Kyuseo's 프로그래밍 규범 (Programming Pattern)

범주 C++ 팁,강좌

버전: 2.1

최종수정: 2008-01-03

출처 : http://www.soonsin.com/25

개요..

프로그래밍에서 아무리 강조를 해도 지나침이 없는 프로그래밍 규범과 코딩 스타일의 중요성에 대하여 설명을 드립니다.

프로그래밍의 기본은 프로그래밍 언어입니다. 우선 언어를 능수 능란하게 사용하게 된 이후에 설계나 알고리즘, 최적화, 방법론 등의 추가 기법들을 쉽게 배울 수 있고 다른 사람들의 언어들의 이해가 가능해집니다.

이러한 사항들은 실제의 언어 즉 한국어, 영어, 일본어등과 흡사한 모습을 가지게 되지요. 한국에서 잘 살려면 한국어를 잘 사용하는 것이 기본이지요. 또한 동일한 한국어라고 할지라도 경상도 전라도 등의 사투리가 존재하고 각 지역별로 그 지역에서 잘 살려면 그 지역의 사투리를 잘 배우고 사용해야 합니다.

여기서 한국어, 영어 등은 C++이나 Java 등으로 볼 수 있고 각 사투리들을 코딩스타일에 비유를 할 수가 있습니다. 각 언어별 물론 표준어라는 것이 있어서 의사소통의 기준을 마련하지만 코딩 스타일에서는 표준이라는 것이 존재하지 않습니다. 따라서 적어도 동일한 지역에서는 동일한 사투리만을 쓰는 것이 상호 의사소통에 큰 도움을 줄 것입니다.

따라서 동일한 회사 내에서는 최소한 동일한 팀 내부에서는 같은 코딩 스타일을 결정하고 관리하고 유지하는 것이 서로에게 오해로 인한 불신이 줄어들고 서로 만든 코드를 잘못 이용하는 상황을 감소 시킬 수가 있겠지요.

또한 이직이 잦은 IT 업계에서 하나의 보험의 역할도 할 수 있고, 하나의 코드를 여러 사람이 작업시 더욱 빛을 발하게 됩니다.

코딩 스타일과 주석이 통일되지 않은 코드들은 자물쇠가 없는 보물 상자와 같습니다.

아무리 강조하여도 지나침이 없는 코딩 스타일과 규칙들 잘 지켜 멋진 코드를 작성하시기를 기원합니다.

★ 변수 (variable)

- 전역변수와 멤버변수 등 중요한 변수는 가급적 헝가리안식 표기법을 기본으로 한다.

```
예) int nNumber, char szName, CString strInfo, CSurface surSportCar
```

- 소문자로만 이루어진 변수는 헝가리안식 표기법을 사용하지 않는다.

```
int type_max, char file_infomation
```

- 각 단어 별 구분은 대문자 또는 "_" 를 이용한다.

```
int nTypeMax, char szFileInfomation,
int type_max, char file_infomation
```

- 클래스 멤버변수는 "m_" 기호와 함께 헝가리안식 표기법과 대문자 구분을 사용한다.

```
m_surBack, m_szName, m_dwCode
```

- 글로벌(Global)변수는 "g_" 기호와 함께 헝가리안식 표기법과 대문자를 사용한다.

```
int g_nGameTimer, char g_szAppName;
```

- 구조체 멤버 변수는 헝가리안 표기법을 사용하지 않고 소문자과 포인터 표기한다.

```
struct STRUCT
{
	BYTE type;
	char id[13];
	WORD version;
	char ip_name[10];
}
```

- 지역변수는 자유롭게 사용한다.

