C++ Style Guide

Introduction

The following style guide is intended for CO2402: Advanced C++. You are required to use this style guide for code submissions for this module. You may be required to use it for other modules as well.

Do note that this style guide is much smaller than a real guide would be.

How should you name things?

- Absolutely no meaningless names. All names must be meaningful.
- A function must have a strong verb. The form should be verb followed by an object. The object is often a noun, e.g.

```
        DisplayRecord( );
```

- Avoid meaningless or wishy-washy verbs.

- If a function returns a value then use a description of the return value in the name, e.g.

```
        GetName( ); // returns the name
```

- A variable name is likely to be a noun, e.g.

```
        CRecord studentRecords;
```

- A name should be neither too short nor too long.
- Short variable names can only be used if the context is completely unambiguous, e.g. x and y are acceptable in the case of coordinates.
- Single character variable names are OK for loop counters and maths.
- If you do use a single character for a loop counter then it must follow the convention: i, j, k, unless there is an explicit reason otherwise, e.g.

```
        for( int i = 0 ; i < total ; i++ )
```

Names

- Underscores should not be used. The only exception to this is in the naming of constants (see below). If a name is made up of more than one word then each succeeding word should begin with a capital letter, e.g.

```
        largestValue;
```

- Variable names begin with a lower-case character. If a variable is made up of more than one word then each word must begin with an upper-case character (apart from the one beginning the variable name). All other characters must be lower-case. For example:

```
        float wriggle;
        int noOfRecords;
        CEmployee fullTimeEmployees;
```

- Function names begin with an upper-case character. Each word must begin with an upper-case character. All other characters must be lower-case. For example:

```
        DisplayRecord( );
```

- Constants are written all upper-case. Underscores can be used to separate words in the name of a constant. This is the only exception to the rule that underscores should not be used., e.g.

```
        MAX
        ARRAY_SIZE
```

- Member variables begin a lower case *m*. Each word must begin with an upper-case character. All other characters must be lower-case. For example:

```
        mProductCode
```

- Global variables begin a lower case *g*. Each word must begin with an upper-case character. All other characters must be lower-case. For example:

```
        gDatabaseSize
```

- Pointers begin a lower case *p*. Each word must begin with an upper-case character. All other characters must be lower-case. The *m* or *g* prefix is placed before the *p* prefix, e.g.

```
        pDatabaseRecords
        mpDatabaseSize
        gpDataFormat
```

- Class types begin with an upper-case C. Each word must begin with an upper-case character. All other characters must be lower-case. For example:

```
        CTimer
        CEmployee
```

- Enumerated types begin with an upper-case E. Each word must begin with an upper-case character. All other characters must be lower-case. For example:

```
        EKeyCode
```

- Structure types begin with an upper-case S. Each word must begin with an upper-case character. All other characters must be lower-case. For example:

```
        SAddress
```

- Here are the prefixes in table format. The first character of a name indicates its type.

Type	**Prefix**	**Example**
class	C	CTableOfValues
structure	S	SProductSpecification
enumeration	E	ECategoryCode
pointer variable	p	pElement
member variable	m	m
global variable	g	g

Variables

- All variables should be given a default (and sensible) value when they are declared if this is at all possible.
- A pointer should be assigned a value of 0 if is not currently pointing at a known memory address.
- Each variable should be declared on its own line, with its own type preceding it.

```
        int age;
        string name;
```

- Pointers are declared with asterisk next to the type.

```
        CQueue* pProductList = new CQueue;
```

Tabs, white space and layout

- Use a tab character rather than spaces.
- Tab distance should be 4 (i.e. the equivalent of 4 spaces).

- Generally speaking put spaces around all mathematical operators and most other operators, e.g.

```
result = myMoney + yourMoney;
bill && ben
```

- It is difficult to be more precise than “most other operators” without a long list, but unary operators such as increment and decrement don’t get spaces whilst a logical operator would, e.g.

```
daysGoneBy++;
christopher || alice
```

- The curly braces of a control block are aligned with the left hand side of the block. Each brace is placed line of its own.

```
while( currentSpeed < maxTolerance )
{
    IncreaseThrust( );
}
```

Global Variables

- In general you should avoid the use of global variables. It is better to use access routines in the place of global data.
- There are reasons to use global variables. Reasons to use them include:
    - Declaring constants.
    - Eliminating tramp data – a variable that is used in many functions and needs to be passed on and on via a function chain. It will be obvious when you see this happening! The reason for this is that the data is used throughout a program. This genuinely would appear to be global data.

Comments

- Write comments at the level of the code’s intent.
- Focus on why rather than how.
- Place a comment before each block of statements.
- Document the source of algorithms.

General Points

- Don’t abbreviate by just removing one character from a word – what’s the point?
- Abbreviations must be consistent.
- A variable should only be used for one purpose only.