# C++ Programming Standards

## Contents

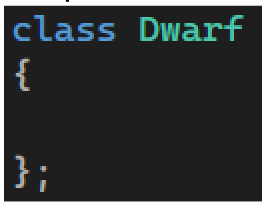
## Programming Standards

When using C++, the following industry standards and naming conventions should always be followed.

### Naming Conventions

#### Classes / Structs

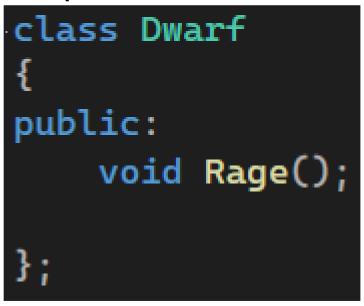
**Description:** Classes should always be PascalCase.
**Example:**



```
class Dwarf
{

};
```

#### Methods

**Description:** All methods are to be in PascalCase.
**Example:**



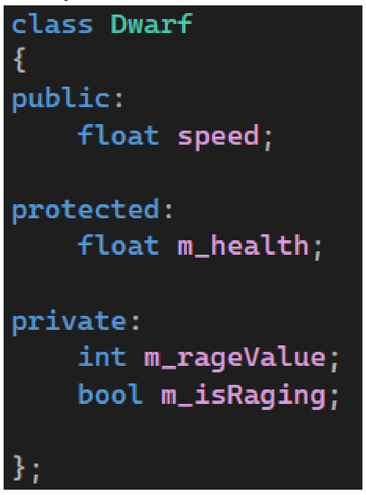
```
class Dwarf
{
public:
    void Rage();

};
```

#### Variables

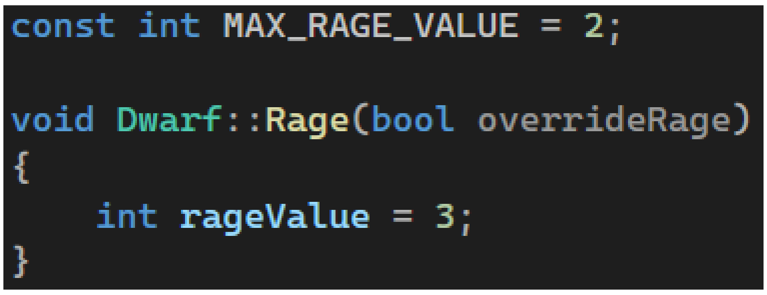
**Description:**

| Type | Convention | Prefix | Suffix |
|---|---|---|---|
| Public Members | camelCase | | |
| Protected Members | camelCase | m_ | |
| Private Members | camelCase | m_ | |
| Local Variables | camelCase | | |
| Global Variables | camelCase | | |
| Parameters | camelCase | | |
| Constants | ALL_UPPER | | |

**Examples:**



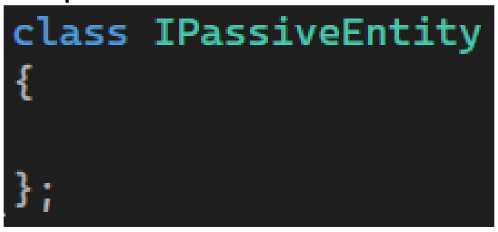
```
class Dwarf
{
public:
    float speed;

protected:
    float m_health;

private:
    int m_rageValue;
    bool m_isRaging;

};
```



```
const int MAX_RAGE_VALUE = 2;

void Dwarf::Rage(bool overrideRage)
{
    int rageValue = 3;
}
```

## Interfaces

**Description**: Must have a prefix of "I" followed by standard PascalCase.
**Example:**



```
class IPassiveEntity
{

};
```

## Styles

### Braces

**Description:**

- **All** braces get their own line.
- In a **switch** statement, all **case** statements should be indented from the **switch** statement.
- **Always** use braces.

