Chapter 11

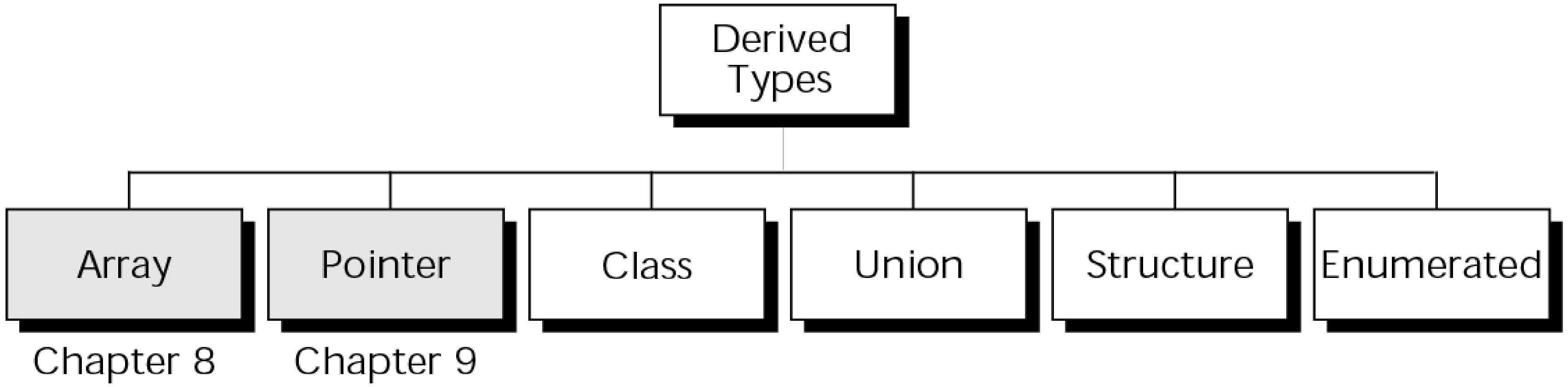
More Class Features and Other Types

OBJECTIVES

After studying this chapter you will be able to:

- ❑ **Write explicit and implicit inline functions.**
- ❑ **Use an initialization list to initialize the members of a class.**
- ❑ **Write overloaded functions.**
- ❑ **Overload operators.**
- ❑ **Use static members in a class.**
- ❑ **Use pointers to classes.**
- ❑ **Declare a function as a friend of a class.**
- ❑ **Create an enumerated type.**
- ❑ **Define and use a structure.**
- ❑ **Define and use a union.**
- ❑ **Understand how functions are coupled.**

Figure 11-1 Derived types



11.1

INLINE FUNCTIONS

Note:

Inline functions are used to improve the efficiency of a program.

INITIALIZATION LIST

Figure 11-2 Using initialization lists

```
Fraction::Fraction (int num,
                    int denom)
{
   numerator   = num;
   denominator = denom;
   …
}  // Fraction Constructor
```

(a) Using the assignment operator

```
Fraction::Fraction (int num,
                    int denom)
   : numerator   (num),
     denominator (denom)
{
   …
}  // Fraction Constructor
```

(b) Using the initialization list

OVERLOADING

Note:

C++ provides a bitwise overloaded assignment operator if we don't overload it ourselves.

Note:

The assignment operator is different from the copy constructor.

Note:

