**Procedure Oriented Programming**

In the POP approach, the problem is viewed as a sequence of things to be done such as reading, calculating and printing. A number of functions are written to accomplish these tasks. The primary focus is on functions.

In multi-function program, many important data items are placed as global so that they may be accessed by all the functions. Each function may have its own local data.

Global data are more vulnerable to an inadvertent change by a function. In a large program it is very difficult to identify what data is used by which function.

Another serious drawback with the procedural approach is that it does not model real world problems very well. This is because functions are action-oriented and do not really corresponding to the elements of the problem.

**Characteristics of POP**

* Emphasis is on doing things.(algorithms)
* Large programs are divided into smaller program known as functions.
* Most of the functions share global data.
* Data move openly around the system from function to function.
* Functions transform data from one form to another.
* Employ top down approach in program design

**Object Oriented Programming**

The major motivating factor of OO approach is to remove some of the flaws encountered in procedural approach. OOP treat data as critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it, and protects it from accidental modification from outside the functions. OOPS allow decomposition of problem into number of entities called objects and builds data and objects around these objects. The data of an object can be accessed only by the functions associated with that object. However the functions of an object can access functions of another object.

**Features of OOP are:**

* Emphasis on data rather than procedure
* Programs are divided into what are known as objects
* Data structures are designed such that they characterize the objects.
* Functions that operate on the data of an object are tied together in the data structure
* Data is hidden and cannot be accessed by external functions.
* Objects may communicate with each other through functions.
* New data and functions can be easily added whenever necessary.
* Follows bottom up approach in system design.

OOP is an approach that provides a way of modularizing program by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such module on demand

**Problems with procedural languages**

There are two related problems. First, functions have **unrestricted access** to global data. Second, **unrelated functions and data**, the basis of the procedural paradigm, provide a poor model of the real world.

1. **Unrestricted Access**

In a procedural program, one written in C for example, there are two kinds of data. Local data is hidden inside a function, and is used exclusively by the function.

Important data in a program—then the data must be made global, Global data can be accessed by any function in the program. (We ignore the issue of grouping functions into modules, which doesn’t materially affect our argument.) The arrangement of local and global variables in a procedural program is shown in Figure 1.1.

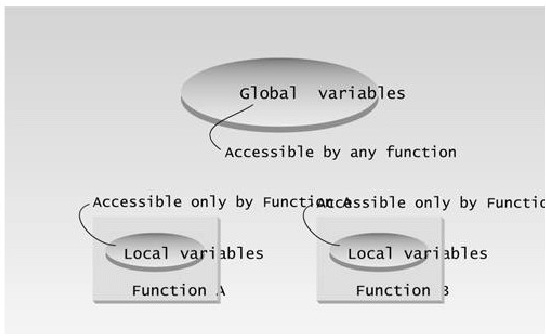


Figure 1.1

In a large program, there are many functions and many global data items. The problem with the procedural paradigm is that this leads to an even larger number of potential connections between functions and data, as shown in Figure 1.2.

This large number of connections causes problems in several ways. First, it makes a program’s structure difficult to conceptualize. Second, it makes the program difficult to modify. A change made in a global data item may result in rewriting all the functions that access that item

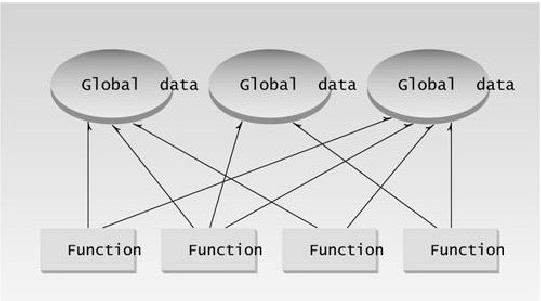


Figure 1.2 Procedural Programming

When data items are modified in a large program it may not be easy to tell which functions access the data, and even when you figure this out, modifications to the functions may cause them to work incorrectly with other global data items. Everything is related to everything else, so a modification anywhere has far-reaching, and often unintended, consequences.

1. Real-World Modeling

The second—and more important—problem with the procedural paradigm is that its arrangement of separate data and functions does a poor job of modeling things in the real world. In the physical world we deal with objects such as people and cars. Such objects aren’t like data and they are not like functions. Complex real-world objects have both attributes and behavior.

* Attributes

Examples of attributes (sometimes called characteristics) are, for people, eye color and job titles; and, for cars, horsepower and number of doors. As it turns out, attributes in the real world are equivalent to data in a program: they have a certain specific values, such as blue (for eye color) or four (for the number of doors).

* Behavior

Behavior is something a real-world object does in response to some stimulus. If you ask your boss for a raise, she will generally say yes or no. If you apply the brakes in a car, it will generally stop. Saying something and stopping are examples of behavior. Behavior is like a function: you call a function to do something, like display the inventory, and it does it. So neither data nor functions, by themselves, model real world objects effectively.

1. New Data Types

There are other problems with procedural languages. One is the difficulty of creating new data types. Computer languages typically have several built-in data types: integers, floating-point numbers, characters, and so on. What if you want to invent your own data type? Perhaps you want to work with complex numbers, or two-dimensional coordinates, or dates—quantities the built-in data types don’t handle easily. Being able to create your own types is called extensibility; you can extend the capabilities of the language. Traditional languages are not usually extensible. Without unnatural convolutions, you can’t bundle both x and y coordinates together into a single variable called Point, and then add and subtract values of this type. The result is that traditional programs are more complex to write and maintain.

**The Object-Oriented Approach**

The fundamental idea behind object-oriented languages is to combine into a single unit both data and the functions that operate on that data. Such a unit is called an object. An object’s functions, called member functions in C++, typically provide the only way to access its data. If you want to read a data item in an object, you call a member function in the object. It will access the data and return the value to you. You can’t access the data directly. The data is hidden, so it is safe from accidental alteration. Data and its functions are said to be encapsulated into a single entity. Data encapsulation and data hiding are key terms in the description of object-oriented languages.

If you want to modify the data in an object, you know exactly what functions interact with it: the member functions in the object. No other functions can access the data. This simplifies writing, debugging, and maintaining the program.

A C++ program typically consists of a number of objects, which communicate with each other by calling one another’s member functions. The organization of a C++ program is shown in Figure 1.3. Also, data items are referred to as attributes or instance variables. Calling an object’s member function is referred to as sending a message to the object. These terms are not official C++ terminology, but they are used with increasing frequency, especially in object-oriented design.

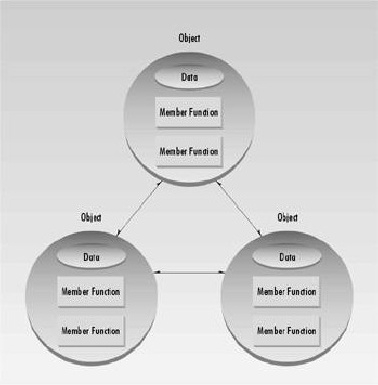


Figure 1.3 The object-oriented paradigm