**Example:**

```
void Dwarf::Tick(float deltaTime)
{
    switch (m_rageValue)
    {
    case 0:
        {
            std::cout << "Dwarf is calm.\n";
        }
        break;
    case 1:
        {
            std::cout << "Dwarf is annoyed.\n";
        }
        break;
    case 2:
        {
            m_isRaging = true;
            std::cout << "Dwarf is furious.\n";
        }
        break;
    default:
        {
            std::cerr << "DwarvenRage Error: Dwarf is somehow asleep.\n";
        }
        break;
    }
}
```

### Forward Declaration

**Description:** Globally forward declare types.

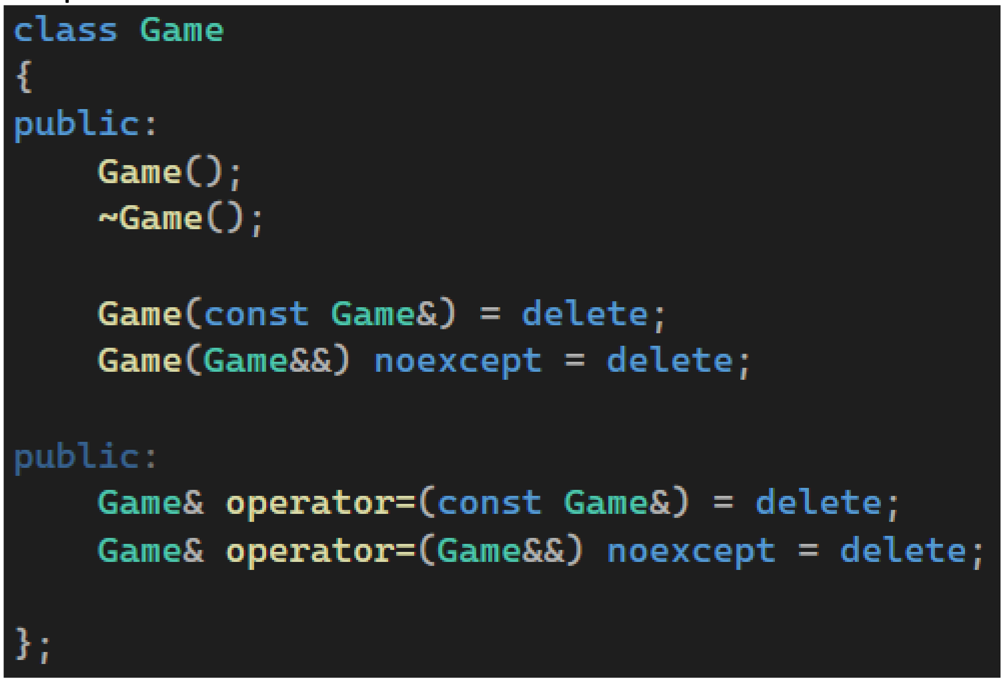
**Example:**

```
class Player;
class Room;

class Game
{

};
```

