우 일반적으로 '::' 을 가능한 사용하지 않는다.

멤버 함수와 동일한 이름인 경우에만 '::' 을 사용한다, (이와는 별개로 전역 함수의 사용은 최소한 한다.

올바른 예) `int nCount = ::GetWindowCount();` `int nSize = strlen( szName );`
잘못된 예) `int nCount = GetWindowCount();` `int nSize = ::strlen( szName );`

2.7 STL 사용시 std:: 는 사용하지 않는다.

프로젝트에서 STL(Standard Templete Library)을 표준으로 사용하기를 결정했다면 'using std;'을 사용한다.

올바른 예) `std::list List`
잘못된 예) `list List`

2.8 지역변수는 함수 시작부분에 선언 및 초기값을 지정한다.

Ex) int i = 0;
CString str = _T("");
FILE *pFile = NULL;

Ubisam's C++ 코딩 스타일

2.9 매직 넘버(Magic Number)는 가능한 사용하지 않는다.

가능한 #define 이나 `enum, static const` 등을 이용하여 매직 넘버의 사용을 최소화한다.
개수, 최대값, 최소값, 범위 등의 매직 넘버는 사용을 최대한 자제하나 오히려 가독성과 사용용도가 떨어지는 이미지의 좌표값과 같은 일시적인 것은 특별히 사용을 회피 할 필요는 없다. 또한 `0, 1, -1`의 경우는 매직 넘버에서 예외가 될 수 있다.

Ex)

```
#define 1 one;
#define 2 two;

Int n;

Switch(n)
{
   case one:
   case two:
}
```

2.10 `goto` 의 사용을 자제한다.

`goto` 의 사용은 프로그램의 구조를 파악하기 어렵게 한다. 하지만 매우 특수한 경우에는 사용 될 수 있다.

Ubisam's C++ 코딩 스타일

2.11 전역 변수나 멤버 변수보다는 지역변수를 사용한다.

전역 변수와 멤버 변수는 매우 오랜 기간 동안 생존해있고 중요한 변수이기 때문에 작성자를 포함한 프로그래머들은 그 변수들을 분석하고 기억하려고 한다. 따라서 사용을 최소화 한다면 그러한 수고가 줄어든다.

2.12 포인터, 핸들은 NULL 과 비교하고 숫자는 0 과 비교한다.

2.13 복잡한 구문은 임시 변수를 사용한다.

부적절한 3 항 연산자나 각종 비교구문의 중첩을 이용한 복잡한 코드는 가독성을 저해한다.

올바른 예)

```
BOOL bCompare1 = (nCount > 100 && nSize < 50);
BOOL bCompare2 = (nCount == 50 || nSize < 20);
if(bCompare1 || bCompare2)…
```

잘못된 예)

```
if((nCount > 100 && nSize < 50) || (nCount == 50 || nSize < 20))…
```

Ubisam's C++ 코딩 스타일

¶ 3. 클래스 (class)

3.1 클레스의 기본 구조는 다음과 같다.

생성자, 공개함수, 비공개함수, 멤버변수의 순서로 배열하고 각각은 public, protected 등으로 추가로 명기하여 구분한다.

예)
```
class CSound
{
public:                         : public 함수를 최 상단에 배치한다.
    CSound();                   : 생성자를 최 상단에 배치한다.
    virtual ~CSound();          : 종결자는 일반적으로 virtual 이다.

    void Play();                : 멤버 함수
    void Stop();                : public, protected 등은 1라인 비운다.
protected:                      : protected 함수를 배치한다.
    void PlayWav();
    void PlayOgg();             : 각각의 함수별 관련성 및 블록이 있다면 1라인 비운다.
    void StopWav();
    void StopOgg();

protected:                      : 멤버 변수를 배치한다.
    char m_szFileName[240];
    int m_nSize;
};
```

Ubisam's C++ 코딩 스타일

3.2 모든 멤버 변수는 public으로 선언하지 않는다.

정보은닉과 캡슐화를 위해서 멤버 변수들은 public으로 선언하지 않는다.

3.3 inline 함수의 경우 헤더 선언에 inline 을 표기하지 않는다.

헤더 파일에 inline으로 명기하여 선언을 하는 것과 별개로 실제 구현에 inline으로 작성하면 함수는 inline으로 작동된다. 클래스 헤더를 잘 알아볼 수 없게 하는 inline은 굳이 사용 할 필요성이 없다.

예) xxx.h

```
class CSound
{
    올바른 예) BOOL Play();
    잘못된 예) inline BOOL Play();
};
```

Ubisam's C++ 코딩 스타일

3.4 `inline` 함수의 경우 선언구문 1라인 아래 줄에 입력한다.

