Distinguish Between Pointers and Lvalue References

The material in this handout is collected from the following references:

- Sections $2.3.1$ and $6.2.2$ of the text book C++ Primer.
- Section 7.7 of C++ Programming Language.
- Item 1 of Effective C++.

Pointers and lvalue references can have similar use cases in C++. Both pointers and lvalue references let you refer to other objects indirectly. Both lvalue references and pointers can be used to provide *pass-by-reference* semantics to a function. Additionally, they both provide an alternative way to access an existing variable: pointers through the variable's address, and lvalue references through another name for that variable. But what are the differences between the two, and how, then, do you decide when to use one and not the other? There are four scenarios to consider.

1. **Initialization**: There is no such thing as a *null reference* but there is such a thing as a *null pointer*. Because there is no such as a null reference, C++ requires that lvalue references be initialized:

   ```
   // a reference must always refer to some object
   std::string &rs; // error!!!
   // references must always be initialized
   std::string s{"hello"};
   std::string &rs{s}; // okay - rs is an alias for s
   // you can use rs anywhere you can use s
   ```

   Pointers are subject to no such restriction:

   ```
   std::string *ps_bad; // uninitialized pointer: valid but risky
   str::string *ps_good {nullptr};
   ```

   Thus, if you have a variable whose purpose is to refer to another object, but it is possible that there might not be an object to refer to, then that variable must be declared as a pointer. On the other hand, if the variable must always refer to an object (there is no possibility that the variable is null), you should make the variable a lvalue reference.

2. **Efficiency**: The fact that there is no such thing as a null reference implies that it can be more efficient to use lvalue references rather than to use pointers. That's because there's no need to test the validity of a lvalue reference before using it:

   ```
   void print_int(int const &ri) {
     // ri will always be a valid alias to an int object that exists
     // in some other scope
     std::cout << ri << "\n";
   }
   ```

   Pointers, on the other hand, should generally be tested against null:

```cpp
void print_int(int const *pi) {
  // there is no guarantee that pi will always point to a valid object
  if (pi != nullptr) {
    std::cout << *pi << "\n";
  }
}
```

3. **Reassignment**: Another important difference between pointers and lvalue references is that pointers may be reassigned to refer to different objects. A lvalue reference, however, will always be an alias to the object with which it is initialized:

```cpp
std::string s1{"Tom"};
std::string s2{"Jane"};
std::string &rs{s1};   // rs refers to s1
std::string *ps{&s1}; // ps refers to s1
rs = s2;   // rs still refers to s1, but s1's new value is "Jane"
ps = &s2;  // ps now points to s2; s1 is unchanged
```

4. **Overloaded** `operator[]`: There's one situation in which you should always use a lvalue reference - that’s when implementing certain operators. The most common example is `operator[]`. This operator needs to return something that can be used as the target of an assignment:

```cpp
std::vector<int> v(10); // create an int vector of size 10

/* some other code here */

// vector::operator[] returns a reference or const reference
v[5] = 10; // the target of this assignment is the return value of op[]
```

If `operator[]` returned a pointer, then we would need to write:

```cpp
std::vector<int> v(10); // create an int vector of size 10

/* some other code here */

// if vector::operator[] returns a pointer
*v[5] = 10; // correct - but it make v look like a vector of pointers
```

Final Thoughts

Lvalue references are generally easier and safer than pointers. As a decent rule of thumb, lvalue references should be used in place of pointers when possible.

In general, you should use a pointer whenever you need to take into account the possibility that there's nothing to refer to (in which case you set the pointer to null) or whenever you need to be able to refer to different things at different times (in which case you change where the pointer points). Lvalue references are the feature of choice when you know you have something to refer to, when you'll never want to refer to anything else, and when implementing operators whose syntactic requirements make the use of pointers undesirable. In all other cases, use pointers.

The following list summarizes some of the differences between pointers and lvalue references, as well as when each should be used:

Lvalue references	**Pointers**
References must be initialized when they are declared. This means that a reference will always point to data that was intentionally assigned to it.	Pointers can be declared without being initialized, which is dangerous. If this happens mistakenly, the pointer could be pointing to an arbitrary address in memory, and the data associated with that address could be meaningless, leading to undefined behavior and difficult-to-resolve bugs.
References cannot be null. This means that a reference should point to meaningful data in the program.	Pointers can be null. In fact, if a pointer is not initialized immediately, it is often best practice to initialize to `nullptr`, a special type which indicates that the pointer is null.
When used in a function for pass-by-reference, the reference can be used just as a variable of the same type would be.	When used in a function for pass-by-reference, a pointer must be dereferenced in order to access the underlying object.