Object-Oriented Programming

Grouping Functions

Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

Free Functions

A function that is not a member of a class

- Useful for algorithms performed on the parameters
- All IN data is from a value or const reference parameter
- All OUT data is from a reference parameter or the return value
- Does not save *state* between calls
- Typically group (or organized) by file, e.g., xml_parser.hpp

Problem: Conflicting Names

```cpp
// pdf_parser.hpp
std::string parse(std::string_view filename);

// yaml_parser.hpp
std::string parse(std::string_view filename);

// main.cpp
#include <iostream>
#include <string>
#include <filesystem>
#include <pdf_parser.hpp>
#include <yaml_parser.hpp>

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

    std::filesystem::path filePath(argv[1]);

    // parse the input depending on the file type
    if (filePath.extension() == '.pdf') {
        // parse pdf file
        std::cout << parse(argv[1]) << '\n';
    } else {
        // parse yaml file
        std::cout << parse(argv[1]) << '\n';
    }

    return 0;
}
```

- Multiple free functions with the same name and parameter list
- Causes a name conflict
- Program cannot be built Converter
- Means that pdf_parser.hpp and yaml_parser.hpp cannot be used in the same code
- We should not restrict the potential environment for our code to be used in

Problem: Conflicting Names, Solution?

```cpp
// pdf_parser.hpp
std::string pdf_parse(std::string_view filename);

// yaml_parser.hpp
std::string yaml_parse(std::string_view filename);

// main.cpp
#include <iostream>
#include <string>
#include <filesystem>
#include <pdf_parser.hpp>
#include <yaml_parser.hpp>

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

    std::filesystem::path filePath(argv[1]);

    // parse the input depending on the file type
    if (filePath.extension() == '.pdf') {
        // parse pdf file
        std::cout << pdf_parse(argv[1]) << '\n';
    } else {
        // parse yaml file
        std::cout << yaml_parse(argv[1]) << '\n';
    }

    return 0;
}
```

- Has a prefix, `pdf_` for the pdf parser, `yaml_` for the yaml parser
- Prefix would be used for all functions provided
- Solves the conflict
- More of a C-library form
- Could also extend to class names, variables, and other program elements that we name
- However, we have better opt ions available
- We should not restrict the potential environment for our code to be used in

Better Solution: Namespaces

```cpp
// pdf_parser.h
namespace PDFParser {
    std::string parse(std::string_view filename);
}

// yaml_parser.h
namespace YAMLParser {
    std::string parse(std::string_view filename);
}

// main.cpp
#include <iostream>
#include <string>
#include <filesystem>
#include <pdf_parser.hpp>
#include <yaml_parser.hpp>

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

    std::filesystem::path filePath(argv[1]);

    // parse the input depending on the file type
    if (filePath.extension() == '.pdf') {
        // parse pdf file
        std::cout << PDFParser::parse(argv[1]) << '\n';
    } else {
        // parse yaml file
        std::cout << YAMLParser::parse(argv[1]) << '\n';
    }

    return 0;
}
```

- *A language feature to group functions and provide an organizing name*

- Can contain:

  free functions

  variables

  classes

  `typedef`

Cumulative Namespaces

```
namespace Utility {
    // average
    int avg(int n1, int n2);
}

namespace Utility {
    // minimum
    int min(int n1, int n2);
}
```

- All of the contents of a particular namespace do not have to be declared at the same time
- They can even be declared in separate files

Namespace Aliases

```cpp
#include <boost/optional.hpp>
#include <optional>

namespace opt = boost;

struct Data {
    opt::optional<int> counts;
};
```

- The type `optional` exists in both `std::` and `boost::`
- The namespace `opt` can refer to either `boost` or `std`
- Does abstract the choice but requires tracing back when needed
- Useful in conversion or potential flexibility
- Not a long-term solution due to the abstraction cost

Another Option: Static Methods of a Class

```cpp
class PDFParser {
public:
    static std::string parse(std::string_view filename);
};

class YAMLParser {
public:
    static std::string parse(std::string_view filename);
};
```

- `static` class methods do not have access to data members/fields
- They do have access to `static` data members/fields
- We will consider that another time
- For now, considering methods that could be free functions but, due to naming issues, are not
- No advantage over namespaces, except that it may be joined with non-static member functions
- We can easily put classes in a namespace
- We c
class
caus
time
some