`inline` 함수는 헤더 파일의 하단에 선언하고 너무 많은 `inline` 함수가 존재한다면 `*.inl` 파일을 작성하여 헤더에서 include 한다.

예)
```
class CSound
{
public:
    BOOL CSound ::Play();
};

inline BOOL CSound ::Play()
{
    …
}
 --- 또는 ---
class CSound
{
public:
    BOOL CSound ::Play();
};

#include "Sound.inl"
```

Ubisam's C++ 코딩 스타일

3.5 클래스 선언문에 어떠한 함수라도 구현을 하지 않는다.

(클래스의 선언문은 매우 중요한 공간이다. 그 공간에 함수의 구현이 들어간다면 함수의 이름들을 한눈에 볼 수 없어 가독성이 매우 떨어지게 된다. 따라서 모든 구현은 cpp 파일 및 별개의 인라인 함수로 구현한다.)

잘못된 예) xxx.h

```
class CTest
{
    BOOL Play()
    {
        …
    }
};
```

3.6 클래스의 생성자 및 종결자는 inline함수로 구현하지 않는다.

성능향상을 위한 몇몇 경우를 제외하면 생성자와 종결자는 호출되는 회수는 매우 적다. 따라서 inline으로 구현하지 않는다.

3.7 종결자는 일반적으로 virtual로 선언한다.

종결자가 virtual이어야 상속될 경우 등에서 올바르게 종결자가 작동된다. 따라서 상속이 되지 않기를 바라는 경우와 클래스 크기를 줄이기 위한 경우 및 성능향상을 위한 경우를 제외하고 가상함수로 선언한다.

Ubisam's C++ 코딩 스타일

¶ 4. 파일 및 디렉토리

4.1 클래스명과 파일명은 'C'를 제외한 동일한 이름을 가진다.

예) 클래스명 : `CFileName`
파일명 : `FileName.h, FileName.cpp`

4.2 1클래스 2파일(.h .cpp)을 가진다.

4.3 클래스의 복잡도에 따라서 여러 개의 cpp 파일을 가질 수 있다.

(종종 멤버 함수가 많다면 함수의 검색 및 수정 등이 불편하여 관리가 어려워진다. 그러한 경우 비슷한 함수 별로 파일을 만든다면 쉽게 관리가 가능하다.)

예) 클래스명 : `CSurfaceGdi`
파일명 : `SurfaceGdi.h, SurfaceGdiPut.cpp SurfaceGdiInit.cpp, SurfaceGdiEffect.cpp`

Ubisam's C++ 코딩 스타일

4.4 Inline 함수가 많이 존재할 경우 `'.inl'` 파일을 생성한다.

예) `FileName.inl`

4.5 매우 간단하면서 비슷한 여러 개의 클래스 및 구조체의 경우 1개의 파일에 포함될 수 있다.

(작은 클래스, 구조체마다 각각의 파일을 가진다면 파일개수의 복잡성이 증가한다.)

4.6 문서(기획서, UML등) 파일은 소스파일과 다른 특정 디렉토리를 사용한다. (예:Document)

(각종 문서들을 소스파일과 함께 관리한다면 더욱더 복잡한 파일목록들이 만들어질 것이다.)

Ubisam's C++ 코딩 스타일

4.7 일정 규모이상의 프로젝트의 시작은 폴더구조를 설계한 이후 시작한다.

(프로젝트 시작 시 폴더의 구조와 각 폴더 별 연관성 등을 고려하여 폴더구조를 정의하고 상호 참조 관계 등을 정의하는 것으로 프로젝트를 시작해야 한다.)

예)
- [App]
- [Common]
- [Core]
- [Sound]
- [Image]
- [Document]

4.8 #include 시 상대주소를 사용한다. 예) #include '../BugGame/BugMain.h'

(절대주소를 사용한다면 다른 프로그래머들은 컴파일이 안될 수도 있기 때문이다.)

Ubisam's C++ 코딩 스타일

4.9 #include 는 시스템 헤더 -> 라이브러리 -> 자체 코드 등의 순서로 작성한다.

#include 또한 상위 -> 하위 관계를 지키도록 하면 가독성이 높아진다.

예)

```
#include 'stdio.h'
#include 'afxinet.h'
#include 'MyImageEngine.h'
#include 'MyFile.h'
```

4.10 백업은 다른 디스크드라이브에 받고 일정기간마다 CD 또는 별개의 매체에 저장한다.

백업의 중요성은 아무리 강조해도 지나치지 않다.

