How to optimise code

Code Optimisation

The following optimisations can be easily added to existing code.

1 Initialisation lists:

CFasterClass::CFasterClass(const TDesC& aInfo) : iInfo(aInfo)

```
{
}
```

is better than:

CSlowerClass::CSlowerClass(const TDesC& &aInfo)

```
{
     iInfo = aInfo;
}
```

The reason why the former is quicker is because only the copy constructor will get called. In the latter example the assignment operator is called too along with the variable's default constructor.

2 Optimise For Loops

It is always quicker to count down to zero. Therefore, it is quicker to write; for (i = n-1; i >= 0; --i) as opposed to, for (i = 0; i < n; ++i)

3 Initialise on Declaration

Try to initialise variables straight away,

```
TMyClass x = data;
```

is faster than

```
TMyClass x;
x = data;
```

Declaration then initialization invokes the object's default constructor then its assignment operator. Initializing in the declaration invokes only its copy constructor.

4 Switch Cases

Always put the most commonly used cases first. Obviously less code will be executed this way.

5 Delay Variable Declarations

Only declare variables when they are needed. Not only is spaced saved but its constructor will be called, which can be a waste of CPU if that variable will not be used. If you are unsure if a costly object will be used at all, consider using "lazy initialisation"

6 Pass By Reference

Always try to pass classes by reference rather than by value. For example, use

```
void foo( CFastClass& aObject )
```

is more optimised than

```
void foo( CSlowClass aObject )
```

7 Use 'op=' Operators

It is generally better to use the ‘op=’ instead of ‘op’

```
x += value;
```

is more optimised than,

```
x = x + value;
```

That way no temp object is created. The downside of this is that it can make the code harder to read.