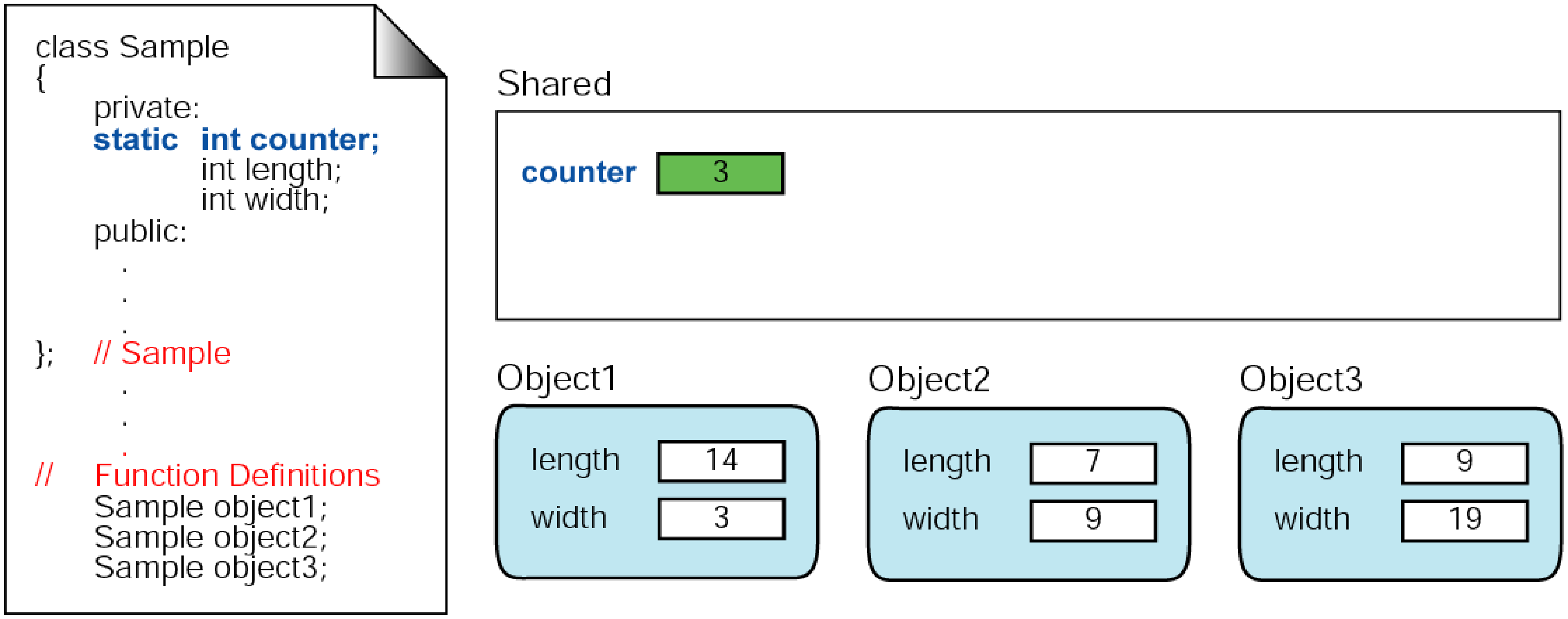
When we need to write a copy constructor, we also need to write an assignment operator and a destructor.

Static Members

Note:

By default, a class member is an instance member.

Figure 11-3 Class members



```
class Sample
{
      private:
      static  int counter;
              int length;
              int width;
      public:
         .
         .
         .
};    // Sample
         .
         .
         .
//    Function Definitions
      Sample object1;
      Sample object2;
      Sample object3;
```

Note:

Static data members are instance independent.

Note:

There are three static data members:

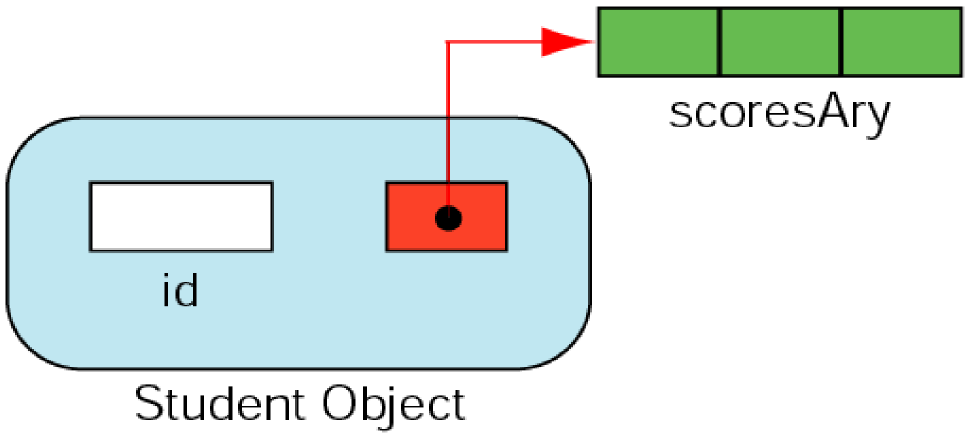
- ***explicit static data members***
- ***enumerated members***
- ***type defined members***

