2.3.1 Structures

The first step in building a new type is often to organize the elements it needs into a data structure, a struct:

```
struct Vector {
      int sz;          // number of elements
      double* elem;    // pointer to elements
};
```

This first version of Vector consists of an int and a double*.

A variable of type Vector can be defined like this:

```
Vector v;
```

need to be defined

However, by itself that is not of much use because v's elem pointer doesn't point to anything. To be useful, we must give v some elements to point to. For example, we can construct a Vector like this:

```
void vector_init(Vector& v, int s)
{
      v.elem = new double[s];   // allocate an array of s doubles
      v.sz = s;
}
```

That is, v's elem member gets a pointer produced by the new operator and v's size member gets the number of elements. The & in Vector& indicates that we ~~pass v by non-const reference (§2.2.5,~~ §7.7); that way, vector_init() can modify the vector passed to it.

The ~~new operator allocates~~ memory from an area called *the free store* (also known as *dynamic memory* and *heap*; §11.2).

A simple use of Vector looks like this:

```
double read_and_sum(int s)
      // read s integers from cin and return their sum; s is assumed to be positive
{
      Vector v;
      vector_init(v,s);              // allocate s elements for v
      for (int i=0; i!=s; ++i)
            cin>>v.elem[i];          // read into elements

      double sum = 0;
      for (int i=0; i!=s; ++i)
            sum+=v.elem[i];          // take the sum of the elements
      return sum;
}
```

There is a long way to go before our Vector is as elegant and flexible as the standard-library vector. In particular, a user of Vector has to know every detail of Vector's representation. The rest of this chapter and the next gradually improve Vector as an example of language features and techniques. Chapter 4 presents the standard-library vector, which contains many nice improvements, and Chapter 31 presents the complete vector in the context of other standard-library facilities.

I use vector and other standard-library components as examples

- to illustrate language features and design techniques, and
- to help you learn and use the standard-library components.

Don't reinvent standard-library components, such as vector and string; use them.

We use . (dot) to access struct members through a name (and through a reference) and -> to access struct members through a pointer. ~~For example:~~

```
void f(Vector v, Vector& rv, Vector* pv)
{
      int i1 = v.sz;       // access through name
      int i2 = rv.sz;      // access through reference
      int i4 = pv->sz;     // access through pointer
}
```