4.11 프로그램 코드, 이미지, 사운드를 포함한 모든 소스는 소스세이프(SourceSafe) 및 Subversion(SVN)등의 소스 관리를 사용한다(권장사항 - 관리자는 죽을지도 모른다).

다른 프로그래머가 소스 관리을 이용하더라도 컴파일과 실행을 할 수 있게 한다.

Ubisam's C++ 코딩 스타일

¶ 5. 레이아웃

5.1 변수 또는 함수 선언 시 TAB으로 정렬하지 않는다(선택사항).

올바른 예)

```
    int nCount;
    int nSize;
    BOOL bSuccess;
    CString strName;
```

잘못된 예)

```
    int         nCount;
    int         nSize;
    BOOL        bSuccess;
    CString     strName;
```

Ubisam's C++ 코딩 스타일

5.2 가독성을 이유로 다음과 같은 정렬 방식은 가능하다(한 화면에 보이지 않는 경우).

예)

```
int nSum = a + b + c +
        d + e;
if(nCount > 100 && nSize > 50 &&
    nHit == 5)
```

5.3 TAB의 크기는 4글자로 한다.

5.4 함수마다 1~2라인을 띄운다.

Ubisam's C++ 코딩 스타일

5.5 '+, -, *, %, /, <, >, =, ==, +=, -=' 와 같은 2개의 변수를 계산하거나 비교하는 경우에는 연산자의 전후는 1칸씩 띄어 쓴다.

변수 단독 붙여서 사용하는 '++, --, !, &(참조), *(포인터)'와 같은 연산자의 경우는 예외이다.

올바른 예)

```
i = a + b – c;
nSum += nCount;
if(nSize < nMaxSize)
i++;
```

잘못된 예)

```
i=a+b–c;
nSum+=nCount;
if(nSize<nMaxSize)
i ++;
```

5.6 일반적으로 i++ 은 사용이 가능하지만 ++i 는 지양한다(선택사항).

속도를 위해서 후자를 사용하는 사례들이 가끔 있지만 사실상 컴파일러의 최적화로 인하여 위 내용의 속도의 차이가 존재하지 않는다. 따라서 가독성과 통일성을 이유로 전자만 사용한다.

Ubisam's C++ 코딩 스타일

5.7 소괄호의 사용은 다음과 같이 공백입력을 하지 않음. (함수, if, while, for, switch, 형 변환 등 모두 동일함) (선택사항)

예)

```
if(bSuccess == TRUE) …
int DoSomething(int nParam1, int nParam2);
for(int i = 0; i < nCount; i++)
int nIntSize = sizeof(int);
int nValue = (int)yValue;
```

이렇게 쓰는 것도 괜찮다.

```
if( bSuccess == TRUE ) …
int DoSomething( int nParam1, int nParam2 );
for( int i = 0; i < nCount; i++ )
int nIntSize = sizeof( int );
```

Ubisam's C++ 코딩 스타일

5.8 중괄호의 사용은 다음과 같은 형식으로 한다. (함수, if, while, for, switch 모두 동일함)
조건문안의 내용이 한 줄로 되어 있어도 반드시 형식을 맞춘다.

올바른 예)

```
if(bSuccess == TRUE)
{
    DoSomething();
    …
}

int DoSomething()
{
    …
}
```

잘못된 예)

```
if( bSuccess == TRUE ){
        DoSomething();
        …
   }

   if( bSuccess == TRUE )
        {
        DoSomething();
        …
        }
```

Ubisam's C++ 코딩 스타일

5.9 가독성을 위하여 다음과 같이 TAB으로 정렬이 가능하다.

본 가이드의 핵심은 가독성이 좋은 코드를 만드는 것을 도와주는 것이다. 따라서 그것을 위해서 몇몇 규범들은 위배 될 수 있다.
하지만 아래 내용을 절대로 남용하지 않는다.

예)

```
    nValue =(nSize * nCurrentSize) / nDiv +
            (nCount * nCurrentCount) / nDiv +
            (nTime * nCurrentTime) / nDiv;

    switch(nState)
    {
        case STATE_NORMAL :
          DoNormal();
          break;
        case STATE_OVER :
          DoOver();
          break;
        case STATE_DOWN :
          DoDown();
          break;
    }
```

Ubisam's C++ 코딩 스타일

5.10 Switch 구문은 다음과 같은 형식으로 한다. (':' 의 띄어쓰기에 유의한다.)

예)

```
    switch(nState)
    {
    // case 관련 주석은 여기에…
    case STATE_NORMAL :
        // 아래 주석은 여기에…
        DoSomething();
        break;

    case STATE_OVER :
        DoSomething();
        break;

    default :
        DoSomething();
        break;
}
```

Ubisam's C++ 코딩 스타일

5.11 한 줄 조건문 및 반복문은 다음과 같이 모두 사용 할 수 있다. 하지만 동일 구문에 여러 스타일을 혼용하여 사용하면 안된다.

