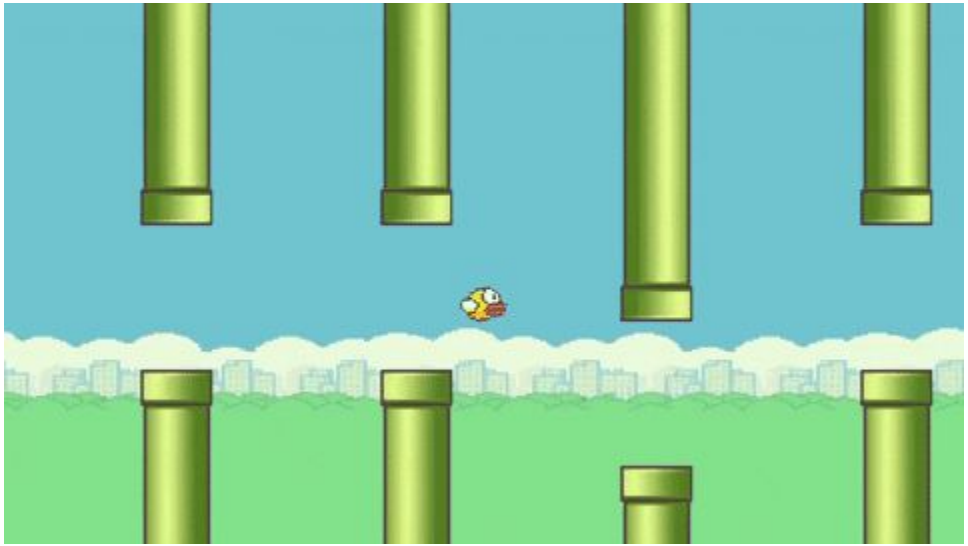
Functional programming techniques in OOP languages (C++)

Structure

- What is functional programming?
- Functional Programming Concepts
  - Referential transparency
  - First Class & Higher Order Functions
  - Currying
- OOP Patterns Revisited
- Conclusion

Sample application



What is functional programming?

Definition

“**Functional programming** is a style of programming which models computations as the evaluation of expressions.”

https://wiki.haskell.org/Functional_programming

What is a function?

A function maps inputs to outputs.

f(x)=x*x	
Input	**Output**
1	1
2	4
3	9

Functional programming concepts

Referential transparency

Calling a function with a certain set of arguments results in the same result every time the function is called with those arguments.

Referential transparency = Immutable data + Purity

Referential transparency - Code example

```
class Vec2 {
public:
    Vec2(float x, float y) : _x{ x }, _y{ y } { }

    float get_x(void) { return _x; }
    float get_y(void) { return _y; }

    void set_x(float new_x) { _x = new_x; }
    void set_y(float new_y) { _y = new_y; }

private:
    float _x;
    float _y;
};
```

Referential transparency - Code example

```
class Vec2 {
public:
    Vec2(float x, float y) : _x{ x }, _y{ y } { }

    float get_x(void) const { return _x; }
    float get_y(void) const { return _y; }

    void set_x(float new_x) { _x = new_x; }
    void set_y(float new_y) { _y = new_y; }

private:
    float _x;
    float _y;
};
```

Referential transparency - Code example

```
class Vec2 {
public:
    Vec2(float x, float y) : _x{ x }, _y{ y } { }

    float x(void) const { return _x; }
    float y(void) const { return _y; }

    void set_x(float new_x) { _x = new_x; }
    void set_y(float new_y) { _y = new_y; }

private:
    float _x;
    float _y;
};
```

Referential transparency - Code example

```
class Vec2 {
public:
    Vec2(float x, float y) : _x{ x }, _y{ y } { }

    float x(void) const { return _x; }
    float y(void) const { return _y; }

    void set_x(float new_x) const { _x = new_x; }
    void set_y(float new_y) const { _y = new_y; }

private:
    float _x;
    float _y;
};
```