(지역변수는 헝가리안식 표기법 생략이 가능하다.)

```
int width, int height, float percent
char szfile_path[256];  char filename[256]
int nGameTimer, char szAppName;
int n, int num, int i, j, int x, y, cx, cy, dx, dy, type, state
char filename[32], char filesize
for( int i=0; i<10; i++){...}
```

- 지역변수는 단축하여 표기하여도 좋다.

```
CFastFile ff,  CFileName fn,  BOOL b,
int n, i, j;   int X, Y, Z;
```

- 클래스 및 구조체의 변수는 자신의 이름을 사용할 경우 헝가라안식 표기법은 생략가능하다.

```
CSurfaceGdi* psurBody[10], CPoint pntCenter;
BANNER_UNIT m_BannerUnit, FAST_TABLE m_FastTable, CSkyPassword SkyPassword
```

- 포인터는 가급적 헝가리안식 표기법을 사용한다.

```
BYTE* pBuffer, CSurface* pSurface
```

★ #define enum 및 const

- #define 선언은 대문자선언과 단어 사이 "_" 를 기본으로 한다.

```
#define WM_USER_PING_THREAD_END WM_USER+0x102

#define PACKET_LOGIN_ANSWER_BAD_USER_OVER
```

- enum 선언은 대문자선언과 단어 사이 "_" 를 기본으로 한다.

```
enum BUTTON_STATE{ BUTTON_NORMAL, BUTTON_OVER, BUTTON_DOWN, BUTTON_DISABLE };
```

- #define 보다는 const 를 선호한다. (#define 과 같은 선언 구조 사용)

- 일반적인 const 는 일반 변수선언과 동일하게 사용한다.

★ 함수 (function)

- 함수의 각 단어별 구분은 대문자로 한다.

```
int GetType();   BOOL SetMaxRange( int cx, int cy ), void SetFont( HFONT
hfont )
```

- 함수인자의 선언규칙은 소문자를 원칙으로 한다.

★ 구조체 (struct)

- 구조체 선언은 대문자로 한다.

```
struct BANNER_UNIT{ … }
```

- 변수선언은 소문자를 원칙으로 한다.

★ 파일 관리 (file & directory)

- 기본적으로 1 클래스 2 파일 구조를 채택한다.

- 클래스와 파일은 "C"를 제외한 동일한 이름을 가진다.

```
클래스 : CFileName
파일   : FileName.h, FileName.cpp
```

- 클래스가 복잡하다면 다수의 cpp 파일을 가질 수 있다.

```
클래스 : CSurfaceGdi
파일   : SurfaceGdi.h, SurfaceGdiPut.cpp SurfaceGdiInit.cpp,
SurfaceGdiEffect.cpp
```

- #include 는 상대주소를 이용한다.

```
#include "../BugGame/BugMain.h"
```

- 문서(기획서, UML 등) 파일은 소스파일과 같은 디렉토리를 이용하지 않는다.

- 작은 백업은 날짜별로 압축하여 관리한다.

- 큰 백업은 별도의 디스크 드라이브에 저장한다.

- 백업은 자동화된 툴을 사용한다.

★ 주석 (comment)

- 파일 최상단에 주석을 입력한다. (프로젝트명, 시작날짜, 관리자명등)

```
/**
      @file  PSeTranslator.cpp
      @date  2005/1/18
      @author 채경석(kyuseo99@chol.com) PMangoEngine™
      @brief
*/
```

- 멤버 변수/함수의 주석은 선언 옆에 "//<" 를 이용한다.

```
DWORD* Decode( DWORD* pBuffer, DWORD* pSize );       //< 암호화 버퍼와 변경된 size리턴 (4바이트 증가)
long m_lFrameTic; ///< 1000/fps로 이시간후 (ms) 프레임을 이동한다.
long m_lNewTime; ///< 새로운 프레임 진행 시간
long m_lOldTime; ///< 이전의 프레임 진행 시간
```

★ 함수명 (function name)

- 함수명은 누구나 알 수 있는 쉬운 단어를 선택한다.

- 함수명은 올바른 짝을 이루어 사용한다.

- 생성, 초기화하는 부분은 BOOL 을 리턴하고, 파괴, 닫는 부분은 void 형 함수로 처리한다.