올바른 예)

```
// 주석은 여기에
if(bSuccess == TRUE)
{
    // 주석은 여기에
    DoSomething();
}
// 주석은 여기에
else
{
    // 주석은 여기에
    DoSomething();
}
```

잘못된 예)

```
if( bSuccess == TRUE ) DoSomething();
else
{
    DoSomething();
}

if( bSuccess == TRUE ) DoSomething();
else
    DoSomething();

if( bSuccess == TRUE )
{
    DoSomething();
}
else    DoSomething();
```

Ubisam's C++ 코딩 스타일

5.12 'return' 의 윗라인은 1줄 여백으로 한다.

올바른 예)

```
BOOL IsSuccess()
{
    BOOL bRtn = m_bRtn;

    return bRtn;
}
```

잘못된 예)

```
BOOL IsSuccess()
{
    BOOL bRtn = m_bRtn;
    return bRtn;
}
```

Ubisam's C++ 코딩 스타일

5.13 중괄호를 닫는 부분과 if, for등의 조건, 반복 구문에서는 1줄 여백을 가진다.

적절한 줄 여백은 코드를 읽기 쉽게 한다. 참고로 `if-else` 구문은 `else` 에만 적용한다.

올바른 예)

```
    if( bSuccess == TRUE )
    {
        …
    }
    else
    {
        …
    }

    nSum += nCount;
```

잘못된 예)

```
    if( bSuccess == TRUE )
    {
        …
    }
    nSum += nCount;
```

Ubisam's C++ 코딩 스타일

¶ 6. If / switch / for / while / switch 구문

6.1 If / switch / for / while 구문 자체에서 복잡한 계산을 하지 않는다.

조건문 내의 복잡한 계산은 코드의 가독성을 저해한다.

올바른 예)

```
nSum = 0;
for(int I = 0; I < 100; I++)
{
    nSum += I;
    printf( "%d", nSum );
}

hFile = open(szFileName, 'w');
if(hFile)
{
}
```

잘못된 예)

```
for( int I = 0, nSum = 0; I < 100; I++, nSum +=
I; )
{

    printf( "%d", nSum );
}

if( hFile = open( szFileName, 'w' ) )…
{
}
```

Ubisam's C++ 코딩 스타일

6.2 Do - While 은 가능한 사용하지 않는다.

(While만을 사용하는 것이 가독성이 증가된다.)

6.3 무한 루프 구문은 while(TRUE) 또는 while(1)을 사용 한다.

올바른 예)

```
while(TRUE)
{
}

while(1)
{
}
```

잘못된 예)

```
for( ; ; )
{
}

do
{
} while( 1 )
```

Ubisam's C++ 코딩 스타일

6.4 BOOL및 HRESULT 등의 리턴 값을 TRUE, FALSE, SUCCEED(), FAILED()등 과 비교한다.,

코드의 간소함보다 '!' 이 눈에 잘 띄지 않으므로 오히려 가독성을 저해하고 코드 실수를 유발할 가능성이 크므로 이와 같이 변경하였다.

올바른 예)

```
if(IsMain() == TRUE) …
if(IsMain() == FALSE) …
if(IsMain() == NULL) …
```

잘못된 예)

```
if( IsMain()) …
if( !IsMain()) …
```

Ubisam's C++ 코딩 스타일

¶ 7. 주석 (자세한 내용은 Ubisam's 주석 스타일 가이드라인 참고)

7.1 주석은 모국어(한국어)로 작성한다.

7.2 복잡성에 따라서 세부적인 설명 및 예제를 포함할 수 있다.

7.3 일반적인 주석의 경우 `/*, */` 보다는 가능한 `//` 으로 주석을 사용한다. (Doxygen의 경우를 예외로 한다)

7.4 수정 이력 관리를 위한 주석의 경우 - 필요하다면 작성하도록 한다.

Ubisam's C++ 코딩 스타일

7.5 주석은 내용의 앞 라인과 같은 컬럼에 기록한다.

몇몇 특수한 상황에서 라인 뒤에 입력하는 경우도 있다.