## Best Practices

### Rule of Five

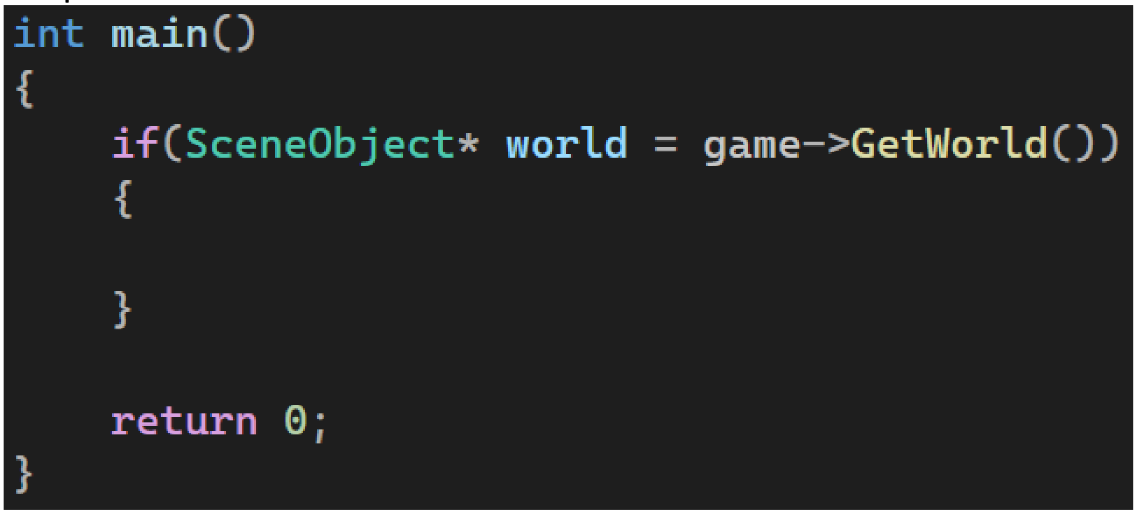
**Description:** When implementing rule of five, mark move constructors as "noexcept" for safety
**Example:**



```
class Game
{
public:
    Game();
    ~Game();

    Game(const Game&) = delete;
    Game(Game&&) noexcept = delete;

public:
    Game& operator=(const Game&) = delete;
    Game& operator=(Game&&) noexcept = delete;

};
```

### Pointer Accessing

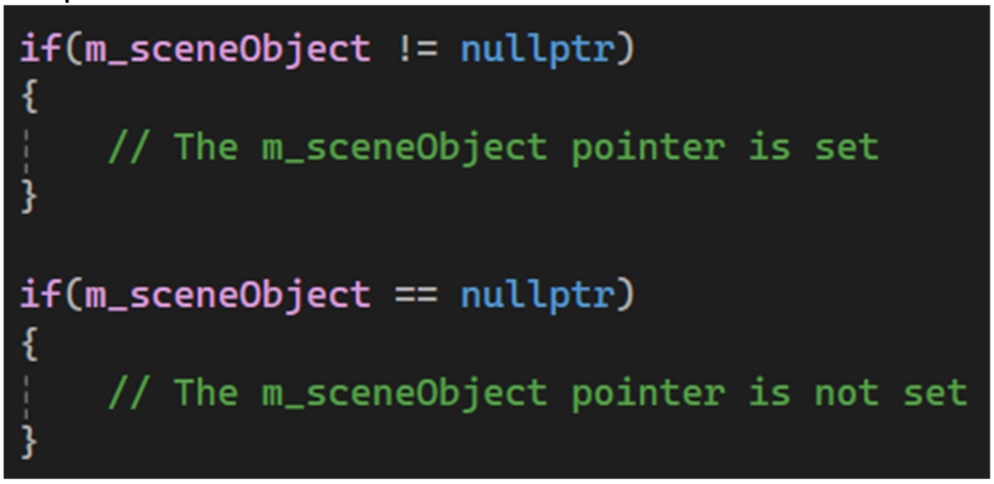
**Description:** When getting a pointer from any type (such as the world from SceneObject class), get and assign it using an in-if statement for safety.
**Example:**



```
int main()
{
    if(SceneObject* world = game->GetWorld())
    {

    }

    return 0;
}
```

## Nullptr Checking

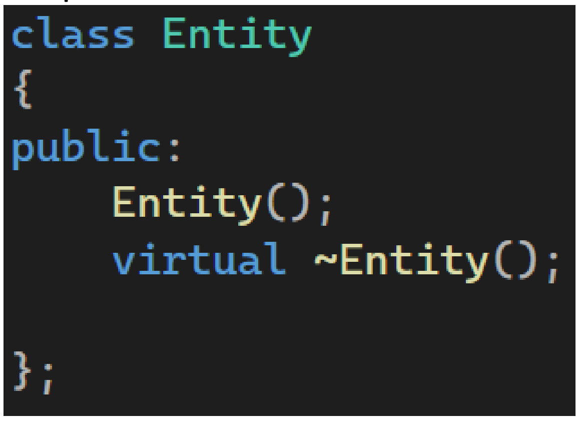
**Description**: Always use != nullptr and == nullptr when validating pointers are set.
**Example:**



