Правила оформления исходных текстов.

Отступы.

Величина отступа равняется **3** пробелам, причем символ табуляции заменяется соответствующим количеством пробельных символов.

Выражения и скобки в выражениях.

Выражения и аргументы функций, заключенные между круглыми, угловыми, либо квадратными скобками, отделяются от скобок одним пробелом. Если используется пустая скобочная последовательность **()**, **{}**, **<>**, **[]**, тогда пробела между скобками не ставится. Между операторами в выражении также ставятся пробелы. После ключевых слов типа `if`, `for` и т.п., а также после имен классов у функций пробел перед круглыми скобками не ставится. Примеры:

```
template< typename T >
void Func( T a )
{
   a.DoSomething();
}

template<>
void Func< int >( int a )
{
   a = ( 1 + 2 ) * 3;
}

for( int j = 0; j < n; ++j )
```

Оператор «?:».

При написании оператора «?:» каждая часть оператора должна быть отделена пробелами:

```
5 == a ? 1 + b : *vec.begin();
```

Недопустимо использование оператора «?:» в качестве замены условиям «if».

CV-квалификаторы типов переменных, указатели и ссылки.

Квалификаторы **`const`** и **`volatile`** идут после имени типа при объявлении переменных и обозначении типов возвращаемого значения для функции. Символы **`*`** и **`&`** пробелом слева не отделяется от типа, а справа от имени переменной отделяется одним пробелом. Пример:

```
char const* a; // указатель на char const
char* const a; // константный указатель на char

int F( std::vector< int > const& v )
{
   assert( !v.empty() );
   return v[ 0 ];
}
```

Группировка условий в сравнениях.

Условия проверки на равенство и неравенство(операторы == и !=) выполняются в стиле left-hand comparison, то есть по левую сторону от знака сравнения стоит rvalue, а по правую – lvalue сущность. Пример:

```
int exit_code = Procedure();

if( 0 == exit_code )
   DoSomething();
```

Неправильный вариант:

```
int exit_code = Procedure();

if( exit_code == 0 )
   DoSomething();
```

Несколько условий группируются так, чтобы знаки конкатенации условий `||`, `&&` находились на той же строке, что и первое условие из комбинации двух условий, сами условия, если их несколько, заключаются в скобки и выравниваются, чтобы они начинались строго друг под другом. Пример:

```
if( ( 1 == a ) &&
    ( invalid_size != vec.size() ) &&
    ( std::string( "blah" ) == my_str ) )
{
   DoSomething();
}

if( ( ( 1 == a ) &&
      ( invalid_size != vec.size() ) ) ||
    ( something >= 12 ) )
{
   DoSomething();
}
```

Фигурные скобки.

Открывающая фигурная скобка ставится на новой строке без отступа. Текст, следующий после скобки, начинается со следующей строки с отступом. Закрывающая фигурная скобка ставится ровно под открывающей. Пример:

```
if( 1 == a )
{
   a = 2;
}

do
{
   DoSomething();
}
while( true == condition );
```

Правила наименования переменных.

Недопустимо использование ведущих символов нижнего подчёркивания в названиях переменных. Транслитерация имен переменных(то есть написание русских слов английскими буквами) тоже недопустима. Переменные-члены класса начинаются с префикса «`m`»(например: `mIntValue`), а далее имя переменной продолжается в camel case. Имена аргументов функции также следуют в camel case, только первое слово начинается с маленькой буквы(например: `myArgument`). Имена локальных переменных в функциях именуются по традиционным

правилам именования в C++, когда все слова пишутся с маленькой буквы, а отдельные слова разделяются символами нижнего подчёркивания(пример: `my_var`). Имена классов, функций, шаблонных аргументов также подчиняются соглашению camel case, в данном случае все слова, включая первое, пишутся с большой буквы(пример: `class MyClassFactory : public AbstractFactory`). Функции, возвращающие значения опций, состояний, размеров должны именоваться без префикса «Get». Например: int Size(), int Options(). Функции, устанавливающие значения опций, состояний, размеров должны именоваться с префиксом «Set». Например: SetSize(int), SetOptions(int). Функции, проверяющие какое состояние и возвращающие bool, должны именоваться с префиксом «Is». Например: bool IsVisible().