올바른 예)

```
// 성공했다면…
if(bSuccess == TRUE)
{
    // 어떤일을 한다.
    DoSomething();
}
```

잘못된 예)

```
// 성공했다면…
if( bSuccess == TRUE )
{
// 어떤일을 한다.
    DoSomething();
}

if( bSuccess == TRUE ) // 성공했다면…
{
    DoSomething(); // 어떤일을 한다. (X)
}
```

Ubisam's C++ 코딩 스타일

7.6 주석 - 클래스(class)

하나의 클래스는 하나의 .h 파일과 하나의 .cpp파일로 구성되는 것을 지향한다.
클래스의 설명은 .cpp파일의 상단에 간략하게 기록한다.
 - .h과 .cpp 파일에 모두 기록할 경우, 나중에 두 파일의 설명이 다를 수도 있다.
클래스 설명에 이어서 수정이력을 기록한다.

```
Class Cubisam
 - Ubisam.h
 - Ubisam.cpp

In Ubisam.cpp
// Ubisam.cpp : implementation file
// ubisam 클래스에 대한 간단한 설명
// 수정이력
// [2012/10/31] ubs SHKWON – 수정사항 설명(1줄)

#include "stdafx.h"
#include "macmic.h"
#include "AFManualDlg.h"
.
.
```

Ubisam's C++ 코딩 스타일

7.7 주석 - 함수(function)

함수 위에 간단하게 함수, 파라미터, 리턴값을 기록한다.
수정이력은 함수 설명 밑에 기록한다.

```
/*
// ubisam에 대한 설명을 하는 함수

// nNum: nNum에 대한 설명
// strStr: strStr에 대한 설명

// return: 어떤 값이 리턴되는지 설명
*/
// [2012/10/31] ubs SHKWON – 수정사항 설명(1줄)
int ubisam(int nNum, Cstring strStr)
{
    .
    .
    .
    return xxx;
}
```

Ubisam's C++ 코딩 스타일

7.8 주석 - 구조체(structure)

구조체 위에 간단하게 구조체설명, 변수설명을 기록한다.
수정이력은 구조체 설명 밑에 기록한다.

```
/*
// ubisam에 대한 설명을 하는 구조체

// nNum: nNum에 대한 설명
// strStr: strStr에 대한 설명

*/
// [2012/10/31] ubs SHKWON – 수정사항 설명(1줄)
typedef struct ubisam
{
    int nNum;
    Cstring strStr;
} UBISAM;
```

Ubisam's C++ 코딩 스타일

7.9 주석 - 변수(variable)

전역 및 멤버 변수는 .h파일에 접근 지정자(public, private, protected)별로 모아서 선언한다.
전역 및 멤버 변수는 형(int, bool, Cstring…) 별로 모으지 않아도 된다.
전역 및 멤버 변수에 대한 주석은 변수를 선언한 줄에 간단하게 기록한다.

지역 변수는 함수내의 상단에 형(int, bool, Cstring…) 별로 모아서 선언한다.
지역 변수에 대한 주석은 변수를 선언한 줄에 간단하게 기록한다.

```
Class ubisam : public Cdialog
{
   public:
      int m_nNum;         // m_nNum 설명
      BOOL m_bNum;    // m_bNum 설명

   private:
      int m_nNum;         // m_nNum 설명
      BOOL m_bNum;    // m_bNum 설명

   protected:
      int m_nNum;         // m_nNum 설명
      BOOL m_bNum;    // m_bNum 설명
}
```

```
void ubisam()
{
   int nNum = 0;    // nNum 설명
   int nNumber = 0    // nNumber 설명
   BOOL bNum = FALSE; // bNum 설명
   BOOL bNumber = FALSE;    // bNumber 설명
   Cstring strNumber = "";    // strNumber 설명

}
```

UbiSam

Ubisam's C++ 코딩 스타일

¶ 8. 권장 함수명

일반적으로 생성, 초기화하는 부분은 BOOL 을 리턴 하고 파괴, 제거, 닫는 부분은 void 형 함수로 처리한다.
권장하는 함수명 뒤에 `CreateImage`, `OpenFile`과 같이 각종 명사를 붙여서 사용 할 수 있다.

함수명		용도		
Create	**Destroy**	생성	파괴	
Open	**Close**	열다	닫다	
Init	**Final**	초기화	마지막	
Play	**Stop**	시작	정지	
Get	**Set**	얻음	넣음	
Add	**Remove**	추가	삭제	일반적으로 순서는 상관 없다
Insert	**Delete**	삽입	삭제	중간에
Start	**Stop**	시작	정지	
Suspend	**Resume**	일시 정지	재 시작	
Begin	**End**	시작	끝	
First	**Last**	처음	끝	
Next	**Previous**	이전	다음	
Increment	**Decrement**	증가	감소	
Old	**New**	이전	새로	
Up	**Down**	상	하	
Min	**Max**	최소	최대	
Show	**Hide**	보여줌	가림	

감사합니다

UbiSam