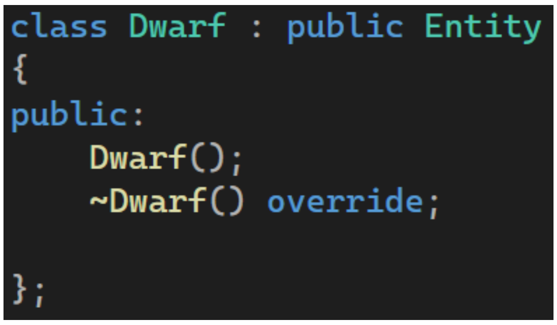
```
if(m_sceneObject != nullptr)
{
    // The m_sceneObject pointer is set
}

if(m_sceneObject == nullptr)
{
    // The m_sceneObject pointer is not set
}
```

## Inheritance and Destructors

**Description:** Always mark parent destructors as “virtual” and child destructors as “override”.
**Example:**



```
class Entity
{
public:
    Entity();
    virtual ~Entity();

};
```



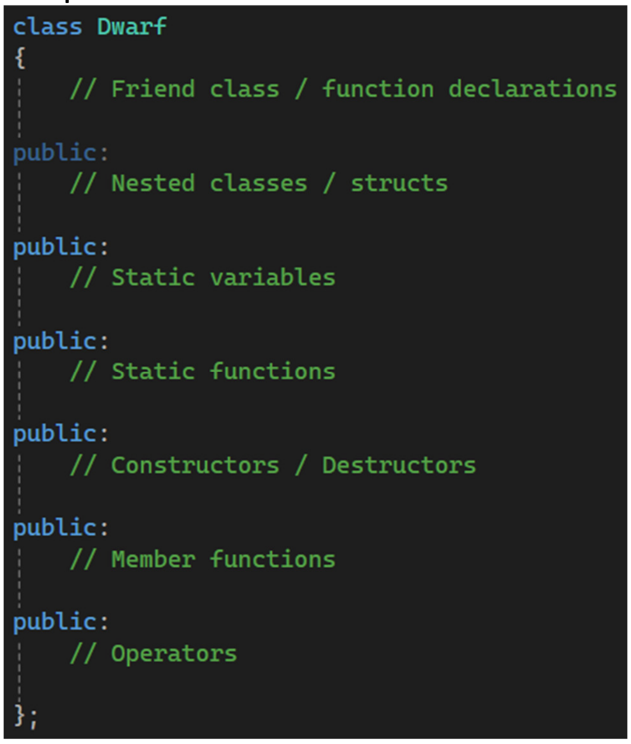
```
class Dwarf : public Entity
{
public:
    Dwarf();
    ~Dwarf() override;

};
```

## Class Structure

**Description:** must be laid out in the following structure, separating each type by a protection keyword:

- Friend class / function declarations.
- Public
  - Nested classes / structs.
  - Static variables.
  - Static functions.
  - Member variables
  - Constructors / destructors.
  - Functions.
  - Operators.
- Protected
  - Nested classes / structs.
  - Static variables.
  - Static functions.
  - Member variables
  - Constructors / destructors.
  - Functions.
  - Operators.
- Private
  - Nested classes / structs.
  - Static variables.
  - Static functions.
  - Member variables
  - Constructors / destructors.
  - Functions.
  - Operators.

**Example:**



```
class Dwarf
{
    // Friend class / function declarations

public:
    // Nested classes / structs

public:
    // Static variables

public:
    // Static functions

public:
    // Constructors / Destructors

public:
    // Member functions

public:
    // Operators

};
```

## Further Notes

- Precompute whenever possible.

- Sloppy code must be marked for re-coding later: **// RECODE**. And must be fixed before submission.
- Single line comments should be used as often as humanly possible to elaborate code flow.

- Code must be legible to be read by other developers by using logical naming and commenting.

- Unless something is accessed outside the class, all variables should be private or protected.

- Avoid magic numbers at all costs.