Lecture 8 Notes

Object-Oriented Programming

Object-Oriented programming is a style of programming that encourages the use of "objects". An object is a self-contained unit that houses variables and functions. The variables inside an object are attributes. The functions inside an object are methods. The data-type of an object is it's class. It is considered good practice to adhere to encapsulation in an object-oriented program.

Encapsulation

Encapsulation is when an object's attributes are not directly accessible from outside the object. The only code that can directly access and directly assign values to an object's attributes is the methods inside that object. Here is an example of a C++ class that uses encapsulation:

```
class Vector2 {
private:
    double x, y;
public:
    double get_magnitude() const;
    double get_direction() const;
    double get_x_component() const;
    double get_y_component() const;
    void set_magnitude(double m);
    void set_direction(double d);
};
```

Inside the class we can have "private" or "public" labels preceeding any attribute or method declarations. Class members that are declared under private scope cannot be accessed by any code that is outside that class, only methods inside the class can use them. Class members that are declared under public scope can be accessed from anywhere in the program, especially outside the class.

In our example code, the methods that are named with a "get_" prefix all have something in common, they all return information about the state of the object without modifying it. The state of an object is determined by the values of it's attributes. In C++, if we suffix the keyword "const" to a method, this means that the method does not assign any values to it's object's attributes, doing so results in a compiler error. These type of methods are known as accessors.

The methods that are named with a "set_" sprefix also have something in common, they modify the state of the object by modifying one or more of it's attributes. A method that changes the object's state is known as a mutator, they usually have no return value.

The idea behind encapsulation is that an object's methods are the sole means to reading or modifying an object's state. If a programmer created a class that other programmers use, the author programmer may modify some code inside the class without breaking compatibility with other programmers' code, just so long as class's public methods work the same way as they used to.

Encapsulation is also thought to make debugging easier, because a malfunctioning object can be traced to code inside it's class.

Programming Paradigms

A programming paradigm is a particular way to think about and create computer programs. Before you started learning about classes, you primarily programmed by creating and calling functions; programming in this way is known as procedural programming.

Procedural Programming

Procedural programming is usually the first programming paradigm that people use when they first start learning about computer programming. This is also perhaps the most popular programming paradigm because it's relatively simple to learn (compared to the other paradigms) and simple to use.

A procedural program works by breaking the problem down into seperate functions. Procedural programs may also pass around references to arrays or structures for these functions to operate on. Functions in a procedural programs are not restricted to reading or writing data from it's own parameters. Functions may operate on global variables or data from the computer's file system.

For example, if I wanted to create a program that generated a CSV file from input data, a procedural program might do it like this:

```
int main()
{
    char *raw_data;
    struct data_sample sample;

    while ((raw_data = read_raw_data()) != 0) {
        process_raw_data(raw_data, &data_sample);
        print_data_sample(&data_sample);
        free(raw_data);
    }
    return 0;
}
```

Object-Oriented Programming

Another programming paradigm that you have just been introduced to is object-oriented programming. This paradigm makes heavy use of "objects", which are variables that contain "methods" (member functions) and "attributes" (member variables). A data type which describes an object is known as a class.

So while procedural programs seperate the data from the functions. Object-oriented programs combine data and functions into classes and objects. An object-oriented program might look like this:

```
int main()
{
    DataSampleReader reader;
```

```
    DataSample sample;

    while (not reader.empty()) {
        sample = reader.read();
        sample.print();
    }
    return 0;
}
```

In Object-Oriented programming, each object is like a mini-program. Ideally, the attributes in an object should only be directly accessible by that object's methods. Thus, modifying an object's state should only be done through the interface of the methods that it provides. In a pure object-oriented language, every data type is an object and every object adheres to strict encapsulation. C++ is not a pure object-oriented language because it supports non-object data types such as int and double.

Functional Programming

A functional program operates like the evaluation of a mathematical function. That function may branch out and call other functions. In a pure functional program, each function does not read any data or variable that was not passed as an argument into that function's parameters, also assigning a new value to a variable is highly discouraged, thus the order of the chain of command lies exclusively in the entry point of the program.

Functional programming is usually performed in a programming language other than C++. Programming languages like Haskell and Scheme are specifically designed to support functional programming. Nevertheless, functional programming can more or less be simulated in C++ like this:

```
int main()
{
    WriteOutput(ProcessDataSamples(ReadInput()));
    return 0;
}
```

Function composition is common in functional programming, notice how the flow of data linearly moves from one function to the next. You can also perform a similar form of functional programming in a bash shell, for example this is a command to format a source file and send it to the printer:

```
cat program.cpp | nl | pr | expand --tabs=3 | lpr
```

Here each command is seperated by the pipe symbol. The pipe symbol is an operator that executes it's left-hand command and sends it's output as input to the execution of the right-hand command. So the output of one program is the input to the next program. In command-line terminology, this is known as pipelining.

Object-Oriented programming and Functional programming can be seen as variations of procedural programming with stricter rules. Object-Oriented programming encourages "encapsulation", which means only a class's methods should have access to that class's attributes. Functional programming discourages "side effects" which is the modification of any data from within a function.

These restrictions are believed to result in better organized programs which may be easier to maintain, easier to debug, and easier to extend. However, nobody really agrees on which programming paradigm is best. In practice, a purely object-oriented or purely functional program is very rare because procedural programming elements are often needed to get any real work done.

Nevertheless, there is some wisdom that can be found in these programming paradigms. Good software design is the goal, and the paradigms attempt to act as a means to get there. One must know to what degree a certain programming paradigm should be incorporated into a project. It varies with the particular problem. A good programmer is a pragmatic programmer.