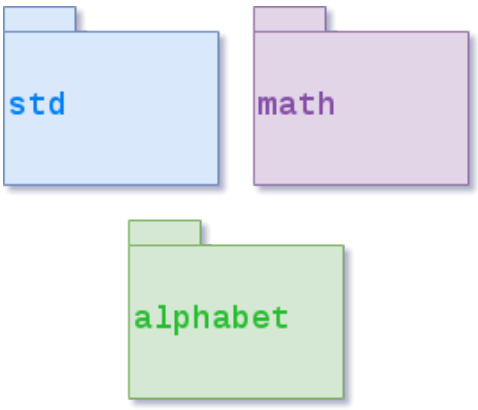
Namespaces



Namespaces allow you to use the same variable identifiers for different entities. In a particular namespace, there can only be one identifier per entity. For example, you cannot make a variable of type **int** and a variable of type **double** and use the same variable identifier (name) for both.

```
int x;
double x; //Can't do in same namespace
```

The reason for this is the compiler will not know which x to use. For example, displaying x in a cout statement. If you would like to use the same variable to mean something different, you are able to use specific namespaces. For this example, take two namespaces, a math namespace and an alphabet namespace. Let's say the alphabet namespace represents the alphabet, and the math namespace represents algebraic variables.

```
namespace math {
    int x = 500;
    int y = 600; }
namespace alphabet {
    char x = 'x';
    char y = 'y'; }
int main() {
    std::cout << math::x << "is in variable " << alphabet::x; }
```

Now the compiler will know which **x** to use at the appropriate time. More on the standard namespace on the next page (**std::**). For main, you could also do a longer sequence via scope. Here is an alternate main function:

```
int main() {
    { using namespace math
      std::cout << x << ' ' << y << std::endl; }
    { using namespace alphabet
      std::cout << "500 is in variable " << x
                << " and 600 is in variable " << y << std::endl; }
}
```

Notice how each namespace is encapsulated in between curly brackets. We need to do this for the scope (see the **Scope** page for more information/variable overrides). Inside the scope we use the **using namespace** you are used to seeing on top of a program. This allows the variables to be used without prepending **math::** or **alphabet::** The **::** is called the **scope resolution operator** and is used to denote which **namespace** or **class** you are using at that particular time.