Глобальные переменные, все функции, методы классов, типы данных, классы – все слова в имени должны начинаться с буквы в верхнем регистре, остальные буквы в нижнем регистре, подчёркивание не используется. Например: MainDB, Buf, DataView, GetPos.

Константы и перечисления (enum) – для каждой группы констант заводится префикс – несколько букв в нижнем регистре, например, для перечисления «ResourceTypes» префикс будет «rt». Имена всех констант данной группы должны начинаться с такого префикса, все остальные буквы имени должны быть в верхнем регистре, слова в имени разделяются подчёркиванием. Например: rtMENU_BAR, ftTEXT, msgSET_TEXT.

Метки оператора goto, а также имена макросов и константы пишутся в верхнем регистре, слова разделяются подчёркиванием. Например: DONT_USE_GOTO, CONST_STRING, MY_MACRO.

Правила описания функций.

При описании функций и методов классов не допускается опускать имена параметров, если не очевидно, что этот параметр означает. Например, так писать нельзя:

```
int SetParams(int,int,int);
```

Нужно писать так:

```
int SetParams( int x, int y, int size );
```

А вот так писать можно, так как из наименования функции и типа параметра ясно, что должна делать данная функция:

```
int Delete( Entry& );
```

Правила описания классов.

В описании класса сначала должен идти раздел public, далее раздел protected, а потом private. Внутри каждого раздела должны быть сначала описаны методы, а потом поля класса. Перед словами «protected» и «private» должна быть оставлена пустая строка. Пустая строка должна быть и между полями и методами класса. Имена полей и методов класса выравниваются по левому краю, типы полей и типы возвращаемых методами значений выравниваются по правому краю. Перечисление «друзей» класса идёт самым последним перед закрывающей скобкой. Если класс переопределяет наследуемый виртуальный метод, то перед этим методом обязательно нужно указать «virtual». Пример описания класса:

```
class MyItem : public AbstractItem
{
   typedef AbstractItem MyParent;

   public:

   enum ItemType
   {
      itSIMPLE,
      itCOMPOSITE
   };

```

```
13
14                          MyItem( ItemType );
15    virtual              ~MyItem();
16
17     std::string const& Name() const;
18                   void SetName( std::string const& name );
19
20               ItemType Type() const;
21                   void SetType( ItemType type );
22
23    protected:
24
25    virtual          void DoSave() const;
26    virtual          void DoLoad();
27
28    virtual std::string HandleEvent( Event evt )
29                         {
30                            // некоторая своя обработка
31                            return MyParent::HandleEvent( evt );
32                         }
33
34    private:
35
36             std::string mName;
37    std::vector< Opts > mOptions;
38               ItemType mType;
39 };
```

Правила написания комментариев.

Комментирование классов, методов и различных флагов крайне желательно. Комментарии к классам и функциям даются в нотации doxygen'а:

```
1 /**
2  * \author Вася Пупкин
3  * \brief Функция, вычисляющая длину окружности радиуса radius
4  * \param radius радиус окружности
5  * \details если radius < 0, будет сгенерировано исключение типа InvalidRadiusException
6  */
7 double CircleLength( double radius );
```

Прочие рекомендации по стилю программирования.

Крайне не рекомендуется использование оператора `goto`! Константы не стоит определять с помощью макросов `define`, и вообще их использование в целом нежелательно и его стоит избегать. Магические числа в коде не допускаются, нужно заводить именованные типизированные константы, помеченные квалификатором `const`.

Если функция принимает указатель или ссылку на нечто, что не должно этой функцией изменяться, то не забывайте указывать «const». Особенно это касается параметров «char*».

Если метод класса не изменяет поля класса, то указывайте «const» после описания такого метода. Например: int Count() const;.

В большинстве случаев экземпляры классов в функцию нужно передавать по ссылке.