1 The Many Uses of const

1.1 const Constants

If you assumed the keyword `const` has something to do with constants, you have correctly uncovered one of its uses. Defining a constant with `const` is just like defining a variable, except that the compiler guarantees that code cannot change th evalue.

```
const float versionNumber = 2.0f;
const std::string productName = "Super Hyper Net Modulator";
```

1.2 const to Protect Parameters

You can cast a non-const variable to a const variable. It offers some degree of protection from other code changing the variable. If the function attempts to change the value of the parameter, it will not compile.
In the following code, a `string*` is automatically cast to a `const string*` in the call to `mysteryFunction()`. If you attempt to change the value of the passed string, the code will not compile.

```
void mysteryFunction(const std::string* someString)
{
  *someString = "Test"; // Will not compile
}

int main()
{
  std::string myString = "The string";
  mysteryFunction(&myString);
  return 0;
}
```

1.3 const References

You will often find code that uses `const` reference parameters. At first, that seems like a contradiction. The main value, however, is that the `const` reference parameters provides efficiency. When you pass a value to a function, an entire copy is made. When you pass a reference, you are really just passing a pointer to the original so the computer doesn't need to make the copy. BY passing a const reference, you get the best of both worlds — no copy is made but the original variable cannot be changed.

```
void printString(const std::string& myString)
{
     std::cout << myString << std::endl;
}
int main()
{
     std::string someString = "Hello World";
     printString(someString);
     return 0;
}
```