Object-Oriented Programming

Convert

Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

Concerns

- Our programs are full of concerns
- It is easy to forget some of them, as we are so familiar with them
- With small programs, we can often compensate for the lack of separation of concerns
- But as programs increase in size, number of parts, and the complexity of each part, separation of concerns becomes even more important
- Abstraction mechanisms allow us to separate these concerns and isolate some of them into separate parts of the program

Concerns

- language features
- types
- formats
- concepts
- variables

Convert

```cpp
/*
    convert.cpp

    Convert text with various conversions. E.g.,
        convert --upper "abcdefg"
*/

#include <iostream>
#include <string>
#include <string_view>
#include <cctype>

int main(int argc, char* argv[]) {

    // requires conversion option and string
    if (argc != 3) {
        std::cerr << "usage: " << argv[0] << " <option> <string>\n";
        return 1;
    }

    // convert the string according to the option
    std::string text(argv[2]);
    if (std::string_view(argv[1]) == "--upper") {

        for (auto pc = text.begin(); pc != text.end(); ++pc)
            *pc = std::toupper(*pc);

    } else if (std::string_view(argv[1]) == "--lower") {

        for (auto pc = text.begin(); pc != text.end(); ++pc)
            *pc = std::tolower(*pc);

    } else {

        std::cerr << "Invalid conversion option: " << argv[1] << '\n';
        return 1;
    }

    // output converted text
    for (auto pc = text.cbegin(); pc != text.cend(); ++pc)
        std::cout << *pc;
    std::cout << '\n';

    return 0;
}
```

- An example to show the separation of concerns and techniques to separate them

- At this point, the program only converts to uppercase and lowercase, but this could easily be expanded

- We will try to separate concerns as much as possible while staying with a one-file program

Goal

```
/*
    convert.cpp

    Convert text with various conversions. E.g.,
        convert --upper "abcdefg"
*/

#include <iostream>
#include <string>
#include <string_view>
#include <cctype>

int main(int argc, char* argv[]) {

    // requires conversion option and string
    if (argc != 3) {
        std::cerr << "usage: " << argv[0] << " <option> <string>\n";
        return 1;
    }

    // convert the string according to the option
    std::string text(argv[2]);
    if (std::string_view(argv[1]) == "--upper") {

        for (auto pc = text.begin(); pc != text.end(); ++pc)
            *pc = std::toupper(*pc);

    } else if (std::string_view(argv[1]) == "--lower") {

        for (auto pc = text.begin(); pc != text.end(); ++pc)
            *pc = std::tolower(*pc);

    } else {

        std::cerr << "Invalid conversion option: " << argv[1] << '\n';
        return 1;
    }

    // output converted text
    for (auto pc = text.cbegin(); pc != text.cend(); ++pc)
        std::cout << *pc;
    std::cout << '\n';

    return 0;
}
```

- Remove any unneeded concerns
- Replace groups of related concerns with a single concern
- Isolate as many concerns to small portions of the program as possible using functions or even isolating to sections

@concerns

```
// convert to area
// @concerns height, width, area[out]
const auto area = height * width;
```

- Typically, the concerns are not stated in the code but understood by the developer
- Here, to demonstrate, the concerns for each section of code are shown using the `@concerns` in the comment
- `@concerns name[in]` The default for a concern, meaning it is defined elsewhere in the code and used in this section
- `@concerns name[out]` The concern is defined in that section

- The
- The

Requirements Concerns

- To code some parts of the program, we must understand the *requirements* of the program.
- In many cases, the *requirement* concerns are only described in natural language (e.g., English)
- *input format* What is the format of the program's input? This can include `std::cin` or the program options, `argc` and `argv`. This includes what the parts of the input data are and where they occur in the input.
- *output format* What is the format of the program's output? This can include `std::cout`, the data to output, and the format for each output part.

Other Types of Concerns

- *domain concerns*

- Literal strings and values

- Programming language statements and objects

- Error handling

Isolating a Concern

- We can take a concern that has (potentially) many parts and isolate it to a separate part of the program

- This can be done by creating a variable to pass data from one part (hunk) to another part (hunk)