FRIEND CLASSES

11.6

Classes and Pointers

Figure 11-4 Student class object



Array of Objects

Note:

We are only allowed to call the default constructor when we create an array of objects.

11.8

STRUCTURE

Note:

The class and structure constructs are identical with one exception:

Members in a structure are public by default whereas they are private by default in a class

Figure 11-5 Class and structure declarations

```
class Sample
{
          private:
            ...
          private:
            ...
};  // Sample
```

```
struct Sample
{
          private:
            ...
          private:
            ...
}; // Sample
```

Figure 11-6 Private versus public defaults

```
class Sample
{
            int x;
            int y;
            ...
            void print();
}; // Sample
```

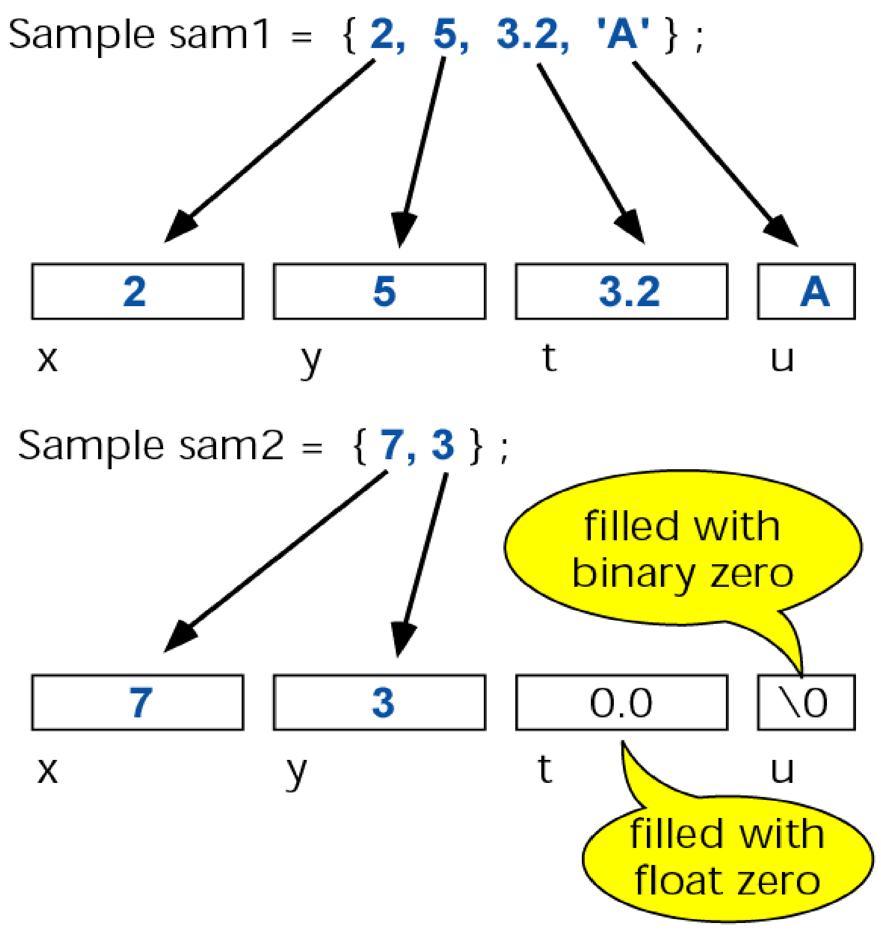
```
struct Sample
{
            int x;
            int y;
            ...
            void print();
}; // Sample
```

Note:

Use structures only for simple constructs that do not require data protection or specialized functions.

