C++ Function Parameter Choices

Sane Defaults

– evade needless (deep) copy, prefer move (pointer swap) –

In

```
void f(const T&);
```

Objects for *reading*: `const` **reference**

```
// example
void Align(const std::vector<int>& page_ids) {
}
Align(page_ids);
```

In for Cheap or Non-copyable Objects

```
void f(T);
```

Primitives and impossible to copy types: **value**

```
// example
void SetOrigin(int count, Point origin,
               std::unique_ptr<Shape> box) {
}
SetOrigin(n, org, std::move(box));
```

In take Ownership

```
void f(T&&);
```

Objects for *ownership*: **rvalue reference**

```
// example
void SetTitle(std::string&& title) {
  m_title = std::move(title);   // steal title's data
}
SetTitle(std::move(app_title)); // app_title empty here
```

In take Copy

```
void f(T&&);
void f(const T&);
```

Provide two overloads; for callers wanting to

- *relinquish* ownership: **rvalue reference**
- *retain* ownership: `const` **reference**

```
// example
void SetTitle(std::string&& title) {       // for relinquishers
  m_title = std::move(title);              // cheap move
}

void SetTitle(const std::string& title) {  // for retainers
  m_title = title;                         // expensive deep copy
}
SetTitle(std::move(page_title));    // page_title empty
SetTitle(doc_title);                // doc_title valid
```

In take Ownership in Constructor

```
void f(T);
```

Objects for *ownership* during construction: **value**

```
// example
Image::Image(Pixels p) : m_pixels(std::move(p)) {
}
auto i1 = Image(std::move(p1));  // relinquish; cheap move
auto i2 = Image(p2);             // retain; expensive deep copy
```

Out

```
T f();
```

Return by **value**; prvalues get moved, copy evaded for move-unfriendly types by RVO

```
// example
Image Render() {
  return Image(/*some params*/);
}
Image i = Render();    // using std::move(Render()) is pessimization
```

Out for Move-Unfriendly types

```
void f(T&);
```

Take **lvalue reference** and fill data
Returning value (both move and copy) will lead to deep copy for PODs

```
// example
// a move-unfriendly type (has no freestore data members)
struct Properties {
  Point                origin;
  float                size_x, size_y;
  std::array<Margin, 4>  margin_sizes;
};
void Deduce(Properties& p) {
}
```

Out Object Reference

```
T& f();
const T& f();
```

Return **lvalue reference** to *object outliving function and caller*

```
// example
struct Application {
  Document& GetDocument { return m_pdf; }  // covariant return type
  PDFDocument m_pdf;
};  // app outlives GetDocument() and temp() calls
void PrintDoc(Application& app) { app.GetDocument().Print(); }
int main() { Application app{str_pdf_path}; PrintDoc(app); }
```

In Out

```
void f(T&);
```

Objects for *reading* and *writing*: **lvalue reference**

```
// example
void AdjustMargins(std::vector<Margin>& margins) {
}
```

Reference: Essentials of Modern C++ Style, *Herb Sutter*