

Object-Oriented Programming

Key Concepts

Procedural Programming

- Procedural programming is a form of structured programming:
 - Generally oriented around *procedures* (also known as *functions*, *routines*, or *subroutines*) which accept data, usually as parameters
 - These process this data according to their internal programming logic, using basic statements or by calling other procedures
 - In the end, they produce a result and return it to where they were called from
 - Typically, a main procedure drives the program by executing statements and procedure calls in sequence until completion

Procedural Programming

- Procedural programming is also sometimes referred to as imperative programming
 - That said, procedural programming uses blocks and scoping rules that non-structured imperative languages do not
- Examples of procedural languages
 - Basic, Fortran, Pascal, C
 - Go is a more modern example
 - Python and Rust can be, but they can also be other things, including object-oriented

Procedural Programming

```
void main() {  
    int i = 0;  
    int j = 0;
```

```
    i = sum(j, 10);
```

```
    printf("i is %d\n", i);
```

```
    j = sum(i, 20);
```

```
    printf("j is %d\n", j);
```

```
}
```

```
int sum(int a1, int a2) {  
    return (a1+a2);  
}
```

- Procedures are called for performing an operation (in this case summing two numbers).
- After a procedure completes its operation, the control of the program returns to a point right after the calling point.

Object-Oriented Programming

- Object-Oriented programming
 - Structured around *objects*
 - Objects are instances of *classes*
 - Objects accept *messages* from other objects; these messages essentially *call methods* in the receiving object in order to perform an action (that is the programming logic of the invoked method)
 - You may think of methods as procedures that are encapsulated inside a class
- Examples of object-oriented languages
 - Java, Objective-C, Swift, C#, and of course C++

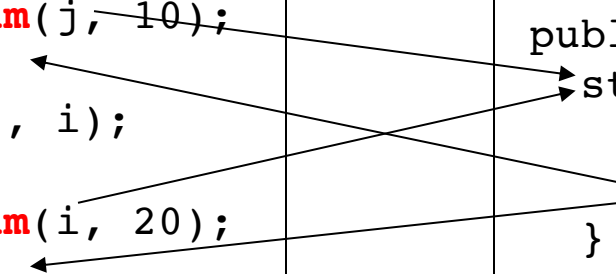
Object-Oriented Programming

- Objects
 - They combine (i.e. encapsulate) in one unit both *data* and *functionality*
 - The data encapsulated in an object are called *data members* of the object
 - The functionality is provided by functions called *methods*; the methods encapsulated in an object are also referred as *member functions* of the object
 - Usually methods of an object provide the only way for an external piece of code (i.e. another object) to access its data members
 - Objects are instances of corresponding types that are called *classes*
 - There is a direct link between the concept of a class and that of a traditional Abstract Data Type

Object-Oriented Programming

```
int main() {  
    int i = 0;  
    int j = 0;  
  
    i = Calculator::sum(j, 10);  
    printf("i is %d\n", i);  
    j = Calculator::sum(i, 20);  
    printf("j is %d\n", j);  
}
```

```
class Calculator {  
public:  
    static int sum(int a1,  
                  int a2) {  
        return (a1+a2);  
    }  
};
```



The diagram consists of four arrows. Two arrows originate from the `sum` method calls in the `main` function. One arrow points from `Calculator::sum(j, 10)` to the `static int sum` definition in the `Calculator` class. The other arrow points from `Calculator::sum(i, 20)` to the same `static int sum` definition. Additionally, there are two arrows pointing from the `sum` definition back to the `sum` calls in the `main` function, indicating the return flow.

- Methods are called in a very similar fashion as procedures were called earlier.
- Notice that methods can also be attached to a class and not just an instance of the class.
- Please note there are several things wrong with what we are doing. As we go through the course, you will see why!

Procedural vs. Object-Oriented Programming

- There is a very similar control flow between procedure and object-oriented programming
 - Very similar call-and-return mechanisms, in one case using procedures and in the other using methods on classes and objects
- The key distinction is that procedural languages organize by procedures with data and functionality separate (though sometimes loosely collected into *modules*), whereas object-oriented languages organize by classes and objects, with data and functionality effectively bundled together

Benefits of Object-Oriented Programming

- Better modeling of elements of the real world
- Better application and use of Software Engineering principles
 - Information/data hiding
 - Encapsulation
 - Abstraction
 - Modularization
- Better integration with other programs and libraries through agreed upon and standardized interfaces

Benefits of Object-Oriented Programming

- Information hiding:
 - Refers to shielding and “hiding” the underlying design and implementation details of a subsystem (or a class) from the rest of the program
 - Instead of needing to know the details, the rest of the program simply interacts with the subsystem through a published interface
 - For example, if the subsystem is a class, then information hiding is shielding and hiding from the rest of the program (i.e. other classes) how data members and member functions (i.e. methods) are implemented; instead they only need to know how to invoke the relevant methods

Benefits of Object-Oriented Programming

- Encapsulation:
 - Refers to the bundling of data and functionality together into a single package that is accessed through a well defined interface, as opposed to having things spread around a program or otherwise only loosely associated
 - Not only does this promote information hiding, but this also effectively restricts direct access to at least some of the package's data and functions

Benefits of Object-Oriented Programming

- Abstraction:

- Refers to modeling an entity (e.g. program, method, algorithm, data) in a way so that only the important characteristics are presented, while the non important ones are omitted
- “The essence of abstractions is preserving information that is relevant in a given context, and forgetting information that is irrelevant in that context.”

– John V. Guttag

Benefits of Object-Oriented Programming

- Modularization:
 - Refers to the design approach where a software application is implemented as a collection of independent units (subsystems) that communicate by exchanging only the absolutely necessary information (data) that is required to perform an operation
 - In other words, they have low coupling
 - Furthermore, each unit (subsystem) performs a specific operation or a collection of very closely related operations
 - In other words, they have high cohesion
 - Usually, a collection of related classes constitutes a subsystem

Some Caveats

- Object orientation is not the solution to every programming need
 - Data base applications → SQL, 4GLs
 - Embedded systems → Assembly,
- While object-oriented programming has numerous benefits in theory, achieving those benefits in practice can be difficult and developers often fall short of achieving these goals
 - Remember the old adage:

“In theory, theory and practice are the same. In practice, they’re not.”

Some Caveats

- Most importantly, this has been a fairly superficial coverage of object-oriented programming, and many details have been glossed over or left out for now
- As we progress through the course and explore C++ as a language, we will come back and delve deeper into things accordingly ...