PLAYBOOK NOTES: INLINE FUNCTIONS / MEHTODS

Observe the following code:

```
class dataType
{
public:
    int GetValue(void);

private:
    int value;
};

int dataType::GetValue(void)
{
    // return class member value
    return value;
}
```

So far, nothing new. The class dataType has a private member variable value and a public method that returns that value.

But with every method or function call, there is overhead. The address of the function needs to be stored by the CPU and the parameters copied into memory. Control is then transitioned to the function code. When the code has finished executing the code, the return type is copied into memory, and control is transferred back to where the function is called from.

A more efficient way of writing this code is to use an inline method. The easiest way to create an inline method is to define the method in the class.

```
class dataType
{
public:
    int GetValue(void) { return value; }

private:
    int value;
};
```

Notice that the body of the method is defined to the right of the prototype.

So how does an inline method work? When using an inline method / function, the code is expanded into the call at compile time. This can't be done with preprocessor directive (#) because a preprocessor directive can't perform error checking on the parameters. By using an inline method, you avoid the overhead associated with a non-inline method/function call.

Another way to write an inline method:

```
class dataType
{
public:
    inline int GetValue(void);

private:
    int value;
};

inline int dataType::GetValue(void)
{
    // return class member value
    return value;
}
```

Notice the keyword **inline** is used in both the prototype and the method definition. Functions may also be declared as inline and receive the same benefits as an inline method.

When do you use an inline method / function?

- Use inline methods and functions is short and simple
- Do not use inline methods and functions when the code is long and complex