Figure 11-7 Initializing structures

```
struct Sample
  {
     int    x ;
     int    y ;
     float t ;
     char u ;
  } ;  //Sample
```



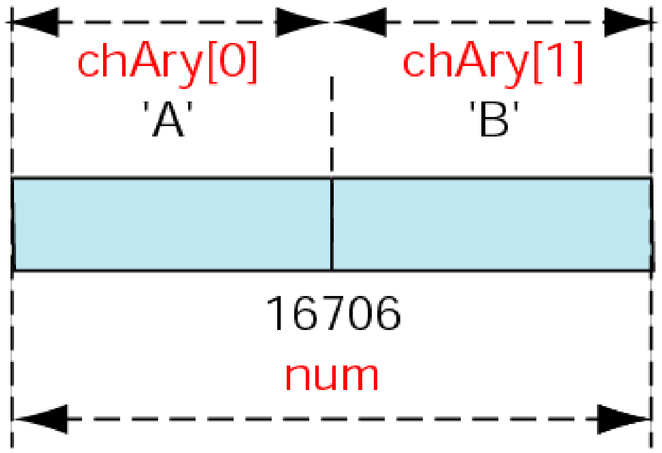
11.9

UNIONS

Brooks/Cole
Thomson Learning™

Figure 11-8 Unions

```
union shareData
    {
     char    chAry[2];
     short   num;
    };
```



Both num and chAry start at the same memory address. chAry[0] occupies the same memory as the most significant byte of num.

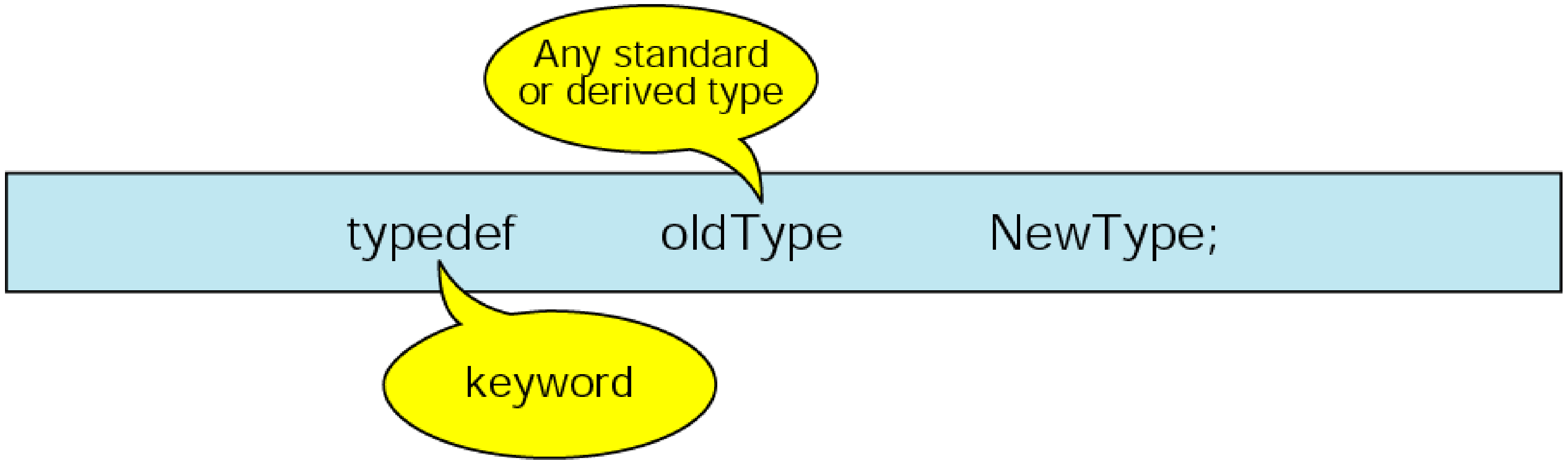
11.10

Enumerated Types

11.11

THE TYPE DEFINITION (*TYPEDEF*)

Figure 11-9 Typed definition format



Note:

The typedef command does not create a new type. It just creates an alias, that is, a new name, for an existing type.