Referential transparency - Code example

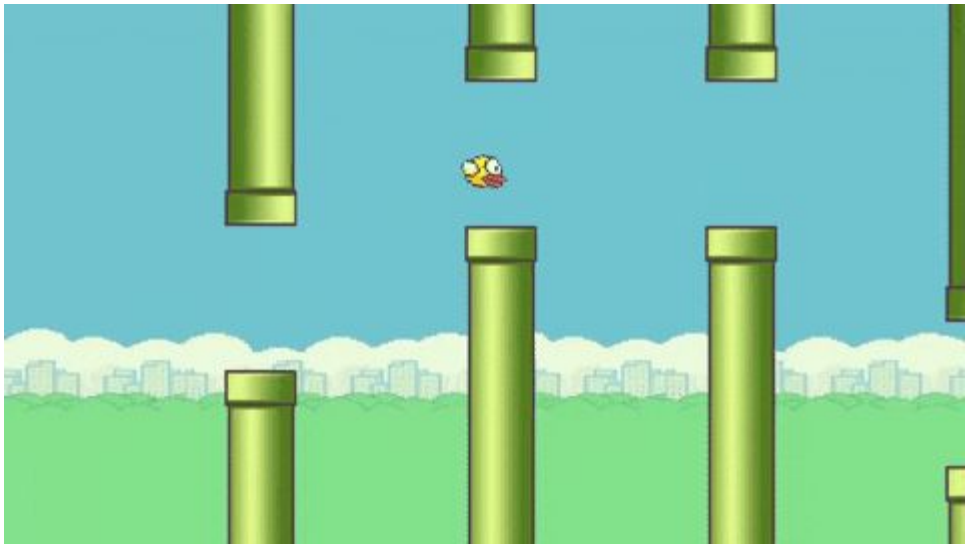
```
class Vec2 {
public:
    Vec2(float x, float y) : _x{ x }, _y{ y } { }

    float x(void) const { return _x; }
    float y(void) const { return _y; }

    Vec2 set_x(float new_x) const { return Vec2(new_x, _y); }
    Vec2 set_y(float new_y) const { return Vec2(_x, new_y); }

private:
    float _x;
    float _y;
};
```

Referential transparency - Code example

```
class Vec2 {
public:
    Vec2(float x, float y) : _x{ x }, _y{ y } { }

    float x(void) const { return _x; }
    float y(void) const { return _y; }

    Vec2 with_x(float new_x) const { return Vec2{ new_x, _y }; }
    Vec2 with_y(float new_y) const { return Vec2{ _x, new_y }; }

private:
    float _x;
    float _y;
};
```

Can this be applied everywhere?

```
class Player {
public:
    void update(void) {
        _fallrate += _FALLSPEED;
        _position.y += _fallrate;
        _rotation = 90.0f * _fallrate;
    }

private:
    Vec2 _position;
    float _fallrate;
    float _rotation;
};
```



Can this be applied everywhere?

```
class Player {
public:
    Player updated(void) const {
        float new_fallrate = _fallrate + _FALLSPEED;

        return Player(_position.with_y(_position.y() + new_fallrate),
                      90.0f * new_fallrate,
                      new_fallrate);
    }

private:
    Vec2 _position;
    float _fallrate;
    float _rotation;
};
```

First Class & Higher Order Functions

First Class Functions

- Functions are like any other value
- Creating, passing, returning and storing them is possible

Higher Order Functions

- Function takes or returns a function

std::function & std::bind

```
class AClass {
public:
    void some_method(const char* param) const {
        std::cout << “Hello “ << param << std::endl;
    }
};

int main(void) {
    AClass instance;

    std::function<void(const char*)> fun = std::bind(&AClass::some_method,
                                                     &instance,
                                                     std::placeholders::_1);

    fun(“world”);
}
```

std::function & std::bind

```
class AClass {
public:
    void some_method(const char* param) const {
        std::cout << “Hello “ << param << std::endl;
    }
};

int main(void) {
    AClass instance;

    auto fun = std::bind(&AClass::some_method,
                         &instance,
                         std::placeholders::_1);
    fun(“world”);
}
```

From std::function to lambdas

```
int main(void) {
    auto fun = [](const char* msg)->void {
        std::cout << “Hello “ << msg << std::endl;
    };

    fun(“world”);
}
```

From std::function to lambdas

```
int main(void) {
    auto fun = [](const char* msg)->void {
        std::cout << “Hello “ << msg << std::endl;
    };

    fun(“world”);
}
```

From std::function to lambdas

```
int main(void) {
    auto fun = [](const char* msg)->void {
        std::cout << “Hello “ << msg << std::endl;
    };

    fun(“world”);
}
```

From std::function to lambdas

```
int main(void) {
    auto fun = [](const char* msg)->void {
        std::cout << “Hello “ << msg << std::endl;
    };

    fun(“world”);
}
```

From std::function to lambdas

```
int main(void) {
    auto fun = [](const char* msg)->void {
        std::cout << “Hello “ << msg << std::endl;
    };

    fun(“world”);
}
```

Old-school callbacks...

```
// glfw3.h: typedef void (* GLFWwindowsizefun)(GLFWwindow*,int,int);
void cursor_callback(GLFWwindow* window, double xpos, double ypos) {
    /* … */
}

class Window {
public:
    Window(void) {
        GLFWwindow* handle = glfwCreateWindow(1280, 720, ”App”, nullptr, nullptr);

        glfwSetCursorPosCallback(handle, cursor_callback);
    };
};
```