- 추천 함수명

Create / Destroy	창조, 생성 / 파괴
Open / Close	열다 / 닫다
Init / Final	초기화 / 마지막
Play / Stop	시작 / 정지
Set / Get	설정 / 얻기

참고:

Geosoft : http://geosoft.no/development/ (저의 것과 가장 유사함)

리차드 스톨맨 씨의 [GNU 코딩 표준]을 근간으로 여러 가지 스타일을 제시: http://people.linuxkorea.co.kr/~yong/programming/intro/Coding_Style/

모질라 코딩 스타일 : http://www.mozilla.org/hacking/mozilla-style-guide.html

Recommended C Style and Coding Standards :http://www.doc.ic.ac.uk/lab/secondyear/cstyle/cstyle.html

각종 C and C++ Style Guides 모음 : http://www.chris-lott.org/resources/cstyle/

MS Lead Program Manager of the Common Language Runtime Team : http://blogs.msdn.com/brada/articles/361363.aspx

그라비티에서 사용하셧다던 코딩 스타일 : http://blog.naver.com/iliyard/9298516

코딩 스타일 통일의 중요성 : http://blog.naver.com/iliyard/9298188

실제 사용 예)

```
/**
	@file  PTimerThread.h
	@date  2003/10/1
	@author		채경석(kyuseo99@chol.com) PMangoEngine™
	@brief
*/

#pragma once

#pragma comment( lib, "winmm.lib" )

#include "PThread.h"

/**
	@class CPTimerThread
	@date  2003/10/1
	@author		채경석(kyuseo99@chol.com)
	@brief 타이머쓰레드클래스
			- 일정시간마다인자로넘어온함수를호출한다.
			-
시간보정을하므로경과시간(Peride )을보장하나작업함수의작업시간이너무길다면(작업시간>Peride )
보정이되지않는다.
*/
class CPTimerThread
{
public:
	CPTimerThread();
	virtual ~CPTimerThread();
```

```
	BOOL Create( P_THREADPROC pfnMainProc, void* pParam, int nDueTime, int
nElapse ); ///< 생성한다.
	BOOL Create( P_THREADPROC pfnMainProc, P_THREADPROC pfnStartProc, P_THREADPROC
pfnEndProc, void* pParam, int nDueTime, int nElapse ); ///< 생성한다.
	void Destroy(); ///< 제거한다.

	void Pause(); ///< 잠시대기한다.
(스레드자체를중단시키는것은아니고다음메인함수에진입을중단을한다.)
	void Resume(); ///< 대기를수정한다.

	BOOL IsActive() const; ///< 쓰레드가활성한지알아본다.

protected:
	static UINT  WINAPI ThreadProc( void* pParam );

	P_THREADPROC m_pfnMainProc;
	P_THREADPROC m_pfnStartProc;
	P_THREADPROC m_pfnEndProc;

	HANDLE m_hEventExit;

	void* m_pParam;
	int m_nElapse;
	int m_nDueTime;

	BOOL m_bPause;

	CPThread m_Thread;
};
```

```
UINT WINAPI CPTimerThread::ThreadProc( void* pParam )
{
	UINT nRtn = 0;
	CPTimerThread* ptt = (CPTimerThread*) pParam;

	// 일정시간대기한다.
	DWORD dwWait = WaitForSingleObject( ptt->m_hEventExit, ptt->m_nDueTime );

	// 처음대기중에종료되었다면어떠한함수의호출없이바로종료한다.
	if( dwWait == WAIT_OBJECT_0 ) return nRtn;

	// 시작함수
	if( ptt->m_pfnStartProc != NULL ) ptt->m_pfnStartProc( ptt->m_pParam );
```

```
	DWORD dwStart;
	int nProgress;
	int nElapse = 0;

	while( dwWait != WAIT_OBJECT_0 )
	{
		// xx ms 마다쓰레드를돌린다.
		dwWait = WaitForSingleObject( ptt->m_hEventExit, nElapse );

		if( dwWait == WAIT_TIMEOUT )
		{
			dwStart = timeGetTime();
			if( ptt->m_bPause == FALSE )
			{
				nRtn = ptt->m_pfnMainProc( ptt->m_pParam );
			}
			nProgress = timeGetTime() - dwStart;

			// 경과시간에따른다음대기시간수정
			if( nProgress >= ptt->m_nElapse ) nElapse = 0;
			else nElapse = ptt->m_nElapse - nProgress;
		}
		else if( dwWait == WAIT_OBJECT_0 )
		{
			break;
		}
		else if( dwWait == WAIT_FAILED )
		{
			ASSERT(0);
		}
	}

	// 종료함수
	if( ptt->m_pfnEndProc != NULL ) nRtn = ptt->m_pfnEndProc( ptt->m_pParam );

	return nRtn;
}
```