PROGRAMMING APPLICATIONS

Figure 11-10 Elevator structure

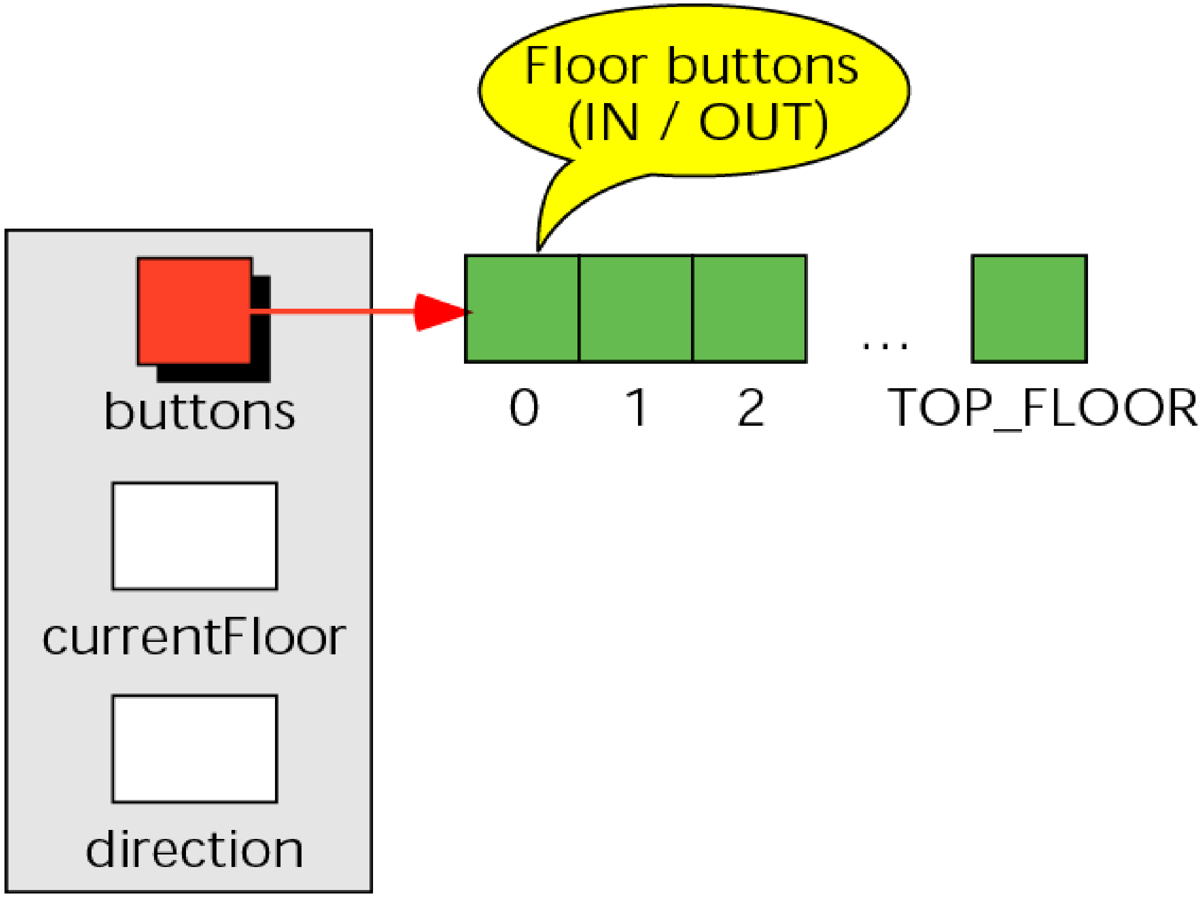


Figure 11-11 Elevator structure chart

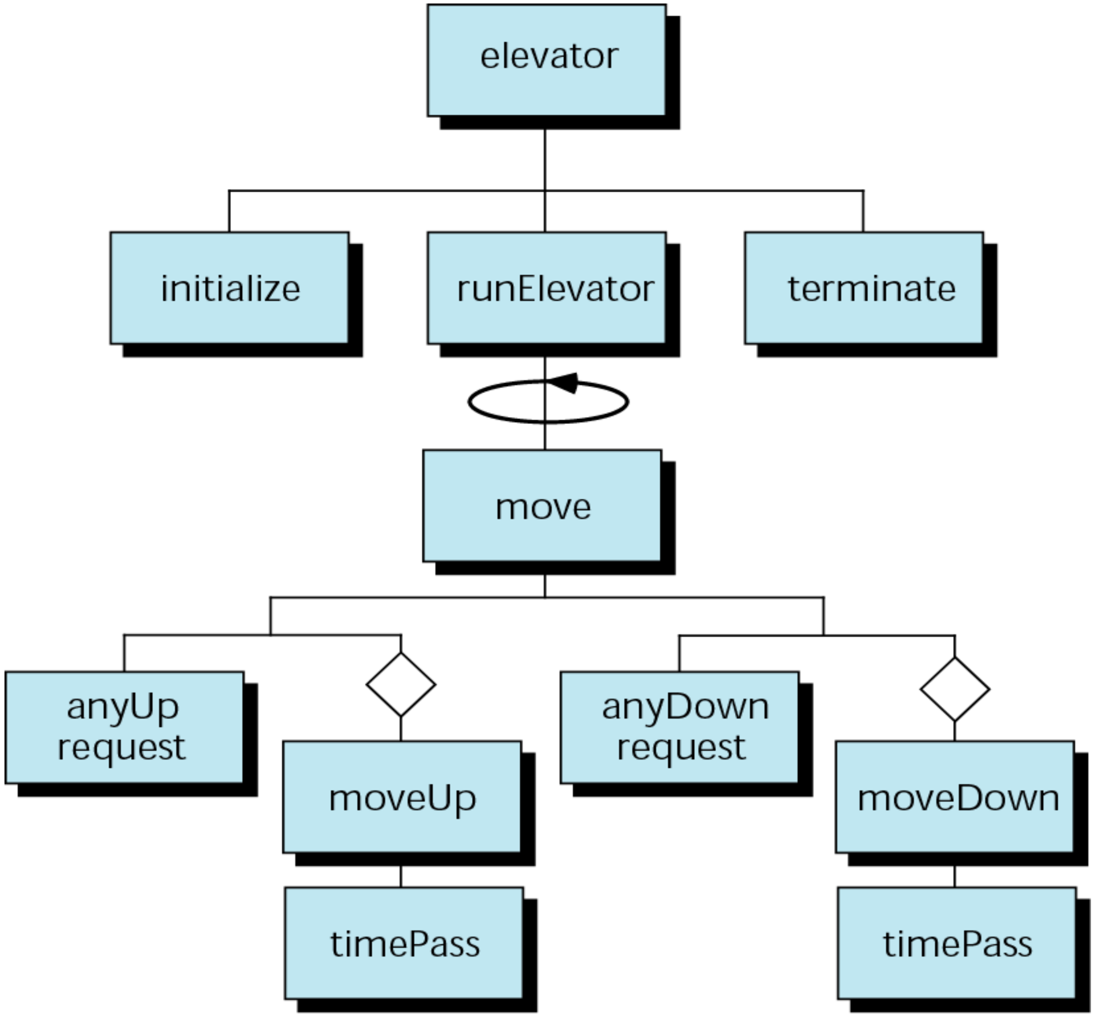
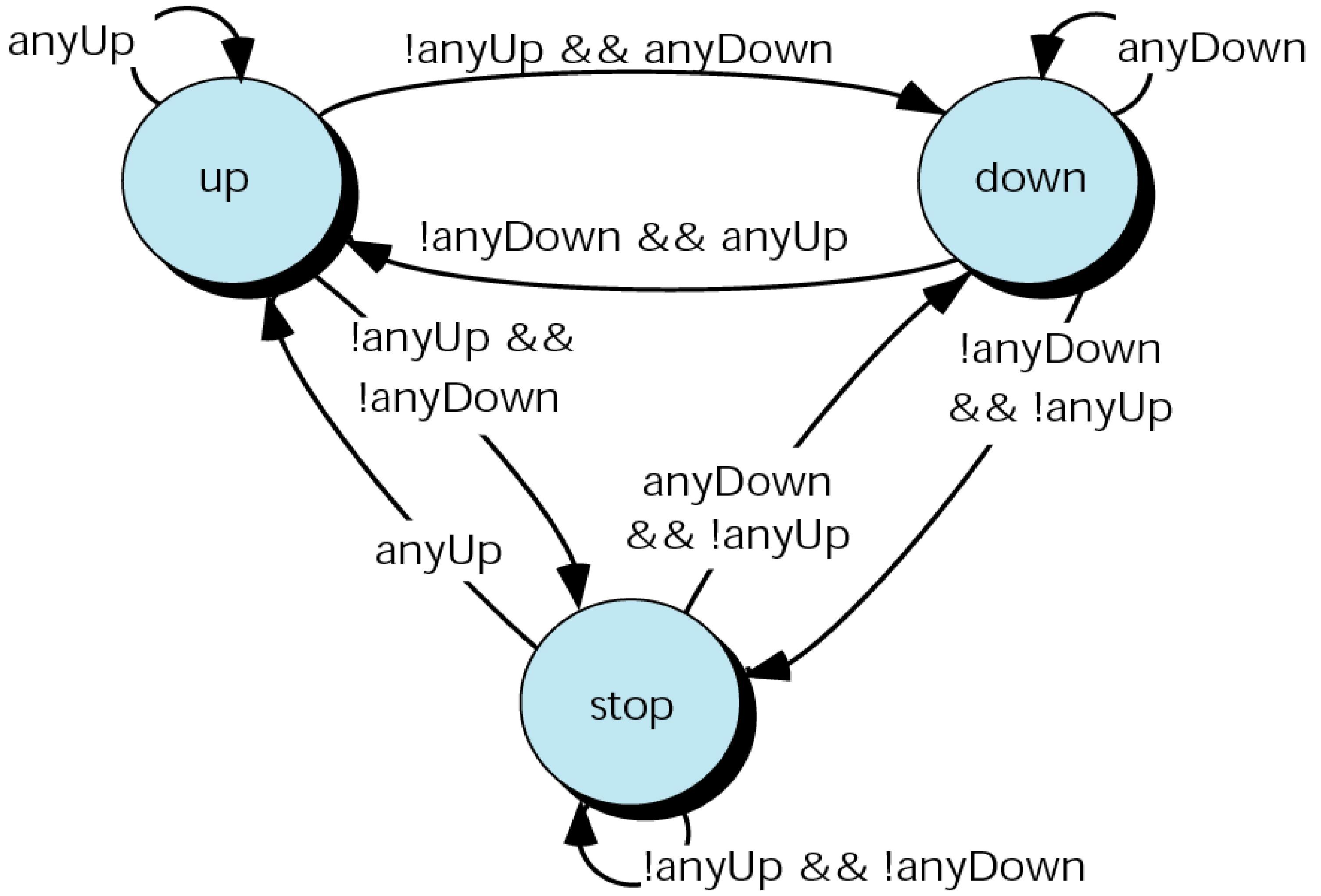


Figure 11-12 Elevator states



Brooks/Cole Thomson Learning

Software Engineering and Programming Style

Note:

Functions in well-structured programs are highly cohesive and loosely coupled.

Note:

Stamp coupling should pass only the data needed.

Note:

Avoid bundling unrelated data just to reduce the number of parameters being passed between functions.

Note:

Control coupling should be used only to pass status.

Note:

Avoid global coupling within a program.

Note:

Never use content coupling.

Note:

Programming Standard:

Do not place any variables in the global area of a program.