...combined with modern features

```
class Window {
public:
    Window(void) {
        GLFWwindow* handle = glfwCreateWindow(width, height, title, nullptr, nullptr);

        glfwSetCursorPosCallback(handle, [](GLFWwindow* window, double xpos, double ypos)->void {
            /* … */
        });
    };
};
```

Higher Order Functions in STL

```
std::transform(_background_tiles.begin(), _background_tiles.end(),
               std::back_inserter(updated_tiles),
               [background_scroll](const auto& bg)->std::unique_ptr<Background> {
                   return bg->with_render_offset(Vec2(0.03003f * background_scroll, 0.0f));
               }
);
```

Higher Order Functions in STL

```
std::transform(_background_tiles.begin(), _background_tiles.end(),
               std::back_inserter(updated_tiles),
               [background_scroll](const auto& bg)->std::unique_ptr<Background> {
                   return bg->with_render_offset(Vec2(0.03003f * background_scroll, 0.0f));
               }
);
```

Higher Order Functions in STL

```
std::transform(_background_tiles.begin(), _background_tiles.end(),
               std::back_inserter(updated_tiles),
               [background_scroll](const auto& bg)->std::unique_ptr<Background> {
                   return bg->with_render_offset(Vec2(0.03003f * background_scroll, 0.0f));
               }
);
```

Higher Order Functions in STL

```
std::transform(_background_tiles.begin(), _background_tiles.end(),
               std::back_inserter(updated_tiles),
               [background_scroll](const auto& bg)->std::unique_ptr<Background> {
                   return bg->with_render_offset(Vec2(0.03003f * background_scroll, 0.0f));
               }
);
```

Currying

- Partial application of functions
- Apply one argument and get back a function for the next one
- Semi-state for a function

Currying - Code example

```
std::transform(_background_tiles.begin(), _background_tiles.end(),
               _background_tiles.begin(),
               [background_scroll](const auto& bg)->std::unique_ptr<Background> {
                   return bg->with_render_offset(Vec2(0.03003f * background_scroll, 0.0f));
               }
);
```

Currying - Code example

```
typedef std::function<std::unique_ptr<Background>(const std::unique_ptr<Background>&)> UpdBGFun;
UpdBGFun upd_bg_fun = updated_background(background_scroll);

std::transform(_background_tiles.begin(), _background_tiles.end(),
               _background_tiles.begin(),
               upd_bg_fun);

/* … */

UpdBGFun updated_background(int32_t background_scroll) {
    return [=](const auto& bg)->std::unique_ptr<Background> {
        return bg->with_render_offset(Vec2(0.03003f * background_scroll, 0.0f));
    };
}
```

OOP Patterns revisited

Command Pattern - OOP example

```
class ICommand {
public:
    virtual void execute(void) = 0;
};

class RiseCommand : public ICommand {
public:
    virtual void execute(void) override {
        _player->rise();
    }
private:
    Player _player;
};

class Player {
public:
   void rise(void);
};
```

Command Pattern - OOP example

```
class ICommand {
public:
    virtual void execute(void) = 0;
};

class RiseCommand : public ICommand {
public:
    virtual void execute(void) override {
        _player->rise();
    }
private:
    Player _player;
};

class Player {
public:
   void rise(void);
};
```

```
class InputManager {
public:
    void handle_events(void) {
        if (space_key)
            _rise_com.execute();
    }

private:
    RiseCommand _rise_com;
};
```

Command Pattern - FP example

```
class Player {
public:
   void rise(void);
};
```

Command Pattern - FP example

```
class Player {
public:
   void rise(void);
};
```

```
class InputManager {
public:
    using RiseFn = std::function<void(void)>;

    void set_rise_com(RiseFn fun) {
        _rise_com = fun;
    }

    void handle_events(void) const {
        if (space_key)
            _rise_com();
    }

private:
    RiseFn _rise_com;
};
```

Conclusion

Advantageous properties of functional programs

- Determinism
- Easy testability
- Descriptive code

Drawbacks

- The real world is not functional.
- Copying, copying, copying.

Key Takeaway

- `const` is your best friend.
- Use OOP for large scale structure and FP for small scale structure.
- Don’t go full Haskell.

Sources

- https://www.keycdn.com/blog/functional-programming
- https://www.guru99.com/functional-programming-tutorial.html
- https://www.gamasutra.com/view/news/169296/Indepth_Functional_programming_in_C.php
- http://www.jot.fm/issues/issue_2009_09/article5.pdf
- https://stackoverflow.com/questions/21471836/what-can-c-offer-as-far-as-functional-programming/21472274#21472274
- https://youtu.be/0if71HOyVjY
- https://youtu.be/Clg6eyJv4dk