

# References & Pointers

## const Reference

In C++, pass-by-reference with `const` can be used for a function where the parameter(s) won't change inside the function.

This saves the computational cost of making a copy of the argument.

```
int triple(int const &i) {  
  
    return i * 3;  
}
```

The last sentence could be confusing. You can use `const` for declaration, or for args. in functions. When your arg

## Pointers

In C++, a *pointer* variable stores the memory address of something else. It is created using the `*` sign.

```
int* pointer = &gum;
```

## References

In C++, a *reference* variable is an alias for another object. It is created using the `&` sign. Two things to note:

1. Anything done to the reference also happens to the original. **also true other way around**
2. Aliases cannot be changed to alias something else.

```
int &sonny = songqiao;
```

compare with: `int sonny = songqiao;`

## Memory Address

In C++, the *memory address* is the location in the memory of an object. It can be accessed with the "address of" operator, `&`.

Given a variable `porcupine_count`, the memory address can be retrieved by printing out `&porcupine_count`. It will return something like: `0x7ffd7caa5b54`.

```
std::cout << &porcupine_count << "\n";
```

## Dereference

In C++, a *dereference reference operator*, `*`, can be used to obtain the value pointed to by a pointer variable.

```
int gum = 3;
```

```
// * on left side is a pointer  
int* pointer = &gum;
```

```
// * on right side is a dereference of that  
pointer  
int dereference = *pointer;
```

## Pass-By-Reference

In C++, *pass-by-reference* refers to passing parameters to a function by using references.

It allows the ability to:

- Modify the value of the function arguments.
- Avoid making copies of a variable/object for performance reasons.

```
void swap_num(int &i, int &j) {  
  
    int temp = i;  
    i = j;  
    j = temp;  
  
}  
  
int main() {  
  
    int a = 100;  
    int b = 200;  
  
    swap_num(a, b);  
  
    std::cout << "A is " << a << "\n";  
    std::cout << "B is " << b << "\n";  
  
}
```

Pointer declaration is dangerous as it's left uninitialized. You can do this: `int* ptr = nullptr`; `nullptr` is like the n