Coding Guidelines

Variable Instantiation

- using namespace std should only be used in a local scope or not at all

```
void function(){
  std::cout << "data" << std::endl;
  std::string var = "data";
  std::shared_ptr<int> int_var = std::make_shared<int>(5);
}
```

```
void function(){
  using namespace std;

  cout << "data" << endl;
  string var = "data";
  shared_ptr<int> int_var = make_shared<int>(5);
}
```

- global variables should not be mutable, rather they should be immutable CONST variables typed in all capitals
- variables should be passed down to functions such that a variable's lifetime that is created by scope is controlled by that scope unless that scope is returning a smart pointer

```
void doOtherThing(int* a)
{
    //do things with 'a'
    //complete things
    //but "doOtherThing" cannot deallocate 'a' because it does not own it
    //so just return
    return;
}

void doSomething()
{
    int* a = new int(5);
    doOtherThing(a); //borrow 'a' to doOtherThing
    delete a; // I'm done with 'a' so since I created it in my scope, I will delete it
}
```

The only exception to this rule would be if doOtherThing was returning a pointer, in that case it should be wrapped in a smart pointer so that we don't have to guess who will delete the pointer

```
std::shared_ptr<int> doOtherThing()
{
    //create and return smart shared pointer
    return std::make_shared<int>(5);
}

void doSomething()
{
    std::shared_ptr<int> user = doOtherThing(); //create
    return;//user will automatically delete itself
}
```

Comment Style

H/HPP Files

- Header

```
/**
 * @file fileName.h fileName.cpp
 * @class className
 * @author Author Name
 * @brief Description of Class
 */
```

- Functions follow a camelCase convention

```
/**
* @brief Function Description
* @param paramName Description of paramater
* @return returnType
*/
```

- Variables follow a snake_case convention

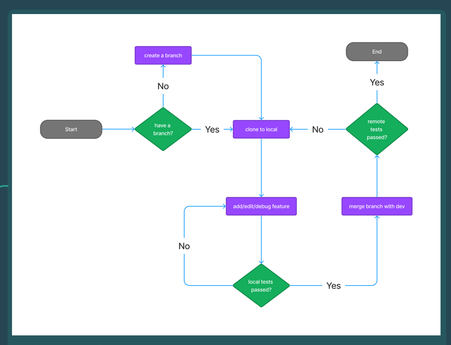
```
/**
 * @brief variable description
 */
```

Function Implementation

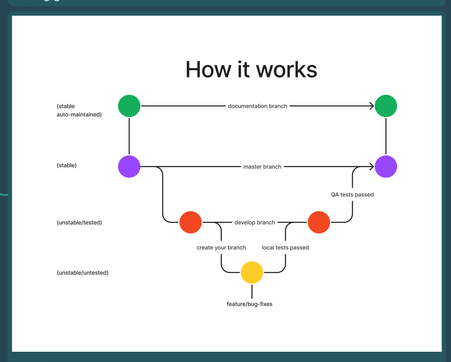
CPP Files

- Avoid Long if, else if, else branching
- Avoid deeply nested if's
- Avoid spagetti recursion
- Avoid making a single function longer than 200-300 lines of code; consider making portions into their own sub functions
- Test a function to make sure it works by using your own tests and since we don't need full unit test coverage, test at least one function in your class with the provided google test library. An example template for tests has already been set up in the tests.cpp files
- Indentation should be standard for readibility, so avoid one lining code, wrap everything in curly braces even if it is a simple if statement

Git Contributions



making git contributions



repo lifetime

CONTRIBUTING.md

Documentation

documentation is automatically generated by a github action whenever changes are made to the src directory on the main branch. The docs are found in the documentation branch

Errors

if you decide to use exceptions, contain them to your package or class only

Github issues

please make use of github issues when you encounter certain code that you are unable to fix. Labels are available when creating your issue and please also assign issues to members who are supposed to work on that portion of code

Github pull requests

please assign labels and assignees who worked on the code in the pr when creating and merging pr's with a branch