Custom Datatypes and Objects

Chantilly Robotics - 612

Let’s review

A variable is like a box that stores a value of a specific type. We’ve seen a couple of examples already:

- `int i = 1;`
- `std::string chant = “Six what???”;`
- `float ratio = 0.7f;`
- `double e = 2.71828;`
- `char a = ‘a’;`
- `bool is_robotics_cool = true;`

But what if I told you we could define our own types??

What like some sort of typedef?

- Not yet ;)

C++ gives us several ways of organizing information. Let’s look at a few

Enumerated lists

- Enumerated lists or enums are a list of possible values that a variable can equal
- Imagine if you had a box of chocolates and the only types of chocolate that existed in the box were dark, milk, and white. You could declare an enum like this:

```
enum chocolate { DARK, MILK, WHITE };
```

- You can then declare a variable of type `chocolate` as you would anything else!

```
chocolate candy1 = chocolate::DARK;
```

Example 1

```cpp
#include <iostream>

enum robot_state { ON, OFF, ERROR };

int main() {
    robot_state state = robot_state::ON; //not necessary, but a good habit

    if(state == robot_state::ON) {
        std::cout << "Robot active!\n";
    }

    return 0;
}
```

Typedefs

You can use `typedef [original] [new]` to rename data types.

This is useful when we start getting into data types that are long to type, or use a lot of ::s

Example 2

```
#include <iostream>

typedef std::string str;

int main() {
     str name = "Sammy Sheirich"; // the typedef orders the computer to replace 'str'
with std::string behind the scenes
     std::cout << name << std::endl; // still prints the name of our wonderful CEO

     return 0;
}
```